You’ve Been Hacked

An (Interactive) Course on Web Security

Paul Duplys

@duplys duplys linkedin.com/in/paulduplys/

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?

Conventions

- Alice, Bob: benign users
- Eve: malicious user & attacker
- Server: web server or service in the cloud running a web application or serving a website
- Client: web browser on a user's computer (and sometimes the entire user's computer)

`docker image ls`

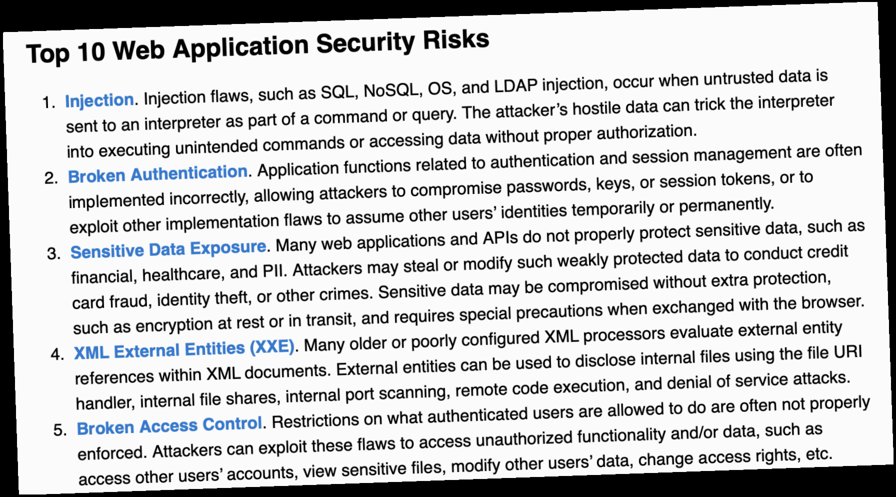
Where are the instruction located for how to build the Docker files and use the repository?

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.

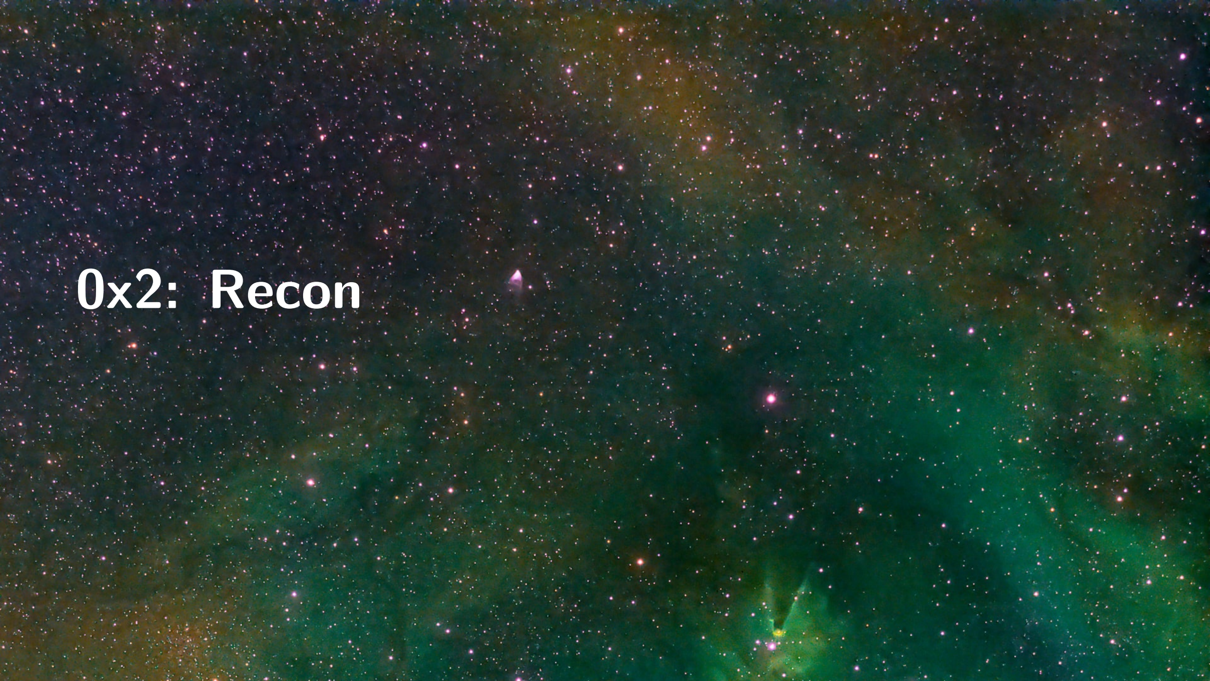


Top 10 Web Application Security Risks

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.

