
F5 iRules Data Plane Programmability Documentation

F5 Networks, Inc.

Feb 11, 2020

Agility 2020 Hands-on Lab Guide

Contents:

1	Getting Started	5
1.1	Lab Topology	5
2	Cooking with iRules - HTTP	7
2.1	Lab Components	7
3	Cooking with iRules - Security/SSL	43
3.1	Security/SSL iRules Labs	43
3.2	Additional Labs	56
4	Introduction to iRules LX	77
4.1	Creating and Implementing an LX iRule	78
4.2	NPM and Exception Handling	82
4.3	Asynchronous Programming	92
4.4	iRules LX Streaming	95

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.

1.1 Lab Topology

The following components have been included in your lab environment:

- 1 x F5 BIG-IP VE
- 1 x Linux LAMP Webserver
- 1 x Windows Jumphost

1.1.1 Lab Components

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

Component	VLAN/IP Address(es)	Credentials
BigIP	Management: bigip1	admin/admin
Jumphost	Jumphost: TBD	student/student

Cooking with iRules - HTTP

This class covers the following topics:

- HTTP Protocol Review
- HTTP Request Side Overview
- HTTP Response Side Overview
- HTTP Related Events
- HTTP Headers
- STREAM Command
- HTTP Payload Capture and Manipulation (If time permits)
- SSL::profile (If time permits)

Expected time to complete: **1.25 hours**

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

2.1 Lab Components

The following table lists the Credentials for all components:

Component	VLAN/IP Address(es)	Credentials
BigIP	Management: bigip1	admin/admin.F5demo.com
Jumphost	Jumphost: TBD	external_user/P@ssw0rd!

2.1.1 Cooking with iRules Labs

This is the collection of HTTP Labs. Here is where you get to prove you listened in class :)

Lab 1 - Create an iRule that Parses the URI to Route Traffic

Creating your first HTTP iRule that routes traffic based upon the value of the Host name.

The goal of this lab is to route incoming HTTP requests to a specific pool based on the incoming http host name.

Please create an iRule that will route traffic based on the following table:

Host Name	Pool Name
dvwa.f5lab.com	dvwa_pool_http
peruggia.f5lab.com	peruggia_http_pool
wackopicko.f5lab.com	wackopicko_http_pool

Important:

- Estimated completion time: 10 minutes
-

1. Open Chrome Browser
2. Enter <https://bigip1> into the address bar and hit Enter

f5 BIG-IP Configuration Utility
F5 Networks, Inc.

Hostname
bigip01.f5demo.com

IP Address
10.1.1.4

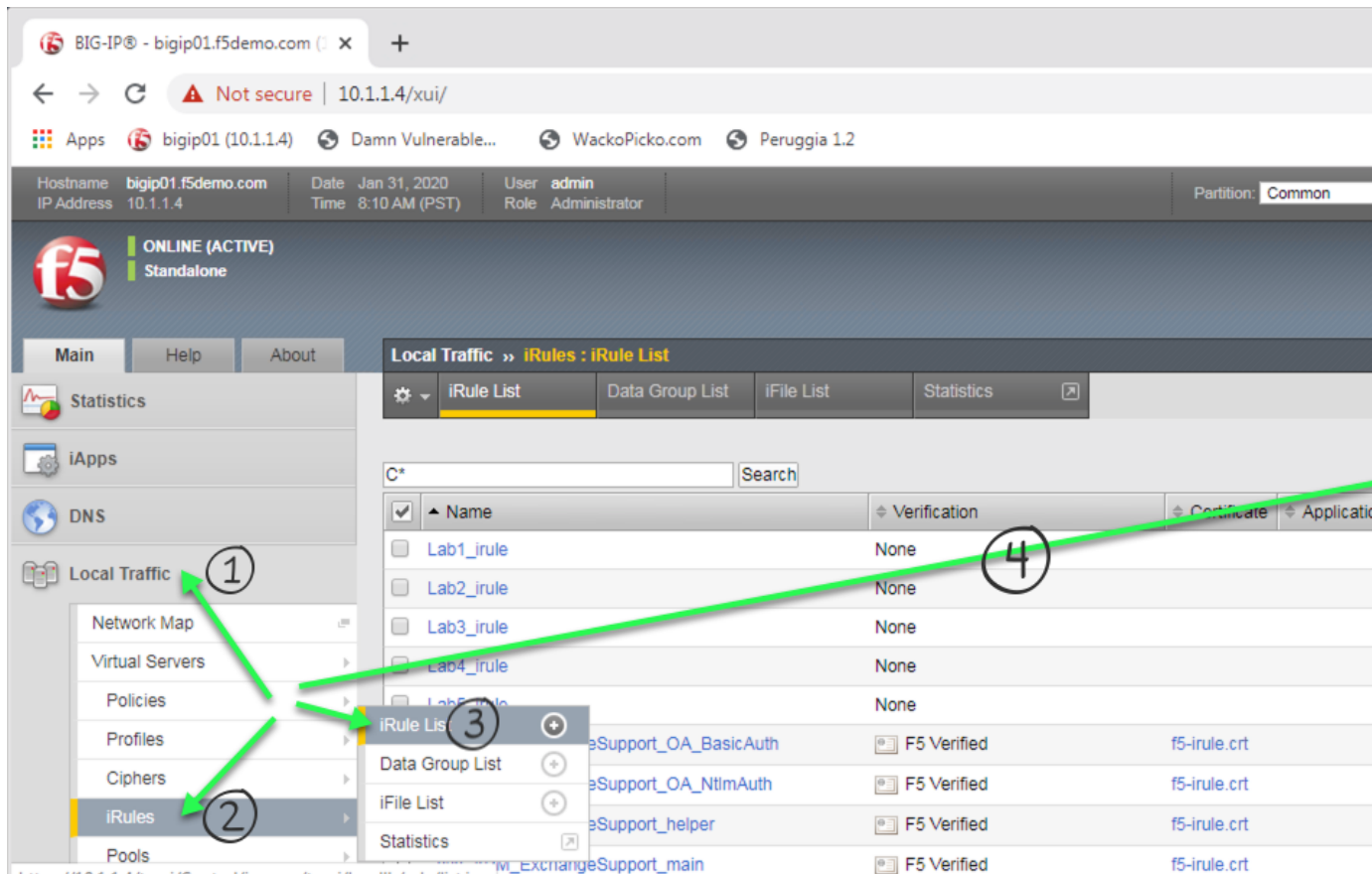
Username
admin

Password
.....

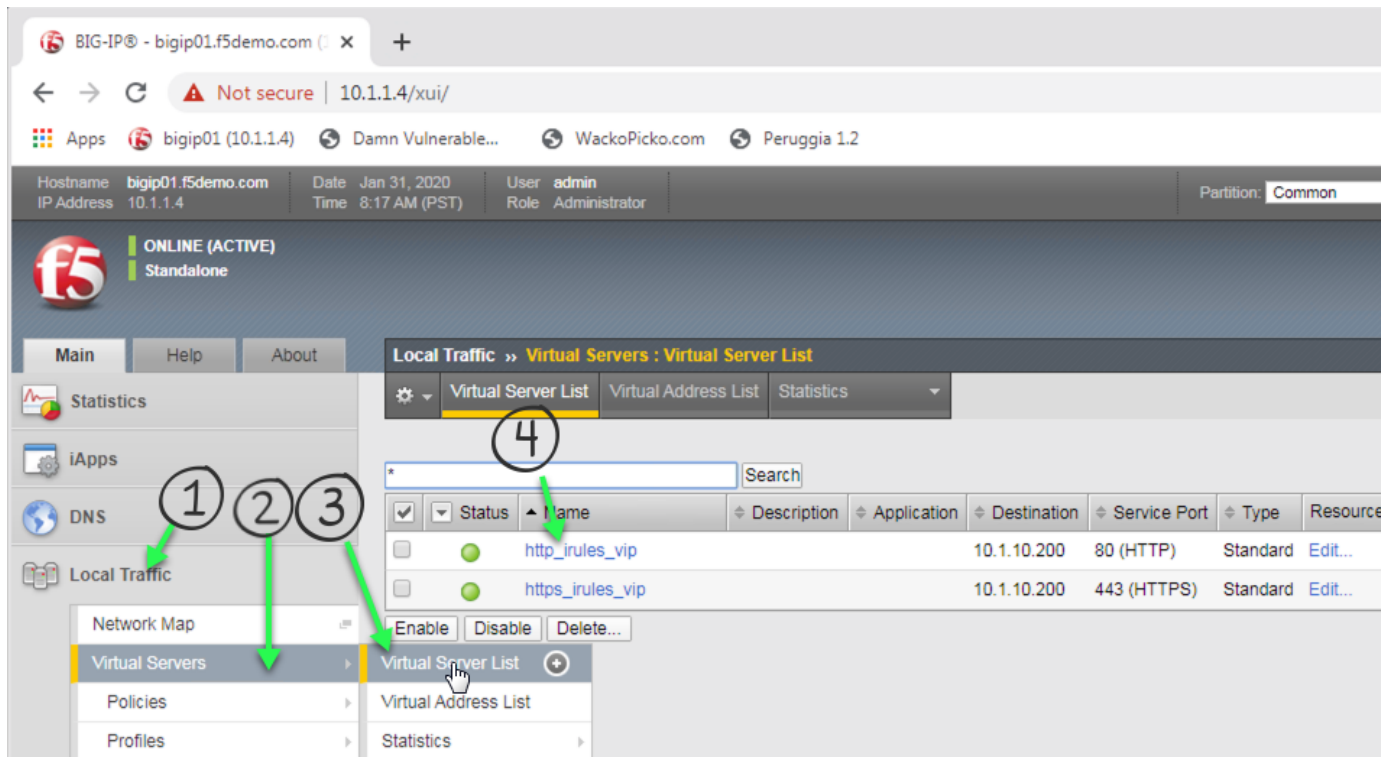
Log in

Welcome to the BIG-IP Configuration Utility.
Log in with your username and password using the fields on the

3. **Login with username: admin password: admin.F5demo.com**
4. Click Local Traffic -> iRules -> iRules List
5. Click **Create** button

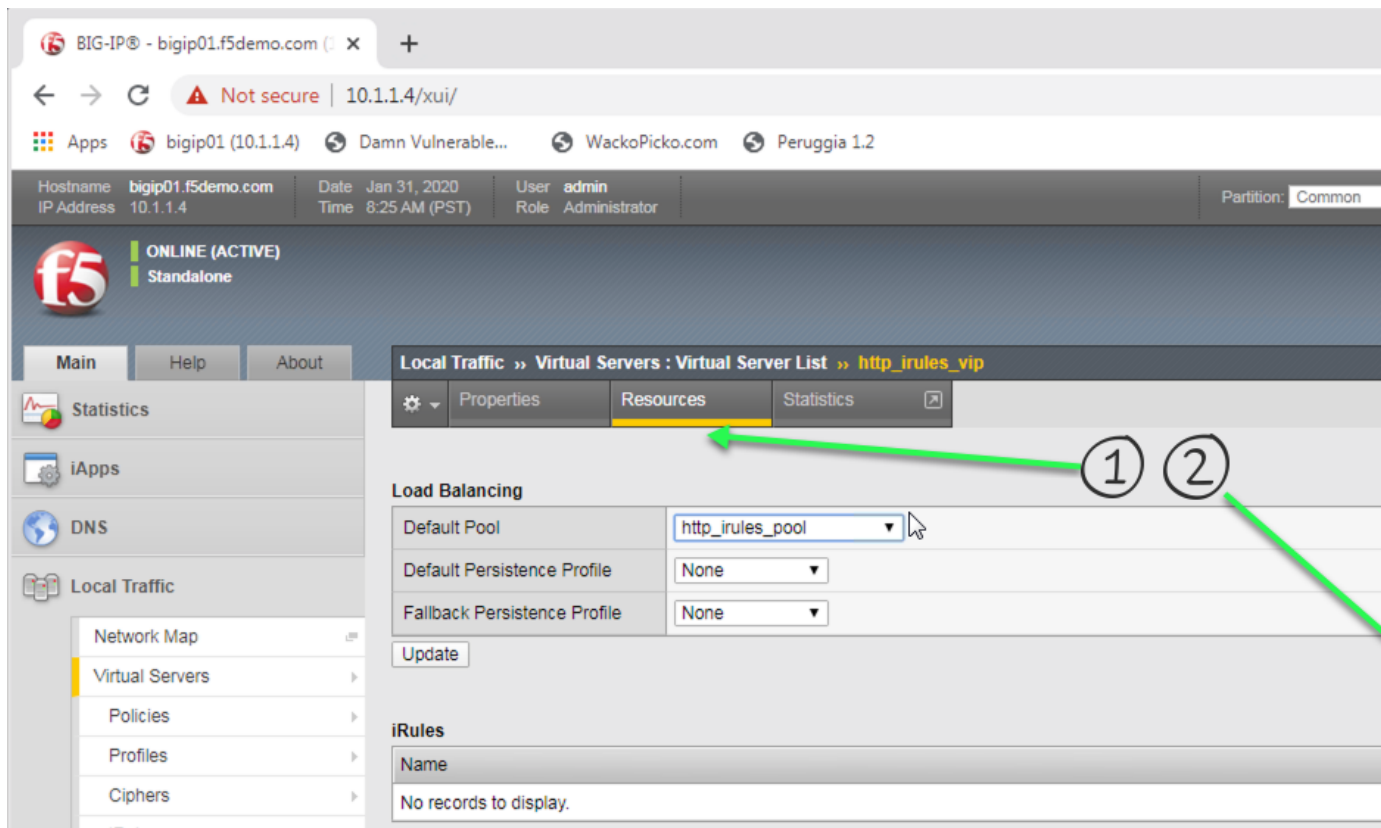


6. Enter Name of **URI_Routing_iRule**
7. Enter your code
8. Click **Finished**
9. Click Local Traffic -> Virtual Servers -> Virtual Server List
10. Click on **http_irules_vip**

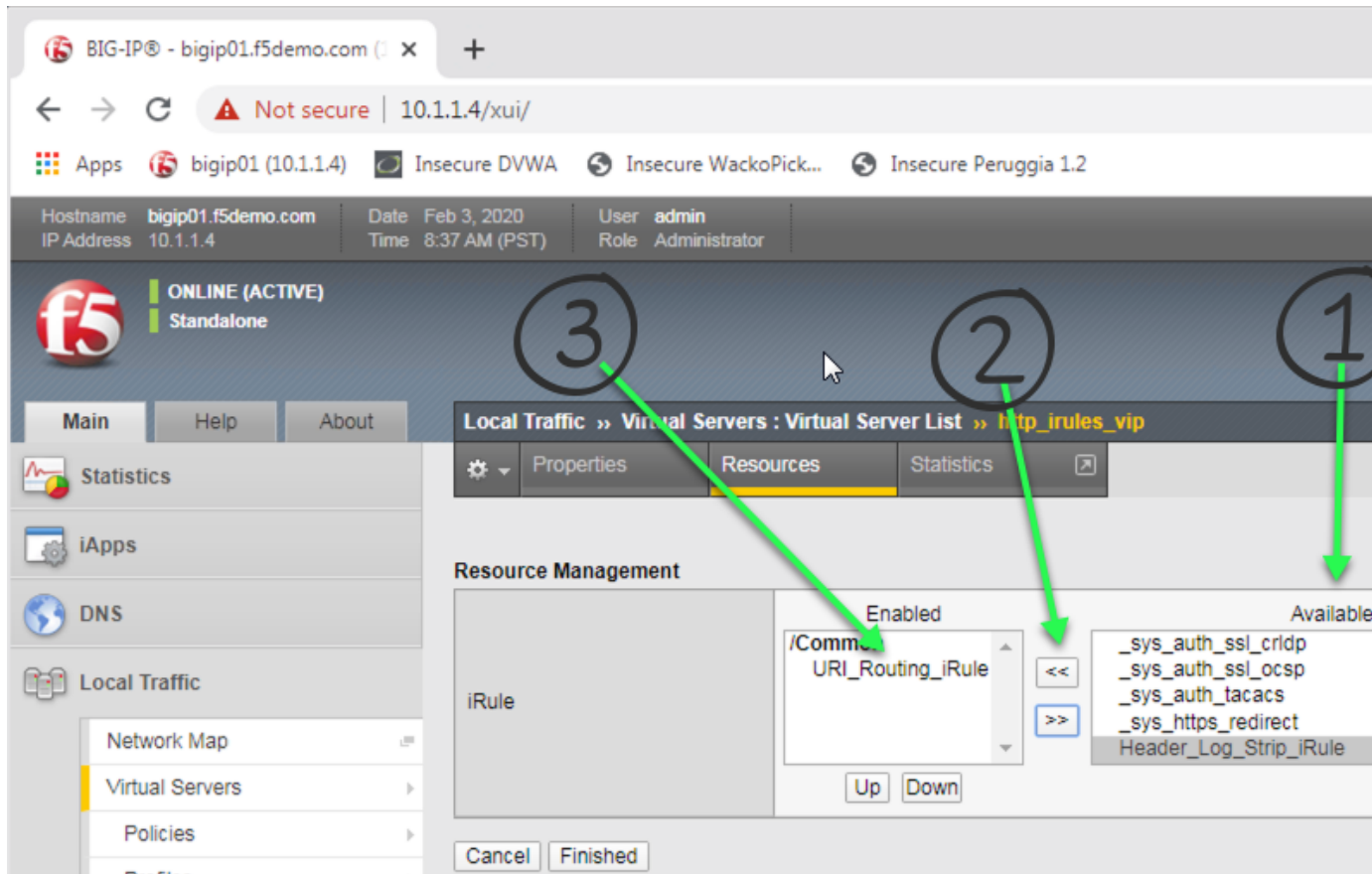


11. Click on the **Resources** tab

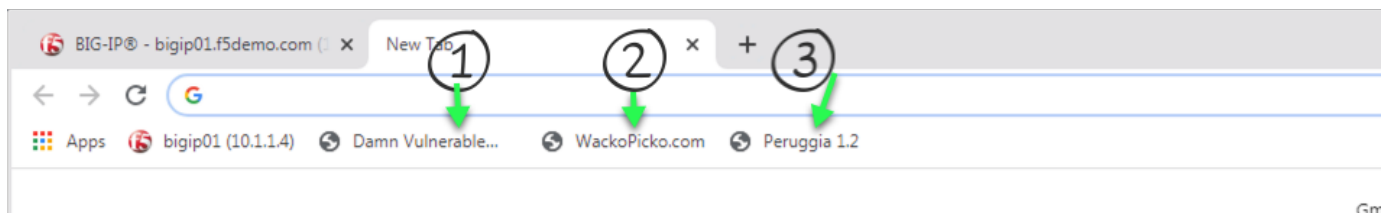
12. Click **Manage** button for the iRules section



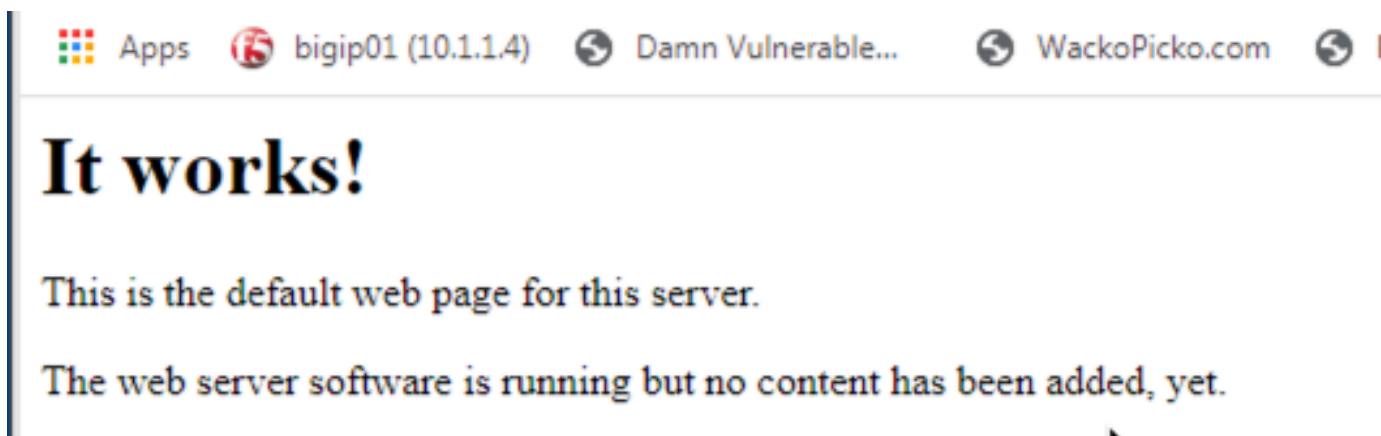
- Click on **URI_Routing_iRule** from the Available box and click the << button, thus moving it to the Enabled box.



- Click the **Finished** button
- Open a new tab in Chrome
- Enter <http://dvwa.f5lab.com/> and ensure you get there
- Now enter <http://peruggia.f5lab.com/> and ensure you get to the app
- Finally, enter <http://wackopicko.f5lab.com/> and ensure you can get to that app



- If you see this image below - it means your iRule did not work.



Hint: If you need a basic hint here is some example code:

Here is a link to DevCentral: https://clouddocs.f5.com/api/irules/HTTP__host.html

If you are really stuck, here is what we are looking for:

1. When HTTP_Request comes in
2. Evaluate the HTTP_host name
3. If it matches send it to the correct pool.
4. Loop through all the host names you want to match on and continue to direct to the correct pools.
5. Now you should have enough to understand and the majority of code needed to create the iRule. If not here is the complete iRule.

Lab 2 - Log and Change Headers

Your iRule should:

1. Log all HTTP **request** headers.
2. Log all HTTP **response** headers.
3. Remove the header named **Server** from all HTTP responses.

Attention: OPTIONAL: Instead of removing the **Server** header in the response, change the value of the **Server** header to **Microsoft-IIS/7.0**.

Important:

- Estimated completion time: 15 minutes

1. Open Chrome Browser
2. Enter <https://bigip1> into the address bar and hit Enter

f5 BIG-IP Configuration Utility
F5 Networks, Inc.

Hostname
bigip01.f5demo.com

IP Address
10.1.1.4

Username
admin

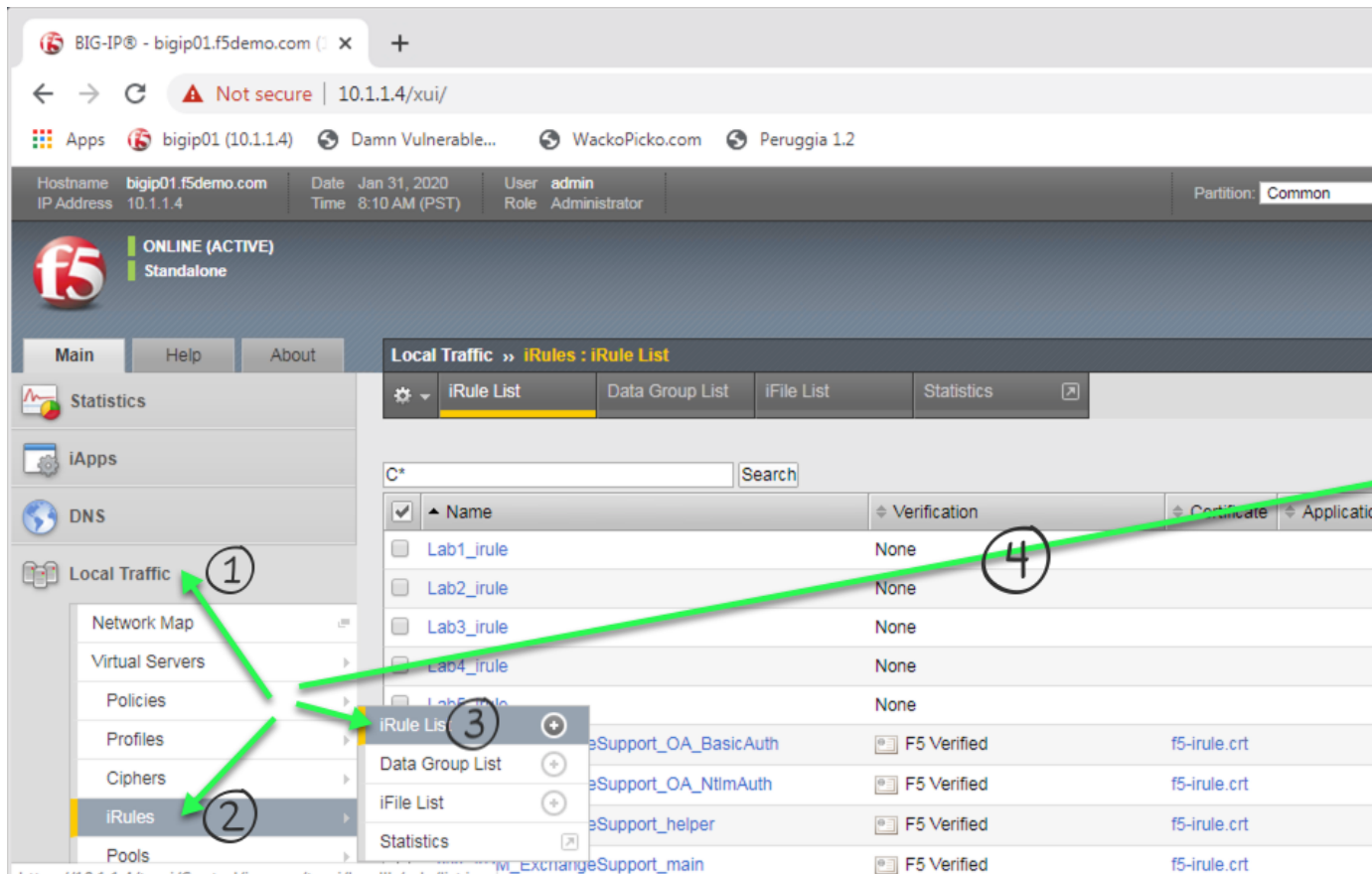
Password
.....

