Utilizing Security Features in SSH

@masonegger

Mason Egger
Developer Advocate

@masonegger

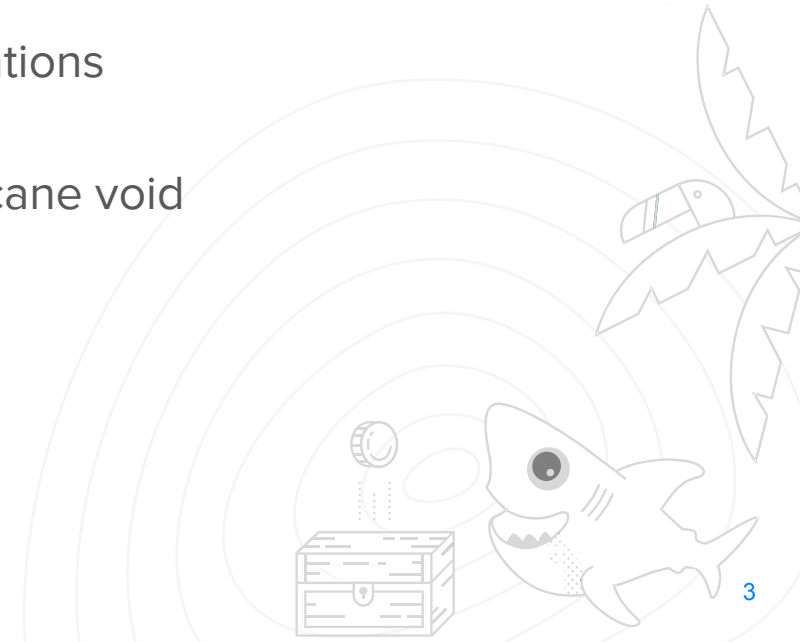
https://mason.dev

mason@do.co

https://twitch.tv/codingwithmason

Goals of the Webinar

- Introduce you to SSH and OpenSSH
- Share some commonly overlooked SSH security features
- Demonstrate some useful SSH applications
- Dive a little to deep(?) into the SSH arcane void

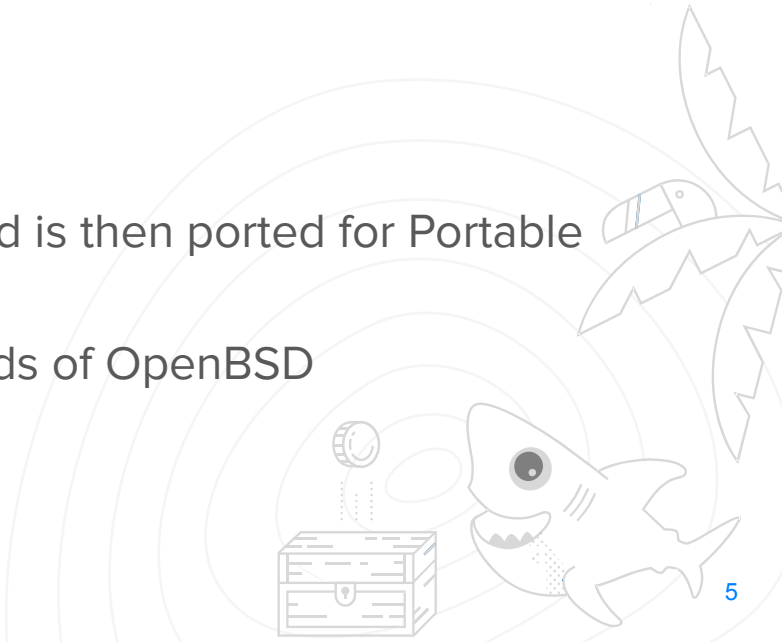


What is SSH

- Secure shell
- Protocol for creating encrypted communication channels between two networked hosts
- Created by Tatu Yölnen in 1995 to replace insecure protocols such as telnet, RSH, and rlogin

