



# Web Exploitation

Eine Einführung

Martin Wagner | 27. April 2023

```
sc[] = "\x6a\x0b" // push byte +0xb
// pop eax
// cdq
// push edx
"\x2f\x73\x68" // push dword 0x6
"\x62\x69\x6e" // push dword 0x6
// mov ebx, esp
// xor ecx, ecx
// int 0x80
```

# Überblick

- Client-Server-Architektur
  - Fehlende Validierung
- Serverseitige Angriffe
  - SQLi
  - Command Injection
- Clientseitige Angriffe
  - XSS
- Tools

# Client-Server-Architektur

- Webanwendungen bestehen aus (mindestens) zwei Teilen
  - Webserver: PHP, Python, JavaScript
  - Web-Client: Website, App, API-Client
- Beide können angegriffen werden
  - Fokus zu Beginn auf Angriffe gegen Server
    - Ziel: Datei, Datenbankeintrag, RCE
  - Auch Angriffe gegen Clients möglich
    - Ziel: Cookies, localStorage

# Der Client

- Wir sind der Client. Wir können alles (ausprobieren)
  - Gegebener Client nur Inspiration dafür wie Anwendung genutzt werden kann
  - Konkretes dazu am Ende bei den Tools
- Jegliche Validierung von Daten auf dem Client wertlos
- Verstecken von Daten im Client unmöglich

# Der Server

- Häufig in Skriptsprachen geschrieben
  - In vielen Fällen schwach typisiert
- Oft sehr tolerant was Akzeptanz von Eingaben angeht
  - `1 == "1", "text" == true`
  - PHP und Javascript haben `===` Operator

# Serverseitige Angriffe

- Injection Angriffe
  - SQL
  - Commands
  - XXE (XML Injection)
- Logik Bugs
  - Fehlende Authentifizierung
  - Versteckte Endpunkte (`/.env`, `/.git/index`, `/robots.txt`)



# SQLi - Grundlagen

```
$username = $_GET['username'];  
  
$result = mysql_query(  
    "select * from users where username='$username'"  
);  
  
print_r($result);
```

# SQLi - Grundlagen

```
$username = "mawalu"
```

```
$result = mysql_query(  
    "select * from users where username='mawalu'"  
);
```

⇒ Läd den Nutzer mawalu



# SQLi - Grundlagen

```
$username = ""
```

```
$result = mysql_query(  
    "select * from users where username=''"  
);
```

⇒ Fehler, ungültige SQL Anfrage

# SQLi - Grundlagen

```
$username = "' OR 1=1 -- Kommentar"
```

```
$result = mysql_query(  
    "select * from users where username=' ' OR 1=1 -- Kommentar"  
);
```

⇒ Läd alle Nutzer aus der Datenbank



# SQLi - Sidechannel

```
$username = $_GET['username'];  
  
$result = mysql_query(  
    "select * from users where username='$username'"  
);  
  
if (count($result) < 1) {  
    die("User not found!");  
}
```



## SQLi - Sidechannel

```
$username = "admin' and substr(password, 0, 1) == 'a' --";

$result = mysql_query(
    "select * from users where username='admin' and substr(password, 0, 1) == 'a' --'"
);

if (count($result) < 1) {
    die("User not found!");
}
```

⇒ Passwort kann Zeichen für Zeichen gebruteforced werden



## SQLi - Mitigation

```
$stmt = $db->prepare('select * from users where username = ?');  
$user = $stmt->query($_GET['username']);
```

⇒ Datenbankserver kennt Struktur der Anfrage bevor Nutzerdaten gesendet werden



# Command Injection

```
$file = $_GET['filename'];
```

```
system("rm $filename");
```



# Command Injection

```
$file = "test.log";  
  
system("rm test.log");
```



# Command Injection

```
$file = "test.log; cp /etc/passwd test.log"
```

```
system("rm test.log");
```

⇒ Mehrere Zeichen möglich: ; && & || |





# Local file inclusion

```
$page = $_GET['page'];  
include("pages/$page");
```



## Local file inclusion

```
$page = "../../../../../../../etc/passwd";  
include("pages/../../../../../../../../etc/passwd");
```

# Injections allgemein

- Schnittstelle zwischen Systemen
  - Webserver und Datenbankserver
- Konvertierung von Daten zu Code
  - Query-String wird vom DB-Server ausgeführt
- Verschiedene Interpretationen des gleichen Wertes
  - Beide Seiten verhalten sich in ihrer Welt korrekt
- Problem in allen möglichen Situationen in nahezu allen Sprachen



# Clientseitige Angriffe

- Cross Site Scripting (XSS)
- Ausführung von böartigem Javascript im Browser des Opfers
- Ziel: Zugriff auf Secrets: Session ID, API Schlüssel etc



# Reflected XSS

```
https://shop.com/search?q=test
```

```
<h1>Ergebnisse für "test"</h1>
```

...



# Reflected XSS

```
https://shop.com/search?q=<script>alert(1)</script>
```

```
<h1>Ergebnisse für "<script>alert(1)</script>"</h1>
```

...

⇒ Eigener Code wird ausgeführt, wenn der Nutzer auf die Seite geht

# XSS - Mitigations

- HTML Tags in Nutzereingaben serialisieren
  - `<script>` => `&lt;script&gt;`
- Content Security Policy (CSP)
  - Webserver legt genau fest, welche Skripte ausgeführt werden können



# Tools

- Anfragen untersuchen
- Eigene Anfragen senden
- Anfragen automatisieren
  - Z.B. um Daten Zeichen für Zeichen zu leaken



# Tools - Devtools





# Tools - HTTP Client

- curl
- Insomnia
- Postman



# Tools - Scripting

- python mit `requests` Paket
- node.js mit `fetch`
  - Firefox hat "Copy request as fetch"
- Shells scripting mit `curl` oder Powershell

# Tools - Burp





# Resources

- OWASP: <https://owasp.org/www-community/attacks/>
- XS Leaks: <https://xsleaks.dev/>
- Portswigger Academy: <https://portswigger.net/web-security>



# Challenges

- OverTheWire Natas: <https://overthewire.org/wargames/natas/>
- picoCTF <https://www.picoctf.org/>
- Portswigger Academy