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
Making the Command-Line Friendly

Requirements

Story

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

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Secrets must remain secret</td>
<td>1. Encrypt device storage</td>
</tr>
<tr>
<td>2. Secrecy during use</td>
<td>2. Per-key encryption</td>
</tr>
<tr>
<td>3. Encryption key per secret</td>
<td>3. Per-key passphrase</td>
</tr>
<tr>
<td>5. Remember passphrase</td>
<td>5. Password manager</td>
</tr>
<tr>
<td>6. One passphrase per key</td>
<td>6. Cache plaintext keys</td>
</tr>
<tr>
<td>7. Plaintext is insecure</td>
<td>7. Only cache keys in use</td>
</tr>
</tbody>
</table>
## Securing Keys

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 secrets must remain secret</td>
<td></td>
</tr>
</tbody>
</table>
## Securing Keys

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 secrets must remain secret</td>
<td>1 encrypt device storage</td>
</tr>
</tbody>
</table>
Securing Keys

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 secrets must remain secret</td>
<td>1 encrypt device storage</td>
</tr>
<tr>
<td>2 secrecy during use</td>
<td></td>
</tr>
</tbody>
</table>
## Securing Keys

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. secrets must remain secret</td>
<td>1. encrypt device storage</td>
</tr>
<tr>
<td>2. secrecy during use</td>
<td>2. per-key encryption</td>
</tr>
</tbody>
</table>
Securing Keys

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 secrets must remain secret</td>
<td>1 encrypt device storage</td>
</tr>
<tr>
<td>2 secrecy during use</td>
<td>2 per-key encryption</td>
</tr>
<tr>
<td>3 encryption key per secret</td>
<td></td>
</tr>
</tbody>
</table>
## Securing Keys

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. secrets must remain secret</td>
<td>1. encrypt device storage</td>
</tr>
<tr>
<td>2. secrecy during use</td>
<td>2. per-key encryption</td>
</tr>
<tr>
<td>3. encryption key per secret</td>
<td>3. per-key passphrase</td>
</tr>
</tbody>
</table>

- Problem 1: Secrets must remain secret
- Problem 2: Secrecy during use
- Problem 3: Encryption key per secret

- Solution 1: Encrypt device storage
- Solution 2: Per-key encryption
- Solution 3: Per-key passphrase
## Securing Keys

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 secrets must remain secret</td>
<td>1 encrypt device storage</td>
</tr>
<tr>
<td>2 secrecy during use</td>
<td>2 per-key encryption</td>
</tr>
<tr>
<td>3 encryption key per secret</td>
<td>3 per-key passphrase</td>
</tr>
<tr>
<td>4 brute-force passphrase</td>
<td></td>
</tr>
</tbody>
</table>
## Securing Keys

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. secrets must remain secret</td>
<td>1. encrypt device storage</td>
</tr>
<tr>
<td>2. secrecy during use</td>
<td>2. per-key encryption</td>
</tr>
<tr>
<td>3. encryption key per secret</td>
<td>3. per-key passphrase</td>
</tr>
<tr>
<td>4. brute-force passphrase</td>
<td>4. long passphrase</td>
</tr>
</tbody>
</table>
## Securing Keys

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. secrets must remain secret</td>
<td>1. encrypt device storage</td>
</tr>
<tr>
<td>2. secrecy during use</td>
<td>2. per-key encryption</td>
</tr>
<tr>
<td>3. encryption key per secret</td>
<td>3. per-key passphrase</td>
</tr>
<tr>
<td>4. brute-force passphrase</td>
<td>4. long passphrase</td>
</tr>
<tr>
<td>5. remember passphrase</td>
<td></td>
</tr>
</tbody>
</table>
## Securing Keys

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. secrets must remain secret</td>
<td>1. encrypt device storage</td>
</tr>
<tr>
<td>2. secrecy during use</td>
<td>2. per-key encryption</td>
</tr>
<tr>
<td>3. encryption key per secret</td>
<td>3. per-key passphrase</td>
</tr>
<tr>
<td>4. brute-force passphrase</td>
<td>4. long passphrase</td>
</tr>
<tr>
<td>5. remember passphrase</td>
<td>5. password manager</td>
</tr>
</tbody>
</table>
Making the Command-Line Friendly

