Capabilities

Das vergessene Linux-Security-Feature

Michael F. Schönitzer
michael@schoenitzer.de

Augsburger Linux-Infotag
22. April 2017

Der Root User

Traditionelles Rechtesystem von UNIX:

- User root mit $UID = 0$ darf alles.

Der Root User

Traditionelles Rechtesystem von UNIX:

- User root mit $UID = 0$ darf alles.
- Alle anderen User dürfen „nichts“.

Der Root User

Traditionelles Rechtesystem von UNIX:

- User root mit $UID = 0$ darf alles.
- Alle anderen User dürfen „nichts“.

Beispiel: NTP-Daemon

Der Root User

Traditionelles Rechtesystem von UNIX:

- User root mit $UID = 0$ darf alles.
- Alle anderen User dürfen „nichts“.

Beispiel: NTP-Daemon

- Muss Zeit setzen & in Echtzeit arbeiten.

Der Root User

Traditionelles Rechtesystem von UNIX:

- User root mit $UID = 0$ darf alles.
- Alle anderen User dürfen „nichts“.

Beispiel: NTP-Daemon

- Muss Zeit setzen & in Echtzeit arbeiten.
  $\Rightarrow$ Muss mit Root-Rechten laufen

Der Root User

Traditionelles Rechtesystem von UNIX:

- User root mit $UID = 0$ darf alles.
- Alle anderen User dürfen „nichts“.

Beispiel: NTP-Daemon

- Muss Zeit setzen & in Echtzeit arbeiten.
  - $\Rightarrow$ Muss mit Root-Rechten laufen
    - $\Rightarrow$ Darf auch Systemdateien ändern, Festplatten formatieren, Kernelmodule laden, Firewalls verstellen, Prozesse killen...

Capabilities

Idee

Trenne die Rechte feingranularer auf Capabilities auf

Capabilities

Idee

Trenne die Rechte feingranularer auf Capabilities auf

Beispiel: NTP-Daemon

- CAP_SYS_NICE
  Erlaubt die Prozess Priorität zu erhöhen, etc.
- CAP_SYS_TIME
  Erlaubt das Verstellen der Uhrzeit

Geschichte

- 1997: POSIX 1003.1e – withdrawn

Geschichte

- 1997: POSIX 1003.1e – withdrawn
- Linux 2.1

Geschichte

- 1997: POSIX 1003.1e – withdrawn
- Linux 2.1
- Linux 2.2.11: Capability bounding set

Geschichte

- 1997: POSIX 1003.1e – withdrawn
- Linux 2.1
- Linux 2.2.11: Capability bounding set
- Linux 2.6.24: File Capabilities

Geschichte

- 1997: POSIX 1003.1e – withdrawn
- Linux 2.1
- Linux 2.2.11: Capability bounding set
- Linux 2.6.24: File Capabilities
- Linux 2.6.25: Capability bounding set auf Prozessebene

Geschichte

- 1997: POSIX 1003.1e – withdrawn
- Linux 2.1
- Linux 2.2.11: Capability bounding set
- Linux 2.6.24: File Capabilities
- Linux 2.6.25: Capability bounding set auf Prozessebene
- Linux 2.6.26: Securebits

Geschichte

- 1997: POSIX 1003.1e – withdrawn
- Linux 2.1
- Linux 2.2.11: Capability bounding set
- Linux 2.6.24: File Capabilities
- Linux 2.6.25: Capability bounding set auf Prozessebene
- Linux 2.6.26: Securebits
- Linux 2.6.33: Filecapabilities nicht mehr Optional

Geschichte

- 1997: POSIX 1003.1e – withdrawn
- Linux 2.1
- Linux 2.2.11: Capability bounding set
- Linux 2.6.24: File Capabilities
- Linux 2.6.25: Capability bounding set auf Prozessebene
- Linux 2.6.26: Securebits
- Linux 2.6.33: Filecapabilities nicht mehr Optional
- Linux 3.2

Geschichte

- 1997: POSIX 1003.1e – withdrawn
- Linux 2.1
- Linux 2.2.11: Capability bounding set
- Linux 2.6.24: File Capabilities
- Linux 2.6.25: Capability bounding set auf Prozessebene
- Linux 2.6.26: Securebits
- Linux 2.6.33: Filecapabilities nicht mehr Optional
- Linux 3.2
- Linux 3.8

