VORTRAG 3

LOGICAL FLAWS & INFORMATION LEAKS

Vortragender: Philipp Wenskus

AUTOMAT

- Getränke über Auswahl-Knöpfe
- Bezahlung vor Auswahl oder danach
  - Nach Auswahl+Bezahlung Getränk
  - Nach Bezahlung+Auswahl Getränk
- Wenn Getränk leer oder Abbrechen gedrückt → Geld zurück

What could possibly go wrong?

RELEVANTER

Online-Shop Rabattaktion

- "5€ Preiserlass, aber nur ab Warenwert von 20€"
  - Bei Eingabe von Rabattcode wird Warenwert geprüft
- Rabattcode gilt für einen Einkauf
  - Nach Verwendung wird Cookie gesetzt
- Codes sind kurz und einprägsam
  - 5 Zeichen A-Z0-9

What could possibly go wrong?

DEFINITION

- Jedes System hat Regeln
  - White-box: Ablesbar
  - Black-box: Durch Interaktion herausfindbar
- Fehler in den Regeln → Logical Flaws
  - Konzeptionelle Fehler
    - Fall wurde nicht bedacht
  - Programmierfehler
    - z.B. goto fail :-) von Apple

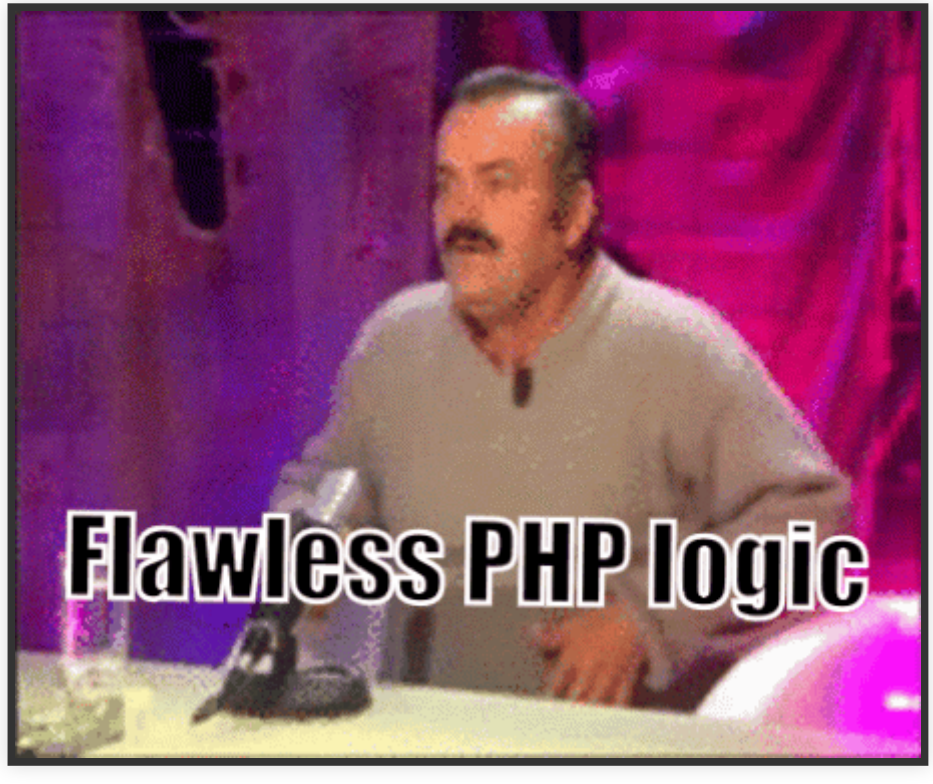
INSUFFICIENT AUTHENTICATION

- Zugangsberechtigung wird...
  - ... nicht richtig geprüft
  - ... nicht überall geprüft
  - ... nicht ausreichend geprüft

PROBLEME MIT SESSIONS

- Fixation
  - unterschieben einer Session
  - Angreifer wartet auf Login
- Hijacking
- Expiration
  - Wie oft?, Logout button?, Wann verwerfen?, ...

