EIJAH.COM

Subverting Privacy Exploitation Using HTTP

Eijah

v 1.0

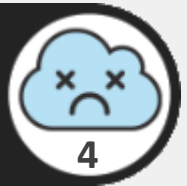
July 27th, 2017

Who am I?

- Founder
- Programmer
- Hacker
- CEO

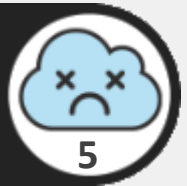
Prerequisites

- Laptop
- Virtual Box (www.virtualbox.org) or any other virtualization software that supports *.vmdk files
- Prior experience in at least 1 programming language



The Hacker Way

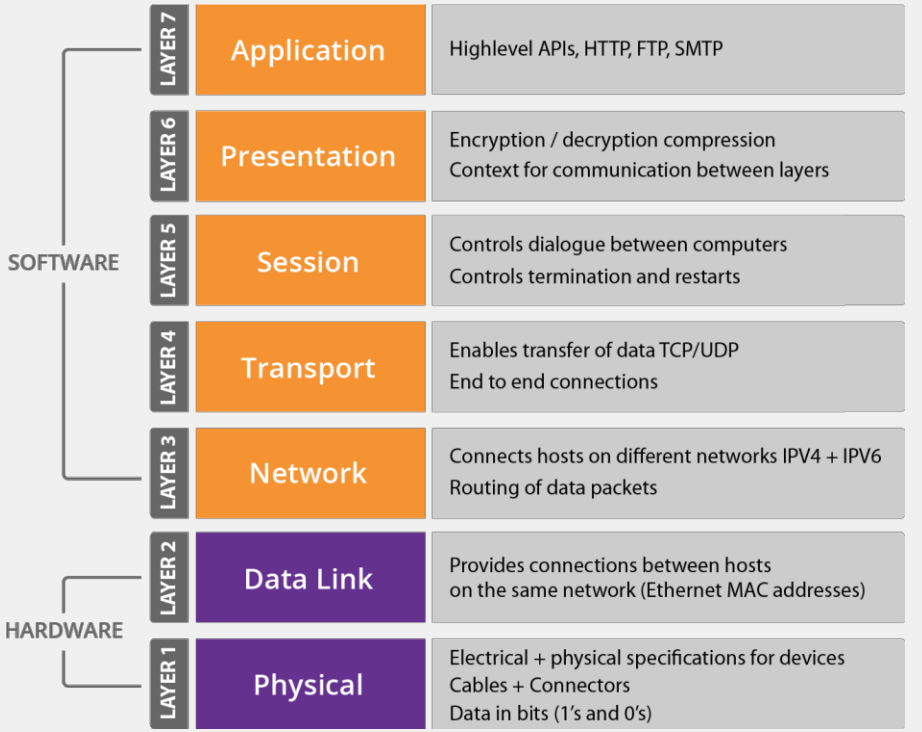
- What does it mean "to be secure"?
- Confidentiality, Integrity, and Availability
- Authenticity and Non-repudiation
- Free Software
  - Open Source
  - Open Infrastructure (Walled Gardens)
  - Runtime verification/Sandboxing
- Evade & Subvert: Our new best friends
  - Escape or avoid, especially by cleverness or trickery
  - Undermining of the power and authority of an established system



HTTP Overview

A Brief History of HTTP

- 30 years ago there was no standard for data communications
- Tim Berners-Lee, CERN, IETF, W3C, and a lot of RFCs
- The OSI Model and Layer 7: The Application Layer
- A flexible request & response protocol
- Welcome to the World Wide Web

