1UP
1337

HI-SCORE
19810

HACKER VPN

1 PLAYER

► 200 PLAYERS

EIJAH

CAVE TWINK

Welcome

Hack the Planet!

Introductions

- Eijah
- Cave Twink

0xAA856A1BA814AB99FFDEBA6AEFBE1C04

Prerequisites

- Verify that you have the following:
  - A laptop with x86-64 instructions, Wi-Fi, and support for bridged mode network adapters (if using VM)
  - VirtualBox installed (potential issues with Secure Boot enabled Linux hosts)
  - The Hacker VPN workshop files
    - PDF presentation
    - Linux Virtual Image
- If you don’t have the workshop files, we have copies available on USB. You can also connect to the DEF CON Wi-Fi and download from https://codesiren.com/defcon33
  - This is the **only** time at DEF CON that you can trust a USB
  - The website also has instructions for alternative deployments
    - NOTE! The code and compilation was only tested on Debian 12
  - Cave Twink will help you get setup at this time

Workshop Goals

- Create a Hacker VPN
  - Write C++ code
  - Use TCP and UDP sockets
  - Use the Linux TUN interface (/dev/net/tun)
  - Use 100% CNSA Suite 2.0 PQC cryptography with OpenSSL and CRYSTALS
  - Network routing
- Discuss advanced topics like packet sharding, random noise injection, multi-hop routing, and anonymity
- Come up with new ways to use the code

Exercise: Ice Breaker

- In your opinion, what is a Hacker VPN?
  - A Virtual Private Network (VPN)?
  - A Post Quantum Computing (PQC) end-to-end-encrypted network (E2EE)?
  - An anonymous routing protocol like The Onion Routing project (Tor)?
  - Software that supports random noise injection, multi-hop routing, packet sharding, and 100% anonymity between network endpoints?
  - A hands-on workshop to build a custom tool to replace legacy VPNs?
  - A lightweight, self-hosted and easily deployed and maintained application that can run on low-end hardware such as shared VPS instances and embedded devices?

Client Lesson 0

Setup & Configuration

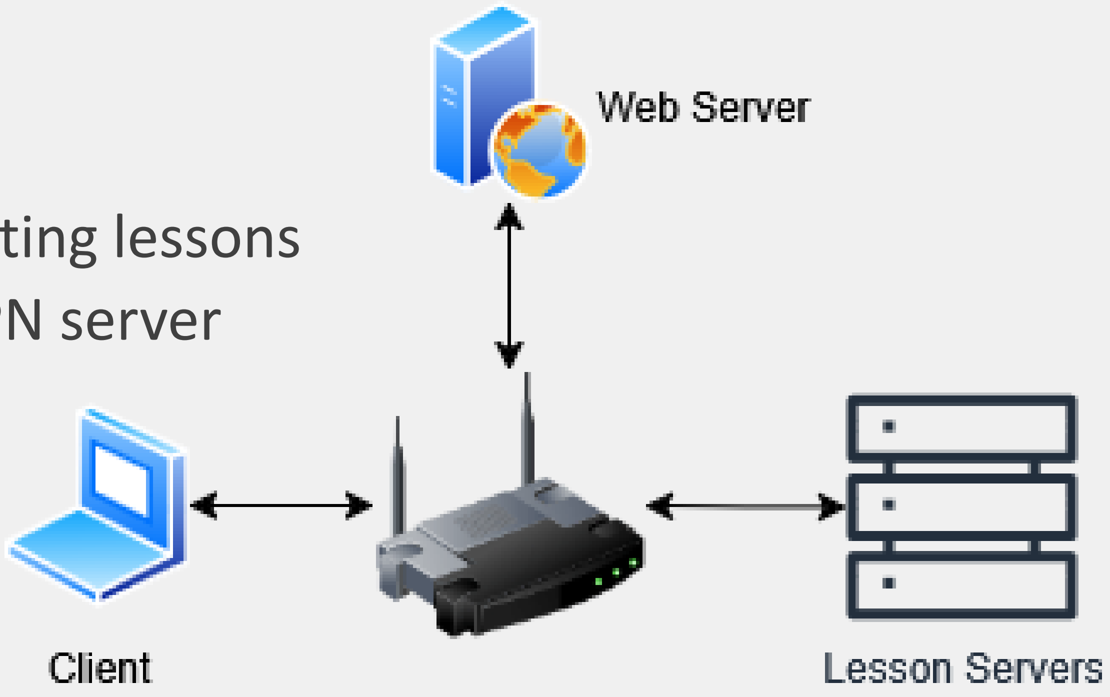
Lesson Goals

- Connect to the wireless network
- Mount the Linux virtual image
- Discuss the different ways you can participate in the workshop

Mac os:
Can you install this 5 year old program?
Nooooo, i can't! this program is too old!
Windows:
Can you install this 25 year old program?
Yes, i can! Installing... done!
Linux
can you install this 25 year old program
it's already installed

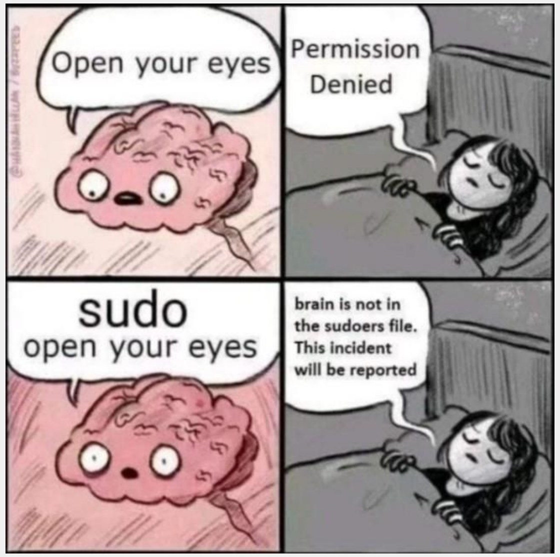
Wireless Network

- Connect to the wireless network. Please don’t hack the wireless networks during the workshop. Ty! <3
- Hacker VPN Network
  - Network Name (SSID): Your_Own_VPN
  - Password: Strong!PQC
  - LAN: 172.16.0.0/12
- Go to http://172.16.0.3
  - Used to see your Source IP when testing lessons
  - Green, you are going through the VPN server
  - Red, you are not