Log in

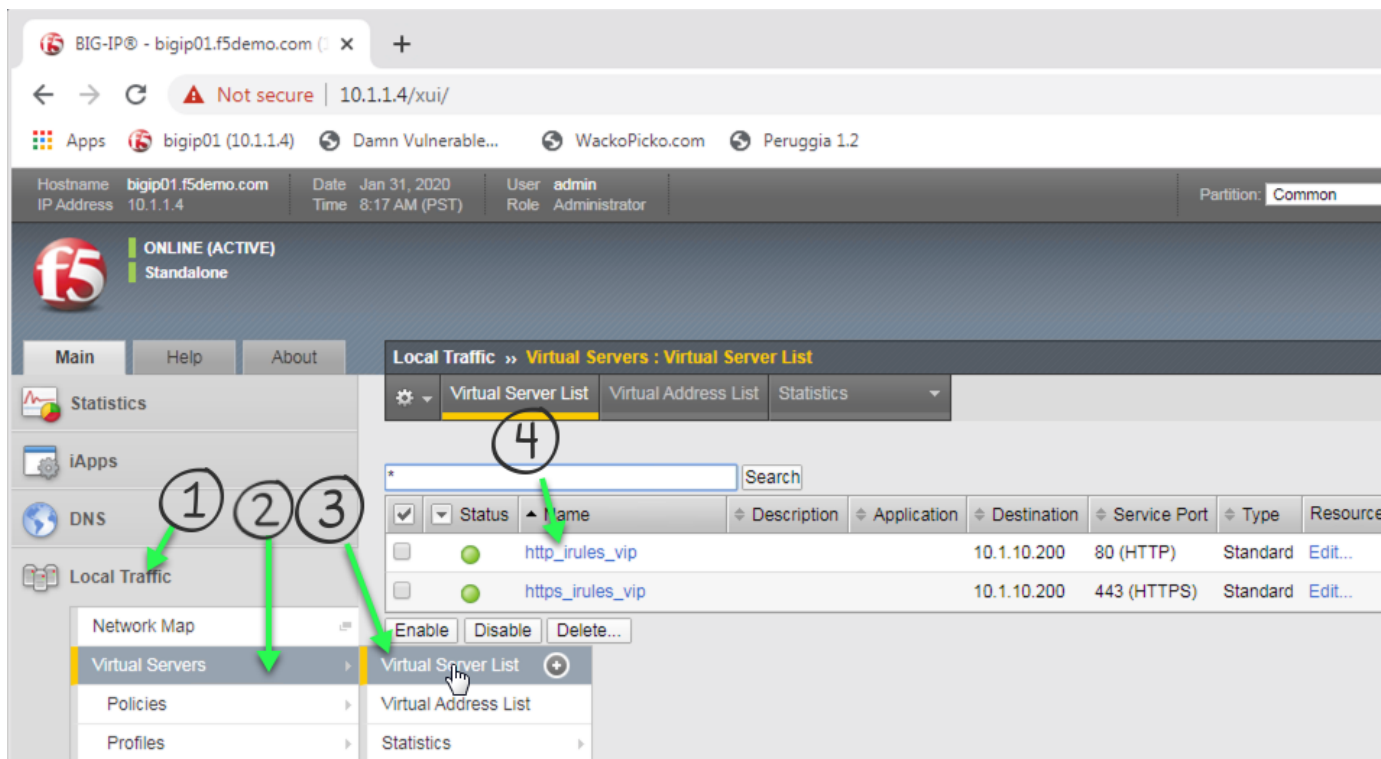
Welcome to the BIG-IP Configuration Utility.

Log in with your username and password using the fields on the

3. **Login with username: admin password: admin.F5demo.com**
4. Click Local Traffic -> iRules -> iRules List
5. Click **Create** button

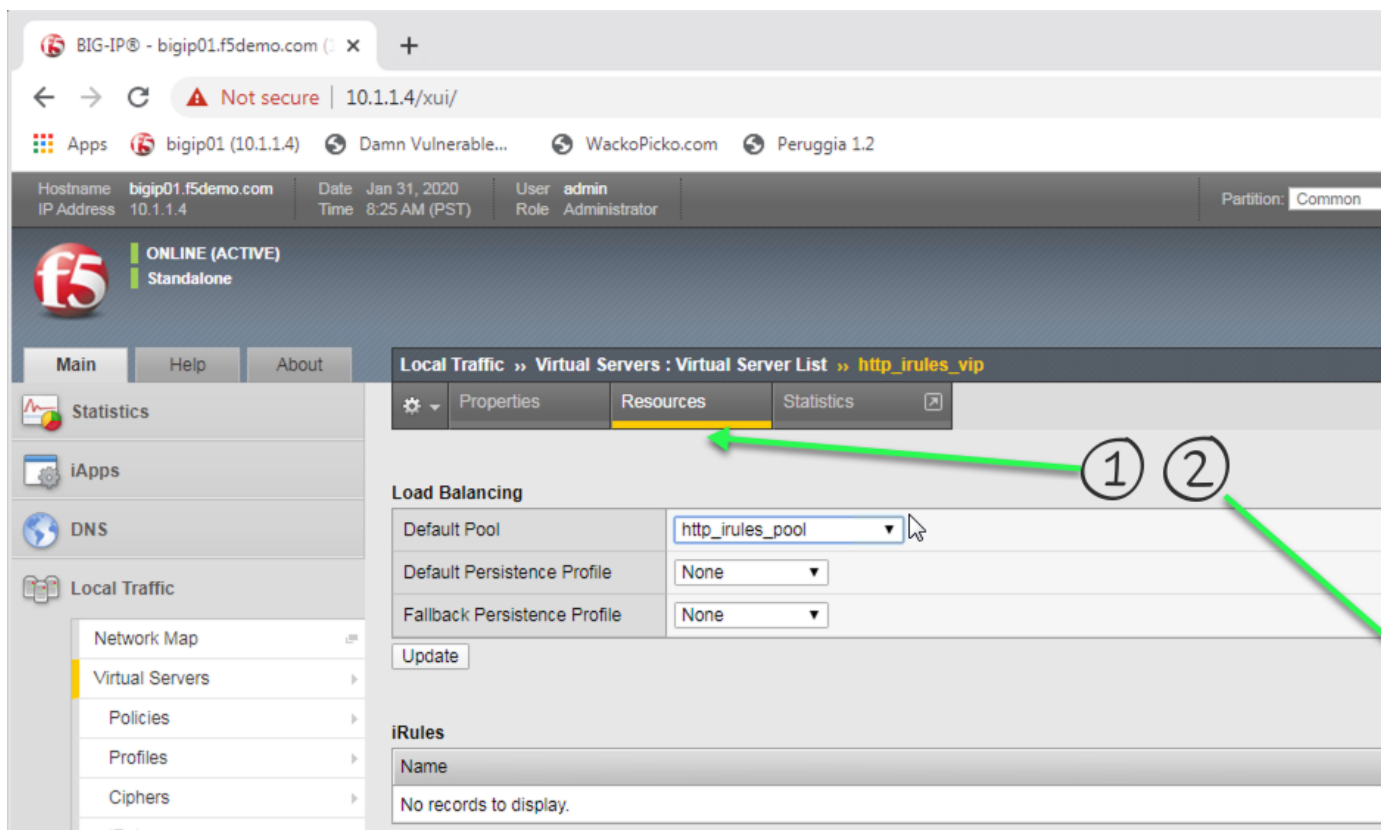


6. Enter Name of **Header_Log_Strip_iRule**
7. Enter Your Code
8. Click **Finished**
9. Click Local Traffic -> Virtual Servers -> Virtual Server List
10. Click on **http_irules_vip**

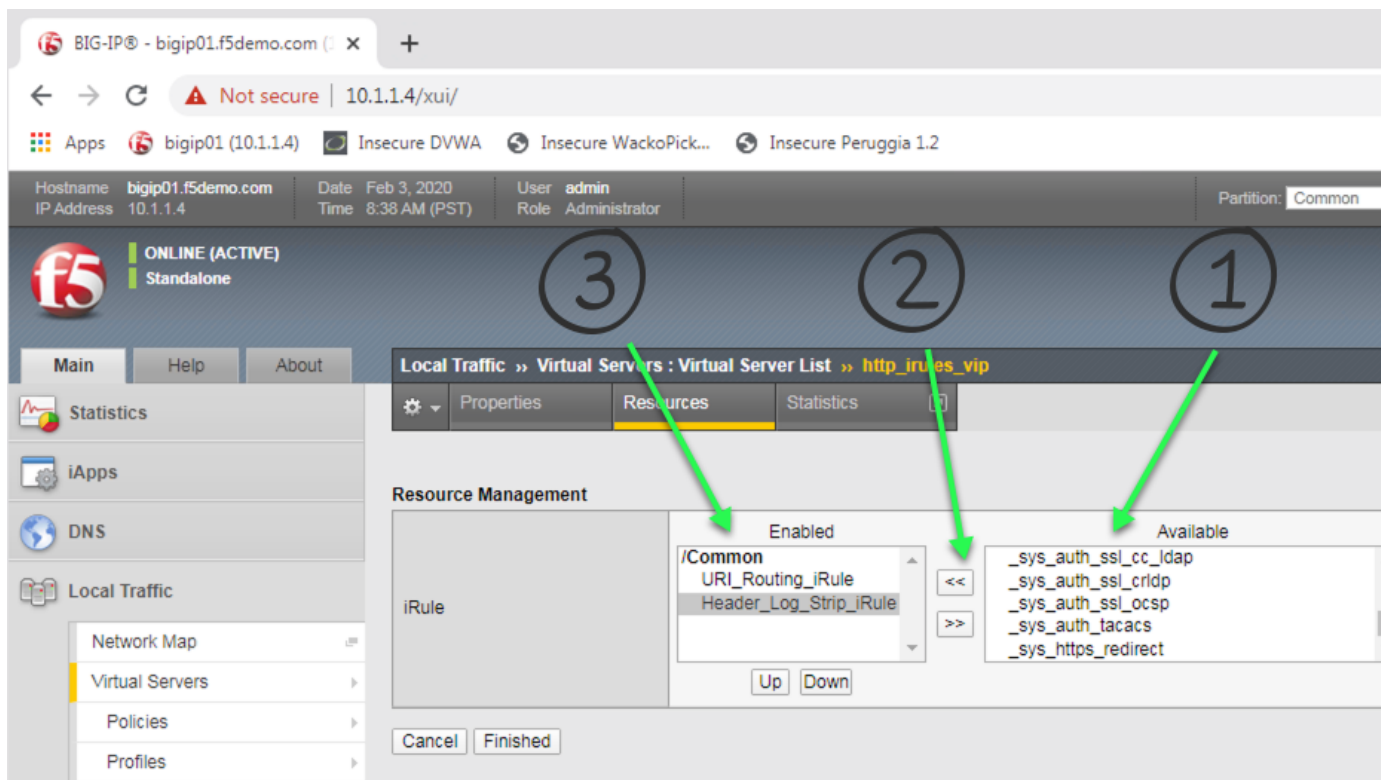


11. Click on the **Resources** tab

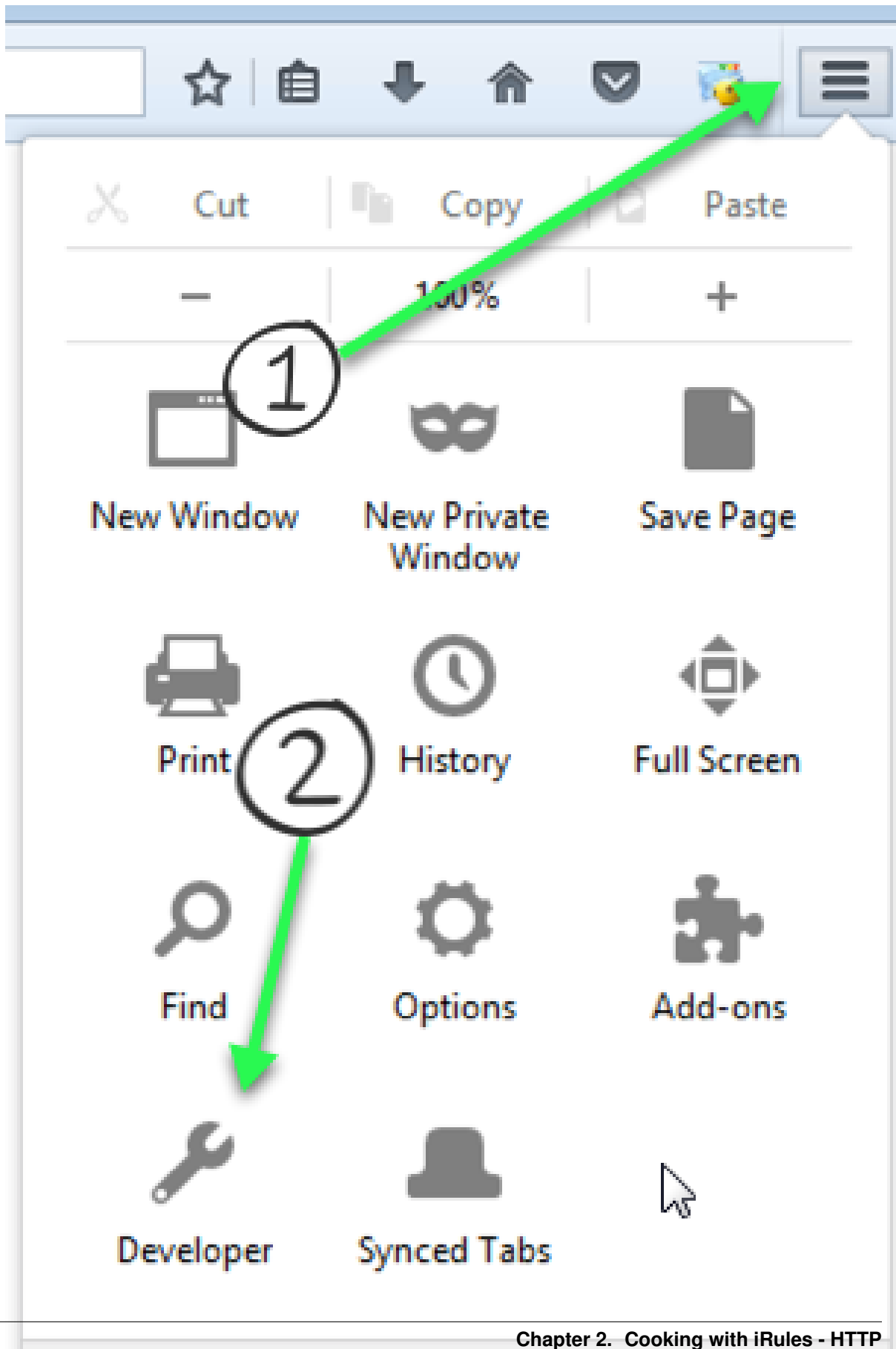
12. Click **Manage** button for the iRules section



13. Click on Header_Log_Strip_iRule from the Available box and click the << button, thus moving it to the Enabled box, your first and now second iRule should be in the Enabled box.



14. Click the **Finished** button
15. Open the Firefox browser
16. Click the 3 horizontal line button on the far right of the address bar
17. Use **developer tools** in Mozilla, or use Chrome to view headers



18. Enter <http://dvwa.f5lab.com/> and ensure you get there
19. Now enter <http://wackopicko.f5lab.com/>
20. Finally, enter <http://peruggia.f5lab.com/> and ensure you can get to that app
21. Look at the headers for each of your requests. Did you log them all? What is the value of the Server header?

The screenshot shows a web browser window with the URL <http://peruggia.f5lab.com/>. The page displays a logo and navigation links. Below the header is a large image of a hedgehog with its mouth open. To the right of the image is a 'Comments' section with a text input field and a 'Comment on this picture' button. A callout box with the text 'No Server Header' points to the right side of the page. On the right side of the browser window, the Network tab is open, showing the request headers for the page. The 'Server' header is listed as 'Microsoft-IIS/7.0'.

Attention: OPTIONAL: Instead of removing the **Server** header in the response, change the value of the **Server** header to **Microsoft-IIS/7.0**.

The screenshot shows the Network tab in a web browser. The 'Headers' section is expanded, showing the 'Request Headers' and 'Response Headers'. The 'Server' header is highlighted with a green arrow and a circled number 4. The 'Server' header value is 'Microsoft-IIS/7.0'. Other headers include 'Date', 'X-Powered-By', 'Expires', 'Cache-Control', 'Pragma', 'Vary', 'Content-Encoding', 'Content-Length', 'Content-Type', 'Connection', and 'Status'.

Hint: Basic Hint if you need a hint here is some example code:

Link to DevCentral: https://clouddocs.f5.com/api/irules/HTTP__header.html

If you are really stuck, here is what we are looking for:

1. When HTTP_Request comes in
 2. Log the headers from the HTTP_REQUEST
 3. When HTTP_RESPONSE comes back
 4. Log the response headers
 5. Now remove the HTTP::header named Server
 6. Now you should have enough to understand and the majority of code to create the iRule. If not here is the complete iRule.
-

Lab 3 - HTTP to HTTPS Redirect

1. Create an iRule to redirect all traffic that arrives at an HTTP virtual server to be redirected to the same IP address but using an HTTPS port.
 2. The full original HTTP request should be maintained when re-directing. Example <http://my.domain.com/app1/index1.html> should redirect to <https://my.domain.com/app1/inex.html>
 3. Traffic goes to the HTTPS virtual server should still perform the pool selection and should still perform the header stripping from previous labs.
-

Important:

- Estimated completion time: 20 minutes
-

1. Open Chrome Browser
2. Enter <https://bigip1> into the address bar and hit Enter

f5 BIG-IP Configuration Utility
F5 Networks, Inc.

Hostname
bigip01.f5demo.com

IP Address
10.1.1.4

Username
admin

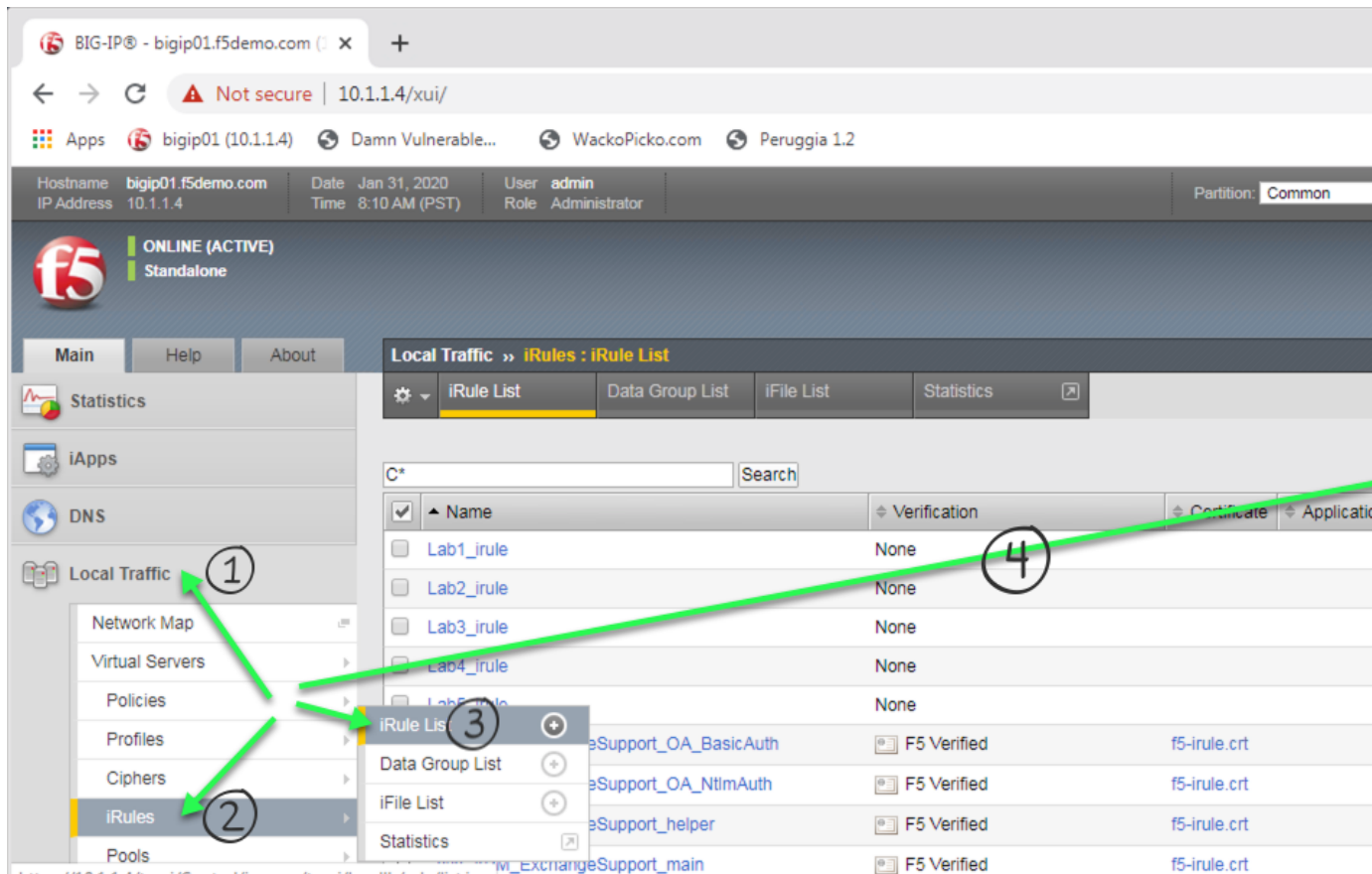
Password
.....

Log in

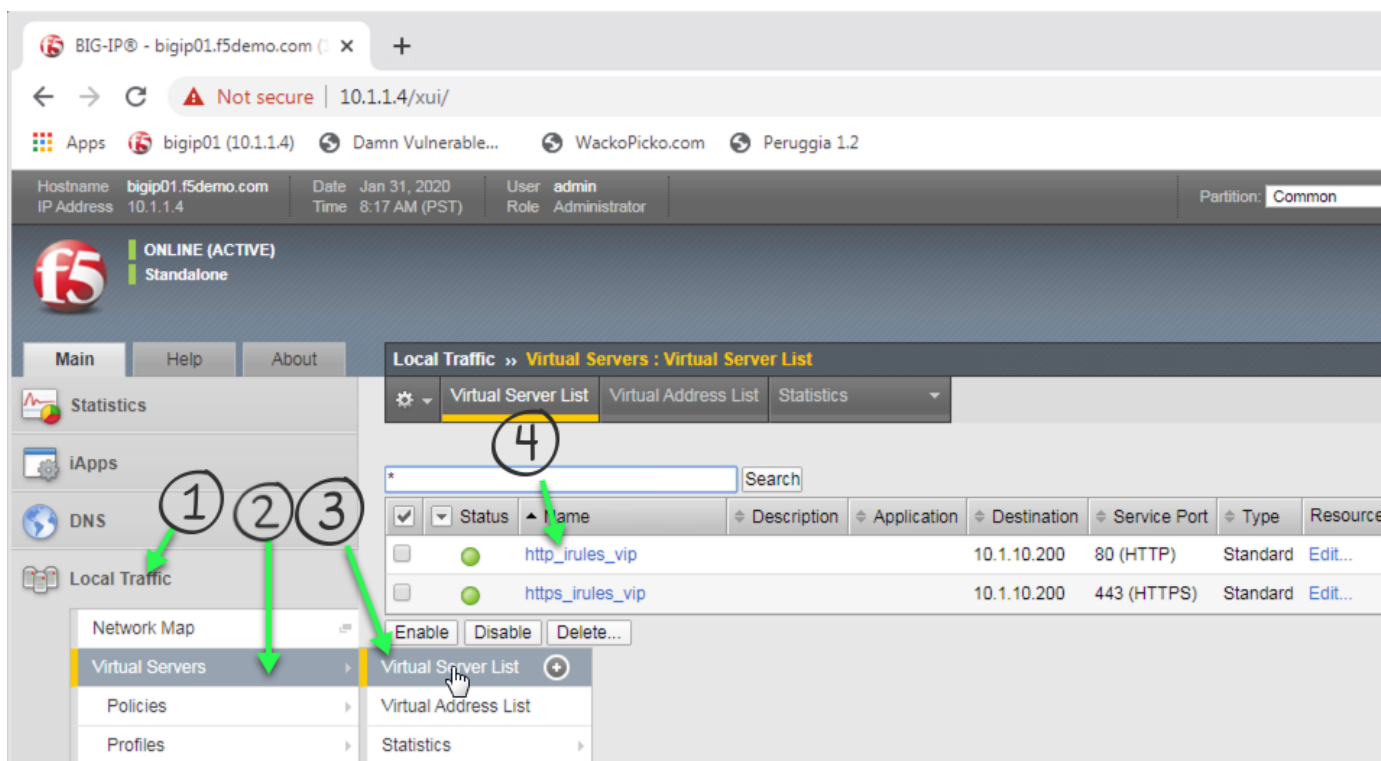
Welcome to the BIG-IP Configuration Utility.

