542.1

Introduction and Information Gathering

All reference links are operational in the browser-based delivery of the electronic workbook.

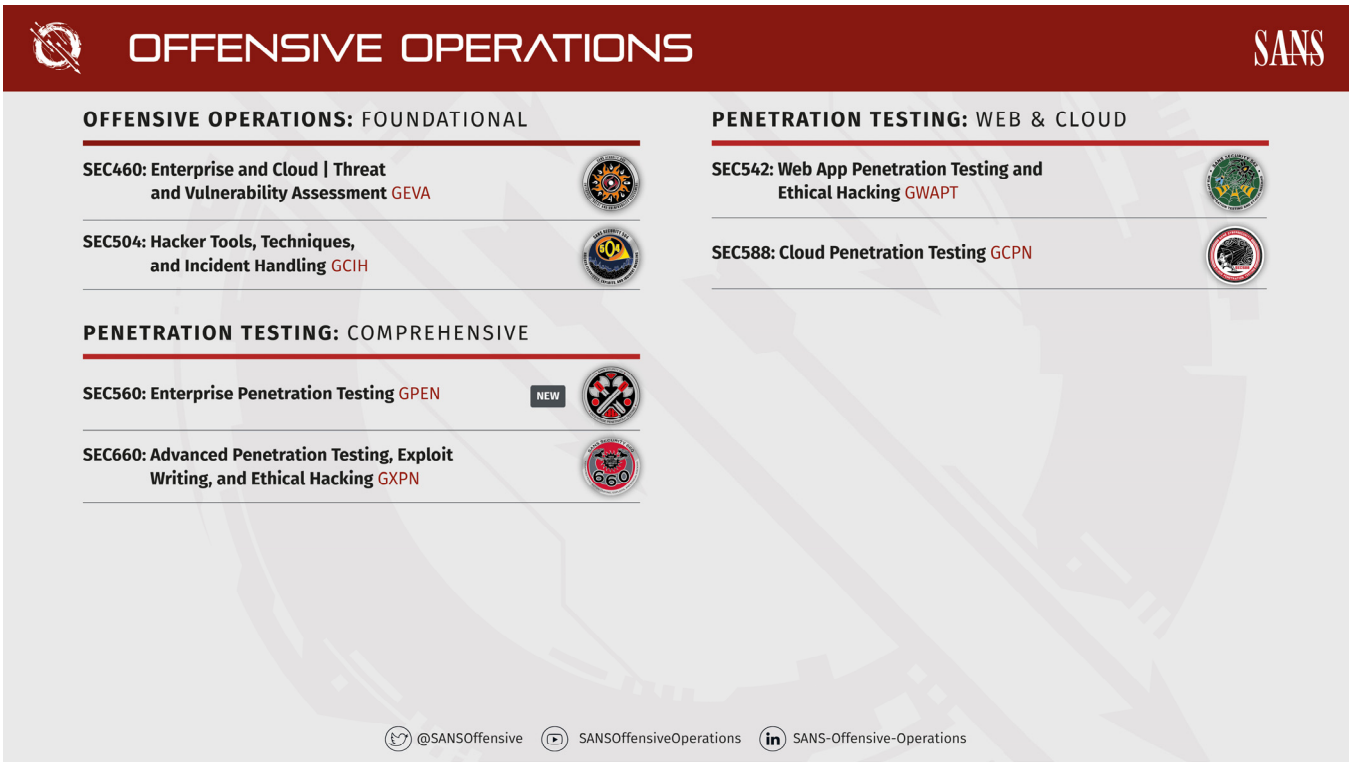
SEC542.1 | Web App Penetration Testing and Ethical Hacking

SANS

Introduction and Information Gathering

Version H01_01

Welcome to SANS Security 542, Section 1!



SANS Offensive Operations leverages the vast experience of our esteemed faculty to produce the most thorough, cutting-edge offensive cyber security training content in the world. Our goal is to continually broaden the scope of our offensive-related course offerings to cover every possible attack vector.

SEC460: Enterprise and Cloud - Threat and Vulnerability Assessment | GEVA | 6 Sections
Learn a holistic vulnerability assessment methodology while focusing on challenges faced in a large enterprise and practice on a full-scale enterprise range.

SEC504: Hacker Tools, Techniques, and Incident Handling | GCIH | 6 Sections
Learn how attackers scan, exploit, pivot, and establish persistence in cloud and conventional systems, and conduct incident response investigations to boost your career.

SEC560: Enterprise Penetration Testing | GPEN | 6 Sections
SANS flagship penetration testing course fully equips you to plan, prepare, and execute a pen test in a modern enterprise.

SEC542: Web App Penetration Testing and Ethical Hacking | GWAPT | 6 Sections
Through detailed, hands-on exercises you will learn the four-step process for web app pen testing, inject SQL into back-end databases, and utilize cross-site scripting attacks to dominate a target infrastructure.

SEC588: Cloud Penetration Testing | GCPN | 6 Sections
The latest in cloud-focused penetration testing subject matter including cloud-based microservices, in-memory data stores, serverless functions, Kubernetes meshes, as well as pen testing tactics for AWS and Azure.

SEC660: Advanced Penetration Testing, Exploit Writing, and Ethical Hacking | GXPN | 6 Sections
This course goes far beyond simple scanning and teaches you how to model the abilities of an advanced attacker, providing you with in-depth knowledge of the most prominent and powerful attack vectors in an environment with numerous hands-on scenarios.

For more information visit **sans.org/offensive-operations**.

SEC467: Social Engineering for Security Professionals | 2 Sections
In this course, you will learn how to perform recon on targets using a wide variety of sites and tools, create and track phishing campaigns, and develop media payloads that effectively demonstrate compromise scenarios.

SEC550: Cyber Deception - Attack Detection, Disruption and Active Defense | 6 Sections
Learn the principles of cyber deception, enabling you to plan and implement campaigns to fit virtually any environment, making it so attackers need to be perfect to avoid detection, while you need to be right only once to catch them.

SEC554: Blockchain and Smart Contract Security | 3 Sections
This course takes a detailed look at the cryptography and transactions behind blockchain and provides the hands-on training and tools to deploy, audit, scan, and exploit blockchain and smart contract assets.

SEC556: IoT Penetration Testing | 3 Sections
Build the vital skills needed to identify, assess, and exploit basic and complex security mechanisms in IoT devices with tools and hands-on techniques necessary to evaluate the ever-expanding IoT attack surface.

SEC565: Red Team Operations and Adversary Emulation | 6 Sections
Learn how to plan and execute end-to-end Red Teaming engagements that leverage adversary emulation, including the skills to organize a Red Team, consume threat intelligence to map and emulate adversary TTPs, then report and analyze the results of the engagement.

SEC575: Mobile Device Security and Ethical Hacking | GMOB | 6 Sections
You will learn how to pen test the biggest attack surface in your organization, mobile devices. Deep dive into evaluating mobile apps and operating systems and their associated infrastructure to better defend your organization against the onslaught of mobile device attacks.

SEC580: Metasploit for Enterprise Penetration Testing | 2 Sections
Gain an in-depth understanding of the Metasploit Framework far beyond how to exploit a remote system. You'll also explore exploitation, post-exploitation reconnaissance, token manipulation, spear-phishing attacks, and the rich feature set of the customized shell environment, Meterpreter.

SEC599: Defeating Advanced Adversaries - Purple Team Tactics & Kill Chain Defenses | GDAT | 6 Sections
Now, more than ever, a prevent-only strategy is not sufficient. This course will teach you how to implement security controls throughout the different phases of the Cyber Kill Chain and the MITRE ATT&CK framework to prevent, detect, and respond to attacks.

SEC617: Wireless Penetration Testing and Ethical Hacking | GAWN | 6 Sections
In this course, you will learn how to assess, attack, and exploit deficiencies in modern Wi-Fi deployments using WPA2 technology, including sophisticated WPA2-Enterprise networks, then use your understanding of the many weaknesses in Wi-Fi protocols and apply it to modern wireless systems and identify, attack , and exploit Wi-Fi access points.

SEC661: ARM Exploit Development | 2 Sections
This course designed to break down the complexity of exploit development and the difficulties with analyzing software that runs on IoT devices. Students will learn how to interact with software running in ARM environments and write custom exploits against known IoT vulnerabilities.

SEC670: Red Team Operations - Developing Custom Tools for Windows | 6 Sections
You will learn the essential building blocks for developing custom offensive tools through required programming, APIs used, and mitigations for techniques used by real nation-state malware authors covering privilege escalation, persistence, and collection.

SEC699: Purple Team Tactics - Adversary Emulation for Breach Prevention & Detection | 6 Sections
SANS's advanced purple team offering, with a key focus on adversary emulation for data breach prevention and detection. Throughout this course, students will learn how real-life threat actors can be emulated in a realistic enterprise environment, including multiple AD forests, with 60% of hands-on time in labs.

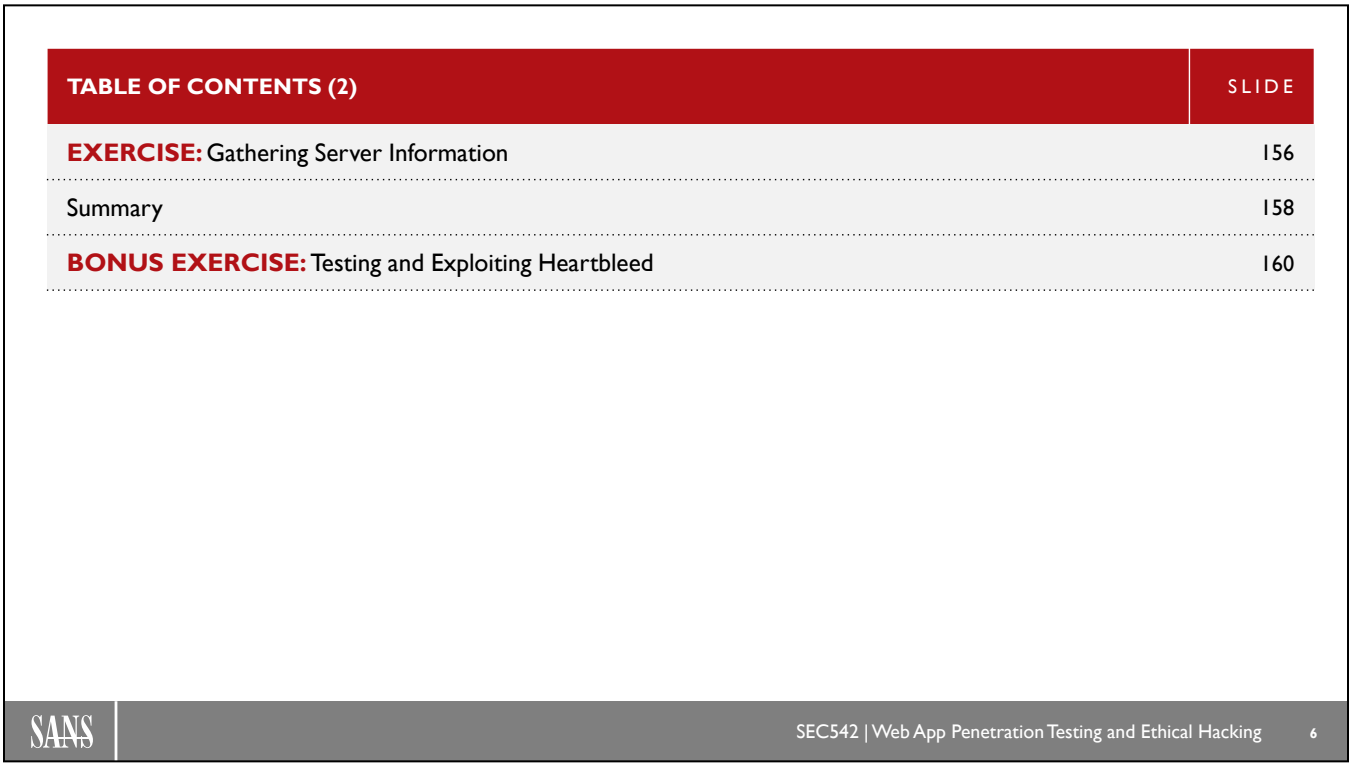
SEC760: Advanced Exploit Development for Penetration Testers | 6 Sections
Learn advanced skills to improve your exploit development and understand vulnerabilities beyond a fundamental level. In this course, you will learn to reverse-engineer 32-bit and 64-bit applications, perform remote user application and kernel debugging, analyze patches for one-day exploits, and write complex exploits against modern operating systems.

For more information visit **sans.org/offensive-operations**.

TABLE OF CONTENTS (1)	SLIDE

542.1 Table of Contents

This table of contents outlines our plan for 542.1.



542.1 Table of Contents

This table of contents outlines our plan for 542.1.

Course Roadmap

- **Section 1: Introduction and Information Gathering**
- Section 2: Content Discovery, Auth, and Session Testing
- Section 3: Injection
- Section 4: XSS, SSRF, and XXE
- Section 5: CSRF, Logic Flaws, and Advanced Tools
- Section 6: Capture the Flag

INTRODUCTION AND INFORMATION GATHERING

1. **Why the Web?**
2. Application Assessment Methodologies
3. Web Application Pen Tester's Toolkit
4. Interception Proxies
5. Exercise: Configuring Interception Proxies
6. Open Source Intelligence (OSINT)
7. Virtual Host Discovery
8. Exercise: Virtual Host Discovery
9. HTTP Syntax and Semantics
10. HTTPS and Testing for Weak Ciphers
11. Exercise: Testing HTTPS
12. Target Profiling
13. Exercise: Gathering Server Information
14. Summary
15. Bonus Exercise: Testing and Exploiting Heartbleed

Course Roadmap

Welcome to Security 542: Web App Penetration Testing and Ethical Hacking!

We'll begin with an introduction to the web, talk about information gathering and virtual host discovery, review HTTPS, and then wrap up with target profiling.

Why Are You Here?

Web applications:

- Are ubiquitous across all sizes and types of businesses
- Enable business-critical functionality
- Provide access to sensitive and/or critical data

In spite of the above, web application security is frequently lackluster

Many organizations don't emphasize application security

- Common for developers to be poorly versed in application security
- Typical for security professionals to lack knowledge and skill in this space

Another reason is because attacking web applications is crazy fun

Why Are You Here?

Hopefully you, and your organization, already have developed some answer to the question of why take this course. However, from the authors' perspectives, there are some aspects we would like to point out.

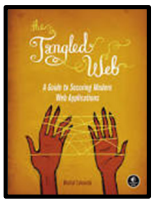
First, as you are no doubt aware, web applications are everywhere; their popularity shows no signs of abating. These applications play increasingly vital roles in business operations. They typically also serve as a primary conduit to business-critical and sensitive information. This data conduit can be abused by adversaries to achieve data exfiltration or perhaps to hold critical data ransom.

In the authors' experience, these highly functional, popular, critical applications unfortunately often don't receive the same level of security attention, testing, or spending as do system and network security initiatives. Perhaps the common lack of knowledge in the application security domain among both developers and security professionals is partially to blame. Thankfully, addressing this shortcoming via courses like this could help partially address this issue.

The final reason we are all here is because attacking web applications is totally fun, intellectually rewarding, and a well-compensated skill in our industry.

Securing the Tangled Web

Michal Zalewski describes the challenge of securing web applications in *The Tangled Web*:



> *"In the traditional model followed by virtually all personal computers over the last 15 years or so, there are very clear boundaries between high-level data objects (documents), user-level code (applications), and the operating system kernel…*
>
> *In the browser world, this separation is virtually nonexistent: Documents and code live as parts of the same intermingled blobs of HTML, isolation between completely unrelated applications is partial at best (with all sites nominally sharing a global JavaScript environment), and many types of interaction between sites are implicitly permitted with few, if any, flexible, browser-level security arbitration frameworks."*[1]

Securing the Tangled Web

The Tangled Web (No Starch Press) is an outstanding book by Michal Zalewski (@lcamtuf). Zalewski is also the author of *Silence on the Wire* and created the tools p0f and American Fuzzy Lop. The book *The Tangled Web* contains a comprehensive overview of the design and security problems associated with the web.

Zalewski describes the history of the web and why old decisions continue to impact the current web, for example:

> *The history of HTTP offers interesting insight into its authors' ambitions and the growing relevance of the Internet. Tim Berners-Lee's earliest 1991 draft of the protocol (HTTP/0.9) was barely one and a half pages long, and it failed to account for even the most intuitive future needs, such as extensibility needed to transmit non-HTML data…*
>
> *Today, all clients and servers support a not-entirely-accurate superset of HTTP/1.0, and most can speak a reasonably complete dialect of HTTP/1.1, with a couple of extensions bolted on. Despite the fact that there is no practical need to do so, several web servers, and all common browsers, also maintain backward compatibility with HTTP/0.9.*[2]

We will discuss the history of HTTP shortly. See https://sec542.com/4f for more information about *The Tangled Web*.

References:

[1] Michal Zalewski, *The Tangled Web: A Guide to Securing Modern Web Applications* (San Francisco: No Starch Press, 2012).

[2] Ibid.

Current Web App Security Testing Is Often Limited

Many enterprises test only the business functionality of applications deployed to the web

- The tide is shifting, but *timely* security testing does not always occur

This lack of testing becomes obvious when you examine the daily reports on vulnerability mailing lists

The NIST National Vulnerability Database includes a large number of vulnerabilities

- The majority of new reports focus on web applications
- Web applications are well represented in the Exploit Database (EDB) as well

Current Web App Security Testing Is Often Limited

Unfortunately, the rush to add features and improve web application capability has led organizations to focus on *business functionality testing*, not *security testing*. Even when security is tested, it's often an afterthought.

If you look at the NIST National Vulnerability Database,[1] you will see that the highest number of reported vulnerabilities are web application related. New web application vulnerabilities are being found and discussed every day on mailing lists such as Full Disclosure and Bugtraq.

Another eye-opening search is one against the Exploit Database,[2] maintained by the folks at Offensive Security, for web application flaws.

References:

[1] NVD – Home, https://sec542.com/6r

[2] Exploit Database, https://sec542.com/6s

Course Roadmap

- **Section 1: Introduction and Information Gathering**
- Section 2: Content Discovery, Auth, and Session Testing
- Section 3: Injection
- Section 4: XSS, SSRF, and XXE
- Section 5: CSRF, Logic Flaws, and Advanced Tools
- Section 6: Capture the Flag

INTRODUCTION AND INFORMATION GATHERING

1. Why the Web?
2. **Application Assessment Methodologies**
3. Web Application Pen Tester's Toolkit
4. Interception Proxies
5. Exercise: Configuring Interception Proxies
6. Open Source Intelligence (OSINT)
7. Virtual Host Discovery
8. Exercise: Virtual Host Discovery
9. HTTP Syntax and Semantics
10. HTTPS and Testing for Weak Ciphers
11. Exercise: Testing HTTPS
12. Target Profiling
13. Exercise: Gathering Server Information
14. Summary
15. Bonus Exercise: Testing and Exploiting Heartbleed

Course Roadmap

Welcome to Security 542: Web App Penetration Testing and Ethical Hacking!

We'll begin with an introduction to the web, talk about information gathering and virtual host discovery, review HTTPS, and then wrap up with target profiling.

Assessing/Improving Application Security

Web application penetration testing provides one method of assessing an application's security posture

- This method will be the focus of this class, but…

Many different viable approaches to assessing application security

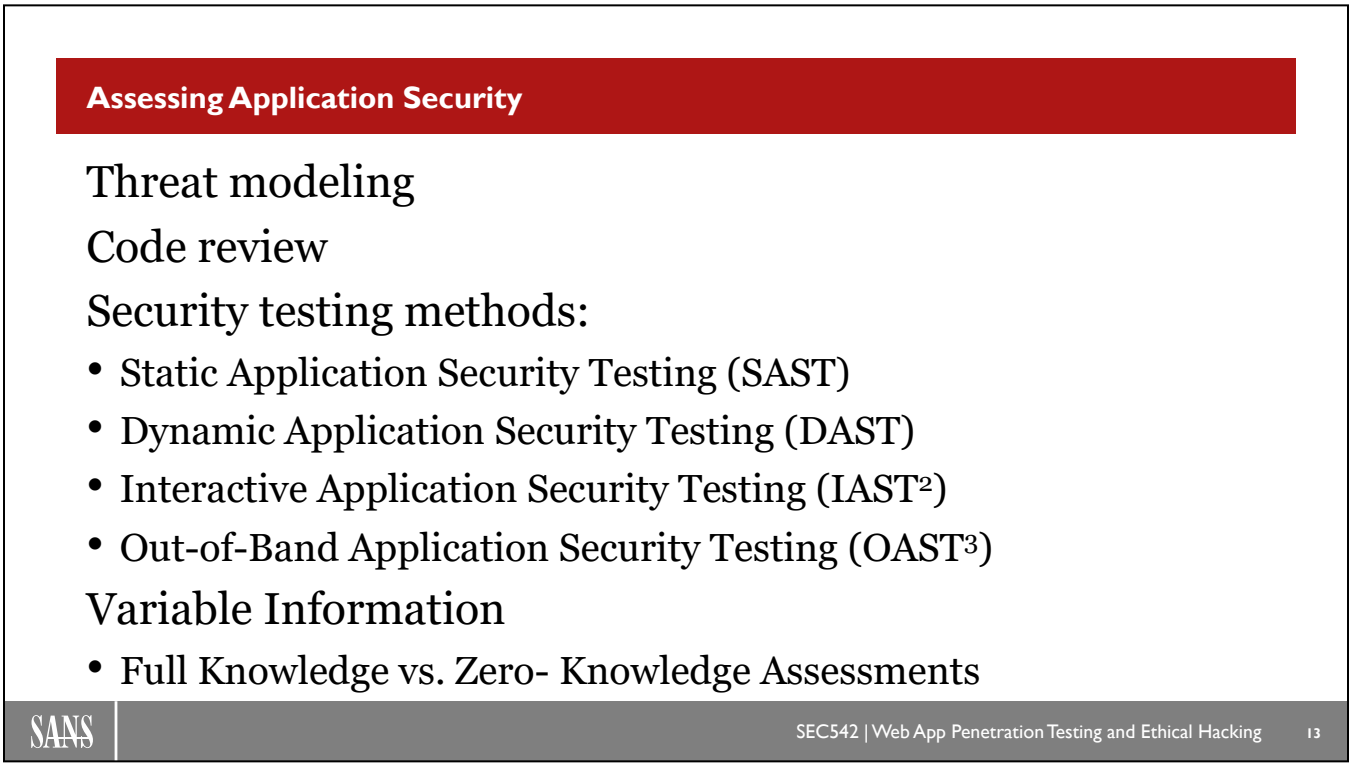
Understanding how web app pen testing fits into the overall spectrum proves important

- Might not be the right or best tool for a given application
- Might need to be complemented by other approaches

Also, varied approaches to web app pen testing itself also exist and could be relevant

Assessing/Improving Application Security

This course will obviously focus on web application penetration testing as a means to assess, with an eye toward improving, an application's security posture. Web application penetration testing is by no means the only approach to assessing application security. Further, it also might not be the best or most effective way to assess an application's security posture. Before focusing exclusively on this one means of assessing security, we will quickly ensure that you have an understanding of how the tool of web application penetration testing fits into the larger application security ecosystem.



Assessing Application Security

Application security assessments can be employed entirely without consideration of live security testing. Techniques such as threat modeling operate more in the application architecture purview. OWASP suggests that fundamentally threat modeling can be considered by answering four questions:

> *What are we building?*
> *What can go wrong?*
> *What are we going to do about that?*
> *Did we do a good enough job?*[1]

Two approaches that will not be immediately discussed in this section are the testing approaches of IAST and OAST. Interactive Application Security Testing (IAST) will be considered during review of Burp's Infiltrator capability.[2] OAST stands for Out-of-band Application Security Testing. This complementary approach to testing will be discussed later during review of Burp Collaborator.[3]

For completeness, RASP, Runtime Application Self-Protection, should also be included in the list, but it will not be discussed in any meaningful way in this course.

References:

[1] OWASP Application Threat Modeling, https://sec542.com/54

[2] Introducing Burp Infiltrator, https://sec542.com/55

[3] OAST (Out-of-band Application Security Testing), https://sec542.com/56

Threat Modeling

Brainstorm potential vulnerabilities and realistic threats to the web application, then think of mitigation strategies for each weakness:

- Consider using NIST 800-30[1], Appendix D, as a resource to assist with threat modeling
- Can help to prioritize limited resources to address the most likely attack vectors and weaknesses in the application

Advantages	Disadvantages
Practical attacker's view of the system	Relatively new technique
Flexible	Good threat models don't automatically mean good software
Early in the SDLC	

Threat Modeling

By understanding what are realistic threats to a system or application, organizations can prioritize remediation efforts to the findings that have the greatest positive impact to the overall security posture. Though threat modeling is only recently popular in association with penetration testing, there are numerous resources available to assist with the process. The Software Engineering Institute (SEI), associated with the Carnegie Mellon University, put together a blog post listing 12 threat modeling methodologies[2].

Chart above from the OWASP Web Security Testing Guide Introduction: https://sec542.com/8k

References:

[1] SP 800-30 Rev. 1, Guide for Conducting Risk Assessments | CSRC https://sec542.com/8l

[2] Threat Modeling: 12 Available Methods https://sec542.com/8m

Source Code Review

Whenever possible, include code review as part of an application's assessment:

- Certain vulnerabilities cannot be found without reading the source code
- The time necessary to develop a suitable exploit can be dramatically reduced
- Usually only performed under a full knowledge pen test, or if a vulnerability permits access to source code within the web root

Advantages	Disadvantages
Completeness and effectiveness	Requires highly skilled security developers
Accuracy	Can miss issues in compiled libraries
Fast (for competent reviewers)	Cannot detect run-time errors easily
	The source code actually deployed might differ from the one being analyzed

Source Code Review

Source code review by qualified individuals can find vulnerabilities that are difficult, or impossible, to find by performing dynamic pen test assessments. Branching within the application that may trigger on parameters or values that would not be seen during normal use. Common issues like "concurrency problems, flawed business logic, access control problems, and cryptographic weaknesses as well as backdoors, Trojans, Easter eggs, time bombs, logic bombs, and other forms of malicious code"[1] may only be able to be identified through source code review.

The assessor should take care to verify that the reviewed source is the same code that is used for the application.

Chart above from the OWASP Web Security Testing Guide Introduction: https://sec542.com/8d

Resource:

[1] Testing for Authorization - OWASP https://sec542.com/8d

Static Application Security Testing (SAST)

SAST involves scrutinizing application source code looking for security deficiencies

Employs tools rather than relying simply on manual code review

- However, SAST can be seen simply as a more automated or efficient type of code review
- Characterized as a full knowledge testing technique due to source code access being required for this type of testing

Strengths: Identifies security deficiencies not readily apparent in deployed application

Weaknesses: Requires access to source code; might overlook APIs or libraries leveraged by the application; overlooks ops side of apps

Static Application Security Testing (SAST)

Performing code review has long been considered a solid practice for reducing bugs/defects and improving applications. Security vulnerabilities can rightly be thought of as a type of defect; however, security flaws are much less likely to have overt stakeholder impact during normal use of the application when compared to more classic bugs or defects. Developers performing code review can still be a good starting point for reducing security flaws in an application.

The most significant shortcoming of manual code review is in the *manual* part. Manual reviews, rather than automated reviews, typically have a higher impact on developer productivity, are only as capable at finding security defects as are the reviewers, and, most importantly, are slow and difficult to scale. Enter automated code review or static analysis tools. Static Application Security Testing (SAST) attempts to automate code review to both decrease the impact on human resources performing reviews and allow for more consistent testing mechanisms.

Although they have significant benefits, SAST tools also have shortcomings. Not all application security flaws are able to be easily identified with an automated analysis tool. This is particularly true of more complex flaws. The SAST tools, being dependent on source code, also have difficulty when applications have flaws due to leveraging external libraries or APIs to which the SAST tools have no access.

Dynamic Application Security Testing (DAST)

The bulk of the focus in this class is on finding vulnerabilities within running applications, also known as penetration testing:

- Commonly used as the sole way to find vulnerabilities in applications
- Pen tests are not always the most efficient method of finding security weaknesses
  - Certain classes of vulnerabilities (i.e., logic flaws) are much easier to detect in a DAST test

Advantages	Disadvantages
Can be automated and fast (and therefore cheap)	Too late in the SDLC
Requires a relatively lower skill set than source code review	Front impact testing only
Tests the code that is actually being exposed	

Dynamic Application Security Testing (DAST)

Penetration testing is the traditional way to find vulnerabilities within web applications, in part due to network-based vulnerability identification that has been rooted in similar activity. While penetration testing can be an effective method to identify vulnerabilities, it requires that the portions of the application that are being tested to be developed. This condition means that the review is fairly late in the Software Development Lifecycle (SDLC) and tends to make remediation efforts more costly than if vulnerabilities were found earlier in the SDLC through manual inspections and source code review.

Chart above from the OWASP Web Security Testing Guide Introduction - OWASP https://sec542.com/8o

DAST and Push Button Pen Testing

DAST tools play an absolutely significant role in web application penetration testing

- However, the ease with which a button can be pushed belies the challenges in most effectively using these tools

Wielding DAST tools effectively proves much more challenging than merely pushing a button; at a minimum it requires:

- Properly configuring scans for the target application
- Guiding scans to ensure comprehensive review
- Following up on results to reduce false positives
- Assessing true positives to understand potential impact
- Understanding and shoring up deficiencies in capabilities

DAST and Push Button Pen Testing

Yes, pushing a button will kick off a scan, but the importance of the person on the other end configuring and guiding the scan, following up on results, and reducing false positives cannot be overstated. Acquiring tools, whether free, open source, or commercial, often proves an easy task for organizations. Most organizations will already have some form of DAST tool at their disposal. However, merely having the tool and being able to point and shoot it at an application rarely yields significant and highly actionable results.

DAST tools, robust as they have grown over the years, still require skill to be able to wield them effectively for improving application security posture. The seemingly simple and obvious task of being able to configure scans according to the target applications does require skill. Even more ability is needed to handle the more challenging tasks of reviewing, assessing, and perhaps exploiting flaws uncovered by the scanner. DAST tools require substantial care and feeding to ensure their efficacy.

Manual Inspections and Reviews

Manual inspections are human reviews that typically test the security implications of people, policies, and processes – WSTG v4.2, p. 18

- Involves reviewing documentation, as well as interviewing designers and system owners
- To ensure accuracy, be sure to verify the documentation and information related in interviews are correct

Advantages	Disadvantages
Requires no supporting technology	Can be time consuming
Can be applied to a variety of situations	Supporting material not always available
Flexible	Requires significant human thought and skill to be effective
Promotes teamwork	
Early in the SDLC	

Manual Inspections and Reviews

When the pen tester can review documentation about the target application and systems, it can speed up the testing process in several ways.

- Automated tools can be tuned to only target technologies present in the application. When millions, or even tens of millions, of requests are necessary to adequately fuzz all the inputs in a large application, limiting the set of fuzzing values can save considerable time throughout the testing window. Only sending SQLi values specific to the backend DBMS used by the application or excluding IIS-specific exploits when the webserver is running Apache increases efficiency.
- The documentation may help the pen tester develop exploits in less time. Some may see this as "cheating", but real-world attackers may have months or years to figure out how to exploit a vulnerability and pen testers usually only have a week or two to complete the entire assessment.
- Much of the guesswork associated with how the components of the application interact can be eliminated.

Documentation tends to become outdated over time. Occasionally, misunderstandings about how the application is put together can lead to inaccuracies. Pen testers should verify the details through their interactions with the system or through interviews with the correct support and development personnel.

As part of an organization's Secure Software Development Lifecycle (Secure SDLC), it is prudent for qualified assessors to review an application's documentation early in the development process. The earlier in the process that architectural flaws, outdated components, or insecure dependencies can be identified, the less work will be required to remediate these issues later in the application's lifespan.

Chart above from the OWASP Web Security Testing Guide Introduction: https://sec542.com/8j

Automated + Manual = Best

- Automated scanning of the applications will quickly test each input within an application:
  - Delivers a very deep investigation by fuzzing each of the inputs for a wide variety of vulnerabilities
  - Scanners tend to find false positives, each finding must be verified
- Manual assessment activities are slow and can be tedious, though they facilitate high-quality pen tests:
  - Provides a broad overview of the application to help the pen tester understand the application's features and capabilities
  - Some categories of vulnerabilities are difficult, if not impossible, to find through automation: CSRF and logic flaws
  - Enables tuning of the automated systems by focusing only on relevant technology and reducing the volume of requests
- *The best of both methods can be realized when combined to complement each other*

Automated + Manual = Best

Automated scans are necessary to test the multitude of inputs for the common vulnerabilities found in web applications. For large applications, a human tester would not be able to perform the tests in a reasonable amount of time, nor in a reliable way. While automated tools are necessary, they are prone to detecting false-positive conditions, where a vulnerability is detected, but the input is not really susceptible to the weakness. Automated scan results should never be sent to developers or IT professionals for remediation without first determining the vulnerability really exists, and, ideally, demonstrating the impact of successful exploitation.

One significant differentiator between a mediocre web application assessment and a high-quality one is that great penetration tests involve manual testing activity. Only manual testing can identify hard to identify vulnerabilities like CSRF and logic flaws. Manual assessment can also find places where automated scans may have missed portions of an application and help the pen tester to tune scanners to be more efficient.

Combining automated and manual techniques through the penetration test tends to lead to the most thorough assessments, often finding vulnerabilities that have been missed by previous testing activity.

More is Definitely Better

- Zero-knowledge represents an assessment in which the pen tester has very few details, perhaps only a URL, about the target application
  - Difficult and time consuming (i.e., expensive) to perform
  - Many people mistakenly think this form of test represents a "real-world" scenario
- Full-knowledge testing involves the pen tester having access to all details about the application, such as architecture drawings, source code, developers, etc.
  - Common with in-house security teams
  - Try to shift assessments as close to a full knowledge assessment as possible
- Real-world tests fall somewhere between these two spectrums

More is Definitely Better

Pen testers are occasionally presented with the myth that a zero-knowledge penetration test, one in which the pen tester has little to no information about the target, is representative of a real-world attacker. Those under the misconception that real-world attackers do not have access to internal information want pen testers to "hack them like the bad guys do." Real-world attackers usually have more than a week or two to attack the systems, providing ample time to map the technologies in use and the system's configuration. Additionally, real-world attackers are not limited to pure web-application penetration testing techniques, and they can employ social engineering or other attacks to gain access to system details.

Without access to system configuration and source code, pen testers activities are inefficient, making certain attacks nearly impossible (we will explore an example, XXE, in Section 4), and others very slow to demonstrate the associated impacts (think LFI or SQLi, which we will see in Section 3).

Pen testers should educate stakeholders about the situation surrounding the amount of information fed into the testing process and try to steer the assessment as close to a full knowledge test as possible.

Most assessments will fall somewhere in-between. At a minimum, pen testers will want the URLs to the application and a set of test accounts. However, it is not uncommon to also collect information about the server-side and client-side technologies, as well as various features found within the application.

Methodology

Several frameworks exist, two that are well suited for web app testing are:

- OWASP Web Security Testing Guide (WSTG)[1]
- Penetration Testing Execution Standard (PTES)[2]

A testing methodology helps to ensure that pen tests are:

- Consistent
- Reproducible
- Rigorous
- Under quality control

Testing Techniques

WSTG	PTES
Manual Inspections and Reviews	Pre-Engagement
Threat Modeling	Intelligence Gathering
Source Code Review	Threat Modeling
Penetration Testing	Vulnerability Analysis
	Exploitation
	Post Exploitation
	Reporting

Methodology

A penetration testing framework can help to ensure that the same set of applicable tests are run across multiple web application assessments. Following a framework ensures that another pen tester could follow in the footsteps of the original assessor and find a very similar set of findings (assuming each tester has a similar skill level). If a suitable framework is used, consistency can lead to a thorough test every time. Since the framework can function as a checklist, indeed the WSTG even provides a checklist[3], the pen testing process is auditable.