PHP



PHP QUIRK #1

```php
if (isset($_GET['name'])) {
    echo htmlentities(trim((string)$_GET['name']));
}
```

Probleme?

PHP nimmt Arrays als GET Parameter an!

Demo

ARRAYS ALS PARAMETER

- PHP erlaubt Arrays in…
  - $_GET (Query String) → ?name[]=0&name[]=1
  - $_POST (POST body) → name[]=0&name[]=1
- Nützlich z.B. für Mehrfachauswahl-Listen
- ~~Daher immer auf den richtigen Typ casten~~
- Daher immer prüfen mit is_string()/is_array()

PHP QUIRK #1

in sicher

```php
if (isset($_GET['name']) && is_string($_GET['name'])) {
    echo htmlentities(trim($_GET['name']));
}
```

Und bei Zahlen?

```php
$num = intval($_POST['num']);
```

PHP QUIRK #2

```php
$stored_hash = '0e133713371337133713371337133713371337';
if (is_string($_POST['pw'])) {
    $hashed_input = md5($_POST['pw']);
    if ($hashed_input == $stored_hash) {
        // eingeloggt
    } else {
        // Fehler
```

== ist ein typunsicherer Vergleich.

`1 == "1" => true`

Wie nutzen wir das Ganze aus?

"0e912..." == "0e1337..."

Demo

MAGICAL HASHES

```
var_dump(md5('240610708') == md5('QNKCDZO'));
var_dump(md5('aabg7XSs') == md5('aabC9RqS'));
var_dump(sha1('aaroZmOk') == sha1('aaK1STfY'));
var_dump(sha1('aaO8zKZF') == sha1('aa3OFF9m'));
```

DIE TYPSICHERHEIT

- == ist nicht typsicher → Type Juggeling
- Beispiele
  - "1e3" == "1000" → true
  - "0e324" == "0e1241421989" → true (wissenschaftliche Notation)
  - 0123 == "83" → true (Octal Darstellung)

