Attacking Network Protocols

SIG Cybersecurity
Starting soon...

What is a Network?

- **Set of computes communicating with each other.**

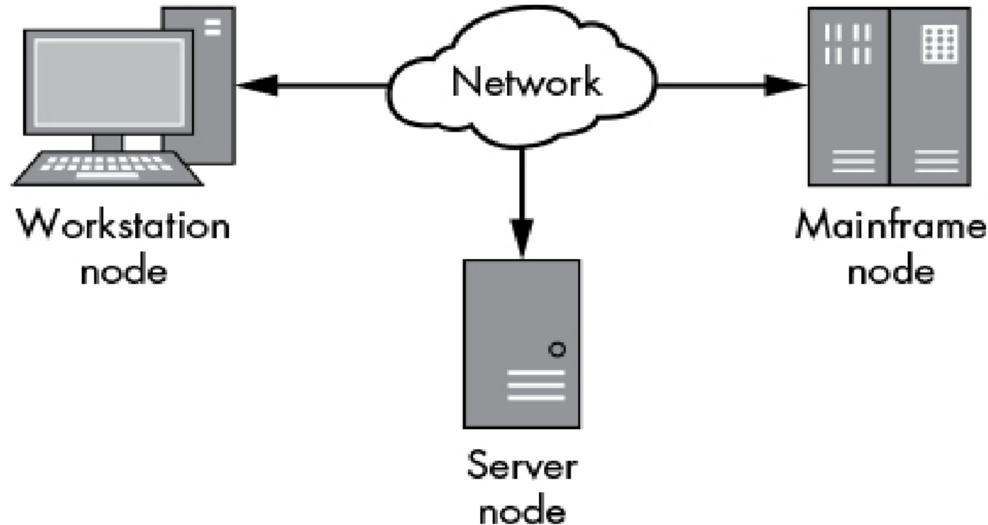
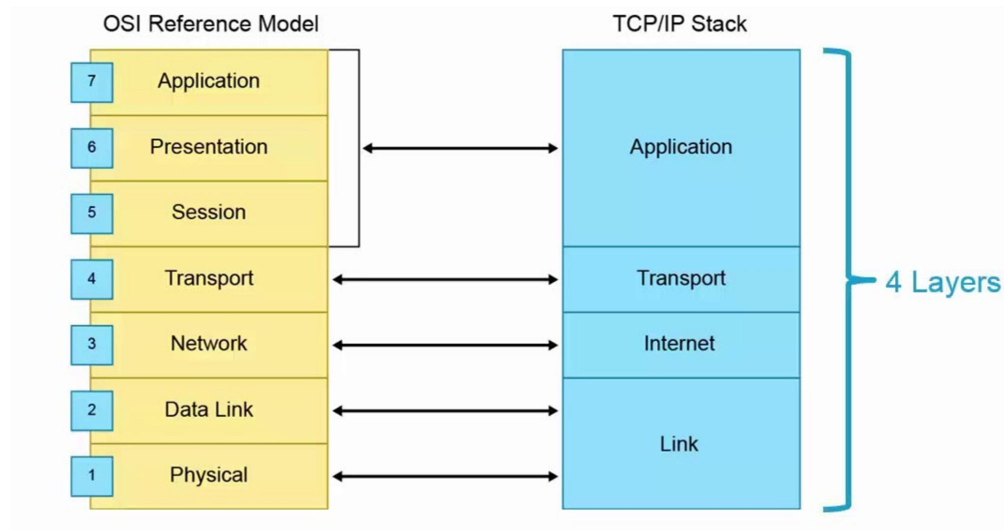


Figure 1-1: A simple network of three nodes

(can also have smartphones, laptops, etc. Also doesn't need to be wired)

You might be familiar with this stack

- (might be slightly different, if you have networking units just follow your lecture slides)
- We’ll focus on the Application layer, which is the payload by the programmer of an application.



IP Addresses

There are IPV4 and IPV6 IP Addresses.

- An IPV4 address might look like XXX.XXX.XXX.XXX, with each XXX being an 8 bit number.
- IPV6 was made when they realised 32 bits weren't enough for every internet-connected device in the world, so they transitioned to 128 bits, like 2001:0db8:85a3:0000:0000:8a2e:0370:7334 (btw we're using Hex now for compactness)

IP Addresses

There are local IP Addresses and public IP Addresses.

- Within your local network at home, each device in the network has an IP Address, e.g 192.168.0.XXX (can be other formats depending on your router). Usually you are assigned one IP Address by asking the DHCP server (usually part of your router).
- The public IP Address is by your router to communicate with the rest of the world.

IP Addresses

- Note: 127.0.0.1, also known as localhost, is a way to refer to “yourself” or home.