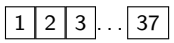
Geschichte

- 1997: POSIX 1003.1e – withdrawn
- Linux 2.1
- Linux 2.2.11: Capability bounding set
- Linux 2.6.24: File Capabilities
- Linux 2.6.25: Capability bounding set auf Prozessebene
- Linux 2.6.26: Securebits
- Linux 2.6.33: Filecapabilities nicht mehr Optional
- Linux 3.2
- Linux 3.8
- Linux 4.3: Ambient Capabilities

Theorie

Prozess

Effective E

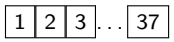


Theorie

Prozess

Permitted P

Effective $E \subseteq P$



Theorie

Prozess

Permitted P

1 2 3 ... 37

Effective $E \subseteq P$

1 2 3 ... 37

Datei (XAttr)

Permitted P_{Datei}

1 2 3 ... 37

Effective E_{Datei}

Theorie

Prozess

Permitted P
1 2 3 ... 37

Effective $E \subseteq P$
1 2 3 ... 37

Datei (XAttr)

Permitted P_{Datei}
1 2 3 ... 37

Effective E_{Datei}
□

Kindsprozess

Permitted $P_{Kind} = P_{Datei}$
1 2 3 ... 37

Effective $E_{Kind} = E_{Datei} \ ? \ P_{Kind}$
1 2 3 ... 37

Theorie

Prozess

Permitted P
1 2 3 ... 37

Effective $E \subseteq P$
1 2 3 ... 37

Inheritable I
1 2 3 ... 37

Datei (XAttr)

Permitted P_{Datei}
1 2 3 ... 37

Effective E_{Datei}
□

Inheritable I_{Datei}
1 2 3 ... 37

Kindsprozess

Permitted $P_{Kind} = P_{Datei}$
1 2 3 ... 37

Effective $E_{Kind} = E_{Datei} \ ? \ P_{Kind}$
1 2 3 ... 37

Theorie

Prozess

Permitted P
1 2 3 ... 37

Effective $E \subseteq P$
1 2 3 ... 37

Inheritable I
1 2 3 ... 37

Datei (XAttr)

Permitted P_{Datei}
1 2 3 ... 37

Effective E_{Datei}
□

Inheritable I_{Datei}
1 2 3 ... 37

Kindsprozess

Permitted $P_{Kind} = P_{Datei}$
1 2 3 ... 37

Effective $E_{Kind} = E_{Datei} \ ? \ P_{Kind}$
1 2 3 ... 37

Inheritable $I_{Kind} = I$
1 2 3 ... 37

Theorie

Prozess

Permitted P
1 2 3 ... 37

Effective $E \subseteq P$
1 2 3 ... 37

Inheritable I
1 2 3 ... 37

Datei (XAttr)

Permitted P_{Datei}
1 2 3 ... 37

Effective E_{Datei}
□

Inheritable I_{Datei}
1 2 3 ... 37

Kindsprozess

Permitted $P_{Kind} = P_{Datei} \cup (I \cap I_{Datei})$
1 2 3 ... 37

Effective $E_{Kind} = E_{Datei}\ ?\ P_{Kind}$
1 2 3 ... 37

Inheritable $I_{Kind} = I$
1 2 3 ... 37

Theorie

Prozess

Permitted P
1 2 3 ... 37

Effective $E \subseteq P$
1 2 3 ... 37

Inheritable I
1 2 3 ... 37

Ambient A
1 2 3 ... 37

Datei (XAttr)

Permitted P_{Datei}
1 2 3 ... 37

Effective E_{Datei}
□

Inheritable I_{Datei}
1 2 3 ... 37

Kindsprozess

Permitted $P_{Kind} = P_{Datei} \cup (I \cap I_{Datei})$
1 2 3 ... 37

Effective $E_{Kind} = E_{Datei}\ ?\ P_{Kind}$
1 2 3 ... 37

Inheritable $I_{Kind} = I$
1 2 3 ... 37

Theorie

Prozess

Permitted P
1 2 3 ... 37

Effective $E \subseteq P$
1 2 3 ... 37

Inheritable I
1 2 3 ... 37

Ambient A
1 2 3 ... 37

Datei (XAttr)