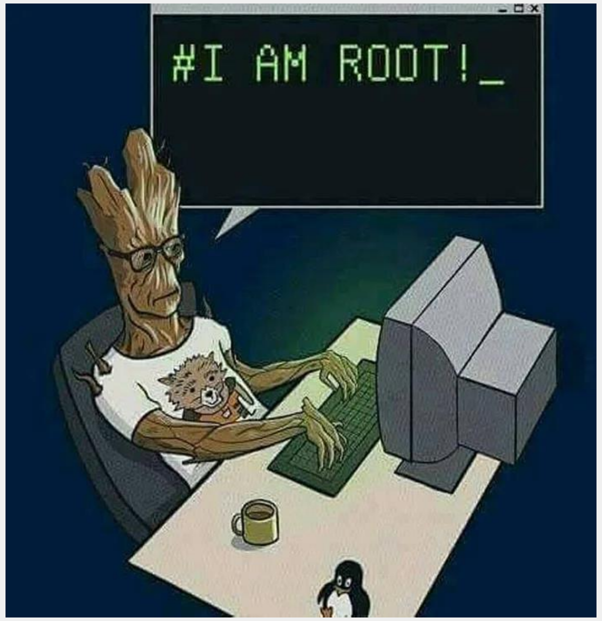
Linux Virtual Machine (VirtualBox)

- Locate the Hacker_VPN.zip file on your machine and extract it
- Open VirtualBox and navigate to File > Preferences
- Change to advanced (if not already set) and click OK
- Go to File > Import Appliance and select the Hacker_VPN.ova as the source
- On the import dialogue that appears, you can expand Settings and modify the CPU and RAM if needed for your machine
  - CPU is set for 4 cores and 4096 MB of RAM by default for a better experience, but you can decrease by half or increase if needed
- Click Finish


Open your eyes
Permission Denied
sudo open your eyes
brain is not in the sudoers file. This incident will be reported

Linux Virtual Machine (VirtualBox cont.)

- Select the Hacker_VM in VirtualBox and click Settings; navigate to Network
- You should see an “Attached to:” option under the Adapter 1 tab
- Use the dropdown to select “Bridged Adapter”
- In the Name field that appears; select your active Wi-Fi adapter
- Click OK to apply
- Now you can start the “Hacker_VPN” virtual machine
- Login with:
  - User: user
  - Password: defcon33



Linux Virtual Machine (Other)

- As mentioned before, this workshop is designed around the x86-64 instruction set and using a Bridged mode network adapter. The instructions below are suggestions to get you in the right direction and not a full guide
- If using Mac
  - You can use UTM to emulate x86-64
    - Create an x86-64 emulated VM.
    - Use the qcow2 file from https://codesiren.com/defcon33/ as the disk
    - Configure the appropriate CPU/RAM and Bridged mode network adapter
- If using Linux
  - Secure Boot causes issues with bridged mode adapters in VirtualBox
  - Recommend installing the dependencies and not using a VM or downloading the qcow2 disk and creating a KVM machine. Note, our code was only tested on Debian 12
    - If creating a VM verify CPU/RAM and Bridged mode network adapter
- Cave Twink will assist as possible, but you might need to follow along without compiling

Additional Information

- You can progress in a variety of ways. In each lesson folder there is a main.cpp and solution.cpp file
- The main.cpp file is used to create your code in
- You can also use the example code in solution.cpp
- If you want to always build the solution, you can optionally set a variable for this
  - To set, open /defcon/code/CMakeLists.txt
  - Change line 11 “set(DEFCON_SOLUTION off)” from off to on
- In /defcon you will find two scripts (client and server). The scripts will be run later in the appropriate lessons
- If you encounter issues, remember Cave Twink can assist

Client Lesson 1

Hello, Hacker World!

Lesson Goals

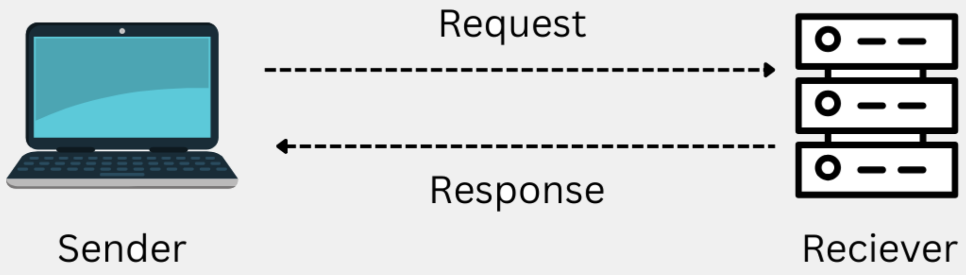
- Learn about the Workshop code base
- Verify that you have your build environment setup correctly
- Print out “Hello, Hacker World!” to the screen
- Use a variety of tools
  - Debian Linux
  - C++ programming language
  - Boost Libraries (especially ASIO)
  - The GNU Compiler
  - CMake & Ninja build systems
  - OpenSSL (Includes PQC algorithms as of 3.5.0)
  - CRYSTALS PQC reference algorithms
    - Dilithium (ML-DSA-87)
    - Kyber (ML-KEM-1024)

