You've Been Hacked

An (Interactive) Course on Web Security

Paul Duplys

@duplys

January 14, 2021


0x0: Preliminaries

whoami

short intro/bio.

man slides

(Interactive) course on web security based on Carsten Eiler's book "You've Been Hacked".
Who is the audience? How can I use the book? How can I explore the app?

...

Where are the instruction located for how to build the Docker files and use the repository?

0x1: Web Security 101

In a nutshell, **to find vulnerabilities in your web application**, ...

1. ... test various values for parameters used by the web application and see what happens (conceptually similar to fuzzing)
2. ... check web application code for bugs that may lead to security vulnerabilities (typically missing checks of input values or missing countermeasures against certain types of attacks)

The Open Web Application Security Project (OWASP) maintains a list of Top 10 vulnerabilities in web applications.

Top 10 Web Application Security Risks

1. **Injection**. Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
2. **Broken Authentication**. Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.
3. **Sensitive Data Exposure**. Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser.
4. **XML External Entities (XXE)**. Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.
5. **Broken Access Control**. Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.

Fahrplan

- Get to know your target
- Test for stateful attacks
- Test for attacks on authentication
- Test for cross-site-scripting (XSS)
- Test for SQL injection
- Test for other injection-based vulnerabilities
- Test for attacks on file operations
- Test for buffer overflows, format strings and integer bugs
- Test for architectural attacks
- Test for attacks on the web server

0x2: Recon

reconnaissance: *n*. 1. Military observation of a region to locate an enemy or ascertain strategic features. 2. Preliminary surveying or research.

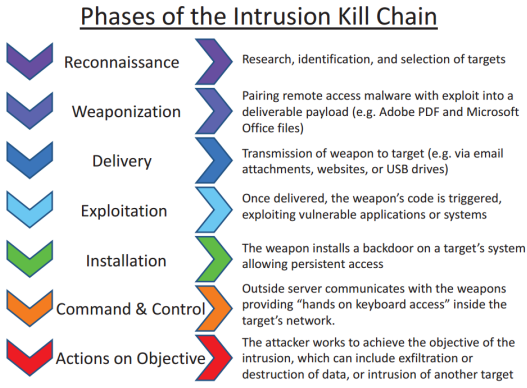
Kill Chain

The term kill chain was originally coined by the military to describe the structure of an attack: finding adversary targets suitable for engagement; fixing their location; tracking and observing; targeting with a suitable weapon or asset to create desired effects; engaging the adversary; assessing the effects;

In 2011, Hutchins et al [1] from Lockheed-Martin expanded this concept to an **intrusion kill chain** for (network) security. They defined intrusion kill chain as reconnaissance, weaponization, delivery, exploitation, installation, command and control (C2), and actions on objectives.

Later on, security organizations have adopted this concept under the name "cyber kill chain".

Intrusion Kill Chain



Source: Wikimedia Commons.

Mimicking the Attacker

Every attack starts by collecting information about the target, e.g., a web application ($\rightarrow$ cyber kill chain).

Likewise, to test a web application for security vulnerabilities, you must first get to know it. You must understand what functions it uses and what parameters these functions have. You have to test every parameter whether it can be exploited (e.g., using illegitimate values). You need to check whether the web application code contains known vulnerabilities.

Systematic collection and documentation allows you to understand what a real attacker would learn and where she could break your application's security.

Collecting Rudimentary Information

Document any **easy-to-spot hints** to security vulnerabilities:

- Suspicious comments in the HTML code (`<!-- default password:...`)
- Sensitive information embedded in the HTML code
- Error messages from the web application
- Error messages from the webserver and http responses

Learning the Web Application Structure

Catalogue **all pages, resources and parameters** that belong to or are used by the web application:

- using a general-purpose tool like `wget`
- using a special-purpose tool like the OWASP Zed Attack Proxy (ZAP) crawler
- by manually visiting all web pages of the application (works well for small web applications)

While `GET` parameters are displayed in the URL, you'll need a web proxy like OWASP ZAP to access `POST` parameters and cookies.

If your web application has different roles (guest, admin, ...), you'll have to test it for all these roles (i.e., first as a guest, then logged in as admin, etc.)

Investigating Individual Web Pages

Visit all pages and **inspect their source code**. Look for things like:

- **Leaky HTML comments** (comments containing e.g., code fragements, configuration parameters, server names, SQL table descriptions, etc.)
- **Hidden input fields** (`<input type='hidden' ...>`)
- **SQL queries** printed in the page source due to programming or configuration errors (reveal the structure of the database and the queries used)
- **IP addresses of internal servers**
- **Web or email addresses**, e.g., email addresses of the developers (might reveal who wrote the application. Maybe it's just an optical tweak of a well-known application?)

Investigating Parameters

Collecting More Information

- Are there resources likes directories and files that are not linked to?
- What else is running on the server? SSH?

Collecting Client-Side Information

Collecting Information about Static Pages

Collecting JavaScript-related Information

Sample frame title

In this slide, some important text will be highlighted because it's important. Please, don't abuse it.

Remark

Sample text

Important theorem

Sample text in red box

Examples

Sample text in green box. The title of the block is "Examples".

Tools

- ZAProxy
- ...

Example

- one

Example

- one
- two

Example

- one
- two
- theorem

Example 1

One

Example 1

One Two

Example 1

One Two Three

0x3: Stateful Attacks

0x4: Attacks on Authentication

0x5: Cross-Site-Scripting (XSS)

0x6: SQL Injection

0x7: Other Injection-Based Vulnerabilities

0x8: Attacks on File Operations

0x9: Buffer Overflows, Format Strings and Integer Bugs

0xA: Architectural Attacks

0xB: Attacks on the Web Server

0xC: Misc

More Things to Consider

One

More Things to Consider

One Two

More Things to Consider

One Two Three

Bibliography


Eric M Hutchins, Michael J Cloppert, Rohan M Amin, et al.
Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains.
Leading Issues in Information Warfare & Security Research, 1(1):80, 2011.