

UNIX-System-Sicherheit

Harald Wilhelmi

30. April 2008,
München

Systeme

- UNIX-Derivate
 - Linux
 - BSD
 - Solaris
 - HP-UX
 - ...

- Windows
- MAC-OS
- Embedded Systems
- ...

Arten von Sicherheit

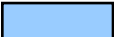
- Vertraulichkeit
- Daten-Integrität
- Zugriffs-Kontrolle

- Verfügbarkeit
- Konsistenz
- Nachvollziehbarkeit

Maßnahmen

- System-Härtung

- Bauliche Sicherung
- Benutzer-Schulung
- Personal-Auswahl
- Kontakt-Pflege
- ...

 Thema dieses Vortrags

- **Angriffsfläche minimieren**
- Grundsicherung durch Rechte und Logging
- Suche nach Sicherheitslöchern
- Techniken und Tools

Allgemeine Betrachtung

Wie macht man das?

Unnötige Programme und APIs werden still gelegt.

Was nutzt das?

- Weniger Angriffspunkte
- Weniger Sicherheitspatches
- Weniger Log-Meldungen

Das System wird einfacher!

Kann man die Angriffsfläche auch anders reduzieren?

Ja, 'Personal Firewalls' helfen auch.

Aber diese erhöhen die Komplexität des Systems.

Prozesse und offene Ports

Was läuft denn so?

```
$ ps -efa | more          # BSD: ps aux | more
```

Welche Ports sind offen?

```
$ netstat -an | more
```

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	0	0.0.0.0:22	0.0.0.0:*	LISTEN
udp	0	0	127.0.0.1:53	0.0.0.0:*	
tcp	0	0	192.168.1.108:39744	217.110.29.210:500	ESTABLISHED

Offene Ports

Lokal gebunden – nicht extern erreichbar

Offene TCP-Verbindung

Und welcher Port gehört zu welchem Prozess?

```
$ lsof -i :22          # Wem gehört Port 22?  
$ lsof -p 123         # Welche Dateien/Ports hat der Prozess mit PID 123 offen?
```

lsof ist kein Standard-Tool. Die meisten Linux- und BSD-Distributionen haben es.

Offizielle Quelle: [ftp://lsof.itap.purdue.edu/pub/tools/unix/lsof](http://lsof.itap.purdue.edu/pub/tools/unix/lsof)

Andere Schnittstellen

Netzwerk: Client-Seite!

- Manchmal ist 'netstat' auch hier hilfreich – aber nicht immer (z.B. UDP).
- Namensauflösung ist immer wieder von Interesse (NIS ist immer BÖSE!):

```
/etc/nsswitch.conf
```

Lokale Schnittstellen: Inter-Prozess-Kommunikation

Der Zugriff wird über User bzw. Gruppen geregelt. Die verantwortliche Applikation muss sinnvolle Rechte setzen. Dies ist oft eine Frage der Konfiguration.

- 'shared memory' und Freunde:

```
ipcs -a      # Systemabhängige Zusatz-Optionen um User zu sehen!
```
- UNIX-Domain-Sockets:

```
$ netstat -an | grep unix  
unix 2      [ ACC ]  STREAM  LISTENING  9332 /tmp/.X11-unix/X0
```
- 'named pipes':

```
$ find / -type p
```

Dienste still legen

Startup-Skripte:

System V/Linux:

`/etc/rc?.d/*`

BSD:

`/etc/rc.local`

(Änderungen werden beim nächsten Run-Level-Wechsel bzw. Reboot wirksam.)

Inetd:

`/etc/inetd.conf` (oder: `/etc/xinetd.d`)

(Änderungen werden nach ' `kill -1 <PID inetd>`' wirksam.)

Init:

`/etc/inittab`

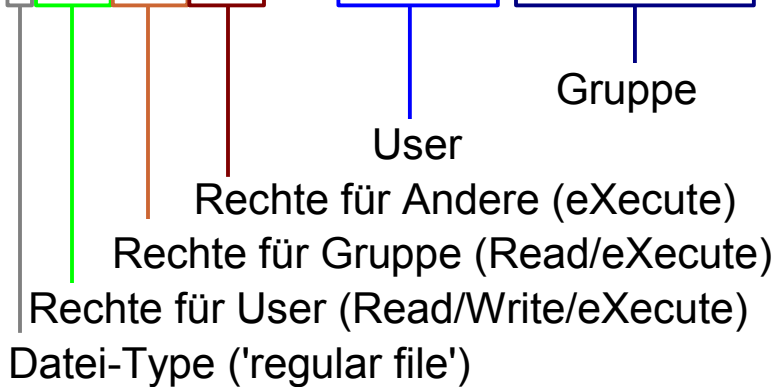
(Änderungen werden nach ' `init q`' wirksam.)