Lesson Exercise

- Open VS Code - Applications > Development > Visual Studio Code
- Navigate to the /defcon/code/client/client_lesson_1 folder
- Modify main_1.cpp
  - If you need help with the lesson, take a look at solution_1.cpp
- Navigate to the /defcon/code folder
- Run ./build.sh
  - If you need to clean the project, run ./clean.sh
- Navigate to the /defcon/bin folder
- Run ./client_lesson_1

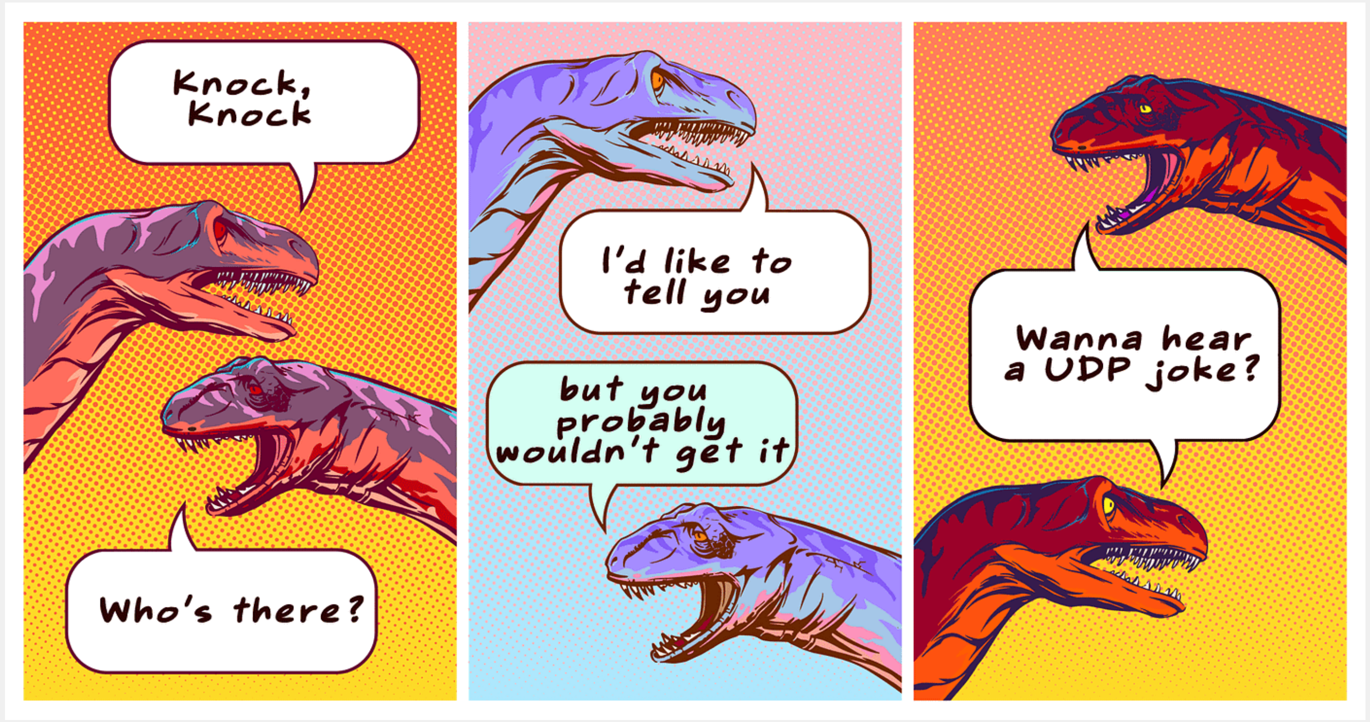
Client Lesson 2

UDP Sockets



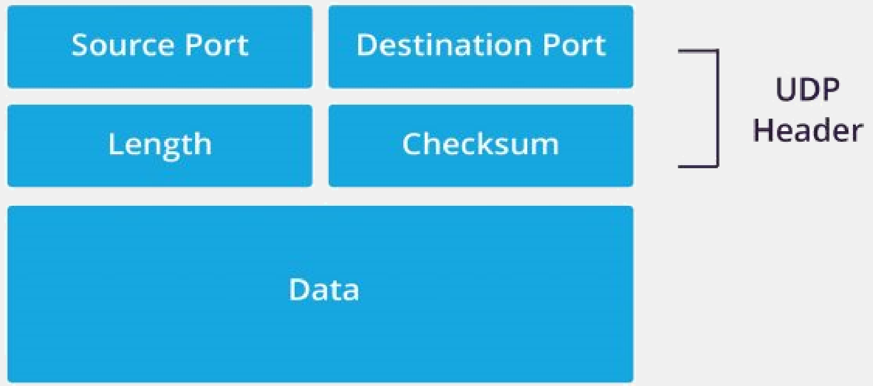
Lesson Goals

- Learn about User Datagram Protocol (UDP)
- Parse command-line arguments
- Write to a UDP socket
- Read from a UDP socket



User Datagram Protocol (UDP)

- UDP is a stateless protocol
- Message-oriented protocol
- Popular for time-sensitive applications
- Fire-and-forget, connectionless protocol with minimal error-checking
- Header Size: 20 bytes (IP) + 8 bytes (UDP)
- Max datagram size: 2^16 (65,536 bytes)
- Maximum Transmission Unit (MTU) problem (1500 – 28 bytes)