Why Reconnaissance?

Hint: consider the anatomy of a typical cybersecurity attack.

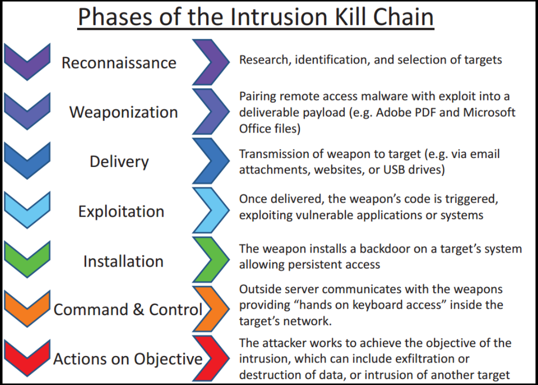
Kill Chain

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



Reconnaissance	Research, identification, and selection of targets
Weaponization	Pairing remote access malware with exploit into a deliverable payload (e.g. Adobe PDF and Microsoft Office files)
Delivery	Transmission of weapon to target (e.g. via email attachments, websites, or USB drives)
Exploitation	Once delivered, the weapon's code is triggered, exploiting vulnerable applications or systems
Installation	The weapon installs a backdoor on a target's system allowing persistent access
Command & Control	Outside server communicates with the weapons providing "hands on keyboard access" inside the target's network.
Actions on Objective	The attacker works to achieve the objective of the intrusion, which can include exfiltration or destruction of data, or intrusion of another target

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 web server 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 fragments, 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

Investigate **all parameters** passed to the application:

- What happens when you change their values or use invalid values, e.g., a string instead of an integer?
- Do invalid parameters return error message that leak information about the application like database table names?

Collecting More Information

- Are there unlinked resources likes directories or files?
- E.g., if there are links to files financial-report18.pdf and financial-report19.pdf, is there an unpublished file financial-report20.pdf?
- Are there any hints to the structure of file names or sub directories?
- E.g., if the web application contains user profiles, is it possible to access an arbitrary profile using `user-[number].html` or `user.php?id=[number]`?
- Are there unlinked sub-directories like `test/` or `admin/`?
- Does the web server contain unused libraries or example application code that contains hints to known vulnerabilities or the internals of the web application?
- What else is running on the server? SSH?

Investigating Client-Side Code

What input parameters are sanitized in the client-side code (JavaScript or plain HTML)?

Investigating Client-Side Code: Static Pages

If user input is checked, i.e., sanitized, at client side (either in JavaScript or in HTML), chances are that no sanitization is implemented on the server.
Because an attacker can easily manipulate the client-side code, she can manipulate the user inputs sent to the server.

Investigating Client-Side Code: Static Pages

Check **all input fields** like text input, drop-down menus, radio buttons and select tags for **constraints for their values**. E.g., is there a maximum text length for a text field? Are numeric input values scoped? If yes, change these values and check how the web application is behaving.

Investigating Client-Side Code: Static Pages

There are 3 ways to change the client-side values:

- Parameters transmitted via `GET` requests can be manipulated directly in the URL
- Parameters transmitted via `POST` requests can be manipulated directly in a local copy of the website code
- On the fly, using a proxy like the OWASP ZAP or web developer tools in the web browser

Investigating Client-Side Code: Static Pages

Hidden forms (`type="hidden"`) are sometimes used to store values used by the web application. A classic mistake is to store the price of the items in a web shop application (since you can easily manipulate them before sending them to the server).