Permitted P_{Datei}
1 2 3 ... 37

Effective E_{Datei}
□

Inheritable I_{Datei}
1 2 3 ... 37

Kindsprozess

Permitted $P_{Kind} = P_{Datei} \cup (I \cap I_{Datei})$
1 2 3 ... 37

Effective $E_{Kind} = E_{Datei}\ ?\ P_{Kind}$
1 2 3 ... 37

Inheritable $I_{Kind} = I$
1 2 3 ... 37

Ambient $A_{Kind} = (\text{Datei priviligiert})\ ?\ 0\ :\ A$
1 2 3 ... 37

Theorie

Prozess

Permitted P
1 2 3 ... 37

Effective $E \subseteq P$
1 2 3 ... 37

Inheritable I
1 2 3 ... 37

Ambient A
1 2 3 ... 37

Datei (XAttr)

Permitted P_{Datei}
1 2 3 ... 37

Effective E_{Datei}
□

Inheritable I_{Datei}
1 2 3 ... 37

Kindsprozess

Permitted $P_{Kind} = P_{Datei} \cup (I \cap I_{Datei}) \cup A_{Kind}$
1 2 3 ... 37

Effective $E_{Kind} = E_{Datei}\ ?\ P_{Kind}\ :\ A_{Kind}$
1 2 3 ... 37

Inheritable $I_{Kind} = I$
1 2 3 ... 37

Ambient $A_{Kind} = (\text{Datei priviligiert})\ ?\ 0\ :\ A$
1 2 3 ... 37

Theorie

Prozess

Permitted P
1 2 3 ... 37

Effective $E \subseteq P$
1 2 3 ... 37

Inheritable I
1 2 3 ... 37

Ambient A
1 2 3 ... 37

Datei (XAttr)

Permitted P_{Datei}
1 2 3 ... 37

Effective E_{Datei}
□

Inheritable I_{Datei}
1 2 3 ... 37

Kindsprozess

Permitted $P_{Kind} = P_{Datei} \cup (I \cap I_{Datei}) \cup A_{Kind}$
1 2 3 ... 37

Effective $E_{Kind} = E_{Datei}\ ?\ P_{Kind}\ :\ A_{Kind}$
1 2 3 ... 37

Inheritable $I_{Kind} = I$
1 2 3 ... 37

Ambient $A_{Kind} = (\text{Datei priviligiert})\ ?\ 0\ :\ A$
1 2 3 ... 37

Theorie

Prozess

Permitted P
1 2 3 ... 37

Effective $E \subseteq P$
1 2 3 ... 37

Inheritable I
1 2 3 ... 37

Ambient A
1 2 3 ... 37

Datei (XAttr)

Permitted P_{Datei}
1 2 3 ... 37

Effective E_{Datei}
□

Inheritable I_{Datei}
1 2 3 ... 37

Kindsprozess

Permitted $P_{Kind} = (P_{Datei} \cap Bset) \cup (I \cap I_{Datei}) \cup A_{Kind}$
1 2 3 ... 37

Effective $E_{Kind} = E_{Datei} \ ? \ P_{Kind} \ : \ A_{Kind}$
1 2 3 ... 37

Inheritable $I_{Kind} = I$
1 2 3 ... 37

Ambient $A_{Kind} = (\text{Datei priviligiert}) \ ? \ 0 \ : \ A$
1 2 3 ... 37

Theorie

Prozess

Permitted P
1 2 3 ... 37

Effective $E \subseteq P$
1 2 3 ... 37

Inheritable I
1 2 3 ... 37

Ambient A
1 2 3 ... 37

Datei (XAttr)

Permitted P_{Datei}
1 2 3 ... 37

Effective E_{Datei}
□

Inheritable I_{Datei}
1 2 3 ... 37

Kindsprozess

Permitted $P_{Kind} = (P_{Datei} \cap Bset) \cup (I \cap I_{Datei}) \cup A_{Kind}$
1 2 3 ... 37

