Security in Plan 9

Russ Cox, MIT LCS
Eric Grosse, Bell Labs
Rob Pike, Bell Labs
Dave Presotto, Avaya Labs and Bell Labs
Sean Quinlan, Bell Labs

rsc,ehg,rob,presotto,seanq
@
plan9.bell-labs.com

What comprises a security architecture? ....

An interface for applications to

- authenticate users and services
- establish secure channels

A mechanism to manage authentication secrets (keys)

Lots of code to implement cryptographic protocols and functions

OS protection: user id, access permission, etc.

Plan 9..

Research operating system developed at Bell Labs

- development since late 1980s
- base for other research since 1995

Easy for us to work on:

- we wrote and control all the source
- simple design makes everything easier

Plan 9 principles (and security implications)..

Plan 9 principles (and security implications)

Everything is a file

- uniform access control mechanism: file permissions
- `alrwxrwxrwx`, `a` is append-only, `l` is exclusive use
- system logs are `a-rw-rw-rw-`
  no syslog daemon: programs write directly to log
- mail spool files are `alrw--w--w-`
  to deliver mail, just open mailbox and write a message

Plan 9 principles (and security implications)....................................

Everything is a file

- uniform access control mechanism: file permissions
- `alrwxrwxrwx`, `a` is append-only, `l` is exclusive use
- system logs are `a-rw-rw-rw-`
  no syslog daemon: programs write directly to log
- mail spool files are `alrw--w--w-`
  to deliver mail, just open mailbox and write a message

All file systems are presented to kernel using one file protocol: 9P

- authentication is only in one place: 9P

Plan 9 principles (and security implications)

Everything is a file

- uniform access control mechanism: file permissions
- `alrwxrwxrwx`, `a` is append-only, `l` is exclusive use
- system logs are `a-rw-rw-rw-`
  no syslog daemon: programs write directly to log
- mail spool files are `alrw--w--w-`
  to deliver mail, just open mailbox and write a message

All file systems are presented to kernel using one file protocol: 9P

- authentication is only in one place: 9P

Each process has its own private, malleable name space

- easy to remove resources from a process: `unmount /net`.
- easy to keep a process from adding resources:
  `rfork(RFNOMNT)` disallows mounting new resources into the name space
- easy sandboxing

Plan 9 host owner

Local machine resources owned by the *host owner*

- normal user account
- no *a priori* special privileges (not *root*)
- on terminals, the user who booted the terminal
- on CPU servers, a pseudo-user

Problems with Plan 9 authentication (now fixed)...............................

Problems with Plan 9 authentication (now fixed)..............................

One security domain

- can’t be `rsc` in one place and `rcox` in another
- can’t have different passwords

Problems with Plan 9 authentication (now fixed)

One security domain

- can't be `rsc` in one place and `rcox` in another
- can't have different passwords

Auth protocol hard wired into 9P

- fixing an auth bug would require redefining 9P

Problems with Plan 9 authentication (now fixed)..............................

One security domain

- can’t be `rsc` in one place and `rcox` in another
- can’t have different passwords

Auth protocol hard wired into 9P

- fixing an auth bug would require redefining 9P

Auth code in all servers, clients, kernels

- fixing an auth bug would require extensive code changes

Problems with Plan 9 authentication (now fixed)..............................

One security domain

- can’t be `rsc` in one place and `rcox` in another
- can’t have different passwords

Auth protocol hard wired into 9P

- fixing an auth bug would require redefining 9P

Auth code in all servers, clients, kernels

- fixing an auth bug would require extensive code changes

Needham-Schroeder-like shared key authentication

- passwords, DES keys short: vulnerable to eavesdropper dictionary attack

Redesign around an agent ....

Factotum, from the OED:

†a. In L. phrases: Dominus factotum, used for ‘one who controls everything’, a ruler with uncontrolled power; Johannes factotum, a Jack of all trades, a would-be universal genius. Also fig.

†b. One who meddles with everything, a busybody.

c. In mod. sense: A man of all-work; also, a servant who has the entire management of his master’s affairs.

Redesign around an agent

Factotum, from the OED:

> †a. In L. phrases: Dominus factotum, used for ‘one who controls everything’, a ruler with uncontrolled power; Johannes factotum, a Jack of all trades, a would-be universal genius. Also fig.
>
> †b. One who meddles with everything, a busybody.
>
> c. In mod. sense: A man of all-work; also, a servant who has the entire management of his master’s affairs.

This is Plan 9: `factotum` is a file server

```
% cd /mnt/factotum
% ls -l
-lrw------- gre gre 0 Jul 31 10:14 confirm
--rw------- gre gre 0 Jul 31 10:14 ctl
-lr-------- gre gre 0 Jul 31 10:14 log
-lrw------- gre gre 0 Jul 31 10:14 needkey
--r--r--r-- gre gre 0 Jul 31 10:14 proto
--rw-rw-rw- gre gre 0 Jul 31 10:14 rpc
%
```

Overview...