A Trivial Example

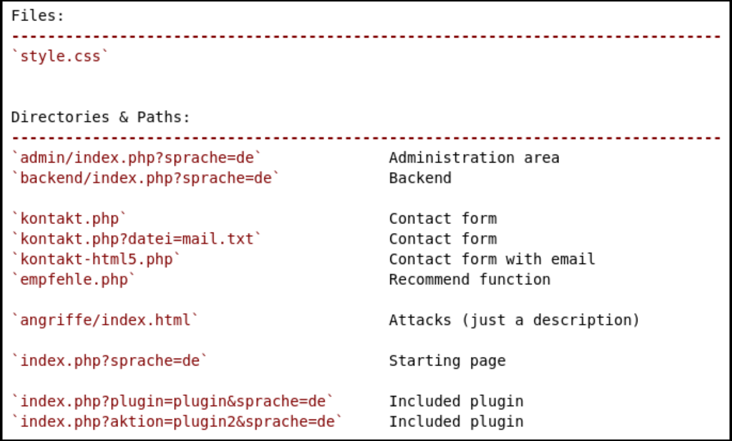
Say there is a drop-down menu to select the number of items to be added to a shopping cart. The drop-down menu allows numbers in the range 0 to 100. What happens when you set this parameter to a negative value and transmit it to the server? Maybe there is no sanitization on the server side because only values greater or equal to zero can be selected from the drop-down menu? In this case, the final price is calculated by multiplying the price of the item with a negative number.

Collecting JavaScript-related Information

After checking HTML, next step is to check the JavaScript code.

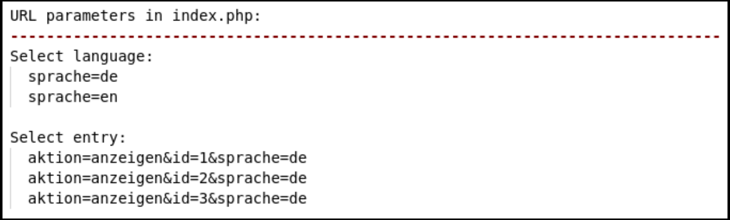
- Is JavaScript used to validate input data? If so, maybe the check on the server side is omitted.
- What files and scripts are being loaded? Any known vulnerabilities?
- Are there vulnerabilities in the JavaScript program logic itself? Any logic flaws that can be exploited?

Reconnaissance of the Demo Application



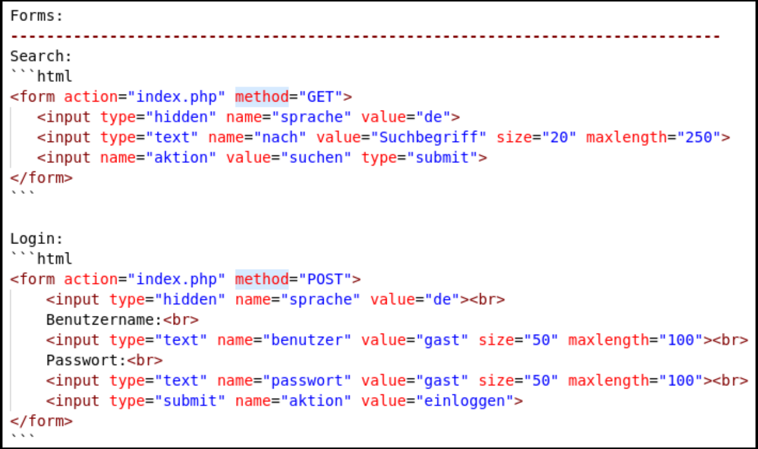
```
Files:
--------------------------------------------------------------------------------
`style.css`


Directories & Paths:
--------------------------------------------------------------------------------
`admin/index.php?sprache=de`               Administration area
`backend/index.php?sprache=de`             Backend

`kontakt.php`                              Contact form
`kontakt.php?datei=mail.txt`               Contact form
`kontakt-html5.php`                        Contact form with email
`empfehle.php`                             Recommend function

`angriffe/index.html`                      Attacks (just a description)

`index.php?sprache=de`                     Starting page

`index.php?plugin=plugin&sprache=de`       Included plugin
`index.php?aktion=plugin2&sprache=de`      Included plugin
```