Port numbers

- Say you have a computer X with local IP 192.168.0.23 and you want to communicate with another computer Y on the same network with IP 192.168.0.55.
- Computer Y is already having a network connection active, perhaps browsing the Internet. What do we do?
- You can have multiple unique network connections simultaneously by using different *port* numbers (it is a 16 bit number)

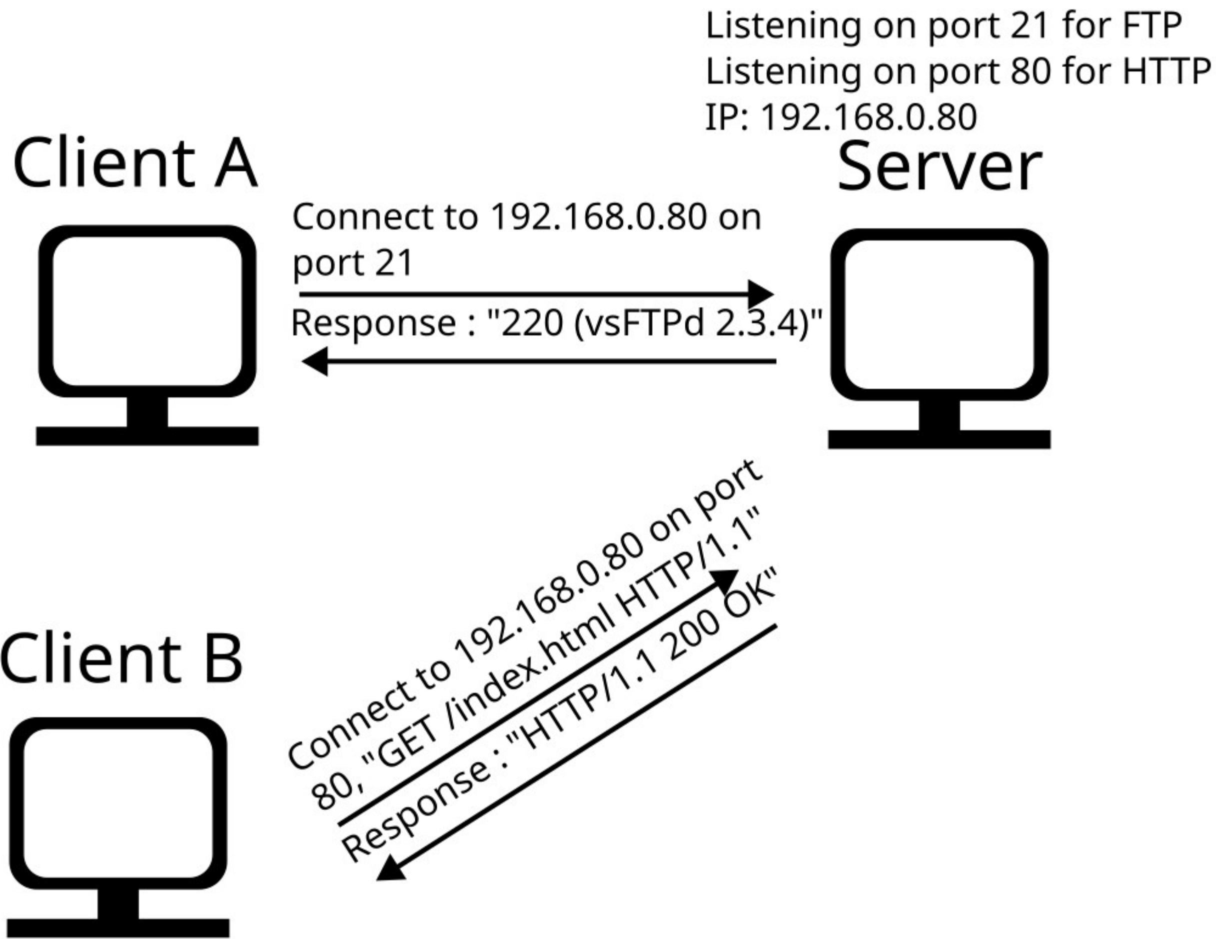
Port numbers

- Computer Y is a server on **IP** 192.168.0.55, *listening* on **port** 1234.
- Computer X connects to **IP** 192.168.0.55 on **port** 1234.
- They established a connection and can now communicate.

Port numbers

- Some port numbers are reserved for specific tasks.
- For example, port 21 is used for the File Transfer Protocol
- Port 80 is used for HTTP
- Port 443 is used for HTTPS
- (You can use ports 1-65535)

So they can send whatever data they want



Demo

- If this is confusing, this should hopefully clear things up. A short python demo of a "server" and a "client".
- Since I'm connecting to myself, I use "127.0.0.1" or "localhost" along with the port number of the server. Otherwise, if you want to connect from a separate device on the network, you need to find the IP address (You can find in Network Settings or ipconfig/ifconfig and looking for "IPV4 Address" or similar.)
- If you want to connect from the internet, you'll need to make sure your firewall is not blocking the port (You should be able to add an exception for either the program or the port number) and find out how to "port forward" on your router, which varies depending on the router. Generally, you'll have to figure out the IP Address of the "server" and enter that along with the port number into your router settings.