Log in with your username and password using the fields on the

3. **Login with username: admin password: admin.F5demo.com**
4. Click Local Traffic -> iRules -> iRules List
5. Click **Create** button

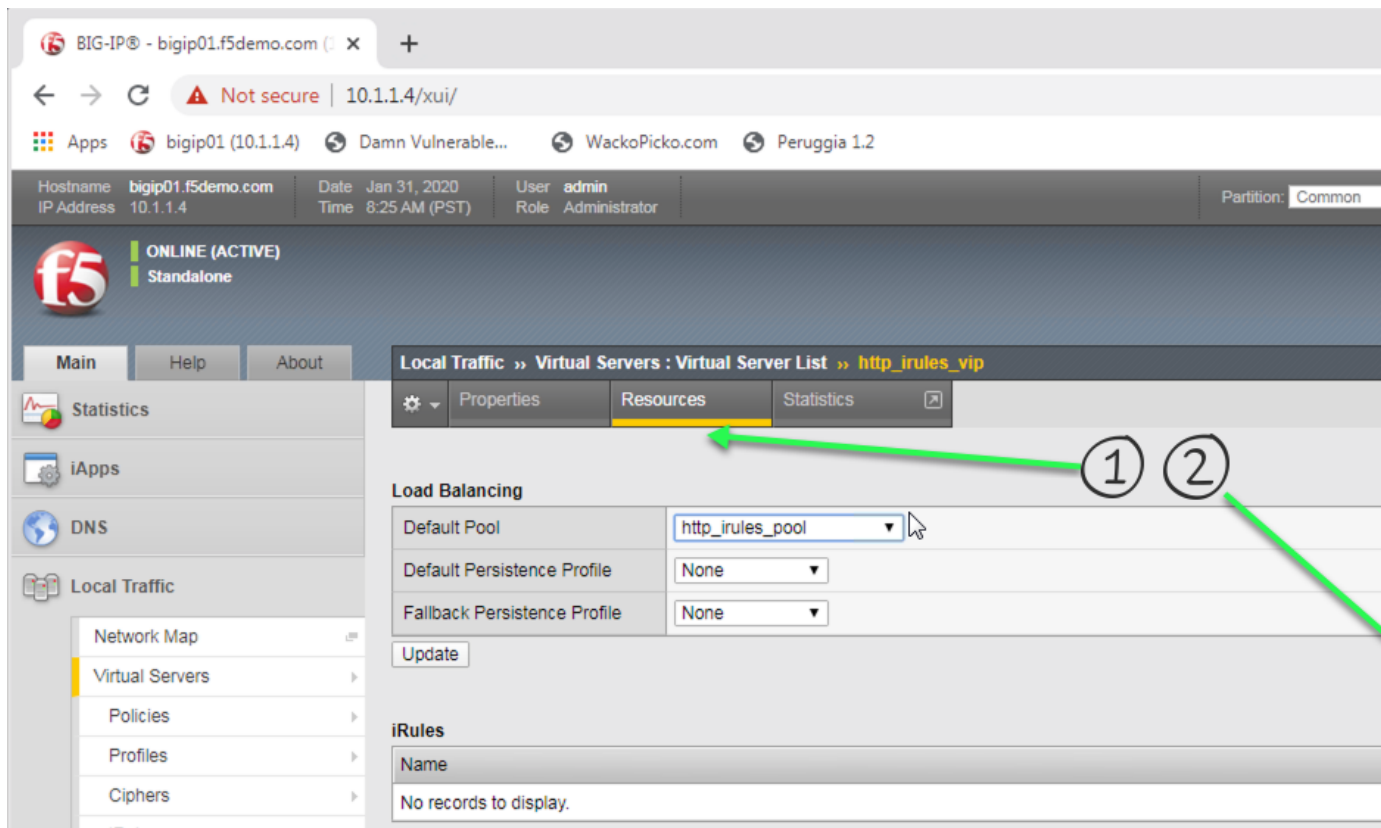


6. Enter Name of **HTTP_to_HTTPS_iRule**
7. Enter Your Code
8. Click **Finished**
9. Click Local Traffic -> Virtual Servers -> Virtual Server List
10. Click on **http_irules_vip**

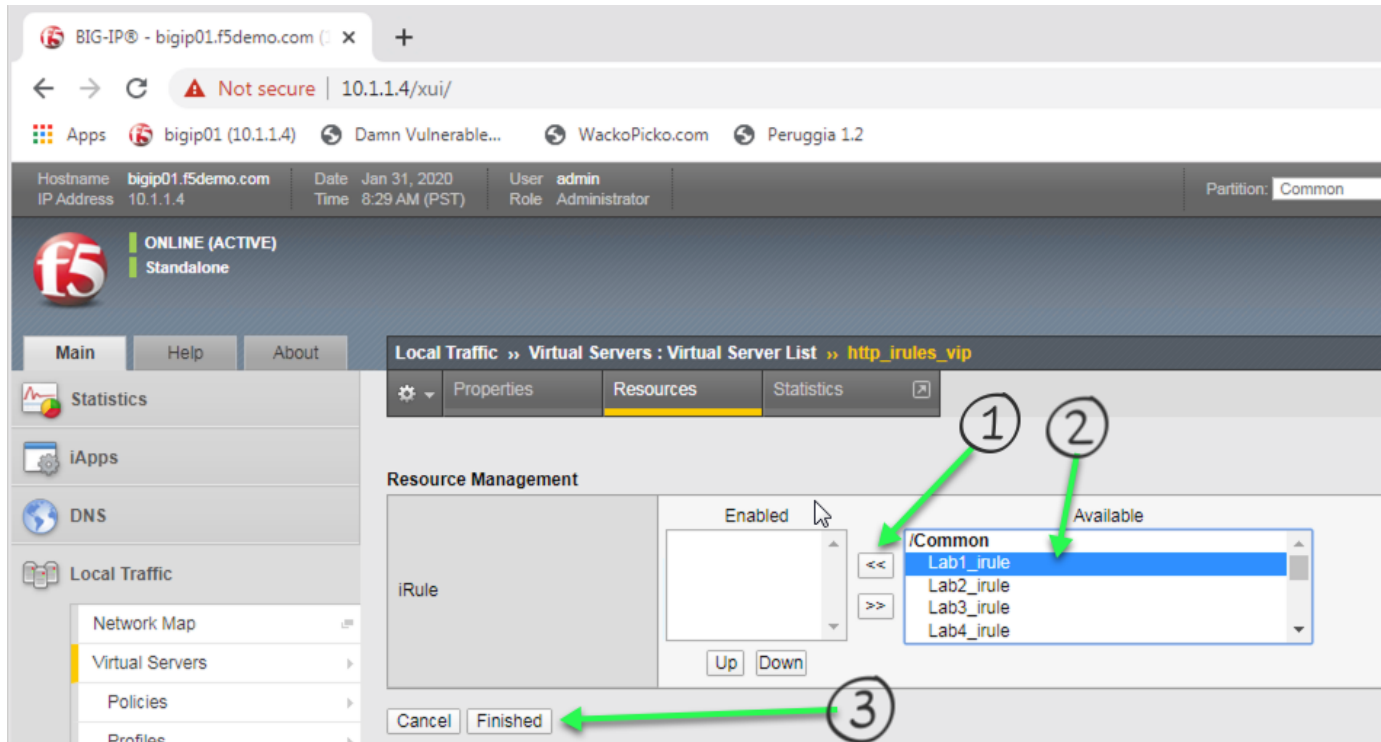


11. Click on the **Resources** tab

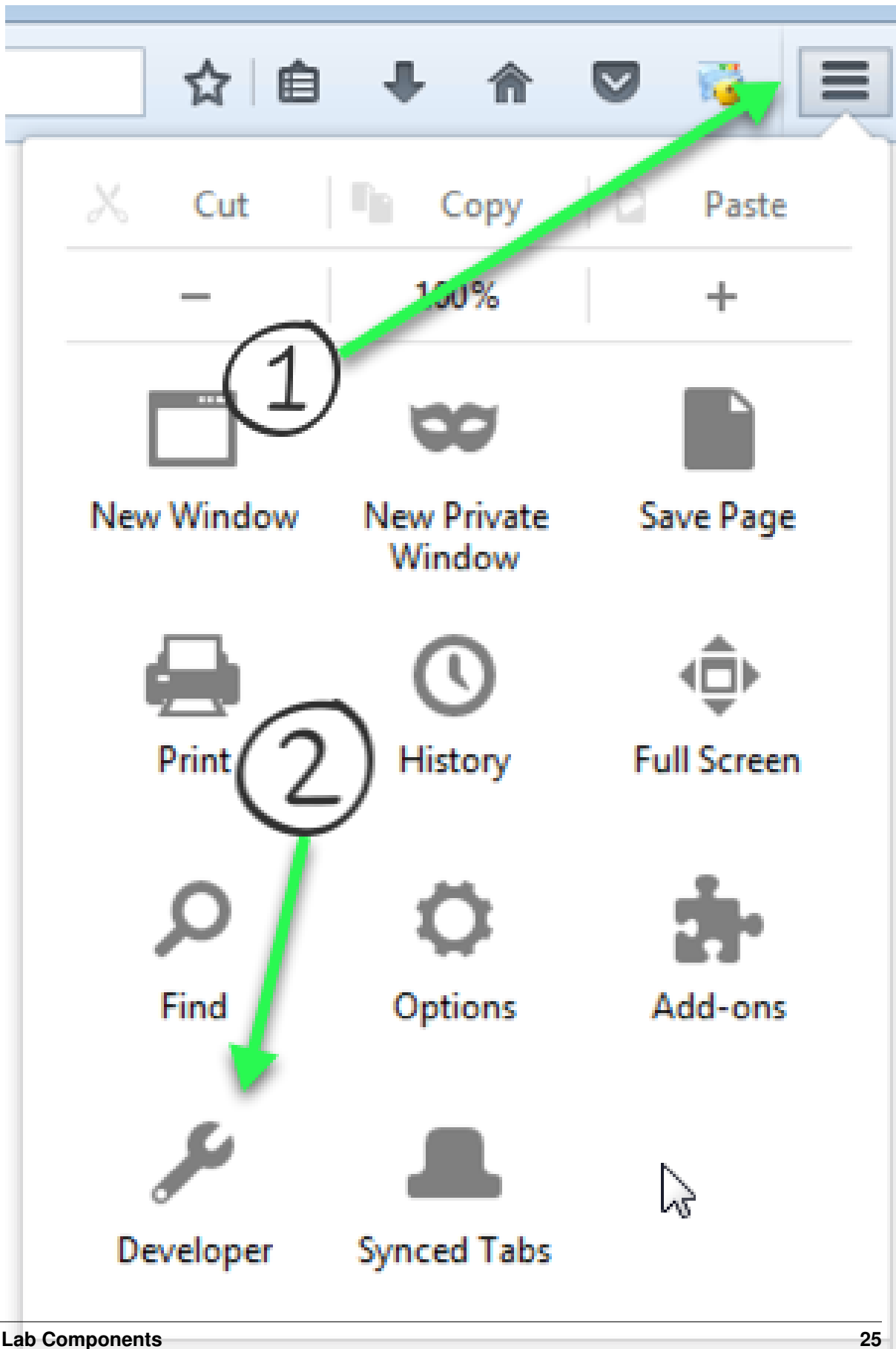
12. Click **Manage** button for the iRules section



- Click on HTTP_to_HTTPS_iRule from the Available box and click the << button, thus moving it to the Enabled box, your first and now second iRule should be in the Enabled box.



- Click the **Finished** button
- Open the Firefox browser
- Click the 3 horizontal line button on the far right of the address bar
- Use developer tools in Mozilla, or use Chrome to view headers



18. Look at the headers for each of your requests. Did you log them all? What is the value of the Server header? None of this should have changed since the last lab.

Hint: Basic Hint if you need a hint here is some example code:

Link to DevCentral: https://clouddocs.f5.com/api/irules/HTTP__redirect.html

If you are really stuck, here is what we are looking for:

1. When HTTP_Request comes in
 2. Redirect from HTTP to HTTPS
 3. Now you should have enough to understand and the majority of code to create the iRule. If not here is the complete iRule.
-

Lab 4 - Stream Profile

Create a Stream Profile to change the body of the DVWA site

Important:

- Estimated completion time: 10 minutes
-

1. Open Chrome Browser
2. Enter <https://bigip1> into the address bar and hit Enter



f5 BIG-IP Configuration Utility
F5 Networks, Inc.

Hostname
bigip01.f5demo.com

IP Address
10.1.1.4

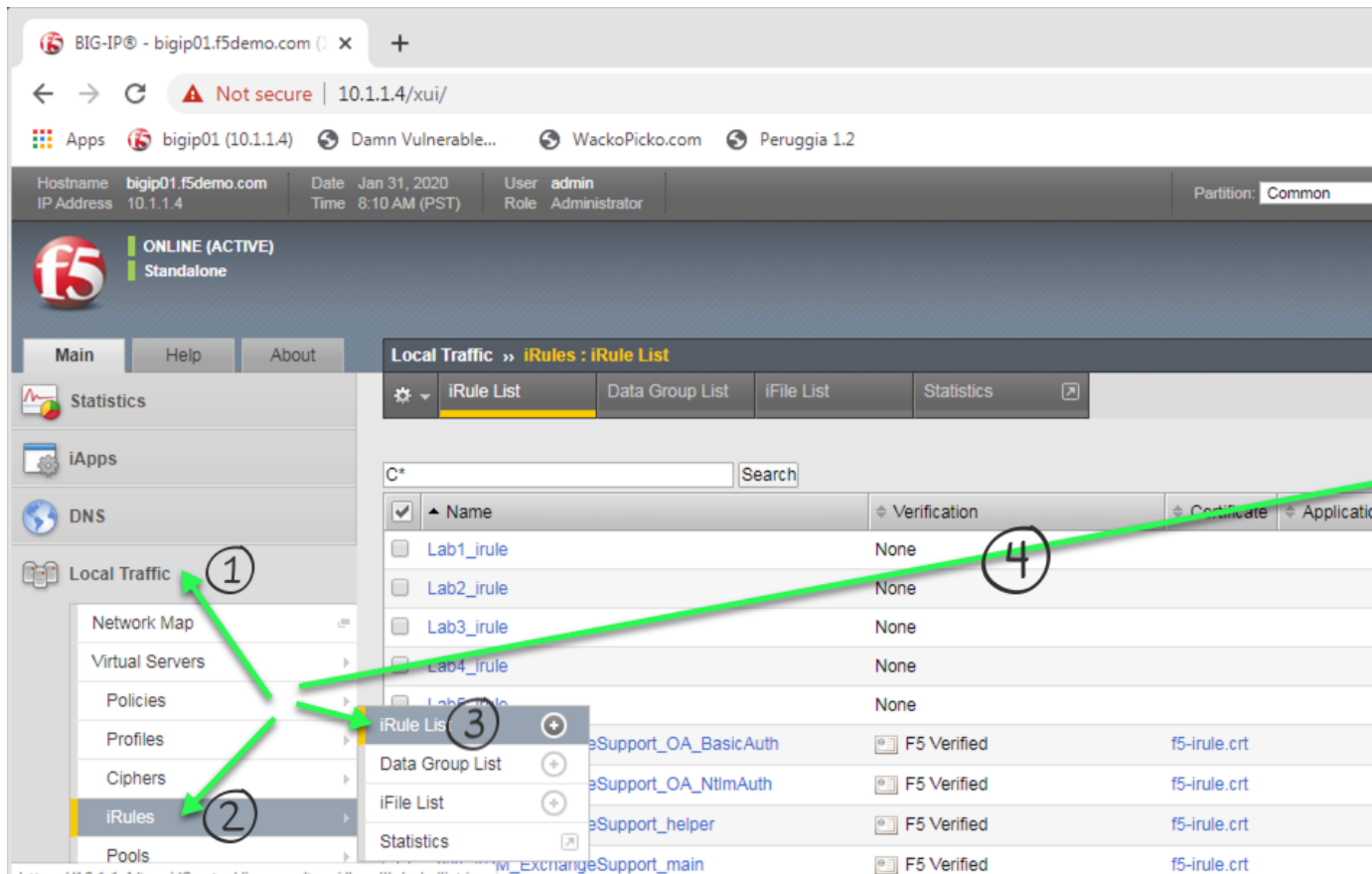
Username
admin

Password
.....

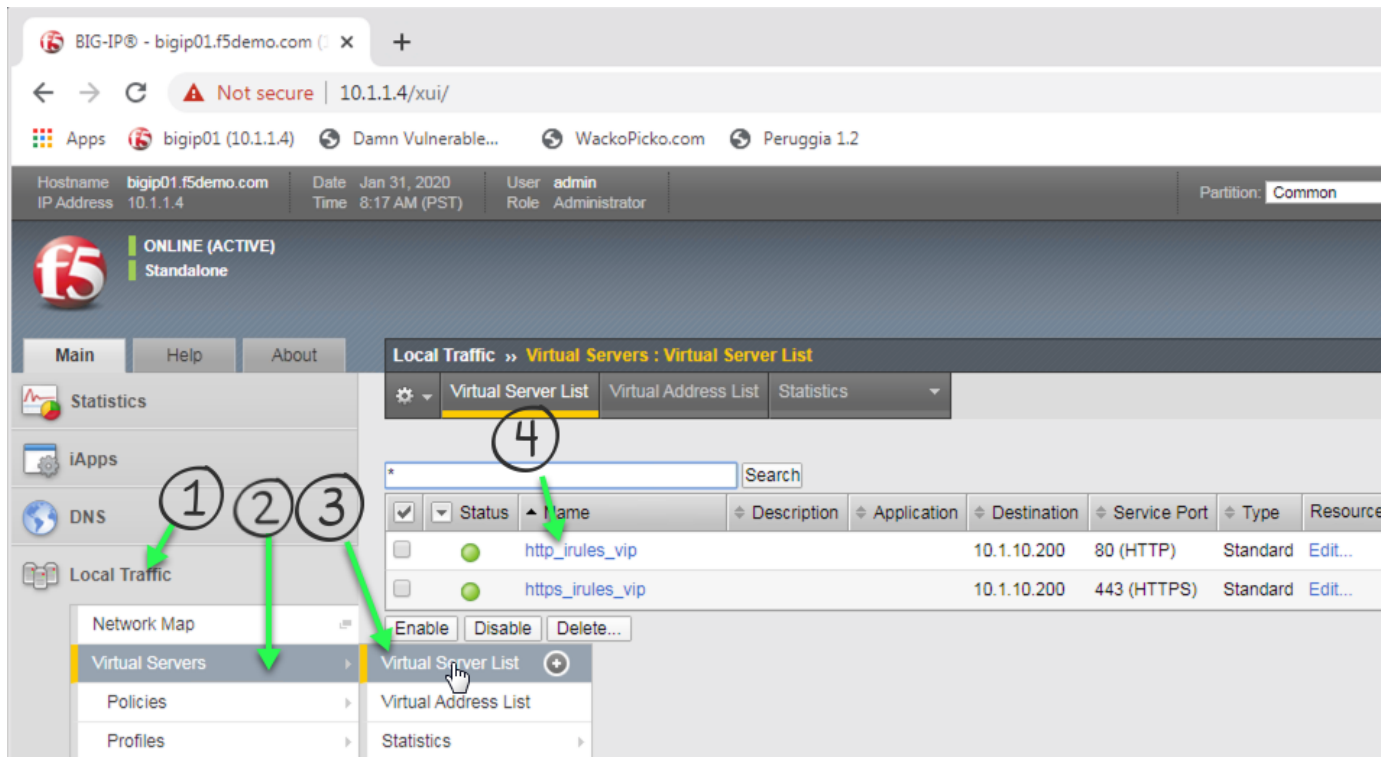
Log in

Welcome to the BIG-IP Configuration Utility.
Log in with your username and password using the fields on the

3. 1. **Login with username: admin password: admin.F5demo.com**
4. Click Local Traffic -> iRules -> iRules List
5. Click **Create** button



6. Enter Name of **Stream_iRule**
7. Enter Your Code
8. Click **Finished**
9. Click Local Traffic -> Virtual Servers -> Virtual Server List
10. Click on **https_irules_vip**



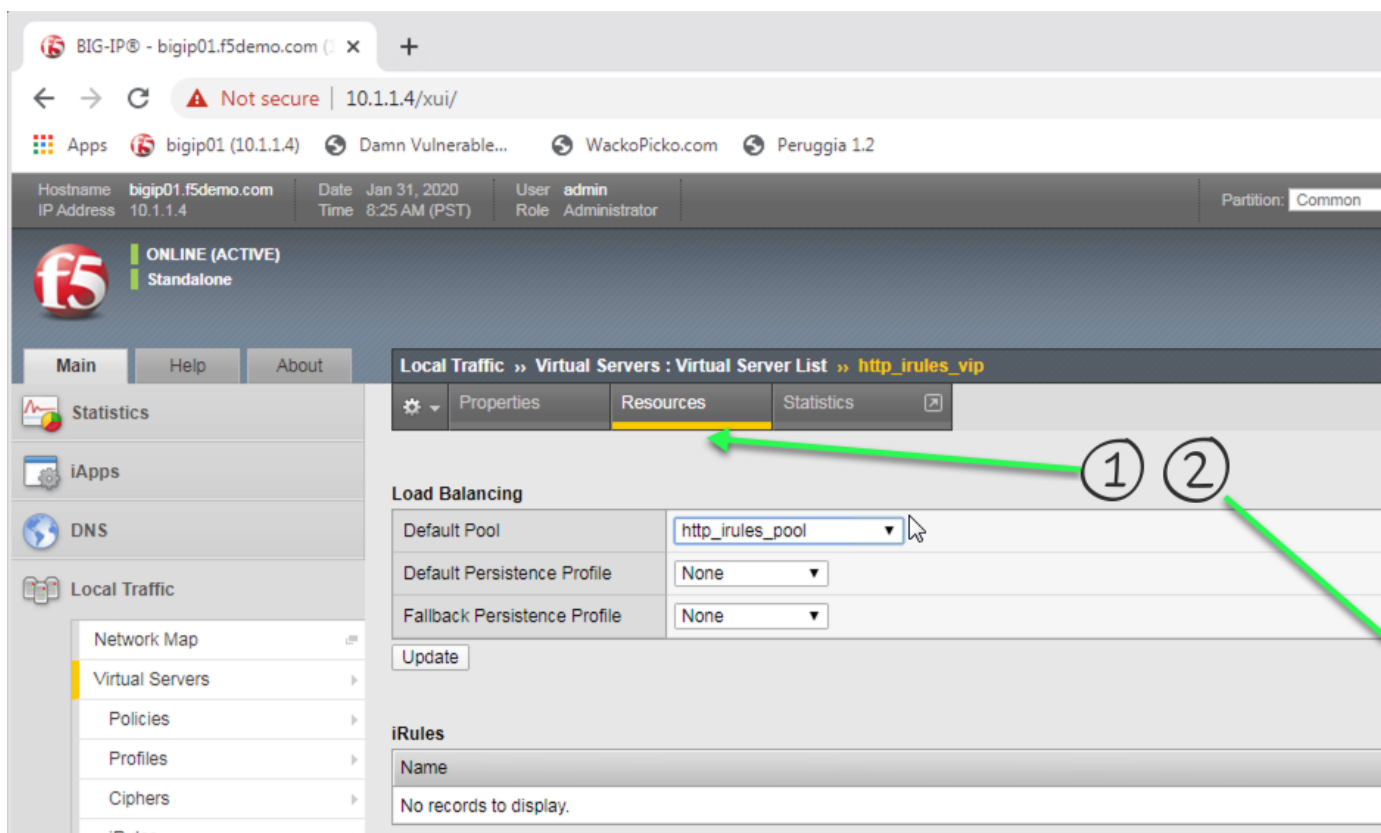
The screenshot shows the F5 iRules configuration interface. The left sidebar contains a tree view with categories: Main, Help, About, Statistics, iApps, DNS, Local Traffic, Acceleration, Device Management, Shared Objects, Network, and System. The 'Local Traffic' category is expanded, showing sub-items: Network Map, Virtual Servers, Policies, Profiles, Ciphers, iRules, Pools, Nodes, Monitors, Traffic Class, and Address Translation. The 'Virtual Servers' sub-item is selected, and the 'https_irules_vip' virtual server is highlighted. The main panel displays the configuration for 'https_irules_vip'. The 'General Properties' section is visible, showing fields for Name, Partition / Path, Description, Type, Source Address, Destination Address/Mask, Service Port, Notify Status to Virtual Address, Availability, Syncookie Status, and State. The 'Configuration' section is also visible, showing fields for Protocol, Protocol Profile (Client), Protocol Profile (Server), HTTP Profile (Client), HTTP Profile (Server), HTTP Proxy Connect Profile, FTP Profile, RTSP Profile, SOCKS Profile, and Stream Profile. A green arrow points to the 'Advanced' dropdown in the Configuration section, labeled with a circled '1'. Another green arrow points to the 'Stream Profile' dropdown, labeled with a circled '2'.

General Properties	
Name	https_irules_vip
Partition / Path	Common
Description	
Type	Standard
Source Address	Host Address List 0.0.0.0/0
Destination Address/Mask	Host Address List 10.1.10.200
Service Port	Port Port List 443 HTTPS
Notify Status to Virtual Address	<input checked="" type="checkbox"/>
Availability	Available (Enabled) - The virtual server is available
Syncookie Status	Inactive
State	Enabled

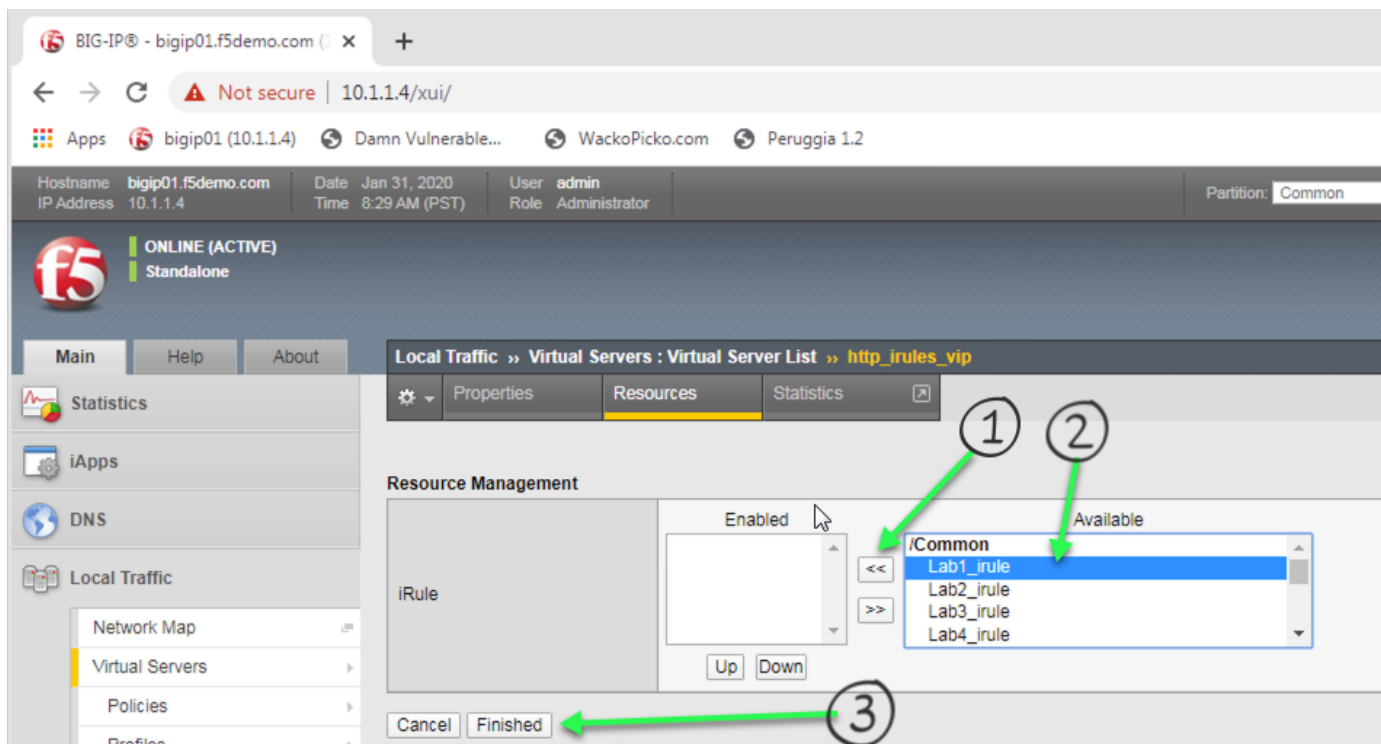
Configuration: Advanced	
Protocol	TCP
Protocol Profile (Client)	tcp
Protocol Profile (Server)	(Use Client Profile)
HTTP Profile (Client)	http
HTTP Profile (Server)	(Use Client Profile)
HTTP Proxy Connect Profile	None
FTP Profile	None
RTSP Profile	None
SOCKS Profile	None
Stream Profile	None

13. Scroll to the bottom and click the **Update** button
14. Click on the **Resources** tab at the top of the page.

15. Click **Manage** button for the iRules section



16. Click on Stream_iRule from the Available box and click the << button, thus moving it to the Enabled box, your first and now second iRule should be in the Enabled box.