Reconnaissance of the Demo Application



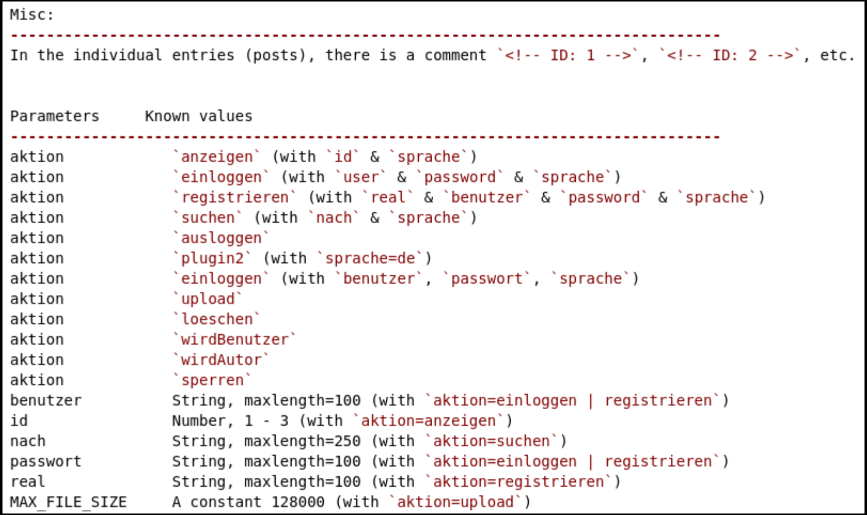
```
URL parameters in index.php:
--------------------------------------------------------------------------------
Select language:
  sprache=de
  sprache=en

Select entry:
  aktion=anzeigen&id=1&sprache=de
  aktion=anzeigen&id=2&sprache=de
  aktion=anzeigen&id=3&sprache=de
```

Reconnaissance of the Demo Application



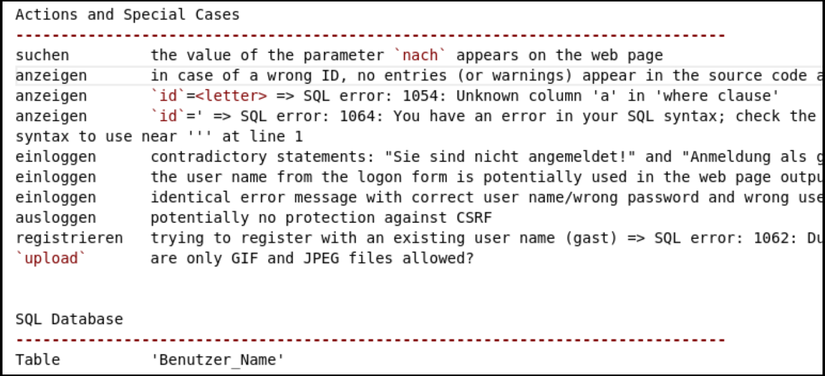
~~~
Forms:
--------------------------------------------------------------------------------
Search:
```html
<form action="index.php" method="GET">
   <input type="hidden" name="sprache" value="de">
   <input type="text" name="nach" value="Suchbegriff" size="20" maxlength="250">
   <input name="aktion" value="suchen" type="submit">
</form>
```

Login:
```html
<form action="index.php" method="POST">
    <input type="hidden" name="sprache" value="de"><br>
    Benutzername:<br>
    <input type="text" name="benutzer" value="gast" size="50" maxlength="100"><br>
    Passwort:<br>
    <input type="text" name="passwort" value="gast" size="50" maxlength="100"><br>
    <input type="submit" name="aktion" value="einloggen">
</form>
```
~~~

