

Sicherheitsrelevante Programmierfehler (Teil 5)

[Thomas Biege](#)

Inhaltsverzeichnis

- [Programmumgebung \(nicht auf C beschränkt!\)](#)
- [User Input](#)
- [offene Ressourcen](#)
- [0,1,2 - Die Standard Filedeskriptoren \(nicht auf C beschränkt!\)](#)
- [Zugriffsrechte \(nicht auf C beschränkt!\)](#)

Programmumgebung (nicht auf C beschränkt!)

Jedes Programm läuft in einer Programmumgebung, die das Verhalten beeinflussen kann. Die Umgebung (Environment) besteht aus einem Feld aus Variablen und deren Wert, z.B. TIMEZONE=GMT oder USER=thomas.

Beispiel:

```
environ[0] = "USER=thomas";
environ[1] = "HOME=/home/thomas";
environ[2] = "MAIL=/var/spool/mail/thomas"
.
.
.
```

Grundsätzlich sollte ein Programm so wenig wie möglich auf seine Umgebung vertrauen, da sie von einem lokalen Benutzer nach Belieben verändert werden kann.

Privilegierte Prozesse, die andere Programme aufrufen, sollten immer ihr Environment verwerfen und ein neues sicheres aufbauen, das von dem aufgerufenen Programm geerbt wird.

Es folgt ein (wahrscheinlich unvollständiger) Blick auf einige Umgebungsvariablen, die problematisch werden können. Grundsätzlich sollte das Environment so wenig wie möglich als Informationsquelle genutzt werden. Die Variablen oder deren Namen sind von Unix-Derivat zu Unix-Derivat unterschiedlich, und andere machen nicht auf allen oder nur alten Systemen Schwierigkeiten.

PATH

Die PATH Variable enthält eine Liste von Verzeichnisnamen, die bei der Angabe von Programmnamen nach dem richtigen Pfad durchsucht wird. Bibliotheksfunktionen, die die Pfadvariable benutzen, sind: *execlp(3)*, *execvp(3)*, *popen(3)* und *system(3)*. Grundsätzlich sollten immer die kompletten Pfade in Programmen benutzt werden, da ein böswilliger Benutzer lediglich die Pfadliste ändern muß, um sein eigenes Programm mit demselben Namen ungewollt zur Ausführung zu bringen. Man sollte grundsätzlich von der Verwendung von *popen(3)* und *system(3)* absehen, da aufgrund der nicht absehbaren Interaktion mit den komplexen Kommandoshells ein zu großes Sicherheitsrisiko entsteht!

IFS

IFS enthält eine Menge von Buchstaben, die als Separator für Shellargumente benutzt wird. Im Normalfall sind es alle Whitespacezeichen. Setzt ein Angreifer `IFS=0` und das Programm führt den Befehl `"/bin/show"` aus, dann wird 'o' als Separator betrachtet und `"/bin/sh"` mit dem Parameter "w" aufgerufen. Das bedeutet, selbst wenn man eine Shell, bzw. *popen(3)* oder *system(3)* (*popen(3)* und *system(3)* benutzen ihrerseits eine Shell, um ihre Aufgabe zu erledigen) mit vollen Pfadnamen aufruft, ist man immer noch über IFS angreifbar. Zum Glück ignorieren die modernen Kommandoshells den Wert von IFS.

LD_PRELOAD / LD_LIBRARY_PATH

Über `LD_PRELOAD` (der Name kann von System zu System variieren) kann explizit der Pfad zu einer *Shared Library* (oder DLL's) angegeben werden. Lenkt nun ein Angreifer den Pfad zur Standard C-Bibliothek auf seine eigene Bibliothek um, so kann er das Verhalten von Funktionen nach seinem Willen steuern. Wird beispielsweise die *crypt(3)* Funktion ersetzt, so daß bei der Paßwortgenerierung immer der Ciphertext für den Benutzer root zurückgegeben wird oder alle Stringvergleichsfunktionen 0 als Rückgabewert haben, kann der Angreifer über das privilegierte Loginprogramm `"/bin/login"` eine Kommandoshell mit erhöhten Rechten erlangen. Glücklicherweise sorgen neuere Implementierungen der *libc* dafür, daß Prozesse mit erhöhten Rechten diese u.ä. Variablen ignorieren. Man sollte jedoch nicht vergessen, daß andere Prozesse, die von dem privilegierten Prozeß erzeugt wurden, selber aber nicht mit dem `SetUID/-GID-Flag` versehen sind, `LD_PRELOAD` wieder benutzen. Dieser Umstand kann zu Sicherheitsproblemen führen, wenn die erhöhten Rechte nicht vor dem Erzeugen verworfen wurden.