Web application penetration testing is different than typical penetration testing. Broader penetration testing methodologies separate vulnerability discovery and exploitation into separate phases. A typical service-side penetration testing methodology tends to be organized as follows:

- Reconnaissance
- Scanning
- Vulnerability assessment
- Exploitation
- Post-exploitation
- Reporting

That doesn't work well for us. Assume a penetration tester enters a single quote into a form and receives a SQL error with a 500 response code. Most penetration testers 'go for the kill' immediately, and attempt SQL injection. But we need to return back to the form, and test for other categories of vulnerabilities, rather than move on. Web application penetration testers tend to jump between steps as opportunities arise.

Both frameworks, WSTG and PTES, provide suitable methodologies for delivering consistent, reproducible, rigorous, and quality controlled web application assessments. Keeping consistent with the course's alignment with the OWASP Web Security Testing Guide, this section focuses on the testing techniques of the WSTG.

NOTE: OWASP Web Security Testing Guide (WSTG) is the new designation OWASP is using for this guide; prior to version 4.2, the project was known as the OWASP Testing Guide (OTG).

References:

[1] OWASP Web Security Testing Guide Project (WSTG) https://sec542.com/2i

[2] PTES, https://sec542.com/3d

[3] WSTG Checklist: https://sec542.com/9e

OWASP

OWASP has made a considerable contribution toward improving the state of software security

> *"The Open Web Application Security Project (OWASP) is a 501(c)(3) worldwide not-for-profit charitable organization focused on improving the security of software. Our mission is to make software security visible, so that individuals and organizations are able to make informed decisions. OWASP is in a unique position to provide impartial, practical information about AppSec to individuals, corporations, universities, government agencies and other organizations worldwide. Operating as a community of like-minded professionals, OWASP issues software tools and knowledge-based documentation on application security."*[1]

One of their largest contributions is the OWASP Top 10

OWASP

In addition to the OWASP Top 10 (discussed next), OWASP is known for a large number of high-quality tools and frameworks. Here is list of OWASP Flagship products:[2]

Tools:

- OWASP Zed Attack Proxy
- OWASP Web Testing Environment Project
- OWASP OWTFOWASP Dependency Check

Code:

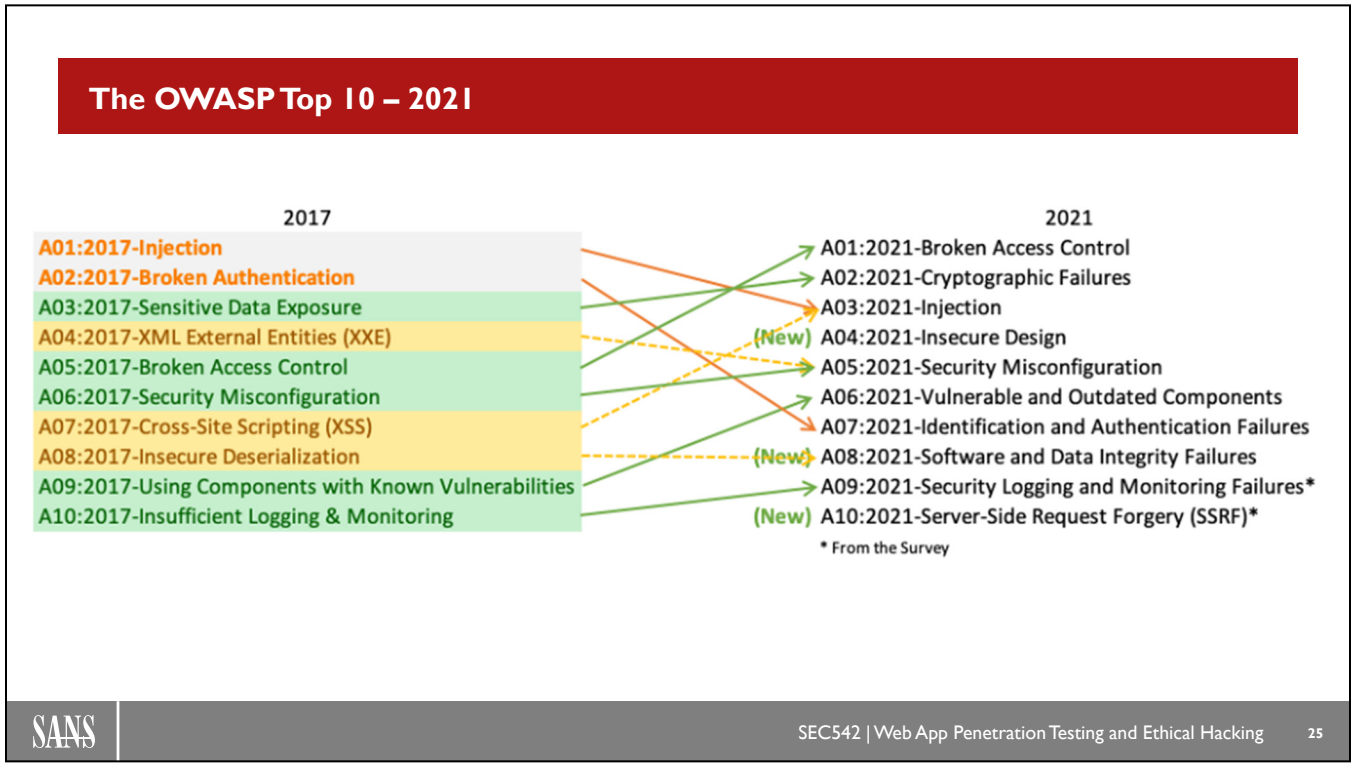
- OWASP ModSecurity Core Rule Set Project
- OWASP CSRFGuard Project
- OWASP AppSensor Project

Documentation:

- OWASP Application Security Verification Standard Project
- OWASP Software Assurance Maturity Model (SAMM)
- OWASP AppSensor Project
- OWASP Top Ten Project
- OWASP Web Security Testing Guide Project[3]

References:

[1] OWASP, http://sec542.com/2f
[2] OWASP Project Inventory, https://sec542.com/2g
[3] Ibid.



The OWASP Top 10 – 2021

The OWASP Top 10 is normally updated every three years. The 2021 version was delayed a year and was finally released in September of 2021. The graphic above compares the 2017 version to the 2021 version. There are three new categories, and many categories were renamed or consolidated.

OWASP described the moves, merges, and changes:

- ***A01:2021-Broken Access Control*** *moves up from the fifth position to the category with the most serious web application security risk; the contributed data indicates that on average, 3.81% of applications tested had one or more Common Weakness Enumerations (CWEs) with more than 318k occurrences of CWEs in this risk category. The 34 CWEs mapped to Broken Access Control had more occurrences in applications than any other category.*
- ***A02:2021-Cryptographic Failures*** *shifts up one position to #2, previously known as A3:2017-Sensitive Data Exposure, which was broad symptom rather than a root cause. The renewed name focuses on failures related to cryptography as it has been implicitly before. This category often leads to sensitive data exposure or system compromise.*
- ***A03:2021-Injection*** *slides down to the third position. 94% of the applications were tested for some form of injection with a max incidence rate of 19%, an average incidence rate of 3.37%, and the 33 CWEs mapped into this category have the second most occurrences in applications with 274k occurrences. Cross-site Scripting is now part of this category in this edition.*
- ***A04:2021-Insecure Design*** *is a new category for 2021, with a focus on risks related to design flaws. If we genuinely want to "move left" as an industry, we need more threat modeling, secure design patterns and principles, and reference architectures. An insecure design cannot be fixed by a perfect implementation as by definition, needed security controls were never created to defend against specific attacks.*
- ***A05:2021-Security Misconfiguration*** *moves up from #6 in the previous edition; 90% of applications were tested for some form of misconfiguration, with an average incidence rate of 4.5%, and over 208k occurrences of CWEs mapped to this risk category. With more shifts into highly configurable software, it's not surprising to see this category move up. The former category for A4:2017-XML External Entities (XXE) is now part of this risk category.*

- ***A07:2021-Identification and Authentication Failures*** *was previously Broken Authentication and is sliding down from the second position, and now includes CWEs that are more related to identification failures. This category is still an integral part of the Top 10, but the increased availability of standardized frameworks seems to be helping.*
- ***A08:2021-Software and Data Integrity Failures*** *is a new category for 2021, focusing on making assumptions related to software updates, critical data, and CI/CD pipelines without verifying integrity. One of the highest weighted impacts from Common Vulnerability and Exposures/Common Vulnerability Scoring System (CVE/CVSS) data mapped to the 10 CWEs in this category. A8:2017-Insecure Deserialization is now a part of this larger category.*
- ***A09:2021-Security Logging and Monitoring Failures*** *was previously A10:2017-Insufficient Logging & Monitoring and is added from the Top 10 community survey (#3), moving up from #10 previously. This category is expanded to include more types of failures, is challenging to test for, and isn't well represented in the CVE/CVSS data. However, failures in this category can directly impact visibility, incident alerting, and forensics.*
- ***A10:2021-Server-Side Request Forgery*** *is added from the Top 10 community survey (#1). The data shows a relatively low incidence rate with above average testing coverage, along with above-average ratings for Exploit and Impact potential. This category represents the scenario where the security community members are telling us this is important, even though it's not illustrated in the data at this time.*[2]

References:

[1] https://sec542.com/a8

[2] Ibid.

OWASP Web Application Penetration Testing Methodology

The OWASP Web Application Security Testing Methodology is fantastic

- Part of the Web Security Testing Guide

It provides a more natural flow for web application penetration testing

- For example, the section "Testing for SQL Injection" (WSTG-INPV-05) describes both testing for and immediately exploiting SQL injection flaws

OWASP Web Application Security Testing Methodology

The OWASP Web Security Testing Guide is available at https://sec542.com/2i.

There is a copy of the guide in /home/student/Documents/ as well as in the root directory of the Security 542 media ISO file. The section "Testing for SQL Injection" (WSTG-INPV-05) begins with vulnerability assessment:

> *The very first test usually consists of adding a single quote (') or a semicolon (;) to the field or parameter under test. The first is used in SQL as a string terminator and, if not filtered by the application, would lead to an incorrect query.*[1]

It then immediately moves to the exploitation phase, which is exactly how most web application penetration testers work:

> *Suppose we insert the following Username and Password values:*
>
> ```
> $username = 1' or '1' = '1
> $password = 1' or '1' = '1
> ```
>
> *The query will be:*
>
> ```
> SELECT * FROM Users WHERE Username='1' OR '1' = '1' AND
> Password='1' OR '1' = '1'
> ```
> [2]

Below we find the various testing categories included in the OWASP Web Security Testing Guide. This represents a fairly comprehensive list for application security testing like we will perform. We will attempt to help convey what is meant by the categories as we delve into them throughout class.

- Information Gathering (INFO)
- Configuration and Deployment Management (CONF)
- Identity Management (IDNT)
- Authentication (ATHN)
- Authorization (ATHZ)
- Session Management (SESS)
- Input Validation (INPV)
- Error Handling (ERRH)
- Cryptography (CRYP)
- Business Logic (BUSL)
- Client Side (CLNT)

Security 542 follows the OWASP Web Application Security Testing Methodology, with some adjustments for course content and flow.

References:

[1] OWASP Web Security Testing Guide Project, https://sec542.com/2i
[2] Ibid.

OWASP Web Security Testing Guide (WSTG)

The OWASP Web Security Testing Guide (WSTG) can serve as a sanity test/checklist[1] for Web App Pen Testing processes

- Located in /home/student/Documents on the Security542 Linux VM
- Also in the root directory of the Security 542 media ISO file

SEC542 will identify relevant Test IDs while progressing through the material

- Providing a simple identifier to allow you (or your clients) to seek additional information about the flaw

Most importantly, the Test IDs provide a jumping-on point for researching how the vulnerable organization can begin to remediate flaws

OWASP Web Security Testing Guide (WSTG)

Although you might not feel like a cool hacker when you reference against the OWASP Web Security Testing Guide (WSTG) as a penetration tester, the overarching goal that drives us should be to improve the target organization's application security posture. With this goal in mind, the WSTG can provide a significant boon to our application penetration testing process.

The content is available in multiple formats: Website[2], Github[3], and PDF[4]. The website allows for easy access to the current, latest, and prior versions of the WSTG. The Github repository hosts the latest, bleeding edge content for the WSTG. The Github site makes available the stable version of the WSTG in PDF format, as well as previous versions.

In particular, two aspects of the WSTG stand out that will improve our process. First, the WSTG can serve as a testing checklist to help ensure appropriate test coverage. Many application penetration testing engagements might not incorporate all elements of the WSTG, but it can provide a starting point for the discussion regarding coverage. The most important benefit from leveraging the WSTG comes from using it as a vehicle to explore, document, and guide target organizations in appropriate remediation activities.

Although SEC542 will not seek to adhere exclusively to the WSTG test items or their particular categorization, when relevant, we will seek to identify WSTG Test IDs throughout the course material.

References:

[1] WSTG Checklist: https://sec542.com/9e

[2] OWASP Web Security Testing Guide Project - Website, https://sec542.com/2i

[3] OWASP Web Security Testing Guide Project – Github Repo, https://sec542.com/8q

[4] OWASP Web Security Testing Guide Project – v4.2 PDF, https://sec542.com/af

And Yet…Not the WSTG

Note: Though the course makes use of and loosely tracks the OWASP Web Security Testing Guide, we reserve the right to and will necessarily deviate from the testing guide where appropriate

- Our flow through the material will quite often not synchronize with the WSTG
- Some elements discussed might not be applicable to the WSTG
- Likewise, some elements of the WSTG will not be reviewed in detail

And Yet…Not the WSTG

Although the OWASP Web Security Testing Guide provides a useful structure and reference point for our material, the course will, by design, frequently depart from the WSTG. Please be aware this is neither an oversight nor an overt criticism of the OWASP Web Security Testing Guide. Our material is built to train students of web application penetration testing, which means we approach the discipline from a different vantage point than does the WSTG. This quite often will impact the ordering of the material being presented. However, it will also entail our including content beyond the explicit scope of the WSTG and/or not including content found within the WSTG.

In short, this is a Public Service Announcement that 542 is not WSTG training and therefore will not be overtly beholden to the guide.

Course Roadmap

- **Section 1: Introduction and Information Gathering**
- Section 2: Content Discovery, Auth, and Session Testing
- Section 3: Injection
- Section 4: XSS, SSRF, and XXE
- Section 5: CSRF, Logic Flaws, and Advanced Tools
- Section 6: Capture the Flag

INTRODUCTION AND INFORMATION GATHERING

1. Why the Web?
2. Application Assessment Methodologies
3. **Web Application Pen Tester's Toolkit**
4. Interception Proxies
5. Exercise: Configuring Interception Proxies
6. Open Source Intelligence (OSINT)
7. Virtual Host Discovery
8. Exercise: Virtual Host Discovery
9. HTTP Syntax and Semantics
10. HTTPS and Testing for Weak Ciphers
11. Exercise: Testing HTTPS
12. Target Profiling
13. Exercise: Gathering Server Information
14. Summary
15. Bonus Exercise: Testing and Exploiting Heartbleed

Course Roadmap

Welcome to Security 542: Web App Penetration Testing and Ethical Hacking!

We'll begin with an introduction to the web, talk about information gathering and virtual host discovery, review HTTPS, and then wrap up with target profiling.

Web Application Pen Tester's Toolkit

- Careful planning should go into building out a web application pen tester's toolkit
- Some key considerations are:
  - Attack platform
  - Dynamic web application security scanner(s)
  - Browser(s)
  - Interception proxies

Web Application Pen Tester's Toolkit

To carry out our previously discussed attack methodology in a highly effective and efficient manner, care should be taken when building a web application penetration tester's toolkit. Though this is not by any means an exhaustive review, there are four key items that should be considered some of the most important elements of the pen tester's toolkit: the attack platform used to launch attacks, the dynamic web application security scanner or scanners, the browsers used for manual inspection and verification, and the interception proxy employed.

Attack Platform

Windows, Linux, macOS – not nearly as important as they are with Network Pen Testing

Prebuilt Linux VMs can decrease time/burden on the frontend

Popular approaches are:

- Kali (kali-linux-web metapackage)
- Security542 – Custom Linux VM built specifically for this course

Attack Platform

Those coming to web application penetration testing from a network penetration testing background will have firsthand knowledge about the importance of selecting the appropriate platform OS. Although the selection of OS in web application penetration testing is important, the importance is not nearly so great as it is to the pure network penetration tester.

In fact, whichever platform you are most comfortable with already would typically be more sufficient for the majority of your web penetration testing activities. Although there might be some tools unavailable to you for your OS of choice, typically the more manual aspects of web penetration testing will all be available on macOS, Linux, and Windows alike. Where you might encounter more difficulty is with the automated scanning tools, which tend to be more OS specific. If you are performing internal web application assessments, then the automated tool would likely perform better installed on a server anyway.

Should you not feel particularly beholden to a specific OS, then using a prebuilt Linux VM could provide a great option. Leveraging a Linux build created specifically with web application penetration testing in mind can reduce the initial time and burden of setting up the attack platform. The following Linux distributions, in particular, stand out in this space:

- **Kali** – Formerly known as BackTrack, Kali is, without question, the most popular penetration testing distribution. With the Kali update, the creators have developed metapackages targeted for specific use cases. Kali-web is one of those packages, which is specifically targeted at web application penetration testers.
- **Security542** – A custom Linux-based VM built specifically for this course. Provides a combination of web app pen testing tools along with vulnerable apps and sites.

If you already have a Kali Linux installation, then simply run the following command to install all the tools that are part of the kali-linux-web metapackage:

apt-get update && apt-get install kali-linux-web

The Kali Linux metapackages are available at https://sec542.com/3c.

DAST: Web Application Security Scanner

- Highly automated, push-button, web application security scanners are a boon to a web application penetration tester
- They can afford us the ability to rapidly scan a vast number of large applications
  - Rapid, relative to how long a manual test would take us to perform
- The highly automated and rapid scanning ability is both a great asset and a weakness
  - Speed often comes at the cost of depth
- Which is the best?
  - Appreciate they excel at automation, but often fail to find more subtle issues
  - Must determine which ones work best for your apps in your organization
- We will necessarily employ a web application security scanner, but know that it is not the only tool we need to employ:
  - ZAP Active Scan
  - Arachni Scanner
  - Nikto
  - Acunetix Vulnerability Scanner
  - BurpSuite Pro Active/Live Scan
  - Fortify WebInspect
  - HCL AppScan
  - Qualys WAS
  - Rapid7 AppSpider
  - Whitehat Sentinel
  - Invivti (Formerly Netsparker)

DAST: Web Application Security Scanner

Automated scanners can provide a huge, and much needed, efficiency gain. Many organizations have a tremendous number of vast web applications. Though a capable penetration test could be more thorough in some respects, and discerning, the volume of interactions required necessitates the use of automated scanning tools.

If you have previous exposure to network vulnerability scanning, using tools such as Nessus, then you might have an inappropriate amount of confidence in these dynamic web vulnerability scanners. The task required of web application scanners is drastically different from that imposed on network vulnerability scanners.

A network vulnerability scanner's primary job is to determine what applications or services are running and whether those items are at the current patch level. Where those items are behind on patch level, the network vulnerability scanner will provide an assessment of the severity of the flaw and recommendations on remediating. Although this approach can be successful against off-the-shelf web applications, such as WordPress, Joomla, phpBB, and SharePoint, it fails when dealing with custom web applications.

In order for a web application security scanner to be effective, it will need to directly interact with the web application and based on that interaction determine whether vulnerabilities exist. Although conceptually simple, in practice, consistently doing this well across the immense diversity of custom applications proves fiendishly difficult. Understand that web application security scanners are not going to ever be nearly as successful at detecting flaws as their network vulnerability scanning brethren.

There are a tremendous number of options in the web application security space. They range from free to amazingly expensive. Deployment options also vary widely, ranging from mobile applications, to local desktop apps, to multi-server, to cloud-based configurations.

Which is the best? None of them are consistently better than the others. Be cautious with the expert comparison reports because they are not representative of your applications and how the tools will perform in your environment.

Browsers

- Not that you should, but we could perform most aspects of a web app pen test with nothing but a browser
- Not trying to start a browser war, but the browser needs to be considered:
  - Don't base browser choice on vanity or the tool's geek cred
  - Choice and configuration of browser could have test implications
- Some thoughts:
  - We need a browser that will not get in the way of our security testing (XSS in particular)
    - Or could be configured to get out of the way
  - Extending the capabilities of the browser via extensions or add-ons could also be advantageous
  - Most important is to know your tool and how it behaves


Browsers

Although choice of browser might seem like little more than preference, most often seemingly based on familiarity or geek credibility, it can have significant implications for our testing. The primary considerations are how extensible the browser is and whether it will get in the way of our testing. With respect to extending the browser capabilities, we are specifically looking for whether there are tools that can make the browser more capable from a penetration-testing standpoint. Chrome and Chromium, through the Web Store, and Firefox, through its Add-Ons marketplace, are both highly extensible.

Chromium is the fully open source version of Chrome (which contains non-open elements). Chromium is an excellent choice for web application penetration testers since it supports (most of the) Chrome functionality but is fully open and more configurable (a penetration tester may disable or change Chromium functionality by altering the source code and recompiling).

The question of ensuring the browser does not get in the way of testing might be less obvious to those new to the discipline. The most common way in which the browser can cause our pen tests harm is by being overly helpful, from a security standpoint, with respect to our XSS PoC (Proof-of-Concept) payloads. Some modern browsers have security capabilities built in that we might need to disable on an as-needed basis. The key regard on this is to know your tools. Browsers are always in a state of flux and seem to be consistently updating their built-in security capabilities, so be mindful on this front.

Course Roadmap

- **Section 1: Introduction and Information Gathering**
- Section 2: Content Discovery, Auth, and Session Testing
- Section 3: Injection
- Section 4: XSS, SSRF, and XXE
- Section 5: CSRF, Logic Flaws, and Advanced Tools
- Section 6: Capture the Flag

INTRODUCTION AND INFORMATION GATHERING

1. Why the Web
2. Application Assessment Methodologies
3. Web Application Pen Tester's Toolkit
4. **Interception Proxies**
5. Exercise: Configuring Interception Proxies
6. Open Source Intelligence (OSINT)
7. Virtual Host Discovery
8. Exercise: Virtual Host Discovery
9. HTTP Syntax and Semantics
10. HTTPS and Testing for Weak Ciphers
11. Exercise: Testing HTTPS
12. Target Profiling
13. Exercise: Gathering Server Information
14. Summary
15. Bonus Exercise: Testing and Exploiting Heartbleed

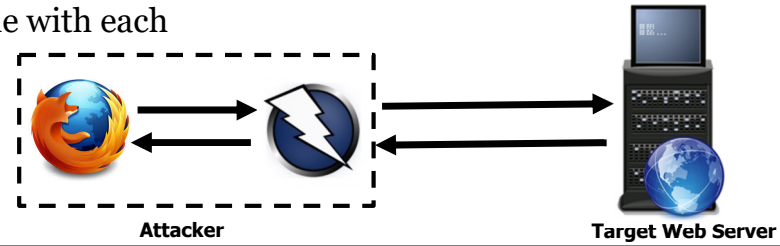
Course Roadmap

Welcome to Security 542: Web App Penetration Testing and Ethical Hacking!

We'll begin with an introduction to the web, talk about information gathering and virtual host discovery, review HTTPS, and then wrap up with target profiling.

Interception Proxies

- On one end of the spectrum we have automated scanners, and on the other we have browsers
- Interception proxies are a major piece of our toolkit, providing a middle ground between the fully automated and the overtly manual
- The interception proxy occupies a primary role in the web penetration tester's arsenal
- ZAP and Burp will be the proxies of choice for the class, and we will spend much time with each



Interception Proxies

One of the most important tools for a web application penetration tester is the interception proxy. This tool will allow us to be more methodical than the simple push-button scanner. The interception proxy will also facilitate much more efficient testing than through just manually leveraging a browser. So, on one end of testing we have the browser, and on the other end we have automated scanners. Quite apropos, the interception proxy sits somewhere between these two ends.

The interception proxy is not your enterprise forward proxy that includes web filtering capabilities. We are talking about a software-based proxy that runs on our own attack machine that we have configured as an intentional man-in-the-middle (MITM).

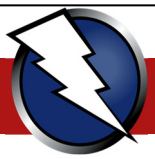
The typical setup involves the proxy software running and listening on a particular port (8080 seems to be de facto for almost all tools), the browser configured to proxy all its traffic to 127.0.0.1:8080, and the target server http://victim.tld. When we put http://victim.tld into the address bar, the browser, configured to proxy, will connect to our interception proxy, which will, in turn, connect to http://victim.tld. The response traffic from the server will come back through the proxy on the way to the server.

The two interception proxies we will be leveraging the most during class are the Zed Attack Proxy (ZAP) and the Burp Suite Professional. Developing significant comfort with and facility in wielding these tools is one goal of the class.

ZAP is available at https://sec542.com/2j.

Burp Suite is available at https://sec542.com/1k.

Interception Proxies: OWASP Zed Attack Proxy (ZAP)

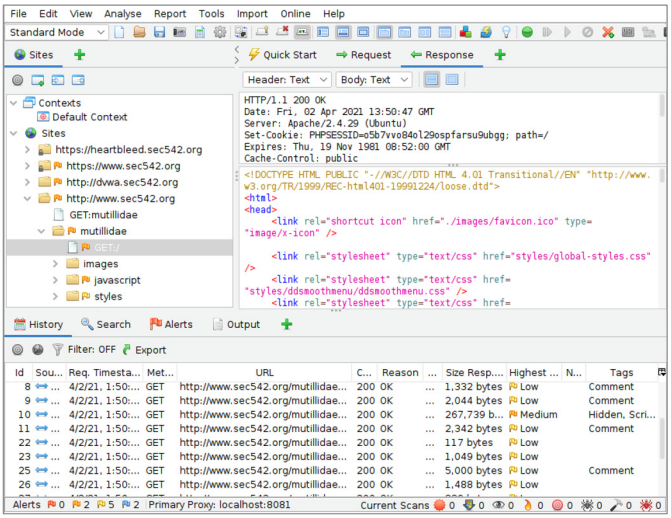


OWASP ZAP (Zed Attack Proxy) is a fully featured open source interception proxy

- Fork of the old (retired) Paros proxy

ZAP Version 2.4 featured a number of significant improvements:

- The interface is far less cluttered
- ZAP's fuzzer is (now) outstanding



Interception Proxies: OWASP Zed Attack Proxy (ZAP)

ZAP was forked from the (now retired) Paros Proxy. ZAP is written in Java, so it runs on just about any operating system.

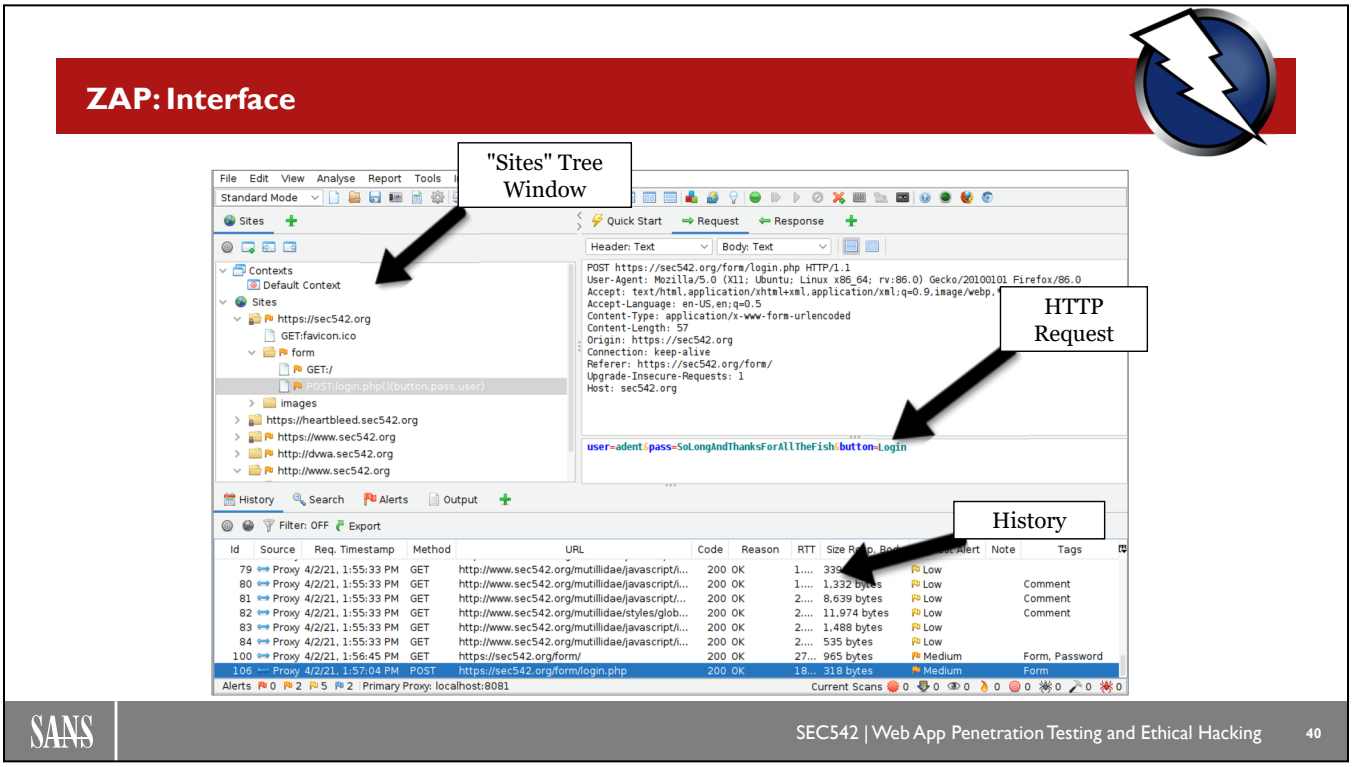
ZAP has long been considered a lesser interception proxy compared to the Burp Suite, but its lack of (comparable) features was mitigated by its price: Free. ZAP has been continually improved and now features commercial-quality features. Most notable is ZAP's fuzzer, which is now quite powerful.

ZAP is an OWASP "Flagship" project: "The OWASP Flagship designation is given to projects that have demonstrated strategic value to OWASP and application security as a whole."[1]

OWASP Zed Attack Proxy (ZAP) is available at https://sec542.com/2j.

Reference:

[1] OWASP Flagship, https://sec542.com/2h



ZAP: Interface

ZAP's help menu describes the ZAP interface; this text is available in your Security542 Linux VM by pressing <F1> in ZAP:

By default, ZAP displays:

- *A top-level menu that gives access to many of the automated and manual tools*
- *A top-level toolbar that includes buttons for commonly used features*
- *A tree window on the left side that displays the Sites tree and the Scripts tree*
- *A workspace window on the top-right side, which enables you to display and edit requests, responses, and scripts*
- *An information window underneath workspace window, which displays details of the automated and manual tools*
- *A footer that displays a summary of the alerts found and the status of the main automated tools*

By default, only a small number of tabs are shown when ZAP starts.

- *Other tabs appear when you run the related tools or can be added manually via the green plus tabs.*
- *You can pin tabs so that they are always shown when you restart ZAP.*[1]

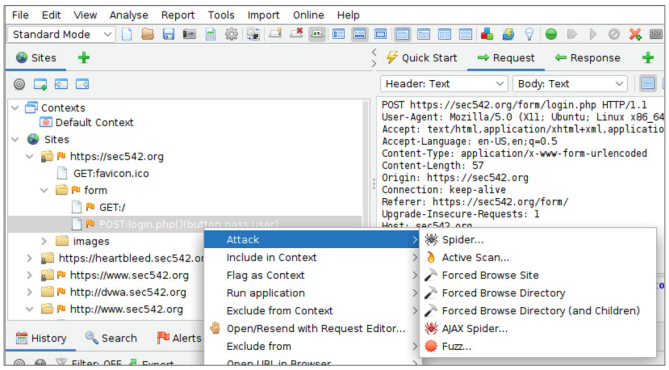
Reference:

[1] zap-core-help, https://sec542.com/1e

ZAP: Attack Menu

ZAP version 2.4+ consolidated a number of functions into the "Attack" menu:

- Active scan: Actively scans a site, searching for vulnerabilities (ZAP's passive scanner is always running)
- Forced browsing (DirBuster-style functionality)
- Spider and AJAX spider (see notes for more information)
- Fuzzing



ZAP: Attack Menu

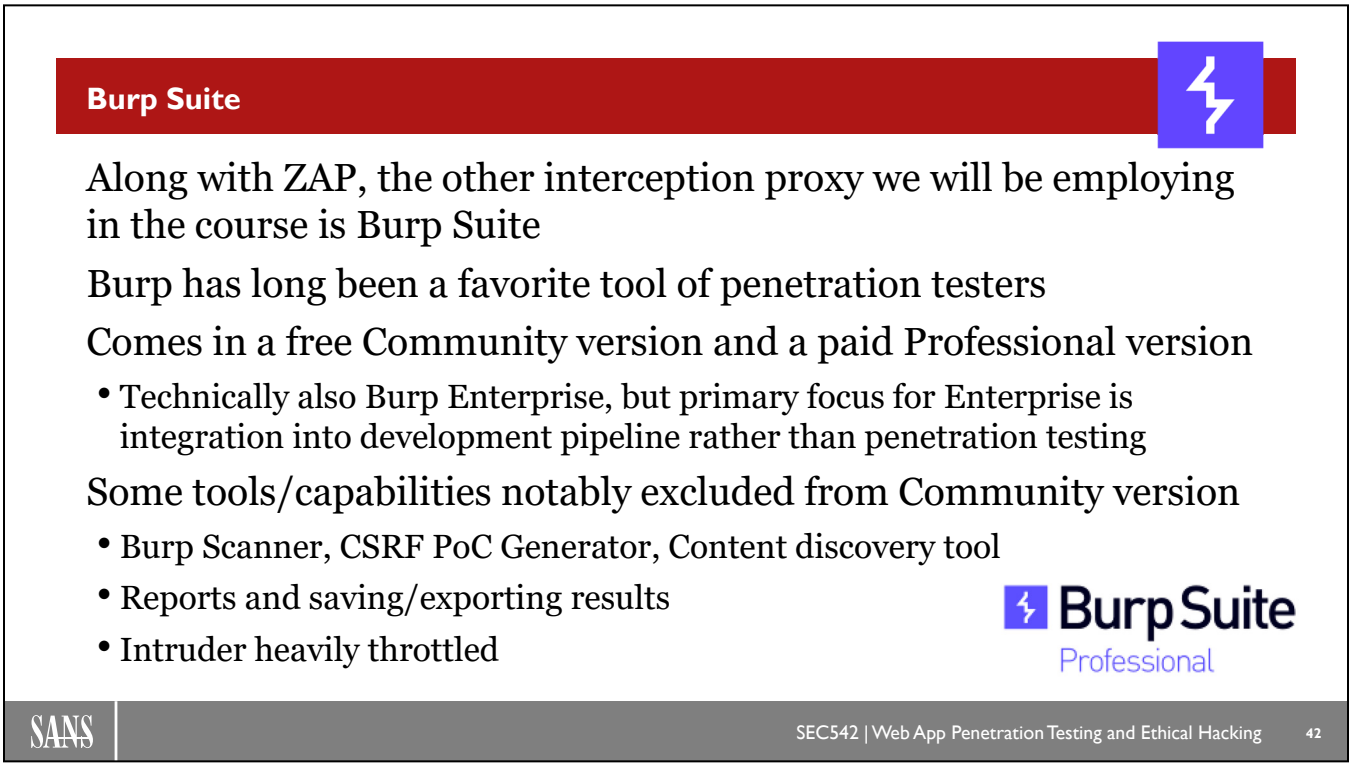
ZAP's Attack menu is new as of Version 2.4 and includes a number of functions.

ZAP's passive scanner is always running. The active scanner actively scans a site, searching for vulnerabilities.

ZAP's spider is a traditional web spider, which attempts to identify all content on a website. The AJAX spider can parse AJAX content, enabling it to (potentially) discover additional dynamic content that is missed by most web spiders.

Forced browsing incorporates DirBuster's functionality into ZAP. DirBuster is a standalone Java application that is no longer under active development.