Effective $E_{Kind} = E_{Datei} \ ? \ P_{Kind} \ : \ A_{Kind}$
1 2 3 ... 37

Inheritable $I_{Kind} = I$
1 2 3 ... 37

Ambient $A_{Kind} = (\text{Datei priviligiert}) \ ? \ 0 \ : \ A$
1 2 3 ... 37

Securebits flags & UID changes

- Exec mit SUID bit $\Rightarrow$ $P_{Inheritable} = P_{Effective} = P_{Permitted} = 1$

Securebits flags & UID changes

- Exec mit SUID bit $\Rightarrow$ $P_{Inheritable} = P_{Effective} = P_{Permitted} = 1$
- Exec mit UID $= 0$ $\Rightarrow$ $P_{Inheritable} = P_{Effective} = P_{Permitted} = 1$

Securebits flags & UID changes

- Exec mit SUID bit $\Rightarrow$ $P_{Inheritable} = P_{Effective} = P_{Permitted} = 1$
- Exec mit UID $= 0$ $\Rightarrow$ $P_{Inheritable} = P_{Effective} = P_{Permitted} = 1$
- UID $= 0 \rightarrow > 0$ $\Rightarrow$ $P_{Effective} = P_{Permitted} = 0$

Securebits flags & UID changes

- Exec mit SUID bit $\Rightarrow$ $P_{Inheritable} = P_{Effective} = P_{Permitted} = 1$
- Exec mit UID $= 0$ $\Rightarrow$ $P_{Inheritable} = P_{Effective} = P_{Permitted} = 1$
- UID $= 0 \rightarrow > 0$ $\Rightarrow$ $P_{Effective} = P_{Permitted} = 0$
- UID $> 0 \rightarrow = 0$ $\Rightarrow$ $P_{Effective} = P_{Permitted}$

Securebits flags & UID changes

- Exec mit SUID bit $\Rightarrow$ $P_{Inheritable} = P_{Effective} = P_{Permitted} = 1$
- Exec mit UID $= 0$ $\Rightarrow$ $P_{Inheritable} = P_{Effective} = P_{Permitted} = 1$
- UID $= 0 \rightarrow > 0$ $\Rightarrow$ $P_{Effective} = P_{Permitted} = 0$
- UID $> 0 \rightarrow = 0$ $\Rightarrow$ $P_{Effective} = P_{Permitted}$

Deaktivierbar durch:

- SECBIT_KEEP_CAPS
- SECBIT_NO_SETUID_FIXUP
- SECBIT_NOROOT
- SECBIT_NO_CAP_AMBIENT_RAISE

Securebits flags & UID changes

- Exec mit SUID bit $\Rightarrow$ $P_{Inheritable} = P_{Effective} = P_{Permitted} = 1$
- Exec mit UID $= 0$ $\Rightarrow$ $P_{Inheritable} = P_{Effective} = P_{Permitted} = 1$
- UID $= 0 \rightarrow > 0$ $\Rightarrow$ $P_{Effective} = P_{Permitted} = 0$
- UID $> 0 \rightarrow = 0$ $\Rightarrow$ $P_{Effective} = P_{Permitted}$

Arretierbar durch:

- SECBIT_KEEP_CAPS_LOCK
- SECBIT_NO_SETUID_FIXUP_LOCK
- SECBIT_NOROOT_LOCK
- SECBIT_NO_CAP_AMBIENT_RAISE_LOCK

CAPABILITY-AWARENESS

- capability-dumb binaries

Beispiel:

```
# old system
$ ls -l /usr/bin/ping
-rwsr-xr-x 1 root root 60K 16. Nov 08:57 /usr/bin/ping

# modern system
$ ls -l /usr/bin/ping
-rwxr-xr-x 1 root root 60K 16. Nov 08:57 /usr/bin/ping
$ getcap /usr/bin/ping
/usr/bin/ping = cap_net_raw+ep
```

