




Welcome to

5. Web Application Security: Offensive

Security in Web Development Elective, KEA

Henrik Kramselund he/him han/ham hkj@zencurity.com @kramse  

Slides are available as PDF, kramse@Github

5-web-app-hacking-offensive-security-in-web.tex in the repo security-courses

Goals for today



Today's goals:

- Web Application Hacking – offensive
- Talk about how to perform offensive testing on web applications
- Talk about common patterns and findings in web applications
- Discuss how we can do this more structured, hint OWASP Testing Guide

Photo by Thomas Galler on Unsplash

Plan for today



Subjects

- Offense against web application, as an introduction to what our apps get
- The list of offensive methods from the Web Application Security book

Exercises

- Discussion about JuiceShop Attacks, but consider how you would avoid them

Time schedule



- Structured approach to offensive, we will download the OWASP Testing Guide
- 1) Going over my presentation, first part summary of recon – first 45min
- 2) The book chapters 9-16, a structured method – next 2x 45min
- 4) Introduction to the exam project!

Reading Summary



Web Application Security, Andrew Hoffman, 2020, ISBN: 9781492053118

Part II. Offense, chapters 9-16, very short chapters

9. Introduction to Hacking Web Applications

10. Cross-Site Scripting (XSS)

11. Cross-Site Request Forgery (CSRF)

12. XML External Entity (XXE)

13. Injection

14. Denial of Service (DoS)

15. Exploiting Third-Party Dependencies

16. Part II Summary

What we learnt from the recon part



In Part I of this book, “Recon,” we explored a number of ways to investigate and document the structure and function of a web application. We evaluated ways of finding APIs on a server, including those that exist on subdomains rather than at just the top-level domain. We considered methods of enumerating the endpoints that those APIs exposed, and the HTTP verbs that they accepted.

Source: *Web Application Security*, Andrew Hoffman, 2020, ISBN: 9781492053118

- We are allowed to do reconnaissance
- Having a good inventory of software, hardware, devices, etc. makes it easier to control and secure
- Having knowledge about patching levels is vital!

Sqlmap



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

Automatic SQL injection and database takeover tool <http://sqlmap.org/>

sqlmap features



; Features();--

- Full support for **MySQL, Oracle, PostgreSQL, Microsoft SQL Server, Microsoft Access, IBM DB2, SQLite, Firebird, Sybase, SAP MaxDB and HSQLDB** database management systems.
- Full support for six SQL injection techniques: **boolean-based blind, time-based blind, error-based, UNION query-based, stacked queries and out-of-band**.
- Support to **directly connect to the database** without passing via a SQL injection, by providing DBMS credentials, IP address, port and database name.
- Support to enumerate **users, password hashes, privileges, roles, databases, tables and columns**.
- Automatic recognition of password hash formats and support for **cracking them using a dictionary-based attack**.
- Support to **dump database tables** entirely, a range of entries or specific columns as per user's choice. The user can also choose to dump only a range of characters from each column's entry.

Not a complete list!

Source: <http://sqlmap.org/>

PHP shell escapes



```
<pre>
<?php passthru(" netstat -an && ifconfig -a"); ?>
</pre>
```

Many tools have shell escapes:

- Perl: `print `/usr/bin/finger $input{'command'}`;`
- UNIX shell: ``echo hej``
- Microsoft SQL: `exec master..xp_cmdshell 'net user test testpass /ADD'`

So being able to inject SQL or commands can be critical

Shellshock CVE-2014-6271 - and others



```
5. vagrant@ubuntu: ~ (ssh)
hlk@katana:speedtest$ ssh vagrant@192.168.0.179
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-30-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

System information as of Wed Nov  5 07:55:03 CET 2014

System load:  0.46           Processes:            228
Usage of /:   4.5% of 58.20GB Users logged in:        0
Memory usage: 15%           IP address for eth0: 192.168.0.179
Swap usage:   0%

Graph this data and manage this system at:
https://landscape.canonical.com/

Last login: Mon Jul  7 17:08:26 2014
vagrant@ubuntu:~$ dpkg -s bash | grep Version
Version: 4.3-7ubuntu1
vagrant@ubuntu:~$ env x='()' { :;; echo vulnerable' bash -c "echo this is a test"
vulnerable
this is a test
vagrant@ubuntu:~$
```

Source: [https://en.wikipedia.org/wiki/Shellshock_\(software_bug\)](https://en.wikipedia.org/wiki/Shellshock_(software_bug))

Initial Overview of Software Security



- Security Testing Versus Traditional Software Testing
- Functional testing does not prevent security issues!
- SQL Injection example, injecting commands into database
- Attackers try to break the application, server, operating system, etc.
- Use methods like user input, memory corruption / buffer overflow, poor exception handling, broken authentication, ...