Requirements

Security

## Securing Keys

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. secrets must remain secret</td>
<td>1. encrypt device storage</td>
</tr>
<tr>
<td>2. secrecy during use</td>
<td>2. per-key encryption</td>
</tr>
<tr>
<td>3. encryption key per secret</td>
<td>3. per-key passphrase</td>
</tr>
<tr>
<td>4. brute-force passphrase</td>
<td>4. long passphrase</td>
</tr>
<tr>
<td>5. remember passphrase</td>
<td>5. password manager</td>
</tr>
<tr>
<td>6. one passphrase per key</td>
<td></td>
</tr>
</tbody>
</table>
## Securing Keys

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. secrets must remain secret</td>
<td>1. encrypt device storage</td>
</tr>
<tr>
<td>2. secrecy during use</td>
<td>2. per-key encryption</td>
</tr>
<tr>
<td>3. encryption key per secret</td>
<td>3. per-key passphrase</td>
</tr>
<tr>
<td>4. brute-force passphrase</td>
<td>4. long passphrase</td>
</tr>
<tr>
<td>5. remember passphrase</td>
<td>5. password manager</td>
</tr>
<tr>
<td>6. one passphrase per key</td>
<td>6. cache plaintext keys</td>
</tr>
</tbody>
</table>
## Securing Keys

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. secrets must remain secret</td>
<td>1. encrypt device storage</td>
</tr>
<tr>
<td>2. secrecy during use</td>
<td>2. per-key encryption</td>
</tr>
<tr>
<td>3. encryption key per secret</td>
<td>3. per-key passphrase</td>
</tr>
<tr>
<td>4. brute-force passphrase</td>
<td>4. long passphrase</td>
</tr>
<tr>
<td>5. remember passphrase</td>
<td>5. password manager</td>
</tr>
<tr>
<td>6. one passphrase per key</td>
<td>6. cache plaintext keys</td>
</tr>
<tr>
<td>7. plaintext is insecure</td>
<td></td>
</tr>
</tbody>
</table>
## Securing Keys

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 secrets must remain secret</td>
<td>1 encrypt device storage</td>
</tr>
<tr>
<td>2 secrecy during use</td>
<td>2 per-key encryption</td>
</tr>
<tr>
<td>3 encryption key per secret</td>
<td>3 per-key passphrase</td>
</tr>
<tr>
<td>4 brute-force passphrase</td>
<td>4 long passphrase</td>
</tr>
<tr>
<td>5 remember passphrase</td>
<td>5 password manager</td>
</tr>
<tr>
<td>6 one passphrase per key</td>
<td>6 cache plaintext keys</td>
</tr>
<tr>
<td>7 plaintext is insecure</td>
<td>7 only cache keys in use</td>
</tr>
</tbody>
</table>
Key Cache
Making the Command-Line Friendly

Requirements

Security

Key Cache

1. allow for single unlock per key
   - shells — terminal emulators, TTYs
   - graphical applications
Making the Command-Line Friendly

Security

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
Making the Command-Line Friendly

Key Cache

ssh-agent

Packaged with modern releases of OpenSSH, keys are only accessible from the owning user, making it vulnerable to malicious root. It uses environment variables, separate instance per shell, which is not convenient for other applications.
Making the Command-Line Friendly

Key Cache

ssh-agent

- packaged with modern releases of OpenSSH
Making the Command-Line Friendly

Key Cache

- `ssh-agent`

- `ssh-agent`

- packaged with modern releases of OpenSSH
- keys only accessible from owning user
  - vulnerable to malicious root
Making the Command-Line Friendly

- Key Cache
  - `ssh-agent`

**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
Making the Command-Line Friendly

Key Cache

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
Making the Command-Line Friendly

Key Cache

```
ssh-agent
```

Running agent

<table>
<thead>
<tr>
<th>interactive shell</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>eval $(ssh-agent)</code></td>
</tr>
</tbody>
</table>

Problems
Running agent

interactive shell

`eval $(ssh-agent)`

Problems

It’s dumb
Running agent from RC

$HOME/.zshrc

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

Problems
Running agent from RC

$HOME/.zshrc

```bash
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
Making the Command-Line Friendly

Key Cache

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
)

Making the Command-Line Friendly

Key Cache

keychain

Running keychain

$HOME/.zshrc

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

Making the Command-Line Friendly

Key Cache

ssh -agent

Back to raw agent

$HOME/.zshrc

```bash
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
```
Making the Command-Line Friendly

Key Cache

ssh -agent

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
Making the Command-Line Friendly

Requirements

Usability

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