17. Click the **Finished** button
18. Open the Firefox browser
19. Enter <https://dvwa.f5lab.com> and ensure you get there and it is HTTPS and that the word **Damn** is replaced with **Darn**

Hint: Basic Hint if you need a hint here is some example code:

Link to DevCentral: <https://clouddocs.f5.com/api/irules/STREAM.html>

If you are really stuck, here is what we are looking for:

1. When HTTP_Request comes in
 2. Second we need to disable both encoding the stream profile for the request
 3. When HTTP_RESPONSE comes back
 4. Next we need to change our stream matching string and turn on the stream profile again.
 5. Now you should have enough to understand and the majority of code to create the iRule. If not here is the complete iRule.
-

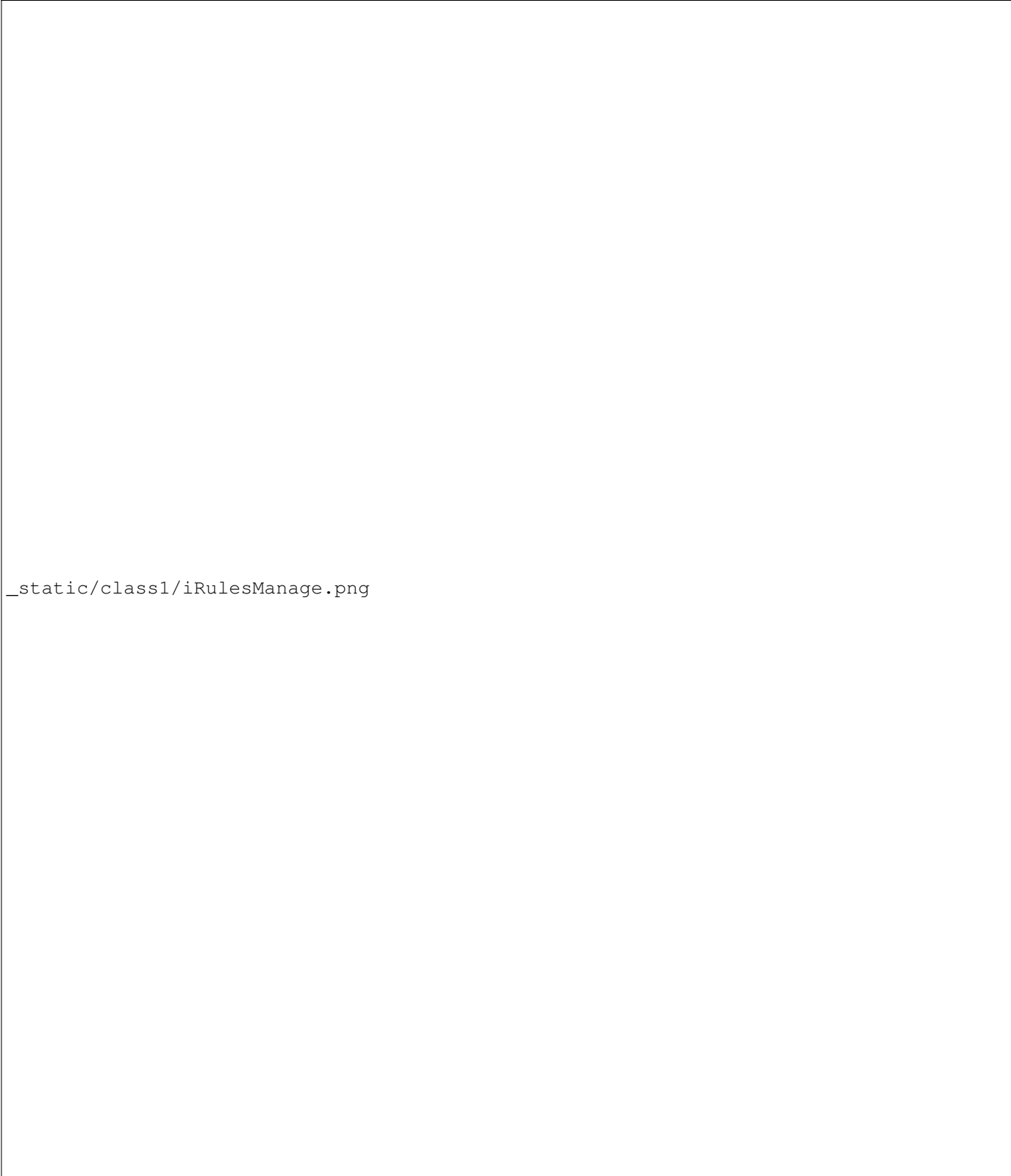
Lab 5 - HTTP Payload Manipulation

Collect an HTTP payload, change it, and release it to the client. As in the previous lab replace Damn with Darn, or get creative.

Important:

- Estimated completion time: 20 minutes
-

1. Open Chrome Browser
2. Enter <https://bigip1> into the address bar and hit Enter
3. Login with username: admin password: admin.F5demo.com
4. Click Local Traffic -> iRules -> iRules List
5. Click Create button
6. Enter Name of HTTP_Payload_iRule
7. Enter Your Code
8. Click Finished
9. Click Local Traffic -> Virtual Servers -> Virtual Server List
10. Click on https_irules_vip
11. Click on the Resources tab
12. Click Manage button for the iRules section



`_static/class1/iRulesManage.png`

13. What should you do here? (Hint: Remove Stream_iRule)
14. Click the Finished button

15. Open the Firefox browser

16. Enter <https://dvwa.f5lab.com> and ensure you get there and it is HTTPS

Hint: Basic Hint

if you need a hint here is some example code:

Link to DevCentral: https://clouddocs.f5.com/api/irules/HTTP__collect.html

Link to DevCentral: https://clouddocs.f5.com/api/irules/HTTP__release.html

If you are really stuck, here is what we are looking for:

1. When HTTP_Request comes in
 2. Second change the version of HTTP and disable compression for the request
 3. When HTTP_RESPONSE comes back
 4. Next we need to collect some HTTP::collect some data.
 5. Now when we get HTTP_RESPONSE_DATA
 6. Now we will set some find and replace strings.
 7. Finally we will perform a regsub on the payload and replace with new text.
 8. Now you should have enough to understand and the majority of code to create the iRule. If not here is the complete iRule.
-

iRules Summary

Here is a summary of all the irules used in this lab

Lab 1 - Complete iRule

Completed iRule

```
# if / elseif version

when HTTP_REQUEST {
    if {[HTTP::host] equals "dvwa.f5lab.com"} {
        pool dvwa_pool_http
    } elseif {[HTTP::host] equals "peruggia.f5lab.com"} {
        pool peruggia_http_pool
    } elseif {[HTTP::host] equals "wackopicko.f5lab.com"} {
        pool wackopicko_http_pool
    }
}

# switch version

when HTTP_REQUEST {
    switch [HTTP::host] {
        dvwa.f5lab.com { pool dvwa_pool_http }
        peruggia.f5lab.com { pool peruggia_http_pool }
        wackopicko.f5lab.com { pool wackopicko_http_pool }
    }
}
```

(continues on next page)

(continued from previous page)

```

    }
}

# Advanced, data group lookup version!

when HTTP_REQUEST {
    if { [class match [HTTP::host] equals "hostnames_dg"] } {
        pool [class lookup [HTTP::host] "hostnames_dg"]
    }
}

```

Lab 2 - Complete iRule

Completed iRule

```

# Header_Strip_Log_iRule

when HTTP_REQUEST {
    log local0. "Request Headers: [HTTP::header names]"
}

when HTTP_RESPONSE {
    log local0. "Response Headers: [HTTP::header names]"
    HTTP::header remove Server
}

# Advanced - Bonus and prettier

when HTTP_REQUEST {
    foreach header [HTTP::header names] {
        log local0. "Request Header $header: [HTTP::header $header]"
    }
}

when HTTP_RESPONSE {
    foreach header [HTTP::header names] {
        log local0. "Response Header $header: [HTTP::header $header]"
        if {$header equals "Server"} {
            HTTP::header remove $header
        }
    }
    HTTP::header insert Server "Microsoft-IIS/8.0"
}

```

Lab 3 - Complete iRule

Completed iRule

```

# HTTP_to_HTTPS_iRule

```

(continues on next page)

(continued from previous page)

```
when HTTP_REQUEST {
    HTTP::redirect "https://[HTTP::host][HTTP::uri]"
}

# Factory F5 https redirect iRule

when HTTP_REQUEST {
    HTTP::redirect https://[getfield [HTTP::host] ":" 1][HTTP::uri]
}
```

Lab 4 - Complete iRule

Completed iRule

```
# Stream_iRule

when HTTP_REQUEST {
    HTTP::header remove Accept-Encoding
    STREAM::disable
}

when HTTP_RESPONSE {
    STREAM::expression @Damn@Darn@
    STREAM::enable
}
```

Lab 5 - Complete iRule

Completed iRule

```
# HTTP_Payload_iRule

when HTTP_REQUEST {
    HTTP::version 1.0
    HTTP::header remove Accept-Encoding
}

when HTTP_RESPONSE {
    HTTP::collect [expr 1024*1024]
}

when HTTP_RESPONSE_DATA {
    set find "Damn"
    set replace "***"

    if {[regsub -all $find [HTTP::payload] $replace new_response] > 0} {
        HTTP::payload replace 0 [HTTP::payload length] $new_response
    }
}
```

iRules Events

Here is the link to the iRule Events

Complete listing of events - <https://clouddocs.f5.com/api/irules/Events.html>

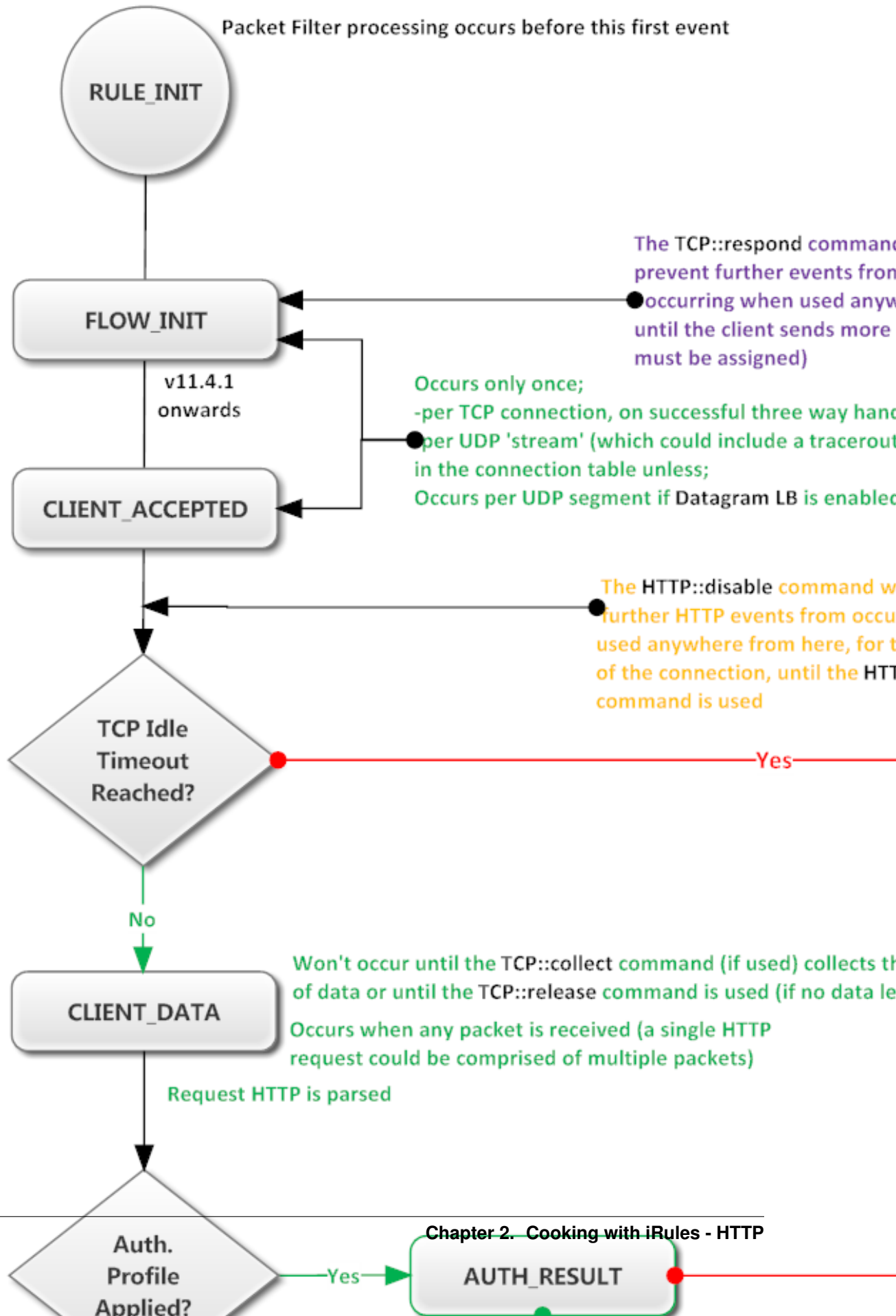
Master list of iRule Commands - <https://clouddocs.f5.com/api/irules/Commands.html>

BIG-IP Commands and Events by Version - https://clouddocs.f5.com/api/irules/BIGIP_Commands_by_Version.html

iRules HTTP Events

Here is the link to the iRule Events flow order

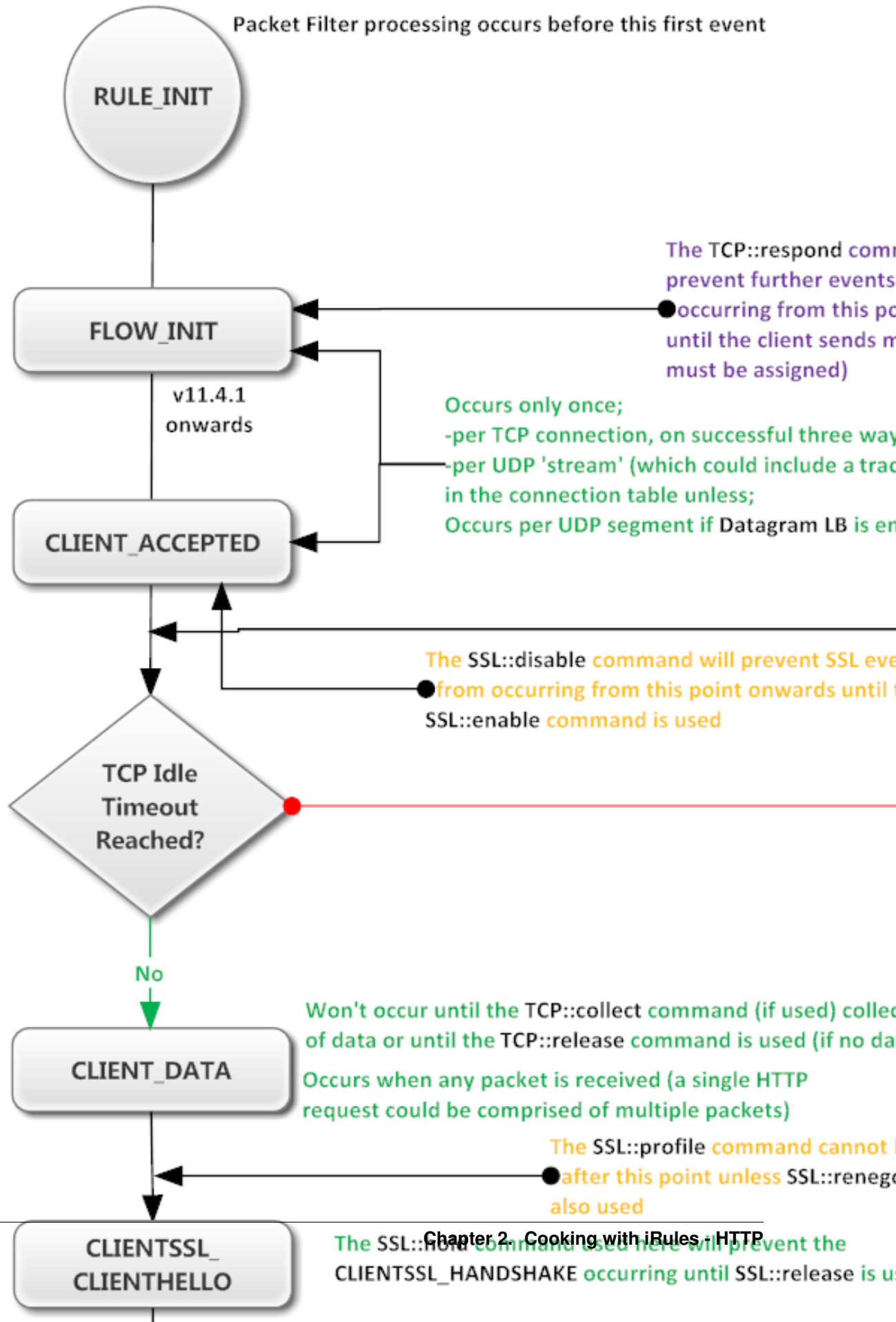
HTTP Flow Order grabbed from here - <https://devcentral.f5.com/s/contentdocument/0691T000005oRxAQAU>



iRules HTTPS Events

Here is the link to the iRule Events flow order

HTTPS Flow Order grabbed from here - <https://devcentral.f5.com/s/contentdocument/0691T000005nCt6QAE>



Cooking with iRules - Security/SSL

Security vulnerabilities are on the rise. In 2018, over 16,000 new CVEs were published, a more than 300% increase over 5 years (according to Skybox Security's 2019 Vulnerability and Threat Trends Report), and the number of known critical vulnerabilities is so large that application security teams simply cannot patch quickly enough. As you're no doubt aware by now, F5 iRules can often "fix" things that are missing or hard to solve in the existing application stack. Of course, custom code should never take the place of a good security product, but arguably there's no single security device that can address every vulnerability; and malware and other exploits generally move faster than software updates and patches. F5 iRules give you the power and flexibility to fill those gaps, and these labs provide just a taste of some of that power. In the following, you'll see a handful of ways to use iRules to defend and protect against malicious activity in your enterprise.

Ultimately though, this is **incomplete**. There's simply no way to include iRules code for every possible vulnerability, and we'd rather not hand you a 500-page guide. If you need to fix some specific security issue that we haven't included here, or simply have questions, please let the F5 DevCentral community know.

<https://devcentral.f5.com>

iRules rock!

3.1 Security/SSL iRules Labs

3.1.1 Lab 1 - TLS Version Control

Scenario

If you care about security at all, then you also need to care about SSL and TLS. These are the protocols that provide encryption to the Internet traffic. It's the "S" in HTTPS, FTPS, and IMAPS. You get the picture!

Without encryption, all communication between a client and server is visible for any third party to see and this can have some pretty devastating effects. Encryption is a cornerstone of network security. If you've been paying attention to the news the last few years, you're probably also aware that SSL and TLS are not without their own flaws. Exploits like BEAST, Heartbleed, and POODLE take advantage of specific holes in the SSL and TLS protocols. To stay ahead of these vulnerabilities, industry best practices suggest that we avoid some versions of these protocols specifically SSLv2, SSLv3, and now TLSv1. This also applies to certain ciphers such as MD5, RC4, and eventually the RSA key exchange.

The F5 BIG-IP platforms make it extremely easy to control and enforce these protocols and ciphers but at the same time, you may not simply want to “break” some users. If your income depends on Internet commerce, the last thing you may want to do is block customers from buying from you because they have a slightly out-of-date browser. Eventually, most of these older browsers will go away or at least enough of them that you can justify disabling lower encryption protocols and ciphers. But, how do you know?

F5 iRules have access to everything in the OSI layers from 3 to 7 that includes some pretty rich information about SSL and TLS. Let's take a look at an iRule that will catalog the SSL/TLS versions that clients are using.

Note: A word on “mixed content” – All too often, developers or development platforms provide access to both HTTP and HTTPS content within the same application. This is referred to as “mixed content” and can have some significant security implications. For example, an application may redirect a user to an HTTPS URL to authenticate, but then switch back to HTTP after authentication to access normal application content. At the very least, the HTTPS authentication is very likely going to set up a “session” with the user, which will also very likely to be maintained by an HTTP cookie in the browser. If the application session is now communicating via HTTP, all of that user session information is in the clear. Long story short, don't allow mixed content in your applications. If there is *anything* in your application worth protecting with encryption, it is a best practice to encrypt *everything*.

Requirements

- BIG-IP LTM, web server, client browser, and a SSL server certificate. If you don't have certificates to test with, you can use the CA certificate, server certificate, and private key provided in the Client Certificate Inspection lab from Additional labs section.

The iRule

```
1 when CLIENTSSL_HANDSHAKE {  
2     ISTATS::incr "ltm.virtual [virtual name] c [SSL::cipher version]" 1  
3 }
```

Analysis

- iStats are user-created custom statistics, accessible from both the data plane (iRules) and the control plane (tmsh, on-box scripts, etc.). What we're doing here is simply cataloging the SSL/TLS version from each client SSL handshake and storing these in an iStats table. That data can then be accessed from pretty much anywhere. For example, see David Holmes' excellent guide on dumping this information to a Google pie chart:

<https://devcentral.f5.com/s/articles/categorize-ssl-traffic-by-version-display-as-graph>

Testing

- Apply this iRule to an SSL VIP and test across several browsers and platforms. If at all possible, stage this iRule at someplace that it can be accessed by a larger audience.
- At any point you can access the data collected in the iStats by simply typing the following at the BIG-IP command line:

```
istats dump
```

Hint: For the purpose of this lab we can use

```
openssl s_client -connect www.f5demolabs.com:443 <cipher>
```

where cipher options could include {-tls1, -tls1_1, -tls1_2} to simulate different connections.

Here's an example output:

```
[ ltm.virtual=/Common/stdsslvip ][TLSv1.1] = 54 (2015-09-01 13:52:20)
[ ltm.virtual=/Common/stdsslvip ][TLSv1.2] = 282 (2015-09-01 13:52:20)
[ ltm.virtual=/Common/stdsslvip ][TLSv1] = 32 (2015-09-01 13:52:20)
```

Bonus version

Now that you have a better idea of what protocols are being used on your site, you may now want to ease your customers into a better security posture than simply denying access with no warning. To do this, we'll first test the SSL/TLS version. If it's below our security threshold, we'll redirect the user to another page or site to inform them that they need to upgrade their browser.

```
1 when HTTP_REQUEST {
2     if { (( [SSL::cipher version] equals "TLSv1" ) or ( [SSL::cipher version] equals
3     ↪ "SSLv3" )) and not ( [HTTP::uri] equals "/insecure.html" ) } {
4         set redirect "https://www.f5demolabs.com/insecure.html"
5         HTTP::respond 302 Location "${redirect}"
6     }
7 }
```

You're still allowing SSLv3 and TLSv1 at this point, which is definitely bad, but you're not allowing access to the application for anything less than TLSv1.1.

Hint:

1. Change client ssl cipher from DEFAULT to DEFAULT:SSLv3
2. Use `openssl s_client -connect www.f5demolabs.com:443 -tls1` to connect
3. Move bonus version of irule, `sec_irules_tls_version_control_2`, to the selected list of iRules on the generic-app HTTPS virtual server

Note: Lab Notes:

- Use the Chrome browser to manage the BIG-IP.
- **Use the Firefox browser to perform access testing.**
 - Modify Firefox's TLS version by navigating to [about:config](#) and modifying the "security.tls.version.max" value.
 - 1 = TLSv1.0
 - 2 = TLSv1.1
 - 3 = TLSv1.2
- The test site URL is <https://www.f5demolabs.com>. A hosts file entry is already applied to the lab desktop.

- **Use a command line client to also test access:**
 - `curl -vk https://www.f5demolabs.com --[tlsv1.0|tlsv1.1|tlsv1.2]`
 - `openssl s_client -connect www.f5demolabs.com:443 --[tls1|tls1_1|tls1_2]`
 - **Three TLS version control iRules are provided:**
 - Basic istats capture
 - Redirect to insecure page if TLSv1 or SSLv3
 - Provide David Holmes' iRules and access to the /sslversions URL.
-

3.1.2 Lab 2 - HTTP Throttling

Scenario:

Your company has setup a new web application and did not have the time to develop a WAF policy. Due to your company's profile, several bad actors have threatened to attack using SlowLoris and SlowPost attacks to create a denial of service. Although Advanced WAF or ASM can handle this very easily and would be the best tool to use, we don't have the resources available to test implementing the policy. In this lab, we are going to use an iRule that throttles the number of requests coming into the application.

Restrictions:

The following restrictions complicate the request to implement the throttling of HTTP requests:

- You need to understand the number of the requests that would be coming from a single IP address.
- Some requests may be coming from companies using a single proxy IP address to make the request. Throttling those requests to only 10 per 10 seconds could impact the ability of a partner company to access the site.

Requirements:

To meet the business's objectives the iRule must meet the following requirements:

- The rule must keep a single client from making too many HTTP requests to a single VIP thus stopping a SlowLoris or SlowPost attack.
- The rule should be able to adjust to the multiple customers coming through a proxy IP address.

The iRule

```
1  when RULE_INIT {  
2      # This defines the maximum requests to be served within the timing interval_  
    ↳ defined by the static::timeout variable below.  
3      set static::maxReqs 4;  
4  
5      # Timer Interval in seconds within which only static::maxReqs Requests are allowed.  
6      # (i.e: 10 req per 2 sec == 5 req per sec)  
7      # If this timer expires, it means that the limit was not reached for this interval_  
    ↳ and the request
```

(continues on next page)

(continued from previous page)

```

8      # counting starts over.
9      # Making this timeout large increases memory usage. Making it too small
↳negatively affects performance.
10     set static::timeout 30;
11 }
12
13 when HTTP_REQUEST {
14     # The iRule allows throttling for only sepecific Methods. You list the Methods_to_
↳throttle
15     # in a datagroup.
16     # If you need to throttle by URI use an statement like this:
17     #                                     if { [class match [HTTP::uri] equals URIs_to_
↳throttle] }
18     # Note: a URI is everything after the hostname: e.g. /path1/login.aspx?name=user1
19     #
20
21     if { [class match [HTTP::method] equals Methods_to_throttle] } {
22
23         # The following expects the IP addresses in multiple X-forwarded-for headers.
↳It picks the first one.
24         if { [HTTP::header exists X-forwarded-for] } {
25             set client_IP_addr [getfield [lindex [HTTP::header values X-Forwarded-For]
↳0] ", " 1]
26         } else {
27             set client_IP_addr [IP::client_addr]
28         }
29         # The matching below expects a datagroup named: Throttling_Whitelist_IPs
30         # The Condition of the if statement is true if the IP address is NOT in the
↳whitelist.
31         if { not ([class match $client_IP_addr equals Throttling_Whitelist_IPs ] ) } {
32             set getcount [table lookup -notouch $client_IP_addr]
33             if { $getcount equals "" } {
34                 table set $client_IP_addr "1" $static::timeout $static::timeout
35                 # record of this session does not exist, starting new record, request is
↳allowed.
36             } else {
37                 if { $getcount < $static::maxReqs } {
38                     log local0. "Request Count for $client_IP_addr is $getcount"
39                     table incr -notouch $client_IP_addr
40                     # record of this session exists but request is allowed.
41                 } else {
42                     HTTP::respond 403 content {
43                         <html>
44                         <head><title>HTTP Request denied</title></head>
45                         <body>Your HTTP requests are being throttled.</body>
46                         </html>
47                     }
48                 }
49             }
50         }
51     }
52 }

```

Analysis

- Notice that RULE_INIT sets up a set of variables that respectively define the maximum rate of requests and a timeout value. Alter these values according to your local preferences.
- The iRule creates an internal table for each client source address, and an entry for each request is added to this table.
- As each new request arrives, the iRule counts the number of entries in the respective table. If the count doesn't exceed the maxRate threshold, a new entry is created and the request is allowed through.
- If the request exceeds the maxRate threshold, the iRule returns an HTTP error response to the client.

Testing

A very simple way to test this iRule implementation is with a cURL script from the Terminal command line. Here's a Bash representation of that script. We have already put the script on the Ubuntu client and instructions follow the sample code below.

```
1 #!/bin/bash
2 while [ 1 ]
3 do
4     curl http://www.f5demolabs.com --write-out "%{http_code}\n" --silent -o /dev/null
5 done
```

- In Terminal, cd to scripts directory and run `bash http_throttling`.
- Notice that you are getting 200 responses from each request. We will now add the iRule to the VIP.
- Login to BIG-IP from Chrome browser.
- Go to Local->Virtual Servers and select the http virtual server.
- Select the resources tab and select Manage for iRules.
- Select the Lab2_1 irule and move it into Enabled.
- Select Finished.
- To view logging information on the F5 BIG-IP follow these instruction:
- **Modify the iRule on the F5 to uncomment the line that states:** `log local0. "Request Count for $client_IP_addr is $getcount"`
- Click on Update on the iRule.
- Open ssh, connect to BIG-IP, and enter the bash shell.
- Run a tail of the BIG-IP LTM log from command line as follows:

```
tail -f /var/log/ltm
```

The script will make repeated HTTP GET requests. When it exceeds the threshold the iRule will generate a 403 error response and prevent access to the web server until the **timeout** static variable time is reached.