CAPABILITY-AWARENESS

- capability-dumb binaries
- capability-aware binaries

Beispiel:

```
#  ifdef HAVE_LINUX_CAPABILITY
   // ...
   cap_t caps;
   char *captext;

   captext = want_dynamic_interface_tracking
      ? "cap_sys_nice,cap_sys_time,cap_net_bind_service=pe"
      : "cap_sys_nice,cap_sys_time=pe";
   caps = cap_from_text(captext);
   // ...
   cap_free(caps);
#  endif  /* HAVE_LINUX_CAPABILITY */
```

NTPsec: `ntpd/ntp_sandbox.c`

Probleme

CAP_SYS_ADMIN

> *It can plausibly be called „the new root“, since on the one hand, it confers a wide range of powers, and on the other hand, its broad scope means that this is the capability that is required by many privileged programs.*
>
> — man CAPABILITIES(7)

Probleme

```
uid_t euid=geteuid();
if(euid != 0)
{
  // Tell user to run app as root, then exit.
}
```

Probleme

- Root User ist Besitzer aller Systemdateien: /etc/shadow, /usr/bin/su, ...

Probleme

- Root User ist Besitzer aller Systemdateien: /etc/shadow, /usr/bin/su, ...
  $\Rightarrow$ Root ohne Capabilities ist exploitable!

Probleme

- Root User ist Besitzer aller Systemdateien: /etc/shadow, /usr/bin/su, ...
  - $\Rightarrow$ Root ohne Capabilities ist exploitable!
    - $\Rightarrow$ Verwende User mit $UID > 0$!

Probleme

Etwa 20 der Capabilities sind Exploitable und erlauben es volle Rechte zu erlangen!

Probleme

Etwa 20 der Capabilities sind Exploitable und erlauben es volle Rechte zu erlangen!
Einfaches Beispiel:
CAP_FOWNER

```
$ chown hacker /etc/shadow
$ chmod 777 /etc/shadow
$ vim /etc/shadow
$ su
```

Probleme

Etwa 20 der Capabilities sind Exploitable und erlauben es volle Rechte zu erlangen!
Einfaches Beispiel:
CAP_FOWNER

```
$ chown hacker /etc/shadow
$ chmod 777 /etc/shadow
$ vim /etc/shadow
$ su
```

Betrifft vor allem capability-dump binaries.

Probleme

Etwa 20 der Capabilities sind Exploitable und erlauben es volle Rechte zu erlangen!
Einfaches Beispiel:
CAP_FOWNER

```
$ chown hacker /etc/shadow
$ chmod 777 /etc/shadow
$ vim /etc/shadow
$ su
```

Betrifft vor allem capability-dump binaries. Einschränkbar durch Capability-aware binaries, strengere Dateirechte, capability bounding set, Grsec, Selinux, etc. . .

Praktisch

- getcap: Capabilities einer Datei anzeigen

Praktisch

- getcap: Capabilities einer Datei anzeigen
- setcap: Capabilities einer Datei setzen

Praktisch

- getcap: Capabilities einer Datei anzeigen
- setcap: Capabilities einer Datei setzen
- filecap: Capabilities aller Dateien anzeigen

Praktisch

- getcap: Capabilities einer Datei anzeigen
- setcap: Capabilities einer Datei setzen
- filecap: Capabilities aller Dateien anzeigen
- getpcaps: Capabilities eines Prozesses anzeigen

Praktisch

- getcap: Capabilities einer Datei anzeigen
- setcap: Capabilities einer Datei setzen
- filecap: Capabilities aller Dateien anzeigen
- getpcaps: Capabilities eines Prozesses anzeigen
- pscap: Capabilities aller Prozesse anzeigen

Praktisch

- getcap: Capabilities einer Datei anzeigen
- setcap: Capabilities einer Datei setzen
- filecap: Capabilities aller Dateien anzeigen
- getpcaps: Capabilities eines Prozesses anzeigen
- pscap: Capabilities aller Prozesse anzeigen
- capsh: Wrapper, um ein Shell mit Capabilities zu setzen

Praktisch

- getcap: Capabilities einer Datei anzeigen
- setcap: Capabilities einer Datei setzen
- filecap: Capabilities aller Dateien anzeigen
- getpcaps: Capabilities eines Prozesses anzeigen
- pscap: Capabilities aller Prozesse anzeigen
- capsh: Wrapper, um ein Shell mit Capabilities zu setzen
- captest: Testet die Capabilities

