Design and Implementation of an Alternative to SSH

Index

Introduction

Design and implement an alternative to SSH (prototype)
Implementation language: Go (Golang)
Target platform: GNU/Linux

Telnet

Precursory Works

telnet(1) is old (RFC15 1969, RFC854 1983)
No secure connection (except: TELNETS)
"Go-Telnet"

Berkeley `r`-Commands

Precursory Works

Frequently used Linux commands made into `r`-Commands:

- `login`(1) ⇒ `rlogin`(1)
- `sh`(1)/`bash`(1) ⇒ `rsh`(1)/`rexec`(1)*
- `cp`(1) ⇒ `rcp`(1)
- `who`(1) ⇒ `rwho`(1)
- `stat`(1) ⇒ `rstat`(1)
- `uptime`(1) ⇒ `ruptime`(1)

Useful (especally for scripts), but no secure connection

OpenSSH

Replaces telnet(1) and Berkeley `r`-commands
Secure connection (own protocol)
Plethora of features:

- Remote user login
- Auth via keys
- Port forwarding
- X11-forwarding
- Auth agent connection forwarding (!)
- Compression (used by `rsync`(1))

⋮

Secure Connection

Oh-My-Gosh

Prevent MITM, provide integrity & privacy

TLS 1.3

Server: `openssl(1)` $\rightarrow$ key & X.509 certificate

`crypto/tls`

Encrypted channel

Self signed server certificate: Ignores trust chain

No client certificates (!) $\rightarrow$ Cannot authenticate the connecting client

Authentication via Password

Oh-My-Gosh

`/etc/passwd` (!)
PAM
No Go-package for PAM
Failure in test environment $\rightarrow$ `login`(1)
Failure in same environment using `login`(1)
Too time consuming to switch back
`login`(1) allows root login
Prefetch credentials on client

Authentication via Keys

Authenticate via public key cryptography
Store authorized public keys on server
Authorized keys stored in `/root/.gosh` (plain-text)
 $\rightarrow$ Hash in `~/.gosh/authorized_keys`
 $\rightarrow$ Important for privilege separation

Privilege Separation

Oh-My-Gosh

Shell should run with appropriate permissions (`setuid`(2)/`setgid`(2))

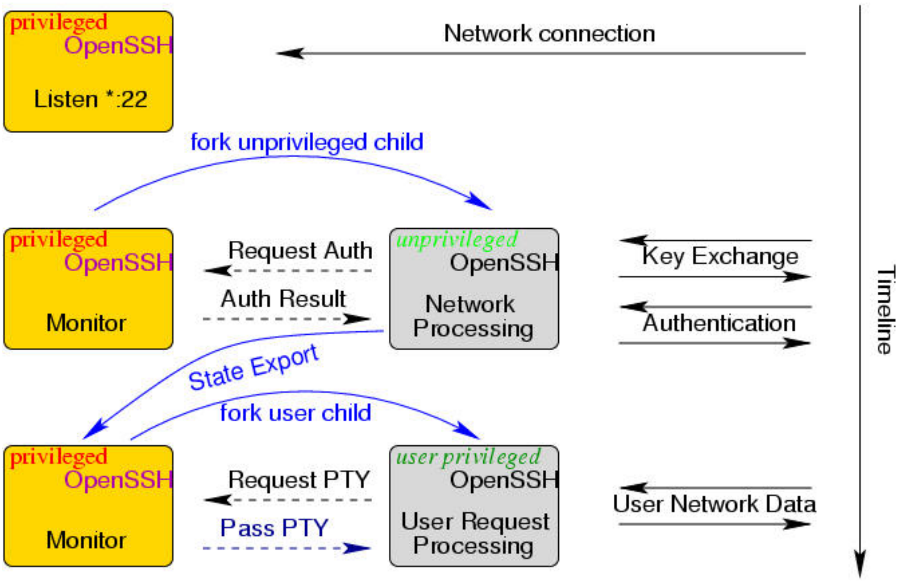
Failure to drop privileges after login (operation not supported)

  Thank you, Go $\rightarrow$ spawn shell with appropriate UID & GID

SSH more sophisticated

Privilege Separation

Oh-My-Gosh



Forking

- Server spawns child to handle connection
- `fork`(2)
- Go: No support for forking
- CGO fork fails
- `syscall.ForkExec`
  → High level connection object gets corrupted
- Create host application
- Transfer `fd` as argument to child
  → Low level socket from `x/sys/unix` (x-package!)
- Prospect: Implement proper privilege separation