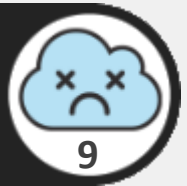


OSI Layers

Layer		OSI protocols	TCP/IP protocols	Signaling System 7[10]	AppleTalk	IPX	SNA	UMTS	Miscellaneous examples
No.	Name								
7	Application	FTAM · X.400 · X.500 · DAP · ROSE · RTSE · ACSE[11] · CMIP[12]	NNTP · SIP · SSI · DNS · FTP · Gopher · HTTP · NFS · NTP · DHCP · SMPP · SMTP · SNMP · Telnet · LDAP · SMB · BGP · FCIP	INAP · MAP · TCAP · ISUP · TUP	AFP · ZIP · RTMP · NBP	SAP	APPC		HL7 · Modbus
6	Presentation	ISO/IEC 8823 · X.226 · ISO/IEC 9576-1 · X.236	MIME · SSL · TLS · XDR		AFP				TDI · ASCII · EBCDIC · MIDI · MPEG
5	Session	ISO/IEC 8327 · X.225 · ISO/IEC 9548-1 · X.235	Sockets (session establishment in TCP / RTP / PPTP)		ASP · ADSP · PAP	NWLink	DLC?		Named pipes · NetBIOS · SAP · half duplex · full duplex · simplex · RPC · SOCKS
4	Transport	ISO/IEC 8073 · TP0 · TP1 · TP2 · TP3 · TP4 (X.224) · ISO/IEC 8602 · X.234	TCP · UDP · SCTP · DCCP			DDP · SPX			NBF
3	Network	ISO/IEC 8208 · X.25 (PLP) · ISO/IEC 8878 · X.223 · ISO/IEC 8473-1 · CLNP X.233 · ISO/IEC 10589 · IS-IS	IP · IPsec · ICMP · IGMP · OSPF · RIP	SCCP · MTP	ATP (TokenTalk / EtherTalk)	IPX		RRC / BMC	NBF · Q.931 · NDP · IS-IS
2	Data link	ISO/IEC 7666 · X.25 (LAPB) · Token Bus · X.222 · ISO/IEC 8802-2 · LLC (type 1 / 2)[13]	PPP · SBTV · SLIP	MTP · Q.710	LocalTalk · ARA · PPP	IEEE 802.3 framing Ethernet II framing	SDLC	PDCP[14] · LLC · MAC	ARP · ARQ · ATM · Bit stuffing · CDP · CRC · DOCSIS · FDDI · FDP · Fibre Channel · Frame Relay · HDP · HDLC · IEEE 802.3 (Ethernet) · IEEE 802.11a/b/g/n (Ethernet MAC and LLC) · IEEE 802.1Q (VLAN) · ISL · ITU-T G.hn DLL · Linux interface bonding · PPP · Q.921 · Token Ring
1	Physical	X.25 (X.21bis · EIA/TIA-232 · EIA/TIA-449 · EIA-530 · G.703)[13]		MTP · Q.710	RS-232 · RS-422 · STP · PhoneNet		Twinax	L1 (physical)	RS-232 · Full duplex · RJ45 (8P8C) · V.35 · V.34 · I.430 · I.431 · T1 · E1 · 10BASE-T · 100BASE-TX · 1000BASE-T · POTS · SONET · SDH · DSL · 802.11a/b/g/n PHY · ITU-T G.hn PHY · CAN bus · DOCSIS · DWDM · OTN

Overview of HTTP

- HTTP is not scary (it's easy to understand the basics)
  - Different versions (1.0, 1.1, 2.0)
  - Request/Response (GET, POST)
  - Header fields, body, session, cookies
- wget
  - http://www.eijah.com
  - wget -r --no-parent http://site.com/songs/
- curl
  - http://www.eijah.com
- telnet
  - telnet www.google.com 80
  - GET / HTTP/1.0
  - Host: www.google.com



HTTP Request Format

- https://www.tutorialspoint.com/http/http_requests.htm
- A Request-line
  - Method *SPACE* Request-URI *SPACE* HTTP-Version *CRLF*
- 0:N header fields followed by CRLF
- Empty line (i.e., a line with nothing preceding the CRLF) indicating the end of the header fields
- Message-body (optional)

HTTP GET

- Example 1

  GET /index.html HTTP/1.0

  Host: www.google.com

- Example 2

  GET /index.html HTTP/1.1

  User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)