Reconnaissance of the Demo Application



```
Misc:
--------------------------------------------------------------------------------
In the individual entries (posts), there is a comment `<!-- ID: 1 -->`, `<!-- ID: 2 -->`, etc.


Parameters     Known values
--------------------------------------------------------------------------------
aktion           `anzeigen` (with `id` & `sprache`)
aktion           `einloggen` (with `user` & `password` & `sprache`)
aktion           `registrieren` (with `real` & `benutzer` & `password` & `sprache`)
aktion           `suchen` (with `nach` & `sprache`)
aktion           `ausloggen`
aktion           `plugin2` (with `sprache=de`)
aktion           `einloggen` (with `benutzer`, `passwort`, `sprache`)
aktion           `upload`
aktion           `loeschen`
aktion           `wirdBenutzer`
aktion           `wirdAutor`
aktion           `sperren`
benutzer         String, maxlength=100 (with `aktion=einloggen | registrieren`)
id               Number, 1 - 3 (with `aktion=anzeigen`)
nach             String, maxlength=250 (with `aktion=suchen`)
passwort         String, maxlength=100 (with `aktion=einloggen | registrieren`)
real             String, maxlength=100 (with `aktion=registrieren`)
MAX_FILE_SIZE    A constant 128000 (with `aktion=upload`)
```

Reconnaissance of the Demo Application



```
Actions and Special Cases
--------------------------------------------------------------------------------
suchen         the value of the parameter `nach` appears on the web page
anzeigen       in case of a wrong ID, no entries (or warnings) appear in the source code a
anzeigen       `id`=<letter> => SQL error: 1054: Unknown column 'a' in 'where clause'
anzeigen       `id`=' => SQL error: 1064: You have an error in your SQL syntax; check the
syntax to use near ''' at line 1
einloggen      contradictory statements: "Sie sind nicht angemeldet!" and "Anmeldung als g
einloggen      the user name from the logon form is potentially used in the web page outpu
einloggen      identical error message with correct user name/wrong password and wrong use
ausloggen      potentially no protection against CSRF
registrieren   trying to register with an existing user name (gast) => SQL error: 1062: Du
`upload`       are only GIF and JPEG files allowed?


SQL Database
--------------------------------------------------------------------------------
Table          'Benutzer_Name'
```

0x3: Stateful Attacks

State

- HTTP, the protocol for transfering data between the server and the client, is stateless
- Example: HTTP doesn't manage information about pages previously visited by the client, i.e., any URL can be requested by the client at any time.
- If statefulness is required (e.g., page B shall only be served if the client has previously visited page A), the web application must manage a *session*

statefulness: Web Shop Example

- Malicious user adds items into the shopping cart
- She then skips the page where credit card information must be entered and proceeds directly to the checkout page
- If the web application doesn't enforce the correct sequence of pages, the user would succesfully place an order without having entered any payment details

Managing State Information

There are two options to manage state information in web:

- the state information is stored on the server; the users are identified using a *session ID*
- the state information is stored on the client

Hidden Input Fields

State information can be located in hidden input fields, e.g.,

```
<input name="id" value="1234" type="hidden">
```

Such data fields can be easily manipulated using a tool like ZAP-Proxy or using web developer tools in the web browser. Thus, an attacker can easily manipulate hidden input fields.

Hidden Input Fields: Finding Vulnerabilities

For every hidden input field in the application, you must examine whether the web application's behavior changes if the values of these input fields are altered.
A classical anti-pattern (luckily not so common anymore) is the use of hidden input fields for storing item price in a web shop. Yet another example is the storage of the user's status, e.g., a signed in user instead of a "guest" or administrator instead of a standard user.

Hidden Input Fields: Protecting Against Attacks

- Core issue with hidden input fields: state information is stored *on the client* where it can be easily manipulated by a malicious user
- Main defense philosophy: *never trust the client*
- Critical information must always be stored on the server
- Values received from the client must always be checked & validated
- If values must be stored on the client (e.g., session IDs), they should be encrypted or hashed

URL Parameters

- (State) information can be transmitted in the URL
- In contrast to forms, transmitting information via URL parameters does not require clicking a submit button

URL Parameters: Finding Vulnerabilities

- Using URL parameters to transmit information is a threat, because it is trivial to manipulate them
- Look for URL parameters identified during the recon phase
- Investigate what happens when you change these parameters. Does this cause an unexpected and unintended change of the state of the web application?
- `http://www.webapp.example/editprofile.php?id=123`
- What happens when you change `id`? Can you edit the profile of a different user?
- Are there (hidden) parameters to activate debug information, e.g., `debug=on`, `debug=1` or `debug=true`?