- Angriffsfläche minimieren
- **Grundsicherung durch Rechte und Logging**
- Suche nach Sicherheitslöchern
- Techniken und Tools

UNIX-Rechte für Anfänger

```
$ ls -l bla
```

```
-rwxr-x--x 1 harald developer 8 Jan  5 16:25 bla
```



Rechte:

r = read, lesen, oktaler Code 4

w = write, schreiben, für Verzeichnis 'Dateien anlegen/löschen', oktaler Code 2

x = execute, für Datei 'ausführen', für Verzeichnis 'wechseln', oktaler Code 1

Ändern:

```
$ chmod 750 bla # User rwx (4+2+1=7), Gruppe r-x, Andere ---
```

```
$ chmod g+w bla # Gruppe erhält Schreibrechte
```

```
$ chmod u-x bla # User verliert Ausführungsrechte
```

```
$ chmod o-x bla # Andere (other) verlieren Ausführungsrechte
```

UNIX-Rechte für Experten

```
$ chmod 1755 bla # oder: chmod o+t bla
```

```
$ ls -l bla
```

```
-rwxr-xr-t 1 harald developer 8 Jan 5 16:25 bla
```

Sticky-Bits setzen:

- Ausführbarer Code: Code wird nach Terminierung im Speicher gehalten.
- Verzeichnis: Nur Eigentümer darf Dateien löschen.

```
# chmod 4755 bla # oder: chmod u+s bla
```

```
# ls -l bla
```

```
-rwsr-xr-x 1 harald developer 8 Jan 5 16:25 bla
```

Set-UID-Bit setzen:

- Ausführbarer Code: *Effective User-ID* wird beim Start auf Eigentümer der Datei gesetzt.

```
# chmod 2755 bla # oder: chmod g+s bla
```

```
# ls -l bla
```

```
-rwxr-sr-x 1 harald developer 8 Jan 5 20:25 bla
```

Set-GID-Bit setzen:

- Ausführbaren Code: *Effective Group-ID* wird beim Start auf die Gruppe der Datei gesetzt.
- Verzeichnis: Neue Dateien erben die Gruppe des Verzeichnisses.

Goldene Regeln – UNIX-Zugriffsrechte

- Benutze Sie!

```
# cp /bin/sh /bin/mkmeroot  
# chmod u+s /bin/mkmeroot # Backdoor für Arme
```

- SID/GID-Bits nur auf abgesicherten, vertrauenswürdigen Programmen.
Achtung: User-mountable Filesysteme immer mit `nosetuid`-Option benutzen (automount, Linux-User-Mount-Option).

```
$ ln -s /darf/nicht/kaputt/gehn \  
/tmp/hier_schreibt_root
```

- Verzeichnisse in denen Dateien mit vorhersagbaren Namen erzeugt werden, müssen geschützt sein.

```
$ cp evil_hack /x/y/z/wichtig  
Permission denied.  
$ rm -rf /x/y; mkdir -p /x/y/z  
$ cp evil_hack /x/y/z/wichtig
```

- Wenn eine Datei geschützt werden muss, dann auch alle Verzeichnisse auf dem Pfad der Datei.

```
$ cat /writable/passwds  
Permission denied.  
$ rm /writable/passwds  
$ cp my_passwds  
/writable/passwds
```

- World-writable Verzeichnisse müssen das Sticky-Bit haben.

Syslog

- Zentralisierte Log-Dateien für Betriebssystem und Applikationen
- Erlaubt Logging über Netzwerk auf Log-Host
- Gut standardisierte API
- Konfiguration in `/etc/syslog.conf` (`man syslog.conf`)
- Test/Benutzung aus der Shell über `/usr/bin/logger` (`man logger`)

Und: Rechte-Trennung zwischen Logs und Applikation!

Format der Konfiguration:

```
<facility>.<severity><Tab><Log-File>  
<facility>.<severity><Tab>@<Log-Host>
```

Beispiel:

```
mail.*           /var/log/mail  
local1.debug    /var/log/app11.debug  
*.err           @loghost.tngtech.com
```

Einzige Falle: Nach `<severity>` muss ein oder mehrere `<Tab>`-Zeichen kommen.
`<Space>` ist falsch!