- The interpretation of the data they send is the protocol.
- If you decide that “if the first line says “LOGIN”, and the next line means the name, third line means the password”, that is a protocol you just made up.
- There are various protocols that are well-defined. HTTP, Websockets, FTP, etc. You can usually find an RFC to read up how the protocol is defined.

Byte representation

- A single memory address points to a single “byte”, which is 2^8 possible values. We’ll represent it in hex, since it has the nice property that the minimum is 00 and the max is FF.
- Data is not always sent as nice readable strings, so you can take a look at the hex dump of traffic and try to figure it out.

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

- **If you see hex numbers that are always around 0x20 to 0x7f range, it is likely it is readable ASCII text. Otherwise, it's probably either compacted data directly into the binary without "stringifying" it, or it is compressed and/or encrypted data.**
- **If you're analysing it, you can check out the entropy of the data. If it has a high entropy, it's likely compressed/encrypted.**

Endianness

- Say you have a number "1234" but you can only store 1 digit at each address in a computer. How would you store it?
- Address 0: 1, Address 1: 2, Address 2: 3, Address 3: 4
- Address 0: 4, Address 1: 3, Address 2: 2, Address 3: 1
- **The first method is called Big Endian, second is Little Endian. Some computers might use one or another (Except it's in base 16 instead of base 10, so it's each "hex digit" stored backward)**
- **Networking usually always uses Big Endian.**

Endianness

- **So if you have a number 0x12F, you will see it in a hex dump of the traffic as “01 2F”.**
- **Decimal numbers are a bit trickier. Usually I use python to unpack it.**

HTTP Protocol – An Application level protocol

- https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview
- https://rfcs.io/http
- DEMO
- Note: HTTP is stateless. Also it's mostly readable text.
- Note: The javascript won't execute ofc.
- Note: servers can respond differently depending on the headers, for example, some people block the python user agent.

Wireshark to sniff your network traffic.

- Note: There are other tools like Burp Suite or Fiddler designed specifically with analysing specific application level protocols like HTTP in mind.
- If you’re just interested in traffic from your web browser, there are developer tools in most browsers that let you see requests/responses.

Wireshark

attacks.pcap   File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help   13:33 09/07/2022

ip.addr == 192.168.37.128 and ftp

No.	Time	Source	Destination	Protocol	Length	Leftover Capture Data	Data
83075	61.671206	192.168.37.131	192.168.37.128	FTP	86		
83076	61.672352	192.168.37.131	192.168.37.128	FTP	86		
83079	61.674047	192.168.37.131	192.168.37.128	FTP	86		
83081	61.676839	192.168.37.131	192.168.37.128	FTP	86		
83083	61.681561	192.168.37.131	192.168.37.128	FTP	86		
83085	61.684215	192.168.37.131	192.168.37.128	FTP	86		
83087	61.686835	192.168.37.131	192.168.37.128	FTP	86		
83089	61.688644	192.168.37.131	192.168.37.128	FTP	86		
83091	61.691889	192.168.37.131	192.168.37.128	FTP	86		
83093	61.694206	192.168.37.131	192.168.37.128	FTP	86		

```
> Frame 83075: 86 bytes on wire (688 bits), 86 bytes captured (688 bits)
> Ethernet II, Src: VMware_b9:9a:36 (00:0c:29:b9:9a:36), Dst: VMware_6e:cb:16 (00:0c:29:6e:cb:16)
> Internet Protocol Version 4, Src: 192.168.37.131, Dst: 192.168.37.128
> Transmission Control Protocol, Src Port: 21, Dst Port: 41314, Seq: 1, Ack: 1, Len: 20
> File Transfer Protocol (FTP)
  [Current working directory: ]
```

```
0000  00 0c 29 6e cb 16 00 0c  29 b9 9a 36 08 00 45 00   ··)n···· )··6··E·
0010  00 48 0d 0b 40 00 40 06  61 51 c0 a8 25 83 c0 a8   ·H··@·@· aQ··%···
0020  25 80 00 15 a1 62 f0 7e  7c 46 ac 0f 4c 33 80 18   %····b·~ |F··L3··
0030  00 b5 d3 6b 00 00 01 01  08 0a 00 00 71 4d 9f 86   ···k···· ····qM··
0040  61 c9 32 32 30 20 28 76  73 46 54 50 64 20 32 2e   a·220 (v sFTPd 2.
0050  33 2e 34 29 0d 0a                                  3.4)··
```

Bytes 66-85: Text item (text)   Packets: 234384 · Displayed: 1278 (0.5%)   Profile: Default