URL Parameters: Defending Against Attacks

- Core issue with URL parameters: state information is stored *on the client* where it can be easily manipulated by a malicious user
- URL parameters must always be checked & validated
- If possible, they should be encrypted or hashed

Cookie Parameters

- For a long time, cookies were the only option to persistently store data on the client
- It is still the prevailing solution
- Cookies are frequently used for user identification, e.g., at consecutive visits of a web application
- Attacks based on cookie manipulation are called *cookie poisoning*

Cookies

- Persistent vs non-persistent cookies
- `secure` vs `HttpOnly` cookies
- Persistent cookies are stored on client's hard drive as long as their date is validated
- Non-persistent cookies are stored in RAM and are deleted when the web browser is closed
- `secure` cookies are transmitted only via an HTTPS connection
- `HttpOnly` cookies are transmitted via HTTP or HTTPS, but cannot be accessed by JavaScript

Cookie Parameters: Storage and Manipulation

- All web browsers store cookies in known filesystem locations and in known formats
- An attacker can easily manipulate this data before it is used by a web application
- It is possible to manipulate cookies *on the fly*, e.g., using a tool like the ZAProxy

Cookie Parameters: Storage and Manipulation

- Unlike with URL parameters, the attacker can also manipulate the date when the cookie expires, i.e., she can manipulate the cookie's lifetime
- In general, cookies can give you access to things like sessions, profiles, etc.

Cookie Parameters: Storage and Manipulation

Example:

- `weather.xyz` offers detailed weather information against payment
- `weather.xyz` uses cookies that contain `userID` to identify users
- The cookie is valid for the user's subscription period, e.g., a month
- Information stored in the cookie is processed by `weather.xyz` without additional authentication
- Attack 1: malicious user manipulates `userID` to acess `weather.xyz` as another user
- Attack 2: malicious user extends their subscription by manipulating the cookie expiration date (by changing the Unix-timestamp in the cookie)
- Attack 3: cookies often store the number of failed login attempts. When they reach a specific threshold, say 5, `weather.xyz` deactivates that user account for 10 minutes to protect against brute force attacks. The attacker bypasses this defense by setting the value of failed login attempts to 0 after each login attempt. Since this can be easily automated, brute force attacks become trivial.

Cookie Parameters: Finding Vulnerabilities

- For every cookie found during recon phase, you must check whether a parameter manipulation results in an illegal state change of the web application
- Manipulation should also include cookie parameters like expiration
- To detect cookie-related vulnerabilities more efficiently, *decrease* the expiration date and check whether the web application denies its service
- As an alternative – if you have access to the server-side source code of the web application – check whether the source code uses the cookie expiration date as input parameter

Cookie Parameters: Defending Against Attacks

- In general, cookie poisoning – attacks that manipulate cookies – cannot be eliminated
- Cookies are stored on the client and a malicious user can always manipulate data on the client (even data of other users)
- Two prominent attacks in this context are *cross-site-scripting* (XSS) and *cross-site-request-forgery* (CSRF)
- Attacker exploits web browser capabilities to set a cookie for a different domain

Cookie Parameters: Defending Against Attacks

- As a general rule, never trust data supplied by the client
- For example, store subscription's expiration date or the number of failed login attempts on the server
- If storing data on the client is unavoidable, encrypt or hash that data
    - Question to the audience: why?

URL Jumping

- Eve abuses the intended sequence of web pages. This gives her access to functions that by design must be unavailable at that point in time.

URL Jumping: An Example

In a typical web shop, the intended sequence of transactions – i.e., pages that a shop's customer visits – could look something like this:

1. Search for a suitable product at `www.shop.xyz/catalogue.php`
2. add an article to the shopping cart at `www.shop.xyz/add.php`
3. verify the customer order at `www.shop.xyz/order.php`
4. provide shipping information (customer's postal address) at `www.shop.xyz/shipping.php`
5. enter credit card information at `www.shop.xyz/payment.php`
6. checkout & finish the order at `www.shop.xyz/checkout.php`