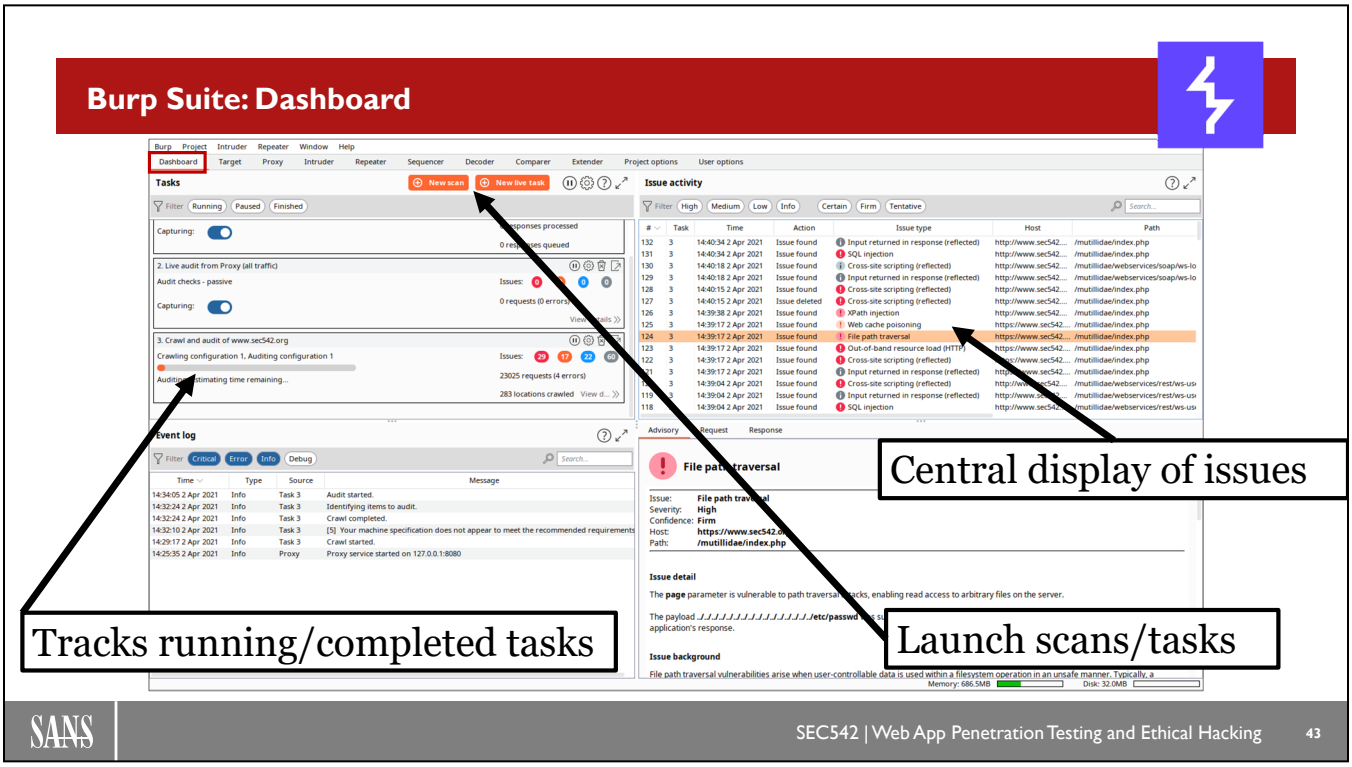
ZAP's fuzzer has improved significantly as of Version 2.4 and now rivals Burp's fuzzer.



Burp Suite

Now we get to begin exploring what is likely to become a favorite tool in your arsenal if it isn't already: Burp Suite. Burp is often cited as one of the favorite tools of almost every penetration tester, and certainly every application penetration tester. Dafydd Stuttard (@PortSwigger) created Burp Suite and also co-authored both first and second editions of the phenomenal *The Web Application Hacker's Handbook* (Wiley). Although we introduce Burp as an interception proxy, the tool, as the name suggests, is really a suite of tools much larger than the proxy.

The actual tools available to you in the suite depends on the version of Burp Suite you are using. The free Community version, while highly capable, lacks some key functionality for professional penetration testers. Perhaps the most notable omission is Burp Scanner functionality, which is the dynamic application scanning function of the suite. However, even without the Scanner, the Community version could provide much of what is needed for professional penetration testing engagements were it not for the lack of ability to save or export results or generate reports.



Burp Suite: Dashboard

The dashboard represents Burp Suite's high-level HUD (Heads-Up Display). You will not find all of the individual requests and responses sent on the dashboard. Rather, you will be able to see any currently running tasks or scans, launch new scans, and also see any security issues found.

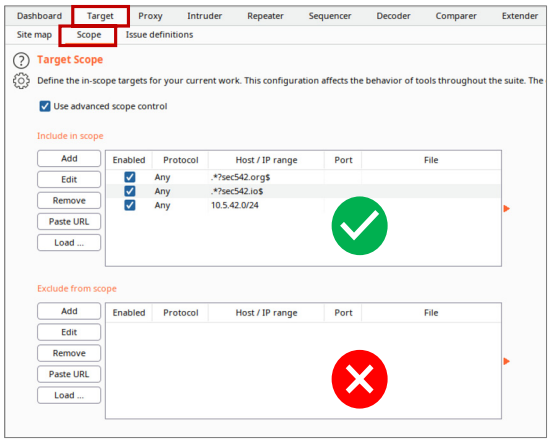
The most prominent top-left position is reserved, by default, for the Tasks pane. Here, new scans can be launched and currently running or completed scans can be seen. The right side of the dashboard allows for quickly sorting, filtering, and seeing associated request and responses associated with issues or vulnerabilities identified. Digging into the requests/responses associated with issues allows for quickly pivoting (with a simple right-click) to other aspects of Burp Suite. Additionally, the event log, at the lower-left corner, allows for quickly identifying and tool failures that might be impeding the current engagement. The dashboard debuted with the 2.x strain of Burp and seeks to make an overview of the current session much more accessible and readily available.

Burp Suite: Target - Scope

Before digging deeper into Burp Suite's myriad tools/capabilities...scope

Properly defining the scope of your application assessment:

- Makes the difference between tools running forever and completing
- Keeps you from accidentally overlooking relevant resources
- Allows you to avoid paths/systems explicitly excluded
- Ensures you generally don't run afoul of your authority and scan the universe



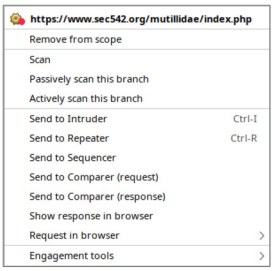
Burp Suite: Target - Scope

Though forgettable, perhaps the most essential piece of Burp Suite to understand is the Target - Scope tab. This is how we keep ourselves out of trouble while wielding this massively powerful suite of tools. Most of the automated functionality's default behavior is to honor what is referred to as the “suite scope.” This tab is where we define the suite scope. We can scope specific items in and out using this tab. The inclusion or exclusion of resources can be broadly or narrowly defined to accommodate our specific needs and the requirements of our engagement.

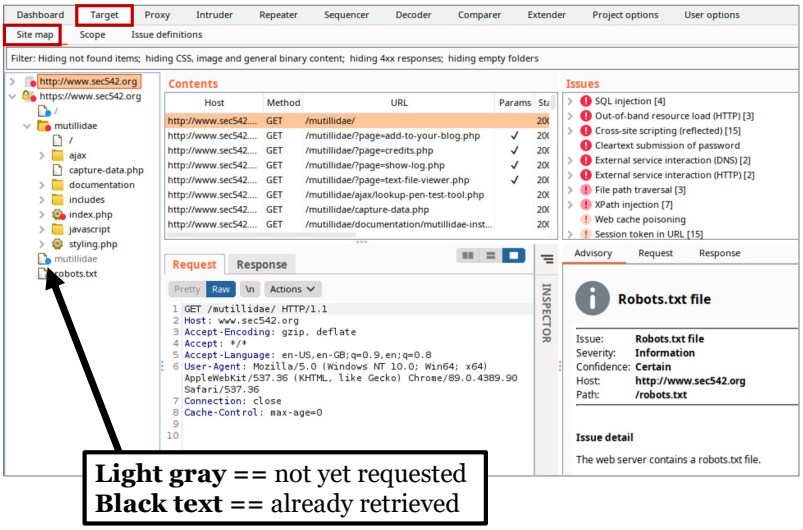
Burp Suite: Target - Site Map

Much manual testing kicks off from Burp's **Target**: **Site Map**

- Right-click a path/request and choose a tool or task



Crawling/spidering now simply a type of scan to be performed

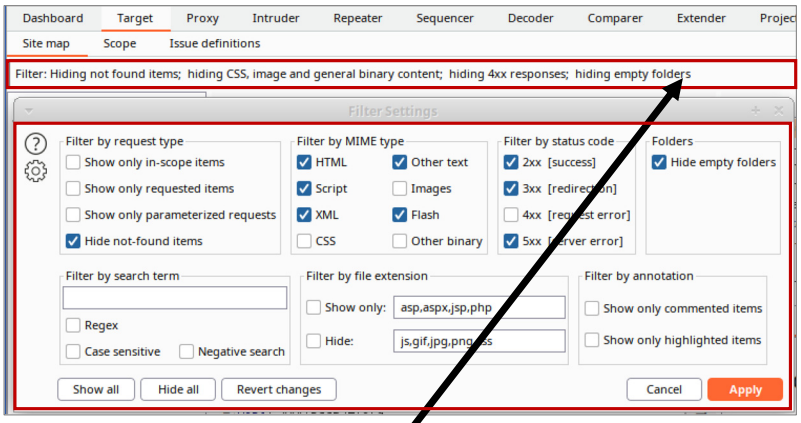


Burp Suite: Target - Site Map

Dashboard may be the overall project HUD, but Burp Suite's Target: Site map serves as the key jumping-on point for much Burp Suite functionality. The site map gives us a tree view of the various sites that have been seen by Burp. Clicking on folders or particular resources in the tree changes what is shown in the Contents and Issues portion of the site map view. Right-clicking on resources either within the tree view or under the contents portion will bring up a context menu that we very commonly use to pivot to other portions of the Burp Suite.

What is visible (or not) in the site map is dictated by the filter, which can be seen just below Burp Suite's tabs and just above the tree. This vital piece of the site map is rather commonly overlooked by folks new to Burp.

Burp Suite: Target - Site Map - Filter



Clicking in the Filter bar opens the Filter Settings dialog box and allows for tweaking

New Burp users very often overlook the **Site Map: Filter**

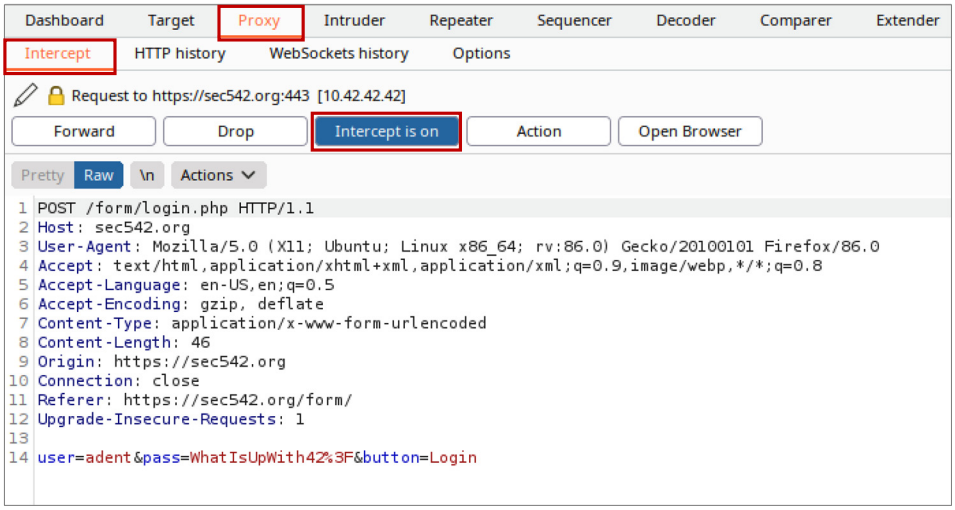
- Once it's discovered, simple filtering makes finding desired data much easier
- Also makes finding inadvertently hidden data easier

Burp Suite: Target - Site Map - Filter

The filter is one of the most commonly overlooked components in the entirety of the Burp Suite. Unfortunately, the filter can be the reason you completely overlook something, because the filter removes it from the display. Important to note is the filter doesn't remove any data from the session and doesn't keep data from being sent (that would be the suite scope). Rather, the filter merely governs what is visible in the display.

One example of default behavior that can throw some testers is that all HTTP responses with a status code in the 400s will not be shown. This is a feature if you have caused tremendous numbers of 404 messages to be generated by trying to forcibly discover content, but it can be problematic if you are trying to identify resource requests that might have been otherwise accessible had you been authenticated properly (or with appropriate credentials).

Burp Suite: Proxy



Proxy->Intercept is the reason we can refer to Burp as an interception proxy

When intercepting, it allows for editing requests generated by the browser

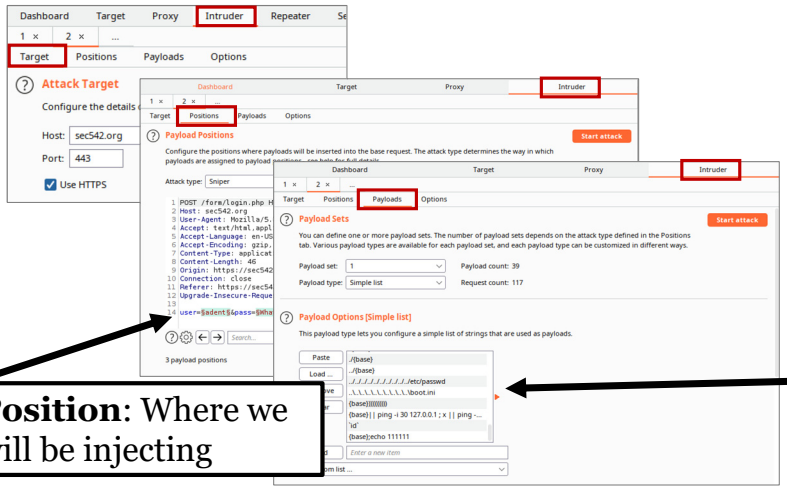
- Less commonly, responses from the server

Burp Suite: Proxy

If interception is turned on, then some requests generated by the browser will be held for the tester to review and, if desired, edit prior to forwarding the traffic on to the server. Although employed much less commonly than for requests, responses from the server can also be intercepted and edited before being released to the browser for rendering.

Proxy - Options provides many different ways to determine what will and will not be proxied. Also allows for determining what will be intercepted to allow for editing requests/responses.

Burp Suite: Intruder



- Efficient means of automating large-volume custom injections
- Obviates the need for much scripting skill

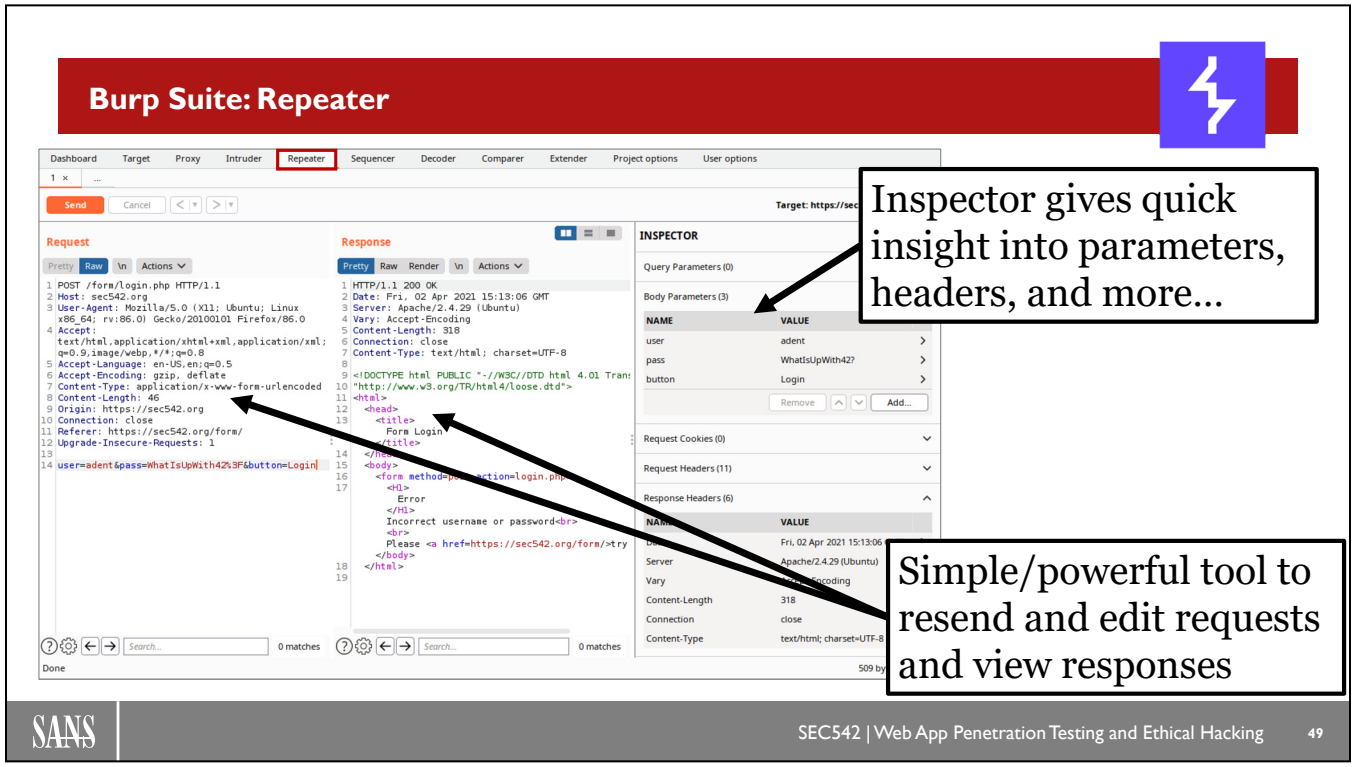
Position: Where we will be injecting

Payload: What will be injected

Intruder: the first tool that makes most testers love Burp and yearn for Pro

Burp Suite: Intruder

Intruder is easily the Burp capability that sells more Professional licenses than any other. Intruders' simple workflow consists of moving from Target->Positions->Payloads->Options. Target typically doesn't require tweaking, unless we are sending the requests to a different site from the original request. Positions allows us to highlight one or more elements of the Request header or body that will be systematically replaced. Payloads is where we define what will be injected into those positions. Many different and very powerful built-in payloads are already provided and can be tweaked in various ways. The main reason that testers often hit the Options tab is to avail themselves of the Grep options. The Grep options of match, extract, and payloads present simple means of automatically identifying relevant information in the results of the Intruder run.



Burp Suite: Repeater

A simple yet powerful testing tool is found in Burp's Repeater. This tool allows resending individual requests with or without editing and seeing the application's response. This allows for iterative tweaking of test cases to achieve the desired outcome or response. As with most elements of Burp, getting individual requests into Repeater typically involves finding the request in, for instance, the Site Map, and then right-clicking and using the context menu to send the request to Repeater.

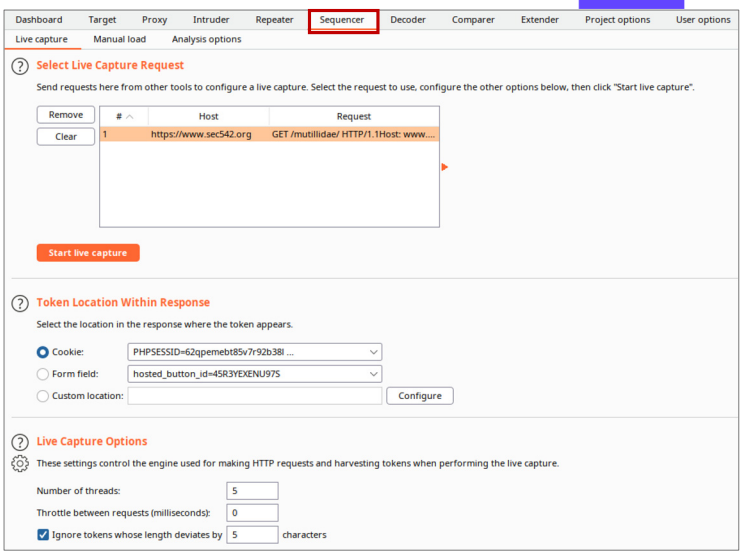
Testing tip when using Repeater: Resend the initial request without editing prior to tweaking. This allows you to ensure the previously experienced/expected application response will still result without any changes before trying to make changes to the request to achieve different results.

Burp Suite: Sequencer

Assesses data randomness

Primarily used to test whether an application's method of session generation is predictable

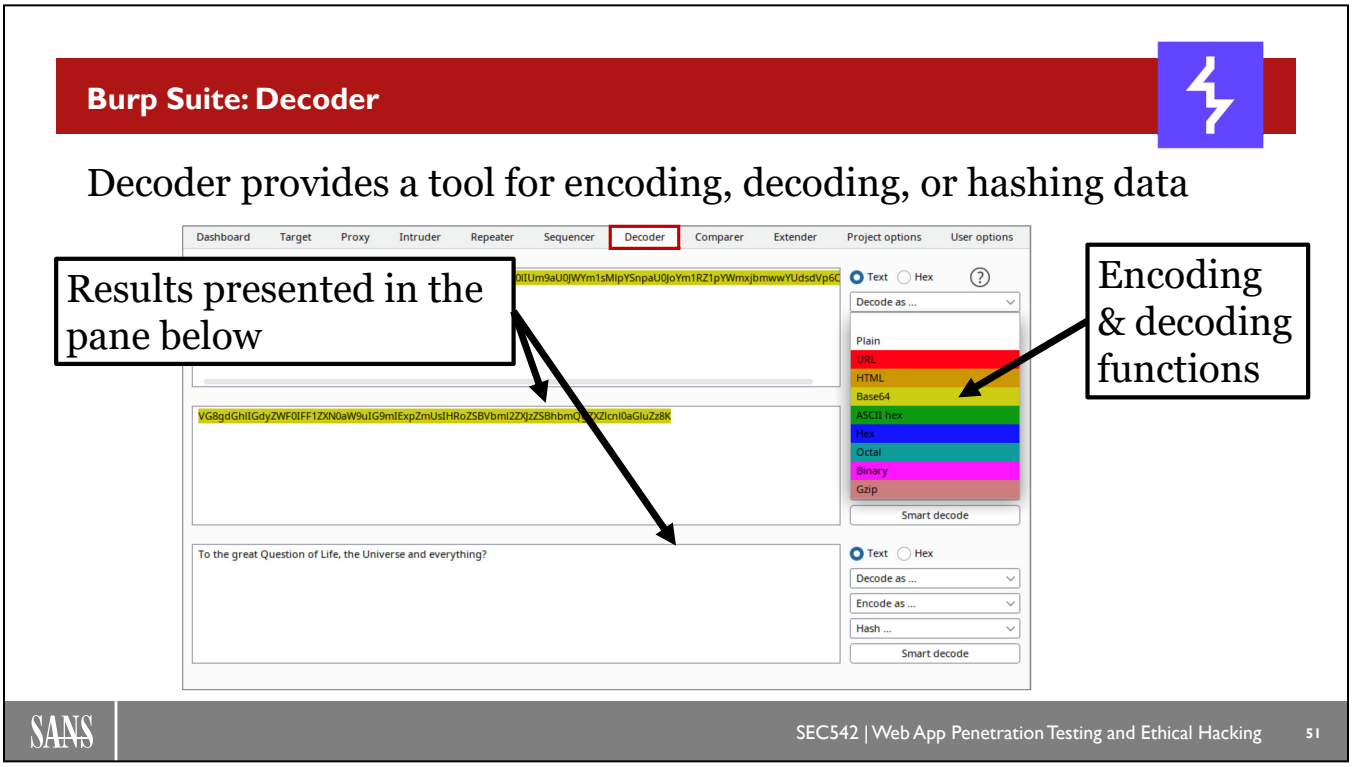
- Can be more widely applicable to assess values of anything that should be random



Burp Suite: Sequencer

The primary use-case for Sequencer has long been to perform session analysis to determine whether an application's session-generation algorithm might allow for predictable sessions, which could possibly allow an attacker to hijack sessions through knowing in advance a session token.

While Sequencer can be, and still is, used for session analysis, the likelihood of identifying predictable session generation has significantly diminished over the years. However, Sequencer can be used to assess CSRF tokens or any other element of an application that depends on being random.

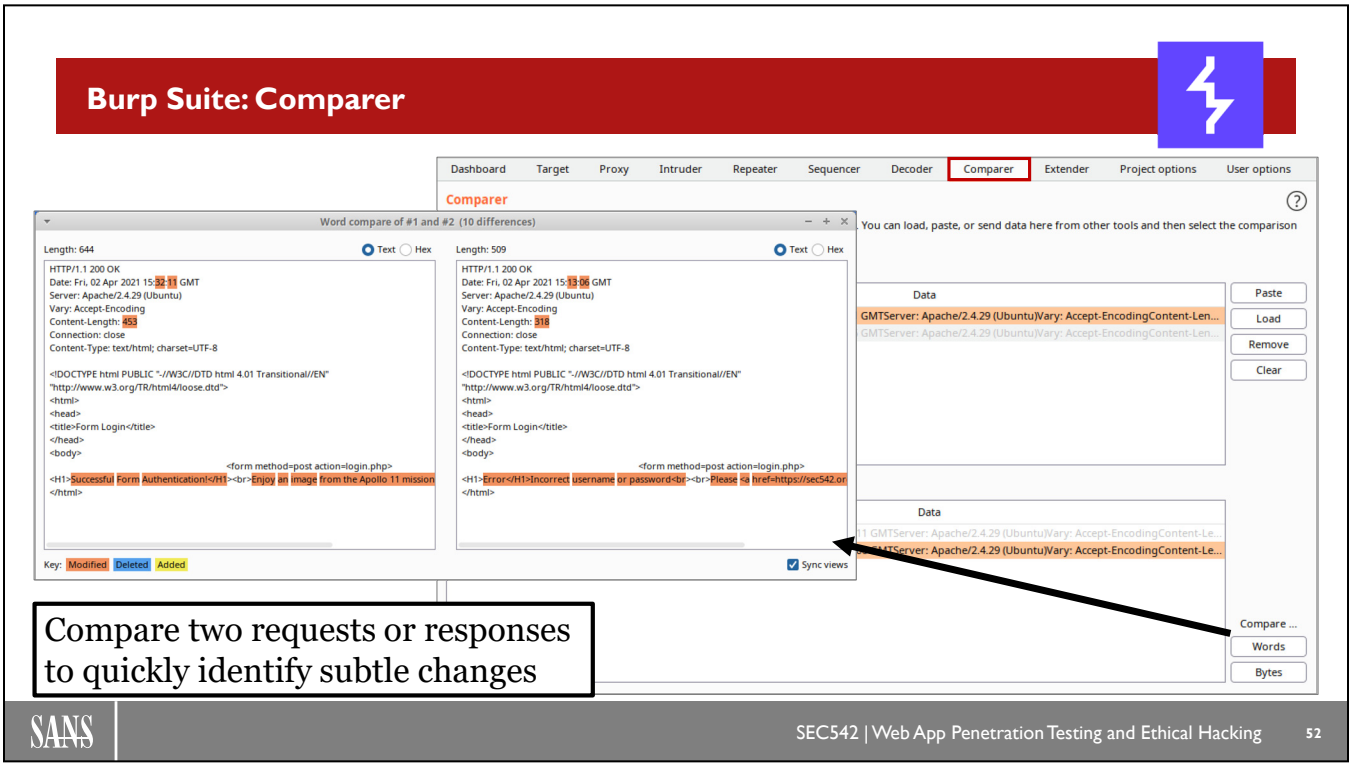


Burp Suite: Decoder

This relatively simple tool provides a handy way for the tester to manually test encoding, decoding, or hashing strings without having to leave her primary testing tool.

Burp Decoder allows the tester to encode or decode using many common methods, including URL, HTML, Base64, and Hex. Though not, strictly speaking, encoding, Gzip and its reverse can also be performed.

Hashing functionality also present in this tool includes the most commonly encountered hashing algorithms, including various forms of Secure Hash Algorithm (SHA, SHA-256, and SHA3-512) as well as MD5.



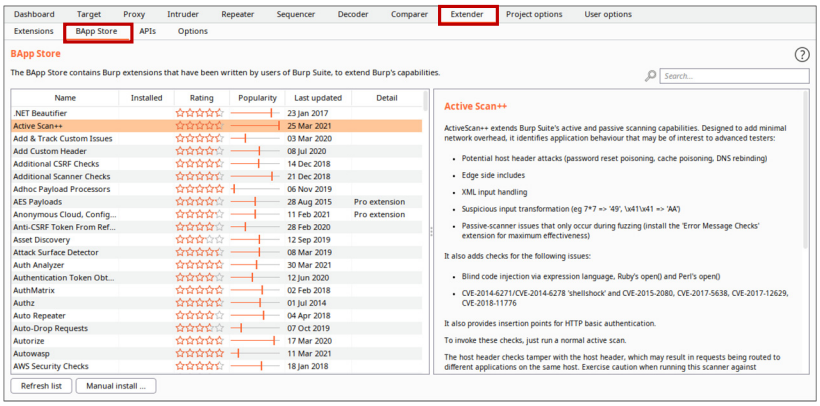
Burp Suite: Comparer

Though the naked eye might not notice it, sometimes there are significant and meaningful changes between pairs of requests or pairs of responses. Burp Suite's Comparer tool can make much quicker and easier work of identifying these, sometimes extremely subtle, changes from one response to the next.

Burp Suite: Extender

Burp's built-in capabilities not enough for you? Then Extender is your jam

Allows for building your own custom extensions or leveraging third-party-developed extensions from the BApp Store



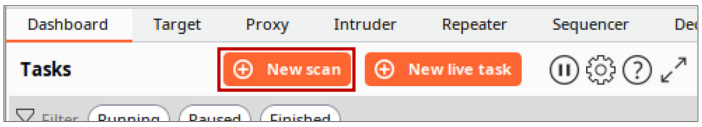
Burp Suite: Extender

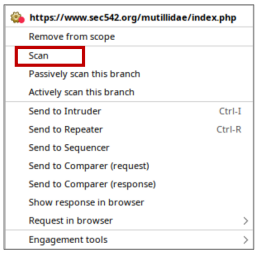
Burp makes extending functionality with your own or third-party-developed capabilities rather straightforward. The most common use of Extender is to install Burp extensions from the BApp Store, which is a marketplace for third-party extensions. Simple sorting based on last update, rating, or popularity can allow for digging into extensions.

Though Burp is Java-based, custom extensions can be written not only in Java but also in Python and Ruby.

Burp Suite: Crawling/Auditing

Clicking **New Scan** on the Dashboard or **Scan** from the context menu for a site/resource that will allow ad hoc scanning to be started





In addition to ad hoc scanning, automated active scanning (a.k.a. live auditing) can also be configured as a Live Task

Burp Suite: Scanning/Auditing

A nomenclature shift seems to be occurring within Burp Suite. While we still find references to scans and scanning, there is increasing use of "auditing" rather than scanning. Scans are now used for kicking off website crawling (following links) as well as what has more classically been referred to as scanning, which is looking for vulnerabilities. Burp now uses the term auditing for attempting to identify vulnerabilities.

Burp Suite: Engagement Tools

Though not in their own tab, the engagement tools offer some extremely powerful functionality, but are only available in the Professional version.

A sample of some of the features available in this context menu:

- **Search** - Searches across all data seen by Burp
- **Find Comments / Find Scripts** - Looks for comments and script content
- **Target Analyzer** - Quick report of static and dynamic URLs and parameters previously seen in the target application
- **Content Discovery** - Complements crawling by trying to forcibly browse paths and resources based on wordlists
- **Generate CSRF PoC** - Creates a form that includes parameters to allow testing for Cross-Site Request Forgery
- **Simulate Manual Testing** - Sends requests for random resources in the site map; makes you look busy

Burp Suite: Engagement Tools

Not all of Burp's awesomeness is found as an individual tab in Burp Suite. Some seriously cool functionality can be found under Engagement Tools in the context menu.

Depending on where you access the Engagement Tools context menu, different options will appear in the list. There are other engagement tools beyond what are highlighted in the slide. These are merely some of the highlights.

Course Roadmap

- **Section 1: Introduction and Information Gathering**
- Section 2: Content Discovery, Auth, and Session Testing
- Section 3: Injection
- Section 4: XSS, SSRF, and XXE
- Section 5: CSRF, Logic Flaws, and Advanced Tools
- Section 6: Capture the Flag

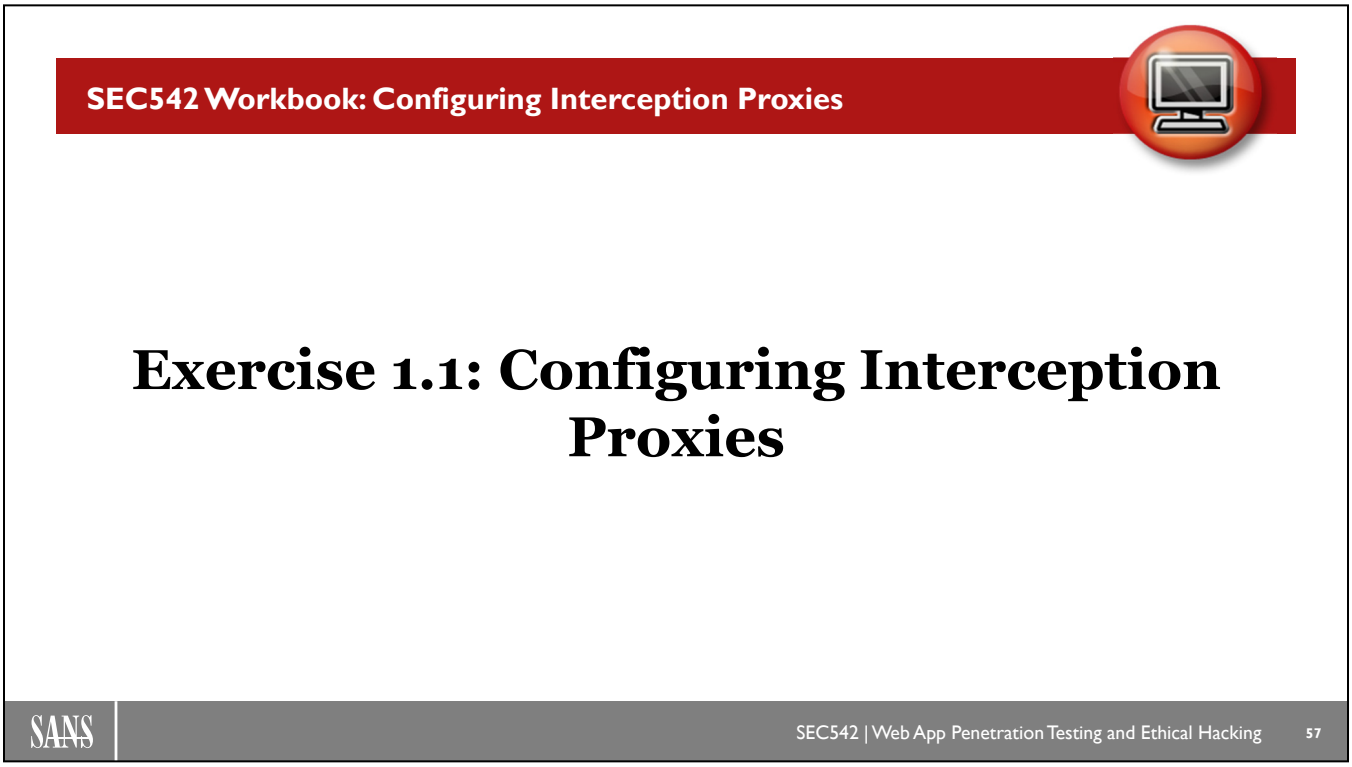
INTRODUCTION AND INFORMATION GATHERING

1. Why the Web?
2. Application Assessment Methodologies
3. Web Application Pen Tester's Toolkit
4. Interception Proxies
5. **Exercise: Configuring Interception Proxies**
6. Open Source Intelligence (OSINT)
7. Virtual Host Discovery
8. Exercise: Virtual Host Discovery
9. HTTP Syntax and Semantics
10. HTTPS and Testing for Weak Ciphers
11. Exercise: Testing HTTPS
12. Target Profiling
13. Exercise: Gathering Server Information
14. Summary
15. Bonus Exercise: Testing and Exploiting Heartbleed

Course Roadmap

Welcome to Security 542: Web App Penetration Testing and Ethical Hacking!