Lesson Exercise

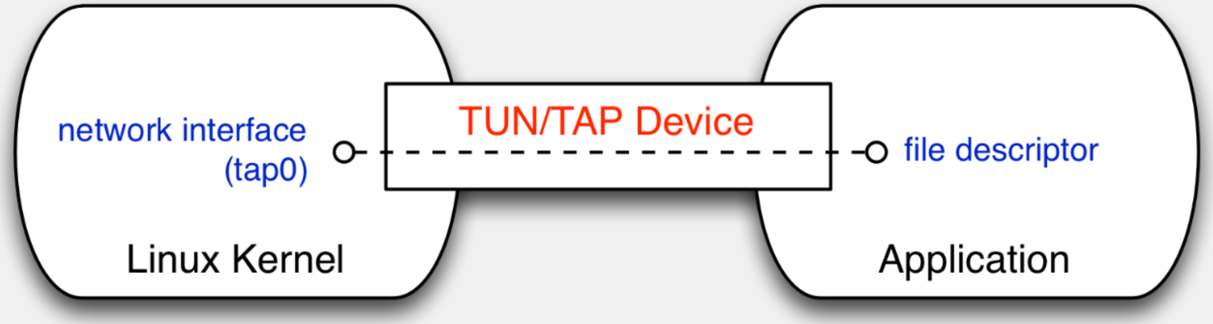
- Navigate to the /defcon/code/client/client_lesson_2 folder
- Modify main_2.cpp
  - If you need help with the lesson, take a look at solution_2.cpp
- Navigate to the /defcon/code folder
- Run ./build.sh
  - If you need to clean the project, run ./clean.sh
- Navigate to the /defcon/bin folder
- Run ./client_lesson_2 <server_endpoint> <message>
  - Server endpoint is 172.16.0.2:2000
  - Message needs to be enclosed in quotes if it contains spaces

Client Lesson 3

TUN Interface

Lesson Goals

- Learn about TUN/TAP interfaces
- Setup Linux routing rules
- Create a TUN interface
- Read from the TUN interface
- Print out the size of packets as they flow through the TUN interface



Lesson Exercise

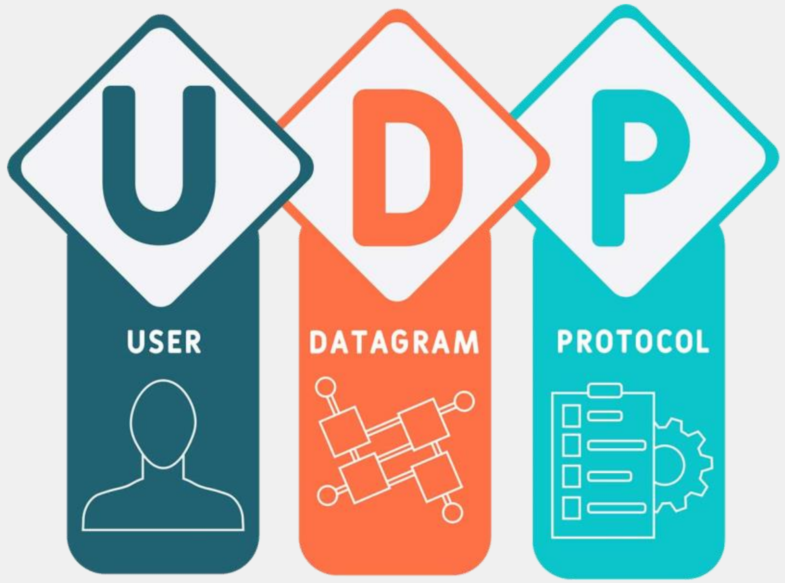
- Navigate to the /defcon folder
- Run: sudo ./client.sh create
  - Write down your LAN and TUN (pqc0) addresses for later.
  - If you need this info again, you can run: ip addr
  - You should not need to undo the changes, but you can with: sudo ./client.sh cleanup
- Navigate to the /defcon/code/client/client_lesson_3 folder
- Modify main_3.cpp
  - If you need help with the lesson, take a look at solution_3.cpp
- Navigate to the /defcon/code folder
- Run ./build.sh
  - If you need to clean the project, run ./clean.sh
- Navigate to the /defcon/bin folder
- Run ./client_lesson_3 <tun_address>
  - TUN address is the address of the pqc0 adapter (10.x.y.z)

Client Lesson 4

TUN Interface + UDP Sockets

Lesson Goals

- Learn about TUN addresses and why they are important for VPNs
- Create a TUN interface
- Read from the TUN interface
- Write to the UDP socket
- Read from the UDP socket
- Write back to the TUN interface

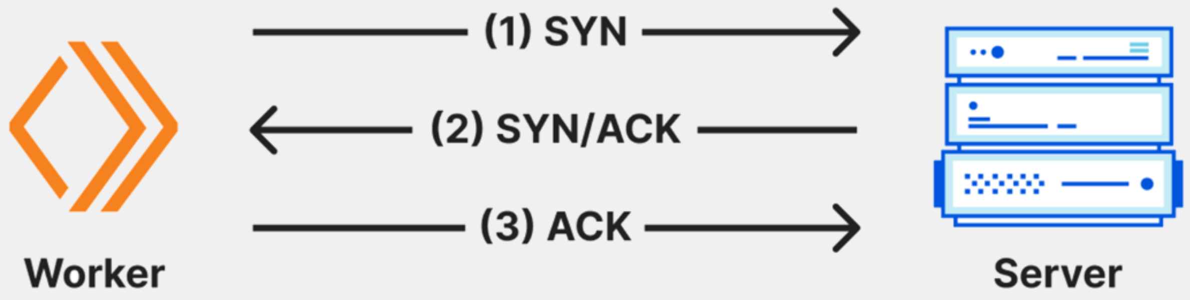


Lesson Exercise

- Navigate to the /defcon/code/client/client_lesson_4 folder
- Modify main_4.cpp
  - If you need help with the lesson, take a look at solution_4.cpp
- Navigate to the /defcon/code folder
- Run ./build.sh
  - If you need to clean the project, run ./clean.sh
- Navigate to the /defcon/bin folder
- Run ./client_lesson_4 <server_endpoint> <tun_address> <lan_address>
  - Server endpoint is 172.16.0.2:4000
  - TUN address is the address of the pqc0 adapter (10.x.y.z)
  - LAN address is the address of the LAN adapter (172.x.y.z)