Logging Best Practices

- Unwichtige Meldungen wegfiltern.
 - Wieso? Langweile und Information-Overload vermeiden.
 - Wie? Z.B. mit LogCheck/LogSentry.
- Übrige Meldungen per Email an System-Administrator verschicken.
 - Wieso? Logs inspizieren wird sonst vergessen.
 - Wie? Ebenfalls per LogCheck/LogSentry.
- Log-Rotator benutzen.
 - Wieso?
 - Vermeidet 'filesystem full' und nimmt dem Administrator Arbeit ab.
 - Sabotage durch Überlastung ist schwieriger.
 - Email/Web-Logs: Weniger Datenschutz-Sorgen.
 - Wie? Z.B mit LogRotate.
- Log-Dateien auf eigenes Datei-System oder zentralen Log-Server
 - Wieso? Wenn es doch mal ein 'filesystem full' gibt, sollen die Applikationen weiter laufen.
 - Wie? Syslog sauber aufsetzen.
- Bei Connection-Logging IP-Adressen loggen – keine Domain-Namen.
 - Wieso?
 - Keine Performance-Probleme mit langsamen/fehlerhaften DNS-Servern.
 - Manipulation ist schwerer.
 - Wie? Hängt von der Applikation ab.

- Angriffsfläche minimieren
- Grundsicherung durch Rechte und Logging
- **Suche nach Sicherheitslöchern**
- Techniken und Tools

Demo

Klassische Lücken: SQL-Injection

Daten aus suspekter Quelle werden ohne Prüfung bzw. Filterung benutzt um ausführbaren SQL-Code zu bilden. Beispiel:

```
my $name=param('name');      # Parameter aus HTML-Formular
my $sth=$dbh->do(
    "insert into appl_user ( name, admin_level ) values ( '$name', 0)"
);
```

Der Programmierer erwartet, dass in jedem Fall ein nicht-privilegiertes Nutzer eingerichtet wird, weil `admin_level=0`. Der Angreifer gibt ein:

```
Name: x', 100) on duplicate key update name=concat('x
```

Ausgeführter SQL-Code:

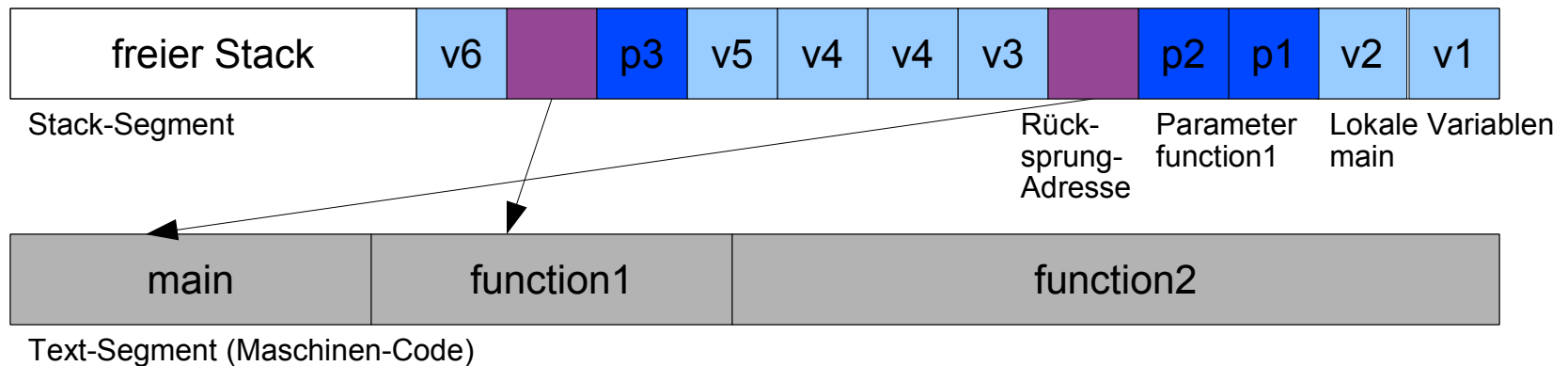
```
insert into appl_user ( name, admin_level ) values ( 'x', 100 )
on duplicate key update name=concat('x', 0)
```

Geht nicht nur mit SQL – manchmal findet man auch eine Shell-Injection.