- Use the CTRL-C keyboard combination to stop the script.

Bonus version

The above iRule presents an extremely simple approach to HTTP request throttling and is based solely on client source address. The following bonus example extends that functionality to allow for throttling of specific URLs.

```

1 when RULE_INIT {
2     # The max requests served within the timing interval per the static::timeout_
    ↪variable
3     set static::maxReqs 4
4     # Timer Interval in seconds within which only static::maxReqs Requests are_
    ↪allowed.
5     # (i.e: 10 req per 2 sec == 5 req per sec)
6     # If this timer expires, it means that the limit was not reached for this_
    ↪interval and
7     # the request counting starts over. Making this timeout large increases memory_
    ↪usage.
8     # Making it too small negatively affects performance.
9     set static::timeout 2
10 }
11 when HTTP_REQUEST {
12     # Allows throttling for only specific URIs. List the URIs_to_throttle in a data_
    ↪group.
13     # Note: a URI is everything after the hostname: e.g. /path1/login.aspx?name=user1
14     if { [class match [HTTP::uri] equals URIs_to_throttle] } {
15         # The following expects the IP addresses in multiple X-forwarded-for headers.
16         # It picks the first one. If XFF isn't defined it can grab the true source IP.
17         if { [HTTP::header exists X-forwarded-for] } {
18             set cIP_addr [getfield [lindex [HTTP::header values X-Forwarded-For] 0]
    ↪", " 1]
19         } else {
20             set cIP_addr [IP::client_addr]
21         }
22         set getcount [table lookup -notouch $cIP_addr]
23         if { $getcount equals "" } {
24             table set $cIP_addr "1" $static::timeout $static::timeout
25             # Record of this session does not exist, starting new record
26             # Request is allowed.
27         } else {
28             if { $getcount < $static::maxReqs } {
29                 log local0. "Request Count for $cIP_addr is $getcount"
30                 table incr -notouch $cIP_addr
31                 # record of this session exists but request is allowed.
32             } else {
33                 HTTP::respond 403 content {
34                     <html>
35                     <head><title>HTTP Request denied</title></head>
36                     <body>Your HTTP requests are being throttled.</body>
37                     </html>
38                 }
39             }
40         }
41     }
42 }

```

By running the `http_throttling_bonus` script, you are checking HTTP requests limits against the URL paths in the `URIs_to_throttle` datagroup. Here's a Bash representation of that script.

```

1 #!/bin/bash
2 while [ 1 ]
3 do
4     curl http://www.f5demolabs.com/admin/ --write-out "%{http_code}\n" --silent -o /
    ↪dev/null
5 done

```

3.1.3 Lab 3 - Securing Cookies

Scenario

The security team has done an application scan and found that the HTTP cookies are being issued unsecured. They have asked the application team to verify that all the cookies get the Security and httpOnly flags set at the application tier. But, the app team is in the middle of a new deployment and can't reallocate resources to rewrite the cookie code. So, they have asked the F5 team to set the flags on the cookies.

Restraints

The following restraints prevent from implementing this solution:

- The F5 administrators need to know the name of the HTTP Cookie or Cookies that are being used.

Requirements

To meet the business's objectives the iRule must meet the following requirements:

- The iRule must take a HTTP Cookie being sent from the application server and set the Secure flag and the HTTPOnly flag.
- The security team also requires that the HTTP Cookie not be sent to Javascript Agents.

Lab Requirements:

- BIG-IP LTM, web server, client browser, and SSL server certificate. If you don't have certificates to test with, you can use the CA certificate, server certificate and the private key provided in the Client Certificate Inspection lab from Additional Labs section.

The iRule

```
1 when HTTP_RESPONSE {  
2     set ckname "mycookie"  
3     if { [HTTP::cookie exists $ckname] } {  
4         HTTP::cookie secure $ckname enable  
5         HTTP::cookie httponly $ckname enable  
6     }  
7 }
```

Analysis

HTTP cookie misuse represents one of the greatest vulnerability vectors known to Internet communications. It's number 2 on the Open Web Application Security Project's (OWASP) Top Ten list as "broken authentication" (https://www.owasp.org/index.php/OWASP_Top_Ten_Cheat_Sheet), and also touches other top ten list items, including XSS, security misconfiguration, sensitive data exposure, and cross site request forgery.

So why do we use cookies if they're so easy to get wrong? Well, as it turns out, HTTP as a "stateless" protocol doesn't provide its own session management mechanism. So cookies are basically the best and potentially the only way to maintain information about a user session across multiple HTTP requests and responses. Too often, however, applications employ mixed content (HTTP and HTTPS in the same application), or worse, store too much information in that cookie.

If an HTTP cookie is being used to maintain an authentication application session, it's always a best practice to encrypt every part of that application. And it is never a best practice to store anything more than an identifier (some seemingly random and unpredictable blob of characters) in a cookie.

There are two built-in mechanisms defined in RFC 6265 that can help. But first, let's first understand what an HTTP cookie is. An HTTP cookie is essentially an HTTP header that is sent from the server to the client, and then sent back to the server in each and every subsequent request. To send a cookie, the server will format the header as given below:

```
Set-Cookie: foo=bar; path=/; domain=domain.com; expires=Sat, 02 May 2009
23:38:35 GMT
```

where

- `foo=bar` represents the mandatory name=value content
- `path=/` represents the mandatory path, or scope of the cookie, such that the browser will only return this cookie if the request is in the designated path (in this case the entire website)
- `domain=` represents another, albeit optional, scoping mechanism that tells the browser that this cookie can be sent with any request in the same domain or subdomain. Unlike the path attribute which reduces visibility, the domain attribute increases visibility by making the cookie potentially available across multiple subdomains. Be careful with the domain attribute though. This is an often-used way to build single sign-on, where users authenticate at one domain, but are allowed access to other subdomains by virtue of the cookie being available to all. At the very least, if you don't own every subdomain that this cookie could possibly be sent to, then someone else may get your user's session cookies.
- `expires=` represents an optional attribute that indicates how long the client should retain this cookie. If the expires attribute is not in the Set-Cookie header, then the cookie is considered "session-based" and will generally live for as long as the browser session (ie. closing the browser deletes the cookie). If the expires attribute exists, the cookie is typically stored on disk or other long-term memory that will persist after the browser is closed and a new one is opened. This is, generally speaking, not a best practice from a security perspective. If a cookie is used to maintain some sort of client state information, and the computer itself is compromised, then that state can also be compromised. It is typically far better to not include an expires attribute, thus deleting the cookie when the browser closes. You can, however, delete an existing in-memory session-based cookie by sending a new cookie with the same attributes but with an expires attribute in the past.

Once the cookie has been received, and depending on scope, the client will transmit that information back to the server in every request.

```
Cookie: foo=bar
```

Notice that a request cookie doesn't have path, domain or expires information. This information is meant for the client only, so doesn't need to be relayed back to the server.

Aside from adjusting path and domain attributes accordingly to limit or expand visibility, there are two additional scoping attributes that play a crucial part in cookie security.

- `secure` represents a single (no value) flag that tells the browser client to only transmit this cookie back in a secure (ie. encrypted HTTPS) request. This attribute is critical against attacks like cross site scripting (XSS) or request forgery (CSRF) where a browser may otherwise be tricked into sending session cookies to an unencrypted host. If you're encrypting the entire application, the secure cookie flag is an excellent option.
- `httpOnly` represents a single (no value) flag that tells the browser client to only transmit this cookie back to non-scripted user agents. In other words, if a JavaScript agent makes a request inside the browser, the cookie will not be sent with this request. Many XSS and CSRF exploits rely on the ability to grab session cookies with rogue browser scripting (ex. JavaScript, vbscript, etc.). There

are of course instances where a JavaScript agent needs to send the cookie, like in side-channel Ajax requests, but if not, this flag is highly useful.

So putting these attributes together might look something like this:

```
Set-Cookie: foo=bar; path=/; secure; httponly
```

we have removed the **expires** attribute because file-based cookies are almost always a bad idea. And we removed the **domain** attribute because there are better and more secure ways to do single sign-on. So in this example, we are setting a cookie called “foo” with a value of “bar”, that is scoped to all paths within this host (path=/), and will only be transmitted over HTTPS and only to non-script agents. As I mentioned a few times, there’s simply no substitute for a good security product (ie. web application firewall, malware scanner, etc.) and no excuse not to write secure code, but if you find yourself in a situation where secure cookie coding isn’t happening in the application, then here’s a quick and easy way to enable it with F5 iRules.

- In this very simple iRule, we’re triggering an event on the HTTP response being sent from the application server, looking for the cookie `mycookie`. If it exists, enables the `secure` and `httpOnly` flags. This command effectively includes the `secure` and `httpOnly` flags in the `Set-Cookie` header being sent to the client.

Testing

In the BIG-IP,

- Access HTTPS URL without iRule to see current cookie status.

```
curl -vk https://www.f5demolabs.com
```

- Attach the iRule to the HTTPS VIP
- Access the HTTPS URL to see the change in the cookie information.

```
curl -vk https://www.f5demolabs.com
```

A word on cookie security – the `secure` and `httpOnly` flags are exceedingly important for the proper and secure use of HTTP cookies, but alone they are not perfect. There are still ways to compromise HTTP cookies, even with these flags enabled, so do take additional precautions which should definitely include a solid web application firewall product and malware scanning and intrusion detection products.

3.1.4 Lab 4 - Client Certificate Inspection

Scenario:

Your company uses smart cards for two-factor authentication. Users access different resources from a single url and need to be given access to those resources based on the properties of a client certificate. Users have physical smart cards and software-based client certificates and authentication decisions will need to be made based on certificate attributes.

Requirements:

- BIG-IP LTM, web server, client browser, SSL server and client certificates

To meet the business’s objectives while still maintaining a strong security policy, an iRule solution must meet the following requirements:

- inspect certificate attribute to give access to correct resource

Server private key

```

-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAYzJyCMIp1vTECCfTcCu08Fx8xaEy+ipAV/+Skzl09u5xY4jw
DAeaNKP/6d6h3x2IppVgw9lICiMsdpQIPg04RF9K+sA2+DF1vrRmSqDpvsWzJehh
15wfqaUBJxqHsZlVnH2Joa0ukJaaQR0k5ieqShjRchZ/tfL909hC7GV0cxBTjH06
FjuaFHOMj2zyh4IfLPTKDNtQOr8QH3+6dDJss1Q0ncZRmDEkc9u+M/wXxGi97jk
Yc3YIQ1+Xvln/Qq7cXawuK4oKDHjxLr4ldw40nKOIW+N01vBUurZIAaJwcf+dwPs
yXKCp8JKjLfPl64b1bly8d7uHuuc+rNRICB/ou/wIDAQABAoIBADEduetSDIC292
/yq2p18tXeFxY646MQ5aA853jtVdqGNv3497YiIdPl/HjCtSLTLB387NJWgepuD
YqUhk4gKyT+tmNdDHdqYq4IkaPj4pZpQRA/avKRRkvkNdbyslhmpaxtDZ/+VP0GL
JvPDTqGkGik5cHdUBsoEwnQ4W/ZRaP+hrvFDguYlwZAE+iN35AXWdvIU7IzldZN
mcsmPEyqQoHlWvmS15i9IqSkUabbvt/fWCZQTmAQHdc4J+gyYekcLf+ubVgEzB4C
Yh/cibO+MMLH0w6aG2l2dnAwPephhsRYvKdC4GmxHanMNdnXuI02HpY8ySL2Ue
cPmlnSECgYEA5gixIImQTN0Tbq0VPOYFs09/GD1lk57rQmXQ4FTTd0t++tSyV/oX
ugDXeHa10/K3iufaJNfktj7bUAlux740nqgOqaq/NENiLvF3RMWfVn0UJ008loHx
4ZcpuWfSt/6TRgrHg+V+H0OMCEwUcebG6123Wd43b3JipHttLWFxQpkCgYEA4iI+
4bIN61ptzZVdmWc7hvIdDvFnyqotOj1wL5RAucV6W0T6SYcQAOjB6UXYeDf0isHQv
i5c+oEqVzHly53+Bx6zRT9zpEhJfDoWkP2FVCvCgYB+TMFs+4334KbvOosS7ZBN
r1U66uLtlXDYS0zRbuGYe1QhxkyRb1g9oR6tGvcDax3xX3FvjyFwVlZN8I/pja54
eg9q6rwGpwSuf5ebo90c9BnuUzgFbXluXj/jc3TH3zffWiXHbma8JasqFxoWoj4P
lqoH5rGLOEOeycHdC8ZS6QKBgQCXr7MQf/h4TANlpfHugiJh4oVah9eQcLu0IKhV
8gHFSFbQazGS0wSZ6vnotzMMWK9jF7zjXQPET+Ob8tb707KfogdMxyBSLa8lZmKE
NJukCx53uVxyRXpCVf5+xe5sVI4iAP2jPxdPjNle2aPqbPsm00+BfYdj/APxfcJv
Xe7dJwKBgQDgeLXskt1ymndPFdY9XphX/DksZThxy3gFZPicns4mTJ716VRpoAd3
tJUawHyG97Gdo6XSfVn4Ge7FhMgskqZcXHGr6dtmxdbdheY4uyZp+Kep5gmVmyng
2Kz+pbG3E5IAf/AlmxCGEe7EDTZUpqCuTeIRKslBBPG6mir2vLFNTA==
-----END RSA PRIVATE KEY-----

```

Client certificate (user@f5test.local)

-----BEGIN CERTIFICATE-----
MIIElDCCA3ygAwIBAgIBBDANBgkqhkiG9w0BAQsFAADBAQswCQYDVQQGEwJVUzEV
MBMGA1UEChMMZjV0ZXN0LmxxvY2FsmR4wHAYDVQQLEwVVDXJ0aWZpY2F0ZSBBdXRo
b3JpdHkxFTATBgNVBAMTDG9yZC5sb2NhbDAAeFw0xNzA3MTIwODA2MjdaFw0y
MDA1MDEwODA2MjdaMH0xZCzAjbG9NVBAYTA1VMTMRUwEwYDVQQKEwxxNHRlc3QubG9y
YWxxGTAXBgNVBAsTEFVzZXIgcQ2VydGlmawNhdGUxGjAYBgNVBAMTEXVzZXIuZjV0
ZXN0LmxxvY2FsmSAwHgYJKoZIhvcNAQkBFHf1c2VyQG9yZC5sb2NhbDCCASIW
DQYJKoZIhvcNAQEBBQADgGEPADCCAQoCggEBAJmy1XU/hJCbvIT5Dsb4s59yep4j
zR0OScuFiokeAaZhqdKxW69LN61/M4a7ohRQHj1YEHTRLMQuzSolkeoVqm52KKEY
Ws9lkpq3S00bn+JcN1ZcvYbW7FDVBPN4Z+Rkd5VsSwh84zue7B+is5L0XhKUPb4B
WXD0nHms/TUH55nxiFQnyqgr69qMK+pflQhCkH8g84zpuje9Qsk5iV1xeRdEq
bOME/VYr11zvYrBrhCzcftJp+PtbY571/CSawg0P/GeVrPmJoo9e9H0/vcoG9HmtDX
s8mt dg6mUKCYBVhED2362bj1KiDZ6t7IoCafBXM94oPlDAG8tAucGbH5gJcCAWEA
AaOCAT8wggE7MDsGCWCGSAGG+EIBDQQuFixPcGvUu1NMIGdlbmVyYXRlZCBzbWYy
dGNhcmQgdXNlcibjZXJ0aWZpY2F0ZTAdBgNVHQ4EFgQUUaMwNzNNL4dhB/AzQBAj
AkindiUwHwYDVR0jBBBgwFoAUp1RR2qzOWUoZT921koMLiOwHOFIwDgYDVR0PAAQh/
BAQDAgBAMCKGA1UdJQQiMCAGCCSGAQUFBwMCBgorBgEEAYI3FAICBggrBgEFBQCD
BDA/BgNVHREEOADA2gRF1c2VyQG9yZC5sb2NhbKAhBgorBgEEAYI3FAIDoBMM
EXVzZXJAZjV0ZXN0LmxxvY2FsmMCMGA1UdIAQCMBoCwYJYIZIAWUCAQsJMAsgCWCG
SAFlAgELEzAbBgNVHQkEFEDASMBAGCCSGAQUFBwKEMQQTA1VTMA0GCSqGSIb3DQEB
CwUAA4IBAQAfKi84V5UX1BiY/XG4gkCwP63JmWwBl9DgFjdG9eXPlfFzIGw/mlEj
uULGdHLVqOJlNSEuNdbbHic3anxN7TF1ZTm+92xX6/mQhumabvXGqq5s9FjvzmQl
6LSEH8U1oGBr1ByV44U3ifJXUsJyrUtfcZN0BifskcAa05C2pJtKdMxHnGln/s2C
lu+cf2AqAoogZcZ2PsfGJtbnV5XkcxZ+ASWAp2R41tNWqIbaKEFGs0b91Ja53qmQc
25iGpuAqm/ierJcVdFdfLbENWK6vWKJ37MnbnwVG6Rot08uYnyBvgK2JzoGMVhjysQ
peMa0CNvHvZB/PtbPaNtCKqHJhz6zOI3

(continues on next page)

(continued from previous page)

-----END CERTIFICATE-----

Client private key

```

-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEAAmbLVdT+EkJu8hPkOxvizn3J6niPNHQ5Jy4WLSR4BpmGp0rFb
r0s3rX8zhruiffAePVgQdNEwtC7NKjWR6hWqbnYooTJaz2WSmrdLTScH6MI3Vly9
htbsUNUE+d7hn5GR3lWxLCFFbATsH6KzkvReEpQ9vgFZd06ceZL9NQfkzmfGIVCf
KDGvr2owr6l8uocKTWfyDz jOm6MT1BKSzmJXXF5F0Sps4wT9ViuWXO9isFGELNx+
0mn4+1t jnuL8JJrCDQ/8Z5G8+Ymh70c7+9ygb0ea0Nezya12DqZQoJgFWEQPbfrZ
uPUqINnq3sigJp8FcZ3ig+UMAbY0C5wZsfmAlwIDAQABaoIBAGlmF7d1vWS1R5ww
Zw/PUO5QxQFZL7lZK0vmQmp7rcn5Q0n20hbdj+rsRdtpJHalknciwwY41htZ1NvT
LKL1BL4HTU1t jJSY5PYwJ/VahLP7K5OPuXCURi4QRn9LdpHEc7FyNjM7F4KtXbU
TizCYxh+i/CWYFHOMMNOJ1GMfj2EIFsUh7i3D9W3A/HKaEn7RWfFWBpF8Owff7B1
k/qyhjIjv8ux3f7K9izvUiVWH/T9FMPXhb89ieT6Up5Qgrqlejq6JnHkUhZvrA3N
AFWUI2SxMGMy+jS7HCwj5fM3it/FkkG2uf2v3CXx5CP//lmBWid3nCCr9FtB0UgK
BwrQ7nECgYEAYxViZTBuPdH0q/GVHcknlIXv10B4Ah5pNdgfl345fkOLjtXe5HoR
MMuLHGACD0/mVn4r1/obU/359ANOOOrdGT/66AAD24VhNrtvoeMzDRXJ+Y9QNdBwo
tNHntZzp4msolFkSiHUObHG5jXcxryDig2Y54ZLeRJCLCFqBXRlHfTsCgYEAwb8+
LJYC/SIsbSq607cUhiOgcyTkKmKueFUH7ic8JzYXNOTu/mAJuVWb9X1rzCRLc6wj
MXj9lKZoyVHaoY7aAtD0y75MuOH0FEZG7btE6iba48ZTiAKc3hZXFOszYdPwWUjI
fRQK3g0aRPFrgXhkTFG/aXc6rWFbXZCd9x1YBFUCgYATMmNJs2lIWldrJXv2A9TE
+mAqiQKPGlbtSym5VUo0AEiJ6PeX214SobrlpLgtJtlcIbMXO6Inr2NYSJO1go5M
c4S7iVvM817iqtjvylNPfKSRzI6XosOhKUFit6k84Ize7P/yCjj4WAr2i+NIWuo
BhrEkvCFxLKE9qEyBmxi jwKBgFzLVgtOVgqHGyQQq5C8PKQAawsqchf8jsjlhELl
HwtX/PiImCrXylgwuwGe7FPKRz8kFw++gl+G1pFIpPp3owJfyglyqhl2+8/IznNo
KifXD3bM/folvo8hyQknqNBMLV6x7idCt982CxVshcfjMLwDKjLoTwMYvkbhC0yU
DkKtAoGABYODvNIuhUQGk8sKc jByZIpMBeeaFBqPSn0dClUvZnTDTA5sKpblnzQ7
xj1IK+ZEQQewJ4Tift4TctskkUYDoGz21vsq1BJGXzq/mQP jbyYmeE43jxik7hZ1E
M33AhM3mAkOT6tnFoD78DNZn8H1HKuaqt1ljYCCCiH7tkA59Cuw=
-----END RSA PRIVATE KEY-----

```

Baseline Testing:

Prior to defining a solution, validate that users do not have the correct access.

- **From the client work station, ensure you have access:** `curl -kv https://www.f5test.local`.
- You should have full access to the url.

The iRule

F5 iRules have complete access to the x509 properties of a client certificate during that authentication and can look at the attribute of the certificate to make decisions.

```

when RULE_INIT {
    set static::debug 1
}
when CLIENTSSL_CLIENTCERT {
    # Example subject:
    # C=US, O=f5test.local, OU=User Certificate, CN=user/emailAddress=user@f5test.
    ↪ local
    set subject_dn [X509::subject [SSL::cert 0]]
    if { $subject_dn != "" } {

```

(continues on next page)

(continued from previous page)

```

        if { $static::debug } { log "Client Certificate received: $subject_dn" }
    }
}
when HTTP_REQUEST {
    if { [HTTP::uri] starts_with "/" } {
        if { $subject_dn contains "CN=user.f5test.local" } {
            HTTP::uri /headers.php
        } else {
            reject
        }
    }
}
}

```

Analysis

- The above iRule inspects the x509 subject value in the client's certificate and makes an access decision based on that value. In this very simple example, a specific set of users may access different corporate resources hosted behind the same VIP.

Testing

- In the Client Authentication section of the client SSL profile `Lab4_Clientssl`, set Client Certificate to `Require`, and assign `f5test.local` to the Trusted Certificate Authorities option.
- Test accessing the URL `https://www.f5test.local` from the client. First do not include the client certificate:

```
curl -vk https://www.f5test.local
```

You should receive a failed handshake error. Try again, but include the certificate: `curl -vk -cert /etc/ssl/certs/f5test.pem https://www.f5test.local`

You should now be able to pass through to the application.

- In the Resources section of the `f5test_local` virtual, add the `Lab4` irule.
- **Watch the log file on the BIG-IP:** `tail -f /var/log/itm`
- **Access the URL again from the client:** `curl -vk -cert /etc/ssl/certs/f5test.pem https://www.f5test.local`

You should now get a different response page. Notice the Client Certificate log message on the BIG-IP.

3.2 Additional Labs

3.2.1 Additional Lab 1 - ASM Hooks

Scenario

When applications are moved to your production environment, they are secured with F5 Application Security Manager (ASM) with a well-defined security policy for each application. The policies deployed have been tested, tuned, and are currently part of an automated build process for deployment. Amongst other protections, an ASM policy filters all inbound requests that attempts to inject a null byte character from ever reaching the protected web servers.

In general, this protection is well advised and does not cause issues for most of your customers. However, it has recently come to the attention of your business representatives that the policy is blocking traffic from one of your most important business partners. This partner uses automated scripts to scrape your current inventory. A bug in the partners scraping code is adding extra characters to each request which is violating the ASM's policy. The business team would like your security team to disable this protection, so that it no longer causes issue for an important partner.

Restraints

The following restraints complicate the request from the business team:

- In ASM, the change to relax protection for null byte injection attacks is a global policy setting, and it is enforced across all requests for all users. Disabling this protection weakens your security policy at a more global level than you are comfortable.
- This baseline policy is deployed across a number of applications and is ingested by the DevOps team when deploying new applications. The impact of a policy modification may have broader impact than intended.
- Your security and compliance teams run routine scans of your applications looking for vulnerabilities that can be exploited. By modifying the policy on a global basis, you are certain to get notifications and audit findings for a number of applications.

Requirements

To meet the business's objectives while still maintaining a strong security policy, an iRule solution must meet the following requirements:

- Any modifications to the application security policy must only affect the relevant business partner, and only for the in-scope application:
 - Partner IP = 10.1.10.51
 - URL = [http://hackazon.f5demo.com/product/view?id={\[\]0-100{\[\]}%00](http://hackazon.f5demo.com/product/view?id={[]0-100{[]}%00)
- If a request contains violations other than the violation identified above, the request should be blocked.
- Direct modifications to the ASM security policy are not allowed
- Prior to releasing the in-scope requests to the application server, the request must be sanitized by removing the "%00" null byte string terminating partners request.