Heres a pcap I found online

- I said earlier we're only interested in the Application level. The text before it are from the other levels. (If you use wireshark on HTTP server earlier, you will see other things before the data you sent too)

```
0000  00 0c 29 6e cb 16 00 0c  29 b9 9a 36 08 00 45 00   ··)n····  )··6··E·
0010  00 48 00 bf 40 00 40 06  6d 9d c0 a8 25 83 c0 a8   ·H··@·@·  m···%···
0020  25 80 00 15 a1 80 f0 88  71 f2 3d 6c 63 48 80 18   %·······  q·=lcH··
0030  00 b5 35 22 00 00 01 01  08 0a 00 00 71 50 9f 86   ··5"····  ····qP··
0040  61 ca 32 32 30 20 28 76  73 46 54 50 64 20 32 2e   a·220 (v  sFTPd 2.
0050  33 2e 34 29 0d 0a                                  3.4)··
```

- **These are definitely in plain text, you can Google it.**

```
0000  00 0c 29 6e cb 16 00 0c  29 b9 9a 36 08 00 45 00   ··)n····  )··6··E·
0010  00 56 6f 56 40 00 40 06  fe f7 c0 a8 25 83 c0 a8   ·VoV@·@·  ····%···
0020  25 80 00 15 a1 6e f0 72  1e 2d 40 64 3f 94 80 18   %····n·r  ·-@d?···
0030  00 b5 be 2b 00 00 01 01  08 0a 00 00 71 70 9f 86   ···+····  ····qp··
0040  63 2a 33 33 31 20 50 6c  65 61 73 65 20 73 70 65   c*331 Pl ease spe
0050  63 69 66 79 20 74 68 65  20 70 61 73 73 77 6f 72   cify the  passwor
```

https://en.wikipedia.org/wiki/List_of_FTP_server_return_codes

250	Requested file action okay, completed.
257	"PATHNAME" created.
300 Series	**The command has been accepted, but the reques**
331	User name okay, need password.
332	Need account for login.
350	Requested file action pending further information

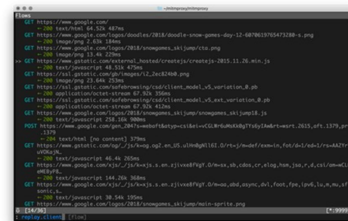
Sniffing from other devices

- You probably won't always be able to install programs on other devices, e.g: If you're reverse engineering an IoT device and want to see what data it sends over the network.
- One way to still sniff traffic is through configuring a proxy.
- Device goes through your laptop as a proxy, which does a Man-In-The-Middle, and sniffs the traffic, before forwarding it towards the intended recipient, and vice versa.

mitmproxy

- mitmproxy is a free and open source interactive HTTPS proxy.
- https://mitmproxy.org/

Command Line | Web Interface | Python API

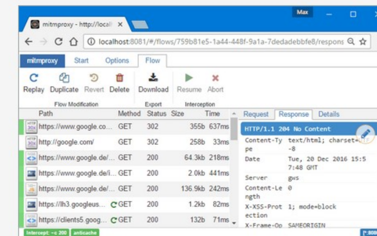


Command Line

mitmproxy is your swiss-army knife for debugging, testing, privacy measurements, and penetration testing. It can be used to intercept, inspect, modify and replay web traffic such as HTTP/1, HTTP/2, WebSockets, or any other SSL/TLS-protected protocols. You can prettify and decode a variety of message types ranging from HTML to Protobuf, intercept specific messages on-the-fly, modify them before they reach their destination, and replay them to a client or server later on.

Web Interface

Use mitmproxy's main features in a graphical interface with mitmweb. Do you like Chrome's DevTools? mitmweb gives you a similar experience for any other application or device, plus additional features such as request interception and replay.



addon.py

```python
from mitmproxy import http

def request(flow: http.HTTPFlow):
    # redirect to different host
    if flow.request.pretty_host == "example.com":
        flow.request.host = "mitmproxy.org"
    # answer from proxy
    elif flow.request.path.endswith("/brew"):
        flow.response = http.Response.make(
            418, b"I'm a teapot",
        )
```

