Sicherheitsprobleme bei Webapps

Sichere Software zu programmieren ist doch ganz einfach, oder?

Ich bin...

- Patrick Cornelißen
- Dipl. Informatiker
- Selbständig
- Softwareentwickler
- Coach/Trainer

- Java und früher PHP
- Agile!
- Software Craftsmanship
- Security (Web)

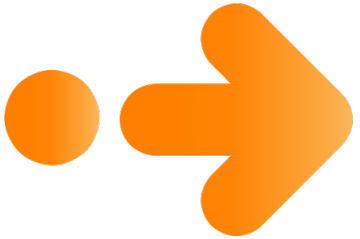
Mich kann man mieten! ;-)

Agenda

- Sicherheit ist doch kein aktuelles Thema mehr!?
- Beispiel für konkrete Angriffsmöglichkeiten bzw. Probleme

Sicherheit meint übrigens Security und nicht Safety!

Ein paar Highlights der letzten Monate...



App-Burbs!

Die Verantwortung liegt bei uns!

Sichere Software ist Sisyphusarbeit

http://switchboard.nrdc.org/blogs/lwillcox/assets_c/2010/06/Sisyphus_starwars-321.html

Es gibt aber Leitfäden...

OWASP

The Open Web Application Security Project

http://www.owasp.org

Passwort Authentifizierung

http://en.wikipedia.org/wiki/File:Double-alaskan-rainbow.jpg (Eric Rolph)

Hash? Salt?

Hash

- „Checksumme“
- „nicht“ rekonstruierbar
- viele Algorithmen
- mehrere Eingaben ergeben den selben Hash

Salt

- Nutzung Beispiel: salt+hash(salt+pwd)
- nicht geheim!
- aber zufällig je Hash
- erschwert Nutzung von Rainbow Tables

sha1(passwort) = 2e2b6533a81bc15430cf65de46dc097eeb5ba70c

Hashes sind sicher?

1 AMD Radeon HD7970 GPU berechnet bis zu 8.2 Milliarde Hashes pro Sek.

3.108 TB
Für alle möglichen 10 Zeichen Passwörter + MD5 Hash

167 GB
Braucht eine Rainbow Table für 99,9% der obigen Kombinationen

Wortlisten mit 500 Mio. oft benutzten Passwörtern

Just six days after the leak of 6.5 million LinkedIn password hashes in June, more than 90 percent of them were cracked.

http://security.stackexchange.com/questions/17798/how-can-crackers-reconstruct-200k-salted-password-hashes-so-fast
http://arstechnica.com/security/2012/08/passwords-under-assault/

Abhilfe!

Password-Based Key Derivation Function 2 (PBKDF2 bzw. PKCS#5)

```
key = hash(password)
for 1 to runden do
  key = hash(key)
```

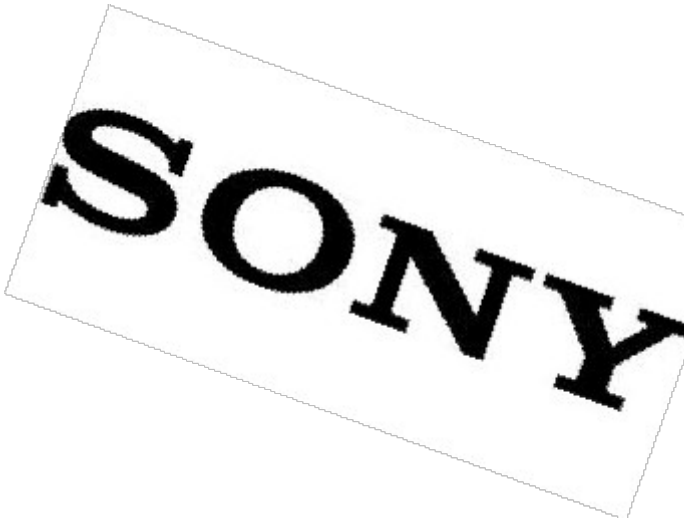
- Zeit zur Berechnung wächst linear mit der Anzahl der Runden
- Sicherheit nachträglich verbesserbar! (Einfach noch mehr Runden berechnen)

Bcrypt

- Nutzt Blowfish Verschlüsselung und eigenen Salt
- „teuer“ zu berechnen
  - leider auch teurer zu überprüfen
  - Durchsatz der Cracker signifikant geringer
- Nachträglich Sicherheit verbesserbar

Oder zB. OpenID nutzen und das Passwort Problem durch jemand anderen lösen lassen. ;-)