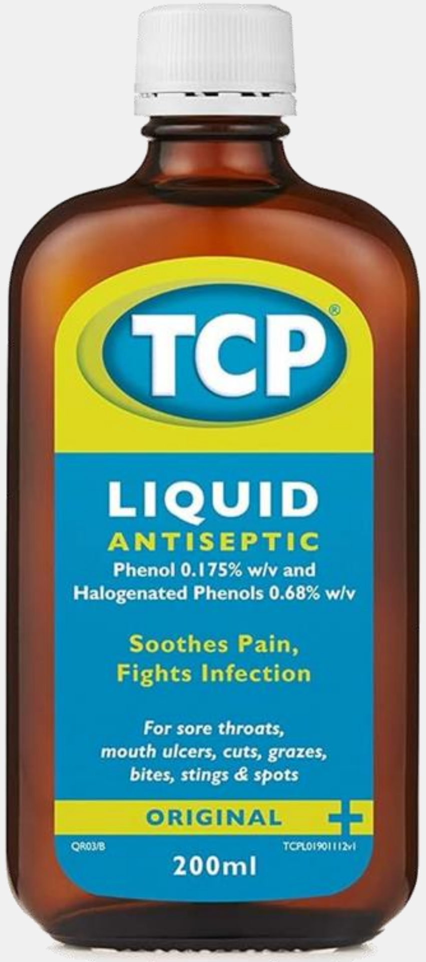
Client Lesson 5

TCP Sockets



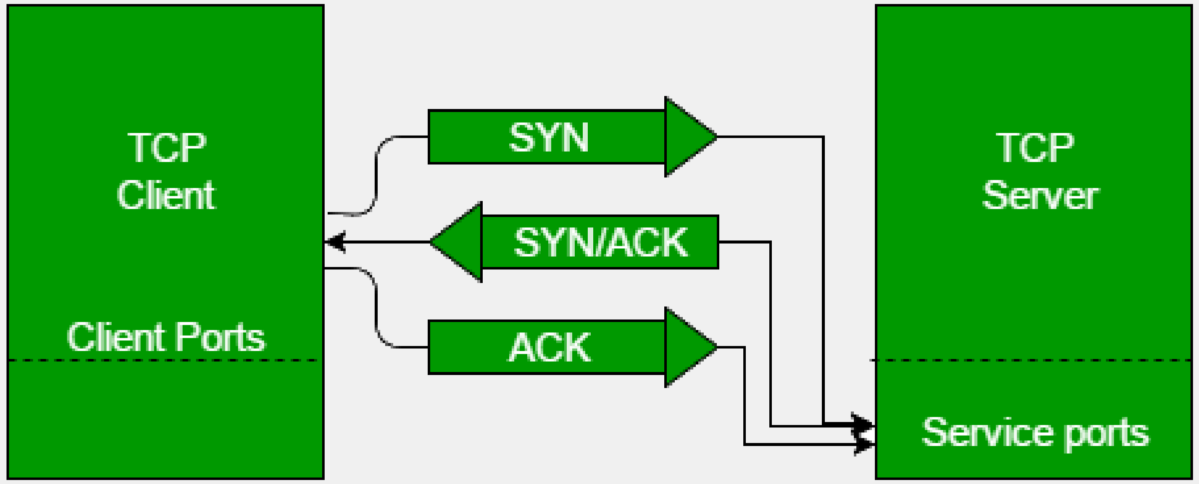
Lesson Goals

- Learn about Transmission Control Protocol (TCP)
- Connect to a TCP socket
- Write to a TCP socket
- Read from a TCP socket



Transmission Control Protocol (TCP)

- Stream-oriented protocol
- Popular for most applications
- Connection-oriented, reliable, flow & congestion control, ordered, retransmission
- Runtime & bandwidth overhead compared to UDP
- Header size: 20 bytes (IP) + 20-60 bytes (TCP) = 40-80 bytes (a lot)



Lesson Exercise

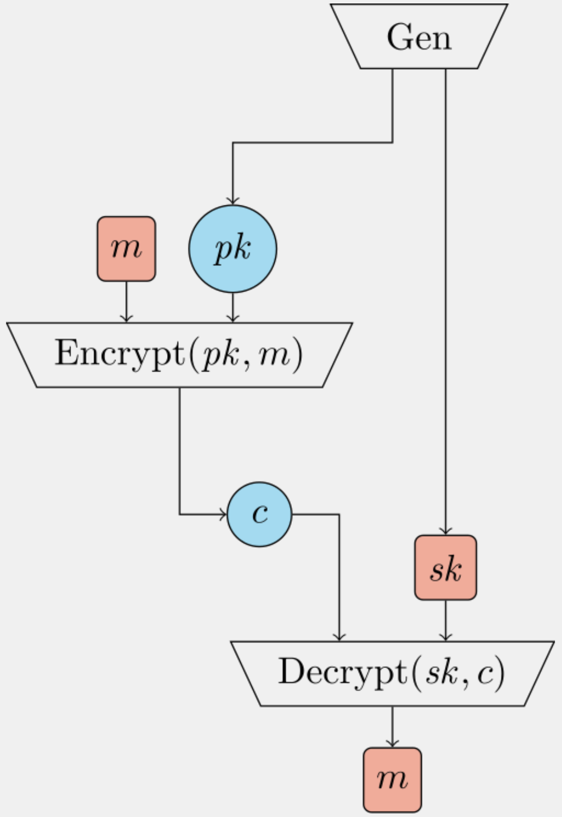
- Navigate to the /defcon/code/client/client_lesson_5 folder
- Modify main_5.cpp
  - If you need help with the lesson, take a look at solution_5.cpp
- Navigate to the /defcon/code folder
- Run ./build.sh
  - If you need to clean the project, run ./clean.sh
- Navigate to the /defcon/bin folder
- Run ./client_lesson_5 <server_endpoint> <message>
  - Server endpoint is 172.16.0.2:5000
  - Message needs to be enclosed in quotes if it contains spaces

