Making the Command-Line Friendly

Joe LaFreniere (lafrenierejm)

Linux Users Group @ UT Dallas

2017-10-07

Outline

1 Requirements
- Story
- Security

2 Key Cache
- ssh-agent
- keychain
- ssh-agent
- ssh-ident

3 Requirements
- Accessibility

4 Password Manager
- pass

5 Requirements
- Usability

6 zsh Autocompletion
- Writing

User Story

1. Boot our Linux machine.
2. Login and start a window manager.
3. Open shell 1 in terminal 1 and SSH into server 1.
4. Open shell 2 in terminal 2 and SSH into server 1.
5. Open a shell 3 in terminal 2 and SSH into server 2.
6. Close the SSH connection in shell 1 and SSH into server 2.

Artifacts

- multiple terminal emulators
- multiple shells within each terminal, x total
  - may be initialized before or after first connection
- one or more SSH connection per shell, y total
- one or more remote servers, z total

Securing Keys

Problem

Solution

Securing Keys

Problem

1. secrets must remain secret

Solution

Securing Keys

Problem

1. secrets must remain secret

Solution

1. encrypt device storage

Securing Keys

Problem

1. secrets must remain secret
2. secrecy during use

Solution

1. encrypt device storage

Securing Keys

Problem

1. secrets must remain secret
2. secrecy during use

Solution

1. encrypt device storage
2. per-key encryption

Securing Keys

Problem

1. secrets must remain secret
2. secrecy during use
3. encryption key per secret

Solution

1. encrypt device storage
2. per-key encryption

Securing Keys

Problem

1. secrets must remain secret
2. secrecy during use
3. encryption key per secret

Solution

1. encrypt device storage
2. per-key encryption
3. per-key passphrase

Securing Keys

Problem

1. secrets must remain secret
2. secrecy during use
3. encryption key per secret
4. brute-force passphrase

Solution

1. encrypt device storage
2. per-key encryption
3. per-key passphrase

Securing Keys

Problem

1. secrets must remain secret
2. secrecy during use
3. encryption key per secret
4. brute-force passphrase

Solution

1. encrypt device storage
2. per-key encryption
3. per-key passphrase
4. long passphrase

Securing Keys

Problem

1. secrets must remain secret
2. secrecy during use
3. encryption key per secret
4. brute-force passphrase
5. remember passphrase

Solution

1. encrypt device storage
2. per-key encryption
3. per-key passphrase
4. long passphrase

Securing Keys

Problem

1. secrets must remain secret
2. secrecy during use
3. encryption key per secret
4. brute-force passphrase
5. remember passphrase

Solution

1. encrypt device storage
2. per-key encryption
3. per-key passphrase
4. long passphrase
5. password manager

Securing Keys

Problem

1. secrets must remain secret
2. secrecy during use
3. encryption key per secret
4. brute-force passphrase
5. remember passphrase
6. one passphrase per key

Solution

1. encrypt device storage
2. per-key encryption
3. per-key passphrase
4. long passphrase
5. password manager

Securing Keys

Problem

1. secrets must remain secret
2. secrecy during use
3. encryption key per secret
4. brute-force passphrase
5. remember passphrase
6. one passphrase per key

Solution

1. encrypt device storage
2. per-key encryption
3. per-key passphrase
4. long passphrase
5. password manager
6. cache plaintext keys

Securing Keys

Problem

1. secrets must remain secret
2. secrecy during use
3. encryption key per secret
4. brute-force passphrase
5. remember passphrase
6. one passphrase per key
7. plaintext is insecure

Solution

1. encrypt device storage
2. per-key encryption
3. per-key passphrase
4. long passphrase
5. password manager
6. cache plaintext keys

Securing Keys

Problem

1. secrets must remain secret
2. secrecy during use
3. encryption key per secret
4. brute-force passphrase
5. remember passphrase
6. one passphrase per key
7. plaintext is insecure

Solution

1. encrypt device storage
2. per-key encryption
3. per-key passphrase
4. long passphrase
5. password manager
6. cache plaintext keys
7. only cache keys in use

Key Cache

Key Cache

1. allow for single unlock per key
   - shells — terminal emulators, TTYs
   - graphical applications

Key Cache

1. allow for single unlock per key
   - shells — terminal emulators, TTYs
   - graphical applications
2. cache keys only as they are used

Key Cache

1. allow for single unlock per key
   - shells — terminal emulators, TTYs
   - graphical applications
2. cache keys only as they are used
3. be stateless WRT initialization order

Key Cache

1. allow for single unlock per key
   - shells — terminal emulators, TTYs
   - graphical applications
2. cache keys only as they are used
3. be stateless WRT initialization order
4. provide mechanism for clearing cache
   - manually
   - on timeout from cache
   - on timeout from last use