We'll begin with an introduction to the web, talk about information gathering and virtual host discovery, review HTTPS, and then wrap up with target profiling.



SEC542 Workbook: Configuring Interception Proxies

Please go to Exercise 1.1 in the 542 Workbook.

Course Roadmap

- **Section 1: Introduction and Information Gathering**
- Section 2: Content Discovery, Auth, and Session Testing
- Section 3: Injection
- Section 4: XSS, SSRF, and XXE
- Section 5: CSRF, Logic Flaws, and Advanced Tools
- Section 6: Capture the Flag

INTRODUCTION AND INFORMATION GATHERING

1. Why the Web?
2. Application Assessment Methodologies
3. Web Application Pen Tester's Toolkit
4. Interception Proxies
5. Exercise: Configuring Interception Proxies
6. **Open Source Intelligence (OSINT)**
7. Virtual Host Discovery
8. Exercise: Virtual Host Discovery
9. HTTP Syntax and Semantics
10. HTTPS and Testing for Weak Ciphers
11. Exercise: Testing HTTPS
12. Target Profiling
13. Exercise: Gathering Server Information
14. Summary
15. Bonus Exercise: Testing and Exploiting Heartbleed

Course Roadmap

Welcome to Security 542: Web App Penetration Testing and Ethical Hacking!

We'll begin with an introduction to the web, talk about information gathering and virtual host discovery, review HTTPS, and then wrap up with target profiling.

WSTG: Information Gathering[1]

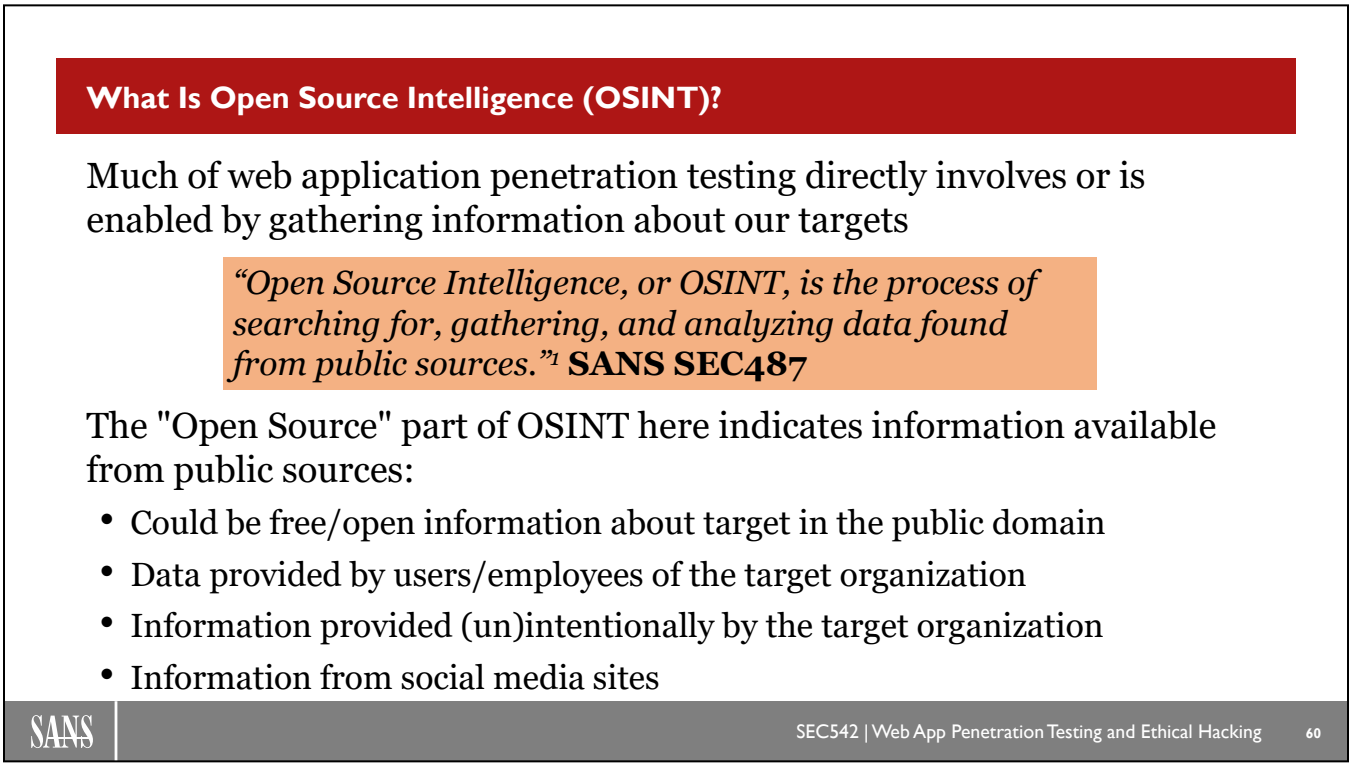
WSTG-INFO-01	Conduct Search Engine Discovery and Reconnaissance for Information Leakage
WSTG-INFO-02	Fingerprint Web Server
WSTG-INFO-03	Review Webserver Metafiles for Information Leakage
WSTG-INFO-04	Enumerate Applications on Webserver
WSTG-INFO-05	Review Webpage Content for Information Leakage
WSTG-INFO-06	Identify application entry points
WSTG-INFO-07	Map execution paths through application
WSTG-INFO-08	Fingerprint Web Application Framework
WSTG-INFO-09	Fingerprint Web Application
WSTG-INFO-10	Map Application Architecture

WSTG: Information Gathering

The table above highlights the particular Test IDs for the Information Gathering section of the OWASP Web Security Testing Guide. Note that some elements above might be discussed during a separate section of the course for pedagogical purposes. For additional details on this category, see below.

Reference:

[1] WSTG: Testing Information Gathering, https://sec542.com/57



What Is Open Source Intelligence (OSINT)?

Gathering information about target organizations and applications can greatly improve the likelihood of successful compromise for adversaries. While penetration testers are generally not expected to perform full adversary emulation exercises, reconnaissance can make our process both more efficient and effective. Open source here has nothing to do with "open source software" vs. proprietary software. The term open source here simply indicates information that is available to the public in some way.

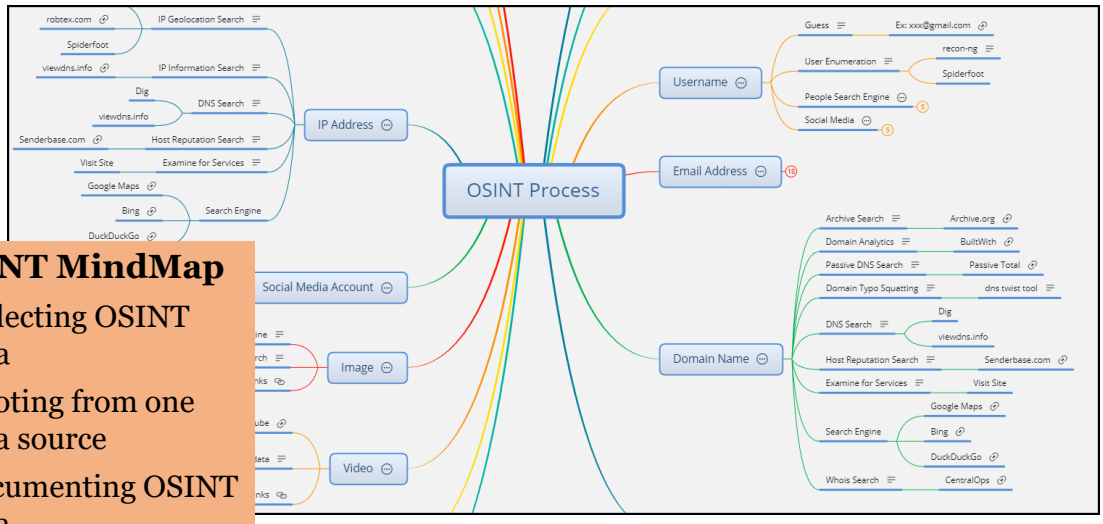
Open Source Intelligence or OSINT is the process of searching for, gathering, and analyzing data found from public sources.[1]

If you think that OSINT sounds broad enough and cool enough that you'd think SANS would even have a class dedicated to it…you'd be exactly right. Micah Hoffman's SEC487: Open Source Intelligence Gathering and Analysis is that class.

Reference:

[1] SEC487: Open-Source Intelligence (OSINT) Gathering and Analysis, https://sec542.com/58

Micah Hoffman's OSINT MindMap



OSINT MindMap

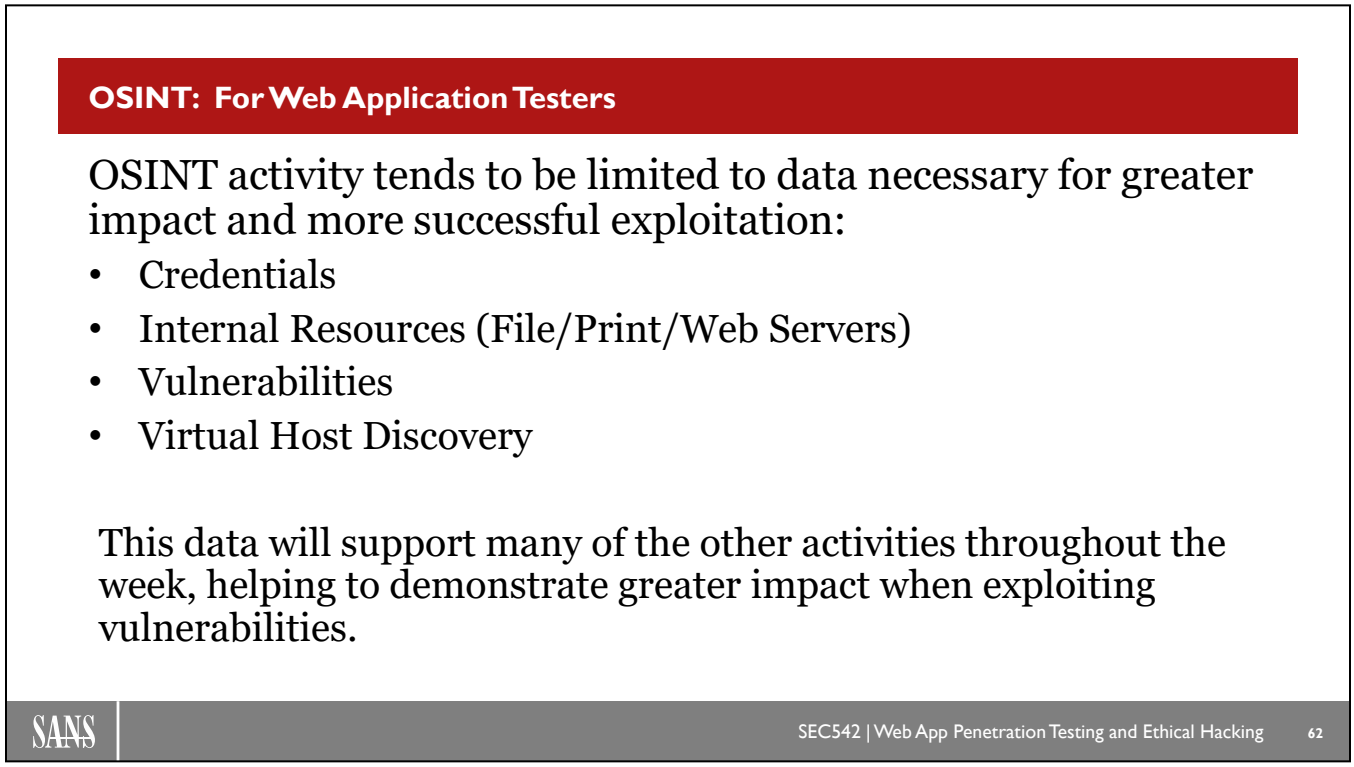
- Collecting OSINT data
- Pivoting from one data source
- Documenting OSINT data

Micah Hoffman's OSINT MindMap

The screenshot in the slide shows a segment of the OSINT MindMap created by SEC487 author Micah Hoffman (@WebBreacher). The mindmap provides substantial guidance about where to find types of data as well as what to do with the data. Much of the content of the mindmap goes well beyond the scope of OSINT for application penetration testing, but certainly IP Address, Domain Name, Email Address, Username, Image, Social Media Account, and others could well be relevant.

Reference:

[1] OSINT MindMap, https://sec542.com/59



OSINT: For Web Application Testers

While more comprehensive red team assessments will look for large quantities of information to help in breaching an organization and demonstrating the full impact of a sophisticated, determined adversary, OSINT for web application testing tends to require less effort.

Checking previous breach results for domain names associated with the target application can help to determine if password reuse allows for successful authentication to the web application. This becomes especially helpful if the web application uses email addresses for the username. If you find password values associated with those email addresses in breach data, you can try those passwords in the application you are testing.

IMPORTANT NOTE: DO NOT USE PREVIOUSLY COMPROMISED CREDENTIALS TO ACCESS THE SITES FROM WHICH THE CREDENTIALS WERE STOLEN. You are not authorized to reuse those credentials on the site from which the original attacker took them. For example, about 100 million credentials stolen from the 2012 LinkedIn breach1 were found on PasteBin in 2016 and include many email address and password pairs. Unless your scope authorizes you to interact with LinkedIn.com, you cannot legally reuse the credentials to try to authenticate to LinkedIn.com. Depending on the laws in your country, you may be able to reuse those credentials to try to authenticate to the web application that is in scope.

Later in this section we will explore discovering paths and IP addresses for internal resources, as well as using search engines to identify vulnerabilities. These data points do not usually gain much at this point in the assessment. However, once flaws that allow us to retrieve remote files, or execute commands on the underlying operating system, are discovered, the information gathered here will help demonstrate the impact of those vulnerabilities.

In the next section we will explore using OSINT to help with Virtual Host Discovery.

Reference:

[1] 2012 LinkedIn Breach: https://sec542.com/8s

OSINT: Search Engines

Perhaps the most obvious tool of the OSINT trade is the simple search engine

Most folks just leverage their favorite search engine and don't give much thought to which one to employ

- For basic searching this is likely fine, but the advanced search operators and capabilities vary somewhat across search platforms

OSINT: Search Engines

Though Google is, without question, the market leader in search and has more documentation regarding its use for OSINT, both DuckDuckGo and Bing warrant at least passing mention as well. When performing casual internet searches, the choice of browser likely doesn't cross most peoples' minds. However, our use case goes beyond the casual search, so some consideration would be appropriate.

Beyond the differing capabilities and operators present in the various search engines, there is also consideration of a search 'filter bubble.'[1] A filter bubble is "the manipulation of your search results based on your personal data."[2] The filter bubble obviously could have some impact on the results. One of the main points as it relates to OSINT for pen testing is that different people performing the same search could well receive different results. Note that the differing results can occur in spite of being logged out or searching in private/incognito mode.

References:

[1] Beware Online Filter Bubbles, https://sec542.com/5a

[2] Google Filter Bubble Study, https://sec542.com/5b

OSINT: Key Search Operators

Google

General Search Operator	Purpose/Meaning
-	Exclude elements
"Zaphod beeblebrox"	Exact match
site:sec542.com	Filter to only particular site
cache:	Search cache rather than live index
ext: pdf filetype: pdf	Filter to only include results of particular extension/filetype
intitle: inurl:	Search in the web page title or url rather than content

DuckDuckGo	Purpose/Meaning
!	! Followed by one of the 10,000 plus providers allows directly searching other sites from DDG search

Bing	Purpose/Meaning
ip:1.1.1.1	Constrain search to sites hosted at particular IP
prefer:	Emphasizes the preferred terms in search results
url:	Determine if the address found within Bing index
hitchhiker **near:10** towel	The words hitchhiker and towel must be within 10 words of each other n search results
contains:asp	Return sites with links to specified content type, as opposed to links to the files themselves as **filetype**

OSINT:

One really nifty search operator for DuckDuckGo is the bang (!). Bangs allow you to outsource your DuckDuckGo search to say Google, Bing, Google Maps, Yandex. So, from the DuckDuckGo search bar, you can perform a Google, Bing, LinkedIn, Yandex, or Baidu search just by prepending **`!google`**, **`!bing`**, **`!linkedin`**, **`!yandex`**, or **`!baidu`**, respectively, to your searches.

DuckDuckGo Search Syntax, https://sec542.com/5e

Google Refine Web Searches, https://sec542.com/5f

Bing Search Syntax

Bing Advanced Operator Reference, https://sec542.com/5g

Bing Advanced Search Keywords, https://sec542.com/5h

Reference:

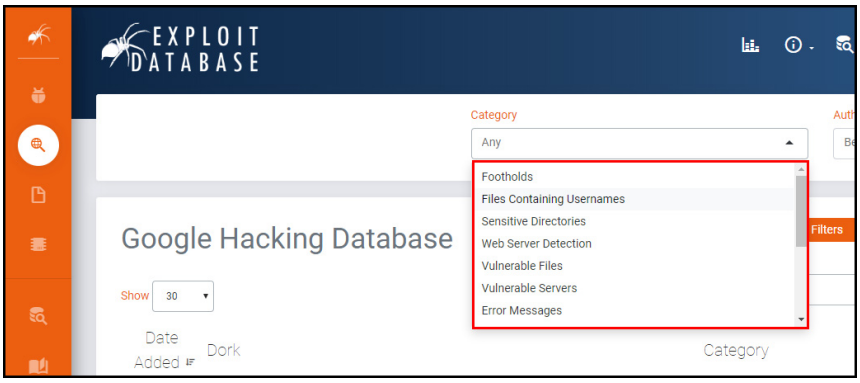
[1] DuckDuckGo Bangs, https://sec542.com/5d

OSINT: Google Dorks

Google dorks, or simply dorks, are specially crafted queries to return very specific and meaningful results

- These queries make heavy use of advanced operators for precision searching

The **Google Hacking Database** (**GHDB**), now maintained by Offensive Security, provides a categorized and substantial collection of dorks



OSINT: Google Dorks

Rather than simply supplying a keyword, as would be the case in most general-purpose searches, these crafted searches employ special operators to filter results substantially. These crafted queries can return data suggesting particular vulnerabilities being exhibited by the target. Vulnerability identification without even interacting directly with the target is extremely powerful, and just play fun/cool.

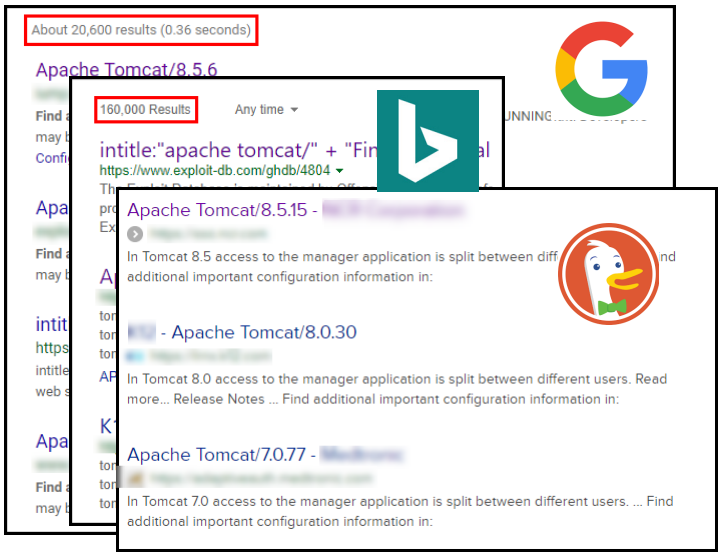
Though the name *Google dorks* clearly suggests the use of a particular search engine, the idea is more encompassing than when originally coined. Quite frequently now the term 'dork' will be used in favor of 'googledork,' perhaps to emphasize the increased scope of using crafted searches.

OSINT: Dorking

For example, here is GHDB-ID 4804[1]:

```
intitle:"apache tomcat/"
+ "Find additional
important configuration
information in:"
```

Not all, but many 'dorks' work beyond Google and could provide different results



OSINT: Dorking

The Google Hacking Database provides unique ID numbers and descriptions associated with each dork. Although Google dork remains the predominant name used for this style of search against search engines, the more generic term dork is increasingly used. The term dork, as opposed, to googledork, rightly suggests that other search engines might also be leveraged.

If not specified in the description of the dork, generally assume that the dork is written for Google. Might be that with tweaking it could be more widely applicable. Likewise, some dorks will explicitly specify the use of Bing or other search engines.

Reference:

[1] GHDB-ID:4804, https://sec542.com/5j

OSINT: Social Media - LinkedIn

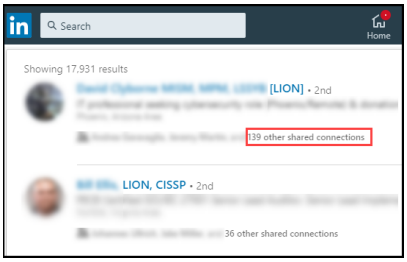
Social media contains an overwhelming volume of information

Carefully consider scope and rules of engagement

Acceptable to connect to employee's personal account?

- What if we simply share a common connection?

Shared connections often make us privy to more information



Leveraging publicly shared data gleaned from individuals may be less objectionable

- Likewise, connecting to company-sponsored accounts

LIONs FTW!

OSINT: Social Media - LinkedIn

Though some analysts distinguish social media intelligence gathering (as SOCMINT) from OSINT, for our purposes social media simply serves as a different source of information to be gathered.

More information is typically shared with connections/friends/etc. than with the general public. So, we could attempt to have employees accept us as connections or friends to allow greater access to information. Our goal of connecting to an employee is not to be their friend. Rather, we are connecting to them for the purpose of weaponizing information they might share.

One terribly significant question to ask yourself before connecting is whether it is acceptable to initiate connections with company employees' personal accounts for the purpose of gaining intelligence to be used in testing? Do the scope and rules of engagement allow us to target individuals that work for the target organization?

One way of skirting the issue of soliciting employee connections is to connect with someone that is also connected to the target individual. Enter the LIONs, or LinkedIn Open Networkers. LIONs are individuals "open to networking with people that they have never met before."[1] These individuals can allow us to connect, albeit indirectly, to the target employee. While this does not ensure that we can see full details of their profile, as it depends on privacy settings, typically this will allow access to substantially more information than the account offers publicly.

Reference:

[1] What Is a LinkedIn LION? https://sec542.com/5c

OSINT: theHarvester

theHarvester collects email addresses and hostname information from common search databases and information repositories.

For complete results, some API keys must be configured in the tool

```
[/opt/theHarvester]$ python3 theHarvester.py -d sans.org -b google -f sans.html
```

OSINT: theHarvester

Here is the command line help from the tool to show options:

Usage: theharvester options

-d: Domain to search or company name
-b: data source: baidu, bing, bingapi, censys, crtsh, dogpile,
google, google-certificates, googleCSE, googleplus, google-profiles,
hunter, linkedin, netcraft, pgp, threatcrowd,
twitter, vhost, virustotal, yahoo, all
-g: use Google dorking instead of normal Google search
-s: start in result number X (default: 0)
-v: verify host name via DNS resolution and search for virtual hosts
-f: save the results into an HTML and XML file (both)
-n: perform a DNS reverse query on all ranges discovered
-c: perform a DNS brute force for the domain name
-t: perform a DNS TLD expansion discovery
-e: use this DNS server
-p: port scan the detected hosts and check for Takeovers (80,443,22,21,8080)
-l: limit the number of results to work with(Bing goes from 50 to 50 results,
Google 100 to 100, and PGP doesn't use this option)
-h: use SHODAN database to query discovered hosts[1]

Reference:
[1] GitHub – theHarvester, https://sec542.com/6j

OSINT Suites

OSINT suites can accelerate our acquisition of pertinent OSINT

- While purpose-built specific tools could be more capable for individual tasks, the efficiency gains from OSINT suites often overcome this disparity

Suites will generally expose various APIs for our consumption and use in transforming provided source data into additional information/intelligence

- We will characterize as suites both simple open source command-line tools like theHarvester and full-blown commercial OSINT tools like Maltego

APIs we leverage for OSINT come in different forms:

- Open/public APIs
- APIs requiring a free account
- Commercial APIs ($$$)

OSINT Suites

For our use case, much greater efficiency in performing OSINT can be achieved through the use of OSINT suites rather than performing ad hoc one-off searches or wielding singularly focused tools. No specific criteria exist for characterizing a tool as being an OSINT suite rather than a one-off, but our general rule will be to require a tool to provide functionality that keeps us from having to use multiple individual tools. The primary goal of using suites is to increase efficiency, so, even though a particular suite might not have all the functionality we require or desire, it could still obviate the need for multiple one-off products. Especially as OSINT is not our primary end goal, suites will serve us quite well.

Most OSINT suites will allow us to consume many different service APIs from one interface. Many extremely valuable public and open APIs exist, but we could well have need of APIs that will require us to register an account, which could be free or cost substantial sums of money. Our discussion will typically tend toward freely available sources, as OSINT is not our primary purpose.

OSINT Suites: SpiderFoot

SpiderFoot[1]: comprehensive open source OSINT suite that allows both rapid and comprehensive analysis of a given target starting point (IP, domain name, etc.):

- As with many OSINT suites, SpiderFoot will often require us to provide API keys (some free…others $$$) for some functionality
- Unlike Maltego, open source SpiderFoot is CLI driven

For getting started more quickly, use of Docker is suggested[2]

SpiderFoot HX: cloud-hosted commercial version that includes many features beyond the OSS version[3]

OSINT Suites: SpiderFoot

Another OSINT suite beyond Maltego is available via the open source SpiderFoot. This command-line-driven tool offers many different integrations that allow for quickly gathering data on our target. As is the case with many OSINT tools, you will encounter much functionality that requires input of an API key. Some of these API keys merely require us to create a free account with the service. However, others could require substantial cash outlay to leverage their functionality.

SpiderFoot can run on both Windows and Linux natively. For an easier path that obviates the need to deal with setup or prerequisites, SpiderFoot works well under Docker. For yet another approach, you could explore whether SpiderFoot HX might be an appropriate choice. This is SpiderFoot's commercial cloud-hosted offering that provides some interesting enterprise-targeted capabilities like customer support and being designed for teams to drive SpiderFoot. Another interesting capability of SpiderFoot HX is the ability to get automated OSINT change detection reports. While not overtly a big selling point for most penetration testing engagements, the capability is compelling for organizations worried about OPSEC.

References:

[1] SpiderFoot, https://sec542.com/6h

[2] SpiderFoot Docker, https://sec542.com/6g

[3] SpiderFoot HX, https://sec542.com/6i

Course Roadmap

- **Section 1: Introduction and Information Gathering**
- Section 2: Content Discovery, Auth, and Session Testing
- Section 3: Injection
- Section 4: XSS, SSRF, and XXE
- Section 5: CSRF, Logic Flaws, and Advanced Tools
- Section 6: Capture the Flag

INTRODUCTION AND INFORMATION GATHERING

1. Why the Web?
2. Application Assessment Methodologies
3. Web Application Pen Tester's Toolkit
4. Interception Proxies
5. Exercise: Configuring Interception Proxies
6. Open Source Intelligence (OSINT)
7. **Virtual Host Discovery**
8. Exercise: Virtual Host Discovery
9. HTTP Syntax and Semantics
10. HTTPS and Testing for Weak Ciphers
11. Exercise: Testing HTTPS
12. Target Profiling
13. Exercise: Gathering Server Information
14. Summary
15. Bonus Exercise: Testing and Exploiting Heartbleed

Course Roadmap

Welcome to Security 542: Web App Penetration Testing and Ethical Hacking!

We'll begin with an introduction to the web, talk about information gathering and virtual host discovery, review HTTPS, and then wrap up with target profiling.

Virtual Host Discovery

HTTP version 1.1 introduced virtual hosts:

- Uses the domain name to distinguish between web applications on the web server

To find these virtual hosts, we frequently need to perform OSINT"

- DNS:
  - Querying DNS servers directly
  - Querying online DNS databases
- HTTPS Certificates:
  - Certificate transparency databases
  - Webserver configuration

Virtual Host Discovery

Virtual host discovery involves identifying the DNS records associated with the web server's IP address. Not all DNS records will map to a web application, so any discovered names must be tested to see if a virtual host is configured. A request for the webroot in a browser or using curl is frequently enough to identify a web application associated with a virtual host. Occasionally, forced browsing (which we will talk about in Section 2) may be required to find content.

Active virtual host discovery techniques involve making requests directly to DNS servers. Passive techniques involve querying services that make collected DNS data available as a service. Lastly, HTTPS certificates can be a great source to find virtual hosts.

All three techniques are presented in this section.

DNS

- DNS is the Domain Name System, a global hierarchical database of domain names:
  - Uses UDP port 53 for payloads <= 512 bytes*
  - Uses TCP port 53 for payloads > 512 bytes, notably zone transfers
- As of early 2021, there were more than 1500 DNS Top Level Domains (TLDs)[1]
- DNS can provide a wealth of information and is specifically useful for virtual host discovery

DNS

The bulk of DNS operates via UDP port 53. Most zone transfers occur via TCP port 53 because they are typically larger than 512 bytes. A zone transfer is the download DNS zone or database of names for a given domain or subdomain.

It is possible to perform a zone transfer via UDP for zones 512 bytes or less. The transfer tool must support this functionality. Recent versions of dig have removed this option and force all transfers via TCP (even when the +notcp option is supplied). Some sites simply block TCP port 53, thinking this also blocks DNS zone transfers. Most zones are larger than 512 bytes, so this will block most transfers, but there is a (small) window of opportunity in that case if the zone is 512 bytes or less.

* Note that 512 bytes was the historic limit for DNS UDP transfers, but Extension mechanism for DNS (EDNS) enables up to 4096 bytes transferred via UDP.

The list of TLDs has grown substantially in the age of "dot anything." Any organization that owns the proper intellectual property can pay ICANN a $185,000 “evaluation fee” to apply to reserve a custom TLD. So, we now have .beer and .pizza, .cisco, and .microsoft (to sample a few geek-oriented TLDs). We even have .sex (and .sexy).

Reference:

[1] TLDs IANA, https://sec542.com/6u

Querying DNS Servers Directly

- Virtual hosts are identified by their associated domain name, so it makes sense to interact with DNS directly to try to find relevant names
- Several techniques are available:
  - DNS zone transfers:
    - Not usually possible with publicly available DNS servers
  - DNS "brute force" (dictionary) scans
  - Reverse DNS (PTR) scans

When Zone Transfers Aren't Available

DNS zone transfers are designed to allow secondary DNS servers to slave (mirror) off a primary. If successful, the entire DNS zone is downloaded. Zone transfers are great from a reconnaissance standpoint but not available publicly in most cases unless the organization has delegated DNS to its (non-security-minded) ISP. The syntax to perform a full (AXFR) zone transfer using dig is:

```
$ dig sec542.org -t axfr
```

Since zone transfers are not likely to work, we need names to expand our list of potential victim websites, including virtual hosts; so, what then?

Two basic techniques are quite useful: reverse DNS (PTR) scans and DNS brute force (dictionary) scans.

We discuss each next.

Reverse DNS Scan

Many DNS administrators (and DNS tools) reliably create reverse (**PTR**) records for every forward (**A**) record:

- **A**: www.sec542.org -> 192.168.1.8
- **PTR**: 192.168.1.8 -> sec542.sans.org

The approach: Perform a WHOIS lookup for IP addresses owned by the target organization and then perform a reverse DNS lookup for every IP:

- In-class hypothetical example Sec542, Inc., owns 192.168.1.0/24
- So, we'll perform a scan of the reverse DNS records for 192.168.1.0–192.168.1.255

Reverse DNS Scan

We previously discussed how WHOIS can be used to identify public netblocks owned by a target organization. Once these are identified, reverse DNS scans can be used to resolve the PTR records for each IP address.

Brute force scans (discussed next) will discover more names that lead to virtual hosts, because each PTR record points to one name (only), whereas brute force scans can discover multiple CNAMEs (canonical names, aka DNS aliases) that point to one IP address.

Many tools can perform reverse DNS scans; here is the DNSRecon syntax:

```
$ dnsrecon.py -r 192.168.1.0/24
```

We will learn about DNSRecon shortly.

DNS Brute Force Scans

- Although commonly called “brute force” scans, these DNS scans are actually dictionary attacks:
  - Supply a dictionary of potential DNS names
  - Read each entry
  - Attempt to resolve $entry.example.com
- DNSRecon (discussed shortly) has a number of useful dictionaries (called wordlists)

DNS Brute Force Scans

DNS brute force scans are an additional technique for discovering DNS names, including “hidden” DNS names that are not publicly published outside of DNS and do not have reverse (PTR) records pointing to them.

Most public DNS servers are high performance, and most clients will not notice massive DNS brute force scans (even those that go on for days). One common exception: When DNS logs grow so large that they fill the disk and potentially crash the server. Always scan with permission, and be sure to discuss this risk with your clients if you perform large DNS brute force scans!

DNSRecon (discussed shortly) has a number of useful dictionaries (the Sec542 Linux VM path is shown):

- /opt/dnsrecon/namelist.txt: 1,907 entries
- /opt/dnsrecon/subdomains-top1mil-5000.txt: 5,000 entries
- /opt/dnsrecon/subdomains-top1mil-20000.txt: 20,000 entries

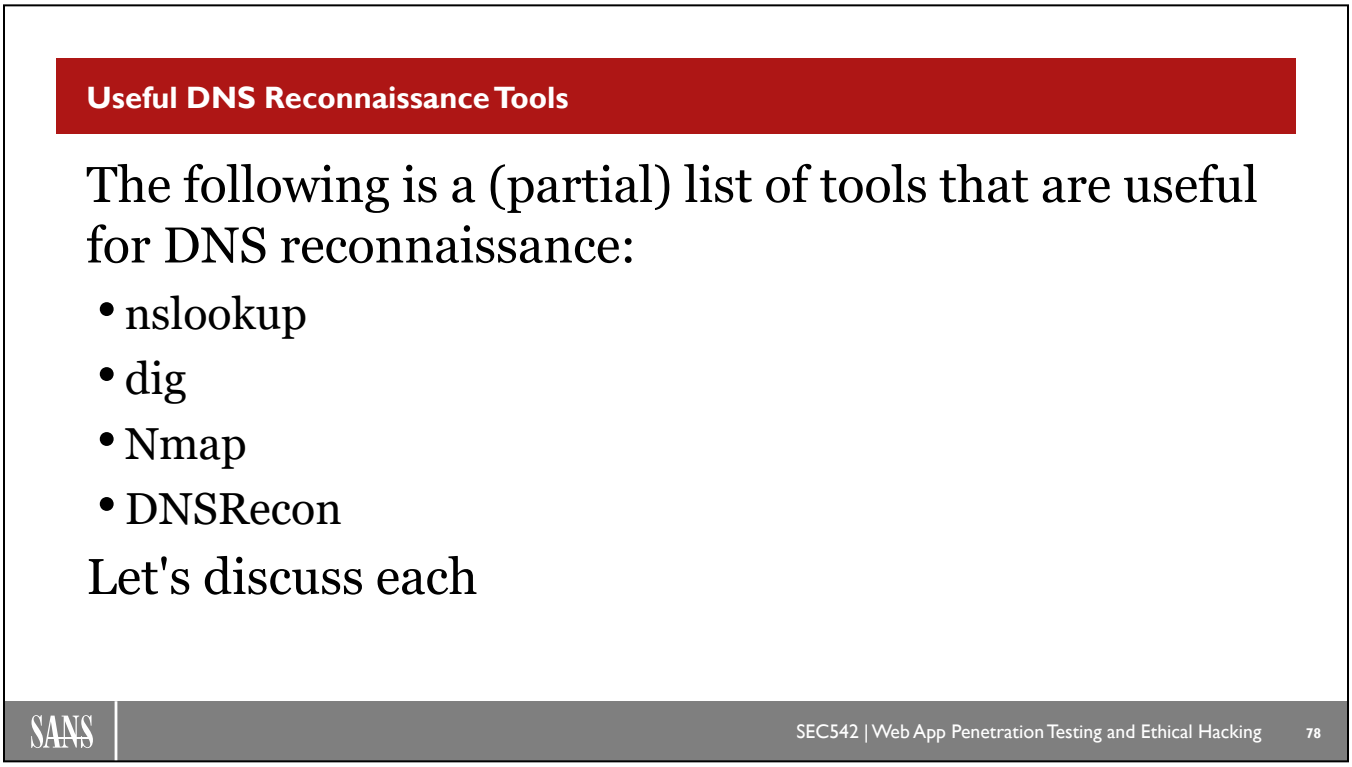
The last two lists were created by Ryan Dewhurst (@ethicalhack3r):

> *I conducted a DNS Zone Transfer (axfr) against the top 2000 sites of the Alexa Top 1 Million. I did this to create a better subdomain brute forcing word list… After creating a multithreaded and parallelised PoC in Ruby to do the Zone Transfers, it took about 5 minutes to conduct the Zone Transfers against the top 2000 compared to the 12 hours it took me to do the top 2000 using a single thread. I decided it was possible to do a Zone Transfer against the whole top 1 million sites.*[1]

These wordlists may be used with any DNS brute force tool, including Nmap and Metasploit (we will discuss both shortly).