Umgebungsvariablen können auch über den Aufruf von *unset(3)* gelöscht werden, dieses Verfahren ist aber nicht sicher, da Umgebungsvariablen mehrmals in der Programmumgebung existieren können, was dazu führt, daß die von dem Angreifer gesetzte Version eventuell nicht gelöscht wird.

Aus diesen Gründen sollte das Environment vor dem Erstellen eines neuen Prozesses immer von Grund auf neu erstellt werden.

User Input (nicht auf C beschränkt!)

Immer da, wo Benutzereingaben gelesen werden, sollte man die Menge der erlaubten Zeichen begrenzen. In der Regel brauchen Menschen nur a-z, A-Z, 0-9 und Satzzeichen einzugeben. Somit verringern wir die Angriffsfläche, da Binärdaten, Formatzeichen oder *Escape-Sequenzen* nicht mehr eingegeben werden können.

Offene Ressourcen

Bevor ein privilegiertes Programm einen benutzerdefinierten Prozeß erzeugt, sollten alle Filedeskriptoren, also auch Verzeichnisse, IPC-Handles und Sockets, geschlossen werden. Um ganz sicher zu gehen, empfiehlt es sich, das **Close-on-Exec Flag** für die sicherheitsrelevanten Filedeskriptoren direkt nach dem Öffnen zu setzen (natürlich mit Vorbehalt), siehe auch Chroot.

0,1,2 - Die Standard Filedeskriptoren (nicht auf C beschränkt!)

Die ersten drei Filedeskriptoren eines Programmes sind der *Standardeingabe*, *-ausgabe* und *-fehlerausgabe* zugeordnet. Werden diese Filedeskriptoren aber von dem *Parent-Prozeß* geschlossen, und der *Child-Prozeß* öffnet nun einen *Raw-Socket*, dann wird dem ersten freien Filedeskriptor der Socket zugeordnet. Ist es nun der Deskriptor für *Standardausgabe* oder *-fehlerausgabe*, dann werden bei jeder Ausgabe des Programms Daten über das Netzwerk geschickt. Kann der Angreifer die Ausgabedaten kontrollieren, dann ist er faktisch selber im Besitz dieses Sockets.

Man sollte also die ersten drei Filedeskriptoren immer überprüfen und falls notwendig selber belegen:

Beispiel:

```
[...]
int fd = 0;

while(fd < 2)
{
    if( (fd = open("/dev/null", 0600)) < 0)
        return(-1);
}
[...]
```

Angriffe dieser Art müssen auf *OpenBSD*-Systemen mit neuerem Kernel und *Linux*-Systemen und neuerer glibc nicht mehr befürchtet werden.

Zugriffsrechte (nicht auf C beschränkt!)

So offen, wie nötig und so restriktiv wie möglich. Das sollte die Grundregel sein. Einige Fehler oder Versäumnisse werden oft im Zusammenhang mit Zugriffsrechten gemacht:

falsche Angaben

Bei *open(2)* o.ä. Funktionen werden für die Angabe des **mode-Arguments** nicht die *S_**-Makros aus *sys/stat.h* genommen bzw. der oktale Wert nicht vollständig angegeben. Wenn als Wert von mode 600 (dezimal) benutzt wird anstatt 0600 (oktal), dann wird die Datei nicht wie gewünscht, also nur les- und schreibbar für den Eigentümer, angelegt.


umask(2)

Mit *umask(2)* läßt sich eine Maske für das Kreieren von Dateien setzen. Das bedeutet, daß das Mode-Argument mit der Maske wie folgt verknüpft wird: *mode & (~mask)*

Somit bewirkt eine Maske mit dem Wert 0027, daß jeder im Filesystem erzeugte Eintrag, also nicht nur Dateien, sondern auch *FIFO's* und Gerädateien, nicht für jederman beschreibbar, lesbar und ausführbar sind, und die Gruppe die Datei nur lesen und ausführen (was imgrunde dasselbe ist, da der Betriebssystemkernel eine Datei lesen muß, damit er sie ausführen kann) aber nicht beschreiben kann. Eine umask von 0027 ist ein guter Ausgangspunkt.

IPC

Oft vergessen Programmierer, die Zugriffsrechte für *FIFO's*, *Shared Memory* Bereiche, Unix Domain Sockets etc. zu setzen, so daß sie für jedermann lesbar oder sogar beschreibbar sind. Ein

böswilliger Benutzer kann so unberechtigt Informationen lesen oder den Prozeß, der von der *IPC*-Struktur liest, stören oder in seinem Sinne manipulieren. 

LinuxKP.org 06.06.2001