What is OpenSSH

- The standard tool for remote management of *Nix systems from servers, to embedded devices, to network devices
- Maintained by the OpenSSH community
- Released in 1999
- Two version
  - OpenBSD
  - Portable OpenSSH
- Development happens in OpenBSD and is then ported for Portable OpenSSH
- Held to the same high security standards of OpenBSD

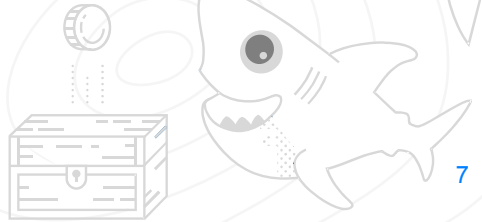


The Pieces

- SSH Server
  - Listens on the network for incoming SSH requests, authenticates them and provides a terminal
- SSH Clients
  - Used to connect to your remote device
    - Popular Clients
      - Windows - Putty, ssh(1)
      - *Nix like systems - ssh(1)
- Protocol Versions
  - Version 2 - Always use
  - Version 1 - Old, barely more secure than unencrypted telnet, may still be found on old embedded devices

A Brief Intro to Encryption

- *Encryption* transforms readable *plaintext* into unreadable *ciphertext* that people without the *key* cannot understand. *Decryption* is the reverse of this process.
- A *key* is text (can include numbers, letters, symbols, etc) used to encrypt messages. Keys are usually chosen by the user or randomly generated.
- *Symmetric algorithms* use the same key for both encryption and decryption. Think *substitution cipher*, where we change 1 letter to mean another letter
- *Asymmetric algorithms* use a different key for encryption and decryption.



Generating Keys

ssh-keygen

usage: ssh-keygen [-q] [-b bits] [-C comment] [-f output_keyfile] [-m format]
[-t dsa | ecdsa | ecdsa-sk | ed25519 | ed25519-sk | rsa]
[-N new_passphrase] [-O option] [-w provider]

- By default generates an RSA 2048 bit key
- *ssh-keygen -t rsa -b 4096 -C mason@do.co*
- Which key is most secure? Depends on what time of the day you ask
  - *dsa* is no longer recommended
  - *rsa* is starting to show its age. Larger key size the better
  - *Ecdsa* new Digital Standard Algorithm standardized by US Government. Use 521 bits
  - *ed25519* new algorithm added, support not universal yet.

Quick *sshd* Security Wins

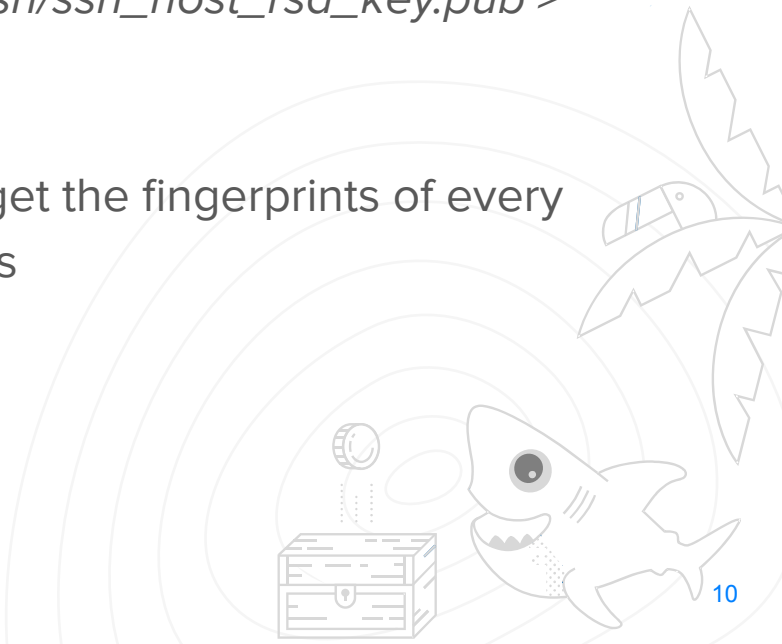
- Usually found in */etc/ssh/sshd_config*