  Host: www.google.com

  Accept-Language: en-us

  Accept-Encoding: gzip, deflate

  Connection: Keep-Alive

HTTP Response

- https://www.tutorialspoint.com/http/http_responses.htm
- A Status-line
  - HTTP-Version *SPACE* Status-Code *SPACE* Reason-Phrase *CRLF*
- 0:N header fields followed by CRLF
- An empty line (i.e., a line with nothing preceding the CRLF) indicating the end of the header fields
- Message-body (optional)

HTTP Response

- Example 1

  HTTP/1.1 200 OK

- Example 2

  HTTP/1.1 200 OK

  Date: Mon, 27 Jul 2009 12:28:53 GMT

  Server: Apache/2.2.14 (Win32)

  Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT

  Set-Cookie: id=1234567890;

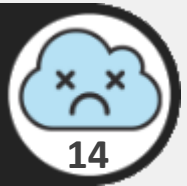
  Connection: Closed

HTTP POST

- Example 1

  POST / HTTP/1.0
  User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
  Host: www.google.com
  Content-Type: application/x-www-form-urlencoded
  Content-Length: 12
  Accept-Language: en-us
  Cookie: id=1234567890
  Accept-Encoding: gzip, deflate

  Hello World!



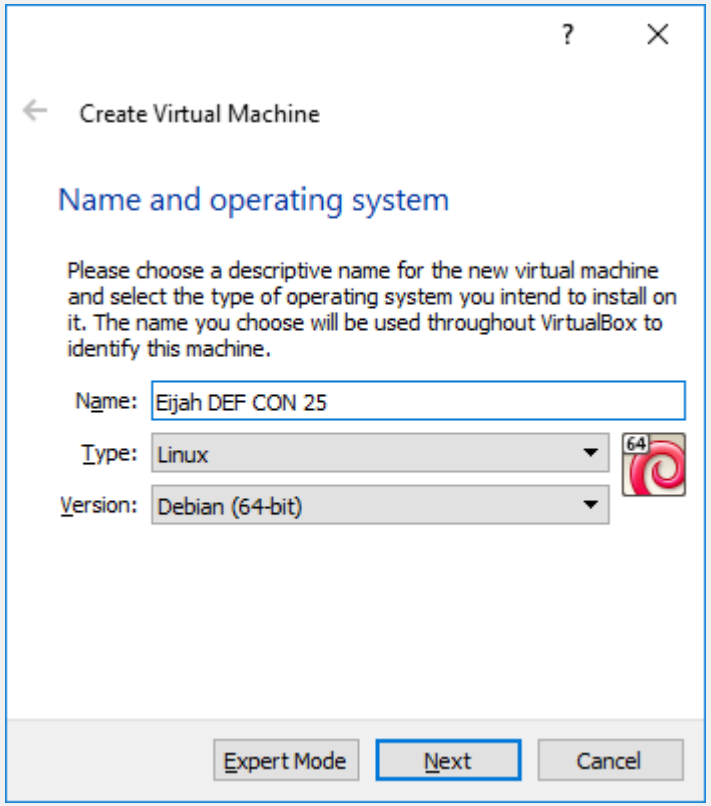
HTTP for Secure Messaging

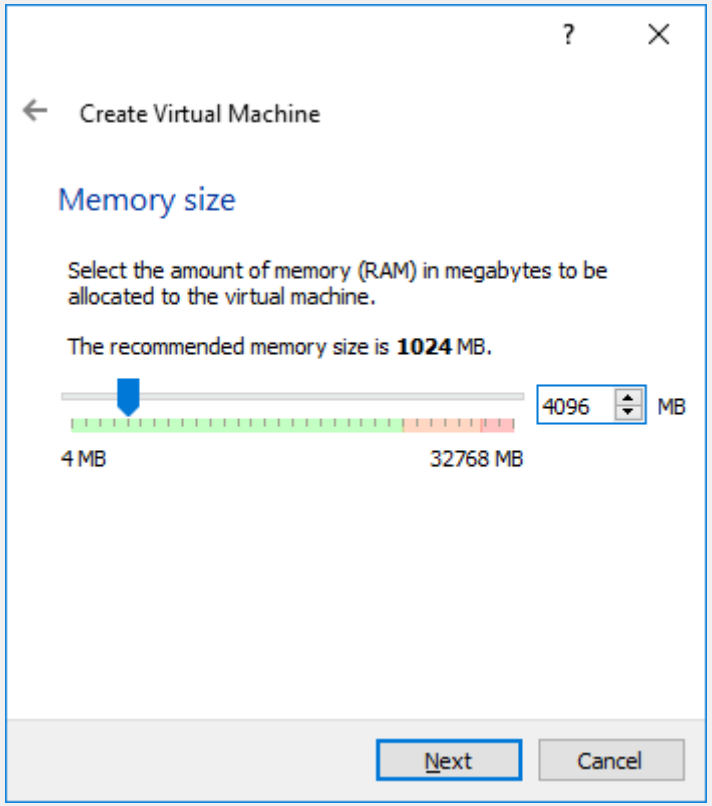
- Isn’t HTTP insecure?
- How can HTTP be used to guarantee confidentiality of data?
- HTTP is…
  - Flexible
  - Adaptive
  - Ubiquitous & Pervasive (everywhere)
  - Open
  - Subversion-friendly
    - Great at hiding underlying data
    - Perfect solution for securing our communications

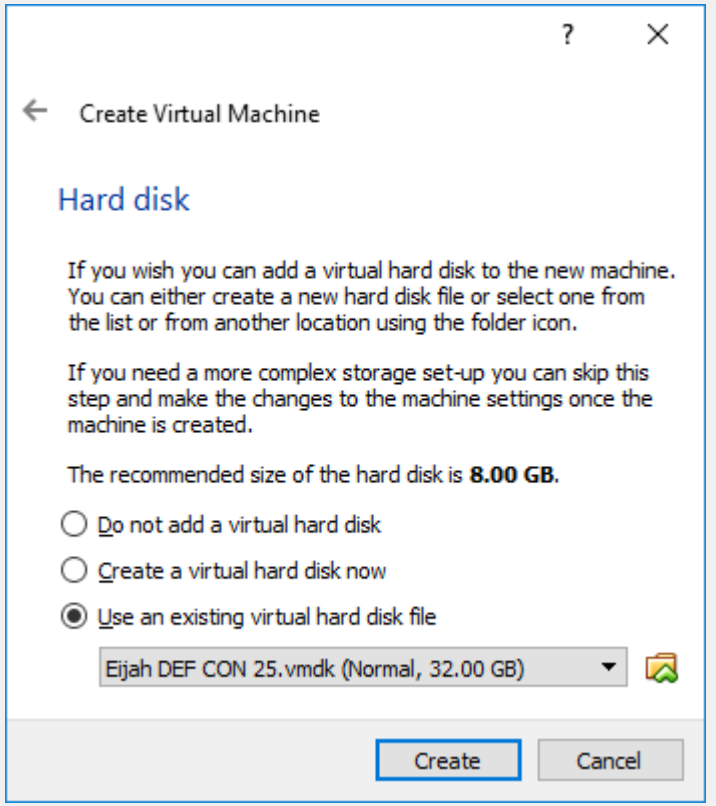
Configuration

Configuring the Environment

- Open Oracle VM VirtualBox Manager (www.virtualbox.org)
- Click on “New” in the toolbar







Configuring the Environment

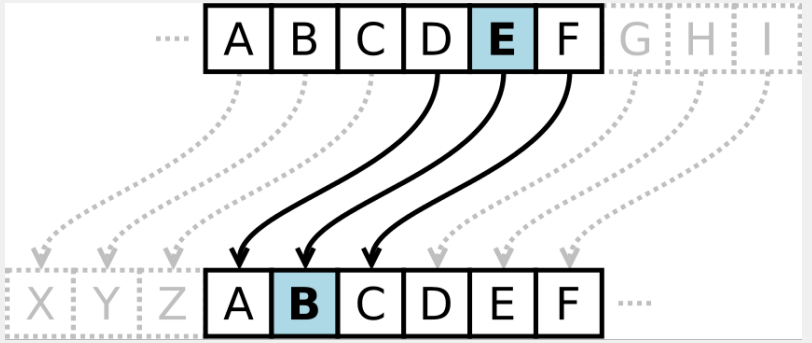
- Start the Virtual Machine
- Login as…
  - root
  - password
- Navigate to /defcon
- Run ./configure
- Run make clean
- Run make –j4
- Navigate to /defcon/_bin/release
- Run ./1_lesson