SQL Injection



Das Problem:

```
SELECT * FROM UserDB where
username='+$username+'
```

Sei: *$username* = `egal' OR '1'='1`

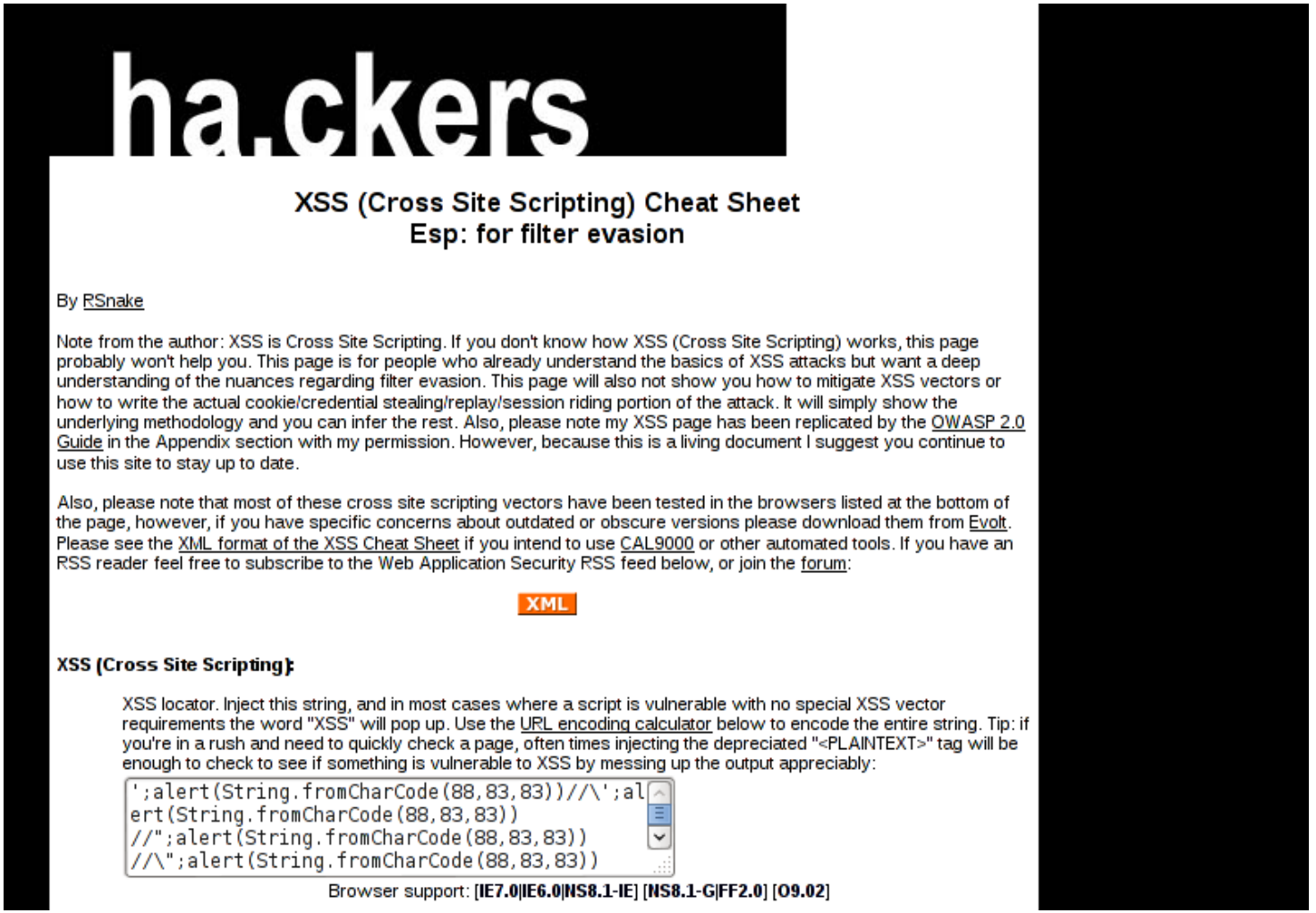
Ergebnis:

```
SELECT * FROM UserDB where
username='egal' OR '1'='1'
```

NoSQL DBs sind sicherer?