Reference:

[1] ethicalhack3r.github.io, https://sec542.com/6v



Useful DNS Reconnaissance Tools

The following is a (partial) list of tools that are useful for DNS reconnaissance:

- nslookup
- dig
- Nmap
- DNSRecon

We discuss each next.

This list is *not* meant to be exhaustive. There are many other DNS tools out there, including the venerable host command (installed in /usr/bin/host in the Sec542 Linux VM), but these have proven the most useful and therefore provide excellent "bang for the buck."

nslookup

Most of us started interrogating DNS via nslookup

- And then we moved on to dig

Pros of nslookup:

- Near-universal availability, including UNIX, Linux, macOS, and Windows
- Allows "living off the land"; the client is usually installed on compromised hosts, including web servers
- Useful for confirmation of blind command injection

Cons:

- It has limited functionality compared to dig
- Functionality has been removed from newer versions

nslookup

Most of us performed our first command-line DNS lookup via nslookup. Then we grew up and used dig, which is far more powerful, as we will learn next.

One of the most useful features of nslookup is when it is used to confirm blind command injection. The nslookup client is available on virtually every operating system used to host web servers.

Outbound DNS is often unfiltered. Even networks that heavily filter outbound traffic usually allow DNS resolution, often via local forwarders (which act as DNS forward proxies).

In these cases, a web application penetration tester who controls the primary name server for sec542.org can attempt blind command injection of "nslookup $uniquestring.sec542.org" and simply tail the DNS logs (or run a sniffer) to see if the victim attempts to resolve the unique name (perhaps via a DNS forwarder). Note that the actual resolved name ("$uniquestring" in the previous example) does not need to exist for this to work.

The nslookup client was listed as deprecated (slated for possible future removal) with the release of Bind 9 in 2000. It received a stay of execution with the release of Bind 9.3 in 2004 and is once again fully supported. It has been simplified, and functionality has been removed (including the ability to perform zone transfers).

dig

dig is a fully featured DNS client

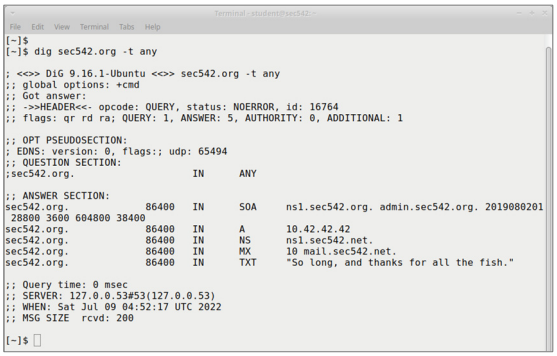
- Dig is the best dedicated DNS client for power use

Available natively on macOS and most UNIX/Linux distros

- It is not installed natively on Windows, unfortunately

One of the clients included in ISC's (Internet Systems Consortium) BIND DNS server package

- Along with nslookup and host
- ISC also has an official Windows BIND port, including all client software



dig

The command above is:

```
$ dig sec542.org -t any
```

This tells dig to look up the sec542.org domain and return all record types ("-t any").

The dig client is part of BIND:

> *BIND is open source software that implements the Domain Name System (DNS) protocols for the Internet. It is a reference implementation of those protocols, but it is also production-grade software, suitable for use in high-volume and high-reliability applications. The name BIND stands for "Berkeley Internet Name Domain," because the software originated in the early 1980s at the University of California at Berkeley.*[1]

The BIND source code may be downloaded from https://sec542.com/6w.

Reference:

[1] ISC BIND, https://sec542.com/6w

dig Syntax

The basic usage is:

```
$ dig @<nameserver> example.com options…
```

- Will use the default DNS name server if @<nameserver> is omitted

Look up all sec542.org records (SOA, A, NS, MX, and more):

```
$ dig @192.168.1.8 sec542.org -t any
```

Look up sec542.org MX records only:

```
$ dig @192.168.1.8 sec542.org -t mx
```

Attempt a zone transfer of sec542.org:

```
$ dig @192.168.1.8 sec542.org -t axfr
```

Simplified PTR (reverse) lookup:

```
$ dig -x 192.168.1.23
```

Query the nameserver's version of BIND:

```
$ dig @192.168.1.8 version.bind chaos txt
```


dig Syntax

You may someday win Hacker Jeopardy by remembering the syntax for querying version.bind via the command line. The query uses the archaic CHAOSNET DNS records. CHAOSNET was an early LAN technology (3Mb Ethernet via coaxial cable) created at MIT in 1975 when Colossal Cave Adventure[1] was considered cutting edge.

Why should web application penetration testers care about the version of BIND? First, an unpatched name server is indicative of deeper problems at an organization. DNS *must* be secure; it's arguably of the same importance as firewall security. Second, a web application penetration test is often part of a larger (broader) penetration test.

The "-x" flag allows greatly simplified lookup of reverse (PTR) records, which are part of the in-addr.arpa zone. For example, here's how we used dig to look up PTR records in the old days:

```
$ dig 23.1.168.192.in-addr.arpa PTR
```

The format is the IP address, octets reversed, followed by .in-addr.arpa. In this case, we're looking up the reverse record for 192.168.1.23. This performs the same lookup and is much easier to remember/type:

```
$ dig -x 192.168.1.23
```

You can try both (or any of the examples on this slide).

Reference:

[1] The Colossal Cave Adventure page, https://sec542.com/6x

Nmap DNS NSE Scripts

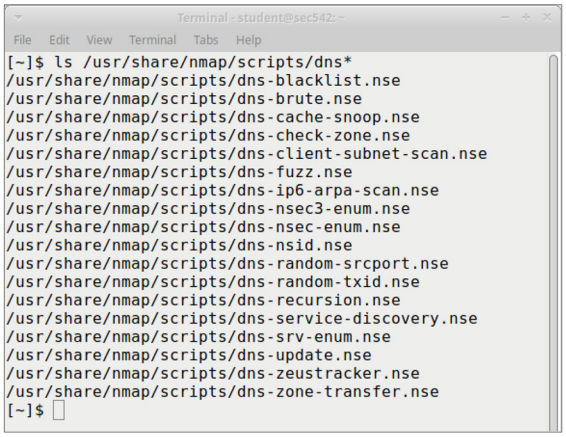
Nmap has a number of DNS-oriented NSE scripts

- NSE is the Nmap Scripting Engine

Some of the scripts replicate functionality available via dig

- Including dns-zone-transfer

dns-brute.nse is useful for discovering CNAMEs



Nmap DNS NSE Scripts

Nmap also includes DNS scanning as part of its core functionality, including reverse DNS scans, which we describe in the upcoming DNS lab.

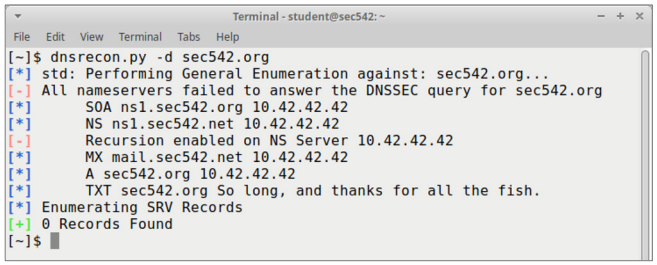
You may view the Nmap DNS NSE scripts installed in the Security542 Linux VM by typing:

```
$ ls /usr/local/share/nmap/scripts/dns*
```

We also use the dns-brute.nse script in the next lab.

DNSRecon

- DNSRecon by Carlos Perez (@darkoperator) performs many DNS reconnaissance functions:
  - Available in /opt/dnsrecon/dnsrecon.py in the Sec542 Linux VM
- Includes a number of useful wordlists for DNS brute force scans
- Advanced features include DNSSEC and mDNS support
- Carlos also ported many DNSRecon functions to Metasploit auxiliary DNS modules



DNSRecon

DNSRecon functionality includes:

- *Check all NS Records for Zone Transfers.*
- *Enumerate General DNS Records for a given Domain (MX, SOA, NS, A, AAAA, SPF and TXT).*
- *Perform common SRV Record Enumeration.*
- *Top Level Domain (TLD) Expansion.*
- *Check for Wildcard Resolution.*
- *Brute Force subdomain and host A and AAAA records, given a domain and a wordlist.*
- *Perform a PTR Record lookup for a given IP Range or CIDR.*
- *Check a DNS Server Cached records for A, AAAA and CNAME Records provided a list of host records in a text file to check.*
- *Enumerate Common mDNS records in the Local Network.*
- *Enumerate Hosts and Subdomains using Google.*[1]

DNSRecon is available at https://sec542.com/6y**.**

Basic usage is dnsrecon.py –d <domain>.

Here is (simplified) DNSRecon syntax (from running dnsrecon.py, with some cleanup/formatting):

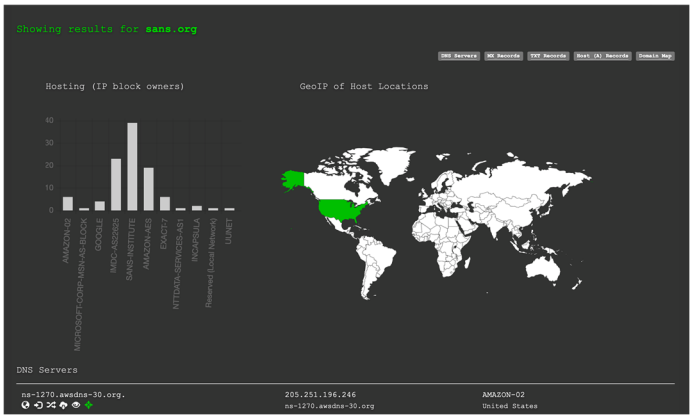
```
$ dnsrecon.py

Usage: dnsrecon.py <options>

Options:
   -h, --help                  Show this help message and exit
   -d, --domain      <domain>  Domain to Target for enumeration.
   -r, --range       <range>   IP Range for reverse lookup brute force
   -n, --name_server <name>    Domain server to use
   -D, --dictionary  <file>    Dictionary file to use for brute force
   -t, --type        <types>   Specify the type of enumeration to perform
   -a                          Perform AXFR with the standard enumeration
   -s                          Reverse lookup of ipv4 ranges in SPF Records
   -g                          Perform Google enumeration
   -w                          Do deep whois analysis and reverse lookup
   -z                          Performs a DNSSEC Zone Walk
```

Reference:

[1] dnsrecon Wiki · GitHub, https://sec542.com/6y

DNS Repositories

- Online services collect DNS data and make it publicly available for research
  - Not all of these services are free
- Examples:
  - Farsight Security's DNSDB[1]: Uses passive DNS[2] data collected from observed DNS requests.
  - Dnsdumpster.com: Uses Open Source repositories to find DNS information



DNS Repositories

Understanding how to interact with DNS and harvest information is necessary for professionals. Similar to how a carpenter knows how to use a manual screwdriver yet prefers to use an electric driver, web app pen testers can save time and gain additional information from some of the high-quality collections of DNS data available through online services. Many products exist, but two high quality examples are Farsight Security's DNSDB (commercial product) and HackerTarget.com's DNSDumpster (free up to 100 A records, commercial product, Domain Profiler[3], available for more data).

Below are descriptions from each of the solutions' website:

"Farsight collects Passive DNS data from its global sensor array. It then filters and verifies the DNS transactions before inserting them into the DNSDB, along with ICANN-sponsored zone file access download data. The end result is the highest-quality and most comprehensive Passive DNS data service of its kind - with more than 100 billion DNS records since 2010."[1] -- FarsightSecurity.com

"No brute force subdomain enumeration is used as is common in dns recon tools that enumerate subdomains. We use open source intelligence resources to query for related domain data. It is then compiled into an actionable resource for both attackers and defenders of Internet facing systems.

More than a simple DNS lookup this tool will discover those hard to find sub-domains and web hosts. The search relies on data from our crawls of the Alexa Top 1 Million sites, Search Engines, Common Crawl, Certificate Transparency, Max Mind, Team Cymru, Shodan and scans.io."[4] – dnsdumpster.com

An early project looking into anomalies within DNS queries by analyzing requests collected through passive DNS techniques is described in the technical paper "Passive Monitoring of DNS Anomalies"[5].

"We collected DNS responses at the University of Auckland Internet gateway in an SQL database, and analyzed them to detect unusual behavior. Our DNS response data have included typo squatter domains, fast flux domains and domains being (ab)used by spammers. We observe that current attempts to reduce spam have greatly increased the number of A records being resolved. We also observe that the data locality of DNS requests diminishes because of domains advertised in spam."[6]

References:

[1] Farsight Security's DNSDB Product: https://sec542.com/8u

[2] Passive DNS Replication Paper: https://sec542.com/8w

[3] HackerTarget.com's Domain Profiler: https://sec542.com/8x

[4] DNSDumpster Website: https://sec542.com/8v

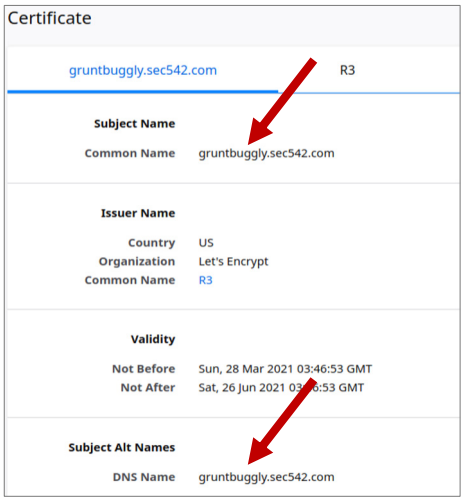
[5] Passive Monitoring of DNS Anomalies", Bojan Zdrnja, Nevil Brownlee, and Duane Wessels: https://sec542.com/9k

[6] Ibid, Abstract

HTTPS Certificate Virtual Host Discovery

Certificates used for HTTPS can be a great place to find virtual host names:

- The Common Name and DNS Name values represent the hostname associated with the web application (virtual host name)
- Sometimes visiting the IP address via HTTPS will present a certificate with a DNS name
- Certificate Transparency databases provide insight into the certificates that have been issued:
  - Not all certificates will provide a host name (wildcards: *.sans.org or *.giac.org)



HTTPS Certificate Virtual Host Discovery

Since HTTPS certificates are frequently purchased for web applications, accessing the certificate may give you insight into one, or more, domain names. For the browser to consider the certificate as valid, the hostname in the address bar must correspond to one of the names on the certificate. For this reason, the names on the certificate will frequently map to configured virtual hosts.

Note that organizations can purchase wild card certificates that match any subdomain name with the parent certificate, for example *.sans.org or *.giac.org. In this case, the certificate will not be of much help in finding virtual hosts.

Certificate Authorities, which are responsible for issuing certificates, publish details in append-only logs about the certificates that are purchased. Databases of these certificate logs are available and can be queried by domain name. If the certificates are created with hostnames, versus using a wildcard, then it can be possible to find virtual hosts by searching certificate transparency databases. The Certificate Transparency site describes the logs: “Certificate logs are append-only ledgers of certificates. Because they're distributed and independent, anyone can query them to see what certificates have been included and when. Because they're append-only, they are verifiable by Monitors.”[1]

Reference:

[1] How Certificate Transparency Works: https://sec542.com/a7

Course Roadmap

- **Section 1: Introduction and Information Gathering**
- Section 2: Content Discovery, Auth, and Session Testing
- Section 3: Injection
- Section 4: XSS, SSRF, and XXE
- Section 5: CSRF, Logic Flaws, and Advanced Tools
- Section 6: Capture the Flag

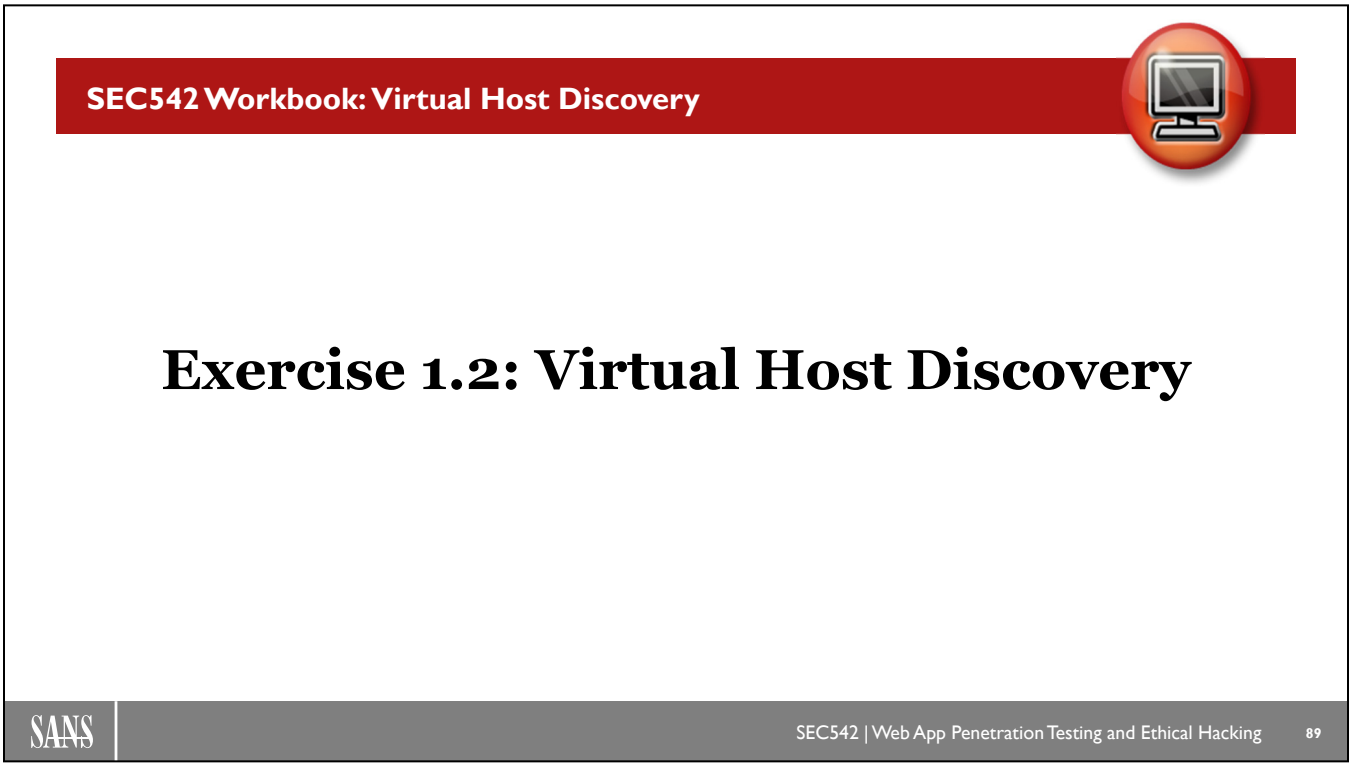
INTRODUCTION AND INFORMATION GATHERING

1. Why the Web?
2. Application Assessment Methodologies
3. Web Application Pen Tester's Toolkit
4. Interception Proxies
5. Exercise: Configuring Interception Proxies
6. Open Source Intelligence (OSINT)
7. Virtual Host Discovery
8. **Exercise: Virtual Host Discovery**
9. HTTP Syntax and Semantics
10. HTTPS and Testing for Weak Ciphers
11. Exercise: Testing HTTPS
12. Target Profiling
13. Exercise: Gathering Server Information
14. Summary
15. Bonus Exercise: Testing and Exploiting Heartbleed

Course Roadmap

Welcome to Security 542: Web App Penetration Testing and Ethical Hacking!

We'll begin with an introduction to the web, talk about information gathering and virtual host discovery, review HTTPS, and then wrap up with target profiling.



SEC542 Workbook: Virtual Host Discovery

Please go to Exercise 1.2 in the 542 Workbook.

Course Roadmap

- **Section 1: Introduction and Information Gathering**
- Section 2: Content Discovery, Auth, and Session Testing
- Section 3: Injection
- Section 4: XSS, SSRF, and XXE
- Section 5: CSRF, Logic Flaws, and Advanced Tools
- Section 6: Capture the Flag

INTRODUCTION AND INFORMATION GATHERING

1. Why the Web?
2. Application Assessment Methodologies
3. Web Application Pen Tester's Toolkit
4. Interception Proxies
5. Exercise: Configuring Interception Proxies
6. Open Source Intelligence (OSINT)
7. Virtual Host Discovery
8. Exercise: Virtual Host Discovery
9. **HTTP Syntax and Semantics**
10. HTTPS and Testing for Weak Ciphers
11. Exercise: Testing HTTPS
12. Target Profiling
13. Exercise: Gathering Server Information
14. Summary
15. Bonus Exercise: Testing and Exploiting Heartbleed

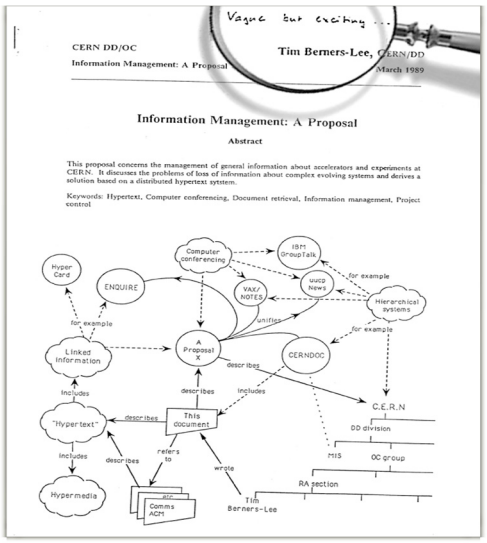
Course Roadmap

Welcome to Security 542: Web App Penetration Testing and Ethical Hacking!

We'll begin with an introduction to the web, talk about information gathering and virtual host discovery, review HTTPS, and then wrap up with target profiling.

In the Beginning…

- On March 12, 1989: Tim Berners-Lee submitted a distributed computing proposal that became the World Wide Web
- His boss at CERN responded: "Vague, but exciting...."[1] and gave the go-ahead to proceed with the project



In the Beginning…

March 12, 1989 was a watershed moment in the history of the internet. On that day, the World Wide Web was born. Mike Sendall, Tim Berners-Lee's boss at CERN (the European Organization for Nuclear Research; the name is Conseil Européen pour la Recherche Nucléaire in French) responded to Berners-Lee's submission by writing "Vague, but exciting…" on the submission's cover page.

The concept of hypertext existed and was used before 1989; HTML is a logical (and originally simpler) ancestor to Generalized Markup Language (GML, including Standard Generalized Markup Language, or SGML). Older related technologies included Memex and later Project Xanadu. You can read more about the history of hypertext at https://sec542.com/2a.

Simply put, HTML renders web data, and HTTP transfers it. The web began to be used outside of CERN beginning in 1991 with the first release of the Hypertext Transfer Protocol (HTTP), which was the revolutionary part of the web, marrying a markup language supporting hyperlinks with a protocol designed specifically to transfer it via TCP/IP.

References:

[1] cern.info.ch - Tim Berners-Lee's proposal, https://sec542.com/74

[2] Ibid.

Set the Wayback Machine to 1989

- 1989 was an unfortunate time for the birth of a new protocol:
  - The internet was still mostly trusted and mostly flat
    - Every “internal” TCP/IP system in most organizations was publicly accessible and unfiltered
      - Including desktops, printers, and more
  - There were no commercial firewalls, only packet filters
    - aka first-generation firewalls
  - Plaintext protocols were used via the public internet:
    - Telnet, FTP, and the infamous r* commands (rlogin, rsh, etc.) were in common usage
  - Kerberos existed for use on LANs/WANs or between trusted organizations

Set the Wayback Machine to 1989

The ARPANET and MILNET of the '70s and '80s became known as the internet. The ARPANET was formally decommissioned in 1990, but the term ”internet" had been in use since well before then. We will use the term ”internet" to describe the past and current network.

The internet was designed to be globally unfiltered, and end-to-end connectivity was expected. RFC 1918 addresses, Network Address Translation (NAT), stateful firewalls, and even proxies did not exist yet. The first commercial firewall (DEC SEAL) would not be available until 1991.

In 1989, the internet was still considered (mostly) trusted. The famous Morris Worm of 1988 began to change that and launched a wave of firewall innovation. Circuit-level firewalls (called second-generation firewalls) and proxies (called third-generation firewalls) followed around 1991. The modern stateful packet inspection firewall was first created by Checkpoint in 1994.

The only commonly available encryption in use back then was Kerberos, and that was used for LANs/WANs and over the internet between trusted organizations.

HTTP Semantics vs. Syntax

Early HTTP versions/RFCs presented a monolithic protocol

More recently, developers realized the benefit of distinguishing HTTP's semantics from its syntax:

*"**Semantics** describe the concept of request and response exchanges including: methods, status codes, header fields (metadata) and bodies (payload). **Syntax** describe how to map semantics to bytes on the wire."*[1]

How this most directly impacts us is to realize that changes in HTTP since 1.1 have largely focused on how to optimize the delivery of HTTP on the wire

- SPDY, HTTP/2, QUIC, and HTTP/3 - Syntax rather than Semantic changes

HTTP Semantics vs. Syntax

Unless you have spent a fair amount of time dealing with HTTP (especially SPDY, HTTP/2, QUIC, HTTP/3) you'd be forgiven for glossing over the differences between HTTP semantics and HTTP syntax. The short story is that the overwhelming majority of changes to HTTP since HTTP/1.1 have been in the way HTTP is communicated over the wire, which is syntax. The semantics have continued to persist largely unadulterated.

Reference:

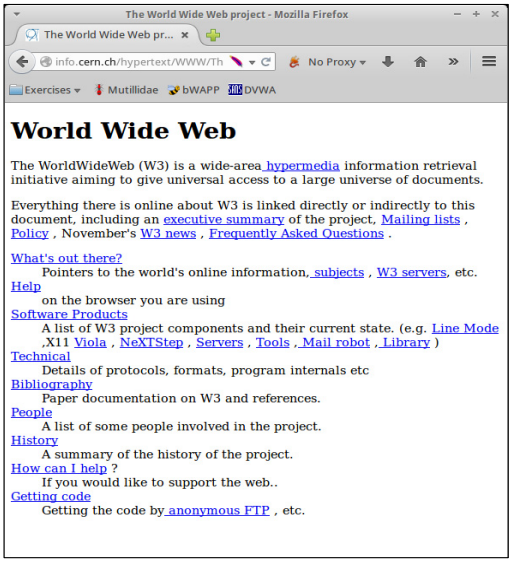
[1] HTTP/3: From root to tip, https://sec542.com/5t

The Web, v1

The World Wide Web (W3) was designed to be the equivalent of an anonymous FTP server that supported hyperlinks:

- All data was public
- No authentication
- No encryption
- Each download was sessionless
- One server == one web server: no virtual hosts

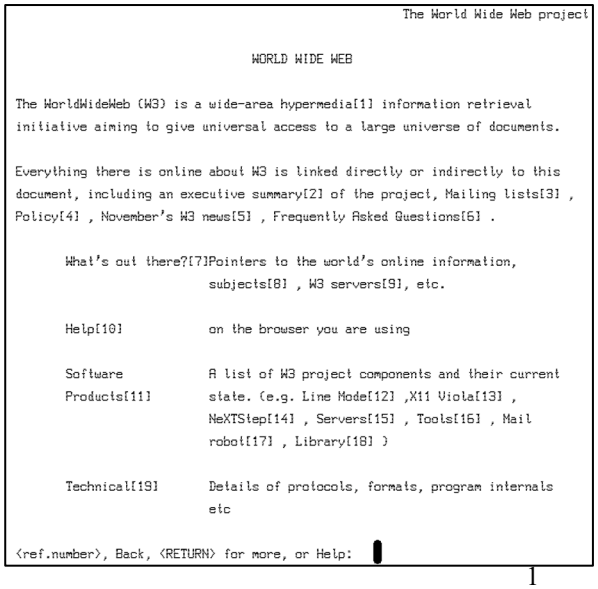
See notes for details on HTTP 0.9 and HTTP 1.0 (described by RFC 1945)



The Web, v1

This screenshot shows the first website and is archived at CERN at http://info.cern.ch/hypertext/WWW/TheProject.html.

The first web browsers were text-based; hyperlinks were chosen by entering the associated number shown in brackets after the link. Here is a screenshot of a modern emulator displaying the first website.



1

Reference:

[1] The World Wide Web project, https://sec542.com/75

HTTP 0.9 (which was a retroactive name) was the first version of HTTP and supported a simple GET method.

- The client GET request was a single line, terminated by CRLF
- No other methods or client headers were supported
- No support for virtual hosts
- Returned HTML only
- One GET == one TCP connection
- Many modern web servers (including Apache and Nginx) still support HTTP/0.9 by default

There is no RFC for HTTP/0.9. The (quite sparse) document found at the following site describes what the protocol was like in 1991: https://sec542.com/3h

The protocol is quite simple; here is a complete request: **`GET /example.txt<CR><LF>.`**

There are no other headers, and the HTTP version number was not sent until the release of HTTP/1.0. Modern servers that support HTTP/0.9 now accept the HTTP version number, as the following example demonstrates.

You can try HTTP/0.9 and experience the glory of the web as it was 25 years ago.

Open a terminal in your Sec542 Linux VM and type the following:

```
$ echo -e "GET / HTTP/0.9\r\n" | nc localhost 80
```

The "-e" flag in echo “enables interpretation of backslash escapes” (see “man echo”), meaning it sends a carriage return (\r) and newline (\n, aka line feed). The echo command sends the literal characters if the "-e" flag is omitted.

HTTP/1.0 was released in May 1996 (the first formal HTTP standard)

- Supports multiple client headers as well as multiple methods beyond GET:
    - GET and HEAD are described as "safe methods" (they do not modify content)
- POST, PUT, and DELETE are also described
- Non-HTML server responses supported:
    - Images, text, binary files, and so on
- Still no support for virtual hosts
- Again, one GET == one TCP connection

HTTP/0.9 servers sent HTML only. HTTP/1.0 server response objects could include any type of data, such as text, images, binary files, and so on. This meant that the Hypertext Transfer Protocol had been extended to transfer nonhypertext data.

RFC 1945 describes HTTP/1.0 and formalized the misspelling of "referrer," then and forever. It had been spelled incorrectly during the development of HTTP/1.0, by Phillip Hallam-Baker:

> *"It's like when I did the referer field. I got nothing but grief for my choice of spelling. I am now attempting to get the spelling corrected in the OED since my spelling is used several billion times a minute more than theirs."*[1]

Try the following command in a Security542 Linux terminal to make a HTTP/1.0 request:

```
$ echo -e "GET / HTTP/1.0\r\n" | nc localhost 80
```

This minimal request is the same as an HTTP/0.9 request, except the version number is HTTP/1.0.

Reference:

[1] Phillip Hallam-Baker "referer" https://sec542.com/76

HTTP/1.1

- RFC 2616 first described HTTP/1.1
  - Released in June 1999
  - Superseded by other HTTP/1.1 RFCs in 2014
- Added virtual host support
  - Host header is mandatory for HTTP/1.1 client requests
- Allows "persistent" connections
  - Multiple requests via one TCP/IP socket pair
- Added the OPTIONS method
- Better support for caching, proxies, and compression

HTTP/1.1

For many years, we looked to RFC 26161 for details about HTTP/1.1. However, in June 2014, the RFCs 7230,2 7231,3 7232,4 7233,5 7234,6, and 72357 replaced RFC 2616 and other supporting RFCs (see the links at the bottom of the page).

This command sends an HTTP/1.1 request but takes ~5 seconds to complete. Try it in a Security542 Linux terminal to see for yourself:

$ echo –e "GET / HTTP/1.1\r\nHost: localhost:80\r\n\r\n" | nc localhost 80

The connection stays open for a short period of time because it is persistent by default. To close the connection and retrieve results immediately, add a “Connection: close” header:

$ echo –e "GET / HTTP/1.1\r\nHost: localhost:80\r\nConnection: close\r\n\r\n" | nc localhost 80

References:

[1] RFC 2616, https://sec542.com/77

[2] RFC 7230, https://sec542.com/1s

[3] RFC 7231, https://sec542.com/1t

[4] RFC 7232, https://sec542.com/1u

[5] RFC 7233, https://sec542.com/1v

[6] RFC 7234, https://sec542.com/61

[7] RFC 7235, https://sec542.com/6o

HTTP/2

- The primary focus of the HTTP/2 update is achieving faster performance by updating the way HTTP works
- HTTP/2 will bring significant changes to HTTP:
  - **Binary protocol**
    - Better compression capabilities
    - Reduced overhead and complexity when parsing
  - **Push Promise** - Web server can send the client content before it is even requested
  - **Multiplexed not Pipelined** - One TCP connection per origin simultaneously performing multiple Requests/Responses
  - **HPACK** - Protocol designed for HTTP/2 Header Compression:
    - Header data is binary, too
  - HTTP/2 does **NOT** require encryption, but implementers might

HTTP/2

HTTP 2.0, which is often written as HTTP/2, is described by RFC 7540.[1] The primary push for HTTP/2 is to increase the performance of HTTP. Given the ubiquity of HTTP, anything that can be done to increase its performance could have significant impacts. HTTP/1.1 was written in 1999, well in advance of the explosion of and penetration of major video streaming and proper social networking. (For example, Facebook wasn't founded until 2004.)

HTTP/2 focuses on improving performance by making changes to the way HTTP behaves on the wire. A major takeaway from that statement is that web applications and their development will not necessarily be impacted. The impact is primarily on the browsers and the web servers, the endpoints responsible for the connection, and also intervening infrastructure. Significant changes come with HTTP/2.

- HTTP/2 will employ binary framing,[2] which means on the wire it will not be our old familiar plaintext. In addition to a more efficient delivery mechanism, parsing the binary data stream will be handled in a more objective fashion than the subjective text parsing for HTTP/1.x.
- Support for Push Promise[3] will allow the web server to send content before the client has even requested it. Classic HTTP would have me request an initial page and then, based on the data sent to me, my browser would perform (potentially many) subsequent requests. With Server Push, the web server could go ahead and shoot me data that it knows my browser will be requesting, in advance of it having even been requested.
- HTTP/2 will be multiplexed[4] rather than pipelined like HTTP/1.1. Pipelining was introduced with HTTP/1.1 as a means to address a major performance issue with HTTP/1.0. The issue was that if in HTTP/1.0 my page employed 50 different resources (images, CSS, and such) then the browser would initiate 50 different requests. Ouch. HTTP/1.1, through pipelining, attempted to address this problem by allowing multiple HTTP requests to be sent over the same TCP connection. The performance impact of pipelining in HTTP/1.1 was dramatic over HTTP/1.0. However, inefficiencies still exist.

