Can you advise me how to recognize if it’s safe or how to set it’s limited rights in the system?

I downloaded this shell script from this site.

It's suspiciously large for a bash script. So I opened it with text editor and noticed that behind the code there is a lot of non-sense characters.

I'm afraid of giving the script execution right with `chmod +x j.d.sh`. Can you advise me how to recognize if it's safe or how to set it's limited rights in the system?

thank you

security  shell  sh  chmod  access-rights

share | improve this question

asked Nov 25 '11 at 12:25

xralf

2,126 • 7 • 31 • 81
“Every program ... should operate using the least amount of privilege necessary to complete the job.”

—Saltzer [CACM ’74]
Every program ... should operate using the least amount of privilege necessary to complete the job.

—Saltzer [CACM ’74]

How do we limit the authority of scripts?

How do we determine what authority is necessary?
Declarative security policy

End-to-end policy enforcement

Kernel-based enforcement for executables

Shill script

sandbox

a.out

policy

policy
Capabilities to manifest authority

Contracts to communicate authority

Contracts and sandboxes to control authority

= shill

Scripting with Least Privilege
Capabilities
Capability-based security

A capability is an unforgeable token of authority.


Dennis and Van Horn

PDP-1 Supervisor CAL-TSS PSOS

KeyKOS NTLSS

EROS Coyotos seL4

CapDesk PLASH Polaris

CapDesk

Google Caja

Joe-E Oz-E CaPerl Emily

W7

Capsicum
A capability-safe script

copy = fun(from_dir,to_dir) {
    for entry in contents(from_dir) do {
        current = lookup(from_dir,entry);
        if is_file(current) then {
            new = create_file(to_dir,name);
            write(new,read(current))
        }
    }
}
A capability-safe script:

```plaintext
capabilities: files, directories, pipes, sockets, ...

```
A capability-safe script

```plaintext
copy = fun(from_dir, to_dir) {
    for entry in contents(from_dir) do {
        current = lookup(from_dir, entry);
        if is_file(current) then {
            new = create_file(to_dir, name);
            write(new, read(current))
        }
    }
}
```

**operations:** require privileges on capabilities

Directories: +contents, +lookup, +unlink, ...
Files: +read, +stat, +path, ...
Sockets: +bind, +send, +receive, ...
A capability-safe script

copy = fun(from_dir, to_dir) {
    for entry in contents(from_dir) do {
        current = lookup(from_dir, entry);
        if is_file(current) then {
            new = create_file(to_dir, name);
            write(new, read(current))
        }
    }
}

Directories: +contents, +lookup, +unlink, ...
Files: +read, +stat, +path, ...
Sockets: +bind, +send, +receive, ...
A capability-safe script

```lisp
copy = fun(from_dir, to_dir) { 
    for entry in contents(from_dir) do {
        current = lookup(from_dir, entry);
        if is_file(current) then {
            new = create_file(to_dir, name);
            write(new, read(current))
        }
    }
}
```

Capability safety

All resources are represented by capabilities
Scripts have no capabilities by default
Contracts
Software contracts

Applying “Design by Contract”,
Bertrand Meyer [IEEE Computer ’92]
Software contracts in Shill

provide find_c :
{ dir : is_dir }
\rightarrow listof(is_filename);

find_c = fun(dir) {
  filter(fun (entry) has_ext("c", entry),
            contents(dir));
}
Capability contracts in Shill

More precise contract specifying `+contents` privilege

```haskell
provide find_c :
  { dir : is_dir(+contents) } -> listof(is_filename);

find_c = fun(dir) {
  filter(fun (entry) has_ext("c", entry), contents(dir));
}
```

Capability contracts both **require** and **restrict** privileges
Sandboxing
Capability-based sandboxing

```
#!/usr/bin/bash
...
./a.out $test &> $log
...
```

unknown binary

Need to enforce security on executables!

Emulate capabilities with Mandatory Access Control
Capability-based sandboxing

File capability with `+exec` arguments

```c
exec(aout, ["test.txt"],
    stdout = log, stderr = log,
    extra = [libc, cwd, ...]);
```

capabilities for stdin/out/err

additional capabilities for sandbox
Mandatory Access Control Policy

```
open(“..;/alice/dog.jpg”, O_RDONLY)
```
Label objects with capabilities

```
open("..;/alice/dog.jpg", O_RDONLY)
```
Label objects with capabilities

Authorize actions against permissions

```c
open("../alice/dog.jpg", O_RDONLY)
```

foo.c
Mandatory Access Control Policy

Label objects with capabilities

Authorize actions against permissions

```c
open("../alice/dog.jpg", O_RDONLY)
```

foo.c
Label objects with capabilities

Authorize actions against permissions

```c
open("../alice/dog.jpg", O_RDONLY)
```
Label objects with capabilities