Key Cache

1. allow for single unlock per key
   - shells — terminal emulators, TTYs
   - graphical applications
2. cache keys only as they are used
3. be stateless WRT initialization order
4. provide mechanism for clearing cache
   - manually
   - on timeout from cache
   - on timeout from last use
5. include hooks for password manager

ssh—agent

ssh–agent

- packaged with modern releases of OpenSSH

ssh–agent

- packaged with modern releases of OpenSSH
- keys only accessible from owning user
  - vulnerable to malicious root

ssh−agent

- packaged with modern releases of OpenSSH
- keys only accessible from owning user
  - vulnerable to malicious root
- uses environment variables

ssh–agent

- packaged with modern releases of OpenSSH
- keys only accessible from owning user
  - vulnerable to malicious root
- uses environment variables
- separate instance per shell

ssh–agent

- packaged with modern releases of OpenSSH
- keys only accessible from owning user
  - vulnerable to malicious root
- uses environment variables
- separate instance per shell
- not convenient for other applications

Running agent

interactive shell

```
eval $(ssh-agent)
```

Problems

Running agent

interactive shell

```
eval $(ssh-agent)
```

Problems

It's dumb

Running agent from RC

$HOME/.zshrc

```
if [ -z "$SSH_AUTH_SOCK" ]; then
    eval $(ssh-agent -s)
    ssh-add
fi
```

Problems

Running agent from RC

$HOME/.zshrc

```
if [ -z "$SSH_AUTH_SOCK" ]; then
    eval $(ssh-agent -s)
    ssh-add
fi
```

Problems

- separate agent for each shell
- keys only cached within that session

keychain

- front-end to `ssh-agent`
- long-running process rather than per-login
- list secret keys in configuration
- unlock all keys upfront

homepage: funtoo.org/Keychain
git repo: github.com/funtoo/keychain

Running keychain

$HOME/.zshrc (broken by comments)

```
eval $(keychain
       --eval          # print for evaluation
       --agents ssh    # manage SSH agent
       --quiet         # suppress most output
       id_ed25519      # keys to manage
    )
```

Running keychain

$HOME/.zshrc

```
eval $(keychain\
       --eval\
       --agents ssh\
       --quiet\
       id_ed25519\
    )
```

Back to raw agent

$HOME/.zshrc

```
SSH_AUTH_SOCK="${HOME}/.ssh/auth-socket"
function start_ssh_agent {
    ssh-agent | sed 's/^echo/#echo/' > "${SSH_AUTH_SOCK}"
    chmod 600 "${SSH_AUTH_SOCK}"
    source "${SSH_AUTH_SOCK}"
}
if [ -f "${SSH_AUTH_SOCK}" ]; then
    source "${SSH_AUTH_SOCK}" >/dev/null
    ps --pid "${SSH_AGENT_PID}" >/dev/null ||
        start_ssh_agent
else
    start_ssh_agent
fi
```

Back to raw agent

Remaining Problems

- all keys in one agent
  - forwarding that agent exposes all keys
- requires cleanup of $SSH_AUTH_SOCK and \tmp

To-Do

github/ccontavlli/ssh-ident claims to solve problems by storing each key in its own agent.

- written in Python
- open PR for BASH autocompletion
- requires replacing `ssh` binary for dependent applications

Managing Key Passphrases

1. generation
   - length
   - symbols
2. encryption
3. synchronization across devices

Managing Key Passphrases

1. generation
   - length
   - symbols
2. encryption
3. synchronization across devices
4. command-line interface
   - access from other scripts/apps
   - autocompletion from shell

pass (passwordstore.org)

"The standard UNIX password manager"

- generation runs `tr` on `/dev/urandom`
  - user sets length, symbols
- git for synchronization
- CLI-first
  - autocompletion for zsh, bash, fish

homepage: passwordstore.org
git repo: git.xz2c4.com/password-store

Shell Autocompletion

General

1. executables from $PATH
2. arguments
   - limit paths by type (`find -type`)
3. executables' options
4. option arguments

Shell Autocompletion

General

1. executables from $PATH
2. arguments
   - limit paths by type (`find -type`)
3. executables' options
4. option arguments

Password Manager

1. arguments
   - generate secret
   - find filename
   - copy secret
2. directories, subdirectories
3. filename

Shell Autocompletion

General

1. executables from $PATH
2. arguments
   - limit paths by type (`find -type`)
3. executables' options
4. option arguments

SSH, SCP, `rsync`...

1. hostname or IP address
2. remote path
   - prompt if key not cached
3. local path

Getting Started

- primary source is man `zshcompsys`
  - "zsh completion system"
- completions loaded from `$fpath`
- filename must start with an underscore

1. create a directory to house custom/prototype completions
2. add that directory to `$fpath` from `.zshrc`
3. initialize completion in `.zshrc`

Using Autocompletions

$HOME/.zshrc

```
zstyle ':completion:*' verbose yes
zstyle ':completion:*:descriptions' format '%B
    %d%b'
zstyle ':completion:*:messages' format '%d'
zstyle ':completion:*' group-name
autoload -U compinit
compinit
```