Even though multiple HTTP requests can be sent, the browser still has to receive and process responses in order. You might have experienced this. Ever hit a web page that seemed to be taking a really long time to render all the graphics? Suddenly one graphic loads, and it is as if someone flipped a switch because all the images immediately load. The response traffic of that one large, high-resolution image was clogging up the pipeline. There have been different hacks to handle this over the years, but HTTP/2 does away with the pipeline and instead allows simply having all HTTP traffic travel over one TCP connection. With HTTP/2, requests can be sent in parallel, and the responses can be received and parsed in parallel as well.

HTTP headers will be compressed via HPACK (RFC 7541)[5] to reduce the performance hit of sending the slew of largely unchanging HTTP headers with every single request in a bloated plaintext format. HPACK is an additional protocol developed alongside HTTP/2.

All HTTP/2 data is not encrypted by default.[6] Many people are still under the mistaken impression that HTTP/2 brings with it HTTPS everywhere. Although there were discussions in the working group about trying to force HTTP/2 to employ ubiquitous encryption, the standard does not include this as a requirement.

References:

[1] RFC 7540 HTTP/2, https://sec542.com/1y

[2] Ibid.

[3] Ibid.

[4] Ibid.

[5] RFC 7541 HPACK, https://sec542.com/1z

[6] HTTP/2 FAQ, https://sec542.com/3s

QUIC - HTTP/3

QUIC (Quick UDP Internet Connections), defined in RFC 9000[1], is a protocol designed by Google that undergirds HTTP/3 and offers significantly faster over-the-wire communications

QUIC uses *__UDP__* ports 80 and 443

- Both are encrypted, but port 80 does not verify the server certificate[2]

QUIC is essentially HTTP/2 over UDP:

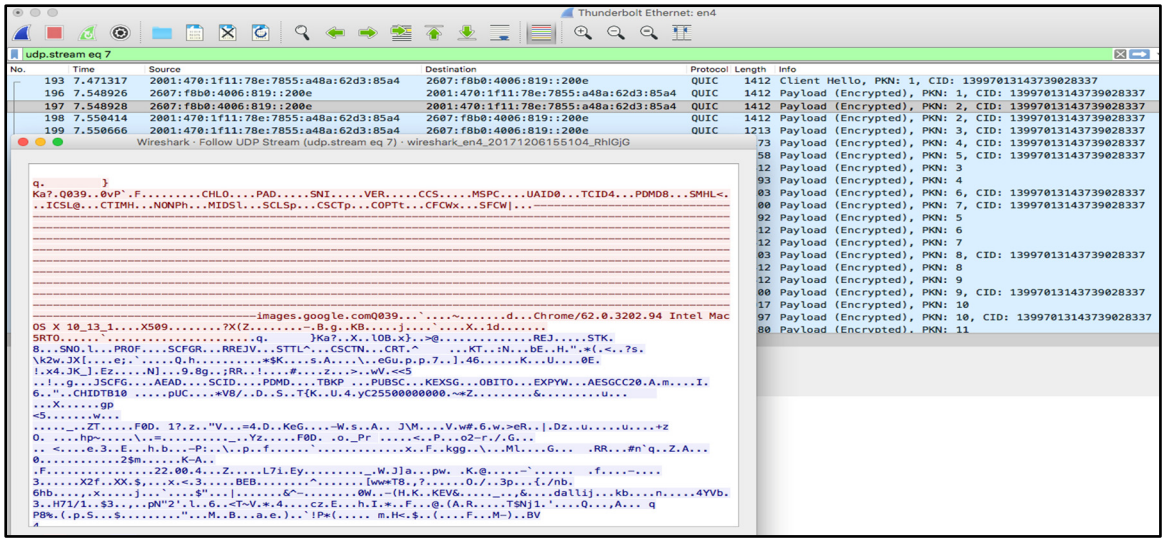
- *QUIC is a new transport which reduces latency compared to that of TCP. On the surface, QUIC is very similar to TCP+TLS+HTTP/2 implemented on UDP. Because TCP is implemented in operating system kernels, and middlebox firmware, making significant changes to TCP is next to impossible. However, since QUIC is built on top of UDP, it suffers from no such limitations.*[3]
- By default: Chrome will use QUIC for QUIC-enabled sites, including Google sites

Tool support is *very* limited

- As of this course's publication, neither ZAP nor Burp Suite supports QUIC

QUIC - HTTP/3

QUIC represents a significant blind spot for most organizations (and penetration testers). The newly defined standard will be the protocol employed by the HTTP/3 standard, which is still a draft RFC[4].



References:

[1] QUIC RFC: https://sec542.com/ag

[2] https://sec542.com/78

[3] QUIC: https://sec542.com/79

[4] HTTP/3: https://sec542.com/ah

HTTP Semantics

Understanding the various aspects of HTTP requests will be vital to our success

Similar initial L7 content (semantics) in HTTP/0.9, HTTP/1.0, HTTP/1.1, HTTP2, and QUIC/HTTP/3

- Primary differences related to network transmission of data (syntax)

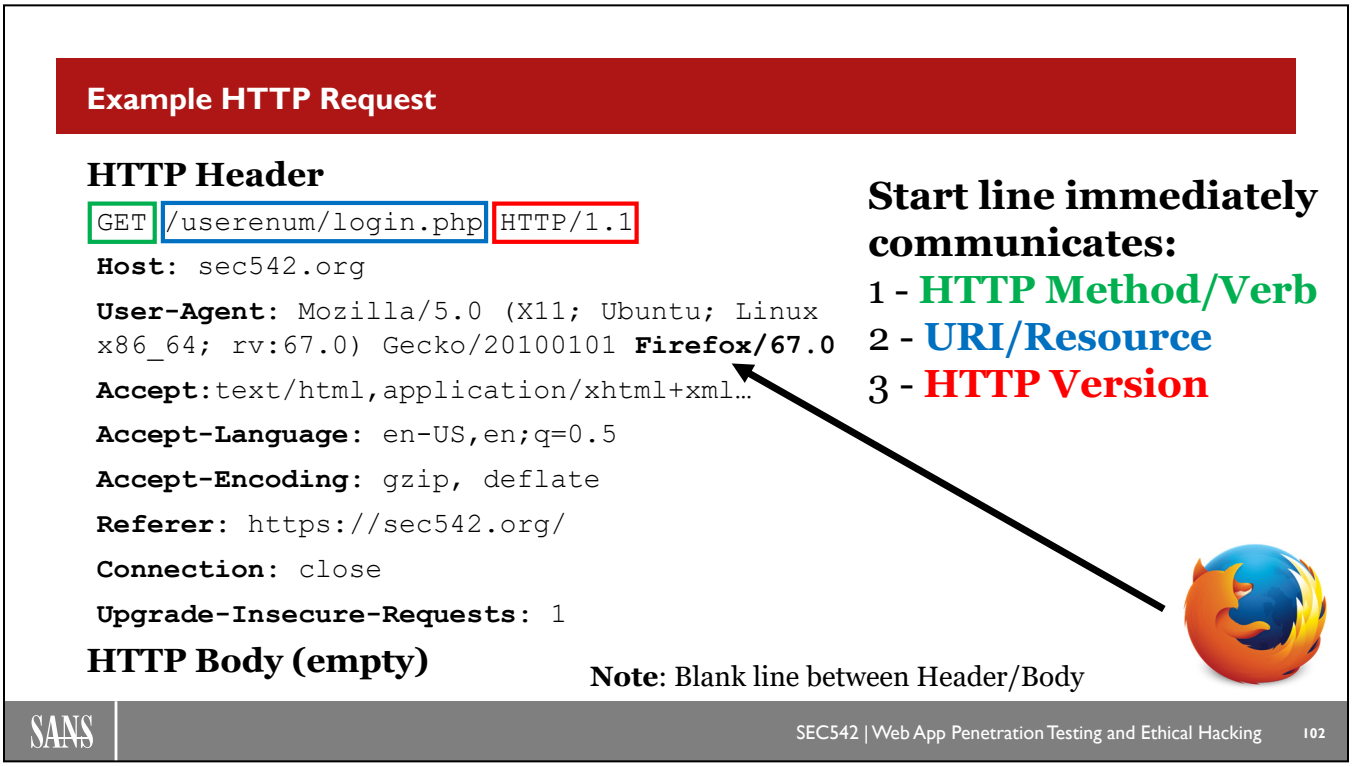
Crafting appropriate requests and inspecting responses—whether from the CLI with cURL or via our interception proxy, Burp, or ZAP—will be an essential task in our testing

Let's highlight the most important parts of HTTP requests and responses

HTTP Semantics

Whether manually created or through the use of tools that abstract us away from the details, successfully interfacing with HTTP requests will play an important role in any application penetration test. Though previous slides have highlighted nuances that vary depending on the version of HTTP employed, in truth, the primary ways in which we interface with the request will largely be the same, regardless of version. There is much more similarity than difference in the actual content slated for transmission via HTTP.

Though well-developed modern tools might seem to obviate the need for understanding this level of detail, this understanding, whether applied in every circumstance or not, will still serve you well in your application security testing. To that end, we will now dig into some of the most important parts of HTTP requests.



Example HTTP Request

A simple HTTP request was intentionally selected. Note that not only the values of the header fields, but the ordering of the fields, and even the presence of the fields themselves, can vary among browsers. Additionally, the context of the request will also impact the headers included.

GET /userenum/login.php HTTP/1.1

Host: sec542.org

User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:67.0) Gecko/20100101 Firefox/67.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Referer: https://sec542.org/

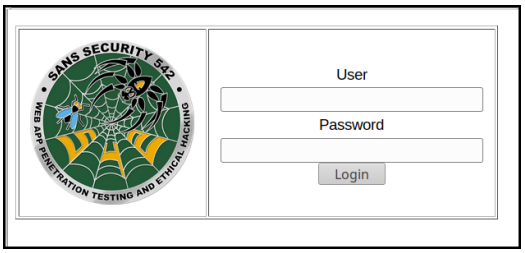
Connection: close

Upgrade-Insecure-Requests: 1

Example HTTP Response

Header

```
HTTP/1.1 200 OK
Date: Sun, 23 Jun 2019 11:46:11 GMT
Server: Apache/2.4.29 (Ubuntu)
Vary: Accept-Encoding
Content-Length: 1061
Connection: close
Content-Type: text/html; charset=UTF-8
```



Body

HTML, JS, CSS in notes

https://sec542.org/userenum/login.php

Example HTTP Response

```
HTTP/1.1 200 OK
Date: Sun, 23 Jun 2019 11:46:11 GMT
Server: Apache/2.4.29 (Ubuntu)
Vary: Accept-Encoding
Content-Length: 1061
Connection: close
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html PUBLIC "-//W3C//DTD html 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<script>
function setFocus(){
document.myform.user.focus();
}
</script>
<meta http-equiv="content-type" content="text/html;charset=ISO-8859-1">
<title>User Account Harvesting</title>
<style type="text/css" media="screen">
html, body {margin: 0; padding: 0; height: 100%;}
```

```
body {font: 14px/1.7 Verdana, Geneva, Arial, sans-serif;}
#content {
text-align: center;
position: absolute;
top: 50%;
left: 50%;
height: 150px;
margin-top: -75px;
width: 400px;
margin-left: -200px;
}
#content span {font-weight: bold;font-size: 24px}
</style>
</head>
<body onLoad="setFocus()">
<table border=1 cellpadding=10 id=content>
            <tr>
                        <td>
                                    <img width=150 height=150
src=/images/542.png>
                        </td>
                        <td>
                                    <form method=post action=login.php
name=myform>
User<br><input type=text name=user size=30><br>
                                    Password<br><input type=password
name=pass value="" size=30><br>
                                    <input type=submit name=button
value=Login>
                        </td>
            </tr>
</table>
</body>
</html>
```

HTTP Request: Methods/Verbs

First part of the first line of an HTTP request identifies the HTTP verb or request method:

Key characteristics of methods include whether method is safe or idempotent

A **safe** method operates as read-only and doesn't alter the server state:[1]

- Safe methods: GET, HEAD, OPTIONS
- Unsafe methods: POST, PUT, DELETE

Repeated submissions of an **idempotent** method yield same end server state:[2]

- Idempotent methods: GET, HEAD, PUT, DELETE, OPTIONS
- Non-idempotent methods: POST, PATCH

HTTP Request: Methods/Verbs

The method is the first piece of an HTTP Request. The following are commonly reference HTTP methods:

GET

HEAD	**CONNECT**
POST	**OPTIONS**
PUT	**TRACE**
DELETE	**PATCH**

All general-purpose servers are expected to support GET and HEAD request methods. All other methods are considered, strictly speaking, optional.

DELETE stands out as being somewhat odd when first considering idempotent methods.

> *To be idempotent, only the actual back-end state of the server is considered, the status code returned by each request may differ: the first call of a DELETE will likely return a 200, while successive ones will likely return a 404.*[3]

References:

[1] Safe - Mozilla Developer Network Glossary, https://sec542.com/5v

[2] Idempotent - Mozilla Developer Network Glossary, https://sec542.com/5w

[3] Ibid.

HTTP Request Methods: GET

No payload (body) is sent with an HTTP GET request

- Hails from the days of simply GETting static text from servers

All necessary information shared via HTTP headers

- Exclusive use of HTTP headers for data interchange means URI query parameters employed for sending most client-side information

Even if transmitted exclusively over HTTPS, sensitive information being sent this way has security implications

As RFC 3986 for URI notes:

> *"URIs are frequently displayed by browsers, stored in cleartext bookmarks, and logged by user agent history and intermediary applications (proxies)"*[1]

HTTP Request Methods: GET

The most commonly encountered and first method to learn is the GET method. GET along with HEAD are, from an RFC perspective, required to be supported by web servers. As testers, we like GETs because of their tremendous simplicity from a scripting and automation standpoint. This simplicity stems from the fact that GETs do not employ an HTTP body.

The lack of body means that any custom or user-specific data to be submitted must be included exclusively in the headers. Typically, this means that the data to be submitted is sent via URI.

Reference:

[1] RFC 3986 - Uniform Resource Identifier, https://sec542.com/5u

HTTP Request: GET and the URI

Lack of HTTP body usage means all data interchange must be communicated in the HTTP Headers

HTTP GETs employ the HTTP URI for data transfer

POST Form Submission

GET Form Submission

HTTP Request: GET and the URI

Here we see the implications of having no HTTP body by which to send data.

Although today's forms most commonly employ the POST method, the default behavior if no method is specified is for the client to leverage a GET.[1] This behavior makes sense given the requirement that GETs are supported by web servers whereas POST are, strictly speaking, optional for web servers to support.[2]

Note: If during testing anything beyond "unreserved characters"[3] are employed, they would need to be URL-encoded accordingly. While tools such as Burp and ZAP typically will automatically encode this data by default, other tools (or custom scripts) might not follow expected behavior.

References:

[1] HTML 5.2: 4.10. Forms, https://sec542.com/6k

[2] RFC 7231 Section 4.1, https://sec542.com/6l

[3] RFC 3986 Section 2.3, https://sec542.com/6m

HTTP Request: URI Query Parameters

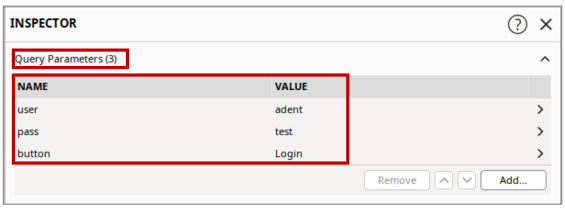
HTTP URI query parameters offer an obvious injection point into target applications

Per the RFC for URIs, the query "is indicated by the first question mark (?)"[1]

Identifying parameter key value pairs can be straightforward in some cases

user=adent&pass=test&button=Login

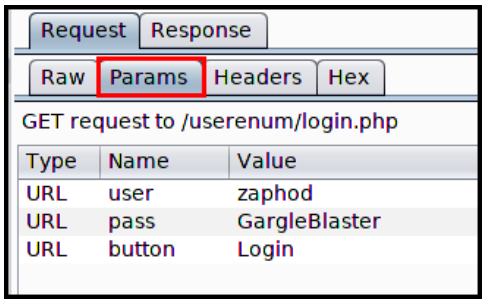
However, query parameters can present in a variety of formats



HTTP Request: URI Query Parameters

Parameters represent a key application injection point and so need to be identified. While tools like Burp will attempt to automatically identify query parameters, a little understanding can go a long way toward helping identify them on your own.

The initial question mark in a URI is expected to offset the start of the query parameters. While conventionally it is common to see schemes such as that shown in the slide, less obvious representations are also commonly employed.



Reference:

[1] RFC 3986 - Uniform Resource Identifier, https://sec542.com/5u

HTTP Request Methods: POST

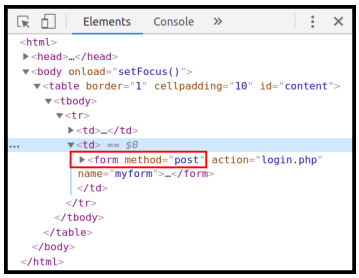
HTTP POST - second most common HTTP Request method

Unlike GET, HTTP POST requests employ data in the HTTP message body for data interchange

- While simple, this represents a significant difference between GET and POST

POSTs use of the HTTP body means this data will not be readily apparent to standard users of browsers, or logged by default in most web servers

HTML forms: incredibly common use-case of POST submission



HTTP Request Methods: POST

POST methods are extremely common to see employed. The use of an HTTP body presents significant advantages beyond HTTP GETs only leveraging the HTTP header. Having a body of variable length allows for much more robust client-side communications.

While simple HTML form submissions are commonly seen, more complex and robust submissions are also possible. The Content-Type request header indicates the type of payload data being transmitted via the body. In a form this might commonly be: **`Content-Type: text/plain`** or **`Content-Type: application/x-www-form-urlencoded`**. Another commonly employed Content-Type sent via POST is **`Content-Type: application/json`**,

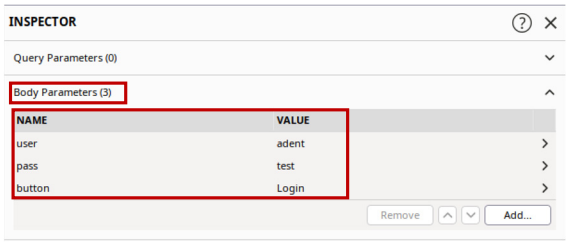
HTTP Request: POST Parameters

HTTP POST requests can include parameters beyond HTTP GETs URI query parameters

- Like URI parameters, these offer an obvious injection point

However, POST parameters are found in the payload of the HTTP request body

- Payload injections require updating the Content-Length header accordingly



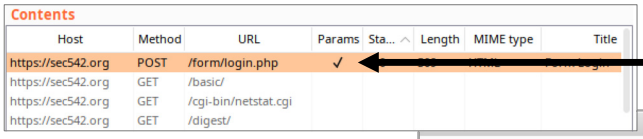
HTTP Request: POST Parameters

Parameters are not simply the domain of HTTP GETs and the URI. The POST method also can include parameters. Like their GET counterparts, the parameters found in POSTs also present an obvious and important application entry point that we will be targeting for input or injections-style payloads as we progress with testing.

Note: If during testing you alter parameters, be sure that either you manually or your tool/script automatically accounts for any changes in payload length and adjusts the Content-Length header accordingly.

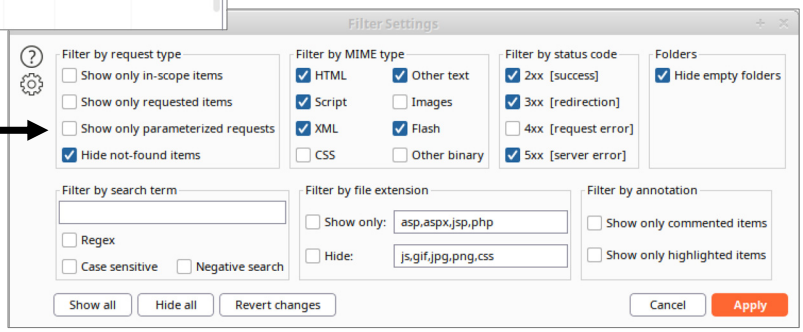
HTTP Request: Burp(ing) Parameters

Burp can make quick work of identifying the injection points available as HTTP request parameters



Most views showing individual requests include a **Params** column

The **Target->Site map** also allows for identifying just requests with parameters

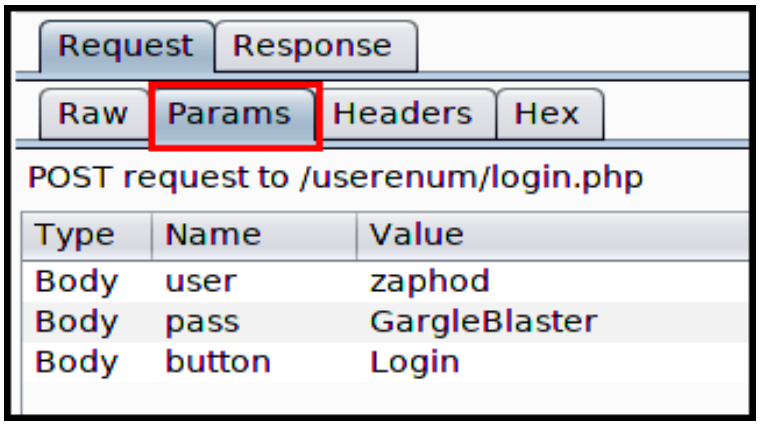


HTTP Request: Burp(ing) Parameters

Given their importance as application entry points, identifying request parameters is important. As shown previously, Burp will make it easy to identify parameters in an individual request via the Inspector pane.

However, having to look at each individual request would prove rather cumbersome. Thankfully, Burp makes it even easier by offering a sortable **Params** column when looking at a list of requests.

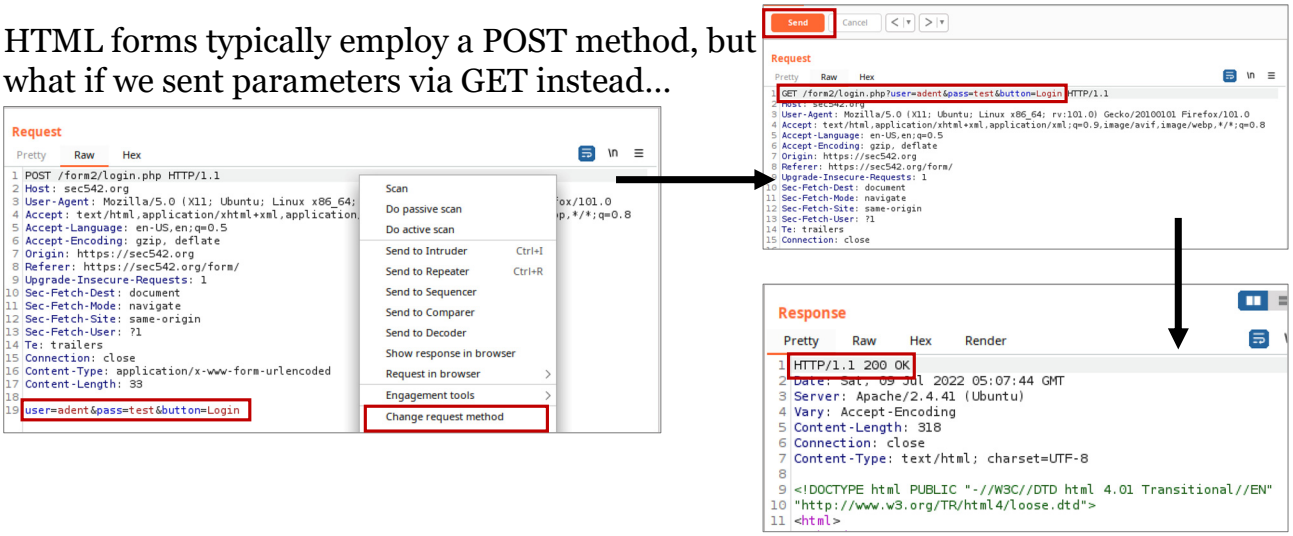
Further, one of the filter options under the site map allows for showing only those requests that include parameters, **Show only parameterized requests**.



Host	Method	URL	Params	Status	Length	MIME type
https://sec542.org	POST	/userenum/login.php	✓	200	1253	HTML
https://sec542.org	GET	/userenum/login.php		200	1253	HTML

HTTP Request: Verb Tampering and GET/POST Method Interchange

HTML forms typically employ a POST method, but what if we sent parameters via GET instead...



HTTP Request: Verb Tampering and GET/POST Method Interchange

Swapping one HTTP method for another or simply testing to see if unexpected methods are allowed can generally be considered verb tampering per the OWASP Web Security Testing Guide.[1]

One of the most significant examples of this comes in the form of what has, at times, been referred to as method interchange and involves sending parameters via a method other than expected by the application. Most HTML forms, for instance, will specify POST as their method, but would the submission via a GET instead be supported. If allowed and unanticipated by developers, then this could yield something interesting to us as testers.

Burp Suite makes testing for method interchange fairly simple. Identify a POST request that includes parameters. Right-click and select "Send to Repeater." In Repeater, first immediately click Go to ensure that the same request will work via Repeater. If so, now right-click and select "Change request method." After Burp changes the request accordingly, simply hit Go and see how the server responds.

The Burp documentation has this to say about "Change request method": "For requests, you can automatically switch the request method between GET and POST, with all relevant request parameters suitably relocated within the request. This option can be used to quickly test the application's tolerance of parameter location, e.g., to bypass input filters or fine-tune a cross-site scripting attack."[2]

References:

[1] Test HTTP Methods (WSTG-CONF-06) – OWASP, https://sec542.com/5y

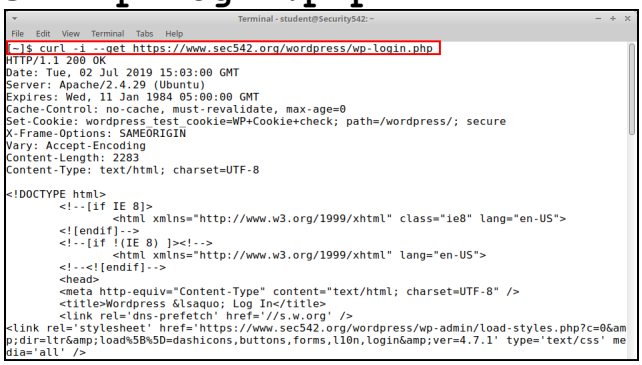
[2] Burp message editor, https://sec542.com/5x

HTTP Request Methods: HEAD

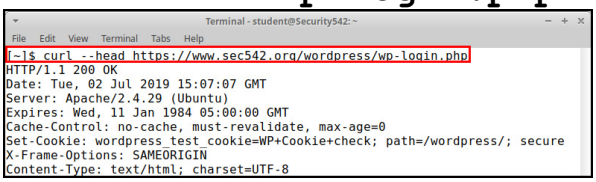
Safe HTTP request method that elicits server response without any associated response body

Only HTTP headers will be returned to a HEAD request

GET wp-login.php



HEAD wp-login.php



- Same headers; no payload data
- HEAD: very quick, useful for simple HTTP response header evaluations (e.g., Cookie without HttpOnly flag)

HTTP Request Methods: HEAD

HEAD is a request method that asks the server to return only a header, no body, regardless of whether a body would exist or not. While not terribly useful for browsing a website, if all we have need of knowing is information included in the HTTP headers, then this is sufficient, and seemingly innocuous.

HEAD can also serve you should you have need of a simple method to employ for ensuring a web server is still responsive.

cURL provides an easy way to craft head requests: simply use **`--head`** (or **`-I`**).

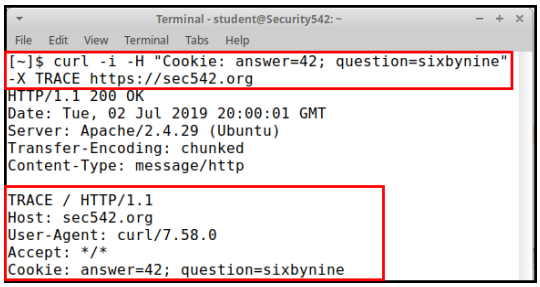
HTTP Request Methods: TRACE

Method designed for troubleshooting that should not be enabled in production

- TRACE support typically considered a finding on any assessment

If identified, Cross-Site Tracing (XST)[1] could possibly be exploited, though updated RFC guidance to user-agents on HTTP TRACE seeks to limit impact

- Unfortunately, RFC guidance is only as good as user-agents' compliance



Purpose: "TRACE allows the client to see what is being received at the other end of the request chain and use that data for testing or diagnostic information."[2]

HTTP Request Methods: TRACE

Cross-Site Tracing was discovered by Jeremiah Grossman of WhiteHat Security while researching methods of bypassing HttpOnly restrictions on cookies. TRACE was found to be a means of gaining access to cookie data without having to employ the more typical XSS route of JavaScript that was restricted by the HttpOnly flag.[3]

When RFC 2616 was superseded by RFC 7231 (among other RFCs), updated details regarding the potential abuse of TRACE were noted and the following guidance was offered:

> *A client MUST NOT generate header fields in a TRACE request containing sensitive data that might be disclosed by the response. For example, it would be foolish for a user agent to send stored user credentials [RFC7235] or cookies [RFC6265] in a TRACE request. The final recipient of the request SHOULD exclude any request header fields that are likely to contain sensitive data when that recipient generates the response body.*[4]

This would help mitigate exposure of sensitive information that an attack like XST targets. Unfortunately, as this is RFC guidance regarding the user-agents, the mitigation is only as good as the various user-agents' adherence to the guidance.

As seen in the screenshot above, cURL clearly doesn't abide by RFC 7231's MUST NOT regarding cookies.

Note: If you are trying to follow along via your VM, please note that the web server configuration was temporarily updated to support TRACE. Your VM, by default, has TRACE disabled and will, by default, result in an HTTP 405 Method Not Allowed.

References:

[1] Cross-Site Tracing, https://sec542.com/68

[2] RFC 7231 – TRACE, https://sec542.com/67

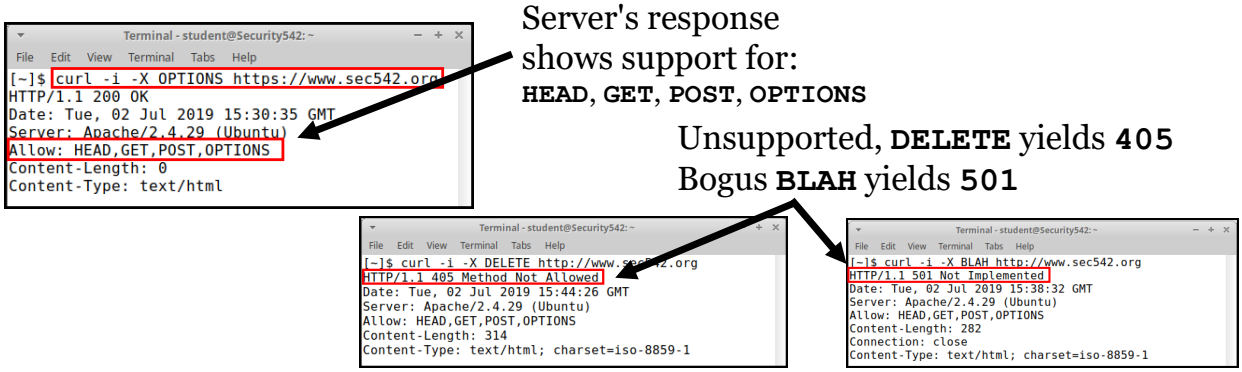
[3] Cross-Site Tracing, https://sec542.com/68

[4] RFC 7231 – TRACE, https://sec542.com/67

HTTP Request Methods: OPTIONS

Like playing the children's card game Go Fish with a 3 year old...*

Client sends an OPTIONS request and, if supported, server responds noting all supported HTTP methods



HTTP Request Methods: OPTIONS

The OPTIONS method allows for quickly asking servers to communicate supported methods. While GET and HEAD methods are expected to be supported, as the standards suggest support is required, the other methods need not be supported. If the receiving server supports the OPTIONS method, it responds with the list of supported methods in the **`Allow`** HTTP response header. We will explore validating method support later in the course, but consider that the server might be lying intentionally or configured such that all supported methods are not exposed via this response.

We again make use of the incredible tool cURL to demonstrate functionality.

First, we send a simple OPTIONS request to www.sec542.org:

```
$ curl -i -X OPTIONS https://www.sec542.org
```

`-i` indicates that response headers should be displayed.

`-X OPTIONS` is how we send the OPTIONS request header.

The response includes the Allow header and provides a list of supported methods. For giggles, we additionally show sending two requests with methods not listed as being supported. First, we send a DELETE request, which is a legitimate method, but not one listed as being supported.

```
$ curl -i -X DELETE https://www.sec542.org
```

The server responded with an HTTP status of **405 Method Not Allowed**.

Next, we send a request with the totally made up BLAH method.

In this case, the server responded differently because BLAH was an unknown method. Rather than the 405, we receive a **501 Not Implemented** status.

* For those not blessed with having experienced the joy of the card game Go Fish, it's a simple card game where your opponent tries to guess a card you might have in your hand that matches a card they hold. If the guess is successful, you will have to give your card to the opponent. Unsuccessful guesses require the guesser to have to "go fish" and pick up a card from the *sea* of cards on the table, hoping for a match. Three-year-old children (at least the author's) seem old enough to play, but often resort to simply asking, "Well what do you have?" to ensure they achieve Go Fish domination.

HTTP Request Methods: Other CRUD

HTTP methods commonly used, as in RESTful web services, to manipulate data stores

- When working with data stores, the acronym CRUD frequently pops up: **C**reate **R**ead **U**pdate **D**elete

Mapping these data store operations to HTTP methods:

Operation	Methods/Verbs	Notes
Create	POST (also PUT)	PUT less commonly used for creating in RESTful services[1]
Read	GET	
Update	POST, PUT, PATCH	In RESTful, PUT more often used for replacing resources[2]
Delete	DELETE	

Note: When an application will use POST vs. PUT vs. PATCH can vary widely

HTTP Request Methods: Other CRUD

The DELETE method shown in a previous slide is one we haven't discussed yet that would fall under the umbrella of methods commonly used for RESTful web services and APIs. The use-case of DELETE is quite likely obvious—the deletion of a resource—but to understand how other methods might be employed, let's consider CRUD. The acronym CRUD, which stands for Create, Read, Update, and Delete, is commonly used when dealing with persistent data stores. Naturally this acronym will come up in discussions of relational data stores, such as SQL databases, but also is certainly relevant for NoSQL stores, web services, and APIs.

GET and POST have previously been discussed and would most naturally fit under the Read and Create portion. POST could also be employed for Update operations, but is less commonly used for this purpose, especially when considering RESTful web services.[3]

You will not find the PATCH method in the HTTP/1.1 RFC 2616 or its more recent counterparts, most notably RFC 7231. Rather the PATCH method has its very own RFC, 5789.[4] The PATCH method, along with PUT, fits within the UPDATE portion of CRUD. In RESTful web services, PUT typically used to update resources via replacement as opposed to modification, which is more commonly associated with PATCH.[5]

While generally trying to understand how the HTTP methods fit within a CRUD model, as well as their common applicability in RESTful services, note that applications can, and do, certainly deviate from these general perceptions.

References:

[1] HTTP Methods for RESTful Services, https://sec542.com/65

[2] Ibid.

[3] Ibid.

[4] RFC 5789 - PATCH Method for HTTP, https://sec542.com/64

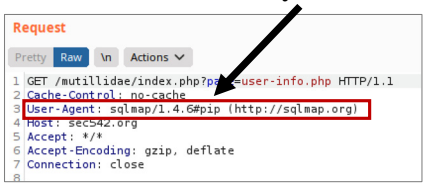
[5] HTTP Methods for RESTful Services, https://sec542.com/65

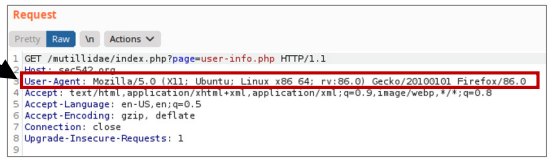
HTTP Request: User-Agent

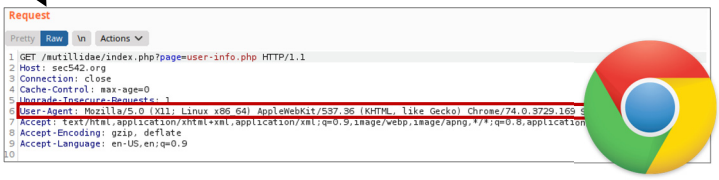
The User-Agent identifies the web client in an HTTP Request

Most often the User-Agent thought to be a browser

- Typically, is for ***normal*** users
- We are routinely ***abnormal***







Guess which attack tool we are using...