- Connection Pollution
  Z.B. „Zieltabelle“ oder „-DB“ verändern von außen
- JSON injection
  analog zu SQL, einbetten von Nutzerdaten verändert Struktur der Daten
- View Injection
  Z.B. Abgelegte Map&Reduce Jobs verändern um Daten zu faken
- Key bruteforcing
  Enumerieren von Items durch raten der Keys (besonders wenn DB erreichbar ist)

Cross Site Scripting (XSS)



ha.ckers

XSS (Cross Site Scripting) Cheat Sheet
Esp: for filter evasion

By RSnake

Note from the author: XSS is Cross Site Scripting. If you don't know how XSS (Cross Site Scripting) works, this page probably won't help you. This page is for people who already understand the basics of XSS attacks but want a deep understanding of the nuances regarding filter evasion. This page will also not show you how to mitigate XSS vectors or how to write the actual cookie/credential stealing/replay/session riding portion of the attack. It will simply show the underlying methodology and you can infer the rest. Also, please note my XSS page has been replicated by the OWASP 2.0 Guide in the Appendix section with my permission. However, because this is a living document I suggest you continue to use this site to stay up to date.

Also, please note that most of these cross site scripting vectors have been tested in the browsers listed at the bottom of the page, however, if you have specific concerns about outdated or obscure versions please download them from Evolt. Please see the XML format of the XSS Cheat Sheet if you intend to use CAL9000 or other automated tools. If you have an RSS reader feel free to subscribe to the Web Application Security RSS feed below, or join the forum:

XML

XSS (Cross Site Scripting):

XSS locator. Inject this string, and in most cases where a script is vulnerable with no special XSS vector requirements the word "XSS" will pop up. Use the URL encoding calculator below to encode the entire string. Tip: if you're in a rush and need to quickly check a page, often times injecting the depreciated "<PLAINTEXT>" tag will be enough to check to see if something is vulnerable to XSS by messing up the output appreciably:

```
';alert(String.fromCharCode(88,83,83))//\';al
ert(String.fromCharCode(88,83,83))
//";alert(String.fromCharCode(88,83,83))
//\";alert(String.fromCharCode(88,83,83))
```

Browser support: [IE7.0|IE6.0|NS8.1-IE] [NS8.1-G|FF2.0] [O9.02]

http://ha.ckers.org/xss.html
http://benhayak.blogspot.de/2012/06/google-mail-hacking-gmail-stored-xss.html

Ausweg?

BBCode

~~Blacklisting~~

Whitelisting

UTF-8 :-(

Jsoup

HTML Purifier

~~WYSIWYG~~ :-(

AntiSamy

Cross Site Request Forgery (XSRF)

GET: http://www.google.com/logmeout

Ist sicher, das geht ja nur mit meiner Session...!?

Cross Site Request Forgery (XSRF)

UUUPS!?

Cross Site Request Forgery (XSRF)

<img src="www.server.local/editUser?
newPwd=p0wned&email=h4x0r@hotmail.com"/>

UUUPS!? Nehmen wir POST!

Cross Site Request Forgery (XSRF)

Leider hilft POST nicht!
(Javascript) :-(

Des Rätsels Lösung: Token

- anhängen bei der Generierung der Seite
- verifizierbar z.B. über Session
- läuft ab
- identifiziert Aktion
- wird ungültig nach Nutzung (wenn möglich)

- Angreifer muss Token kennen
- Replay Angriffe nicht möglich
- Browser History wertlos
- Verhindert auch, daß schnelle „Klicker“ Aktionen 2x auslösen

http://www.google.com/letmeout?token=hjdsgfdhfs

Randbedingungen

- Token sollten nicht die Session verstopfen
- Hashfunktion fast egal
- Tokens sollten nicht zu kurz leben (Frust beim User)
- Brute Forcing erkennen (Intrusion detection)
- Fälschen von Requests durch signieren der wichtigen Daten möglich (Token ist die Signatur)

  http://www.google.com/letmeout?token=hjdsgfdhfs

Wichtige Prinzipien

- **Apply defense in depth (complete mediation)**
- Use a positive security model (fail-safe defaults, minimize attack surface)
- Fail securely
- Run with least privilege
- Avoid security by obscurity (open design)
- Keep security simple (verifiable, economy of mechanism)
- Detect intrusions (compromise recording)
- Don’t trust infrastructure
- Don’t trust services
- Establish secure defaults (psychological acceptability)
- Don't trust user input

https://www.owasp.org/index.php/Category:Principle

Wichtige Prinzipien

- Apply defense in depth (complete mediation)
- **Use a positive security model** (fail-safe defaults, minimize attack surface)
- Fail securely
- Run with least privilege
- Avoid security by obscurity (open design)
- Keep security simple (verifiable, economy of mechanism)
- Detect intrusions (compromise recording)
- Don’t trust infrastructure
- Don’t trust services
- Establish secure defaults (psychological acceptability)
- Don't trust user input

Wichtige Prinzipien

- Apply defense in depth (complete mediation)
- Use a positive security model (fail-safe defaults, minimize attack surface)
- **Fail securely**
- Run with least privilege
- Avoid security by obscurity (open design)
- Keep security simple (verifiable, economy of mechanism)
- Detect intrusions (compromise recording)
- Don’t trust infrastructure
- Don’t trust services
- Establish secure defaults (psychological acceptability)
- Don't trust user input