Lösung: Typsicherer Vergleich ===

```
if ($hashed_input === $stored_hash) {
```

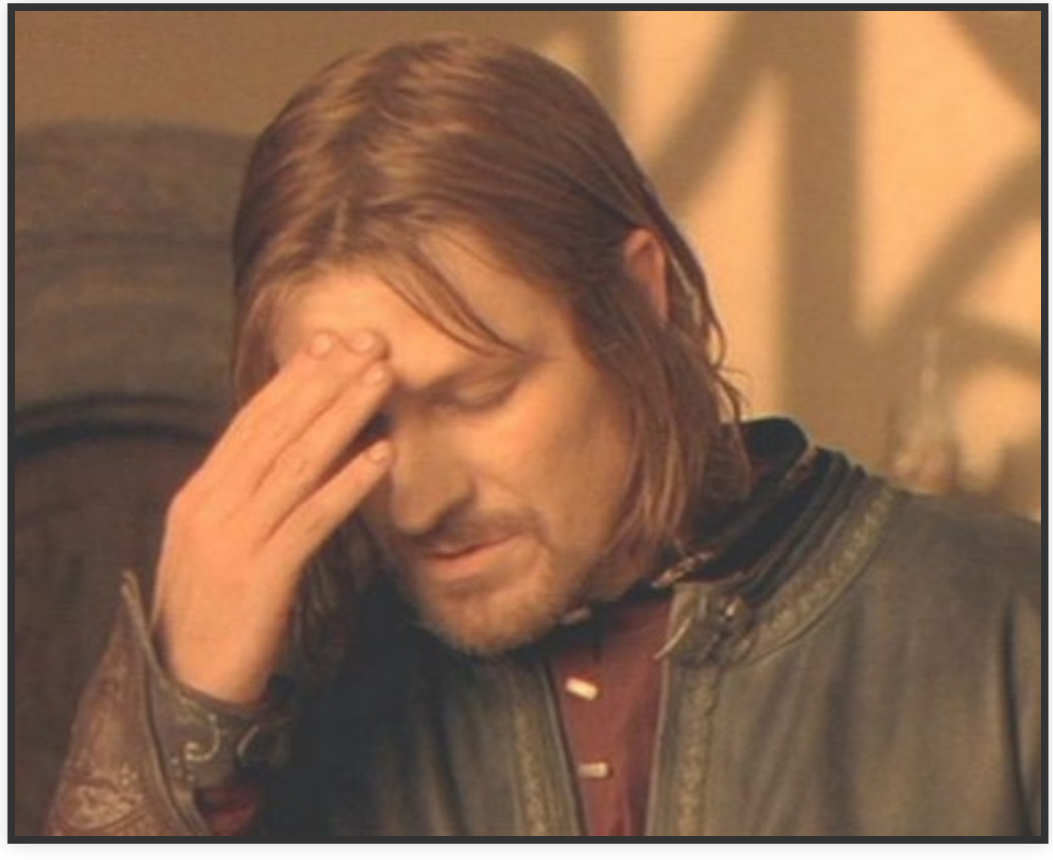
VERGLEICHSTABELLE

Loose comparisons with ==

	TRUE	FALSE	1	0	-1	"1"	"0"	"-1"	NULL	array()	"php"	""
TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
1	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
0	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE
-1	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
"1"	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
"0"	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
"-1"	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
NULL	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
array()	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
"php"	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
""	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE

Quelle: hydrasky.com

DIE TYPSICHERHEIT



QUIRKS KOMBO!!!

```php
if (strcmp($_GET['password'], $password) == 0) {
    // eingeloggt
} else {
    // Fehler
}
```

Erinnerung: == ist typunsicher

$_GET kann auch Arrays enthalten.

strcmp

?password[]=1

Demo

ERKLÄRUNG

- strcmp(array(), 'string') === NULL
  - Mit PHP Warning
- NULL == 0 ist true
  - Typecasting.

```php
// "sicher"
if (is_string($_GET['password'])
        && strcmp($_GET['password'], $password) === 0) {
    // eingeloggt
} else {
    // Fehler
}
```

ERKLÄRUNG

PHP QUIRK #3

```php
if (is_string($_GET['pw']) && $_GET['pw'] === $password) {
    $admin = true;
}

if (isset($admin) && $admin) {
    // Admin-Zugang
}
```

Sicher?

Ist sicher!

Außer mit falscher Konfiguration…

REGISTER GLOBALS

- php.ini-Einstellung
  - register_globals=1
- Benutzereingaben als globale Variablen
  - Alles aus $_REQUEST
- Angriff: ?admin=1
- **Aber:** Entfernt aus PHP (seit Version 5.4.0)

REGISTER GLOBALS

- Trotzdem ein Problem?
- Genug Möglichkeiten Verhalten zu emulieren

```
extract($_REQUEST);
```

```
parse_str($_SERVER['QUERY_STRING']);
```

```
foreach ($_REQUEST as $k => $v) {
    $$k = $v;
}
```

Alle Variablen vor "Emulationscode": Überschreibbar.

PHP QUIRK #4

```
$role = !logged_in() ? "guest" : is_admin() ? "admin" : "user";
```

Solange !logged_in() === true, $role immer "admin"