- Disable Root SSH - *PermitRootLogin no*
- No Password Authentication - *PasswordAuthentication no*
- Disable X11 Forwarding if you don't need it - *X11Forwarding no*

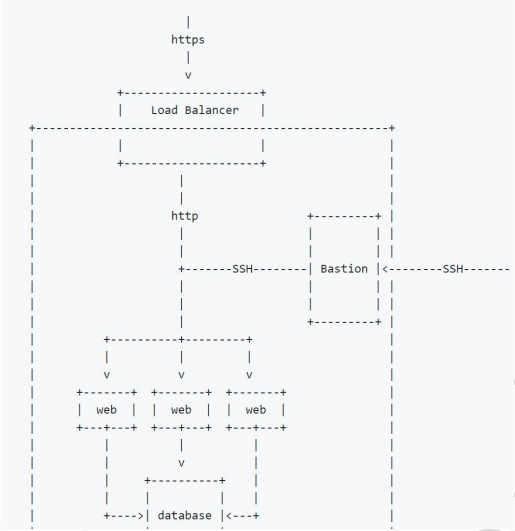
Verify Host Fingerprints

- When you first login you will be presented with a key fingerprint. Verify that this is actually the fingerprint of the server key

- On the server, run *ssh-keygen -lf /etc/ssh/ssh_host_rsa_key.pub > $HOME/fingerprints.txt* from a console
- Verify the fingerprints match
- You may want to to automate this and get the fingerprints of every server and distribute them to your users



SSH Agent Forwarding



- Scenario:
  - Have Droplets behind a load balancer that are not accessible to the public internet
  - Use a bastion host to jump in to private network
  - Don't want to have my ssh key on bastion host

- Run *ssh-agent* on your local machine to turn on
- SSH to host forwarding the agent *ssh -A mason@sammy.shark.codes*
- Now my identity is forwarded through the agent to the bastion and I can access the droples
- Warning: *ssh-agent* keeps the key in memory, so if the bastion is compromised your key could be pulled out of memory

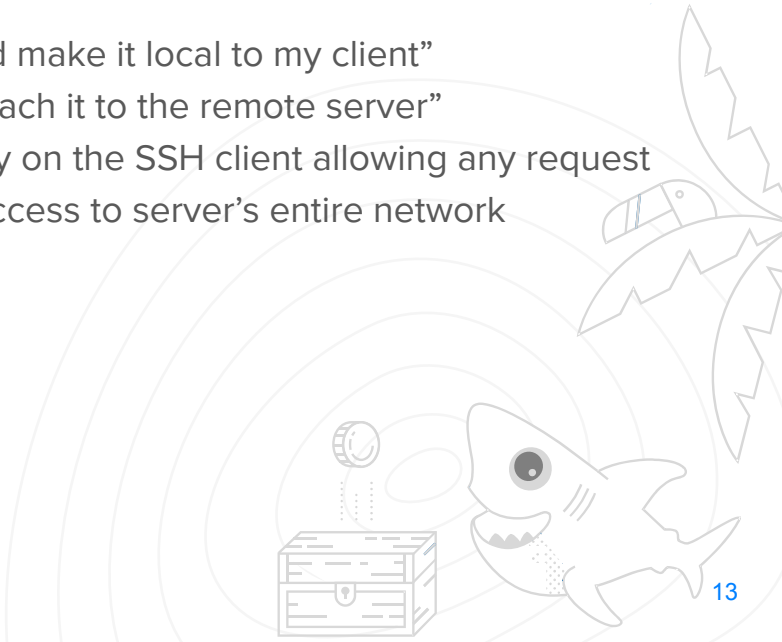
Setting up 2FA with SSH

- Two Factor Authentication is possible with SSH and PAM
- PAM - Pluggable Authentication Modules

- https://www.digitalocean.com/community/tutorials/how-to-set-up-multi-factor-authentication-for-ssh-on-ubuntu-20-04