Login Accounting

Oh-My-Gosh

Not implemented, **but**

`utmpx` $\rightarrow$ `w/who`

PAM: `pam_open_session(3)/pam_close_session(3)`

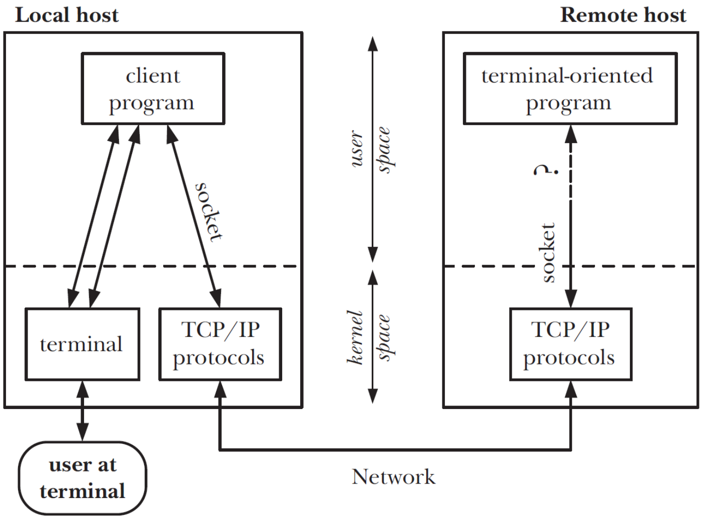
User Data Acquisition

Home directory, shell, UID & GID
Go standard library incomplete (misses shell information)
`/etc/passwd` (!) ⇒ CGO: `getpwnam`(2)/`getpwuid`(2)

Pseudoterminals

Oh-My-Gosh

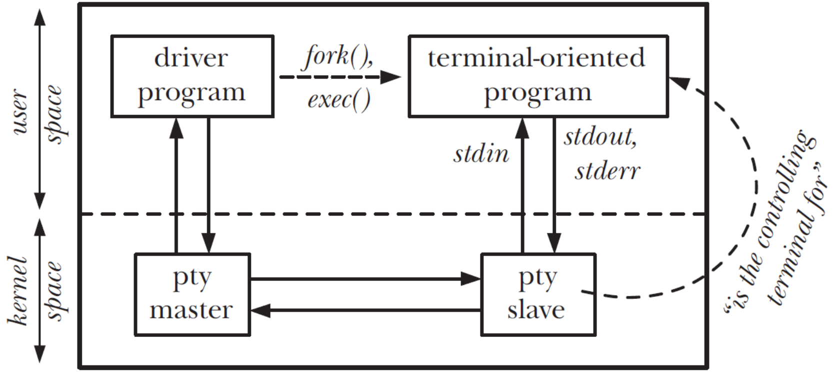
Shells expect to be connected to a TTY



Pseudoterminals

Oh-My-Gosh

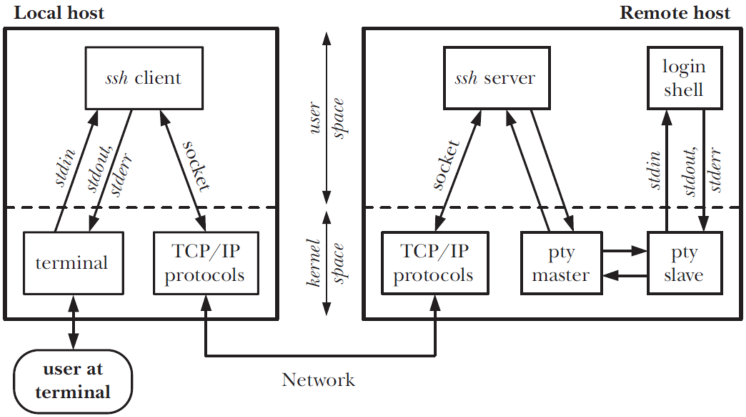
PTY fakes being a TTY



Pseudoterminals

Oh-My-Gosh

Overview



Pseudoterminals

Oh-My-Gosh

`istty`(3) on the connected `fds`
`posix_openpt("/dev/ptmx")`(3) $\rightarrow$ `grant_pt`(3) $\rightarrow$ `unlockpt`(3) $\rightarrow$ `ptsname`(3)
Wrapper function in internal(!) package of the Go standard library
`os/signal/internal/pty`

Starting the Shell

Shell requirements:

- user (UID & GID) & host name
- `TERM` env var (for `ncurses`(3X))
- window resolution (including `SIGWINCH`)
- session leader (controlling terminal)

Transfer of env vars (client $\leftrightarrow$ server)
Continuous transfer of `SIGWINCH` not implemented $\rightarrow$ prospects
Setting `CTTY` flag (for controlling terminal) fails $\rightarrow$ prospects

Terminal Mode

Oh-My-Gosh

Forward all keystrokes without interpretation (client-sside)
`cooked` mode → `raw` mode
x-package (!) `x/crypto/ssh/terminal`

Performance

client $\leftrightarrow$ server $\leftrightarrow$ ptm $\leftrightarrow$ pts $\leftrightarrow$ shell
`/dev/zero` $\rightarrow$ connection (client-side) $\rightarrow$ server $\rightarrow$ `pv -rabtW` $\rightarrow$ `/dev/null`
TLS vs no TLS

Throughput with:	TLS (total)	no TLS (size)
Linux	427MiB/s (25.1GiB)	1177.6MiB/s (69.0GiB)
WSL	69.7MiB/s (4.09GiB)	116MiB/s (6.82GiB)
Linux to WSL (eth*)	85.1MiB/s (4.99GiB)	83.7MiB/s (4.91GiB)

*: Netgear Switch & Cat 5 ethernet cable

Comparison to Telnet

TLS vs plain text
Key auth vs only password auth

Comparison to Berkeley `r`-commands

Only `rlogin`(1) is considered (`rsh`(1))
TLS vs plain text
Key auth vs only password auth

Comparison to OpenSSH

TLS vs own protocol
Privilege separation
Many additional features

Conclusion

Many problems encountered
Many new concepts learned
Mixed feelings

End

Thank you for your attention!