Our Crypto Weapons

- Random Number Generators (RNG)
- Checksums (CRC32, Adler32)
- Encodings (URL, Base64, Hex)
- Hashes (MD, SHA, SHA-3)
- Ciphers (AES, RC6, Serpent, MARS, Twofish)
- Compression (LZ4)

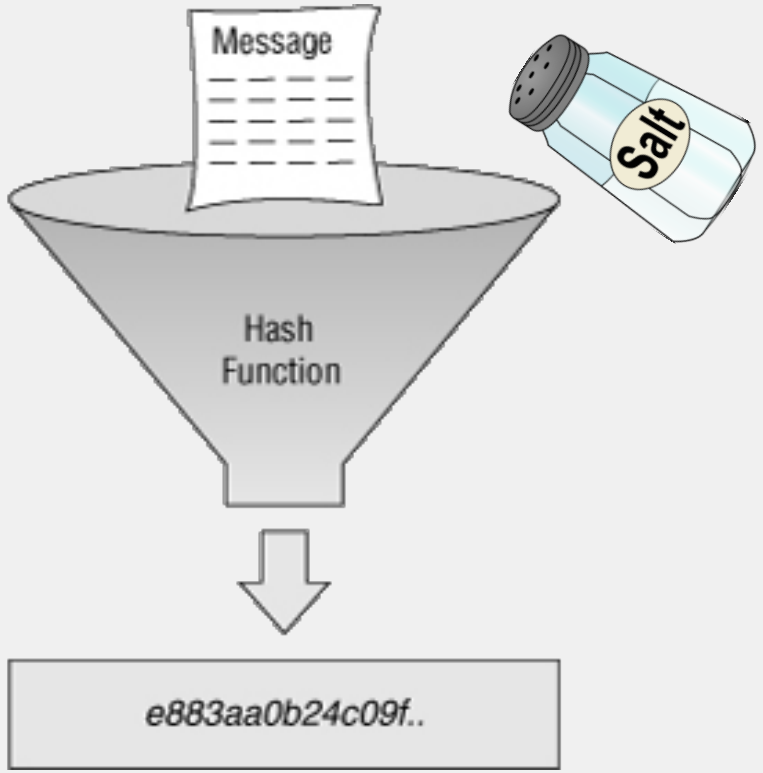
Ciphers

- Algorithm for performing encryption or decryption
  - Plaintext → ciphertext
  - Ciphertext → plaintext
- Usages
  - Symmetric, Asymmetric
  - Everywhere (Software, hardware, networking, databases, DRM, etc.)
- Algorithms
  - Rijndael (AES), Serpent, Twofish, RC6, MARS
  - 128, 192, 256 bit keys
- Notes
  - NIST AES Standardization Process



Hash Functions

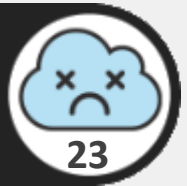
- One-way conversion function
  - Message → (Message) Digest
  - Practically impossible to reverse engineer
  - Fixed sized output w/ optional salt
- Usages
  - Verify file integrity, hashing passwords, digital signatures
  - Checksums, keyed data structure indexes
- Algorithms
  - MD5, SHA-1, SHA-2, SHA-3
- Concerns
  - Collisions (limited key space)
  - Rainbow attacks
  - Length extension hacks



Lessons

Lesson 0 – Interfacing with External Libraries

- 3rd Party Library: Boost
  - https://www.boost.org
- 3rd Party Library: Crypto++
  - https://www.cryptopp.com
- 3rd Party Library: cpp-netlib
  - http://cpp-netlib.org
- 3rd Party Library: Demonsaw
  - https://www.demonsaw.com

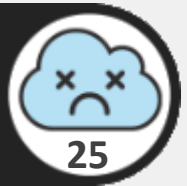


Lesson 0 – What Do We Want to Say?

