You've Been Hacked

An (Interactive) Course on Web Security

Paul Duplys

@duplys duplys linkedin.com/in/paulduplys/

0x0: Preliminaries

0x2: Attacks on Web Application's State

0x3: Attacks on Authentication

0x4: Cross-Site-Scripting (XSS)

0x5: SQL Injection

0x6: Other Injection-Based Vulnerabilities

0x7: Attacks on File Operations

0x8: Buffer Overflows, Format Strings and Integer Bugs

0x9: Architectural Attacks

0xA: Attacks on the Web Server

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?

whoami

short intro/bio.

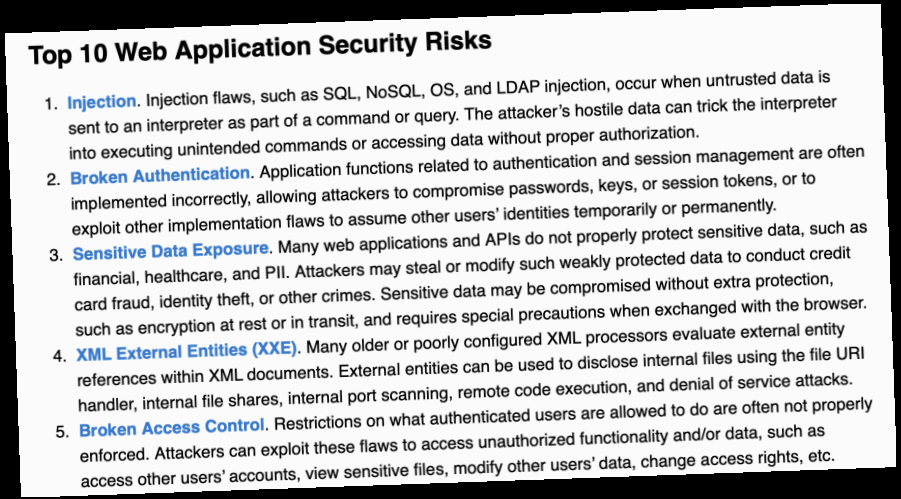
0x0: Preliminaries

Finding Vulnerabilities in Web Applications

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.

Conventions

- Alice, Bob: legitimate users
- Eve: malicious user, attacker
- Server: web server running a web application
- Client: web browser (or computer running the web browser)

GitHub Repo

- https://github.com/duplys/youve-been-hacked
- Dockerfile & setup instructions
- Write-ups
- Code

Building Docker Image

- Running the demo web application in a Docker container is the easiest way to get started
- `Docker` dir contains `Dockerfile` for the vulnerable web application
- Build the container using `make build` or `docker-compose`

Starting Docker Containers

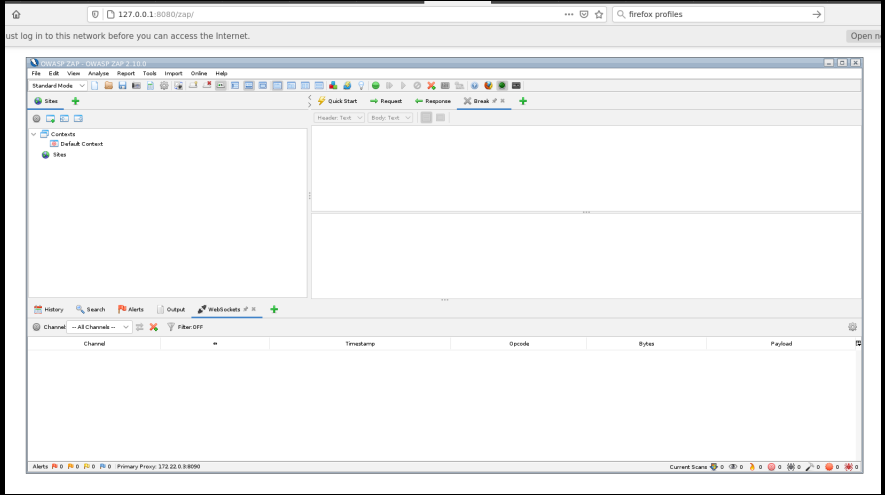
- You'll need an empty directory `tmp` in the `Docker` dir
- In `Docker` dir, run `$ docker-compose up`

Setting up ZAP

You need to activate ZAProxy (ZAP), configure your browser to proxy via ZAP and import the public ZAP Root certificate (see https://www.zaproxy.org/docs/docker/webswing/ for details). Do the following steps:

- Go to `http://127.0.0.1:8080/zap/`
- You'll see how the ZAP Web UI starts
- Start a ZAP session, choose "don't want to persist"
- Select "Update All" in the "Manage Add-ons" window

Setting up ZAP



Setting up ZAP

- Go to "Tools" $\rightarrow$ "Options" $\rightarrow$ "Dynamic SSL Certificates" $\rightarrow$ "Save"
- Save ZAP certificate on your host and import it into your browser.
- Read off the ZAP's IP address and port number at the bottom of ZAP's window
- Configure your web browser to use that IP/port as proxy