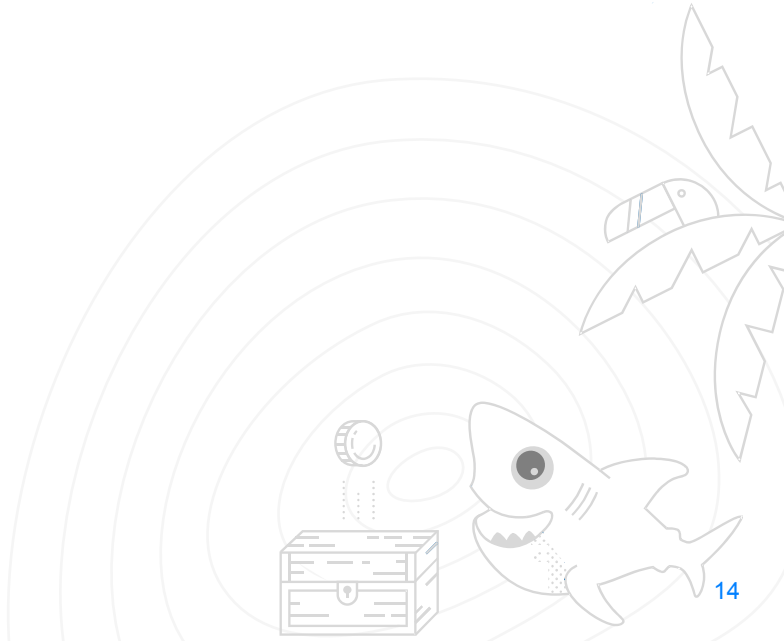
Port Forwarding

- SSH can serve as a wrapper around arbitrary TCP traffic to create a secure way of accessing unencrypted services such as POP3, IMAP, or HTTP.
- Types of Port Forwarding
  - *Local* - “Take this port on the SSH server and make it local to my client”
  - *Remote* - “Take this port on my client and attach it to the remote server”
  - *Dynamic* - Essentially creates a SOCKS proxy on the SSH client allowing any request to the proxy out through the server, giving access to server’s entire network



Local Port Forwarding Example

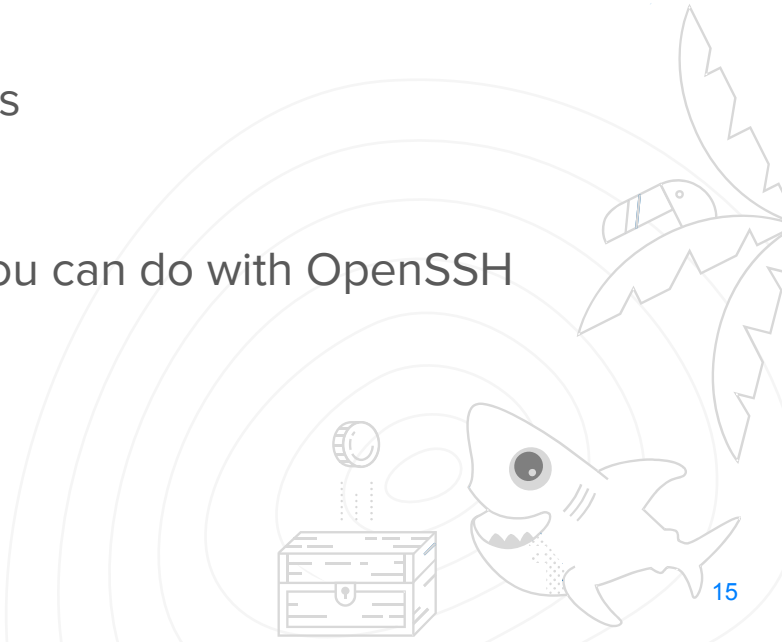
- ssh -L 8080:port.egger.codes:80 root@port.egger.codes



OpenSSH based VPN

- OpenSSH supports building generic tunnels that can pass all traffic and protocols, not just TCP
  - Not supported by PuTTY