Baseline Testing

Prior to defining a solution, validate the issue by testing the application to validate ASM's behavior:

- RDP to the lab jump station
- From the jump station, open Chrome and browse to <http://hackazon.f5demo.com/product/view?id=72%00>
- Verify that you receive the ASM block page
- Test a few more products (e.g. id 73, 7, 45)
- Open Terminal application
- Curl <http://hackazon.f5demo.com/product/view?id=72%00>
- Verify the content returned is a block page from ASM

- Open another Chrome window, and launch BIG-IP Configuration Utility (<https://10.1.1.245>)
- From BIG-IP, verify violation in event logs:
- Click **Security -> Application Security -> Event Logs -> Application -> Requests**
- Examine requests for [HTTP] /product/view
- Check icon on request, then click All Details in the request detail to verify the Request Status is blocked

The iRule

```

1  when ASM_REQUEST_DONE {
2      set asml_debug_verb 1
3      set asml_debug 1
4
5      if { [ASM::status] equals "blocked" } {
6
7          if { $asml_debug_verb } {
8              log local0. "Violation count: [ASM::violation count] "
9              log local0. "Violation names: [ASM::violation names] "
10             log local0. "Violation attack types: [ASM::violation attack_types] "
11             log local0. "Violation details: [ASM::violation details] "
12         }
13
14         if { [ASM::violation count] <= 1 } {
15             # Allow only if this request only violates a specific element of the policy
16             if { [lindex [ASM::violation names] 0] equals "VIOLATION_HTTP_SANITY_CHECK_
17             ↪FAILED" } {
18                 if { $asml_debug } {
19                     log local0. "ASM_OVERRIDE: HTTP Request Blocked by ASM with SANITY_
20             ↪CHECK VIOLATION, URI = [HTTP::uri] "
21                 }
22                 if { [HTTP::uri] starts_with "/product/view?id" && [HTTP::uri] ends_with "
23             ↪%00" } {
24                     if { $asml_debug } {
25                         log local0. "ASM_OVERRIDE: URI Request pattern matches override_
26             ↪request"
27                     }
28                     if { [ASM::client_ip] equals "10.1.10.51" } {
29                         if { $asml_debug } {
30                             log local0. "ASM_OVERRIDE: Partner IP: [ASM::client_ip] matches_
31             ↪override request"
32                         }
33                         #we have a request that matches the OVERRIDE request, override and_
34             ↪modify
35
36                         set new_uri [string trimright [HTTP::uri] "%00"]
37                         HTTP::uri $new_uri
38                         ASM::unblock
39                         if { $asml_debug } {
40                             log local0. "ASM_OVERRIDE: Modified request URI, new uri =_
41             ↪[HTTP::uri]"
42                             log local0. "ASM_OVERRIDE: Unblocking request and releasing_
43             ↪to server"
44                         }
45                     }
46                 }
47             }
48         }
49     }
50 }

```

(continues on next page)

(continued from previous page)

```

38         }
39     }
40 }
41 else {
42     if { $asm1_debug } {
43         log local0. "ASM:OVERRIDE: Request contains multiple violations, will not_
→override sec policy"
44     }
45 }
46 }
47 }

```

Analysis

ASM Event/Command Details:

- `ASM_REQUEST_DONE` event is triggered after ASM has finished processing the request and found all violations of the ASM policy.
- `[ASM::violations]` command will return the list of violations found in the request or response with details on each violation
- `ASM::unblock` command overrides the blocking action for a request that had blocking violation

Rule Details

The rule does the following:

- Inspects the blocking status of the request. If the request is blocked, the rule validates that request contains only a single violation. This violation is the one whose approval has been given to override (`VIOLATION_HTTP_SANITY_CHECK_FAILED`) and the request originates from the expected business partner.
- If the request matches the above conditions, the irule will do the following:
 - Strip the expected violation from the request
 - Unblock the request

Testing

- From BIG-IP Configuration Utility, open **Local Traffic -> Virtual Servers** and select `vs_hackazon_http_virtual`. Click the Resources tab. In the iRules section, click Manage. Move `sec_irules_asm_hook_1` from Available section to the Enabled section and click the Finished button.
- From the Jump Station, open the Terminal application and SSH to the BIG-IP: `ssh root@10.1.1.245`.

```
[root@bigipo01:Active:Standalone] config # tail -f /var/log/ltn
```

- Re-open the Chrome window used in the Baseline Testing section, and again browse to <http://hackazon.f5demo.com/product/view?id=72%00>
- Earlier, this request was receiving an ASM block page. Now, you should be getting access to the page.

- From the SSH session, review the log messages associated with the above request. Details on the request, and the override decision should be present in the logs.
- From BIG-IP, verify violation in event logs:
- Click **Security -> Event Logs -> Application -> Requests**
- Examine requests for [HTTP] /product/view
- Check icon on request, then click All Details in the request detail to verify the Request Status is unblocked

Test additional conditions:

- From Chrome Window, modify the request to include an additional violation: <http://hackazon.f5demo.com/product/view<script>?id=72%00>
- This request should receive a block page because it contains violations that were not approved per override request
- From Chrome window, send requests for additional URLs matching the override pattern: <http://hackazon.f5demo.com/product/view?id=73%00>, <http://hackazon.f5demo.com/product/view?id=7%00>

Review

While a relatively simple scenario, the above lab exercise demonstrates the use of iRules in concert with the F5 ASM to handle special situations. The above example would have required a broader weakening of an organization's application security policy if the request from the business was relaxed directly by the ASM policy tweaks. Also, this type of change when deployed through a policy re-configuration often has downstream impact on orchestration and automation tools and can lead to false positives with vulnerability. Using an iRule, we were able to temporarily override the security policy without any policy changes, mitigate the exposed vulnerability, and meet the requirements outlined by the business representatives.

3.2.2 Additional Lab 2 - Content Rewrite

Scenario:

It probably goes without saying that many application developers aren't aware of (or at least good at) secure coding practices. How many applications in your environment have mixed HTTP and HTTPS content? iRule content rewrite solutions have been around for a long time. So you may have actually seen some of this code before.

And yes, in the case of mixed content HSTS does prevent this sort of oversight. But it's also important to understand that HSTS is a chainsaw when you may only need a butter knife. Once you've enabled HSTS in the browser and if you actually have legitimate HTTP content, you'll most definitely break user access.

The tried and tested example below is a perfect way to force HTTPS upon the browser, only as needed.

Requirements:

- BIG-IP LTM, web server, client browser, and SSL server certificate. If you don't have certificates to test with, you can use the CA certificate and server certificate and private key provided in the Client Certificate Inspection lab from Additional labs section.
- A stream profile attached to the virtual server.

Baseline Testing:

- Access the HTTPS URL <https://www.f5demolabs.com/content.html>
- Click the Test Link which is represented in the page with the href code below.

```
<a href="http://www.f5demolabs.com/images/f5logo.gif">Test Link</a>
```

The link will open as an HTTP URL.

The iRule:

```

1  when HTTP_REQUEST {
2      # Explicitly disable the stream profile for each request so it doesn't stay
3      # enabled for subsequent HTTP requests on the same TCP connection.
4      STREAM::disable
5      HTTP::header remove "Accept-Encoding"
6  }
7
8  when HTTP_RESPONSE {
9      # Apply stream profile against text responses from the application
10     if { [HTTP::header value Content-Type] contains "text" } {
11         # Look for the http:// and replace it with https://
12         STREAM::expression "@http://@https://@"
13         # Enable the stream profile
14         STREAM::enable
15     }
16 }
```

Analysis:

Event/Command details:

- HTTP_REQUEST event is triggered when there is a request for web page
- HTTP_RESPONSE event is triggered when the servers send a web page in response to a request
- STREAM::disable command disables the stream profile attached to the Virtual Server
- STREAM::enable command enables the stream profile attached to the Virtual Server.

Note: The STREAM::expression command always precedes the STREAM::enable command

- HTTP::header remove "Accept-Encoding" command removes "Accept-Enoding" from the http header
- HTTP::header value Content-Type command returns the type of content present in the payload.
- STREAM::expression command is used to search and replace the first argument with the second argument in the data stream

please check the wiki guide for iRules API in DevCentral at <https://devcentral.f5.com/wiki/irules.homepage.ashx> for additional info

Rule Details:

This rule does the following:

- Disables the stream filter attached to the virtual server for every incoming HTTP request
- Removes the support for encoding in the HTTP header for every incoming HTTP request
- Replaces every occurrence of “[http://](#)” with “[https://](#)” in the payload returned in the HTTP response when the content type is in the text format

Note: The STREAM command requires a stream profile attached to the virtual server

Testing:

In the BIG-IP,

- Add the above iRule to the HTTPS VIP
- Access the HTTPS URL <https://www.f5demolabs.com/content.html>
- **Click the Test Link which is represented in the page with the href code below**

```
<a href="https://www.f5demolabs.com/images/f5logo.gif">Test Link</a>
```

- Check that the link will open as an HTTPS URL

Review:

Some of the application developers are going to continue to include HTTP object references in their HTML code no matter how many times you’ve told them not to. In the above lab, we have used iRules to find and replace these references. In the iRule, we used the very powerful `STREAM` command to effortlessly sweep through the response payload and replace any instance of [http://](#) with [https://](#).

Please note that this string matching and replacing is not just limited to [http://](#). It can be applied to any type of text.

Bonus Activity:

Needless to say, `STREAM` is an incredibly powerful command, and a very useful tool in your security arsenal. For example, what if you also wanted to sanitize Social Security and credit card numbers

```
STREAM::expression "@\d3-\d2-\d4@***-**-****@ @\d4-\d4-\d4-\d4@xxxx-xxxx-xxxx-xxxx@"
```

Please refer to <https://devcentral.f5.com/wiki/irules.stream.ashx> for more details on the `STREAM` feature and its commands. You can also find some examples that show the application of the `STREAM` feature under each command.

3.2.3 Additional Lab 3 - HSTS / HPKP

HSTS

As per OWASP and RFC 6797, HTTP Strict Transport Security (HSTS) is an opt-in security enhancement that is specified by a web application through the use of a special response header. Once a supported browser receives this header that browser will prevent any communications from being sent over HTTP to the specified domain and will instead send all communications over HTTPS. It also prevents HTTPS click-through prompts on browsers. In layman’s terms, HSTS is an HTTP header sent from the server to the client browser. The information within that header indicates to the browser that any communication with

that domain must be over HTTPS, regardless of any HTTP URL's encoded into the HTML pages, or what the user types into the browser address bar. This information is also placed into the browser's long term storage, so that subsequent requests will immediately communicate over HTTPS. The header looks like this:

```
Strict-Transport Security: max-age=31536000; includeSubDomains
```

The max-age attribute defines how long the browser should store this information. In this case, it is 1 year. The includeSubDomains attribute is optional and indicates that the browser must communicate over HTTPS to this and any sub-domain of the current domain. To be most effective, this response header should a) only be transmitted over HTTPS in the first place to prevent stripping, and b) be transmitted at the lowest possible point in the domain to include all sub-domains. For example, to cover all sub-domains of domain.com, you might initially redirect a client to <https://domain.com> and then send an immediate redirect back to the requested HTTPS URL and include the HSTS header.

Note: Prior to BIG-IP LTM version 13.0, HSTS was implemented with an iRule (see below). As of v13, HSTS is simply enabled within an HTTP profile.

1. Create an HTTP profile.
2. Under the new "HTTP Strict Transport Security" section (bottom), set Mode to enabled (checked), set a maximum age in seconds and check the "Include Subdomains" option if you want the HSTS header to be sent for subdomains of this URL.
3. The Preload option is used by browser vendors to hard code this information into future browser updates. You must separately submit the URL to the vendors' preload lists. They will check that the preload option is set before hard coding your URL.

A word of warning: once browser vendors hard code this URL into new versions, it is practically impossible to remove it, so make sure this is exactly what you want and that no "mixed" content (HTTP and HTTPS) exists for this URL.

HPKP

As per OWASP and RFC 7469, HTTP Public Key Pinning (HPKP) is a new HTTP header that allows SSL servers to declare hashes of their certificates with a time scope in which these certificates should not be changed. In other words, it is a method by which a server can indicate to a browser the certificate that the client browser should see. This technology helps to prevent active man-in-the-middle attacks whereby an agent is able to silently decrypt and re-encrypt traffic between a client and server. Without knowledge of the server's private key, the agent won't possibly be able to spoof the server's real certificate. Therefore the "forged" certificate coming from the agent will not match the certificate embedded in the HPKP header. That header looks like this:

```
Public-Key-Pins: max-age=2592000;
pin-sha256="E9CZ9INDbd+2eRQozYqqbQ2yXLVKB9+xcprMF+44U1g=";
pin-sha256="LPJNul+wow4m6DsqxnbnihsWHlwfp0JecwQzYpOLmCQ=";
report-uri="http://example.com/pkp-report"
```

The header name is `Public-Key-Pins` and has a similar max-age attribute as HSTS. The header can include multiple encoded certificate hashes indicated as the `pin-sha256` values. If the incoming certificate does not match anyone of these values, a "report-uri" attribute can send the browser to a reporting facility to report this error (possible attack). To be most effective, this response header should only be transmitted over HTTPS to prevent stripping. To create the encoded value for a given certificate, use the following OpenSSL commands:

```
openssl x509 -in <cert> -pubkey -noout | openssl rsa -pubin -outform der |  
openssl dgst -sha256 -binary | openssl enc -base64
```

Scenario

In this lab exercise, we demonstrate how to deploy both the HSTS and HPKP using an iRule.

Requirements

- BIG-IP LTM, web server, client browser, and a SSL server certificate. If you don't have certificates to test with, you can use the CA certificate, server certificate, and the private key provided in the the Client Certificate Inspection lab from Additional labs section.

The iRule

```
1  when RULE_INIT {  
2      set static::fqdn_pin1 "X3pGTSOuJeEVw989IJ/cEtXUEmy52zs1TZQrU06KUKg="  
3  
4      # Set max_age to 180 days  
5      set static::max_age 15552000  
6  }  
7  when HTTP_RESPONSE {  
8      # Insert an HSTS header  
9      HTTP::header insert Strict-Transport-Security "max-age=$static::max_age;␣  
↪includeSubDomains"  
10     # Insert an HPKP header  
11     HTTP::header insert Public-Key-Pins "pin-sha256=\"${static::fqdn_pin1}\" max-age=  
↪${static::max_age}; includeSubDomains"  
12 }
```

Analysis

- The above iRule example will perform two functions. Upon receipt of the HSTS header, any attempt to communicate with that VIP again will only use HTTPS. With the injection of the HPKP header, if the certificate (CA or server certificate) does not match the one provided in the SSL handshake, the browser will either end the connection and potentially follow a report-uri URL if it exists.

Testing

- Apply this iRule to an HTTPS virtual server (VIP).
- **Repeatedly navigate to the HTTP URL <http://www.f5demolabs.com> to** verify that you are indeed talking to the HTTP VIP.
- **Navigate to the HTTPS URL <https://www.f5demolabs.com> one time to** verify that you can access it.
- **Now attempt to go to the HTTP URL <http://www.f5demolabs.com> again.** Depending on the browser it should immediately go to the HTTPS URL.

- **If you're using a Chrome browser, you can navigate to** `chrome://net-internals/#hsts` to see this URL value now added to Chrome's HSTS list. Under Query Domain, enter `www.f5demolabs.com` to Domain: entry box and click Query. Be sure to delete domain before moving on or else you will have an issue with a later lab.
- **Unfortunately, unless you're using a server certificate that chains** up to a public root, you won't be able to test HPKP here. Per the Mozilla Developer Network, "Firefox (and Chrome) disable Pin Validation for Pinned Hosts whose validated certificate chain terminates at a user-defined trust anchor (rather than a built-in trust anchor). This means that for users who imported custom root certificates all pinning violations are ignored."

Hint: You can still use Chrome Developer Tools to see the HPKP header.

3.2.4 Additional Lab 4 - DNS Hooks: Amplification attackers

Scenario:

You have a F5 DNS deployed to service DNS queries for external DNS. To meet the business requirements, you must allow DNS queries from any DNS resolver. Basic DNSSEC has been implemented as part of your GTM deployment. As your DNS deployment expands for more applications, you experience a DNS Amplification attack.

A DNS amplification attack takes advantage of features that allow a very small request to return a much larger response. These attacks also rely on the fact that the attacker can request these large responses on behalf of someone else (the victim). More specifically, DNS amplification attacks are a popular type of a Distributed Denial of Service (DDoS) attack in which attackers use publicly accessible open DNS resolvers to flood a target system with DNS response traffic. One other key piece to this puzzle is that DNS uses the User Datagram Protocol (UDP) to send requests and responses. An attacker floods your DNS system with an "ANY" query that returns all known information about a DNS zone in a single request.

Figure 1:

Figure 2:

Here we see an example of an DNS Amplification attack using an open DNS resolver

Restrictions:

The following restrictions complicate the request from the business to relax the enforced security posture:

- Respond to queries from any source, even open resolvers.
- Respond to DNS queries over both TCP and UDP.

Requirements:

To meet the business's objectives while maintaining a strong security policy, an iRule solution must meet the following requirements:

- Checks to see if the query type is “ANY” and responds with a truncated message which will force the legitimate client to use TCP.
- Allows for a flexible and updatable list of networks allowed to do recursive lookups.
- The only host allowed to do recursive lookups is 10.1.10.51

Baseline Testing:

Prior to defining a solution, validate the current behavior:

- RDP to lab jump station
- From the BIG-IP, Select Local Traffic >> Virtual Servers >> Statistics. Reset the statistics for all the virtual servers.
- Open Terminal application
- From Terminal run the following command against the open resolver (F5 DNS)

```
f5student@xjumpbox~$ dig f5.com @10.1.10.153 ANY +noall +comments
```

You should get an answer from the BIG-IP functioning as our open resolver that looks similar to below image, and most significantly is missing any notes about truncation:

```
f5student@xjumpbox:~$ dig f5.com @10.1.10.153 ANY +noall +comments
; <<>> DiG 9.10.3-P4-Ubuntu <<>> f5.com @10.1.10.153 ANY +noall +comments
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 22104
;; flags: qr rd ra; QUERY: 1, ANSWER: 11, AUTHORITY: 4, ADDITIONAL: 1
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
f5student@xjumpbox:~$
```

- From Terminal run the following command against the open resolver (F5 DNS)

```
f5student@xjumpbox~$ dig f5.com @10.1.10.153 ANY
```

The resolver will recursively query the f5.com domain name, and return all records associated with this domain. Take a look at the MSG SIZE field at the end of the response.

- From the Terminal run the following command against the open resolver (F5 DNS)

```
f5student@xjumpbox~$ dig f5.com @10.1.10.153
```

Again, the resolver will recursively query for the A records from f5.com. Take a look at the MSG SIZE field. The ANY response from f5.com was ~6X the size of the A query. Now, imagine if the attacker sent a query to a bogus domain which they have populated with thousands of bogus records.

- From the Terminal run the following command against the open resolver (F5 DNS)

```
f5student@xjumpbox~$ dig test1.f5demolabs.com @10.1.10.153
```

- Repeat the same command, this time add 'ANY' to the end of the query to request all records for test1.f5demolabs.com

Tip: In this lab, we have two DNS Express zones defined, f5demolabs.com and badf5demolabs.com. The above queries validate we are able to resolve names from f5demolabs.com DNSX zone.

- From the BIG-IP, Select Local Traffic >> Virtual Servers >> Statistics. Check statistics on the sec_irules_dns_udp and sec_irules_dns_tcp virtual servers. At this point, we are forcing any traffic to TCP listener, so all traffic should be hitting the udp virtual server.

With the above steps complete, we have verified that without our iRule solution in place we are able to do the following: - Recursively resolve queries from any host for any record type, which is perfect for an attacker looking to trigger a DNS amplification attack. - Resolve queries from DNS Express zones defined on F5 DNS.

The iRule:

UDP VIP iRule

```
when RULE_INIT {
    set static::dns_dbg 1
}

when DNS_REQUEST {

    if {$static::dns_dbg} {
        log local0. "DNS Question Type: [DNS::question type]"
    }

    if { [DNS::question type] eq "ANY" } {
        DNS::answer clear
        DNS::header tc 1
        DNS::return
    }
}

when DNS_RESPONSE {
    if {$static::dns_dbg} {
        log local0. "DNS Origin: [DNS::origin] "
    }
    if { [DNS::origin] eq "TCL" } {
        return
    } elseif { [DNS::origin] ne "DNSX" } {
        if {$static::dns_dbg} {
            log local0. "Client IP: [IP::client_addr] "
        }
        if { not [class match [IP::client_addr] eq "admin_datagroup" ] } {
            DNS::drop
        }
    }
}
```

TCP VIP iRule

```
when DNS_RESPONSE {  
  if {$static::dns_dbg} {  
    log local0. "Client IP: [IP::client_addr], DNS Origin: [DNS::origin]"  
  }  
  if { [DNS::origin] ne "DNSX" } {  
    if { not [class match [IP::client_addr] eq "admin_datagroup" ] } {  
      DNS::drop  
    }  
  }  
}
```

Rule Details:

UDP VIP iRule

This first part checks if the DNS query type is “ANY” and responds with a truncated header. The second part checks to see if the response packet is built from the first logic (origin = TCL). If yes, then exit and do not process further. If no, then check if the response is from DNS Express. If it is, allow an answer for non “ANY” type. If it is not from DNS Express, check to see if it matches the admin_datagroup created for recursive allowed networks. If it does not match both conditions, then drop.

TCP VIP iRule

Simple logic to check and see if the response is from DNS Express or a part of the admin_datagroup. If it is not from DNS Express, check to see if it matches the admin_datagroup created for recursive allowed networks. If it does not match both conditions, then drop.

Testing:

- From the BIG-IP, Select Local Traffic >> Virtual Servers >> Statistics. Reset the statistics for all the virtual servers.
- Navigate to Local Traffic -> Virtual Servers -> Virtual Server List -> sec_irules_dns_udp
- Click the Resources tab, then the Manage button to the right of the iRules section header
- Move the iRule sec_irules_dns_hook-udp from the Available box to the Enabled box
- Click Finished
- Open Terminal application
- From Terminal run the following command against the open resolver (F5 DNS)

```
f5student@xjumpbox~$ dig f5.com @10.1.10.153 ANY +noall +comments
```

You should get an answer from the BIG-IP functioning as our open resolver that looks below, this time you should see the DNS response has been truncated forcing the client to retry using TCP.

```
f5student@xjumpbox:~$ dig f5.com @10.1.10.153 ANY +noall +comments
;; Truncated, retrying in TCP mode.

; <<>> DiG 9.10.3-P4-Ubuntu <<>> f5.com @10.1.10.153 ANY +noall +comments
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 15365
;; flags: qr rd ra; QUERY: 1, ANSWER: 12, AUTHORITY: 4, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
```

- Navigate to Local Traffic -> Virtual Servers -> Virtual Server List -> `sec_irules_dns_tcp`
- Click the Resources tab, then the Manage button to the right of the iRules section header
- Move the iRule `sec_irules_dns_hook-tcp` from the Available box to the Enabled box
- From Terminal run the following command against the open resolver (F5 DNS)

```
f5student@xjumpbox~$ dig f5.com @10.1.10.153 ANY
```

This time, you will see you get a truncated response over UDP, and attempts to execute the query over TCP fail. The requests over TCP are failing b/c the iRule is filtering all requests for non DNS Express zones, and only allowing clients in the `admin_datagroup` whitelist.

- Navigate to Local Traffic -> iRules >> Data Group List, and select `admin_datagroup`
- Add the address 10.1.10.51 with no value to the list
- From Terminal, repeat the query we just issued in previous step

This time, the query sent over TCP should receive a valid response. With the client IP added to the `admin_datagroup` whitelist, the client is now able to execute DNS queries for non DNS Express domains.

- From BIG-IP return to the data group list, and remove 10.1.10.51 from the address section.
- From the Terminal run the following command against the open resolver (F5 DNS)

```
f5student@xjumpbox~$ dig test1.f5demolabs.com @10.1.10.153
```

- Repeat the same command, this time add 'ANY' to the end of the query to request all records for `test1.f5demolabs.com`
- The last two queries were for records in the `f5demolabs.com` domain previously defined in a DNS Express zone on the F5 DNS. So, even though our client is no longer defined in the `admin_datagroup`, it is still able to use the resolver to resolve entries in the DNS.
- From the BIG-IP, Select Local Traffic >> Virtual Servers >> Statistics. Check statistics on the `sec_irules_dns_udp` and `sec_irules_dns_tcp` virtual servers.
- With the iRules in place, you will see traffic is being picked up on both the TCP and UDP listeners.

Review:

It is absolutely bad practice for most organizations to publicly expose open DNS resolvers. Doing so provides a perfect tool for attackers to trigger amplification attacks against unsuspecting targets. Attackers take advantage of the fact that DNS leverages UDP, and therefore they can use spoofed IP addresses to trigger massive attacks. In this lab, we demonstrated how a customer can use iRules to expose an open resolver, but force recursive queries of the type commonly used in amplification attacks (ANY) to use TCP. TCP makes these kinds of amplification attacks impossible b/c the attacker would essentially wind up attacking

themselves. Then, we extended our iRules to filter any queries requiring recursive lookups to be filtered against a predefined list of allowed sources. Finally, our iRules allow all hosts to be able to execute queries against local hosted DNS Express zones without filtering.

Bonus Activity:

In this lab, we had pretty specific allow/disallow logic in the rules. However, another approach might be to provide rate limiting on the number of recursive queries we would allow from a given host. As a bonus activity, see if you can use some of the logic from the HTTP Throttling lab to provide a solution that rate limits all recursive requests ANY requests.

3.2.5 Additional Lab 5 - Geolocation

Scenario

All of your critical applications are protected by F5 Advanced Web Application Firewall (AWAF), and leverage F5's Layer 7 DoS feature to mitigate bot activity and protect application resources from layer 7 volumetric attacks. To simplify the initial deployment, the application security team elected to disable F5's Proactive Bot Defense (PBD) feature.

Recently, the business analysis team has noticed a significant increase in the application traffic from Russia and believe much of this traffic to be a bot related activity. Since this traffic is having a negative impact on the business's ability to analyze data and increasing load on the server infrastructure, the business is requesting you to take a more aggressive action on traffic sourced from Russia. The security team would like to leverage PBD for this traffic to block the simple automated bot activity.

Restrains

The following restraints complicate this request from the business:

- AWAF DoS Profile allows you to whitelist/blacklist geolocations globally across the DoS profile and allows for specific thresholds to be defined for geolocations for Transaction Per Second (TPS) and Stress-based protections. However, it does not allow for per geolocation enabling/disabling of PBD.

Requirements

To meet the business's objectives, while still maintaining a strong security policy, an iRule solution must meet the following requirements:

- Proactive Bot Defense should be enabled for all traffic from Russia, but disabled for traffic initiated from everywhere else.
- Bot Signature protection should remain enforced for all traffic.
- Selectively enabling PBD should **not** affect any of the existing L7DoS protections currently enforced.

Baseline Testing

Prior to defining a solution, validate the issue by testing the application to validate AWAF's current behavior:

- RDP to the lab jump station
- Open Terminal application

- From Terminal run the following command against the test web application

```
f5student@xjumpbox~$ curl -k http://hackazon.f5demo.com/ -H "X-forwarded-for: 5.16.0.1" | grep -i ?type=
```

- The result of the test should look similar to below, with grep returning no match, and the object response size ~64k

The screenshot shows a terminal window titled "Terminal - f5student@xjumpbox: ~". The command executed is `curl -k -H "X-Forwarded-For: 5.16.0.1" http://hackazon.f5demo.com/ | grep -i ?type=`. The output shows a successful curl request with a 200 status code and a response size of 64286 bytes. The output also shows the XFF header value as 5.16.0.1. The terminal output is as follows:

```
f5student@xjumpbox:~$ curl -k -H "X-Forwarded-For: 5.16.0.1" http://hackazon.f5demo.com/ | grep -i ?type=
% Total    % Received % Xferd  Average Speed   Time    Time     Time
100 64286  100 64286    0     0  519k      0 --:--:-- --
f5student@xjumpbox:~$
```

- PBD is not active and not responding to the HTTP request with javascript challenge
- From Terminal, run the same command, but change the value of the X-forwarded-for header to be 2.2.2.2
- Traffic sourced from Russia should match the behavior of all other geolocations, and no proactive bot defense challenges are being issued.