Accessing the Vulnerable Web App through ZAP

- Look up the vulnerable app container IP address (for `docker-compose`)
- Run `docker container inspect docker_vulnapp_1` and look for `IPAddress:` under `Networks:`
- If the container's IP address on your machine is `x.y.z.w`, you can access the vulnerable web app under `http://x.y.z.w/daten/kapitel1.html`

Cleaning Up

- Run `$ docker-compose down`

Fahrplan

- Get to know your target
- Test for attacks on web application's state
- 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


0x3: Stateful Attacks

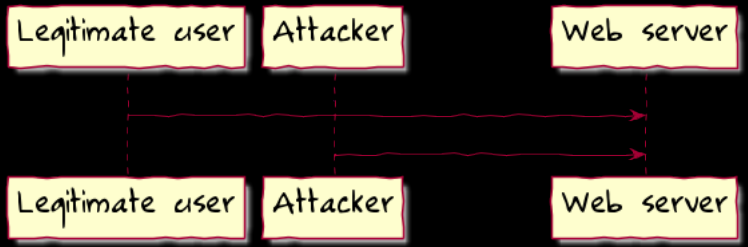
Session Highjacking

- Numerous attacks can be alleviated by storing state information on the server (instead of client)
- In that case, the web application needs a way to identify the user
- The notion of a *session* is used to associate state information (stored on the server) to a specific user
- The web application assigns each session a unique *session identifier* (session-ID)

Session Highjacking

- Session-ID is a number or a combination of numbers and letters that uniquely identifies a session
- Session-ID is assigned to the client (user) upon the first HTTP request to the web application
- Every subsequent request carries the session-ID so that the web application can identify the client (user) and retrieve the corresponding state information stored on the server

Session Highjacking



- Because the session-ID must be stored on the client, it can be manipulated or stolen
- An attack exploiting this vulnerability is referred to as *session hijacking*
- The attackergets hold of the session-ID information to gain unauthorized access to the web application

Session Highjacking

```
Name=Alice
userID=1234
last=2021-08-24
```

- A special case of session hijacking is the so-called *authorization bypass*
- Example: web application identifies the client (user) based on a cookie containing the user name, the user ID and the date of the last session
- Is there are user with ID 1235?
- The attackercan change `userID` and use the manipulated cookie to take on the identity of another user

Finding Session Highjacking Vulnerabilities

- First, identify the session-ID in the web application, e.g., search for parameters whose names contain `ID`, `TOKEN`, `SESSIONID`, `ID` or similar strings. These parameters can be located in hidden input fields, URL parameters or in cookies.
- Once you identified a potential session-ID, change it and check whether you can impersonate another user

Defending against Session Highjacking

- Session hijacking requires a valid session-ID. The attackerhas essentially 4 options to get hold of it:
  - Cross-site-scripting (XSS): XSS attacks are commonly used to steal cookies (more on this later). Make sure your web application has no XSS vulnerabilities. In addition, you can set `HTTPOnly` flag to prevent access to cookies from within JavaScript
  - Eavesdropping network traffic: if the attackercan read network traffic between the client and the web application, she can extract the session-ID. Ensure that you web application uses TLS (HTTPS).
  - Searching logfiles: if the attackerhas access to the web server (or proxy) logfiles, she can search them for session-IDs that were transmitted via `GET` requests. Session-ID should therefore be transmitted in cookies or via `POST` requests.
  - Brute-force attack: the attackercan try to guess the format of your session-IDs and try out various (random) combination until she finds a valid session-ID. You should use large random values as session-IDs.