URL Jumping: An Example

- If the attackercan jump from step 4 (`www.shop.xyz/shipping.php`) directly to step 6 (`www.shop.xyz/checkout.php`), she might be able to order an item without paying for it
- Another example: the attackercould try to post a message to an online forum with creating an account
- In general, the attackertries to identify URLs that must be called in a specific sequence and tries to manipulate this sequence

URL Jumping: Locating the Vulnerabilities

- Make a complete list of URLs in your web application and whether certain URLs must be called in a specific order
- Use information gathered during recon phase
- For every URL sequece:
    - Call the URLs in the intended order and document the parameters passed to the web application
    - Diverge from the intended sequence and see what happens. Are there any error messages (e.g., "You first must create a user") or are specific URL/pages not accessable?
    - If there is no error message and the URLs can be accessed in any order, check whether this represents an actual, exploitable vulnerability

Defense against URL Jumping

- URL jumping is a security issue only when the sequence in which URLs are accessed is relevant for the web application
- In that case, you need to check that the correct sequence is used, e.g., `checkout.php` would check if the user previously visited `payment.php` wich, in turn, would check if the user previously visited `shipping.php`

Defending against URL Jumping

There are 3 options to identify a previously visited URL:

1. The URL (or the corresponding ID) is stored in a hidden input field or in an URL parameter
2. The URL (or the corresponding ID) is stored in a cookie
3. The URL, from which the user is allegedly coming, is compared to the `Referer` field value in the HTTP header

Defending against URL Jumping

- As shown previously, option 1 (hidden input field) and option 2 (cookie) are insecure as they can be easily manipulate by the attacker
- That leaves us with option 3 (`Referer` value)

An HTTP request could look like this:

```
Get /checkout.php HTTP/1.1
Host: www.shop.xyz
User-Agent: Mozilla/5.0 (Macintosh; U; PPC Mac OS X; de-de) [.
Accept: */*
Accept-Encoding: gzip, deflate
Accept-Language: de-de
Connection: keep-alive
Referer: https://www.shop.xyz/payment.php
```

Defending against URL Jumping

- The `Referer` value is automatically set by the web browser
- In case of a bookmark or a direct entry in the web browser's address field the `Referer` value does not exist in the HTTP header
- However, the `Referer` value can be omitted by the web browser (for privacy reasons), removed from the HTTP request by a proxy or manipulated by the attacker
- `Referer` value can be manipulated using a tool like ZAProxy or using a dedicated client where the individual values of the HTTP header can be defined manually
- So `Referer` value is also insecure