Client Lesson 6

Post Quantum Cryptography (PQC)

Lesson Goals

- Learn about Post-Quantum Cryptography (PQC) algorithms
  - Dilithium Digital Signature Algorithm (DSA)
  - Kyber Key Encapsulation Mechanism (KEM), SHA512
  - Advanced Encryption Standard with Galois/Counter Mode (AES-256 GCM)
- Sign/Verify with Dilithium
- Encap/Decap with Kyber
- Encrypt/Decrypt with AES-256 in GCM mode
- SHA-512 hash
- Base16 encode/decode



Lesson Exercise

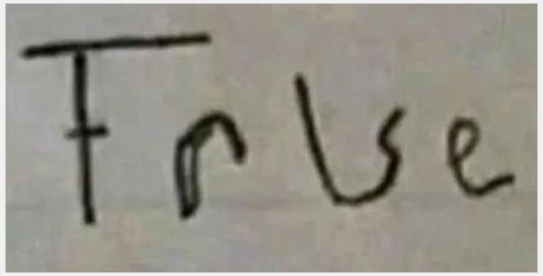
- Navigate to the /defcon/code/client/client_lesson_6 folder
- Modify main_6.cpp
  - If you need help with the lesson, take a look at solution_6.cpp
- Navigate to the /defcon/code folder
- Run ./build.sh
  - If you need to clean the project, run ./clean.sh
- Navigate to the /defcon/bin folder
- Run ./client_lesson_6

Client Lesson 7

PQC Handshake

Lesson Goals

- Learn about CNSA Suite 2.0 compliance
- Design a simple PQC handshake protocol
  - Request, Response
- Generate a PQC request handshake
  - TUN address, Dilithium persistent keys, Kyber ephemeral keys, message signature
- Simulate a PQC response handshake
  - Cryptographic message verification, Kyber encap
- Consume the PQC response
  - Message verification, Kyber decap, Derive shared AES-256 key


False

Lesson Exercise

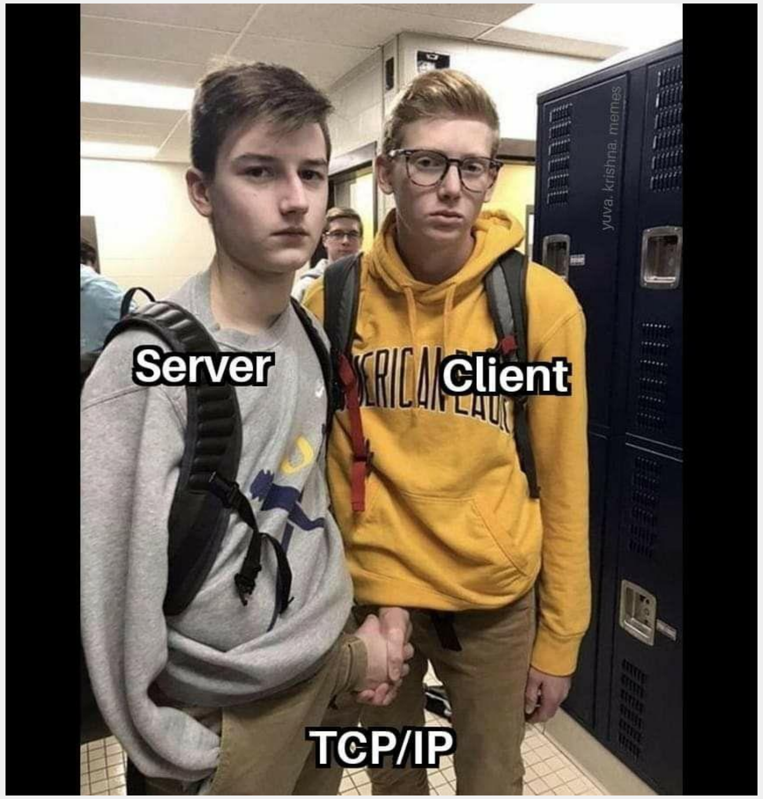
- Navigate to the /defcon/code/client/client_lesson_7 folder
- Modify main_7.cpp
  - If you need help with the lesson, take a look at solution_7.cpp
- Navigate to the /defcon/code folder
- Run ./build.sh
  - If you need to clean the project, run ./clean.sh
- Navigate to the /defcon/bin folder
- Run ./client_lesson_7

Client Lesson 8

PQC Handshake + TCP Sockets

Lesson Goals

- Connect to a TCP socket
- Write PQC handshake request to a TCP socket
- Read PQC handshake response from a TCP socket
- Derive a shared AES-256 key

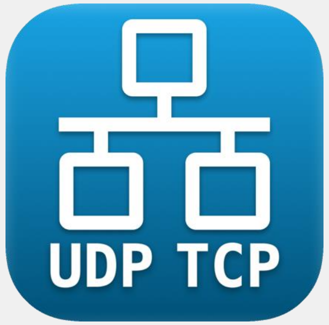


Lesson Exercise

- Navigate to the /defcon/code/client/client_lesson_8 folder
- Modify main_8.cpp
  - If you need help with the lesson, take a look at solution_8.cpp
- Navigate to the /defcon/code folder
- Run ./build.sh
  - If you need to clean the project, run ./clean.sh
- Navigate to the /defcon/bin folder
- Run ./client_lesson_8 <server_endpoint> <tun_address>
  - Server endpoint is 172.16.0.2:8000
  - TUN address is the address of the pqc0 adapter (10.x.y.z)

