



#### Eine Einführung

Martin Wagner | 27. April 2023



1/30 27. April 2023 KITCTF: Web Exploitation kitctf.de

### Überblick



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





- 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)





```
$username = $_GET['username'];

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

print_r($result);
```





```
$username = "mawalu"
$result = mysql_query(
  "select * from users where username='mawalu'"
⇒ Läd den Nutzer mawalu
```





```
$username = "'"

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

$\Rightarrow$ Fehler, ungültige SQL Anfrage
```





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

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

⇒ Läd alle Nutzer aus der Datenbank
```





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





```
$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</pre>
```





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

⇒ Datenbankserver kennt Struktur der Anfrage bevor Nutzerdaten gesendet werden





```
$file = $_GET['filename'];
system("rm $filename");
```





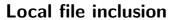
```
$file = "test.log";
system("rm test.log");
```





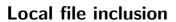
```
$file = "test.log; cp /etc/passwd test.log"
system("rm test.log");

>> Mehrere Zeichen möglich:; && & || |
```





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





```
$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





- Cross Site Scripting (XSS)
- Ausführung von bösartigem 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"
- Shellscripting 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