Public-key sudo

Matthew Burnside, Mack Lu, Angelos Keromytis
Columbia University

LISA ’08
12 November 2008
Motivation

$ ssh medusa
$ sudo ls
Password:
Overview

- Sudo
- SSH
- Authentication protocols
- SudoPK implementation
- Analysis
Sudo

- Execute a command as another user.
  
  $ sudo ls

- By default, prompts for password

- On OpenBSD, also supports:
  
  - S/Key, Kerberos, Radius, etc.
SSH

• Secure remote shell

    $ ssh bob@medusa

• By default, prompts for password.

• On OpenBSD, also supports:

    • S/Key, Kerberos, Radius, public keys, etc.
SSH protocols

- Three layers
  - Transport [SSH-TRANS]
  - User authentication [SSH-USERSAUTH]
  - Connection [SSH-CONNECT]
Transport protocol

- Server host authentication
- Key exchange (Diffie-Hellman)
- Provides a confidential channel
  - Encryption
  - Integrity
User authentication

- Identifies the client to the server
- Required protocols:
  - "password"
  - "publickey"
## Password authentication

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>SSH_MSG_USERAUTH_REQUEST</td>
</tr>
<tr>
<td>string</td>
<td>user name</td>
</tr>
<tr>
<td>string</td>
<td>service name</td>
</tr>
<tr>
<td>string</td>
<td>&quot;password&quot;</td>
</tr>
<tr>
<td>boolean</td>
<td>FALSE</td>
</tr>
<tr>
<td>string</td>
<td>plaintext password</td>
</tr>
</tbody>
</table>
Password authentication

byte

SSH_MSG_USERAUTH_SUCCESS
Password authentication

• Easy! But...
• Susceptible to brute force
• What if the server is compromised?
Public-key authentication

- Key distribution:
  - Generate \(<\text{id\_rsa}_s, \text{id\_rsa\_pub}\>
  - \text{id\_rsa} \rightarrow \text{client:~/.ssh/id\_rsa}
  - \text{id\_rsa\_pub} \rightarrow \text{server:~/.ssh/authorized\_keys}
Public-key authentication

byte  SSH_MSG_USERAUTH_REQUEST
string user name
string service name
string "publickey"
boolean FALSE
string public key algorithm name
string public key blob (certs)
Public-key authentication

byte        SSH_MSG_USERAUTH_PK_OK
string      public key algorithm name from the request
string      public key blob from the request
Public-key authentication

byte \(\rightarrow\) SSH_MSG_USERAUTH_REQUEST
string user name
string service name
string "publickey"
boolean TRUE
string public key algorithm name
string public key
string signature
Public-key authentication

- More difficult to set up. (Not default!)
- Requires S on every connection
- No password or private key sent to remote host
Public-key + ssh-agent

- ssh-agent manages your private keys
- Prompt for password only once
Public-key + ssh-agent
Public-key + ssh-agent
Public-key + ssh-agent

byte SSH_MSG_USERAUTH_REQUEST
string "publickey"
...
Public-key + ssh-agent

 byte ... SSH_MSG_USERAUTH_PK_OK
Public-key + ssh-agent

byte
...

SSH_MSG_USERAUTH_PK_OK
Public-key + ssh-agent

byte      SSH_MSG_USERAUTH_REQUEST
string    signature
Public-key + ssh-agent

• What about subsequent outbound connections?
ssh-agent forwarding

- Create chain to forward authentication requests back to originating agent

- SSH_AUTH_SOCK
ssh-agent forwarding
ssh-agent forwarding
ssh-agent forwarding

byte      SSH_MSG_USERAUTH_REQUEST
string    "publickey"
...
ssh-agent forwarding

byte

SSH_MSG_USERAUTH_PK_OK

...
ssh-agent forwarding

byte      SSH_MSG_USERAUTH_REQUEST
string    signature
ssh-agent forwarding

- Added convenience.
- Private key and password never appear on the wire or at the remote host.
- But - agent hijacking!
SSH public key authentication

• Goal
  • Build a generic interface to SSH-USERAUTH

• Insight
  • Use the agent-forwarding tunnel to avoid re-implementing SSH components
SudoPK core: login_pubkey

- BSD authentication module
- Presents a `bsdauth` API to the `SSH_AUTH_SOCK`

- Easily portable to PAM
login__pubkey

sudo

login__pubkey
ac = ssh_get_authentication_connection();

for (…) {
   client_key = ssh_get_next_identity(ac, &comment, version);
   if (!user_key_allowed(pw, client_key)) {
      // REJECT
   }

   ssh_agent_sign(ac, client_key, &signature, &slen, buf, BUFLEN);
   if (key_verify(client_key, signature, slen, buf, BUFLEN)) {
      // SUCCESS
   } else {
      // FAIL
   }
}
login_pubkey protocol

SSH_AUTH_SOCK → login_pubkey → sudo

get identities

{pubkey1, pubkey2, ...}

Key allowed?

agent_sign

signature

Verify

sudo ls
$ ssh medusa
$ sudo -a pubkey ls
...
Brief security analysis

- `login_pubkey` provides ease-of-use layer on top of existing agent forwarding

- No worse than plain agent forwarding
Agent hijacking

- **SSH_AUTH_SOCK** is a tunnel for signing requests

- Protect by making socket hard to find
  - Randomly chosen name, restricted file permissions

- Root can still get it
Local confirmation

- `ssh-add -c .ssh/id_rsa`
- Agent requests password on every signature
Local confirmation

• Present message contents to user before signing.

• SSHD should verify and confirm contents. RFC does not enforce this!

• SudoPK package includes ssh-add and ssh-agent patches
Conclusion

• Thanks!

• Code:

  http://www.cs.columbia.edu/~mb/code/sudopk