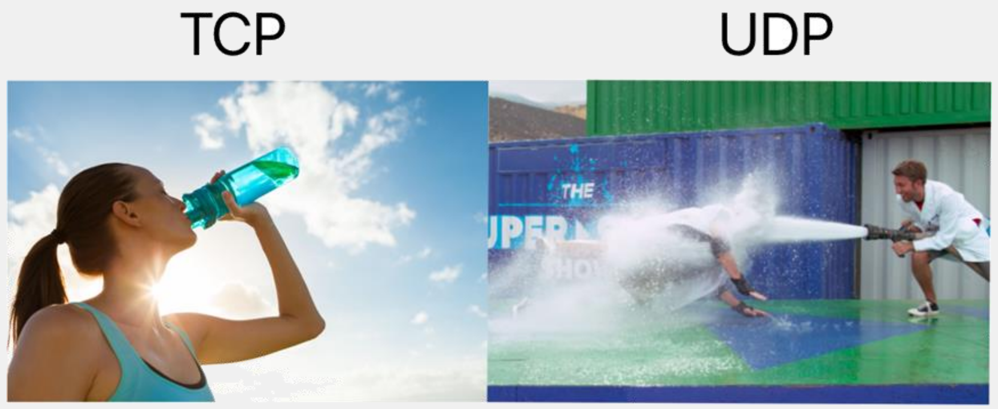
Client Lesson 9

PQC Handshake + TCP/UDP Sockets



Lesson Goals

- TCP socket
  - Connect to TCP socket, write PQC handshake request to TCP socket, read PQC handshake response from TCP socket, verify signature, derive AES-256 key
- TUN Interface
  - Read from TUN interface, encrypt data with AES-256, prepend 4 byte ID to encrypted message, write to UDP socket
- UDP Socket
  - Read from UDP socket, decrypt data with AES-256, write to TUN interface



Lesson Exercise

- Navigate to the /defcon/code/client/client_lesson_9 folder
- Modify main_9.cpp
  - If you need help with the lesson, take a look at solution_9.cpp
- Navigate to the /defcon/code folder
- Run ./build.sh
  - If you need to clean the project, run ./clean.sh
- Navigate to the /defcon/bin folder
- Run ./client_lesson_9 <server_endpoint> <tun_address> <lan_address>
  - Server endpoint is 172.16.0.2:9000
  - TUN address is the address of the pqc0 adapter (10.x.y.z)
  - LAN address is the address of the LAN adapter (172.x.y.z)

Server Lesson 2

UDP Sockets

Lesson Exercise

- Navigate to the /defcon/code/server/server_lesson_2 folder
- Modify main_2.cpp
  - If you need help with the lesson, take a look at solution_2.cpp
- Navigate to the /defcon/code folder
- Run ./build.sh
  - If you need to clean the project, run ./clean.sh
- Navigate to the /defcon/bin folder
- Run ./server_lesson_2 <port>
  - Port should be 2000

Server Lesson 4

TUN Interface + UDP Sockets

Lesson Exercise

- Navigate to the /defcon folder
- Run sudo ./server.sh create
  - To remove, run sudo ./server.sh cleanup
- Navigate to the /defcon/code/server/server_lesson_4 folder
- Modify main_4.cpp
  - If you need help with the lesson, take a look at solution_4.cpp
- Navigate to the /defcon/code folder
- Run ./build.sh
  - If you need to clean the project, run ./clean.sh
- Navigate to the /defcon/bin folder
- Run ./server_lesson_4 <port> <tun_address>
  - Port should be 4000
  - TUN address should be 10.0.0.1

Server Lesson 5

TCP Sockets

Lesson Exercise

- Navigate to the /defcon/code/server/server_lesson_5 folder
- Modify main_5.cpp
  - If you need help with the lesson, take a look at solution_5.cpp
- Navigate to the /defcon/code folder
- Run ./build.sh
  - If you need to clean the project, run ./clean.sh
- Navigate to the /defcon/bin folder
- Run ./server_lesson_5 <port>
  - Port should be 5000

Server Lesson 8

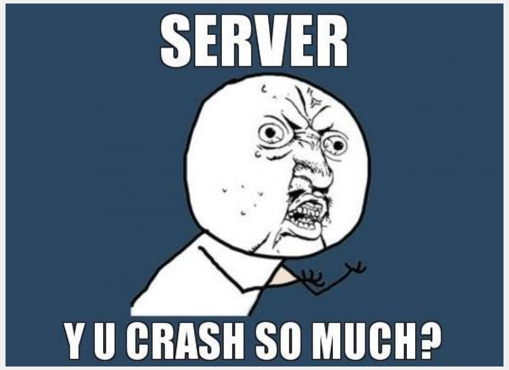
PQC Handshake + TCP Sockets

Lesson Exercise

- Navigate to the /defcon/code/server/server_lesson_8 folder
- Modify main_8.cpp
  - If you need help with the lesson, take a look at solution_8.cpp
- Navigate to the /defcon/code folder
- Run ./build.sh
  - If you need to clean the project, run ./clean.sh
- Navigate to the /defcon/bin folder
- Run ./server_lesson_8 <port>
  - Port should be 8000

Server Lesson 9

PQC Handshake + TCP/UDP Sockets

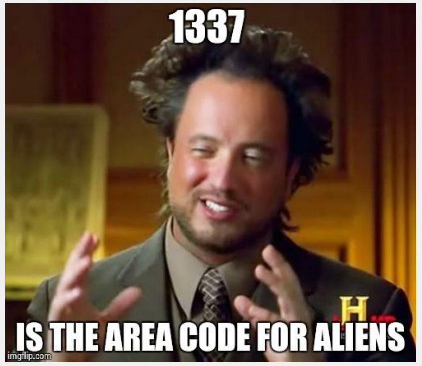


Lesson Exercise

- Navigate to the /defcon/code/server/server_lesson_9 folder
- Modify main_9.cpp
  - If you need help with the lesson, take a look at solution_9.cpp
- Navigate to the /defcon/code folder
- Run ./build.sh
  - If you need to clean the project, run ./clean.sh
- Navigate to the /defcon/bin folder
- Run ./server_lesson_9 <port> <tun_address>
  - Port should be 9000
  - TUN address should be 10.0.0.1