Keine gute Idee...

```
isAdmin = true;
try {
  codeWhichMayFail();
  isAdmin = isUserInRole( “Administrator” );
}
catch (Exception ex)
{
  log.write(ex.toString());
}
```

Wichtige Prinzipien

- Apply defense in depth (complete mediation)
- Use a positive security model (fail-safe defaults, minimize attack surface)
- Fail securely
- **Run with least privilege**
- Avoid security by obscurity (open design)
- Keep security simple (verifiable, economy of mechanism)
- Detect intrusions (compromise recording)
- Don’t trust infrastructure
- Don’t trust services
- Establish secure defaults (psychological acceptability)
- Don't trust user input

Wichtige Prinzipien

- Apply defense in depth (complete mediation)
- Use a positive security model (fail-safe defaults, minimize attack surface)
- Fail securely
- Run with least privilege
- **Avoid security by obscurity (open design)**
- Keep security simple (verifiable, economy of mechanism)
- Detect intrusions (compromise recording)
- Don’t trust infrastructure
- Don’t trust services
- Establish secure defaults (psychological acceptability)
- Don't trust user input

Wichtige Prinzipien

- Apply defense in depth (complete mediation)
- Use a positive security model (fail-safe defaults, minimize attack surface)
- Fail securely
- Run with least privilege
- Avoid security by obscurity (open design)
- **Keep security simple (verifiable, economy of mechanism)**
- Detect intrusions (compromise recording)
- Don’t trust infrastructure
- Don’t trust services
- Establish secure defaults (psychological acceptability)
- Don't trust user input

Wichtige Prinzipien

- Apply defense in depth (complete mediation)
- Use a positive security model (fail-safe defaults, minimize attack surface)
- Fail securely
- Run with least privilege
- Avoid security by obscurity (open design)
- Keep security simple (verifiable, economy of mechanism)
- **Detect intrusions (compromise recording)**
- Don’t trust infrastructure
- Don’t trust services
- Establish secure defaults (psychological acceptability)
- Don't trust user input

Wichtige Prinzipien

- Apply defense in depth (complete mediation)
- Use a positive security model (fail-safe defaults, minimize attack surface)
- Fail securely
- Run with least privilege
- Avoid security by obscurity (open design)
- Keep security simple (verifiable, economy of mechanism)
- Detect intrusions (compromise recording)
- **Don’t trust infrastructure**
- Don’t trust services
- Establish secure defaults (psychological acceptability)
- Don't trust user input

Wichtige Prinzipien

- Apply defense in depth (complete mediation)
- Use a positive security model (fail-safe defaults, minimize attack surface)
- Fail securely
- Run with least privilege
- Avoid security by obscurity (open design)
- Keep security simple (verifiable, economy of mechanism)
- Detect intrusions (compromise recording)
- Don’t trust infrastructure
- **Don’t trust services**
- Establish secure defaults (psychological acceptability)
- Don't trust user input

Wichtige Prinzipien

- Apply defense in depth (complete mediation)
- Use a positive security model (fail-safe defaults, minimize attack surface)
- Fail securely
- Run with least privilege
- Avoid security by obscurity (open design)
- Keep security simple (verifiable, economy of mechanism)
- Detect intrusions (compromise recording)
- Don’t trust infrastructure
- Don’t trust services
- **Establish secure defaults (psychological acceptability)**
- Don't trust user input

Wichtige Prinzipien

- Apply defense in depth (complete mediation)
- Use a positive security model (fail-safe defaults, minimize attack surface)
- Fail securely
- Run with least privilege
- Avoid security by obscurity (open design)
- Keep security simple (verifiable, economy of mechanism)
- Detect intrusions (compromise recording)
- Don’t trust infrastructure
- Don’t trust services
- Establish secure defaults (psychological acceptability)
- **Don't trust user input!**

Pearls of wisdom

- Benutzerdaten sind böse! (besonders wenn es HTML ist)
- Angemessene Sicherheit ist schwer aber nicht unmöglich
- OWASP und security.stackexchange.com sind gute Anlaufstellen
- Open Source Projekte und Firmen sollten eine dedizierte Emailadresse für sicherheitsrelevante Vorfälle haben und publizieren

Danke!

Twitter: @cornelis / @orchit_ek

Email: pcornelissen@orchit.de

Google+: orchit (G+ Seite)
cornelis@pcornelissen.de
pcornelissen@orchit.de

Web: http://www.orchit.de