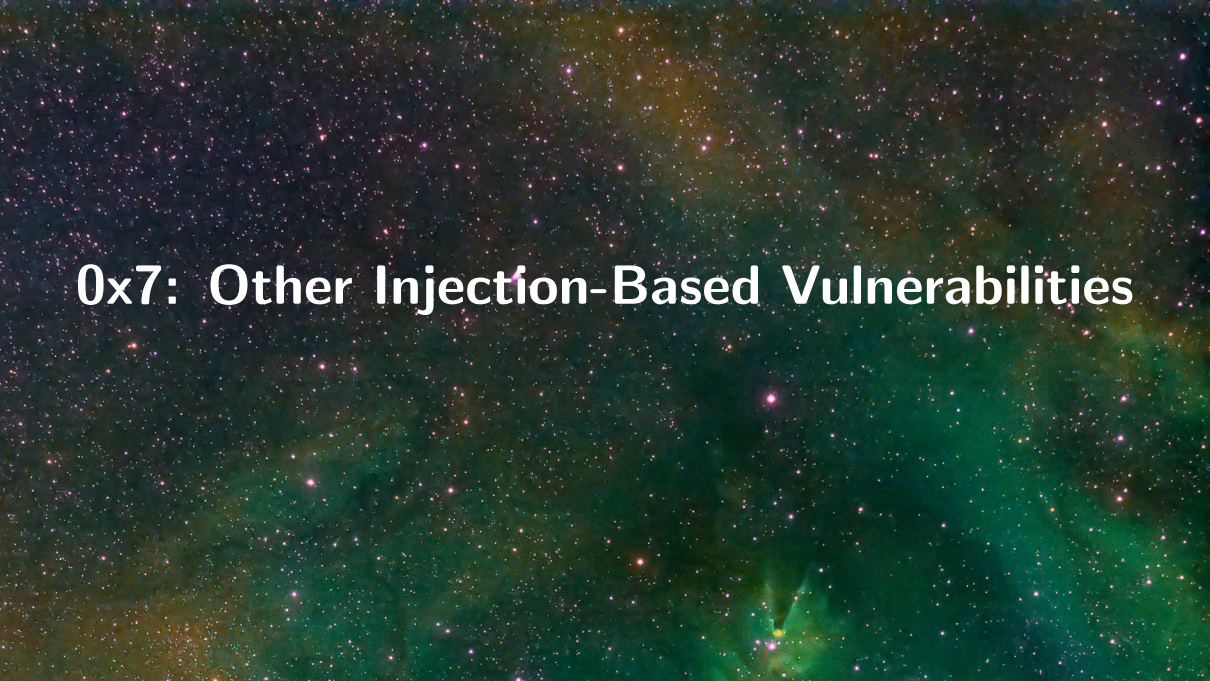
Defending against Session Highjacking

- Performing session identification based on the combination of the session-ID and the IP address of the client makes session hijacking more difficult
- Instead of the IP address, you can also use e.g., the `User-Agent` header (i.e., user the client browser "model" as an additional identification factor)
- To reduce session hijacking-related risks, you should end the sessions after a specific time and make the corresponding session-IDs invalid
- This can be done using a "hard" time-out (e.g., every session ends after 30 minutes) or a "soft" time-out (e.g., session ends after 10 minutes of inactivity)

0x4: Attacks on Authentication

0x5: Cross-Site-Scripting (XSS)

0x6: SQL Injection

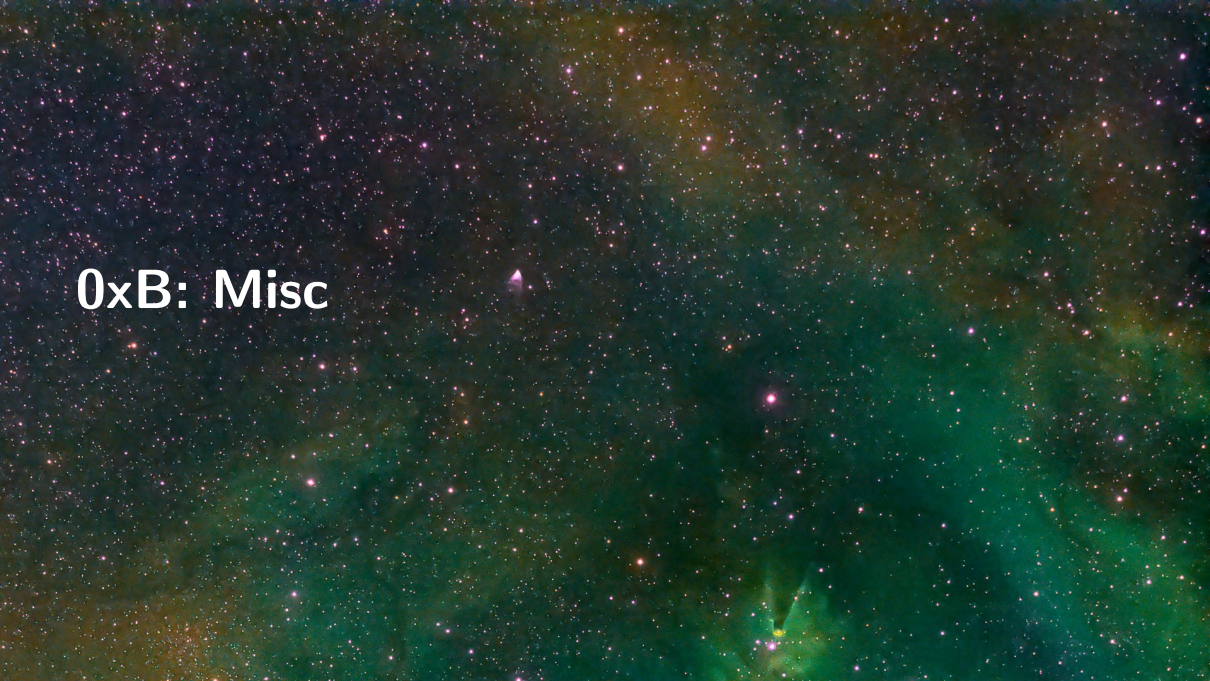

0x7: Other Injection-Based Vulnerabilities

0x8: Attacks on File Operations

0x9: Buffer Overflows, Format Strings and Integer Bugs

0xA: Architectural Attacks

0xB: Attacks on the Web Server


0xB: Misc

Bibliography