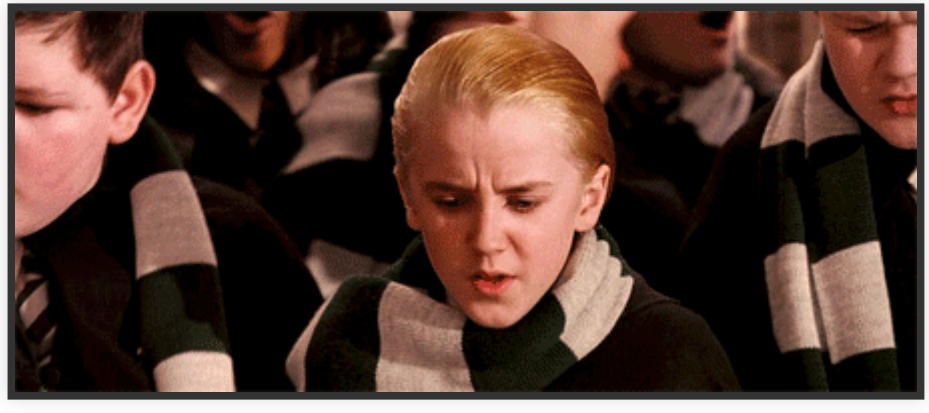
TERNÄRER OPERATOR

- ternärer Operator ist links-assoziativ
  - ... und damit anders als die meisten Sprachen
- Immer ordentlich Klammern!

```
echo (FALSE ? "a" : FALSE ? "b" : "c");  // c
echo (FALSE ? "a" : TRUE  ? "b" : "c");  // b
echo (TRUE  ? "a" : FALSE ? "b" : "c");  // b
echo (TRUE  ? "a" : TRUE  ? "b" : "c");  // b
```

```
echo (TRUE  ? "a" : FALSE) ? "b" : "c");
echo ("a" ? "b" : "c"); // "a" == TRUE
echo (TRUE ? "b" : "c") // b
```

TERNÄRER OPERATOR



PHP 8 TO THE RESCUE

- Veröffentlicht am 26. November 2020
- Viele sinnvolle Änderungen
- u.a. sind typunsichere jetzt Vergleiche sicherer

```
Comparison       | Before    | After
-----------------------------------
0 == "0"         | true      | true
0 == "0.0"       | true      | true
0 == "foo"       | true      | false
0 == ""          | true      | false
42 == "   42"    | true      | true
42 == "42foo"    | true      | false
```

Quelle: wiki.php.net

- Ternäre Operatoren müssen geklammert werden!

FAZIT PHP

- Einfach Fehler zu machen
- Probleme bewusst machen...
- ... konstant testen
- ... defensiv Programmieren
- ... eventuell ein Framework benutzen
- ... oder eine vernünftige Sprache benutzen ;-)
- PHPSadness.com für mehr unintuitive Dinge
- PHPTheRightWay.com PHP best practices

INFORMATION LEAKS

How to find critical information

INFORMATION LEAK

- Bewusst
  - vom Angreifer herbeigeführt
- Unbewusst
  - Programming/Logical-Flaws

BEISPIELE

- Vorhersagbare URLs
  - /user/15/ → /user/1/
  - 01052017.pdf → 02052017.pdf
- Software Banner/Header
- Fehlermeldungen
- Directory Listing

UNBEW. INFOLEAKS FINDEN

- Googeln!
  - z.B. site:rub.de intitle:'Index of'
  - Google Hacking Database
- Soziale Netzwerke
  - Antworten auf Geheimfragen
- Ressourcen-URLs (Bilder, CSS, ...) anschauen
- Bekannte URLs ausprobieren
  - z.B. robots.txt
  - Automatisiert finden mit Gobuster (nicht im HackPra!)

ROBOTS.TXT

```
User-agent: *
Disallow: /admin/
Disallow: /stats/
Disallow: /internaljobs/
Disallow: /internaljobsbyorganization/
Disallow: /internaljobsearch/
```

Problem?

INTERESSANTE DATEIEN

- dump.sql
- Automatisch erstellte Backup-Dateien z.B.:
  - VIM: index.php~, index.php.swp
  - index.php.bak
  - ...
- .git/config → Jackpot
  - Mit git-dumper Git Repository wiederherstelln
- ...

SPOT THE PROBLEM!

```
Das Bild ist <a href="http://kunde20bf:123456@seite.com/bild.jpg">hier</a>
zu finden.
```

ALLGEMEINE VERTEIDIGUNG

- robots.txt nicht mit sensiblen Pfaden füllen

```
<meta name="robots" content="noindex, nofollow">
```

- Sensible Pfade per Authentifikation schützen
- Rate limit bei Votings, Logins, ...
- In "Production"
  - Directory Listing ausstellen
  - Fehlermeldungen ausstellen
    - Leise, mit Benachrichtigung
    - Nicht verwertbar für Angreifer

AUFGABE

- Logical Flaw finden
  - Kundendaten kaufen
- Information Leak finden
  - Document Root herausfinden
- Abgabe bis Dienstag den 16.11.2021 23:59

FRAGEN?

ANTWORTEN!

Auch per Mail an nds+badbank@rub.de

:)