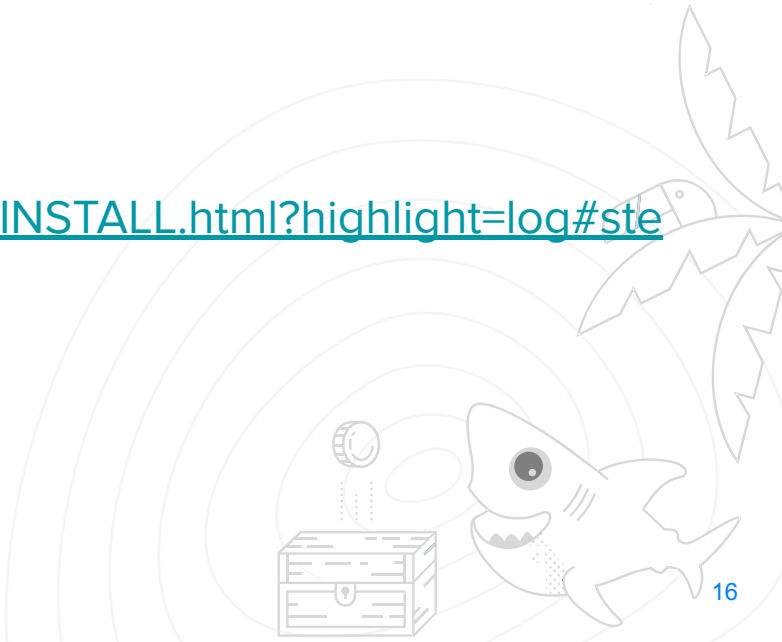
- ….this isn’t the greatest idea
- When a TCP packet is lost, it retransmits
  - So wrapping TCP in TCP amplifies this effect
  - TCP-based VPNs collapse when congested
- Probably the most complicated thing you can do with OpenSSH



Honeypot

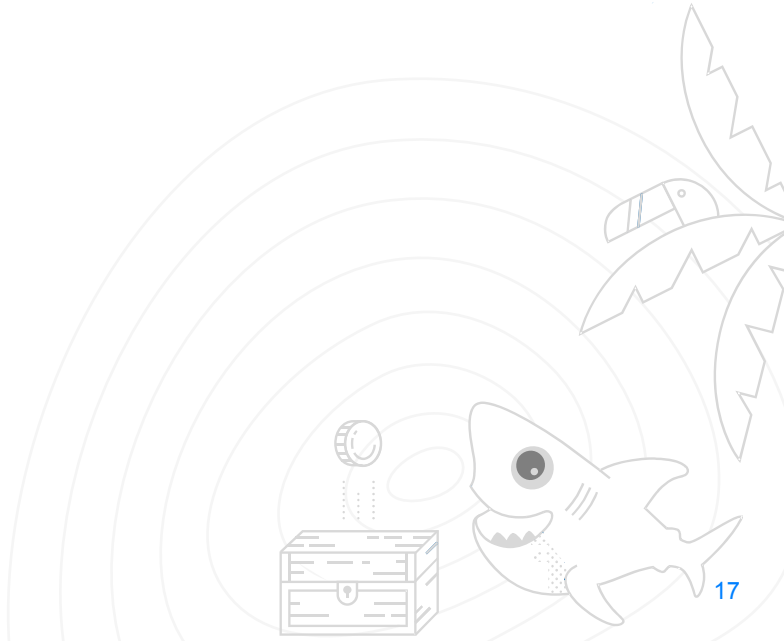
- A honeypot is a server that is intentionally left open for attackers to exploit.
  - Once the attackers are in they are dropped into an environment that looks like a typical server, but is a decoy. Events on this machine are typically ignored and when a user logs off their changes are deleted.

- https://github.com/cowrie/cowrie
- https://cowrie.readthedocs.io/en/latest/INSTALL.html?highlight=log#step-1-install-dependencies



Resources

- https://www.openssh.com/
- SSH Mastery, 2nd Edition - Michael W Lucas



That’s all for this time!

- Be sure to be on the lookout for more DigitalOcean webinars/workshops like this!
- Tune in every last Thursday of the month to watch more of my webinars
- Try out DigitalOcean with $100 free credit for 60 days with https://do.co/mason