Guardian Agent

Secure SSH Agent Forwarding

Dima Kogan, Henri Stern, David Mazières, Keith Winstein

2nd Hebrew University Networking Summer, June 2017

SSH

- SSH (secure shell) provides a secure channel over an insecure network:
  - Remote login/command execution
  - File transfer
  - Port forwarding/tunneling

Limitations

- Connectivity
  - Does not support roaming, sleep/resume
  - Poor UX on slow connections
  - › Addressed in Mosh [WB12]
- Host Key Management - very “local”
  - ‘Trust on first use’ of host keys
  - No global revocation of compromised keys
- **User key management**
  - Key deployment
  - Policy support
  - Auditing key usage
  - **Credential delegation**
  - › This work

Basic Setting

- User ssh-connects to a ‘partially trusted’ (intermediary) host
- User wants to initiate another ssh connection from that intermediary

Goals

- **Security** – use of private key should be tied to ***<client, server, command>***
- **Transparency** – ability to audit all uses of private key
- **Simple key management –**
  - Avoid proliferation of private key
  - Avoid multiple keys
- **“Transport layer friendliness”**
- **Compatibility** with existing server implementations

Existing Solutions

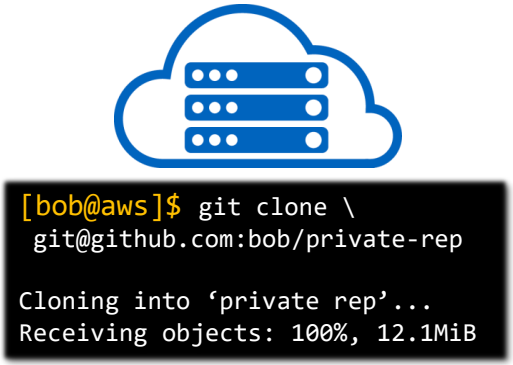
- Copy private key
- If intermediary is compromised
  - Unlimited access to server
  - Often, unlimited access to other services

Existing Solutions

- Fine-grained key management, e.g.,
  - one key-pair per <client machine, server>
  - short-lived keys, frequent revocation
  - Disadvantages:
    - Fine-grained key management
    - No audit-trail of key usage
    - Precision of control might still be insufficient
- Tunneling the entire connection through the trusted host
  - › Disadvantage: doubles/triples the amount of traffic, limited bandwidth

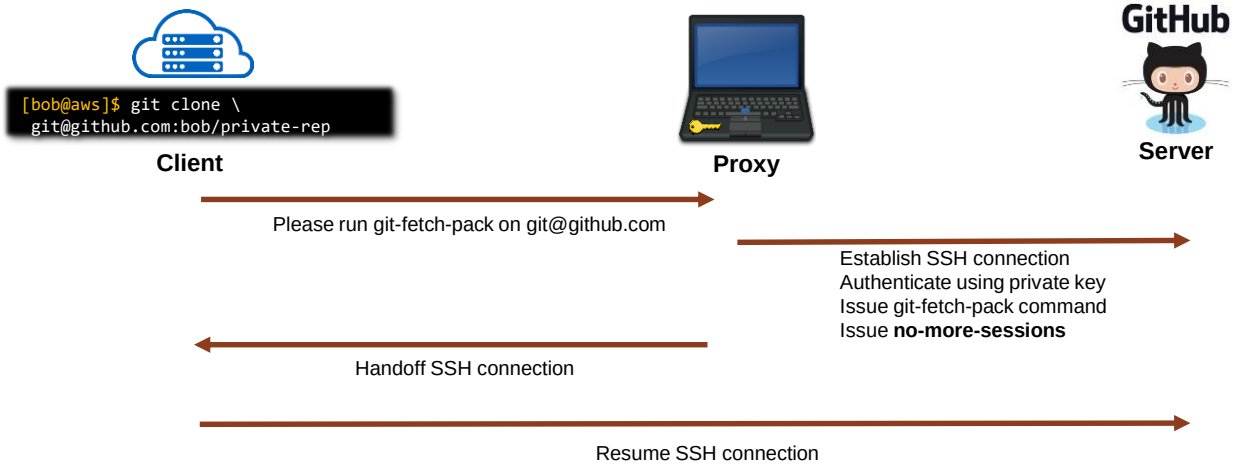
Existing solutions

- SSH agent forwarding
- The protocol does not authenticate the server, the client or the command to the agent
- A malicious intermediary:
  - can “trick” the agent to authenticate to any server
  - gains unrestricted access to the server