Overview

Factotum

- holds keys
- uses keys to execute authentication protocols
- host owner’s `factotum` moderates identity changes on that machine
- user can run his own `factotum`; programs use whatever is mounted at `/mnt/factotum`

Overview..

Factotum

- holds keys
- uses keys to execute authentication protocols
- host owner’s `factotum` moderates identity changes on that machine
- user can run his own `factotum`; programs use whatever is mounted at `/mnt/factotum`

Secstore

- provides safe for holding keys
- consulted by `factotum` to retrieve keys

Overview..

Factotum

- holds keys
- uses keys to execute authentication protocols
- host owner’s `factotum` moderates identity changes on that machine
- user can run his own `factotum`; programs use whatever is mounted at `/mnt/factotum`

Secstore

- provides safe for holding keys
- consulted by `factotum` to retrieve keys

Kernel

- allows host owner’s `factotum` to issue identity change capabilities

Example ....

Example

Boot time

- Factotum fetches keys from secstore

```
user[none]: gre
secstore password: ******
STA PIN+SecurID: **********
```

Example

Boot time

- `Factotum` fetches keys from `secstore`

Apop mail client *C* connects to server *S*

- authenticates by proxying messages between factotum and network

$F_S \leftarrow S$			`start proto=apop role=server`
	$C \rightarrow F_C$		`start proto=apop role=client`

Example

Boot time

- `Factotum` fetches keys from `secstore`

Apop mail client *C* connects to server *S*

- authenticates by proxying messages between factotum and network

F_S	$\leftarrow$	S						`start proto=apop role=server`
				C	$\rightarrow$	F_C		`start proto=apop role=client`
F_S	$\rightarrow$	S	$\rightarrow$	C	$\rightarrow$	F_C		`+OK POP3` *challenge*

Example

Boot time

- `Factotum` fetches keys from `secstore`

Apop mail client *C* connects to server *S*

- authenticates by proxying messages between factotum and network

$$
\begin{array}{ccccccc l}
F_S & \leftarrow & S & & & & & \texttt{start proto=apop role=server} \\
 & & & & C & \rightarrow & F_C & \texttt{start proto=apop role=client} \\
 & & & & & & & \\
F_S & \rightarrow & S & \rightarrow & C & \rightarrow & F_C & \texttt{+OK POP3 } \textit{challenge} \\
 & & & & & & & \\
F_S & \leftarrow & S & \leftarrow & C & \leftarrow & F_C & \texttt{APOP gre } \textit{response}
\end{array}
$$

Example

Boot time

- `Factotum` fetches keys from `secstore`

Apop mail client *C* connects to server *S*

- authenticates by proxying messages between factotum and network

F_S	$\leftarrow$	S					`start proto=apop role=server`
				C	$\rightarrow$	F_C	`start proto=apop role=client`
F_S	$\rightarrow$	S	$\rightarrow$	C	$\rightarrow$	F_C	`+OK POP3` *challenge*
F_S	$\leftarrow$	S	$\leftarrow$	C	$\leftarrow$	F_C	`APOP gre` *response*
F_S	$\rightarrow$	S	$\rightarrow$	C	$\rightarrow$	F_C	`+OK welcome`

Example

Boot time

- `Factotum` fetches keys from `secstore`

Apop mail client *C* connects to server *S*

- authenticates by proxying messages between factotum and network

F_S	$\leftarrow$	S					`start proto=apop role=server`
				C	$\rightarrow$	F_C	`start proto=apop role=client`
F_S	$\rightarrow$	S	$\rightarrow$	C	$\rightarrow$	F_C	`+OK POP3` *challenge*
F_S	$\leftarrow$	S	$\leftarrow$	C	$\leftarrow$	F_C	`APOP gre` *response*
F_S	$\rightarrow$	S	$\rightarrow$	C	$\rightarrow$	F_C	`+OK welcome`
F_S	$\leftarrow$	S					`authinfo`
F_S	$\rightarrow$	S					`ok client=gre capability=`*capability*
				C	$\rightarrow$	F_C	`authinfo`
				C	$\leftarrow$	F_C	`ok`

Example

Boot time

- `Factotum` fetches keys from `secstore`

Apop mail client *C* connects to server *S*

- authenticates by proxying messages between factotum and network

Apop server process changes identity to `gre` before proceeding

- passes capability (issued by server factotum) to kernel

Keys

Key is a list of *attribute=value* pairs:

```
key proto=p9sk1 dom=cs.xyz.com user=gre
    !password=xyzzy confirm=yes

key proto=apop server=comcast.net user=gre12345
    !password=boo
```

- The ‘`!`’ prefix means don’t print this value when displaying keys.

