INF226 – Software Security

Håkon Robbestad Gylterud

2019–10–09

Security shepherd demo

Our security shepherd instance: `https://shepherd.ii.uib.no/`.

Last time

We talked about:

- Confused deputy
- Capability based security

Today:

- Capsicum – an implementation of capabilities in FreeBSD
- Incorrect deserialisation – an up-and-coming class of vulnereabilities.

Capsicum

Priviledge separation

We have previously studied the priviledge separation mechanisms used by OpenSSH:

- Monitor/slave model
- Unpriviledged UID/GID
- chroot to empty, unwriteable directory
- `P_SUGID`

Priviledge separation

Drawbacks:

- Chroot requires UID 0.
- When transitioning between priviledges data must be serialised.
- Relies on shared memory.
- Resoning about security requires modelling monitor as a state machine.
- Does not limit network access from slave.

For something more complicated, like a web-browser, this becomes *difficult*.

Capsicum

Design goals:

- Provide capability based security for Unix programs.
- Extend, instead of replacing, Unix APIs.
- Performance comparable to already employed priviledge separation mechanisms.

Capsicum

Design:

- Introduces a special **capability mode** for processes
- Provide **new kernel primitives** (`cap_enter`, `cap_new`, $\cdots$)
- Changes existing kernel primitives when in capability mode.
- **Userspace library** (*libcapsicum*).

Capsicum

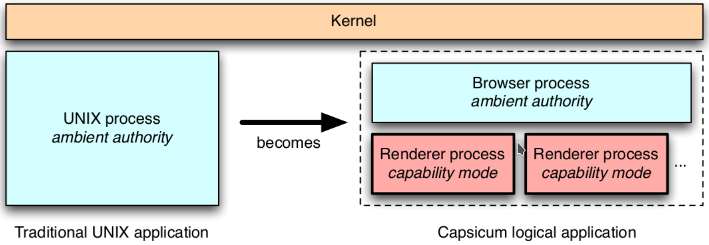


Figure 1: Capsicum capabilities

Capabilities

In capsicum, **capabilities are file descriptors** along with a set of access rights.

- There are roughly 60 possible access rights for a capability in capsicum.

A capability is created though `cap_new` by giving it a file descriptor and rights mask.

- Capabilities are transferred though Inter Process Communication (IPC) channels, such as sockets.

Capabilities

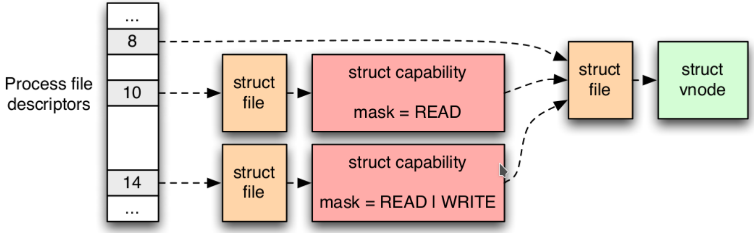


Figure 2: Capsicum capabilities

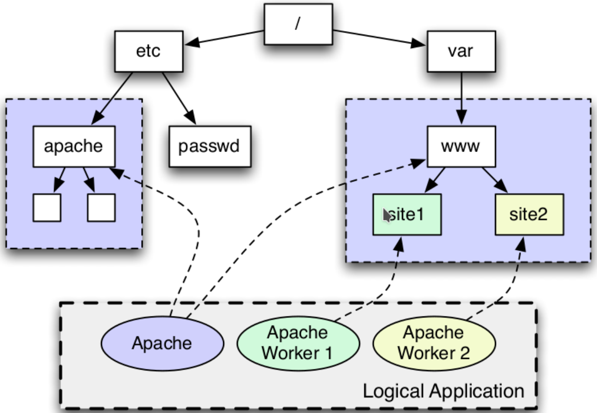
Enforcing capabilities

Capability mode resricts access to global name spaces such as:

- Process ID
- File paths
- POSIX IPC (inter-process communication)
- System clocks/timers

In capability mode these resources **can only be accessed through capabilities**.

Enforcing capabilities



Restricting existing kernel primitives

In order to enforce these restictions, man kernel primitives must be changed:

`openat(desc,path)` opens a file located at relative path from the directory referenced by file descirptor `desc`

Example: In capability mode: If 4 refers to `/lib` then:

- `openat(4,"libc.so.7")` is *valid*
- `openat(4,"../etc/passwd")` is *invalid*

In general **no “..” allowed** in capability mode.