Where to start?



The OWASP Top Ten provides a minimum standard for web application security. The OWASP Top Ten represents a broad consensus about what the most critical web application security flaws are.

The Open Web Application Security Project (OWASP)

OWASP produces lists of the most common types of errors in web applications

<http://www.owasp.org>

Create Secure Software Development Lifecycle

OWASP Web Security Testing Guide



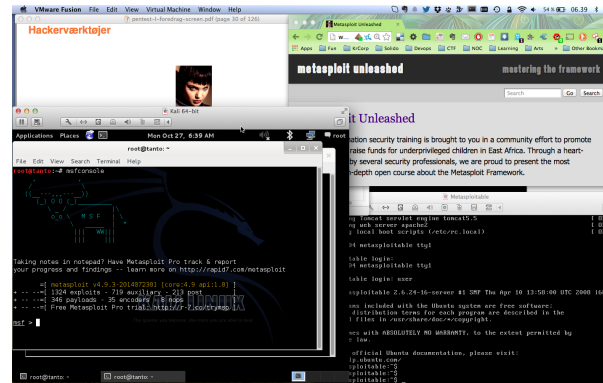
The Web Security Testing Guide (WSTG) Project produces the premier cybersecurity testing resource for web application developers and security professionals.

The WSTG is a comprehensive guide to testing the security of web applications and web services. Created by the collaborative efforts of cybersecurity professionals and dedicated volunteers, the WSTG provides a framework of best practices used by penetration testers and organizations all over the world.

Source: OWASP

- OWASP Web Security Testing Guide
<https://owasp.org/www-project-web-security-testing-guide/>
- Also available as a checklist `OWASPv4_Checklist.xlsx`

Testing Labs

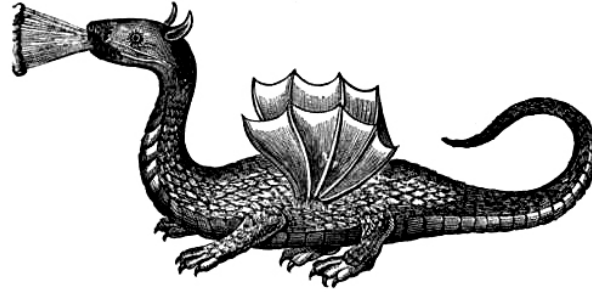


- Sniffers Wireshark and similar tools
- Proxies and fuzzers
- Debuggers
- Virtualisation - can also emulate ARM on Intel etc.
- Laptops and network hardware - dont use a HUB! Cheap managed switch with mirror port is better

We will get to the Defensive part of the book soon



Goals: Secure Software



Here be dragons

- Software is insecure
- How do we improve quality
- Higher quality is more stable, and more secure
- Make sure to test specifically for security issues

We talked about security design with Qmail and Postfix recently. This year has been bad for Exim mailserver: CVE-2019-10149, CVE-2019-13917 and CVE-2019-15846

Software Development Lifecycle



A full lifecycle approach is the only way to achieve secure software.

–Chris Wysopal

- Often security testing is an afterthought
- Vulnerabilities emerge during design and implementation
- Before, during and after approach is needed

Secure Software Development Lifecycle



- SSDL represents a structured approach toward implementing and performing secure software development
- Security issues evaluated and addressed early
- During business analysis
- through requirements phase
- during design and implementation

Functional specification needs to evaluate security



- Completeness
- Consistency
- Feasibility
- Testability
- Priority
- Regulations

Source: The Art of Software Security Testing Identifying Software Security Flaws Chris Wysopal ISBN: 9780321304865

Phases of SSDL



- Phase 1: Security Guidelines, Rules, and Regulations
- Phase 2: Security requirements: attack use cases
- Phase 3: Architectural and design reviews/threat modelling
- Phase 4: Secure coding guidelines
- Phase 5: Black/gray/white box testing
- Phase 6: Determining exploitability

End part I



Exploiting web applications is exploiting software



In the next few chapters, you will learn how to take advantage of web applications through a number of powerful and common exploitation techniques. As you learn about these techniques, consider the lessons from the previous part and attempt to brainstorm how those recon techniques would be useful in helping you find weaknesses in an application where the upcoming exploits you'll learn about be applied.