The iRule

```
1  when CLIENT_ACCEPTED {
2      set geopbd_debug_verb 1
3      set geopdb_debug 1
4  }
5
6  when HTTP_REQUEST {
7      if { [HTTP::header exists "X-Forwarded-For"] } {
8          set XFF [getfield [lindex [HTTP::header values X-Forwarded-For] 0] "," 1]
9      }
10     else {
11         set XFF [IP::client_addr]
12     }
13
14     if {$geopbd_debug_verb} {
15         log local0. "Continent: [whereis $XFF continent]"
16         log local0. "Country: [whereis $XFF country]"
17         log local0. "State: [whereis $XFF state] "
18         log local0. "ISP: [whereis $XFF isp] "
19         log local0. "Org: [whereis $XFF org] "
20     }
21 }
```

(continues on next page)

(continued from previous page)

```

22  if {!([whereis $XFF country] equals "RU")) {
23      if {$geopdb_debug} {
24          log local0. "De-activating PBD: Not Russia source"
25      }
26      BOTDEFENSE::disable
27  }
28
29  }
30
31  when BOTDEFENSE_ACTION {
32      #catch the inbound status
33      if {$geopdb_debug} {
34          log local0. " Geolocation Country: [whereis $XFF country] "
35          log local0. " Bot Defense Status: [BOTDEFENSE::reason] "
36          log local0. " Bot Defense Action: [BOTDEFENSE::action] "
37      }
38  }

```

Analysis

Event/Command details:

- The iRules `whereis` command can take several options, including:
 - `[whereis [IP::client_addr] continent]`: returns the three-letter continent
 - `[whereis [IP::client_addr] country]`: returns the two-letter country code
 - `[whereis [IP::client_addr] <state|abbrev>]`: returns the state as word or as two-letter abbreviation
 - `[whereis [IP::client_addr] isp]`: returns the carrier
 - `[whereis [IP::client_addr] org]`: returns the registered organization
- `BOTDEFENSE` command enables or disables bot defense processing
- `BOTDEFENSE_ACTION` event is triggered after the HTTP request has been processed, and just prior to taking action on transaction. The event is triggered whenever PBD is enabled, if a DoS L7 attack is configured to trigger PBD, or when a Bot Signature was detected on the request.
- `BOTDEFENSE::reason` returns the reason the for the bot defense action
- `BOTDEFENSE::action` returns the action to be taken by bot defense feature

Rule Details

This rule does the following:

- Inspects the inbound X-Forwarded-For header or Client IP address, and performs a geolocation lookup on the value. If either the XFF or the Client IP do **not** match the Russia country code, "RU", then botdefense is disabled. Otherwise Bot Defense is enabled.
- Logs the geolocation information on to a local logger
- Logs the botdefense reason and action to a local logger

Note: This rule uses the DoS Profile, iRules_Sec, which has been created for you as part of the lab setup

Testing

From BIG-IP UI:

- Navigate to Security -> DoS Protection -> DoS Profiles -> iRules_Sec -> Application Security Tab
- Click the Proactive Bot Defense button, and set the Operation Mode to Always
- Click Update
- Navigate to Local Traffic -> Virtual Servers -> Virtual Server List -> vs_hackazon_http
- Click the Resources tab, then the Manage button to the right of the iRules section header
- Move the iRule sec_irules_geobased_pbdswitcher from the Available box to the Enabled box
- Click Finished
- Open Terminal application, and create a new tab, then run following command

```
f5student@xjumpbox~$ ssh root@10.1.1.245
```

- From BIG-IP console run the following command:

```
f5student@xjumpbox~$ tail -f /var/log/ltn
```

- On original Terminal Application tab, run the following command:

```
f5student@xjumpbox~$ curl -k http://hackazon.f5demo.com/ -H "X-forwarded-for: 5.16.0.1" | grep -i ?type=
```

- Response should look similar to below image. You should see that PBD has injected a javascript challenge, and the response body should be ~5.8K

```
f5student@xjumpbox:~$ curl -k -H "X-Forwarded-For: 5.16.0.1" http://hackazon.f5demo.com | grep -i ?type=
% Total    % Received % Xferd  Average Speed   Time
             Dload  Upload   Total             0 0 0 0 0 0
100  5823  100  5823    0     0  434k    0  --:--:--
<script type="text/javascript" src="/TSPD/08309ab76bab09904bca502c7ddab7c8b988c118be5bc97d9bccad?type=10"></script>
f5student@xjumpbox:~$
```

- From Terminal, run the same command but change the value of the X-forwarded-for header to be 2.2.2.2
- This request is not issued from a Russian source, so PBD does not issue a challenge. The response is missing the challenge, and the response body is ~64K.
- From BIG-IP UI, view the Bot Defense logs:
- Security -> Event Logs -> Bot Defense -> Requests

- In this log, look at requests from 5.16.0.1 and 2.2.2.2
- You will see both requests are properly classified as bots, but only requests from 5.16.0.1 are challenged
- On Xubuntu Jumpbox, open another Firefox tab
- browse to <http://hackazon.f5demo.com/>
- Return to BIG-IP Bot Defense log
- Notice browser issued requests will source from 10.1.10.51, and will show the following:
 - Request Status = Legal
 - Action = allow
 - Reason = Bot Defense Inactive

Note: Bot Defense is inactive, because the request wasnt sourced from “Russia”, and we have disabled PBD.

- Return to Firefox, and right click the Firefox Modify Header Add-on on the right-side of the screen
- Select Open options page
- Scroll all the way to bottom of options screen and click the disable box in the rule for <http://hackazon.f5demo.com>
- verify the box turns blue. This enables insertion of X-Forwarded-For header in browser request
- Again, browse to <http://hackazon.f5demo.com>
- Return to BIG-IP Bot Defense log
- Notice browser issued requests will source from 5.16.0.1, and will show the following:
 - Geolocation = RU
 - Request Status = Legal
 - Action = browser_challenged (on request for first object), and allow on subsequent requests
 - Reason = No Valid Cookie: Challenge is possible (on request for first object), and Valid Cookie: No need to review on subsequent requests

Review

Geolocation, while not foolproof, is often an important piece of context about a user or device. Proactive Bot Defense is a very powerful feature for mitigating bot and automated activity but sometimes, it is challenging to implement in a single broad stroke.

In the above lab, we have used iRules to take advantage of the additional context gained through the iRule geolocation commands to leverage a very powerful security feature in a targeted manner. This is precisely the kind of challenge iRules are best suited for, stitching together pieces of information and features to deliver a solution customized to solve a business challenge.

Bonus Activity

One of our existing requirements was to not change any of our existing L7DoS protections. In the lab, we demonstrated that changes via iRule didnt affect Bot Signatures. As a bonus, you can also verify the iRule

enforced PBD for the Russian sources also doesn't impair the pre-existing L7DoS protections configured in the DoS profile.

- Return to Firefox and right-click the Firefox Modify Header Add-on on the right-side of the screen
- Click the Disable button. This time turning it gray
- From the browser tab, open <http://hackazon.f5demo.com>
- Click the refresh icon rapidly for ~30 seconds
- You will see the requests beginning to fail. This is the L7DoS protection kicking in and rate limiting requests from non-Russian sources
- Return to BIG-IP UI
- Navigate to Security -> Event Logs -> DoS -> Application Events
- You should see a L7DoS attack has been triggered and detected by Source IP TPS
- Repeat same steps, but after re-enabling the X-Forwarded-For header in the browser add-on
- You should be able to trigger an attack, but this time using a Russian source.

With the above steps, you have demonstrated that you can inject PBD challenges for sources from a given geolocation while maintaining all pre-existing protections. We have just used more context to enable more security using an iRule!

3.2.6 Additional Lab 6 - IP Reputation

IP Reputation is a subscription-based service that provides an F5 BIG-IP with an active and upto date set of data for a known "bad actors". In this case we're going to look at the source address (or in our test case an X-Forwarded-For HTTP header) to determine if this is a known bad guy. If you don't already have an IP Reputation subscription, contact your local F5 representative for a time-limited evaluation license.

Requirements

- BIG-IP LTM, web server, and a client (command line cURL)

The iRule

```

1  when HTTP_REQUEST {
2      # Use [HTTP::header values "X-Forwarded-For"] for testing
3      #set iprep_categories [IP::reputation [IP::client_addr]]
4      set iprep_categories [IP::reputation [HTTP::header values "X-Forwarded-For"]]
5      set is_reject 0
6      if { $iprep_categories contains "Windows Exploits" } {
7          set is_reject 1
8      }
9      if { $iprep_categories contains "Web Attacks" } {
10         set is_reject 1
11     }
12     if { $iprep_categories contains "Scanners" } {
13         set is_reject 1
14     }
15     if { $iprep_categories contains "Proxy" } {
16         set is_reject 1

```

(continues on next page)

(continued from previous page)

```
17     }
18     if { $is_reject } {
19         log local0. "Attempted access from malicious IP address [HTTP::header values
↪ "X-Forwarded-For"] ($iprep_categories) - rejected"
20         HTTP::respond 200 content "<HTML><HEAD><TITLE>Rejected Request</TITLE></HEAD>
↪ <BODY>The request was rejected . <BR>Attempted access from malicious IP address</
↪ BODY></HTML>"
21     }
22 }
```

Analysis

- With an active subscription to the IP Reputation service, iRules have access to a wealth of near real-time information about bad actors, including exploit sites, scanners, proxies, and others.
- With this example iRule, a detected bad site will immediately generate an HTML failure page and reject access to the web application.

Testing

- Apply this iRule to an HTTP virtual server (VIP).
- From a command line, issue a cURL command and include an X-Forwarded-For header:

```
curl http://www.f5test.local -H "X-Forwarded-For: 186.64.120.104"
```

- Here are a few bad IP addresses to test:

- 103.4.52.150
- 101.200.81.187
- 103.19.89.118
- 103.230.84.239

- Here are a few bots to test:

- 187.174.252.247
- 188.120.224.250
- 188.219.154.228
- 188.241.140.212

Note: Using `iprep_lookup <IP_addr>` on the BIG-IP command line will provide ip reputation information.

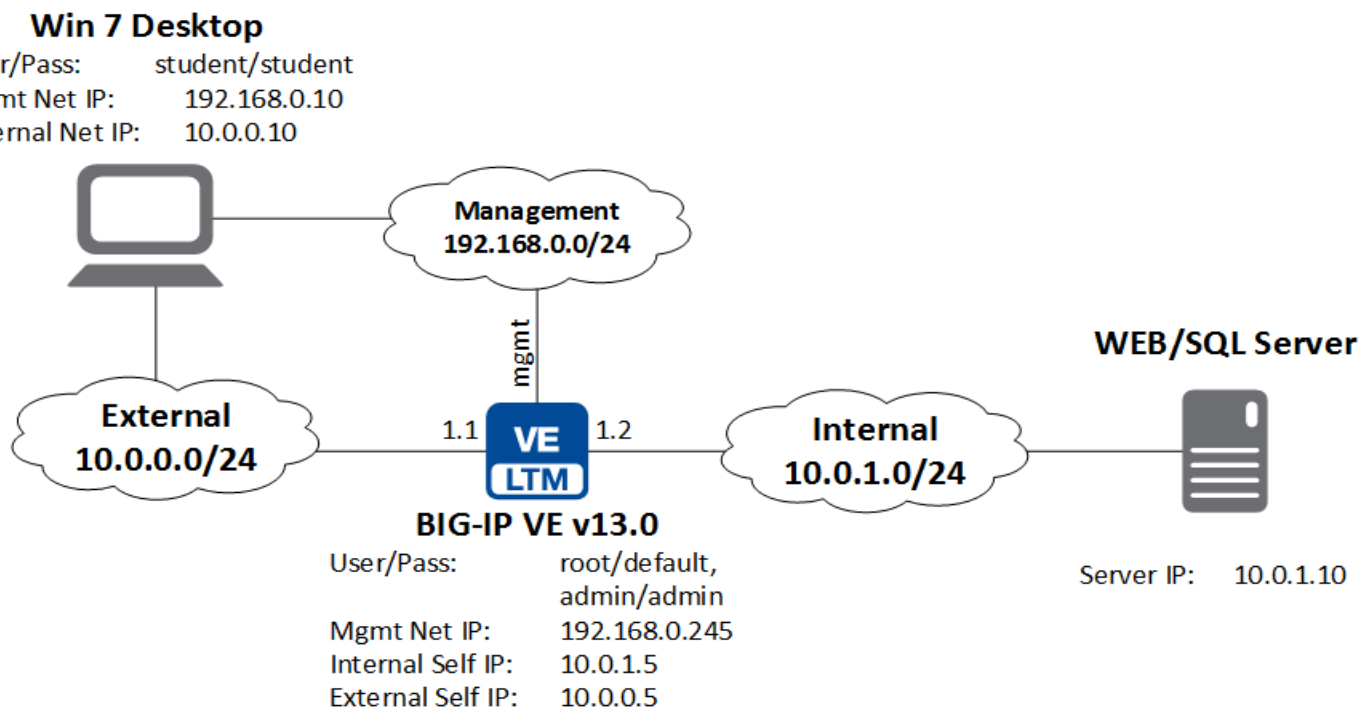
Introduction to iRules LX

In this class we will learn how to use iRules LX with a basic example. We will have a web app that has a web form. When we submit the form, the page will display our POST data. As part of the lab exercise, we will apply an LX iRule that will convert the form POST data into JSON and change the Content-Type header.

Using Your Lab Environment

You will be using Ravello for this lab. We will be working with a Windows 7 desktop, a BIG-IP 13.0 LTM VE and a server with HTTP and SQL services enabled. We will be using the Windows machine as our desktop for accessing the applications on the BIG-IP.

This diagram shows the topology of the network as it is currently configured -



How to Access the Labs

You will receive instructions from your proctor on how to access the workstation in the lab. On this workstation, you will have the following applications –

- Atom Editor – For viewing lab code with syntax highlighting. When you open up Atom, you will see a list of files that will be used in these labs.
- Chrome Web Browser – For testing the applications we create and BIG-IP management access. Links are bookmarked just below the address bar.
- Putty SSH Client – For accessing the BASH and TMSH command line of the BIG-IP. The BIG-IP properties have been saved to the session labeled *BIG-IP*.

4.1 Creating and Implementing an LX iRule

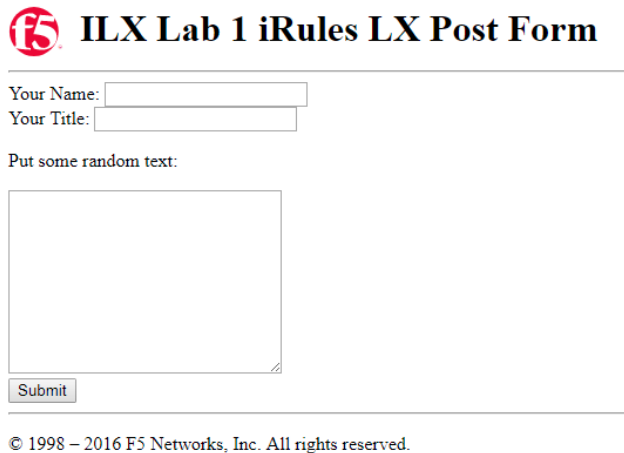
In this lab we will learn how to use iRules LX with a basic example to introduce you to the concepts of iRules LX and their configuration objects. We will have a web app that has a web form. When we submit the form, the page will display our POST data. As part of the lab exercise, we will apply an LX iRule that will convert the form POST data into JSON and change the Content-Type header.

The practicality of this use case could be when you have some type of legacy front end service that can only POST data as a standard query string, but a new back end service takes data as JSON. It would be pretty impractical to use iRules TCL to perform the translation. However, this task is trivial for iRules LX because of the power of Node.js. We will implement an LX iRule that will accomplish this.

4.1.1 Lab 1 - Creating and Implementing an LX iRule

Test and Review the Existing Configuration

To start off we have a web application that has a web form that we enter some information into and submit. Now let's look at the web app at the URL <http://10.0.0.20/ilxlab1/> (Lab 1 on bookmarks). The response of the POST will show our form data and “Content-Type” header. Here is the example of the web form –



The screenshot shows a web browser displaying a form titled "ILX Lab 1 iRules LX Post Form". The form includes a red circular logo with a white 'f5' inside. Below the logo, there are two input fields: "Your Name:" and "Your Title:". Below these fields is a text area with the placeholder text "Put some random text:". At the bottom of the form is a "Submit" button. Below the form, there is a copyright notice: "© 1998 – 2016 F5 Networks, Inc. All rights reserved."

Go ahead and run your own test of the web app. Observe the “Content-Type” header and POST data values. Here is an example of the response to a POST.

iRules LX POST Data

The Content-Type header is: **application/x-www-form-urlencoded**

This is your POST data:

```
name=Eric&title=NPI&randomText=Some+random+text+for+this+lab
```

© 1998 – 2016 F5 Networks, Inc. All rights reserved.

Go ahead and run your own test of the web app. Observe the “Content-Type” header and POST data values. Here is an example of the response to a POST.

Create the LX Workspace

The first thing we need to do is create an LX Workspace. On the BIG-IP, navigate over to the LX workspaces menu in the tab located at *Local Traffic > iRules > LX Workspaces*. Then select the create button at the top right of the table and name the workspace *ilxlab1*. You will now have an empty workspace.

Create the Extension

Next, we create a new extension (the Node.js code that will run). The name of the extension will matter later because we will call that from our iRules TCL code.

To create the extension, click the *Add Extension* button at the bottom of the editor, then give it the name *ilxlab1_ext*. The various files of the extension will show up. Select the *index.js* file and you should see a template of example code in the editor window. Normally you could use this example code as a starting point, but in our case we should delete all the example code from the window. In the Atom editor, locate the file named *ilxlab1.js* and double click it which should open it in a text editor. Copy and paste this into the *index.js* file on our BIG-IP.

Just for reference, here is the code:

```
1 'use strict' // Just for best practices
2 // Import modules here
3 var f5 = require('f5-nodejs');
4 var qs = require('querystring'); // Used for parsing the POST data querystring
5
6 // Create an ILX server instance
7 var ilx = new f5.ILXServer();
8
9 // This method will transform POST data into JSON
10 ilx.addMethod('jsonPost', function (req, res) {
11   // Get POST data from TCL and parse to JS object
12   var postData = qs.parse(req.params()[0]);
13
14   // Turn postData object into JSON and return to TCL
15   res.reply(JSON.stringify(postData));
16 });
17
```

(continues on next page)

(continued from previous page)

```

18 //Start the ILX server
19 ilx.listen();

```

Then you will need to save the changes to this file with the *Save File* button at the bottom of the editor window.

Create the TCL iRule

Next, we need to create the TCL iRule that will call our Node.js code. Click the button *Add iRule* at the bottom of the editor window, name the iRule *json_post* and don't check the box to include example code (we don't need the example code for this lab). In the Atom editor, locate the file named *ilxlab1.tcl* a. Copy and paste this into the *json_post* iRule file.

Just for reference, here is the code.

```

1  when HTTP_REQUEST {
2      # Collect POST data
3      if { [HTTP::method] eq "POST" }{
4          set cl [HTTP::header "Content-Length"]
5          HTTP::collect $cl
6      }
7  }
8  when HTTP_REQUEST_DATA {
9      # Send data to Node.js
10     set handle [ILX::init "ilxlab1_pl" "ilxlab1_ext"]
11     if {[catch {ILX::call $handle jsonPost [HTTP::payload]} result]} {
12         # Error handling
13         log local0.error "Client - [IP::client_addr], ILX failure: $result"
14         HTTP::respond 400 content "<html>There has been an error.</html>"
15         return
16     }
17
18     # Replace Content-Type header and POST payload
19     HTTP::header replace "Content-Type" "application/json"
20     HTTP::payload replace 0 $cl $result
21 }

```






Then you will need to save the changes to this file with the *Save File* button at the bottom of the editor window.

Create the LX Plugin

Now that we have our code in a workspace, you will need to navigate over to the LX Plugins menu in the tab located at *Local Traffic > iRules > LX Plugins*. Click the *Create* button, name the plugin *ilxlab1_pl*, select the *ilxlab1* workspace and click *Finished*. This makes the Node.js code active.

Apply the LX iRule to the Virtual Server

Now that we have our Node.js code running, we can put it to use. In order to use the code from the plugin we must assign the TCL iRule to a virtual server. Just so we can be familiar with it (but it is not required), we will look for the TCL iRule in the *Local Traffic > iRules > iRules List* menu. You will find the iRule that we created in the workspace located there with a Partition/Path that has the same name as our plugin.

	_sys_auth_tacacs		F5 Verified	f5-irule	Common
	_sys_https_redirect		F5 Verified	f5-irule	Common
	json_parse		None		Common/ixlab2_pl
	json_post		None		Common/ixlab1_pl
	mysql		None		Common/ixlab3_pl

You won't be able to make changes from here. This is the same behavior as an iApp with strict updates enabled.

Now navigate over to our virtual server list, click the *Edit* button (under the *resources* column) for the virtual *ixlab1_vs* and select the *Manage* button for iRules. If you scroll to the bottom of the available iRules list, you should see the iRule from our plugin.

Resource Management

iRule

Enabled

Available

<<

>>

Up Down

_sys_https_redirect
/Common/ixlab1_pl
json_post
/Common/ixlab3_pl
mysql

Move this iRule to the over to the enabled section and click finished.

Testing the LX iRule

Now let's navigate to the second tab on the browser with the web page of our app. Go back to the web form and submit the information again. You will see now that the data has been converted to JSON and the *Content-Type* header has been changed.

iRules LX POST Data

The Content-Type header is: **application/json**

This is your POST data:

```
{"name":"Eric","title":"NPI","randomText":"Some random text"}
```

© 1998 – 2016 F5 Networks, Inc. All rights reserved.

As you can see, with iRules LX we can implement solutions with very few lines of code. If we wanted to accomplish the same goal in TCL alone, it would most likely take several hundred lines of code.

Workspace Package Management

Lastly, we will show package management for LX workspaces. While it is fairly simple to move TCL iRules from a dev/test environment to production because it is a single file, iRules LX can have an almost unlimited

number of files depending on how many NPM modules a solution needs. Therefore, workspaces have been given the ability to export and import packages as a `tgz` file to have a more convenient method of transporting iRules LX code. In this exercise, we will export our package and import it back into the same device (but normally import would happen on a separate BIG-IP).

Export/Import a Workspace

Go to the *LX Workspaces* list, check the box of our *ilxlab1* workspace and click the *Export* button below the list. This will save the file to the user's *Downloads* folder.

Now click the *Import* button on the top right hand corner of the workspace list. On the next window give the imported workspace the name of *ilxlab1_restore*, select the option *Archive File*, and use the *Choose File* button to find the `tgz` file in the user's *Downloads* folder. When you click the *Import* button you will be taken back to the workspace list and you should see the imported workspace now. Feel free to navigate into the imported workspace.

You have concluded lab exercise #1.

4.2 NPM and Exception Handling

In this lab exercise, we will cover two topics. In the first topic we will introduce you to the concept of exception handling to prevent crashes of our node process and the second topic will be installing packages with NPM. The second portion topic will build upon the work done in the first topic.

4.2.1 Lab 1 - NPM and Exception Handling

Test and Review the Existing Configuration

In this lab we will be working from the file `ilxlab2_steps.js`. You will be **cutting and pasting** code from this file as directed. We will be working with the virtual server (10.0.0.21) & workspace named *ilxlab2* which already has some base code in it. The plugin has already been created and the TCL iRule are already assigned to the virtual server.

To start off we have a web application with a web form that we enter some information into and submit. The response of the POST will show our form data and "Content-Type" header. This web app is configured on our BIG-IP at the URL <http://10.0.0.21/ilxlab2/>. Now lets look at the web app in the browser. Here is the example of the web form:

iRules LX JSON Parser

Put your JSON text here:

```
{
  "username": "user1",
  "password": "password",
  "email": "nobody@f5.com"
}
```

While we may never have a real use case with JSON in a web form, doing this allows us to use a web browser for the lab rather than having to use command line tools.

Without modifying any text in the form, pressing the submit button should result in this:

iRules LX POST Data

The Content-Type header is: **application/json**

This is your JSON data:

```
{
  "username": "user1",
  "password": "password",
  "email": "nobody@f5.com"
}
```

© 1998 – 2016 F5 Networks, Inc. All rights reserved.

Push the back button and modify the JSON text to so that it will be invalid:

Put your JSON text here:

```
{
  "username": "user1",
  "password": "password",
  "email": "nobody@f5.com"
  randomtext
}
```

Then press submit and you should see this:



iRules LX POST Data

Your JSON was not valid: Unexpected token r

© 1998 – 2016 F5 Networks, Inc. All rights reserved.

This error came from our webserver. You can tell this because the webpage has the logo, header and horizontal rules. Later in this lab we will see errors from the BIG-IP that will be text only.

Now, go back to the web form and click refresh to clear the incorrect JSON.

iRules LX Code Update Behavior

Because the way iRules LX transitions new generations of an LX Plugin, when we make changes to the code we will not see these changes in our original browser window because it is using the same TCP connection and is being serviced by the previous version of the LX plugin. To force the BIG-IP to give us the new generation of the LX plugin, we will be running the following TMSH command after every workspace change:

TMSH

```
restart ilx plugin <plugin_name> immediate
```

Error Logs

In lab exercises from here on out we will need to view the output of STDOUT/STDERR of our Node.js processes. By default, it will be directed to the log files `/var/log/ltm`. Starting in version 13.0, we introduced a new feature to iRules LX allowing for STDOUT/STDERR to be sent to a dedicated log file for each Node.js process for an extension within an LX Plugin.

The extension `ilxlab2_ext` of plugin `ilxlab2_pl` is already configured with the following settings so we can make the logs of the lab easier to find -

Setting	New Value	Reason
Concurrency Mode	Single	Keep logs for all connections in a single file.
iRules LX Logging	Checked	Will make extension send logs to dedicated file.

To see these settings for yourself, click on the `ilxlab2_pl` LX plugin and then click on the `ilxlab2_ext` extension. The dedicated log files can be found in the directory `/var/log/ilx/` and will be named in the format `<partition_name>.<plugin_name>.<ext_name><tmid_id_if_dedicated_mode>`. Here are some examples of file names -

```
Common.ilxlab2_pl.ilxlab2_ext
Common.ilxlab99_pl.some_ext0
Common.ilxlab99_pl.some_ext1
```

Exception Handling

Good software development incorporates exception handling into the code. Without it, our programs would simply crash when there is an uncaught exception. On iRules TCL, the TCL interpreter crashes for an uncaught exception, but the worst consequence is that a single client connection is reset.

Because Node.js in iRules LX is external from TMM, a crash is much more serious. Any connection being serviced by that Node.js process will get reset and all state for any outstanding RPC calls will be lost. A crash triggered from a single function call has the potential to reset hundreds or even thousands of connections on the BIG-IP. Also, any new connections that are trying to establish while Node.js is rebooting could also be reset.

Therefore, it is imperative that we learn proper exception handling.

Handle Errors in JavaScript

Right now the LX workspace code does not have any function call that can throw an exception, but we would like to add more functionality to it. Here is the `addMethod` function that we have in the Node.js code:

```

1 ilx.addMethod('jsonParse', function (req, res) {
2   // Extract JSON from POST data
3   var postData = qs.parse(req.params()[0]).JSON;
4
5   // Send data back to TCL
6   res.reply(postData);
7 });

```

All we are doing is extracting the form input box labeled “JSON”. But we would like to insert more data into the JSON that we send to the application. In order to do that, we must first parse the JSON to a JS object, then stringify it again. Go to the *code_instructions* and complete **code step 1** (remember to copy and paste). The ILX `addMethod` code should look like this after you are done (changes are highlighted) -

Code Step 1

```

1 ilx.addMethod('jsonParse', function (req, res) {
2   // Extract JSON from POST data
3   var postData = qs.parse(req.params()[0]).JSON;
4   var jsonData = JSON.parse(postData);
5
6   res.reply(JSON.stringify(jsonData));
7 });

```

Save and reload the workspace. Now submit some invalid JSON in the form like we did earlier. You will see an text only error like this:



There has been an error.