Authorize actions against permissions

```c
open("../alice/dog.jpg", O_RDONLY)
```

foo.c
Mandatory Access Control Policy

Label objects with capabilities

Authorize actions against permissions

```
open("../alice/dog.jpg", O_RDONLY)
```

foo.c
Label objects with capabilities

Authorize actions against permissions

Propagate new capabilities

open("../../../alice/dog.jpg", O_RDONLY)
Label objects with capabilities
Authorize actions against permissions
Propagate new capabilities

```
open("../alice/dog.jpg", O_RDONLY)
```

```
foo.c
```
Managing capabilities

Difficult to gather all the capabilities needed

```bash
/bin/cat
foo.txt
&gt;stdout
/libexec/ld-elf.so.1
/var/run/ld-elf.so.hints
/etc/libmap.conf
/lib/libc.so.7
/usr/share/locale/en_US.UTF-8
/usr/share/locale/UTF-8/LC_CTYPE
```

Need abstractions: **capability wallets**
package sets of capabilities with contracts
standard library for running binaries

```plaintext
populate-native-wallet(wallet,path,ld_path,...);
...
gcc = pkg-native(wallet,“gcc”);
gcc([source,“-o”,“myprog”]);
```
Putting it together
Architecture of shill

contract:
  file(+read)

capability-safe
script

shill

kernel

... ...
foo.txt
Architecture of shill

Where do capabilities come from?

contract: file(+read)
capability-safe script

foo.txt
Architecture of shill

Where do capabilities come from?

ambient script

contract: file(+read)
capability-safe script

Limited expressiveness

shill

kernel

... ...

foo.txt
Architecture of shill

Where do capabilities come from?

ambient script

contract: file(+read)

capability-safe script

foo.txt +read

Limited expressiveness

shill

kernel

foo.txt
Architecture of shill

shill

kernel

ambient
script

foo.txt

contract:
file(+read)

capability-safe
script

foo.txt

+read

foo.txt
Architecture of shill

ambient script

contract: file(+read)

capability-safe script

foo.txt

+read

sandbox

executable

“foo.txt”

shill kernel

foo.txt
Architecture of shill

ambient script

contract: file(+read)

capability-safe script

foo.txt +read

sandbox

executable

“foo.txt”

shill

kernel

foo.txt +read
Architecture of shill

ambient script

contract: file(+read)

capability-safe script

sandbox executable

“foo.txt”

shill

kernel
Example: grading script

G1. Don’t corrupt my other files
G2. Don’t modify or leak the test suite
G3. Don’t allow submissions to interact
Example: grading script

- Create log file
- Create work dir
- Copy in code
- Compile
- Run tests
- Clean up

✓ Reuse the glue for security
Scripting with Least Privilege

provide grade :
{  submission : is_file && readonly,  
  tests      : is_dir   && readonly,  
  working    : is_dir(+create_dir with full_privilege),  
  grade_log  : is_file && appendonly,  
  extras     : libc_wallet } → void;

G1. Don’t corrupt my other files

G2. Don’t modify or leak the test suite

G3. Don’t allow submissions to interact
Scripting with Least Privilege

provide grade :
{  submission : is_file && readonly,
    tests      : is_dir  && readonly,
    working    : is_dir(+create_dir with full_privilege),
    grade_log  : is_file && appendonly,
    extras     : libc_wallet } → void;

✓ G1. Don’t corrupt my other files
G2. Don’t modify or leak the test suite
G3. Don’t allow submissions to interact
Scripting with Least Privilege

provide grade:
{
    submission : is_file && readonly,
    tests : is_dir && readonly,
    working : is_dir(+create_dir with full_privilege),
    grade_log : is_file && appendonly,
    extras : libc_wallet } → void;

can’t modify test suite

no network capability to leak tests

✓ G1. Don’t corrupt my other files

✓ G2. Don’t modify or leak the test suite

✓ G3. Don’t allow submissions to interact
Scripting with Least Privilege

- **submissions isolated**
  ```
  provide grade :
  {
    submission : is_file && readonly,
    tests      : is_dir && readonly,
    working    : is_dir(+create_dir with full_privilege),
    grade_log  : is_file && appendonly,
    extras     : libc_wallet } -> void;
  ```

- **can’t modify test suite**
- **no network capability to leak tests**

- ✓ G1. Don’t corrupt my other files
- ✓ G2. Don’t modify or leak the test suite
- ✓ G3. Don’t allow submissions to interact
Evaluation
Implementation

Racket

- Capability-safe subset of racket
- Capability-based systems library
- Contracts built with Racket combinators

FreeBSD®

- Capability-based sandbox
- Policy module for Trusted MAC framework
- Few additional capability-safe system calls
Case studies

- Grading OCaml assignments
  sandboxed execution, isolation between students

- GNU Emacs installer
  sandboxed execution, install/uninstall locations

- Apache webserver
  read-only access to config and content directories

- find
  find and execute
  sandboxed execution per-file
Performance

Overhead generally below 20%

Overhead proportional to security guarantees

Baseline
Installed
Sandboxed
shill version

Performance Overhead

Overhead generally below 20%

Overhead proportional to security guarantees

Baseline
Installed
Sandboxed
shill version
Capabilities to manifest authority

Contracts to communicate authority

Contracts and sandboxes to control authority

= www.shill-lang.org