Source: *Web Application Security*, Andrew Hoffman, 2020, ISBN: 9781492053118

- Attacking web applications attack the whole organisation
- Web applications are exposing the organisation
- Vulnerabilities in the web applications typically allow access to data/databases



9. Introduction to Hacking Web Applications

Software engineers measure productivity in value-add through features, or improvements to an existing codebase. A software engineer might say, “I added features x and y, hence today was a good day.” Alternatively, they might say, “I improved the performance of features a and b by 10%,” alluding to the fact that the work of a software engineer, while difficult to measure compared to traditional occupations, is still quantifiably measurable.

Hackers measure productivity in ways that are much more difficult to discern and measure. This is because the majority of hacking is actually data gathering and analysis. Often this process is riddled with false positives and might look like time wasted to an uneducated onlooker.

Most hackers don’t deconstruct or modify software but instead analyze software in order to work with the existing codebase—seeking entrypoints rather than making them. Often the skills used to analyze an application while seeking entrypoints are similar, if not identical, to the skills presented in the first part of this book.

Source: *Web Application Security*, Andrew Hoffman, 2020, ISBN: 9781492053118

10. Cross-Site Scripting (XSS)



Cross-Site Scripting (XSS) vulnerabilities are some of the most common vulnerabilities throughout the internet, and have appeared as a direct response to the increasing amount of user interaction in today's web applications. At its core, an XSS attack functions by taking advantage of the fact that web applications execute scripts on users' browsers. Any type of dynamically created script that is executed puts a web application at risk if the script being executed can be contaminated or modified in any way—in particular by an end user.

XSS attacks are categorized a number of ways, with the big three being:

- Stored (the code is stored on a database prior to execution)
- Reflected (the code is not stored in a database, but reflected by a server)
- DOM-based (code is both stored and executed in the browser)

Note: attacks the user, his/her data mostly.

11. Cross-Site Request Forgery (CSRF)



Sometimes we already know an API endpoint exists that would allow us to perform an operation we wish to perform, but we do not have access to that endpoint because it requires privileged access (e.g., an admin account).

In this chapter, we will be building Cross-Site Request Forgery (CSRF) exploits that result in an admin or privileged account performing an operation on our behalf rather than using a JavaScript code snippet.

CSRF attacks take advantage of the way browsers operate and the trust relationship between a website and the browser. By finding API calls that rely on this relationship to ensure security—but yield too much trust to the browser—we can craft links and forms that with a little bit of effort can cause a user to make requests on his or her own behalf—unknown to the user generating the request.

- Often seen in small CPE routers in homes, if the user activates an evil link, their router might be reconfigured or taken over

12. XML External Entity (XXE)



XML External Entity (XXE) is a classification of attack that is often very simple to execute, but with devastating results. This classification of attack relies on an improperly configured XML parser within an application's code. Generally speaking, almost all XXE attack vulnerabilities are found as a result of an API endpoint that accepts an XML (or XML-like) payload. You may think that HTTP endpoints accepting XML is uncommon, but XML-like formats include SVG, HTML/DOM, PDF (XFDF), and RTF. These XML-like formats share many common similarities with the XML spec, and as result, many XML parsers also accept them as inputs.

The magic behind an XXE attack is that the XML specification includes a special annotation for importing external files. This special directive, called an external entity, is interpreted on the machine on which the XML file is evaluated. This means that a specially crafted XML payload sent to a server's XML parser could result in compromising files in that server's file structure.

- Include file X, and the XML parser does as instructed!
- Often I consider these along the same lines as Java serialization attacks, sending data in JAVA that gets converted/expanded etc.