HTTP Request: User-Agent

One of the most well-known of HTTP Request headers is the User-Agent. Typically, this is considered by most folks to represent the browser. While it can identify the browser, it, strictly speaking, identifies the web client. In most, even technical circles, this would sound like a pedantic distinction with little difference. However, for our purposes, it is important to appreciate that web clients are not the same thing as browsers.

This becomes important as we interface directly with web servers and applications by means of tools outside of the browser. Whether it is sqlmap, nmap, or a custom script that leverages Python's urllib3, the HTTP user-agent might shine a flashlight on the fact that we are not a normal user interacting with the application. Depending on the engagement, this might be suboptimal.

Thankfully, we, like our tools, can customize what is sent in the user-agent. The easiest way to do this is to ensure all interaction with the application is proxied and have our proxy supply a user-agent string of our choosing.

HTTP Request: Referer

The Referer [sic] request header identifies the current location of the user-agent when a link is followed

Applications can be designed with a particular logic or workflow in mind

- One use of the Referer is to allow applications to check or possibly try to enforce a particular flow through the application

Note: If applications employ GETs with sensitive information, there is risk of inadvertent information disclosure via the Referer[1]

HTTP Request: Referer

The ignominiously misspelled Referer (rather than Referrer) request header is used to identify for the target server what page the user-agent was viewing when a link was clicked. This seems innocuous enough, but unfortunately, given the potential for sensitive information being included on the URL, it makes the use of the Referer field somewhat less innocuous.

Note: There are methods application developers can use to try to limit the exposure of the data included in the Referer field. Of common interest is not including the Referer field when following links to external domains.

Reference:

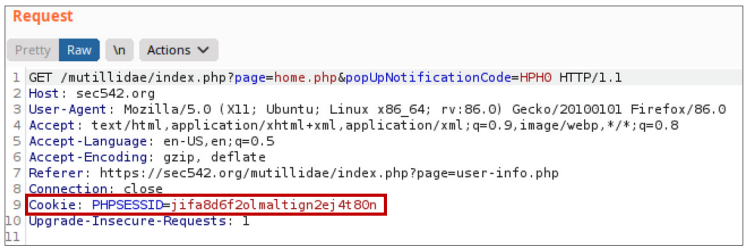
[1] Referer header: privacy and security concerns, MDN, https://sec542.com/66

HTTP Request: Cookie

The HTTP protocol provides stateless communications

- However, applications often have need of being stateful and tracking users' communications across multiple requests

Sessions will be discussed more fully later in the course, but one-way applications can establish/track sessions is through the use of HTTP cookies



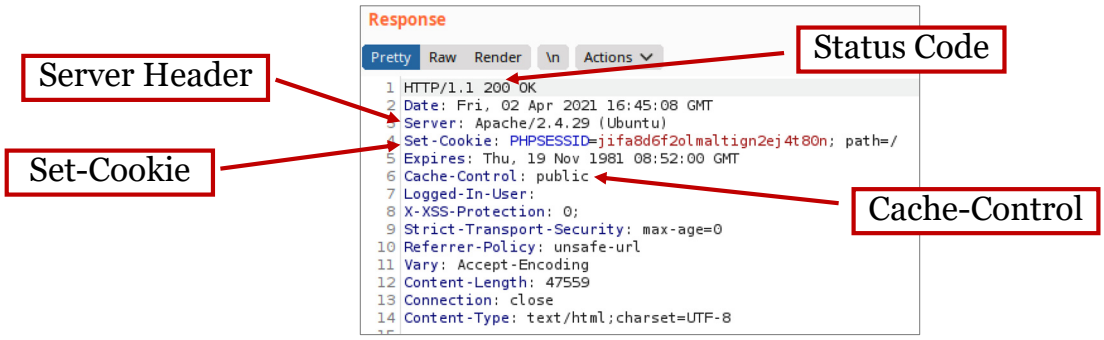
HTTP Request: Cookie

The Cookie HTTP request header will follow a server's response in which the Set-Cookie header was sent. The Set-Cookie response header is discussed next. Although cookies are used for more than just session management, that will be our main focus for them.

Cookies have long been a high-value target for attackers due to their association with sessions, particularly authenticated ones. Additional control flags have been created that can try to help protect cookies from inadvertent exposure and theft. These flags will be discussed in more detail in subsequent sections of this course.

HTTP Response Highlights

Response Element	Description
Status Code	Relates the outcome of a request processed by the web server.
Server Header	Typically, gives details about the software, and sometimes the operating system, of the web server.
Cache-Control Header	Request/Response header to direct how content is cached. Responses to GET and HEAD requests are cacheable.[1]
Set-Cookie	Sets a name/value pair that is stored by the browser.



Status Codes

HTTP status code presents a simple three-digit number that indicates the outcome of a request processed by the web server. Five categories encompass the more than 50 officially recognized status codes[2]. The categories are divided such that even if the full meaning of the status code and associated message are not fully understood, merely observing the first digit will provide insight.

1xx: **Informational** - Request received, continuing process
2xx: **Success** - The action was successfully received, understood, and accepted
3xx: **Redirection** - Further action must be taken in order to complete the request
4xx: **Client Error** - The request contains bad syntax or cannot be fulfilled
5xx: **Server Error** - The server failed to fulfill an apparently valid request[3]

4XX
400 Bad Request
401 Unauthorized
403 Forbidden
404 Not Found
5XX
500
501 Not Implemented
503
504 Gateway Timeout

Server Header

Like the User-Agent string, the Server string can be configured, but, also like the User-Agent, it often isn't. However, the Server string can often be impacted by application layer proxy servers.

Here's a simple use of curl for finding server strings:

```
$ curl --head sec542.org | grep -i server
```


And here's an alternative approach that employs a list of target URLs in a file:

```
$ curl -s --head -K servers.txt | grep -i server
```

Cache-Control Header

Browsers having the ability to cache server responses is an efficiency win. Why repeatedly send the exact same data back to subsequent browsers requests if the data has not changed and the browser indicates willingness to cache the data? General users can be safely oblivious to caching, but testers must be aware of circumstances where caching might preclude requests actually being delivered or the server delivering new content.

Whether caching will occur depends on both the client and the server indicating support via information in the HTTP Request and Response headers. Only certain request methods (GET and HEAD) are treated as cacheable. Likewise, a subset of HTTP response status codes will allow for caching. However, the most important element affecting cacheability is the Cache-Control header.

The Mozilla Developer Network provides this explanation of Cache-Control:

> *"The **Cache-Control** general-header field is used to specify directives for caching mechanisms in both requests and responses. Caching directives are unidirectional, meaning that a given directive in a request is not implying that the same directive is to be given in the response."*[4]

Set-Cookie Header

The Cookie HTTP request header will follow a server's response in which the Set-Cookie header was sent. Although cookies are used for more than just session management, that will be our main focus for them.

Cookies have long been a high-value target for attackers due to their association with sessions, particularly authenticated ones. Additional control flags have been created that can try to help protect cookies from inadvertent exposure and theft. These flags will be discussed in more detail in subsequent sections of this course.

References:

[1] Cacheable, MDN, https://sec542.com/5z

[2] Hypertext Transfer Protocol (HTTP) Status Code Registry, https://sec542.com/62

[3] Ibid.

[4] Cache-Control – HTTP, MDN, https://sec542.com/60

Course Roadmap

- **Section 1: Introduction and Information Gathering**
- Section 2: Content Discovery, Auth, and Session Testing
- Section 3: Injection
- Section 4: XSS, SSRF, and XXE
- Section 5: CSRF, Logic Flaws, and Advanced Tools
- Section 6: Capture the Flag

INTRODUCTION AND INFORMATION GATHERING

1. Why the Web?
2. Application Assessment Methodologies
3. Web Application Pen Tester's Toolkit
4. Interception Proxies
5. Exercise: Configuring Interception Proxies
6. Open Source Intelligence (OSINT)
7. Virtual Host Discovery
8. Exercise: Virtual Host Discovery
9. HTTP Syntax and Semantics
10. **HTTPS and Testing for Weak Ciphers**
11. Exercise: Testing HTTPS
12. Target Profiling
13. Exercise: Gathering Server Information
14. Summary
15. Bonus Exercise: Testing and Exploiting Heartbleed

Course Roadmap

Welcome to Security 542: Web App Penetration Testing and Ethical Hacking!

We'll begin with an introduction to the web, talk about information gathering and virtual host discovery, review HTTPS, and then wrap up with target profiling.

OWASP A02:2021-Cryptographic Failures

- Covers issues associated with unauthorized system or data access through misconfigurations, use of insecure components, or other problems with how cryptography is implemented
- Could affect components intended to protect data in transit, as well as data at rest
- Some common weaknesses and exposures are "CWE-259: Use of Hard-coded Password, CWE-327: Broken or Risky Crypto Algorithm, and CWE-331 Insufficient Entropy."[1]

A02:2021-Cryptographic Failures

Cryptography is used to protect the confidentiality and integrity of sensitive data that is stored, transmitted, and/or processed by the application and components of its technology stack.

Weaknesses in the cryptographic implementation that leads to the exposure of credentials (passwords, tokens, or session IDs) can lead to unauthorized access to the system or application. Consider the situation where an attacker gains access to password values stored as MD5 hashes. Modern GPU-based cracking systems are capable of cracking MD5 hashes at the rate of hundreds of billions, to trillions, of password guesses per second.

Some ciphers used with TLS encryption for HTTPS communication are susceptible to known plaintext attacks that may allow highly resourced attackers to recover data captured as encrypted information from the network.

Instances where encryption keys are hard-coded, automatically checked out, stored in publicly accessible locations, or are predictable values undermine the value of implementing encryption because attackers frequently can take advantage of these situations to decrypt encrypted data.

Reference:

[1] https://sec542.com/ai

HTTPS: Transmission Security for HTTP

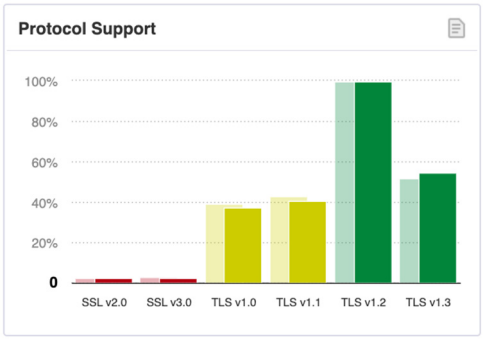
HTTP presents a predominantly cleartext protocol

- Recent versions of HTTP do include binary elements, but for speed not confidentiality of data

SSL/TLS employed to bring transmission security to HTTP:

- Final SSL version 3.0 released in 1996[1]
- Most popular version, TLS 1.2, debuted in 2008[2]
- RFC for TLS 1.3 released 2018[3]

Qualys SSL Labs' SSL Pulse[4]



HTTPS: Transmission Security for HTTP

Given that HTTP typically traverses untrusted systems and networks, methods to help ensure the security of this transmission is vital. HTTP is a cleartext protocol, so encryption of the communication in transit can provide much wanted/needed confidentiality for communications.

TLS 1.0 was based on SSL 3.0, which was the final version of SSL created by Netscape. Though the nomenclature has changed, TLS is merely the moniker for SSL since IETF took over responsibility from Netscape.[5] The most widely used version of SSL/TLS is currently TLS 1.2, even though it was released more than a decade ago and has some known security shortcomings.[6] TLS 1.3 represents a tremendous re-architecture and will take time to achieve needed server, client, and application support.

Note: Many instructors and this courseware will often typically refer to TLS as SSL, even though, strictly speaking, SSL should be considered an extinct protocol.

References:

[1] RFC 2246 - The TLS Protocol Version 1.0, https://sec542.com/6a

[2] RFC 5246 - The Transport Layer Security (TLS) Protocol Version 1.2, https://sec542.com/6c

[3] RFC 8446 - The Transport Layer Security (TLS) Protocol Version 1.3, https://sec542.com/6b

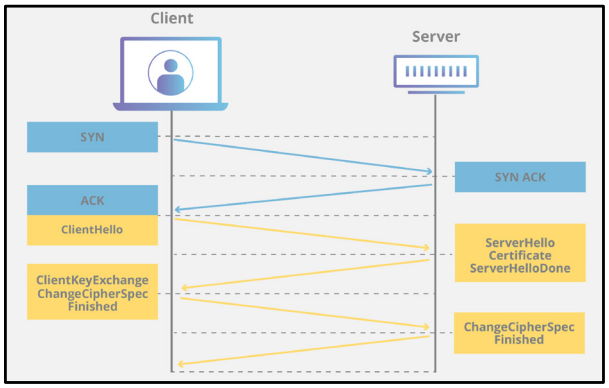
[4] Qualys SSL Labs - SSL Pulse, https://sec542.com/6d

[5] RFC 2246 - The TLS Protocol Version 1.0, https://sec542.com/6a

[6] Why Use TLS 1.3, Cloudflare, https://sec542.com/6e

HTTPS: SSL/TLS Handshake

Establishing a TLS-secured connection between client and server requires back-and-forth communication called the TLS Handshake:



- Determine version of TLS
- Negotiate cryptographic algorithms to use, cipher suite
- Verify trust for server's identity
- Create session keys for encryption of data in transit

Cloudflare: TLS Handshake[1]

HTTPS: SSL/TLS Handshake

Here are more complete details of the TLS handshake process:

> ***"The 'client hello' message:*** *The client initiates the handshake by sending a "hello" message to the server. The message will include which TLS version the client supports, the cipher suites supported, and a string of random bytes known as the "client random."*
>
> ***The 'server hello' message:*** *In reply to the client hello message, the server sends a message containing the server's SSL certificate, the server's chosen cipher suite, and the "server random," another random string of bytes that's generated by the server.*
>
> ***Authentication:*** *The client verifies the server's SSL certificate with the certificate authority that issued it. This confirms that the server is who it says it is, and that the client is interacting with the actual owner of the domain.*
>
> ***The premaster secret:*** *The client sends one more random string of bytes, the "premaster secret." The premaster secret is encrypted with the public key and can only be decrypted with the private key by the server. (The client gets the public key from the server's SSL certificate.)*
>
> ***Private key used:*** *The server decrypts the premaster secret.*
>
> ***Session keys created:*** *Both client and server generate session keys from the client random, the server random, and the premaster secret. They should arrive at the same results.*
>
> ***Client is ready:*** *The client sends a "finished" message that is encrypted with a session key.*
>
> ***Server is ready:*** *The server sends a "finished" message encrypted with a session key.*
>
> ***Secure symmetric encryption achieved:*** *The handshake is completed, and communication continues using the session keys."*[2]

Note that the exact flow and process of the handshake process is dependent on the version of SSL/TLS as well as the cipher suite selected. The flow shown in the slide is for version TLS 1.2, leveraging an RSA-based rather than Diffie-Helman key negotiation.

References:

[1] What Happens in a TLS Handshake? Cloudflare, https://sec542.com/6f

[2] Ibid.

HTTPS: Confidentiality++

Although SSL/TLS is primarily thought of as providing confidentiality, it actually does more

Negotiating a TLS tunnel between client/server requires selection of mutually supported crypto algorithms called a cipher suite

The cipher suite includes:

- A symmetric algorithm for bulk data encryption/confidentiality
- An asymmetric algorithm for key negotiation and endpoint authentication
- A hashing algorithm to ensure integrity

Implementation of individual ciphers can exhibit security flaws

- This is a key focus of HTTPS testing from our perspective

HTTPS: Confidentiality++

Most people think confidentiality when thinking of HTTPS, or really just cryptography in general. While confidentiality might well be front of mind, TLS affords, and HTTPS expects, more than mere confidentiality. Integrity of the data being transmitted is another vital property of HTTPS secured transmissions. Another piece of the HTTPS puzzle includes the authentication of endpoints involved in the communication. No good having secret communications that we are certain have not been modified inappropriately only to send them to an inappropriate recipient.

Achieving all of these separate properties requires not just one but a collection of cryptographic algorithms. This collection is referred to as a cipher suite. The HTTPS client and server must identify a cipher suite that both have in common. Preferable, and by convention, the intent is for TLS negotiation to select the most secure cipher suite that is mutually supported by client and server.

The cipher suite includes all the necessary cryptographic algorithms to achieve confidentiality of data, integrity of communications, and endpoint authentication. No one algorithm or cryptosystem is appropriate for achieving all of these goals. An asymmetric (a.k.a. public key) cryptosystem is used for both authentication of endpoints and also securely negotiating a session key that will be used with the symmetric cryptosystem for encrypting bulk data in transit. Strictly speaking, the asymmetric/public key cryptosystem could be used for confidentiality of data, but asymmetric algorithms are orders of magnitude slower than equivalent symmetric algorithms. So, asymmetric algorithms are instead used to securely negotiate/exchange a key, called a session key, that is used as the shared secret for the symmetric cryptosystem.

For reference, here are examples of algorithms across the three types of cryptosystems that comprise a cipher suite:

Asymmetric - RSA, DH, DSA, ECDH, ECDSA

Symmetric - DES, TDES/3DES, AES

Hashing - MD5, SHA, SHA256, SHA384

HTTPS: Public Keys/Certificates and Certificate Authorities

The asymmetric algorithm (a.k.a. public key) portion of the cipher suite depends on a public key for its role in TLS operations

While anyone can easily generate a public key, clients need to ensure the correct public key is being shared

- Verifiable certificates are the means by which the trustworthiness of public keys can be determined

Certificates must be digitally signed by an entity trusted by the clients of the communication

- Certificate authorities (CAs) are the (possibly) trusted entities that will generate and sign certificates

HTTPS: Public Keys/Certificates and Certificate Authorities

Public key or asymmetric cryptosystems play two vital roles with respect to HTTPS: origin authentication and key exchange/negotiation. The client needs to be able to determine that the public key being shared by the server is actually associated with the server/domain being communicated with. Servers communicate public keys as part of an X.509 certificate. The certificate provides a digitally signed means of sharing the public key and supporting information. The certificate being digitally signed allows the client to ensure the trustworthiness of the public key being shared. The certificate merely being signed doesn't ensure it is trustworthy; the client requires that the certificate has been signed by an entity the client has been configured to trust.

When initially attempting to proxy TLS communications with interception proxies (e.g., Burp or ZAP), TLS errors will be experienced. This is because the certificates are dynamically generated by Burp or ZAP, which is not implicitly trusted by the browser. For this reason, the course has a simple lab that walks through the process of importing the certificates employed by ZAP and Burp to ensure that those dynamically generated certificates do not cause the browser to throw errors.

WSTG-CRYP-01: Testing for Weak Transport Layer Security

"When information is sent between the client and the server, it must be encrypted and protected in order to prevent an attacker from being able to read or modify it."[1]

WSTG-CRYP-01: Testing for Weak Transport Layer Security

The purpose of WSTG-CRYP-01, as well as the larger WSTG Cryptography category, is primarily to ensure that confidential data remains confidential. WSTG-CRYP-01 seeks to ensure security by focusing on testing the efficacy of the transport encryption employed.

Reference:

[1] Testing for Weak Transport Layer Security, https://sec542.com/7a

HTTPS Testing: Versions and Cipher Suites

Only supporting the latest/greatest TLS 1.3 cipher suites would be ideal…

- If all clients could actually handle these communications

Servers must offer SSL/TLS versions that their clients can successfully negotiate

- For public-facing sites, this typically means offering TLS versions and cipher suites that are less than ideal

A vital component of HTTPS testing is assessing what servers will allow clients to successfully negotiate

The *correct* TLS versions and cipher suites to be supported varies

HTTPS Testing: Versions and Cipher Suites

In a perfect world, our servers would only allow for the most secure configurations. With respect to HTTPS, this currently would present as only allowing TLS 1.3 with currently supported cipher suites. If servers were actually configured this way, there would likely be tremendous collateral damage in the form of clients unable to successfully negotiate an HTTPS tunnel.

Unless the application owners have total control, and solid change and configuration management practices, then servers will necessarily have to support less-than-ideal versions of TLS and cipher suites. To do otherwise would mean denying service to likely substantial numbers of legitimate clients.

Even so, appropriate risk-informed decisions about TLS versions and cipher suites to support must be made. There is no one correct answer to what should be offered for clients to negotiate.

HTTPS Testing: Using Nmap to Evaluate HTTPS

- The Nmap NSE script ssl-enum-ciphers evaluates ciphers supported by an HTTPS server
- Categorizes cipher strengths with grades of A through F
- The grade levels might be out of step with current knowledge

HTTPS Testing: Using Nmap to Evaluate HTTPS

The command shown in the slide is:

```
$ nmap -p 443 --script=ssl-enum-ciphers www.sec542.org
```

Note that the version of Nmap installed on Debian systems via apt is usually outdated, and often contains an older version of the ssl-enum-ciphers script, which tests far fewer ciphers and has a simpler grading system of "strong" or "weak." The Security542 VM version of Nmap was compiled locally, to ensure we're using a more recent version.

The ssl-enum-ciphers page describes the grades:

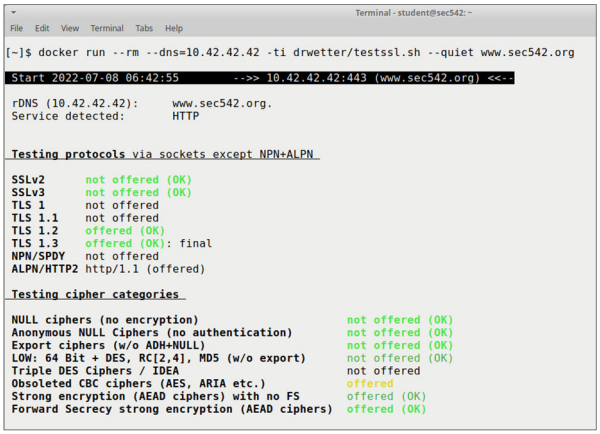
> *"Each ciphersuite is shown with a letter grade (A through F) indicating the strength of the connection. The grade is based on the cryptographic strength of the key exchange and of the stream cipher. The message integrity (hash) algorithm choice is not a factor. The output line beginning with Least strength shows the strength of the weakest cipher offered."*[1]

Reference:

[1] ssl-enum-ciphers NSE Script, https://sec542.com/7b

HTTPS Testing: Using testssl.sh to Evaluate HTTPS

- Developed by Dirk Wetter (@drwetter)
- Thorough reporting of HTTPS configuration
- Does not rely on OpenSSL
- Can run as bash script or docker container



HTTPS Testing: Using testssl.sh to Evaluate HTTPS

The testssl.sh tool is available for download from Github[1] as a bash script, or as a Docker container.

To run the bash script, after downloading it from Github:

```
$ ./testssl.sh <TARGET_HOSTNAME>
```

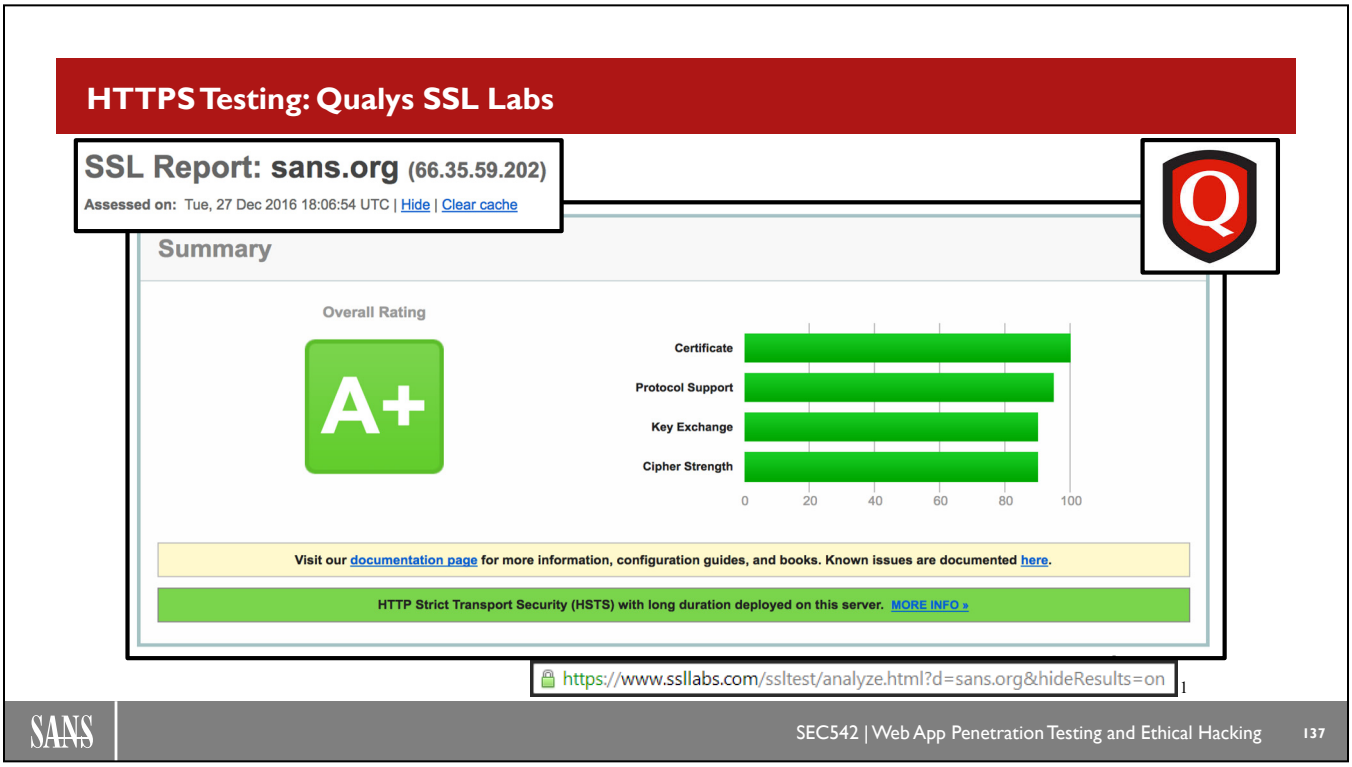
To run the script from Docker:

```
$ docker run --rm -ti drwetter/testssl.sh <TARGET_HOSTNAME>
```

Many tools rely on OpenSSL to gather details about the HTTPS configuration. If OpenSSL is not compiled to support certain protocols, then the tool may miss insecure configurations because OpenSSL's configuration does not support them. For example, most modern Linux implementations (including the latest versions of Kali Linux) have an OpenSSL version that does not support SSLv2, nor SSLv3. If the target web application is configured to support SSLv2/v3, then tools dependent upon OpenSSL will not identify and report these weaker protocols in use. The testssl.sh tool performs all of its checks without using OpenSSL, which enables it to reliably detect all available protocols and ciphers for a web application.

Reference:

[1] testssl.sh Project, https://sec542.com/ae



HTTPS Testing: Qualys SSL Labs

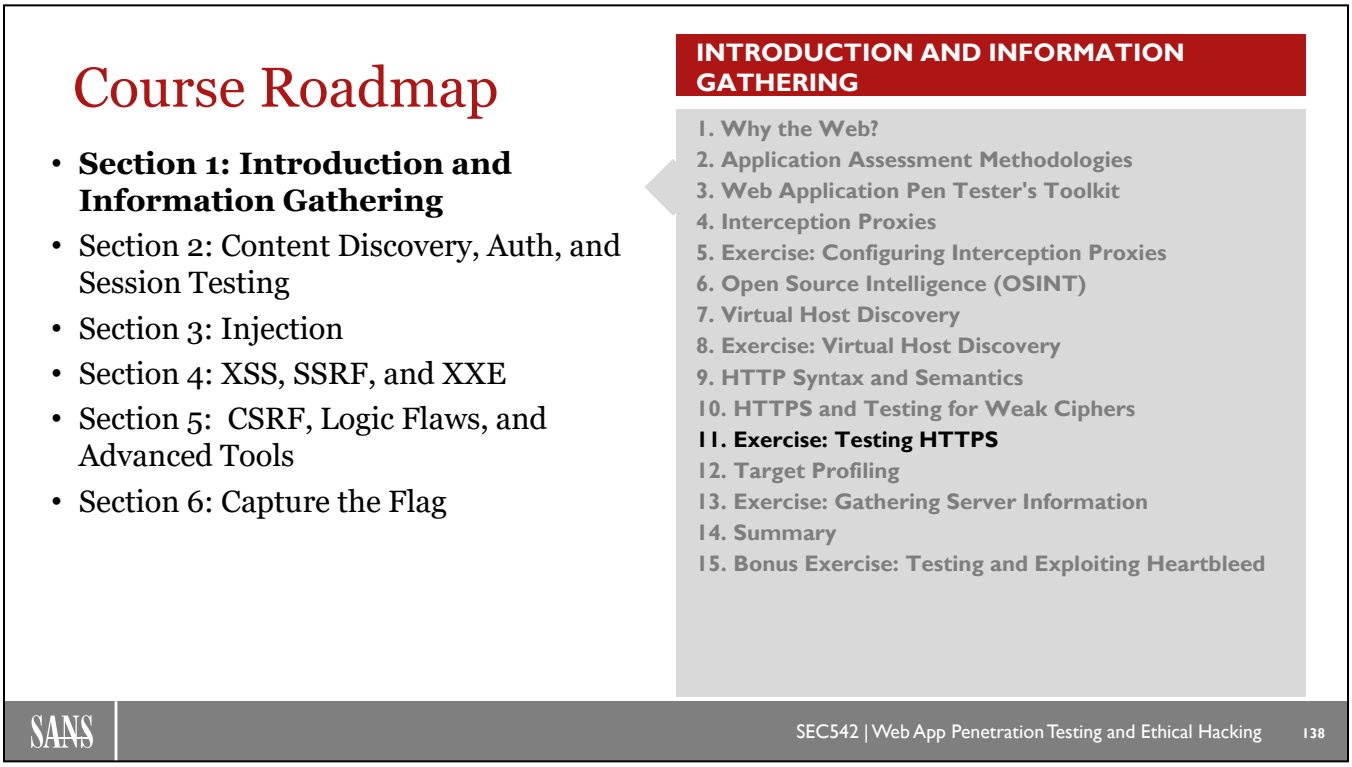
An outstanding public resource for evaluating SSL configurations is available from Qualys. SSL Labs is a free, publicly accessible site, that requires no registration of any kind. Simply navigate to its page and submit the site you would like to be assessed.

Much like with SSLDigger, many people appreciate the letter grade that Qualys provides. Beyond the letter grade, the report shows many items that go well beyond the simple SSL version and basic estimation of the strength of the cipher suite based on key length alone.

A report can be launched easily by simply supplying the domain as the value of a parameter, as seen here for sans.org: **https://www.ssllabs.com/ssltest/analyze.html?d=sans.org**. Note that if you would prefer not to potentially have the results seen publicly, then they should be submitted like this: **https://www.ssllabs.com/ssltest/analyze.html?d=sans.org&hideResults=on**

Reference:

[1] SSL Server Test Qualys, https://sec542.com/2z



Course Roadmap

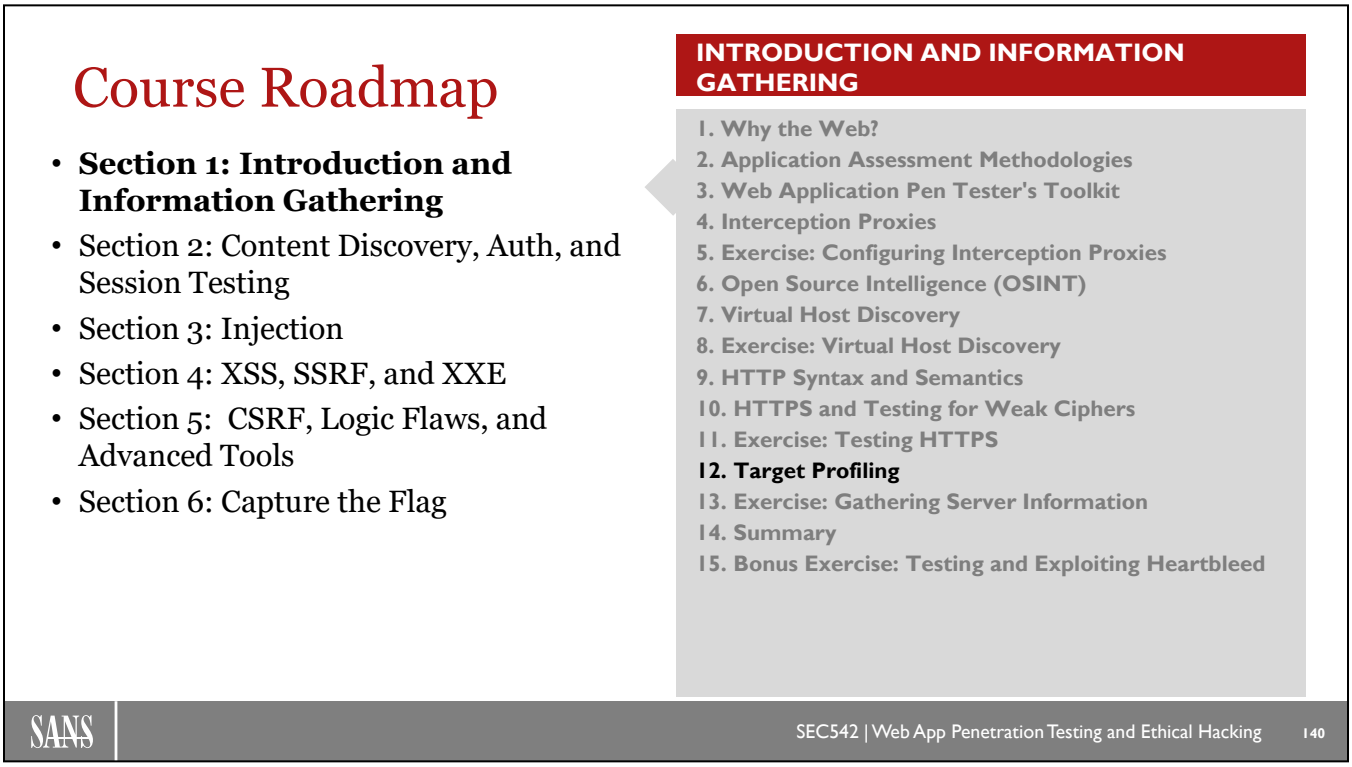
Welcome to Security 542: Web App Penetration Testing and Ethical Hacking!

We'll begin with an introduction to the web, talk about information gathering and virtual host discovery, review HTTPS, and then wrap up with target profiling.

Exercise 1.3: Testing HTTPS

SEC542 Workbook: Testing HTTPS

Please go to Exercise 1.3 in the 542 Workbook.



Course Roadmap

Welcome to Security 542: Web App Penetration Testing and Ethical Hacking!

We'll begin with an introduction to the web, talk about information gathering and virtual host discovery, review HTTPS, and then wrap up with target profiling.

Target Profiling

- Understanding the technology supporting the web application helps focus attacks later in the assessment
- The server and application technology may have vulnerabilities

WSTG-CONF-01	Test Network Infrastructure Configuration
WSTG-INFO-02	Fingerprint the Web Server
WSTG-INFO-04	Enumerate Applications on Webserver
WSTG-CONF-06	Test HTTP Methods
WSTG-INFO-03	Review Webserver Metafiles for Information Leakage

Target Profiling

The configuration of the underlying operating system and applications affects the overall security of the web application. The evaluation of these elements is sometimes overlooked or considered unimportant. Since an attacker would take advantage of a vulnerable service on the target system as an avenue to compromise the environment, it is shortsighted to ignore the overall system configuration during the web app test.

Testing the network infrastructure involves port scanning the in-scope systems, as well as evaluating the listening services and any available configuration details, such as the software version. The next several slides will examine port scanning and some common tools more closely.