Praktisch

- getcap: Capabilities einer Datei anzeigen
- setcap: Capabilities einer Datei setzen
- filecap: Capabilities aller Dateien anzeigen
- getpcaps: Capabilities eines Prozesses anzeigen
- pscap: Capabilities aller Prozesse anzeigen
- capsh: Wrapper, um ein Shell mit Capabilities zu setzen
- captest: Testet die Capabilities
- libcap: C-libary für Capability-aware libaries

Praktisch

- getcap: Capabilities einer Datei anzeigen
- setcap: Capabilities einer Datei setzen
- filecap: Capabilities aller Dateien anzeigen
- getpcaps: Capabilities eines Prozesses anzeigen
- pscap: Capabilities aller Prozesse anzeigen
- capsh: Wrapper, um ein Shell mit Capabilities zu setzen
- captest: Testet die Capabilities
- libcap: C-libary für Capability-aware libaries
- ls – Dateien mit Capabilities sind farbig!

Praktisch

- getcap: Capabilities einer Datei anzeigen
- setcap: Capabilities einer Datei setzen
- filecap: Capabilities aller Dateien anzeigen
- getpcaps: Capabilities eines Prozesses anzeigen
- pscap: Capabilities aller Prozesse anzeigen
- capsh: Wrapper, um ein Shell mit Capabilities zu setzen
- captest: Testet die Capabilities
- libcap: C-libary für Capability-aware libaries
- ls – Dateien mit Capabilities sind farbig!

Praktisch

- getcap: Capabilities einer Datei anzeigen
- setcap: Capabilities einer Datei setzen
- filecap: Capabilities aller Dateien anzeigen
- getpcaps: Capabilities eines Prozesses anzeigen
- pscap: Capabilities aller Prozesse anzeigen
- capsh: Wrapper, um ein Shell mit Capabilities zu setzen
- captest: Testet die Capabilities
- libcap: C-libary für Capability-aware libaries
- ls – Dateien mit Capabilities sind farbig! – Außer auf Debian!

Praktisch

- getcap: Capabilities einer Datei anzeigen
- setcap: Capabilities einer Datei setzen
- filecap: Capabilities aller Dateien anzeigen
- getpcaps: Capabilities eines Prozesses anzeigen
- pscap: Capabilities aller Prozesse anzeigen
- capsh: Wrapper, um ein Shell mit Capabilities zu setzen
- captest: Testet die Capabilities
- libcap: C-libary für Capability-aware libaries
- ls – Dateien mit Capabilities sind farbig! – Außer auf Debian!
- systemd – Capabilities können in Unit-Files spezifiziert werden

Praktisch

- getcap: Capabilities einer Datei anzeigen
- setcap: Capabilities einer Datei setzen
- filecap: Capabilities aller Dateien anzeigen
- getpcaps: Capabilities eines Prozesses anzeigen
- pscap: Capabilities aller Prozesse anzeigen
- capsh: Wrapper, um ein Shell mit Capabilities zu setzen
- captest: Testet die Capabilities
- libcap: C-libary für Capability-aware libaries
- ls – Dateien mit Capabilities sind farbig! – Außer auf Debian!
- systemd – Capabilities können in Unit-Files spezifiziert werden
- pam_cap – inheritable capabilities beim Anmelden setzen

Praktisch

- getcap: Capabilities einer Datei anzeigen
- setcap: Capabilities einer Datei setzen
- filecap: Capabilities aller Dateien anzeigen
- getpcaps: Capabilities eines Prozesses anzeigen
- pscap: Capabilities aller Prozesse anzeigen
- capsh: Wrapper, um ein Shell mit Capabilities zu setzen
- captest: Testet die Capabilities
- libcap: C-libary für Capability-aware libaries
- ls – Dateien mit Capabilities sind farbig! – Außer auf Debian!
- systemd – Capabilities können in Unit-Files spezifiziert werden
- pam_cap – inheritable capabilities beim Anmelden setzen
- Docker – nutzt intensiv die Capabilities, anpassbar

Vielen Dank für Ihre Aufmerksamkeit!