Advanced Lesson

1337 H4x0r Stuff



Lesson Goals

- Random Noise Injection
- Multi-hop Routing
- Packet Sharding
- Anonymity

Random Noise Injection

- There are 2 types of random noise injection
  - Incremental noise to existing packets
  - New packets made up of 100% noise
- Random noise packets can be
  - Variable or fixed-sized packets
  - Variable packets need to discern between message and noise bytes
  - Fixed-size is simpler, wastes bandwidth, but is arguably more secure (Variable Bitrate - VBR)
- Message Design
  - Use a secure source of randomness (OpenSSL RAND_bytes)
  - Incorporate optional random noise into the protocol
    - message = tun_address | cipher_text
    - cipher_text = message_size | message | random_noise (optional)

Multi-hop Routing

- Traffic Routing
  - Traffic doesn’t need to pass directly from the client to the server
  - There can be relays (intermediate nodes) between clients and servers
  - There can be multiple servers (TUN IP uniqueness problem)
- Relays
  - Can be intelligent or simple pass-throughs
  - Can redirect traffic directly to servers or hop to 1:N relays
  - Unwinding the response is the tricky part (TUN IP to UDP endpoint mappings)
- Advanced
  - Hop TTL
  - Encrypted client-to-relay, relay-to-relay, and relay-to-server
  - Prevent reentrancy

Packet Sharding

- Can mean two different things:
  - Spreading packets across multiple nodes
  - Breaking packets up into sub-packet units for network dissemination
- Connection-oriented Protocols
  - TUN-IP uniqueness problem
  - Difficult to shard across N servers
  - Exit node termination is tricky

Anonymity

- 2 Different Types of Anonymity
  - Clients never send VPN encrypted traffic directly to servers
  - Clients never communicate directly to servers
- Different Types of Relays
  - Perform point-to-point pass-through
  - Perform encryption and point-to-point redirection
- TUN IP Uniqueness
  - Clients must be unique to the servers
  - Clients must lease a TUN IP address (10.x.y.z)
- True Anonymity
  - Achieved via initial out-of-band handshakes between clients and servers
  - Achieved via initial out-of-band lease of unique TUN IP
  - Future rekeying can be done via ratchetting of the encryption keys

Summary

"It's in that place I put that thing that one time"
— Hackers, The Phantom Phreak

Knowledge is Pwnage

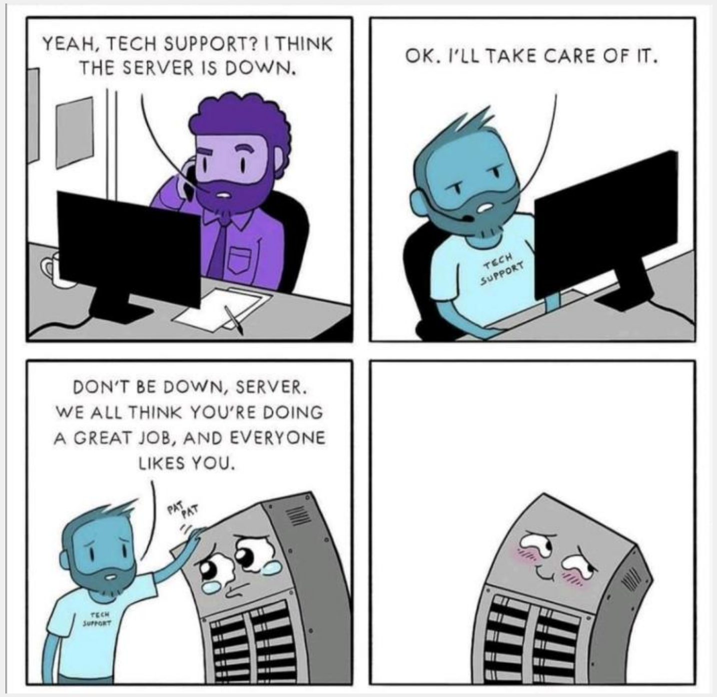
- The Internet is a dangerous place
- But it's much safer when we take control
- What are the possible uses of your Hacker VPN?
- What are your ideas?

Potential Concerns

- This workshop is for educational purposes. The Hacker VPN has been simplified and there are some critical limitations
  - Routing vs Packet Filter (aka Kill Switch)
  - DNS Leaks
  - Simple Session Handling
  - Multi-Platform
  - CGNAT punching

Next Steps

- How do I learn how to become an awesome programmer?
  - The best way to learn is to do
- Open source communities, online resources, books, etc.
- Remember… you're hackers… you can do anything!
- Thanks for spending the day with us!
- We hope you've enjoyed this DEF CON workshop
- Cave Twink & I do this for you. <3


YEAH, TECH SUPPORT? I THINK THE SERVER IS DOWN.
OK. I'LL TAKE CARE OF IT.
TECH SUPPORT
DON'T BE DOWN, SERVER. WE ALL THINK YOU'RE DOING A GREAT JOB, AND EVERYONE LIKES YOU.
PAT PAT

Check us out…

https://www.codesiren.com

(we’re hiring)

References

- https://www.debian.org
- https://gcc.gnu.org
- https://www.boost.org
- https://cmake.org
- https://openssl.org
- https://pq-crystals.org
- https://en.wikipedia.org/wiki/Key_encapsulation_mechanism
- https://en.wikipedia.org/wiki/Digital_Signature_Algorithm
- https://en.wikipedia.org/wiki/Advanced_Encryption_Standard