Python API

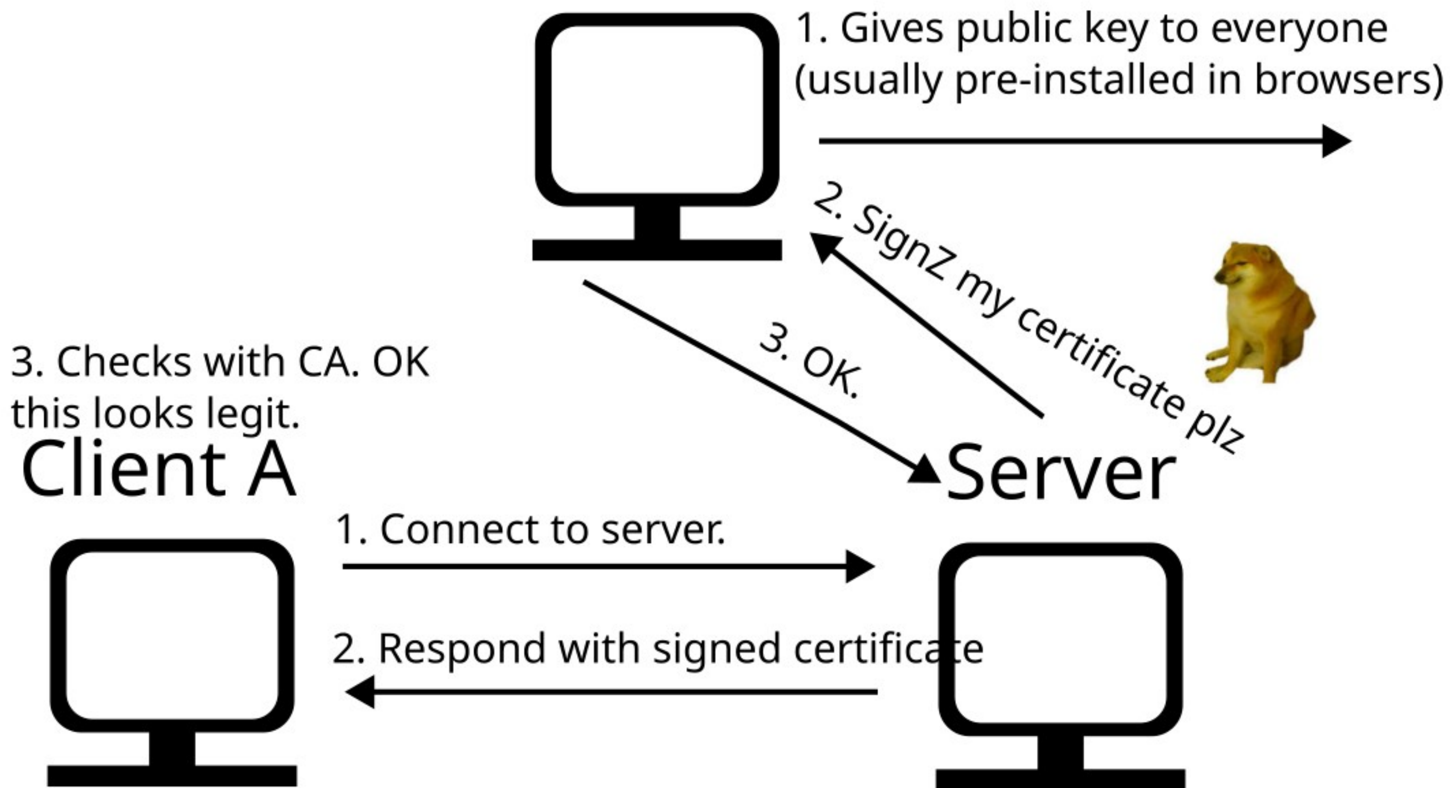
Write powerful addons and script mitmproxy with mitmdump. The scripting API offers full control over mitmproxy and makes it possible to automatically modify messages, redirect traffic, visualize messages, or implement custom commands.

mitmproxy

- Usually all you have to do is run the mitmproxy program, and then on your device, you need to find a setting to configure a network setting for a proxy server, and set it to your laptop’s IP and port number (default is 8080).

Problems

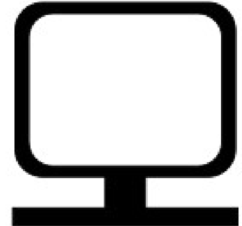
- **If the IoT device doesn't support configuring proxies, you're out of luck.**
  - You might have some luck configuring your laptop as a Wi-Fi hotspot though and having your target device connect to that, then sniff using Wireshark.
- **If the traffic is using SSL/TLS, you won't be able to do a MiTM attack because the device checks against a trusted Certificate Authority to make sure it's really them with some cryptography, and does encryption. (Sniffing SSL traffic is a problem even without a proxy btw)**


Certificate Authority
1. Gives public key to everyone (usually pre-installed in browsers)
2. SignZ my certificate plz
3. OK.
3. Checks with CA. OK this looks legit.
Client A
Server
1. Connect to server.
2. Respond with signed certificate

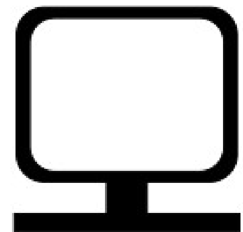
TLS/SSL Traffic

- Mitmproxy solves this by creating a “rogue” Certificate Authority. You’ll have to configure the target device to trust this Certificate Authority somehow, by installing a CA cert on the device.
- Google is your friend. Mitmproxy has a very easy-to-follow guide for common devices as well.