13. Injection



One of the most commonly known types of attacks against a web application is SQL injection. SQL injection is a type of injection attack that specifically targets SQL databases, allowing a malicious user to either provide their own parameters to an existing SQL query, or to escape an SQL query and provide their own query. Naturally, this typically results in a compromised database because of the escalated permissions the SQL interpreter is given by default.

SQL injection is the most common form of injection, **but not the only form**. Injection attacks have two major components: an interpreter and a payload from a user that is somehow read into the interpreter. This means that injection attacks can occur against command-line utilities like FFMPEG (a video compressor) as well as against databases (like the traditional SQL injection case).

- Injecting SQL commands, perform database actions
- Escape into the operating system via SQL, easier in the old days
- Escape into shell from URL parameters, often happens
- Compare to shellshock vuln, sending data that later ends up in the Unix shells

14. Denial of Service (DoS)



Perhaps one of the most popular types of attacks, and the most widely publicized, is the distributed denial of service (DDoS) attack. This attack is a form of denial of service (DoS), in which a large network of devices flood a server with requests, slowing down the server or rendering it unusable for legitimate users.

DoS attacks come in many forms, from the well-known distributed version that involves thousands or more coordinated devices, to code-level DoS that affects a single user as a result of a faulty regex implementation, resulting in long times to validate a string of text. DoS attacks also range in seriousness from reducing an active server, to a functionless electric bill, to causing a user's web page to load slightly slower than usual or pausing their video midbuffer.

Because of this, it is very difficult to test for DoS attacks (in particular, the less severe ones). Most bug bounty programs outright ban DoS submissions to prevent bounty hunters from interfering with regular application usage.

- There have been some hash-based attacks and attacks using data to create bad performance, see PHP security advisories

15. Exploiting Third-Party Dependencies



The rampant use of third-party dependencies, in particular from the OSS realm, has created an easy-to-overlook gap in the security of many web applications. A hacker, bug bounty hunter, or penetration tester can take advantage of these integrations and jumpstart their search for live vulnerabilities. Third-party dependencies can be attacked a number of ways, from shoddy integrations to fourth-party code or just by finding known exploits discovered by other researchers or companies.

- Be careful what you include in your environment

CIS Controls: Offense informs defense



- **Offense informs defense:** Use knowledge of actual attacks that have compromised systems to provide the foundation to continually learn from these events to build effective, practical defenses. Include only those controls that can be shown to stop known real-world attacks.
- **Prioritization:** Invest first in Controls that will provide the greatest risk reduction and protection against the most dangerous threat actors and that can be feasibly implemented in your computing environment. The CIS Implementation Groups discussed below are a great place for organizations to start identifying relevant Sub-Controls.
- **Measurements and Metrics:** Establish common metrics to provide a shared language for executives, IT specialists, auditors, and security officials to measure the effectiveness of security measures within an organization so that required adjustments can be identified and implemented quickly.
- **Continuous diagnostics and mitigation:** Carry out continuous measurement to test and validate the effectiveness of current security measures and to help drive the priority of next steps.
- **Automation:** Automate defenses so that organizations can achieve reliable, scalable, and continuous measurements of their adherence to the Controls and related metrics.

Source: CIS-Controls-Version-7-1.pdf



CIS Control 18:

Application Software Security

Manage the security life cycle of all in-house developed and acquired software in order to prevent, detect, and correct security weaknesses.

Source: Center for Internet Security CIS Controls 7.1 `CIS-Controls-Version-7-1.pdf`

CIS Control 16:

Application Software Security

Manage the security life cycle of in-house developed, hosted, or acquired software to prevent, detect, and remediate security weaknesses before they can impact the enterprise.

Source: Center for Internet Security CIS Controls v8 `CIS_Controls_v8_Guide.pdf`

Exercise time: Discussing the JuiceShop



We know the Juice Shop contains lots of vulnerabilities. How would you proceed if this was your company?

- Test and patch
- Insert Web Application Firewall in front
- Start from scratch

How do we protect against attacks we don't know or understand?

Exercise



Now lets do the exercise

JuiceShop Attacks 60min

which is number **23** in the exercise PDF.

For Next Time



Think about the subjects from this time, write down questions

Check the plan for chapters to read in the books

Visit web sites and download papers if needed

Retry the exercises to get more confident using the tools