Klassische Lücken: Buffer-Overflow (Stack)

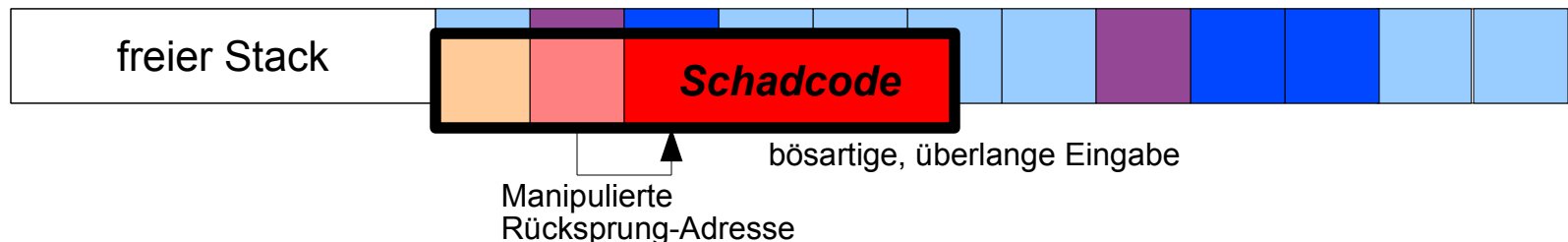
Was ist der Stack?

Ein Speicher-Segment in dem Daten abgelegt werden, welche den Funktionsaufrufen zugeordnet sind – insbesondere: Parameter, lokale Variablen und *Rücksprungadressen*, z.B:



Was ist ein Buffer-Overflow?

Wenn z.B. in v6 die Eingabe eines nicht vertrauenswürdigen Nutzers gespeichert wird *und* die Länge der Eingabe nicht geprüft wird, kann ein Angreifer durch eine überlange Eingabe andere Speicherbereiche überschreiben. Das kann dann so aussehen:



Sicherheitslöcher finden

Finde Eingabedaten aus dubiosen Quellen – z.B.:

- Formular-Felder in Web-Applikationen (einschließlich `<input type="hidden">`)
- URL-Parameter (z.B.: `http://some.domain/path?param=blub`)
- Eingabe-Dateien und Netzwerk-Datenströme
- Bei Set-UI-Programmen auch Kommando-Zeilen-Parameter und Benutzereingaben
- ...

Testen:

- Überlange Eingaben
- Eingaben mit Sonderzeichen: `'`;#
- Binärdaten



Gut:

- Eindeutige Fehlermeldung
- Eingabe wird zurückgewiesen
- Bedingt akzeptabel: Funktioniert auf bössartiger Eingabe korrekt

Böse:

- Segmentation Fault. Core dumped.
- Lauft auf korrupten Daten weiter
- SQL-Fehlermeldungen

- Angriffsfläche minimieren
- Grundsicherung durch Rechte und Logging
- Suche nach Sicherheitslöchern
- **Techniken und Tools**

chroot und Freunde

Eine *Sandbox* ist eine abgeschirmte Umgebung, die den Zugriff von Programmen auf benötigte bzw. als unbedenklich eingestufte Ressourcen beschränkt. Dieses Konzept wird benutzt um besonders exponierte oder unsichere Teile eines Systems zu isolieren.

Wann benutzt man Sandboxing?

- Als Workaround bei Software mit bekannten Sicherheitsproblemen
- Bei Hostern zur Trennung unterschiedlicher Kunden auf einer Maschine
- Zum Testen von Software aus zweifelhaften Quellen

Wie kann man das auf UNIX mit Hausmitteln machen?

- Applikation in einem Verzeichnisbaum isolieren.
- Mit `chroot (1M)` den Zugriff auf diesen Verzeichnisbaum beschränken.
- Applikation mit Rechten eines nicht-privilegierten Users ausführen, z.B. mit `su`.

Tools: Z.B. `makejail` und `mock`.

Welche Sandboxing-Lösungen gibt es noch?

- BSD jails/sysjails: Betriebssystem-Compartment.
- BSD systrace: Firewall für System-Calls.
- Verschiedene UNIXe: Virtuelle Maschinen wie VMware und Xen.

Rate die Passwörter Deiner Nutzer!

... bevor es der Hacker tut.

Das hilft ...

- ... bei der Nutzer-Schulung
- ... gegen automatisierte ssh-Angriffe mit Joe-Passwörtern (User: joe, Passwort: joe)
- ... manuelle Angriffe bzw. automatisierte Angriffe mit begrenzter Bandbreite

Das hilft nicht ...