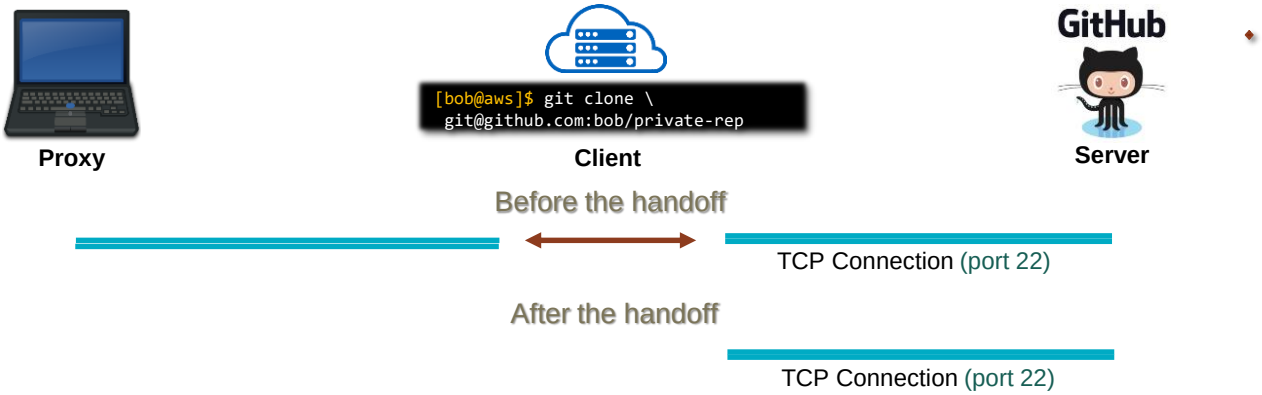
Our approach - SSH authentication proxy

- Client connects to proxy and **issues a connection request**
- Proxy connects to the server and **authenticates** using the local keys
- Proxy **hands off** the established connection to the client



Transport layer

- Constraints:
  - Handoff must be transparent to server
  - Proxy might not have direct connectivity with the server
- Solution:
  - Client establishes a TCP connection to the server
  - Client relays this TCP connection to the proxy



SSH Handoff

- The SSH state consists of
  - Session ID
  - Sequence numbers
  - Crypto state – session key, derived keys (encryption, mac), negotiated algorithms, cipher states (counters, IVs)…
  - SSH connection protocol – channel ids, window sizes…

Messy Details

SSH Key Re-Exchange

9. Key Re-Exchange

... Either party MAY initiate the re-exchange...It is permissible to change some or all of the algorithms during the re-exchange. Host keys can also change. All keys and initialization vectors are recomputed after the exchange. Compression and encryption contexts are reset.

SSH Handoff via Key Re-Exchange

```
[bob@aws]$ git clone \
 git@github.com:bob/private-rep
```

Client

Proxy

GitHub

Server

Before the handoff

SSH State 1
(Alg_1 , $Keys_1$, $SessionID_1$, $SeqNum_1$)

SSH State 2
(Alg_2, $Keys_2$, $SessionID_2$, $SeqNum_2$)

Handoff

Key Re-Exchange

Update to $SessionID_2$, $SeqNum_2$,

SSH State 3
(Alg_3 , $Keys_3$, $SessionID_2$, $SeqNum_2$)

Applications

- SSH agent guard

- Auditing - contextual private key usage trail
- Shared key repository

Questions