- Message protocol design 101
- To start, we need a message ;)
- Is that all?
  - Message?
  - Timestamp?
  - Checksum?
  - Session Ids?
  - Unique identifiers?
  - …?
- Let’s discuss this in more detail
  - [Begin fellow human interaction]

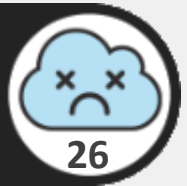
Lessons

- All the lessons are divided into 2 sections.
- The first section is for the student to complete the exercise
- The second section contains a proposed answer by the instructor
- Feel free to “go at it alone”, team up, or even just run/modify the instructor solution.
- Lessons
    1. wget
    2. HTTP Server
    3. HTTP Client (GET Request)
    4. HTTP Client (POST Request)
    5. Advanced Example



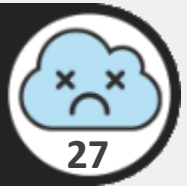
Lesson 1 – wget

- Check out the different methods available on the client, request, response, and uri classes
  - file:///defcon/help/cpp-netlib.org/0.11.2/reference.html
- Helpful Links
  - https://www.jmarshall.com/easy/http/
  - file:///defcon/help/cpp-netlib.org/0.11.2/index.html
  - file:///defcon/help/cpp-netlib.org/0.11.2/examples.html
  - file:///C:/defcon/help/cpp-netlib.org/0.11.2/in_depth.html
  - file:///defcon/help/cpp-netlib.org/0.11.2/in_depth/uri.html
  - file:///C:/defcon/help/cpp-netlib.org/0.11.2/in_depth/http.html



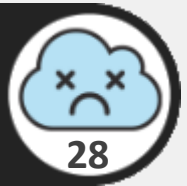
Lesson 1 – wget

- Navigate to /defcon/1_lesson and open main.cpp
- Open main.cpp and modify the code as necessary
- Create your own version of wget in main.cpp. The answer is at the bottom of the file if you need help and/or just want to see it work.
- Navigate to /defcon and run ./configure
- Navigate to /defcon/_bin/release and run ./1_lesson



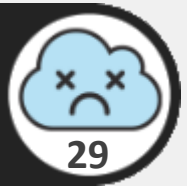
Lesson 2 – HTTP Server

- Listen on a port, usually 80, and respond to incoming requests
- State management via session variables (cookie and set-cookie)
- Handling error conditions and severing sockets
- Delays and avoiding DOS and spam attacks
- Optimization techniques, socket reuse, and multi-threaded scalability



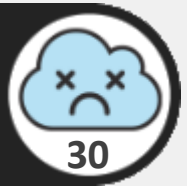
Lesson 2 – HTTP Server

- file:///defcon/help/cpp-netlib.org/0.11.2/examples/http/hello_world_server.html
- I've bootstrapped the code to get you up and running fast
  - Servers running on ports 8000, 8001, 8002
  - Port 8000 is for testing (clear-text)
  - Ports 8001 & 8002 are encrypted communication channels that you will modify to match the crypto from the client
  - You can change IP, add additional threads, etc.
- /defcon/system/utility.h contains helpful functions and global variables that you can change
- Modify 1_lesson, printing out the header variables, setting a session cookie, modifying the response data, etc.
- Have fun and explore what the server can do



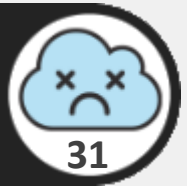
Lesson 2 – HTTP Server

- Navigate to /defcon/2_lesson
- Open main.cpp and modify the code as necessary
- Navigate to /defcon and run ./configure
- Navigate to /defcon/_bin/release and run ./2_lesson
- Open a web browser and go to http://127.0.0.1:8000
- The server response should appear in the browser window