- ... bei massiven Brute-Force-Angriffen auf öffentliche Passwort-Dateien (NIS ist böse!)
- ... bei ausgeplauderten/aufgeschrieben/mitgefilmten Passwörtern
- ... gegen Key-Logger (Internet-Café!)

Wie macht man das?

- crack – ein klassisches Open-Source-Tool
<ftp://ftp.cerias.purdue.edu/pub/tools/unix/pwdutils/crack>
- John the Ripper – ein (etwas weniger) klassisches Open-Source-Tool – teils kommerziell
<http://www.openwall.com/john/>

Erkennung von Manipulationen

Erfolgreiche Angriffe verändern häufig das System:

- Einige Angriffe beschädigen das System: Überschreiben von Konfigurationsdateien!
- Der Hacker will weniger Arbeit haben, wenn er wieder kommt: Backdoors!
- Neue Aufgaben unter neuem Management: Neue Applikationen!

Die Antwort: Systematische Überprüfung des Systems

- Zugriffs-Rechte und Eigentümer
- Read-only: Länge, Check-Summe – im Extremfall Vergleich mit Sicherungskopie.
- Log-Dateien: Länge – dürfen nur wachsen oder auf 0 schrumpfen.

Tools:

- Tripwire – Open-Source oder kommerziell
<http://www.tripwire.com/products/enterprise/ost/>
- Integrit – Open-Source
<http://integrit.sourceforge.net/>

Aber Achtung:

Moderne Root-Kits sind für diese Maßnahmen unsichtbar (Kernel-Manipulationen, Virtualisierung). Das Wettrüsten zwischen Root-Kit-Bauern und Root-Kit-Erkennern ist noch in vollem Gange. Ausgang ungewiss...

Das war's.

Danke!

Sandkasten-Bauanleitung

- Benötigte Dateien finden und Verzeichnisbaum ab / in Sandbox-Verzeichnis nachbauen:
 - Device-Dateien: Meist weiß man welche. Krypto-Applikationen brauchen oft `/dev/urandom`. Solaris-Shared-Libraries brauchen immer `/dev/zero`.
 - Libraries: Das `ldd`-Kommando gibt uns einen ersten Tipp. Für die Namensauflösung braucht man oft Dateien, z. B aus `/usr/lib/nss*`.
 - Binaries: Häufig das `su`-Kommando und eine Shell.
 - Konfigurationsdateien: Für die Namensauflösung braucht man die `/etc/nsswitch.conf` und gegebenenfalls weitere Dateien aus `/etc` (`hosts`, `passwd`, `services`, `protocols`). Meist ist es weniger als man denkt.
 - Die Applikations-Binaries und Daten.

- Rechte geeignet setzen:

```
cd /sandbox
chown -R root . ; chgrp -R $applgroup .
chmod -R o-rwx . ; chmod -R g-w .
chmod 550 $binaries; chmod g+w $dynamic_data
```

- Applikation starten:

```
chroot /sandbox su - $appluser -c /usr/bin/appl
```

Hinweise & Warnungen

Troubleshooting

- Wenn die Fehlermeldungen der Applikation nicht ausreichen um fehlende Dateien zu finden, hilft ein System-Call-Tracer (`strace`, `truss`, `tusc`, `kdump/ktrace`, ...).
- Manche Applikationen benötigen während des Starts root-Rechte, z.B. Web-Server zum Öffnen von Port 80. In diesem Fall muss die Aufgabe der root-Privilegien durch die Applikation erfolgen. Das `su`-Kommando aus der Bauanleitung käme zu früh.

Best Practices

- Integriere das Setup der chroot-Umgebung in das Applikations-Startup-Skript. Da geht es nicht verloren und Applikations-Patches wandern spätestens beim nächsten Reboot in das Sandbox-Verzeichnis.
- Mache ein Backup des Startup-Skriptes. Manche Updates überschreiben auch das Startup-Skript.
- Einige Programme (z.B. `bind9`) haben chroot schon eingebaut, wenn man sie mit den richtigen Optionen startet. Das spart etwas Arbeit. Dokumentation lesen!

Vorsicht!

- Die meisten Devices gehören **nicht** in chroot-Umgebungen: `/dev/kmem`, `/dev/mem` und Platten-Devices erlauben direkte Umgehung der chroot-Beschränkung.
- Das Aufgeben der Root-Rechte ist wesentlich. Sonst kann ein Hacker, der die Applikation übernommen hat, ausbrechen, z.B. indem er seine eigene Device-Files anlegt.