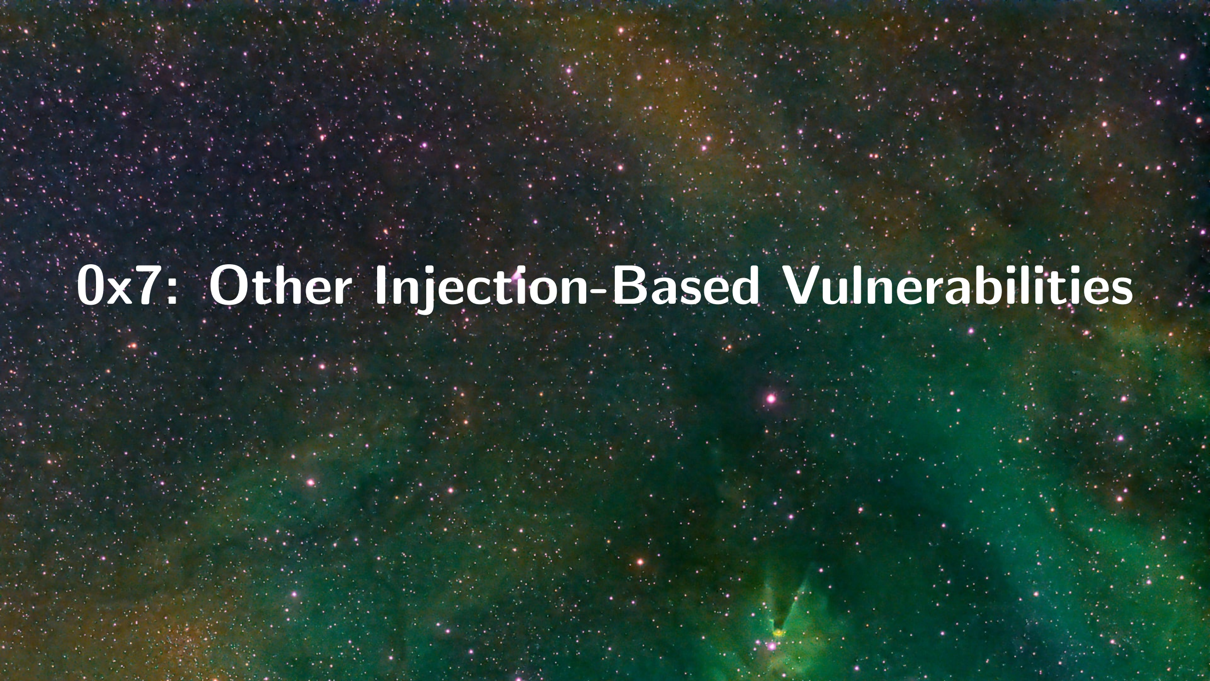
Defending against URL Jumping

- In the end, the only effective defense against URL jumping – similar to other attacks – is to store state information on the server
- Specifically, the information about the visited URLs for each user can be stored temporarily in the web application (or persistently in a database)

0x4: Attacks on Authentication

0x5: Cross-Site-Scripting (XSS)

0x6: SQL Injection



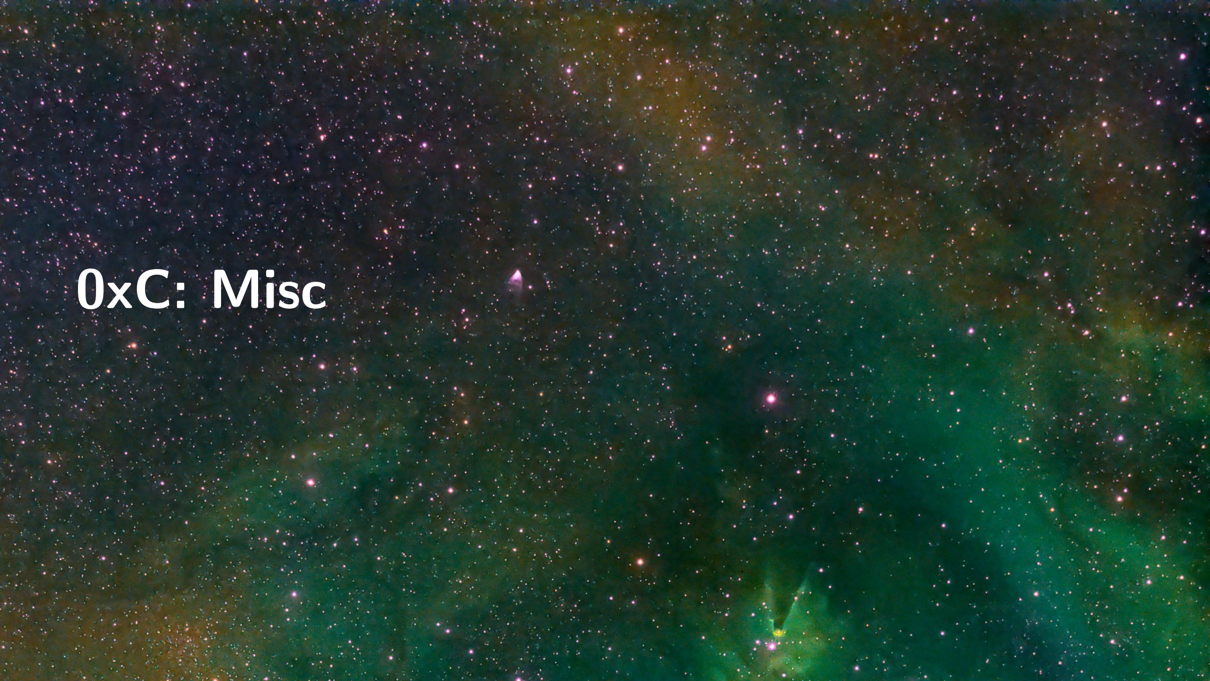
0x7: Other Injection-Based Vulnerabilities

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.