Attributes are free-form, but some have meaning to:

- `factotum` itself: `proto`, `confirm`
- the protocols: `password`, `user`, `server`, `dom`
- the user: anything else

Factotum and keys

Keys are added to `factotum` by writing them to the `ctl` file.

```
% cd /mnt/factotum
% cat >ctl
key dom=bell-labs.com proto=p9sk1 user=gre
    !password=’don’’t tell’
key proto=apop server=x.y.com user=gre
    !password=’bite me’
^D
% cat ctl
key dom=bell-labs.com proto=p9sk1 user=gre !password?
key proto=apop server=x.y.com user=gre !password?
%
```

Key patterns

Key patterns are *attribute=value* pairs. The key must be a superset of the pattern.

```
% cat ctl
key dom=bell-labs.com proto=p9sk1 user=gre !password?
key proto=apop server=x.y.com user=gre !password?
% echo ’delkey proto=apop’ >ctl
% cat ctl
key dom=bell-labs.com proto=p9sk1 user=gre !password?
%
```

Secstore

Secure, encrypted file store for small, precious files

- a safe to hold keys

PAK protocol provides password-based access

- hash password to yield auth key
- actively attacking PAK is equivalent to computational Diffie-Hellman

File encryption/decryption performed by client

- hash password another way to yield crypt key
- if server is compromised, attacker has to break the individual files

Secstore and factotum

`Factotum` fetches file named `factotum` from secstore at boot time.

- assumed to hold initial set of keys

```
user[none]: gre
secstore password: ******
STA PIN+SecurID: **********
```

- can reload the secstore key file into factotum at any time

```
secstore –G factotum >/mnt/factotum/ctl
```

- can edit the key file (fetch to ramfs, edit, put back)

User must remember only one password

- can be fairly high entropy
- stored keys can be arbitrarily high entropy

Factotum interface for programs

`Auth_proxy` executes RPCs over `/mnt/factotum/rpc` to proxy a conversation between factotum and a file descriptor.

```
AuthInfo *ai;
AuthGetkey *getkey;

ai = auth_proxy(fd, getkey,
    "proto=p9any role=client");
```

Last argument is a key pattern

- Actually a `printf`-style string:

```
ai = auth_proxy(fd, getkey,
     "proto=p9any role=client server=%s", machine);
```

`AuthInfo` holds

- user name and domain at other end
- nonce keys for the conversation
- possibly a capability to change user id

Identity changes via capabilities

User id changes are managed by capabilities

- string *oldname@newname@random-bytes*
- allows a process running as *oldname* to start running as *newname*
- single use

Host owner's `factotum` informs the kernel of newly issued capabilities by writing their SHA1 hashes to `/dev/caphash`

```
    echo rsc@rob@xyzzy | sha1sum >/dev/caphash
```

`Factotum` hands capability to another process, which then writes it to `/dev/capuse`

```
    echo rsc@rob@xyzzy >/dev/capuse
```

`/dev/caphash` is removed once the host owner's `factotum` starts, so other host owner processes can't use it.

Unprivileged, safe servers

Servers run as `none`, the opposite of a superuser.

- can’t debug any processes
- explicitly excluded from some file systems (e.g., dump)
- (like everyone else,) requires a capability in order to become another user

Bugs in servers are less critical

- on Unix, servers run as `root`: breaking a server gives you full access
- on Plan 9, servers run as `none`: breaking a server gives you hardly any access

Moving Factotum to Unix? ....

Still have source for (almost) everything

- no one group controls all the source
- in long term, would be good to convince owners to go along with you

Factotum can’t be user-level file system.

- RPCs over Unix socket named by environment variable
- lose the use of `cat`, `echo` to manage keys. Need simple replacements.

Factotum must be separate process

- all authentication logic is encapsulated in one place
- buggy clients cannot compromise `factotum`
- only one program runs with special privileges

Factotum must *not* be shared library

- shared libraries share memory space with buggy clients
- shared libraries require clients to run with special privileges

Moving Factotum to Unix?

Nothing Plan 9-specific about `secstore`

Easy to write `/dev/caphash` kernel driver

- goodbye, setuid bit!
- even `/bin/login` and `/bin/su` don’t need to run as `root`

Summary

Everything is a file, so everything has a uniform access control mechanism: file permission bits

`Factotum`, a protocol-agnostic trusted agent, handles both client and server authentication.

`Secstore` provides convenient but secure storage of keys

Clean separation of security and applications

Mostly applicable to Unix

More in paper:

- one line of code to start TLS on a file descriptor
- factotum protected against debuggers, swapping
- 9P auth protocol is now textual metaprotocol to choose real protocol
- links to more information in paper