Rogue Certificate Authority



Certificate Authority

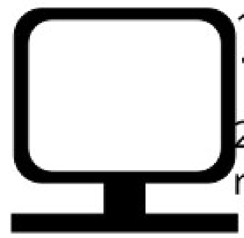


1. Gives public key to everyone (usually pre-installed in browsers)

3. Checks with rogue CA. OK this looks legit.

3. Checks with CA. OK this looks legit.

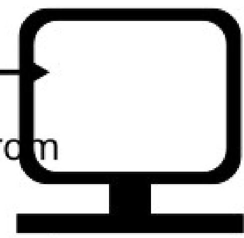
Client



1. Connect to server.

2. Respond with signed from rogue CA certificate

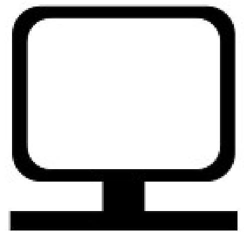
Proxy



1. Connect to server.

2. Respond with signed certificate from CA.

Server



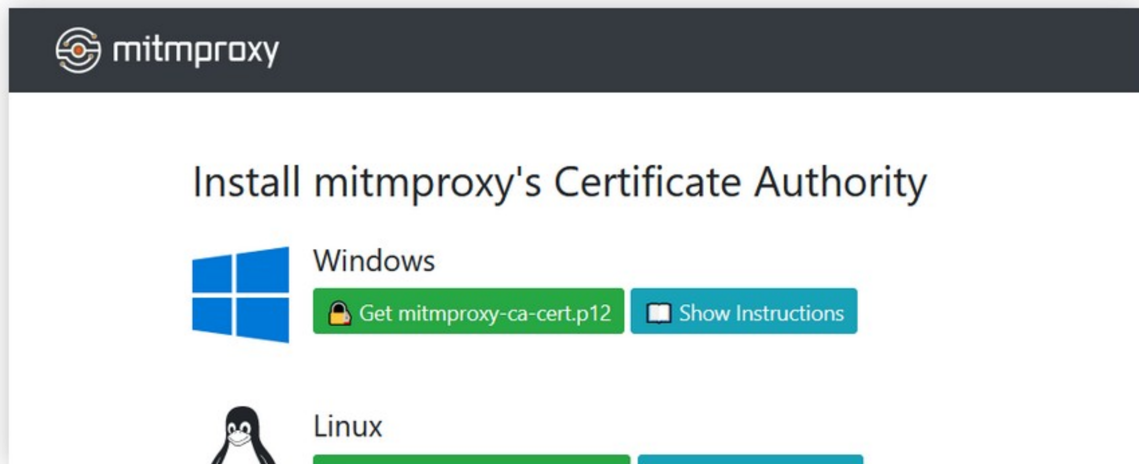
Edit on GitHub

About Certificates

Mitmproxy can decrypt encrypted traffic on the fly, as long as the client trusts mitmproxy's built-in certificate authority. Usually this means that the mitmproxy CA certificate has to be installed on the client device.

Quick Setup

By far the easiest way to install the mitmproxy CA certificate is to use the built-in certificate installation app. To do this, start mitmproxy and configure your target device with the correct proxy settings. Now start a browser on the device, and visit the magic domain mitm.it. You should see something like this:



Click on the relevant icon, follow the setup instructions for the platform you're on and you are good to go.

Examples

- For browsers like Firefox/Chrome, usually it's in the settings page somewhere.
- You might also be able install it to your device itself in some Settings option. You might need to be an admin
- For phones, for current versions of Android and iPhones, you need a rooted/jailbroken device to sniff TLS/SSL traffic.

- BTW, some organisations and workplaces/schools install these CA certs on your work devices to monitor your network, even if you use TLS/SSL they can see what you’re doing.
- DNS requests (use to query to convert website addresses like “http://www.example.com” to an IP Address you can connect to) still sometimes don’t use TLS to encrypt, so even if you connect to a public hotspot they might know what sites you visit.

Demo

- Mitmproxy demo + short tutorial

Mitmproxy addons

- https://docs.mitmproxy.org/stable/addons-overview/
- Pretty powerful. You can create your own scripts to do what you want.
- DEMO

Other Ways to sniff traffic

- When Reverse Engineering, we can split it into “Static Analysis” and “Dynamic Analysis”
- Dynamic is analysis during runtime
- Static is analysis without running the code
- Note: If you’re analysing malware, you might want to avoid using dynamic analysis.

Other Ways to sniff traffic

- You can inject code to print out the traffic when its sent/retrieved.

  https://confused.ai/posts/intercepting-zoom-tls-encryption-bpf-uprobes

https://confused.ai/posts/intercepting-zoom-tls-encryption-bpf-uprobes

Confused AI

twitter.com/alessandrod github.com/alessandrod

Putting it all together

We've written uprobes for `SSL_read`, `SSL_write`, `connect` and `getaddrinfo`. With them we can see what DNS queries the zoom client does, what addresses it connects to and what encrypted data it sends and receives.