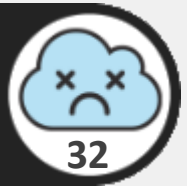
Lesson 3 – HTTP Client (GET Request)

- file:///defcon/help/cpp-netlib.org/0.11.2/in_depth/uri.html
- GET is the simplest of the HTTP commands
- It issues a request for a resource, then receives a response back
- Let's create a simple application that encrypts an HTTP GET request
- Add some header variables for fun
  - You can mimic a standard browser by using the right headers (Lesson 5)
- Sending payload data via the query string
  - All query string data must be properly encoded
  - You can use URL, base 64, or hex encoding to accomplish this
  - /defcon/0_core/security/filter/ have C++ classes that will do this



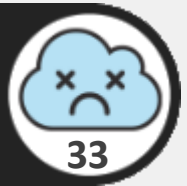
Lesson 3 – HTTP Client (GET Request)

- Navigate to /defcon/3_lesson
- Open main.cpp and modify the code as necessary
- Navigate to /defcon and run ./configure
- Navigate to /defcon/_bin/release and run ./3_lesson



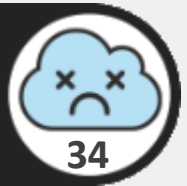
Lesson 4 – HTTP Client (POST Request)

- file:///defcon/help/cpp-netlib.org/0.11.2/in_depth/uri.html
- POST is a more complicated HTTP command that allows the submission of different types of MIME-compliant data
- It allows the client to send a variety of fields/variables so that clients can interact dynamically with the server
- Let's create a simple application that encrypts an HTTP POST request, sending a variety of different types of data



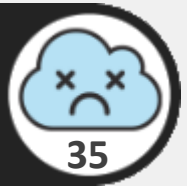
Lesson 4 – HTTP Client (POST Request)

- Navigate to /defcon/4_lesson
- Open main.cpp and modify the code as necessary
- Navigate to /defcon and run ./configure
- Navigate to /defcon/_bin/release and run ./4_lesson



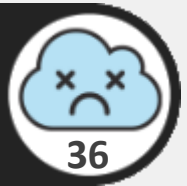
Lesson 5 – Advanced Options

- Hiding in plain sight
- Obfuscate encrypted HTTP in such a way as to remain undetected by network infrastructure devices (e.g. routers, firewalls, and proxies)
- Circumventing Deep Packet Injection (DPI)
- Bypassing firewall rules (like Palo Alto)
- Faking the "warp signature" to look to the world like a legitimate Apache server
- Masquerading as a simple Chrome, Firefox, Edge browser



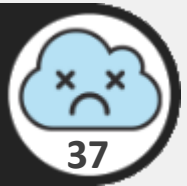
Lesson 5 – Advanced Options

- Navigate to /defcon/5_lesson
- Open main.cpp and modify the code as necessary
- Navigate to /defcon and run ./configure
- Navigate to /defcon/_bin/release and run ./5_lesson
- Obfuscation Techniques
  - Change the URL (requested resource)
  - Modify header parameters
  - Modify the session ID
  - Alternate between GET/POST requests as necessary
  - Change the way data is encoded (base64, hex, URL)
  - Divide up large transfers into small packets (i.e. chunking)



Summary

- Books
- Online resources
- Open source communities

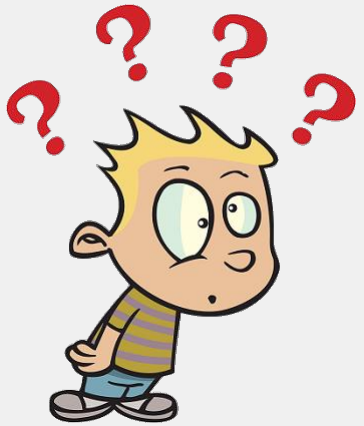


Questions?

www.demonsaw.com

eijah@demonsaw.com

@demon_saw



Eijah