Restricting existing kernel primitives

In capability mode, the only valid PID is the process' own PID.

Child processes (spawned by `fork`) can be accessed through capabilities.

(Following the principle of **access by creation**)

Run-time environment

System calls for execution, such as `fork`, use global name space through the ELF-header:

- The ELF header contains an absolute path to a run-time linker.

`libcapsicum` contains a special-purpose run-time linker, which loads libraries through capabilitities.

Adapting programs to Capsicum

Typical usage of Capsicum

The structure of most programs using capsicum:

1. Obtain resources (using system ambient authorities)
2. Wrap resources in capabilitiets
3. Enter capability mode.
4. Use resources

Observation: Each program uses capabilities in isolation. The system itself still based on traditional security model.

tcpdump

`tcpdump` outputs descriptions of network packets matching a given filter.

- **Question:** What vulnerabilities could we expect in such a program?

tcpdump

`tcpdump` outputs descriptions of network packets matching a given filter.

- **Question:** What vulnerabilities could we expect in such a program?

The structure of the program lends itself well to priviledge separation:

- Priviledges are aquired early.
- Priviledged operations are **separate from the messy parsing of packets**.

tcpdump

`tcpdump` outputs descriptions of network packets matching a given filter.

- **Question:** What vulnerabilities could we expect in such a program?

The structure of the program lends itself well to priviledge separation:

- Priviledges are aquired early.
- Priviledged operations are **separate from the messy parsing of packets**.

Minor quirk: DNS resolver relied on file access, and thus had to be changed to external daemon.

dhclient

`dhclient` is OpenBSD's DHCP client. Uses priviledge separation already.

- Hardening this priviledge separation through Capsicum was a two-line change.

gzip

`gzip` is a command-line file-compession tool.

- **Question**: What kinds of vulnerabilities would you expect in this program?

gzip

gzip is a command-line file-compession tool.

- **Question**: What kinds of vulnerabilities would you expect in this program?

Priviledge separation though chroot/unpriviledged UID is a poor match.

Modifying gzip to use libcapsicum:

- Three critical compression functions are put in capability mode.
- 409 lines added to gzip (16% increase)

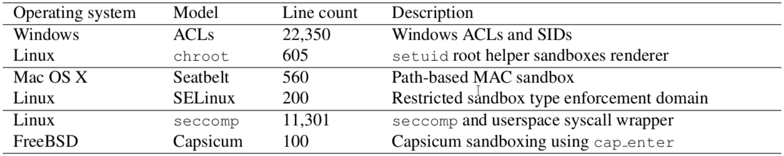
Chromium

Chromium is the open-source sibling of the Chrome web-browser, developed by Google.

- More than 4 million lines of code.
- Chromium has **integrated sandboxing**, with different implementations on different platforms:
  - Each tab is a *renderer process*.
  - Resources already forwarded through file descriptors.

Before Capsicum, the FreeBSD port of Chrome did not use any sandboxing.

Chromium on different priviledge-separation technologies



Operating system	Model	Line count	Description
Windows	ACLs	22,350	Windows ACLs and SIDs
Linux	`chroot`	605	`setuid` root helper sandboxes renderer
Mac OS X	Seatbelt	560	Path-based MAC sandbox
Linux	SELinux	200	Restricted sandbox type enforcement domain
Linux	`seccomp`	11,301	`seccomp` and userspace syscall wrapper
FreeBSD	Capsicum	100	Capsicum sandboxing using `cap_enter`

Figure 4: Chromuim

Insecure deserialisation

Serialization

Serialization is the process of turning objects of a programming language into byte arrays for transport.

Serialization

Serialization is the process of turning objects of a programming language into byte arrays for transport.

Deserialization is the process of turning these byte arrays back into objects.

Serialization

Examples of serialization libraries:

- Java serialization
- JSON (Multiple language support)
- Pickle (Python)
- Protocol buffers

Incorrect deserialization

The code doing deserialization is at the forefront of the program security.

Bugs in deserialization can often lead to *remote code execution*.