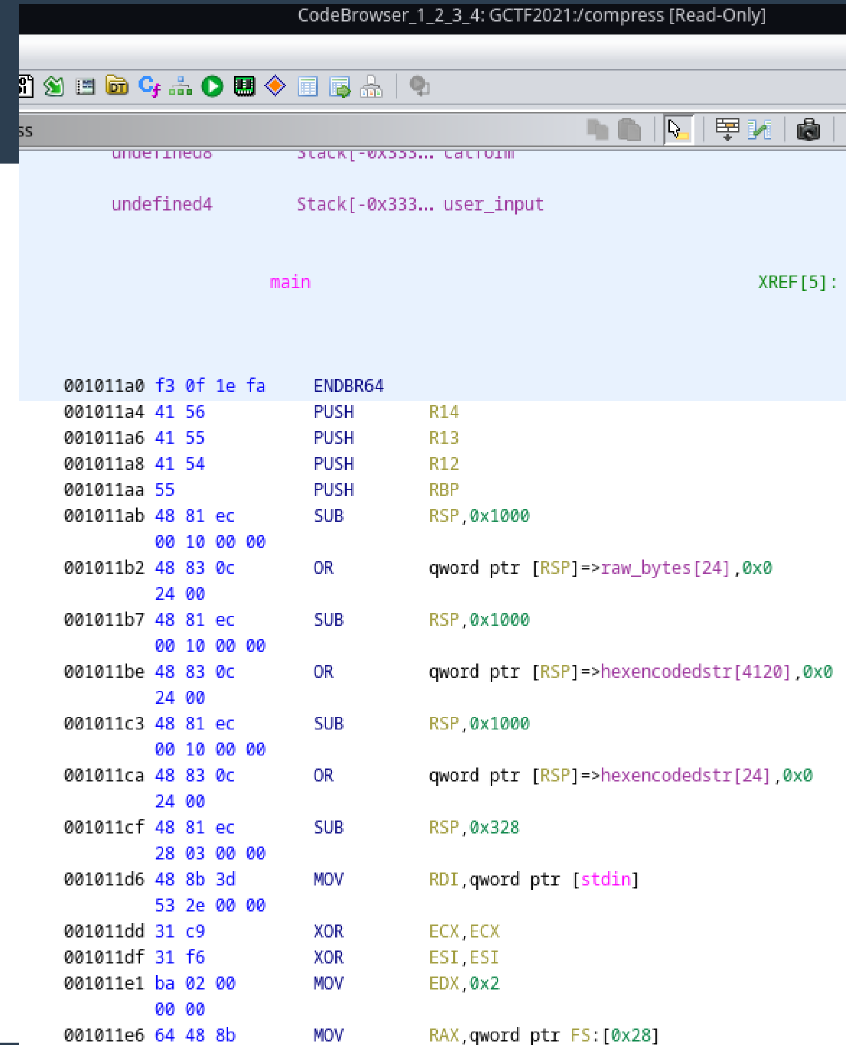
The final output looks like this:

```
$ sudo target/debug/snuffy --hex-dump --trace-connections --command /opt/zoom/zoom --offsets z
[4:56:18] Connected to 127.0.0.53:53
[4:56:18] Resolved us04web.zoom.us to 3.235.69.6
[4:56:18] Connected to us04web.zoom.us:443 (3.235.69.6:443)
[4:56:19] Write 571 bytes to us04web.zoom.us:443 (3.235.69.6:443)
[4:56:19] |504f5354 202f7265 6c656173 656e6f74| POST /releasenot 00000000
[4:56:19] |65732048 5454502f 312e310d 0a486f73| es HTTP/1.1..Hos 00000010
[4:56:19] |743a2075 73303477 65622e7a 6f6f6d2e| t: us04web.zoom. 00000020
[4:56:19] |75730d0a 55736572 2d416765 6e743a20| us..User-Agent:  00000030
[4:56:19] |4d6f7a69 6c6c612f 352e3020 285a4f4f| Mozilla/5.0 (ZOO 00000040
[4:56:19] |4d2e4c69 6e757820 5562756e 74752031| M.Linux Ubuntu 1 00000050
...

[4:56:19] Read 3088 bytes from us04web.zoom.us:443 (3.235.69.6:443)
[4:56:19] |48545450 2f312e31 20323030 200d0a44| HTTP/1.1 200 ..D 00000000
[4:56:19] |6174653a 20467269 2c203034 20536570| ate: Fri, 04 Sep 00000010
[4:56:19] |20323032 30203035 3a31313a 30352047|  2020 05:11:05 G 00000020
[4:56:19] |4d540d0a 436f6e74 656e742d 54797065| MT..Content-Type 00000030
[4:56:19] |3a206170 706c6963 6174696f 6e2f782d| : application/x- 00000040
[4:56:19] |70726f74 6f627566 3b636861 72736574| protobuf;charset 00000050
..
```

Other Ways to sniff traffic

- You can decompile/disassemble the program using a decompiler such as IDA Pro, Ghidra, etc.



