Scripting with Least Privilege

or: Contracts for Security

Scott Moore

(sixth RacketCon)
Fancy Install (Unix)

There's a pretty robust install script at https://www.npmjs.org/install.sh.

Here's an example using curl:

```
curl -L https://npmjs.org/install.sh | sh
```
```bash
cd "$TMP"
&& curl -sSsL "$url" 
  | $tar -xzf - 
&& cd "$TMP"/
&& (ver=""node" bin/read-package.json.js package.json version" install=0
  if [ $ret -eq 0 ]; then
    if [ -d node_modules ]; then
      if "$node" node_modules/semver/bin/semver -v "$ver" -r "1" then
        isrpm10=1
      fi
    else
      if "$node" bin/semver -v "$ver" -r ">=1.0"; then
        isrpm10=1
      fi
    fi
  fi
	ret=0
  if [ $isrpm10 -eq 1 ] && [ -f 'scripts/clean-old.sh' ]; then
    if [ "$x$skipclean" = 'x' ]; then
      (exit 0)
    else
      clean=no
    fi
    if [ "$x$clean" = 'xno' ] 
      || [ "$x$clean" = 'xn' ]; then
      echo 'skipping 0.x cruft clean' >&2
      ret=0
    elif [ "$x$clean" = 'xy' ] || [ "$x$clean" = 'yes' ]; then
      NODE="$node" /bin/bash "scripts/clean-old.sh" ".y'
      ret=1
    else
      NODE="$node" /bin/bash "scripts/clean-old.sh" </dev/tty
      ret=1
    fi
  fi
  if [ $ret != 0 ]; then
    echo 'Aborted 0.x cleanup. Exiting.' >&2
    exit $ret
  fi)
&& (if [ "$x$configures" = "x" ]; then
  (exit 0)
else
  echo "./configure $configures"
  echo "$configures" > nmprc
fi)
&& (if [ "$make" = "NOMAKE" ]; then
  (exit 0)
else
  echo "$make" uninstall install; then
  (exit 0)
)```
How can I recognize if it is safe?

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
Principle of Least Privilege

“Every program ... should operate using the least amount of privilege necessary to complete the job.”

—Jerome Saltzer, CACM
Scripting with Least Privilege

Shill script
Scripting with Least Privilege

Simple security policy

contract

Shill
script
Scripting with Least Privilege

Simple security policy

contract

Shill script

contract

Shill script
Scripting with Least Privilege

Simple security policy

contract

Shill script

contract

Shill script

contract

a.out
Scripting with Least Privilege

Simple security policy

contract

Shill script

contract

Shill script

sandbox

a.out

Kernel-based enforcement for executables
Scripting with Least Privilege

Simple security policy

Start-to-finish security

Kernel-based enforcement for executables
Capability-based security

A capability is an *unforgeable token of authority* that **only accesses system resources through operations on capabilities**.
A Secure Shell Script

```
#lang shill/cap

provide { copy : any/c };

copy = fun(from_dir,to_dir) {
    for entry in contents(from_dir) do {
        current = lookup(from_dir,entry);
        if file?(current) then {
            new = create-file(to_dir,entry);
            write(new,read(current))
        }
    }
}
```

#lang shill/cap

provide { copy : any/c }; 

copy = fun(from_dir,to_dir) {
    for entry in contents(from_dir) do {
        current = lookup(from_dir,entry);
        if file?(current) then {
            new = create-file(to_dir,entry);
            write(new,read(current))
        }
    }
}

Capability safety: scripts have no capabilities by default
Fine-grained security with contracts

```shill/cap
#lang shill/cap
provide { copy : {
  from : dir/c(+contents, +lookup with { +read }),
  to   : dir/c(+create-file with { +write })
} -> void ;
require shill/io;
copy = fun(from_dir,to_dir) {
  for entry in contents(from_dir) do {
    current = lookup(from_dir,entry);
    if file?(current) then {
      fwrite(current,"evil");
      new = create-file(to_dir,entry);
      write(new,read(current))
    }
  }
}
```

Contracts describe exactly how a script will use its capabilities
Fine-grained security with contracts

```shill/cap
#lang shill/cap
provide { copy : {
  from : dir/c(+contents, +lookup with { +read }),
  to   : dir/c(+create-file with { +write })
} -> void;
require shill/io;

copy = fun(from_dir,to_dir) {
  for entry in contents(from_dir) do {
    current = lookup(from_dir,entry);
    if file?(current) then {
      fwrite(current,"evil");
      new = create-file(to_dir,entry);
      write(new,read(current))
    }
  }
}
```
# Programmable sandboxes

```shill/cap

provide { cat : { 
    cat     : file/c(+exec,+read,+stat),
    file    : file/c(+path,+read),
    lookup  : listof(dir/c(+lookup,+stat)),
    libs    : listof(file/c(+exec,+read,+stat)),
    ro      : listof(file/c(+read,+stat)),
    out     : writeable/c } 
  -> integer? };

require shill/contracts;

val cat = fun(cat,file,lookups,libs,ro,out) { 
  exec(cat, ["cat", file], 
         stdout = out, stderr = out, 
         extra = list-append(lookups,libs,ro));
}
```
Under the hood

Black box capability-based sandboxing for executables

+ a few new capability-safe system calls

#lang shill/cap:

Capability-safe safe subset of racket/base

+ a set!-transformer to control mutation

+ a require-transformer to only import shill code

+ a capability-based filesystem library using ffi/unsafe

+ capability contracts using racket/contract

+ a custom reader
What’s next

Developing commercial version of Shill

Porting to Linux

Plug-in framework for new kinds of capabilities (processes, databases, ...)