This error is coming from the iRules TCL code in our “catch” of the ILX call. If we look at the logs we will see the following:

```
..code-block:: console
```

```
# tail -1 /var/log/ltm Jul 11 16:02:15 bigip1 err tmm1[14567]: Rule /Common/ilxlab2_pl/json_parse <HTTP_REQUEST_DATA>: Client - 10.0.0. 10, ILX failure: ILX timeout. invoked from within "ILX::call $handle jsonParse [HTTP::payload]" "
```

```
# tail -1 /var/log/ltm Jul 11 16:02:15 bigip1 err tmm1[14567]: Rule /Common/ilxlab2_pl/json_parse <HTTP_REQUEST_DATA>: Client - 10.0.0. 10, ILX failure: ILX timeout. invoked from within "ILX::call $handle jsonParse [HTTP::payload]" "
```

The log file for the extension should have some entries similar to this:

```
# tail -20 /var/log/ilx/Common.ilxlab2_pl.ilxlab2_ext
Jul 11 16:02:12 pid[15201] undefined:5
Jul 11 16:02:12 pid[15201] randomtext
Jul 11 16:02:12 pid[15201] ^
Jul 11 16:02:12 pid[15201] SyntaxError: Unexpected token w
Jul 11 16:02:12 pid[15201] at Object.parse (native)
Jul 11 16:02:12 pid[15201] at Object.jsonParse (/var/sdm/plugin_store/plugins/
↳:Common: ilxlab2_pl_62102_2/extensions/ilxlab2_ext/index.js:13:23)
Jul 11 16:02:12 pid[15201] at ILXClient.<anonymous> (/var/sdm/plugin_store/
↳plugins/:Common: ilxlab2_pl_62102_2/extensions/ilxlab2_ext/node_modules/f5-nodejs/
↳lib/ilx_server.js:100:46)
<-----Rest of output truncated ----->
```

As you can see, our bad JSON threw an exception that crashed the Node.js process which caused an ILX timeout in TCL. This is the stack track for our exception.

To prevent Node.js from crashing we need to put JSON.parse in a try/catch block. Perform code step 2 on the workspace to do this. The Node function should end up like this –

Code Step 2

```
1 ilx.addMethod('jsonParse', function (req, res) {
2   // Extract JSON from POST data
3   var postData = qs.parse(req.params()[0]).JSON;
4   try {
5     var jsonData = JSON.parse(postData);
6   } catch (err) {
7     console.log('Error with JSON.parse: ' + err.message);
8     return; // Stop processing this function
9   }
10
11   res.reply(JSON.stringify(jsonData));
12 });
```

Save and reload the workspace. Now if you try bad JSON again, you will still get the same error on the web browser, but we will not crash the Node.js process. Doing a tail of the log files again, you will see an error message similar to this:

```
Jul 11 16:14:55 pid[15456] Error with JSON.parse: Unexpected token w
```

Note: Try/catch is only for synchronous functions. Most asynchronous functions handle exceptions/errors in the callback function or with event handlers and vary greatly from one module to the next. You will have to consult the documentation for the module you wish to use.

RPC Status Return Value

While try/catch did help to prevent the Node process from crashing, the error the client received does not help them very much. It would be better if we could give some more info to the client via iRules TCL, but

TCL does not know about the issue that happen with Node.js. Therefore, we should return some type of status to TCL if it the RPC to Node fails.

One way we can accomplish this is by the return of multiple values from Node.js. Our first value could be some type of RPC status value (say an RPC error value) and the rest of the value(s) could be our result from the RPC. It is quite common in programming to make an error value would be 0 if everything was okay but would be an integer to indicate a specific error code.

For this next step, we will make changes to both Node and TCL to create the error communication between Node and TMM. Perform code step 3a and 3b on the workspace. This is what the Node method and the TCL `HTTP_REQUEST_DATA` event should look like after you make the changes:

Code Step 3 Node.js

```

1 ilx.addMethod('jsonParse', function (req, res) {
2   // Extract JSON from POST data
3   var postData = qs.parse(req.params()[0]).JSON;
4   try {
5     var jsonData = JSON.parse(postData);
6   } catch (err) {
7     console.log('Error with JSON.parse: ' + err.message);
8     return res.reply(1);
9   }
10
11   res.reply([0, JSON.stringify(jsonData)]);
12 });

```

As you can see in the `res.reply` function, we can return multiple values back to TCL if we put an array as the argument. TCL will then see these values returned as a TCL list.

Code Step 3 TCL

```

1 when HTTP_REQUEST_DATA {
2   # Send data to Node.js
3   set handle [ILX::init "ilxlab2_pl" "ilxlab2_ext"]
4   if {[catch {ILX::call $handle jsonParse [HTTP::payload]} result]} {
5     log local0.error "Client - [IP::client_addr], ILX failure: $result"
6     HTTP::respond 400 content "<html>There has been an error.</html>"
7     return
8   }
9
10  if {[lindex $result 0] > 0} {
11    # What is our error code?
12    switch [lindex $result 0] {
13      1 { set error_msg "Invalid JSON" }
14    }
15    HTTP::respond 400 content "<html>The following error occurred: $error_msg</html>"
16  } else {
17    #Replace Content-Type header and POST payload
18    HTTP::header replace "Content-Type" "application/json"
19    HTTP::payload replace 0 $cl [lindex $result 1]
20  }
21 }

```

Here we are checking the value of index 0 of the TCL list to see if it is greater than zero. Based upon what that value is we can tailor our return message back to the client. What we have done is allowed Node.js to communicate specific errors that we define back to the client. You would never want to send back all errors because stack traces could reveal sensitive data about your iRule.

Save and reload the workspace. Now when you submit invalid JSON in the browser you should see an error like this –

The following error occurred: Invalid JSON

Now that we have the exception handling taken care of, let's add some more functionality to this iRule. We mentioned a little while ago we would like to add some more data to the JSON that gets sent to the server.

Let's say we wanted to insert random data to act as some type of nonce. In code step 4 let's use the crypto module to insert the random text. This code snippet will show what all the node.js code should look like after this step:

Code Step 4

```
1 'use strict'; // Just for best practices
2 // Import modules here
3 var f5 = require('f5-nodejs');
4 var qs = require('querystring');
5 var crypto = require('crypto');
6
7 // Create an ILX server instance
8 var ilx = new f5.ILXServer();
9
10 // This method will transform POST data into JSON
11 ilx.addMethod('jsonParse', function (req, res) {
12     // Extract JSON from POST data
13     var postData = qs.parse(req.params()[0]).JSON;
14     try {
15         var jsonData = JSON.parse(postData);
16     } catch (err) {
17         console.log('Error with JSON.parse: ' + err.message);
18         return res.reply(1);
19     }
20
21     jsonData.token = crypto.randomBytes(8).toString('hex');
22     res.reply([0, JSON.stringify(jsonData)]);
23 });
24
25 ilx.listen();
```

Save and reload the workspace.

Note: This is not really a proper use of a cryptographic nonce, it is just to show how we can extend functionality with Node.js.

Now this time, send valid JSON text via the web form and we should see a result like this:

iRules LX POST Data

The Content-Type header is: **application/json**

This is your JSON data:

```
{
  "username": "user1",
  "password": "password",
  "email": "nobody@f5.com",
  "token": "d9b87ed4cfc950df"
}
```

© 1998 – 2016 F5 Networks, Inc. All rights reserved.

You can see our token has been added to the JSON.

This concludes the exception handling exercise.

Installing Packages with NPM

You can install modules from NPM when you want to get extra functionality that is not provided with the built in Node.js modules. NPM and the active community around it is one of the primary reasons that Node.js was chosen for iRules LX.

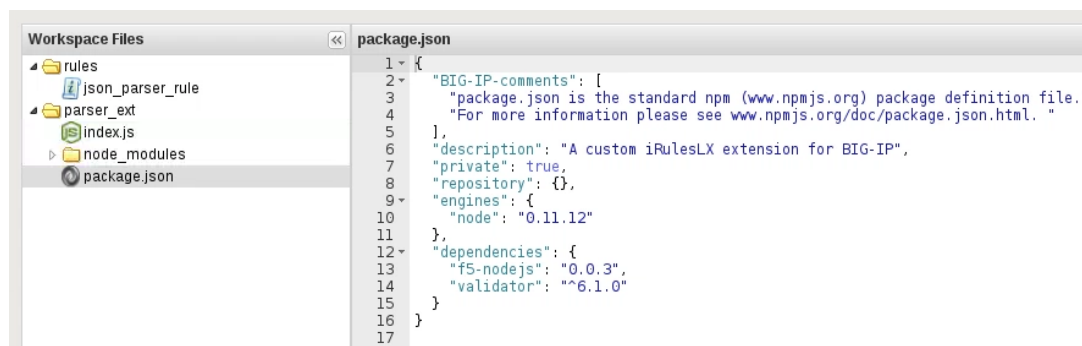
We have a use case requiring us to do syntax validation of an email address that is in the JSON text from a web form. We won't be checking if the email address itself is a working address, just that the syntax is in the correct form. We will download a package from NPM to handle the this.

Installing the Validator Module from NPM

The first thing we must do is install a NPM module for validating email addresses. We will accomplish this with the *validator* module. To install the module into the workspace, we need to access the BASH prompt of our BIG-IP, then `cd` into the workspace directory and run the commands:

```
[root@localhost] # cd /var/ilx/workspaces/Common/ilxlab2/extensions/ilxlab2_ext/
[root@localhost] # npm install validator --save
validator@6.1.0 node_modules/validator
[root@localhost] # ls node_modules/
f5-nodejs  validator
```

The `--save` option saves the module to the `package.json` file dependencies as shown here in the workspace:



Using the Validator Module

To use this module, we must import it into our Node.js code and then call it. In code step 5, we will “require” the module in Node.js, then put some code that will validate if our email address has the proper format. We will also need to add some extra code to TCL to handle 2 more error conditions that email validation brings. The first check ensures that the email value is in our JSON, the second uses the validator module to validate the syntax of the email address. Here is what the code will look like once you are finished:

Code Step 5 Node.js

```

1  'use strict' // Just for best practices
2  // Import modules here
3  var f5 = require('f5-nodejs');
4  var qs = require('querystring');
5  var crypto = require('crypto');
6  var validator = require('validator');
7
8  // Create an ILX server instance
9  var ilx = new f5.ILXServer();
10
11 // This method will transform POST data into JSON
12 ilx.addMethod('jsonParse', function (req, res) {
13   // Extract JSON from POST data
14   var postData = qs.parse(req.params()[0]).JSON;
15   try {
16     var jsonData = JSON.parse(postData);
17   } catch (err) {
18     console.log('Error with JSON.parse: ' + err.message);
19     return res.reply(1);
20   }
21
22   if (!('email' in jsonData)) return res.reply(2); //
23   if (! validator.isEmail(jsonData.email)) return res.reply(3);
24   postData.token = crypto.randomBytes(8).toString('hex')
25   res.reply([0, JSON.stringify(jsonData)]);
26 });
27
28 ilx.listen();

```

You will notice that we check first for the existence of the email property in the JSON and then check if the string in the JSON is valid. If you attempted to only do the email validation but the email property was not present, this would throw an exception for a missing property in the JS object and crash Node.

Code Step 5 TCL

```

1  when HTTP_REQUEST_DATA {
2    # Send data to Node.js
3    set handle [ILX::init "json_parser_pl" "parser_ext"]
4    if {[catch {ILX::call $handle jsonParse [HTTP::payload]} result]} {
5      log local0.error "Client - [IP::client_addr], ILX failure: $result"
6      HTTP::respond 400 content "<html>There has been an error.</html>"
7      return
8    }
9
10   if {[lindex $result 0] > 0} {
11     # What is our error code?
12     switch [lindex $result 0] {
13       1 { set error_msg "Invalid JSON"

```

(continues on next page)

(continued from previous page)

```

14     2 { set error_msg "Property \"email\" missing from JSON."}
15     3 { set error_msg "Property \"email\" not a valid email address."}
16   }
17   HTTP::respond 400 content "<html>The following error occurred: $error_msg</html>"
18 } else {
19   #Replace Content-Type header and POST payload
20   HTTP::header replace "Content-Type" "application/json"
21   HTTP::payload replace 0 $cl [lindex $result 1]
22 }
23 }

```

Both the email property presence check and invalid email error get an error code that we pass over to TCL to give the client a useable error message. Now we can test these error conditions.

Save and reload the workspace. Go to your browser and remove the email property and trailing comma from the password property like so:

iRules LX JSON Parser

Put your JSON text here:

```

{
  "username": "user1",
  "password": "password"
}

```

When you press submit, you should see an error like this:

The following error occurred: Property "email" missing from JSON.

Now go back to the form and refresh the web form back to normal. Now remove the “@” symbol the email address:

iRules LX JSON Parser

Put your JSON text here:

```
{
  "username": "user1",
  "password": "password",
  "email": "nobodyf5.com"
}
```

Submit

Then submit the form and you should see the following:

The following error occurred: Property "email" not a valid email address.

4.3 Asynchronous Programming

In this lab we will demonstrate the concept of asynchronous programming with a LX iRule that will do queries to a MySQL database. For this exercise, we will be using the file `ilxlab3_steps.js` to cut and paste code into the BIG-IP.

4.3.1 Lab 1 - Asynchronous Programming

Test and Review the Existing Configuration

In this lab we will be working with the virtual server (10.0.0.22) & workspace named *ilxlab3*. The plugin and TCL iRule are already assigned to the virtual server. To start off we have a web application that displays a list of users in a database. This web app is configured on our BIG-IP at the URL <http://10.0.0.22/>.

SQL Database Lookup

In this lab we are simply going to view some log statements into the Node.js and look at the order they appear in the log file. First we will review the sql query method in our extension code highlighted below:

```
1 // Add a method
2 ilx.addMethod('get_users', function(req, res) {
3   // Perform the query from pool
4   sqlPool.query(
5     'SELECT id, name, grp FROM users_db.users ORDER BY id;',
6     function(err, rows) {
7       if (err) {
8         // MySQL query failed for some reason, send a 2 back to TCK
9         console.error('Error with query: ', err.message);
10        return res.reply(2);
11      }
12    })
13 }
```

(continues on next page)

(continued from previous page)

```

13     // Check array length from sql
14     if (rows.length)
15         res.reply([0, rows]);
16     else
17         res.reply(1); // if 0 return 1 to the Tcl iRule to show no matching records
18     }
19 );
20 });

```

You will notice that the function has 2 arguments, the first being the text of the actual query. Because this method is asynchronous, the second argument is the callback function that will get executed when the query answer is received by Node.js.

To demonstrate asynchronous behavior, we will put logging statements before and after the query method as such:

Code Step 1

```

1 // Add a method
2 ilx.addMethod('get_users', function(req, res) {
3     // Perform the query from pool
4     console.log('Starting SQL query');
5     sqlPool.query(
6         'SELECT id, name, grp FROM users_db.users ORDER BY id;',
7         function(err, rows) {
8             if (err) {
9                 // MySQL query failed for some reason, send a 2 back to TCK
10                console.error('Error with query: ', err.message);
11                return res.reply(2);
12            }
13            console.log('There are', rows.length, 'records in the DB.');

```

Make sure to use the TMSH plugin restart command after you reload the workspace. Now tail the log contents of the log file with the following BASH command and then refresh the ilxlab3 web page:

```
# tail -f /var/log/ilx/Common.ilxlab3_pl.mysql
```

What do you notice about the order of the log statements?

Now let's make the following changes to the node.js as seen below.

Code Step 2

```

1 // Add a method
2 ilx.addMethod('get_users', function(req, res) {
3     // Perform the query from pool
4     console.log('Starting SQL query');

```

(continues on next page)

(continued from previous page)

```

6      'SELECT id, name, grp FROM users_db.users ORDER BY id;',
7      function(err, rows) {
8          if (err) {
9              // MySQL query failed for some reason, send a 2 back to TCK
10             console.error('Error with query: ', err.message);
11             return res.reply(2);
12         }
13         console.log('There are', rows.length, 'records in the DB.');

```

Use the TMSH plugin restart command after you reload the workspace. Now tail the log contents of the log file again and then refresh the ixlab3 web page. You will see that they are in the right order. The callback function is executed much later because I/O responses take much longer.

But you might ask, how much later is the callback function executing? To answer that question, let's add some more code:

Code Step 3

```

1  // Add a method
2  ilx.addMethod('get_users', function(req, res) {
3      // Perform the query from pool
4      console.log('Starting SQL query');

```

Use the TMSH plugin restart command after you reload the workspace. Now tail the log contents of the log file again and then refresh the ixlab3 web page. Most likely, you are seeing that the time logged is in

the order of tens of milliseconds. As you saw from the I/O time table in the presentation, this is an eternity compared to reads from local memory.

4.4 iRules LX Streaming

In this lab exercise, you will learn how to create LX plugins that can be use in streaming or HTTP mode. In the interest of time, we will taking existing workspaces then and take the code to a full working configuration on a virtual server. We will be using the virtual server `ilxlab4_stream_vs` (10.0.0.23).

4.4.1 Lab 1 - iRules LX Streaming

Creating and Implementing a Streaming LX Plugin

In this lab we will be loading an LX plugin in streaming mode. To keep the lab simple, we will only be loading a plugin that will print the client data to hexdump format in the log files.

Review the LX Workspace and Install NPM package

The first thing we need to do is view the LX Workspace. On the desktop, navigate over to the LX workspaces menu in the tab located at *Local Traffic > iRules > LX Workspaces*. Then click the workspace named *ilxlab4_stream*. You should see an extension named hexdump, then click on the *index.js* file. Also, we should note that you will not see a TCL rule in the workspace.

Just for reference, here is the Node.js code below:

```

1  'use strict';
2  // Hexdump all client data to stdout on L4 virtual server
3  var f5 = require('f5-nodejs');
4  var plugin = new f5.ILXPlugin();
5  var hexy = require('hexy');
6
7  // Register a listener for the client ILXPlugin "connect" event
8  plugin.on('connect', function(flow) {
9    // Register a listener for the ILXStream "data" event
10   flow.client.on('data', function (data) {
11     console.log(hexy.hexy(data)); //Print the client data to STDOUT
12     flow.server.write(data); //Pass the client data to the server stream
13   })
14
15   // Create event listeners for error events
16   flow.client.on('error', function(errorText) {
17     console.log('client error event: ' + errorText);
18   });
19   flow.server.on('error', function(errorText) {
20     console.log('server error event: ' + errorText);
21   });
22   flow.on('error', function(errorText) {
23     console.log('flow error event: ' + errorText);
24   });
25 });
26
27 // Tell TMM not to send data from server to Node

```

(continues on next page)

(continued from previous page)

```
28 var options = new f5.IRXPluginOptions();
29 options.handleServerData = false;
30 plugin.start(options); //Start the plugin in streaming mode
```

As you can see from the code above we are loading the hexy package for doing the hexdumps of the buffer chunk. Therefore, we need to install this package into the workspace. To do this you will need to SSH to the BIG-IP and execute the following commands from the BASH prompt:

```
# cd /var/ilx/workspaces/Common/ilxlab4_stream/extensions/hexdump/
# npm install --save hexy
```

Create the LX Plugin

With our code already in a workspace, you will need to navigate over to the LX Plugins menu in the tab located at *Local Traffic > iRules > LX Plugins*. Click the *Create* button, name the plugin *ilxlab4_stream_pl*, select the *ilxlab4_stream* workspace and click finish to save the changes.

We still need to configure a few more things so once you are back to the LX Plugin list, click on the *ilxlab4_stream_pl* plugin and then click on the *hexdump* extension. Change the following settings:

Setting	New Value	Reason
Concurrency Mode	Single	Keep logs for all connections in a single file.
iRules LX Logging	Checked	Will make extension send logs to dedicated file.

Create the iRules LX Profile

Since iRules LX Streaming does not require the use of TCL iRules, we need a method to associate an LX Plugin to a virtual server. That is done with an iRules LX profile. To create a new iRules LX profile, navigate to the menu *Local Traffic > Profiles > Other > iRules LX* and click the + sign.

Name the new profile *ilxlab4_stream_profile*, select the *ilxlab4_stream_pl* LX Plugin and click finish to save the changes.

Assign the iRules LX Profile to Virtual Servers

Now we need to attach our profile to a virtual server. Go into the virtual server *ilxlab4_stream_vs* main configuration “properties” window (not the resources tab), then expand the Configuration menu to the advanced setting and you will see the iRules LX Profile setting as shown here:

Configuration: Advanced ▾	
Protocol	TCP ▾
Protocol Profile (Client)	tcp ▾
Protocol Profile (Server)	(Use Client Profile) ▾
HTTP Profile	None ▾
HTTP Proxy Connect Profile	None ▾
Traffic Acceleration Profile	None ▾
FTP Profile	None ▾
RTSP Profile	None ▾
SOCKS Profile	None ▾
Stream Profile	None ▾
iRules LX Profile	None ▾
XML Profile	None ▾
MQTT	<input type="checkbox"/>

Select the *ilxlab4_stream_profile* then click update at the bottom to save the changes.

Test the ILX Streaming Plugin

Now we should be able to see the hexdumps in the log file. First, in an SSH session with the BIG-IP, tail the log file of the plugin with the following command:

```
# tail -f /var/log/ilx/Common/ilxlab4_stream_pl.hexdump
```

Then refresh the page in the browser (URL <http://10.0.0.23/ilxlab4stream>) and you should see output like this in the SSH terminal:

```

May 10 15:09:33 pid[16974] 00000000: 4745 5420 2f20 4854 5450 2f31 2e31 0d0a GET./.HTTP/1.1..
May 10 15:09:33 pid[16974] 00000010: 486f 7374 3a20 3130 2e30 2e30 2e32 300d Host:.10.0.0.20.
May 10 15:09:33 pid[16974] 00000020: 0a55 7365 722d 4167 656e 743a 204d 6f7a .User-Agent:.Moz
May 10 15:09:33 pid[16974] 00000030: 696c 6c61 2f35 2e30 2028 5831 313b 204c illa/5.0.(X11;.L
May 10 15:09:33 pid[16974] 00000040: 696e 7578 2078 3836 5f36 343b 2072 763a inux.x86_64;.rv:
May 10 15:09:33 pid[16974] 00000050: 3532 2e30 2920 4765 636b 6f2f 3230 3130 52.0).Gecko/2010
May 10 15:09:33 pid[16974] 00000060: 3031 3031 2046 6972 6566 6f78 2f35 322e 0101.Firefox/52.
May 10 15:09:33 pid[16974] 00000070: 300d 0a41 6363 6570 743a 2074 6578 742f 0..Accept:.text/
May 10 15:09:33 pid[16974] 00000080: 6874 6d6c 2c61 7070 6c69 6361 7469 6f6e html,application
May 10 15:09:33 pid[16974] 00000090: 2f78 6874 6d6c 2b78 6d6c 2c61 7070 6c69 /xhtml+xml,appli
May 10 15:09:33 pid[16974] 000000a0: 6361 7469 6f6e 2f78 6d6c 3b71 3d30 2e39 cation/xml;q=0.9
May 10 15:09:33 pid[16974] 000000b0: 2c2a 2f2a 3b71 3d30 2e38 0d0a 4163 6365 ,*/*;q=0.8..Acce
May 10 15:09:33 pid[16974] 000000c0: 7074 2d4c 616e 6775 6167 653a 2065 6e2d pt-Language:.en-
May 10 15:09:33 pid[16974] 000000d0: 5553 2c65 6e3b 713d 302e 350d 0a41 6363 US,en;q=0.5..Acc
May 10 15:09:33 pid[16974] 000000e0: 6570 742d 456e 636f 6469 6e67 3a20 677a ept-Encoding:.gz
May 10 15:09:33 pid[16974] 000000f0: 6970 2c20 6465 666c 6174 650d 0a43 6f6e ip,.deflate..Con
May 10 15:09:33 pid[16974] 00000100: 6e65 6374 696f 6e3a 206b 6565 702d 616c nection:.keep-al
May 10 15:09:33 pid[16974] 00000110: 6976 650d 0a55 7067 7261 6465 2d49 6e73 ive..Upgrade-Ins
May 10 15:09:33 pid[16974] 00000120: 6563 7572 652d 5265 7175 6573 7473 3a20 ecore-Requests:.
May 10 15:09:33 pid[16974] 00000130: 310d 0a49 662d 4d6f 6469 6669 6564 2d53 1..If-Modified-S
May 10 15:09:33 pid[16974] 00000140: 696e 6365 3a20 4d6f 6e2c 2030 3820 4d61 ince:.Mon,.08.Ma
May 10 15:09:33 pid[16974] 00000150: 7920 3230 3137 2032 323a 3234 3a30 3020 y.2017.22:24:00.
May 10 15:09:33 pid[16974] 00000160: 474d 540d 0a49 662d 4e6f 6e65 2d4d 6174 GMT..If-None-Mat
May 10 15:09:33 pid[16974] 00000170: 6368 3a20 572f 2235 3931 3066 3030 302d ch:.W/"5910f000-
May 10 15:09:33 pid[16974] 00000180: 3236 3222 0d0a 4361 6368 652d 436f 6e74 262"..Cache-Cont
May 10 15:09:33 pid[16974] 00000190: 726f 6c3a 206d 6178 2d61 6765 3d30 0d0a rol:.max-age=0..
May 10 15:09:33 pid[16974] 000001a0: 0d0a ..

```

Create and Implement an HTTP server LX Plugin

In this lab exercise, we will use the LX plugin as an HTTP server. The virtual server that we will use this LX Plugin is the *ilxlab4_http_vs* (10.0.0.24) virtual server which does not have a pool attached to it. This VS does not have an HTTP profile associated with it as use of the iRules LX HTTP server requires this configuration.

Review the LX Workspace

Go to the LX workspace named *ilxlab4_http*, click on the extension folder named *http_server* and click on the *index.js* file. You should see code that looks like this:

```

'use strict';
// Use iRules LX as simple HTTP server
var f5 = require('f5-nodejs');

// Create the HTTP request callback function
function httpRequestCallback(req, res) {
  var msg = '<html><body><h1>ILX HTTP Server</h1>';
  msg += '<p>Welcome client "' + req.client.remoteAddress + '". ';
  msg += 'Your HTTP method is ' + req.method + '.</p>';
  msg += '</body></html>';
  // Set HTTP respond, send reply and close connection.
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end(msg);
}

var plugin = new f5.ILXPlugin();
plugin.startHttpServer(httpRequestCallback);

```

Create the LX Plugin, Profile and Attach to Virtual Server

With our code already in a workspace, all we need to do is create our LX Plugin and iRules LX profile, and attach the profile to the virtual server. Name your LX Plugin *ilxlab4_http_pl*. Create the iRules LX profile with the name of *ilxlab4_http_profile* and attach it to the *ilxlab4_http_vs* virtual server.

Test the ILX HTTP Plugin

In your web browser's 2nd tab type in the URL <http://10.0.0.24>. You should see a web page like this –



ILX HTTP Server

Welcome client "10.0.0.10". Your HTTP method is GET.