Other Ways to sniff traffic

- For interpreted languages like C# and Java there are decompilers like JD-GUI
  and ILSpy

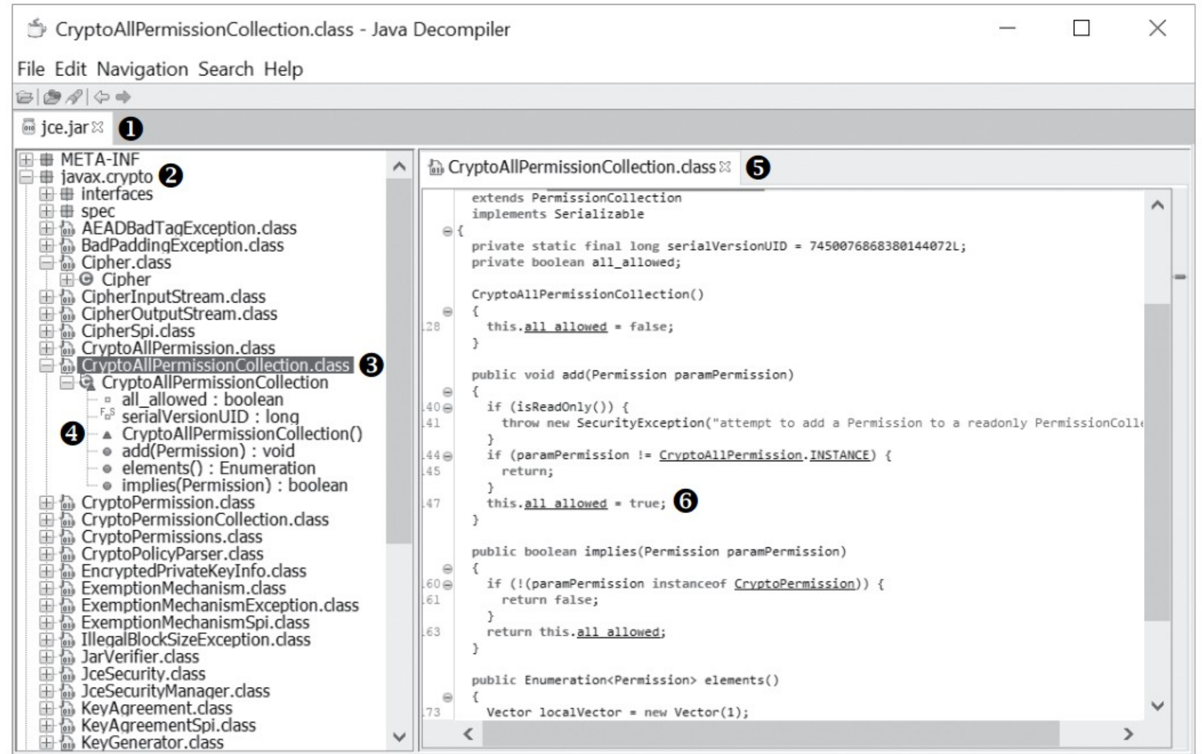


Figure 6-22: JD-GUI with an open JAR File

Other Ways to sniff traffic

- **If you decompile/disassemble it, you just need to read through the code. Depending on how much they *obfuscated* it, (i.e: how much they did to make it as hard as possible to read when decompiled), this can be as easy as just reading normal code and figuring out what it does, or as hard as spending many days trying to make sense of it and figure out what it does.**

Serialisation/Deserialisation

- When we send data over the network, we occasionally want to convert it back into code.
- E.g: Say we have a class for a Character in an MMO, along with its position, rotations, etc in the World. Say we have an object of "Character player1;" and want to send it over the network. We'll want to *serialise* this object into some binary data and send it over the network.
- At the receiving end, we have to *deserialise* this data and turn it back into a class. This deserialisation can be tricky and is one point of attack.

Serialisation/Deserialisation

- For example, perhaps the Character is serialised to a format like “CharacterN<CharacterName>X<Xcoord>Y<Ycoord>Z<Zcoord>”.
- Perhaps during deserialisation, it assumes the character name is less than 20 characters, and overloading it causes a *buffer overflow*. Perhaps Not specifying an X coordinate will crash the server. Etc. Play around!

Application weaknesses

- Sometimes servers rely too much on the client-side checking. By messing with the network protocol directly, you can bypass any client-side checks by sending the data directly.
- E.g: Perhaps there is a check in a game for to prevent sending “/ban player” if you’re not an admin. If they solely relied on the client-side check and the server doesn’t do any check, you can modify the game or send your own data

  to “ban” the player directly.

Root causes of flaws

- Standard binary exploitation stuff, buffer overflows, e.g: You send a packet that is longer than the server program expects and overrun the stack, out-of-bounds buffer indexing where you “ask” for the 10th index of an array that only has 5 elements in it, letting you read some memory, etc.

Sources

- Attacking Network Protocols (No Starch Press)
- Practical Binary Analysis (No Starch Press)
- MITMProxy documentation
- https://realpython.com/python-sockets/
- https://beej.us/guide/bgnet/html/
- https://github.com/gracenolan/Notes/blob/master/interview-study-notes-for-security-engineering.md#networking

Q & A