The WSTG gives us insight into the value of web server fingerprinting:

"Accurately discovering the type of web server that an application runs on can enable security testers to determine if the application is vulnerable to attack. In particular, servers running older versions of software without up-to-date security patches can be susceptible to known version-specific exploits." [1]

Reviewing the supported HTTP methods and information leakage through metafiles, such as the robots.txt file, rounds out the picture of the targets' configuration while possibly revealing potential attack vectors and the location of "interesting" paths within the web application.

Reference:

[1] Fingerprint the Web Server: https://sec542.com/7c

Port Scanning

- Reviewing the services available on a webserver besides 80/TCP and 443/TCP is a key component of a thorough assessment
- Be sure to collect software details, such as the product and version, for each service:
  - Google the specific versions of software for vulnerabilities
  - Search ExploitDB and Metasploit for exploit code
  - Download open source software and search for vulnerabilities
- Several tools and cloud-based resources that provide network service information are introduced in this section

Port Scanning

A port scan attempts to connect to TCP and/or UDP ports and analyzes the response, or lack of a response, to determine whether there is a listening service available. Simple tools like netcat (nc) can be used to perform a port scan.

```
for port in {0..1023}
    do
        nc -vz 127.0.0.1 $port 2>&1 | grep succeeded
done
```

In the example above, a basic port scan of all TCP ports from 0 to 1023 is performed:

- A bash for loop iterates through the numbers 0 to 1023 and stores each value in the $port variable.
- Netcat's -v (verbose) switch is used to report when a connection fails or succeeds.
- Netcat's -z switch causes a connection to be tried without sending any data.
- The errors from netcat are redirected into standard IN (2>&1).
- Grep filters for responses associated with successful connections.

In the next lab, you will get the opportunity to connect to a web server using netcat and collect the web server's Server response header, which, if unmodified, often reveals the web server and its associated version.

Review Port Scan Results

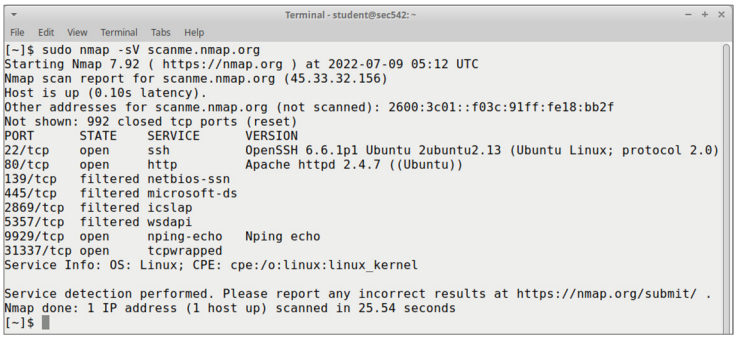
Reviewing these results is key to understanding the system configuration and knowing what attacks may be possible, or how a service may assist in the exploitation of a vulnerability in the web application.

While netcat can function as a port scanner, tools purpose-built for port scanning have additional features that provide greater detail about the targets. In this section, several tools and cloud-based services will be introduced.

- Nmap
- Zenmap
- Shodan
- Netcraft

Nmap

- Nmap is the most popular port scanner, with many features beyond scanning
- For the web application testing, the extras we are most interested in are:
  - Service version detection
  - NSE script engine



Nmap[1] Port Scanner

As its base function, Nmap will scan one or more targets to determine which TCP and/or UDP ports are available. Modern versions of Nmap include many more features, such as advanced operating system detection, service version detection, capabilities to tune performance, report formatting, and a scripting engine.

Service Version detection

Nmap's Service Version detection feature, activated with the -sV switch, will attempt to identify the application behind a listening service, as well as the software's version. Some services, like SSH, deliver a banner that includes the application and version details by default. Other services, such as HTTP, require a request after the initial connection is made. Nmap includes a probes database that has requests used to try to elicit responses from listening services. The database also includes response strings that are compared to the communication from the target to identify version details. On the VM provided for this course, the probes database is found in /usr/local/share/nmap/nmap-service-probes.

NSE Script Engine

Nmap includes a scripting engine that extends the functionality such that vulnerability discovery and exploitation are possible while port scanning targets. Hundreds of scripts are available, with many that focus on HTTP and SSL. Take a look at the scripts available on the course VM in /usr/local/share/nmap/scripts.

One thing to note about NSE scripts is that many scripts are written to only run under certain criteria, such as when a specific port number is discovered or a technology like HTTP. If a webserver is running on a non-standard port, such as 1234/tcp, some NSE scripts will not run against that service *unless* a Service Version scan (using the -sV or -A switches) is included in the Nmap configuration to identify that the service is running a protocol supported by the script.

Nmap Example

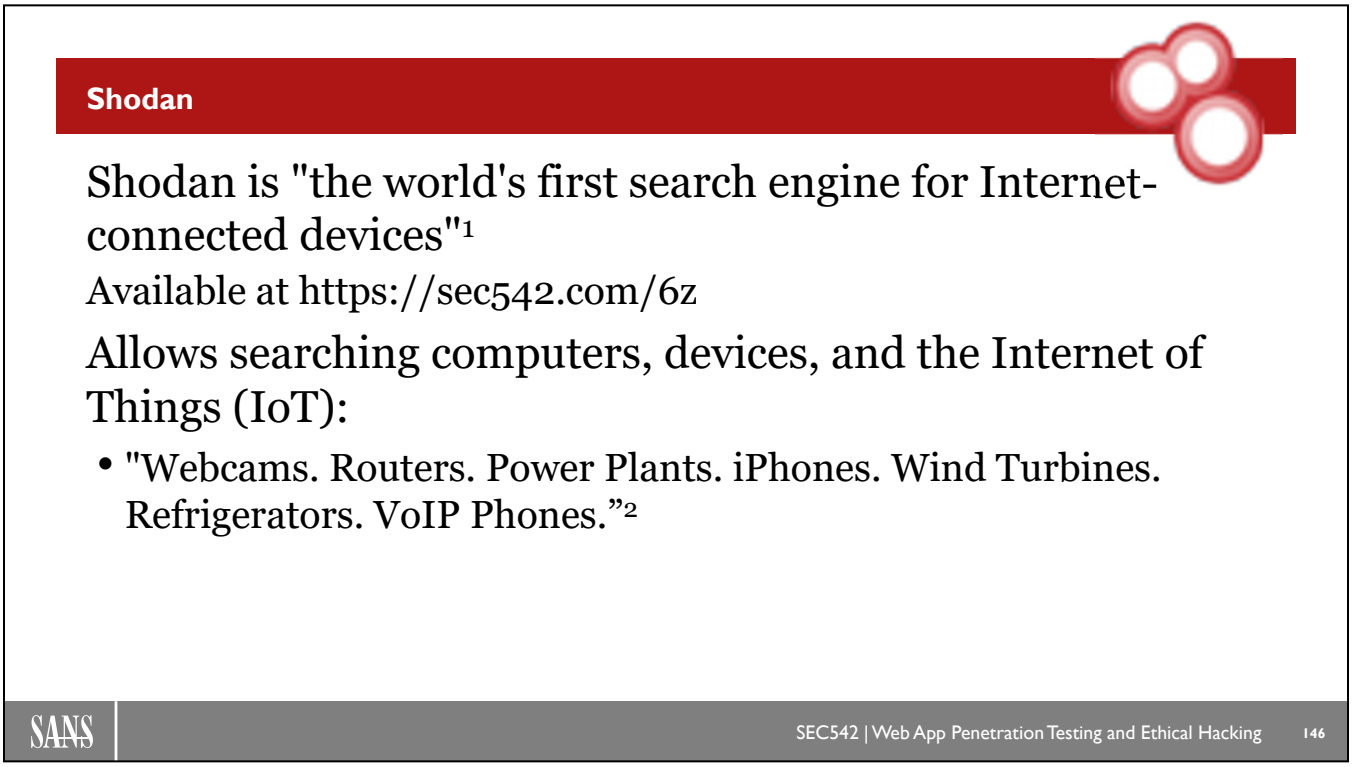
This slide demonstrates the use of Nmap to identify the application service version information of scanme.nmap.org, which is a site that allows public scanning:

> *We set up this machine to help folks learn about Nmap and also to test and make sure that their Nmap installation (or Internet connection) is working properly. You are authorized to scan this machine with Nmap or other port scanners. Try not to hammer on the server too hard. A few scans in a day is fine, but don't scan 100 times a day or use this site to test your ssh brute-force password cracking tool.*[2]

References:

[1] https://sec542.com/1h

[2] https://sec542.com/l



Shodan

Shodan is available at https://sec542.com/6z.

"Shodan is the world's first computer search engine that lets you search the Internet for computers. Find devices based on city, country, latitude/longitude, hostname, operating system, and IP."[3]

As we will see, the types of devices that Shodan finds are often critical, yet suffer from serious vulnerabilities such as default vendor credentials.

References:

[1] Shodan, https://sec542.com/6z

[2] Ibid.

[3] Ibid.

Searching Shodan for Penetration Testers

- Shodan might reveal systems that *should* be in scope, but are not (initially)
  - Might include systems managed by third parties on behalf of the client
- Search Shodan for "Client Name," and pay careful attention to devices outside the client's IP space
- Also, discuss these types of systems with your client: They will be difficult to find if they are outside the client's IP space, and may also lack strings associating them with the client's name

Searching Shodan for Penetration Testers

Bob Radvanovsky of Infracritical created Project SHINE (SHodan INtelligence Extraction). His website can be found at https://sec542.com/3a. He has found the following types of devices with Shodan:

medical devices

traffic management systems

automotive control

traffic light control (includes red-light and speeding cameras)

HVAC/environment control

power regulators/UPSs

security/access control (includes CCTV and webcams)

serial port servers (many of which include Allen-Bradley DF1 capable protocols)

data radios (point-to-point 2.4/5.8/7.8 GHz direct-connected radios)

Some of the more interesting control applications we have uncovered are off-road mining trucks and crematoriums. Why these are connected to the Internet is a mystery to us…

In many cases, a default web interface, usually with administrative privileges, is active on the SCADA or ICS device. Rarely is it ever disabled by the end user. We believe that there are two possible reasons for this.

(1) The systems integrator is a contractor who doesn't have a full knowledge of the product being installed.

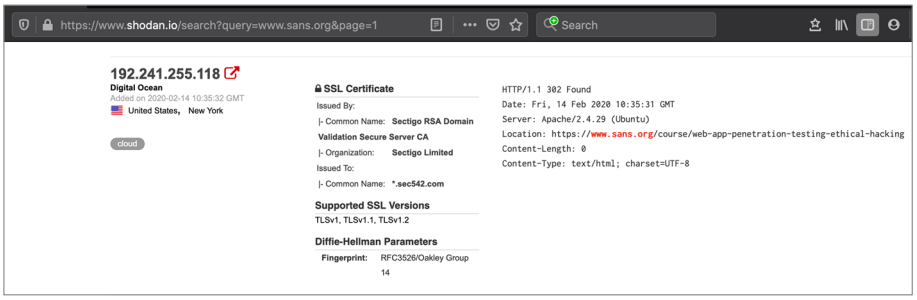
(2) Owner of this system is unaware that there is a web interface that is enabled by default from the manufacturer.1

Reference:

[1] Project SHINE – Are Control Systems REALLY Connected to the Internet? https://sec542.com/73

Shodan as a Port Scanner

- Not just an OSINT tool, Shodan also facilitates as a port scanner
  - Includes details about SSL certificates
  - May aid in the discovery of recent configuration changes to harden the target environment prior to a pen test
  - Can help find neighboring systems



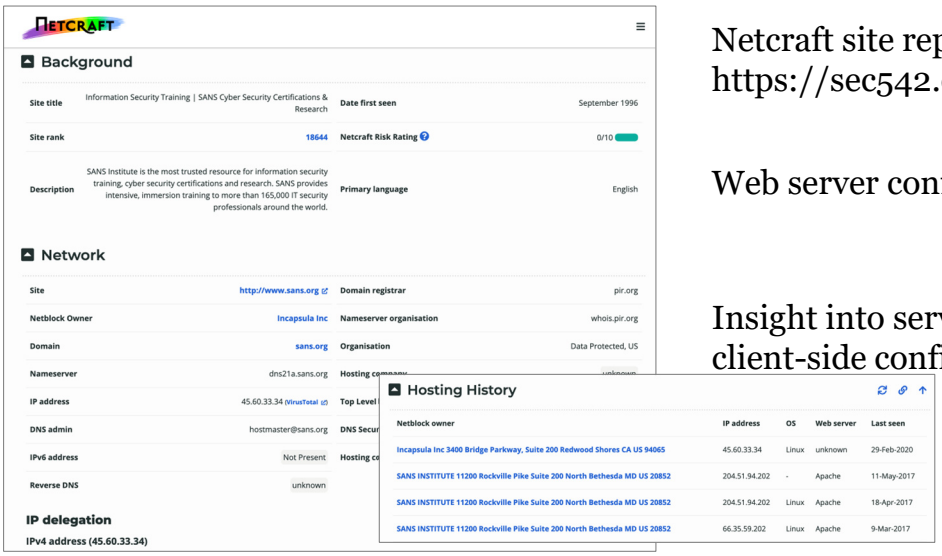
Shodan as a Port Scanner

Introduced in Section 1 in the OSINT portion, the Shodan project regularly scans the entire internet and makes the results available on the Shodan.io website. One nice aspect of having internet port scan data in a database is that the results are immediately available without having to wait for tools like Nmap to finish running. Additionally, Shodan provides a covert way to collect details about in-scope systems without having to send any packets.

Data in Shodan's database can help identify changes to the target environment that affect the security of the application. For example, Shodan results may include an administrative console on a non-standard port that was previously available, yet an Nmap scan reveals the port is currently closed.

Shodan's data can also be used to review the configuration of nearby IP addresses to determine whether additional systems should be in-scope for the current assessment, or perhaps scheduled for subsequent pen tests.

Netcraft Sitereport



Netcraft site report:
https://sec542.com/a0

Web server configuration…
over time

Insight into server-side and client-side configuration

Netcraft Sitereport

The sitereport function on Netcraft.com does not provide details about non-web specific services; however, the cloud-based tool is noteworthy because of the level of detail that it provides about the web server and application configuration. Not only does the report include the web server and its reported version, but that information is listed over the lifespan of the website. In addition to the web server configuration, the server-side and client-side technologies used by the website are also provided.

Netcraft's site report is a stealthy option for target profiling since the attacker does not have to send any packets to the in-scope system to review the configuration.

Application Infrastructure

- Review the target profiling data to understand the security controls and insight into the overall management practices for the web application
- Document the components that make up the website
- Use the details to identify common configuration flaws, default credentials, publicly available exploit code, and other potential attack paths

Application Infrastructure

Be sure to analyze the target profiling data, as it will give a perspective of the management practices for the system. Older outdated software indicates the organization may not prioritize patching. Default content in the webroot suggests that hardening best practices may not have been followed.

Note that the system configuration details may not directly lead to exploitation and a subsequent foothold on the environment, but rather allow for a more in-depth understanding of the impact of discovered vulnerabilities. Consider the following scenario where port scan results include, among other services, 3306/tcp which is associated with a MySQL service. By itself, the MySQL database being exposed to the internet is likely to be a low-severity finding.

Suppose that an unauthenticated Local File Include (LFI) vulnerability is also found in the web app. LFI allows an attacker to read the contents of files on the web server, and the unauthenticated element means that the files can be accessed without first logging in. We will explore LFI and how to exploit this vulnerability later in the class. While exploiting the LFI vulnerability, the connection string to the database server is discovered inside a configuration file that includes the credentials for authenticating to the MySQL server. Now a direct database connection using MySQL client software with the newfound credentials is possible. As a result, the impact of the MySQL service's exposure, as well as the LFI vulnerability, is better understood because the port scan was performed.

Use port scan information that includes software titles and version specifics to look for publicly available exploit code. If time permits, consider downloading any open source applications that are discovered and review the source code for vulnerabilities. Having a local instance of discovered software also provides a platform to test public exploit code or to tune and troubleshoot the exploitation of vulnerabilities in the web application.

The next slide explores various ways to identify components of the web server's infrastructure.

Identifying Web Server Components

Webserver Component	Examples	Common Discovery Path(s)
Web Servers	Apache IIS NGINX	Port scans, default web pages, fingerprinting tools
Application Frameworks	Spring ASP.NET Django Symfony	Default web pages, vulnerability scans, configuration files, administrative pages, fingerprinting tools
Content Management Systems (CMS)	WordPress Drupal Joomla SharePoint	Default web pages, vulnerability scans, configuration files, administrative pages, fingerprinting tools
Databases	MySQL Microsoft SQL Oracle Postgres MongoDB	Port scans, detailed application errors, configuration files
Other Software	OpenSSH Remote Desktop Protocol (RDP) File Transfer Protocol (FTP) Server Message Block (SMB)	Port scans

Identifying Web Server Components

The data collected from target profiling activities should be used to understand the web server and associated target application. Some common paths to discover the web server's components are briefly explored below. Application Frameworks and Content Management Systems will be discussed again in the Spidering Web Applications and Forced Browsing sections later in this section.

Port Scans: Already explored in-depth earlier in this section, a port scan is an integral part of determining key elements of a web application.

Default Web Pages: Finding default pages in a web application can be used to fingerprint the web server, application frameworks, and/or content management systems. Usually automated vulnerability scanners identify these pages for the attacker. Forced browsing, which is explored later in this section, is another way to identify default content. When found, default pages may immediately display software version information in the browser or in the source code for the page; whereas other times, it may be necessary to compare the contents of the page with several instances of the same page, within various editions of the software, to identify the installed software's version.

Vulnerability Scans: Many vulnerability scanners will call out the components that make up the web application and even fingerprint the software versions when possible.

Configuration Files: Discovered through manual inspection and forced browsing, if configuration files specific to an application framework or content management system are found, it is likely those software components exist on the target. Some configuration files include the database connection string, or other related configuration items, that may disclose details about the backend database management system.

Administrative Pages: Even when other pages of the application do not include software titles and version information, frequently the login page of the administrative console includes those details. Sometimes it is as easy as appending /admin to the webroot to find an administrative console. Other times, looking in the robots.txt file, forced browsing, or research on the internet is required to find an admin console. When found, the version information that is sometimes provided can help identify vulnerabilities.

WSTG-CONF-06: Test HTTP Methods

Knowing the available HTTP methods:

- GET: parameters in the URI may disclose sensitive information
- Identifies potential vulnerabilities
- Gives clues about how well the system is maintained
- May reveal additional functionality to test

A few examples of how attackers may use specific HTTP methods:

- GET: parameters in the URI may disclose sensitive information
- POST: parameters can be used to exploit vulnerabilities
- PUT: may be able to upload files to the webroot
- TRACE: identify load balancers or other devices in front of the webserver
- DELETE: remove files from the webroot

WSTG-CONF-06: Test HTTP Methods

”While GET and POST are by far the most common methods that are used to access information provided by a web server, HTTP allows several other (and somewhat less known) methods. Some of these can be used for nefarious purposes if the web server is misconfigured…Since the other methods are so rarely used, many developers do not know, or fail to take into consideration, how the web server or application framework's implementation of these methods impact the security features of the application." 1

Of course, it is possible to manually enumerate the available methods, as shown by the following bash script:

```
for methods in GET HEAD POST PUT DELETE TRACE OPTIONS
  do
    echo "$methods:"
    echo -ne "$methods / HTTP/1.0\n\n" | nc -C 127.0.0.1 80 | grep "HTTP/"
done
```

The preceding one-liner performs the following actions:

Uses a bash for loop to iterate through each of the common HTTP methods:
for methods in GET HEAD POST PUT DELETE TRACE OPTIONS

Outputs the method being tested for quick reference in the output:
echo "$methods:"

Connects to the webserver and tries to use the associated method:
echo -ne "$methods / HTTP/1.0\n\n" | nc -C 127.0.0.1 80

Filters the results to only include the server's HTTP response code:

| grep "HTTP/"

Closes the for loop:

done

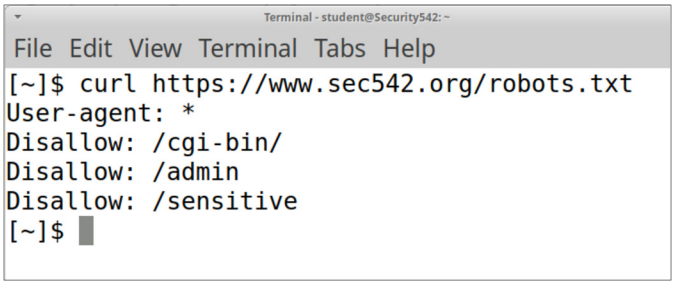
Most of the automated tools are going to identify the methods available on the system, which means there is not much value in enumerating these manually unless an automated scan is not going to be run. For the purposes of this course, the manual method is included because it is good for an attacker to understand how things work behind the scenes.

Reference:

[1] Test HTTP Methods (WSTG-CONF-06): https://sec542.com/7d

WSTG-INFO-03: Review Webserver Metafiles for Information Leakage

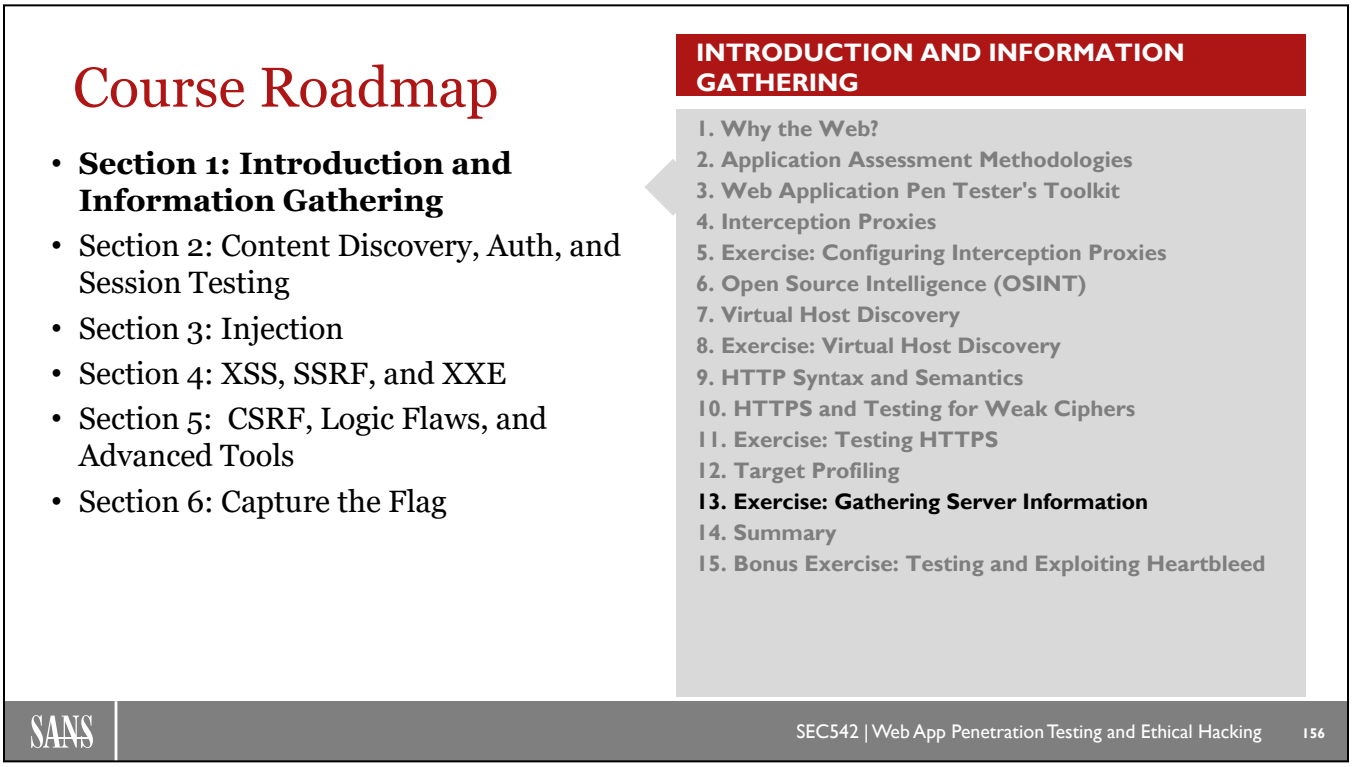
- The robots.txt file lists paths that "well-behaved" spiders will ignore when crawling the website
- Occasionally, robots.txt contains paths to administrative consoles or other locations containing sensitive data
- Curiosity is a vital trait of experts
  - Pen testers: *look at what is in the paths listed in a robots.txt file*



WSTG-INFO-03: Review Webserver Metafiles for Information Leakage

A well-behaved spider, like Wget, will ignore paths that are included in the robots.txt file. Attackers are interested in these paths because, many times, "security by obscurity" is employed through the inclusion of paths that contain sensitive functions within the application, like administrative consoles.

As will be observed later in this section when talking about website spidering, many pen test-focused tools will include the contents of the robots.txt file in the search for content on a website. Likewise, a curious attacker will look in these paths to see what content is being excluded from search engine indexing.



Course Roadmap

Welcome to Security 542: Web App Penetration Testing and Ethical Hacking!

We'll begin with an introduction to the web, talk about information gathering and virtual host discovery, review HTTPS, and then wrap up with target profiling.

Exercise 1.4: Gathering Server Information

SEC542 Workbook: Gathering Server Information

Please go to Exercise 1.4 in the 542 Workbook.

Course Roadmap

- **Section 1: Introduction and Information Gathering**
- Section 2: Content Discovery, Auth, and Session Testing
- Section 3: Injection
- Section 4: XSS, SSRF, and XXE
- Section 5: CSRF, Logic Flaws, and Advanced Tools
- Section 6: Capture the Flag

INTRODUCTION AND INFORMATION GATHERING

1. Why the Web?
2. Application Assessment Methodologies
3. Web Application Pen Tester's Toolkit
4. Interception Proxies
5. Exercise: Configuring Interception Proxies
6. Open Source Intelligence (OSINT)
7. Virtual Host Discovery
8. Exercise: Virtual Host Discovery
9. HTTP Syntax and Semantics
10. HTTPS and Testing for Weak Ciphers
11. Exercise: Testing HTTPS
12. Target Profiling
13. Exercise: Gathering Server Information
14. **Summary**
15. Bonus Exercise: Testing and Exploiting Heartbleed

Course Roadmap

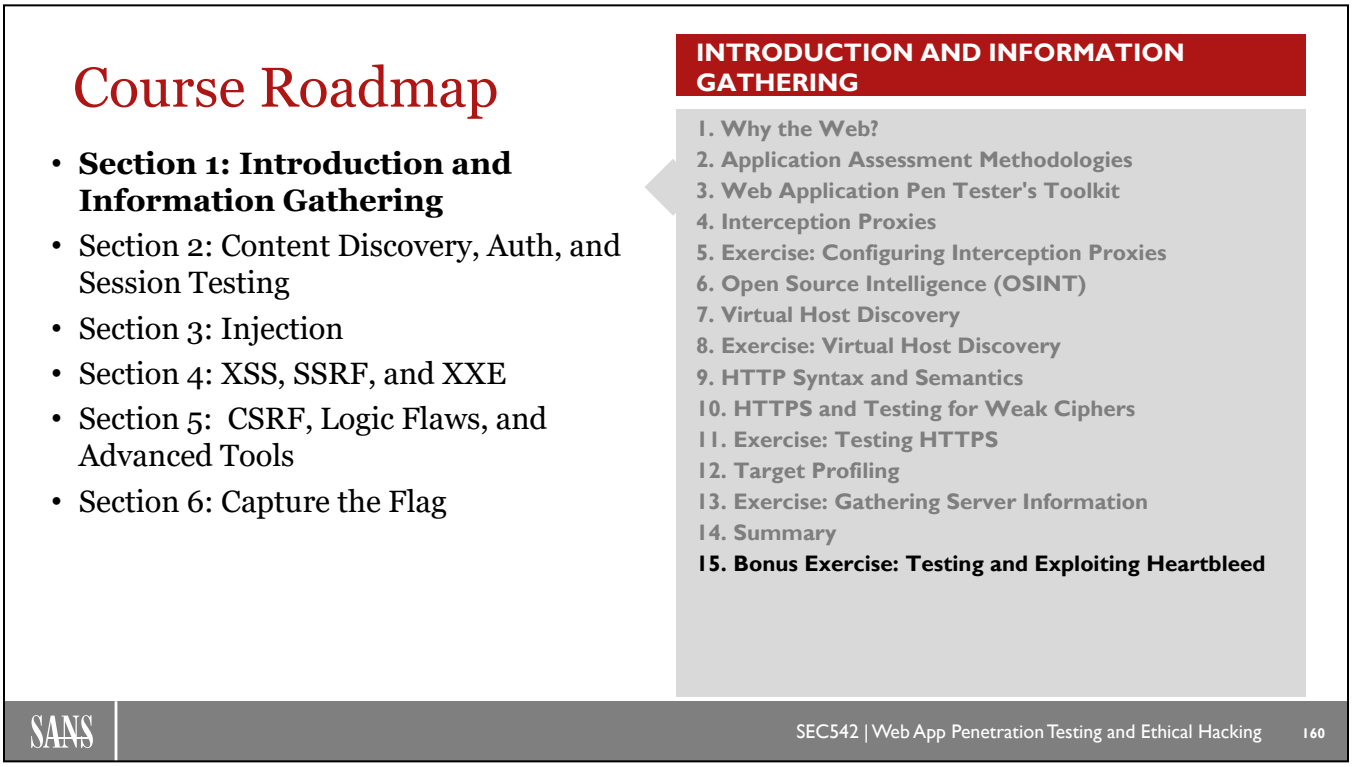
Welcome to Security 542: Web App Penetration Testing and Ethical Hacking!

We'll begin with an introduction to the web, talk about information gathering and virtual host discovery, review HTTPS, and then wrap up with target profiling.

Summary

- We have discussed the web, interception proxies, information gathering, HTTP/HTTPS, and target profiling
- Next up: 542.2, where we will discuss content discovery, identity, authentication/authorization, and session testing
- Thank you!

This page intentionally left blank.



Course Roadmap

Welcome to Security 542: Web App Penetration Testing and Ethical Hacking!

We'll begin with an introduction to the web, talk about information gathering and virtual host discovery, review HTTPS, and then wrap up with target profiling.

Heartbleed

The Heartbleed OpenSSL vulnerability was publicly discovered in April 2014

- Named after the TLS heartbeat extension

Affected OpenSSL versions 1.0.1 -> 1.0.1f, and 1.0.2-beta1

- Vulnerability was unpatched from March 2012 through April 2014
- CVE 2014-0160

Lots of details available at https://sec542.com/a

Heartbleed

CVE 2014-0160 was assigned the Heartbleed vulnerability. [1]

The Heartbleed website has tremendous detail and resources related to this flaw.[2]

RFC 6520, “Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension,” describes the heartbeat extension: "The Heartbeat Extension provides a new protocol for TLS/DTLS allowing the usage of keep-alive functionality without performing a renegotiation and a basis for path MTU (PMTU) discovery for DTLS."[3]

OpenSSL versions number lower than 1.0.1 or higher than 1.0.1g are not vulnerable.

References:

[1] CVE-2014-0160, https://sec542.com/o

[2] Heartbleed Bug, https://sec542.com/a

[3] RFC 6520, https://sec542.com/1r

Effect of the Vulnerability

Heartbleed allows remote reading of 64KB memory chunks directly from a vulnerable OpenSSL server

- Repeated attempts may divulge different chunks of RAM
- Nothing is logged on the web server

What's in RAM?

- Usernames
- Passwords
- Cookies
- And more!

How about the OpenSSL server's private key?

Effect of the Vulnerability

The bug was "independently discovered by a team of security engineers (Riku, Antti, and Matti) at Codenomicon and Neel Mehta of Google Security, who first reported it to the OpenSSL team."[1]

Heartbleed allows a remote user to download 64KB chunks of RAM. The attacker may make repeated requests to attempt to access different chunks:

> *Can attacker access only 64k of the memory?*
>
> *There is no total of 64 kilobytes limitation to the attack, that limit applies only to a single heartbeat. Attacker can either keep reconnecting or during an active TLS connection keep requesting arbitrary number of 64 kilobyte chunks of memory content until enough secrets are revealed.*[2]

References:

[1] Heartbleed Bug, https://sec542.com/a

[2] Ibid.

The CloudFlare Challenge

- Stealing a vulnerable server's private key was originally thought to be unlikely or impossible
- CloudFlare issued a challenge: Steal the private key from a vulnerable server
  - "We set up a nginx server with a vulnerable version of OpenSSL and challenged the community to steal its private key."[1]
- Four people successfully stole the server's private key on the first day of the challenge

The CloudFlare Challenge

Many believed the server's private key could not be stolen via Heartbleed. This (now retracted) post is one example. It is titled "Why Heartbleed Doesn't Leak the Private Key":

> "The headline for #heartbleed is that it might leak the private key. In most software, this cannot happen. That's because memory containing the private key is never freed, and hence allocated heartbleed buffers can never contain it."1

Errata Security gracefully retracted that post when proven wrong and changed the post's title to "I Got This Completely Wrong!"

CloudFlare set up a challenge server running nginx-1.5.13 compiled with OpenSSL 1.0.1.f, running on Ubuntu 13.10 x86_64, at https://www.cloudflarechallenge.com/heartbleed. Note: The server no longer appears to be up, resulting in a redirect loop when this slide was written.

According to CloudFlare, "The first valid submission was received at 16:22:01PST by Software Engineer Fedor Indutny. He sent at least 2.5 million requests over the course of the day. The second was submitted at 17:12:19PST by Ilkka Mattila at NCSC-FI, who sent around a hundred thousand requests over the same period of time."3

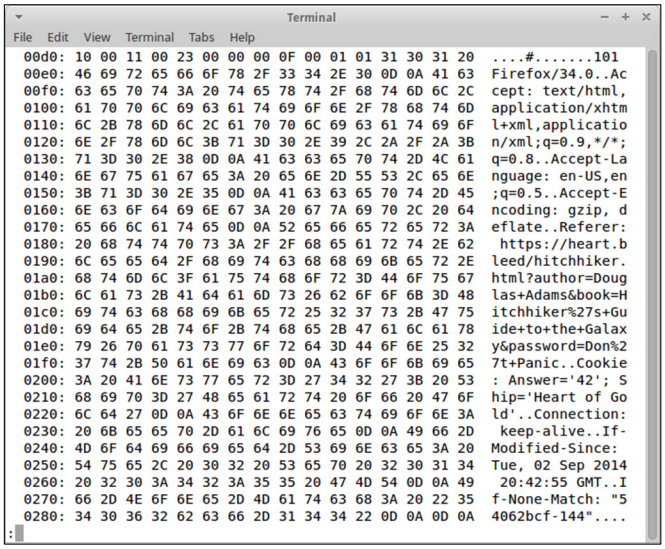
References:

[1] The Results of the CloudFlare Challenge, https://sec542.com/3m

[2] Why Heartbleed doesn't leak the private key, Errata Security, https://sec542.com/3o

[3] The Results of the CloudFlare Challenge, https://sec542.com/3m

Heartbleed Exploit Output



Heartbleed Exploit Output

The screenshot shown on the slide uses Sensepost's Heartbleed PoC code, available at https://sec542.com/19. It is also installed in /usr/local/bin/heartbleed.py.

The form variables for "author," "book," and "password" have been dumped from RAM, as well as a cookie.

```
$ /usr/local/bin/heartbleed.py
Usage: heartbleed.py server [options]

Test for SSL heartbeat vulnerability (CVE-2014-0160)

Options:
  -h, --help            Show this help message and exit
  -p PORT, --port=PORT  TCP port to test (default: 443)
  -n NUM, --num=NUM     Number of heartbeats to send if vulnerable (defines
                        how much memory you get back) (default: 1)
  -f FILE, --file=FILE  Filename to write dumped memory (default: dump.bin)
  -q, --quiet           Do not display the memory dump
  -s, --starttls        Check STARTTLS (smtp only right now)
```

Bonus Exercise: Testing and Exploiting Heartbleed

SEC542 Workbook: Testing and Exploiting Heartbleed

Please go to Exercise 1.Bonus in the 542 Workbook.