Pickles

The Pickle Python library is explicitly dangerous:

> ***Warning:*** *The pickle module is not secure against erroneous or maliciously constructed data. Never unpickle data received from an untrusted or unauthenticated source*

(The python documentation)

Example exploit:
`https://www2.cs.uic.edu/~s/musings/pickle/`

Vulnerability on a Facebook server last year:

https://blog.scrt.ch/2018/08/24/remote-code-execution-on-a-facebook-server/

Java serialization

```
import java.io.Serializable;
public class Person implements Serializable {
    private static final long serialVersionUID
       = -7181352062979002929L;
    private final String name;
    private final Integer age;
// ...
```

Java serialization

Constructors some times do sanity/security checks:

```
public Person(String name, Integer age)
  throws NegativeAgeException {
     this.name = name;
     if(age < 0) throw new NegativeAgeException();
     this.age=age;
 }
```

Java serializaion

Writing an object

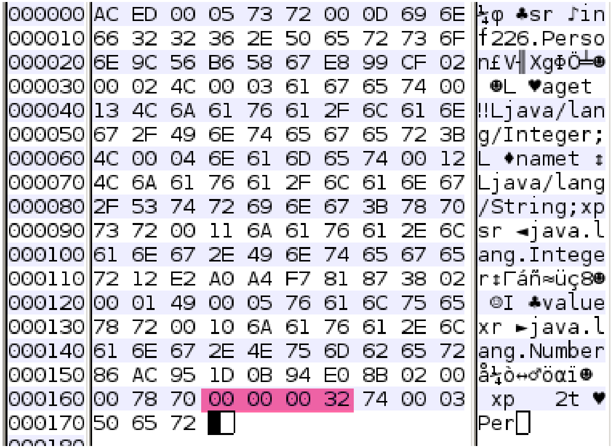
```
Person per = new Person("Per", 50);
ObjectOutputStream oos
  = new ObjectOutputStream(
     new FileOutputStream("/tmp/person.bin"));
oos.writeObject(per);
oos.flush();
oos.close();
```

Java deserialization

Reading an object:

```
ObjectInputStream ois
  = new ObjectInputStream(
      new FileInputStream("/tmp/person.bin"));
Person per = (Person)ois.readObject();
ois.close();
System.out.println(per.getAge());
```

Editing the object before reading:



```
000000 AC ED 00 05 73 72 00 0D 69 6E  ¼φ ♣sr ♪in
000010 66 32 32 36 2E 50 65 72 73 6F  f226.Perso
000020 6E 9C 56 B6 58 67 E8 99 CF 02  n£V╢XgΦÖ╧☻
000030 00 02 4C 00 03 61 67 65 74 00   ☻L ♥aget
000040 13 4C 6A 61 76 61 2F 6C 61 6E  ‼Ljava/lan
000050 67 2F 49 6E 74 65 67 65 72 3B  g/Integer;
000060 4C 00 04 6E 61 6D 65 74 00 12  L ♦namet ↕
000070 4C 6A 61 76 61 2F 6C 61 6E 67  Ljava/lang
000080 2F 53 74 72 69 6E 67 3B 78 70  /String;xp
000090 73 72 00 11 6A 61 76 61 2E 6C  sr ◄java.l
000100 61 6E 67 2E 49 6E 74 65 67 65  ang.Intege
000110 72 12 E2 A0 A4 F7 81 87 38 02  r↕Γáñ≈üç8☻
000120 00 01 49 00 05 76 61 6C 75 65   ☺I ♣value
000130 78 72 00 10 6A 61 76 61 2E 6C  xr ►java.l
000140 61 6E 67 2E 4E 75 6D 62 65 72  ang.Number
000150 86 AC 95 1D 0B 94 E0 8B 02 00  å¼ò↔♂öαï☻
000160 00 78 70 00 00 00 32 74 00 03   xp   2t ♥
000170 50 65 72                       Per
```

Figure 5: person.bin

Bypassing the sanity check in the constructor

If we change `00 00 00 32` to `FF 00 00 32`, the reading program outputs:

`-16777166`, a negative number!

Security holes

For `Person` this might not lead to a security hole directly.

But what if the constructor is used to escape HTML, or SQL data?

Security holes

For `Person` this might not lead to a security hole directly.

But what if the constructor is used to escape HTML, or SQL data?

Then we could get XSS or SQL injection vulnerabilities.

Java reflection & deserialization

Java has relfection, which gives dynamic method invocation.

- Takes a method name string, and argument strings
- Applies it to an object

Together with insecure deserialization this gives **remote code execution**, when the attacker can alter the method name and arguments to something malicious.

Some details:

- https://www.youtube.com/watch?v=VviY3O-euVQ

Muddiest point

Answer on `mitt.uib.no`.