

Project

...

Man In The Middle Attacks: Execution and Detection

Daria Vaskovskaya

Link to github

<https://github.com/BananaAndBread/Project/tree/master>



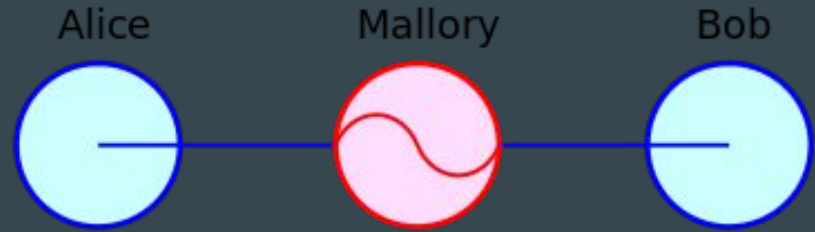
- presentations
 - patches
 - reports
 - scripts
 - readme
-

Agenda

- What is MITM?
- Project goals
- Work done during the iterations

MITM

Man-in-the-middle attack (MITM) is an attack where the attacker secretly relays and possibly alters the communications between two parties who believe that they are directly communicating with each other



Initial goal:

- Study ways of execution and detection of Man In The Middle Attacks.
- Set up a simulation platform
- Run various scripts that demonstrate performing such attacks, detecting them and circumventing that detection.



How to detect Mallory AND be as silent as possible as Mallory to bypass those detections

Iteration I

Iteration I

Goal:

- Study different variants of simulation platforms
- Set up the environment for Man In The Middle Attacks
- Perform some MITM attacks on simulation platforms

Study different variants of simulation platforms

Issues

- Lack of knowledge
- Lack of documentation

Choices:

- VirtualBox - flexibility, my experience with it, and community support.
- mininet - good documentation and simplicity

During 1st iteration stopped on VirtualBox



VirtualBox is used now

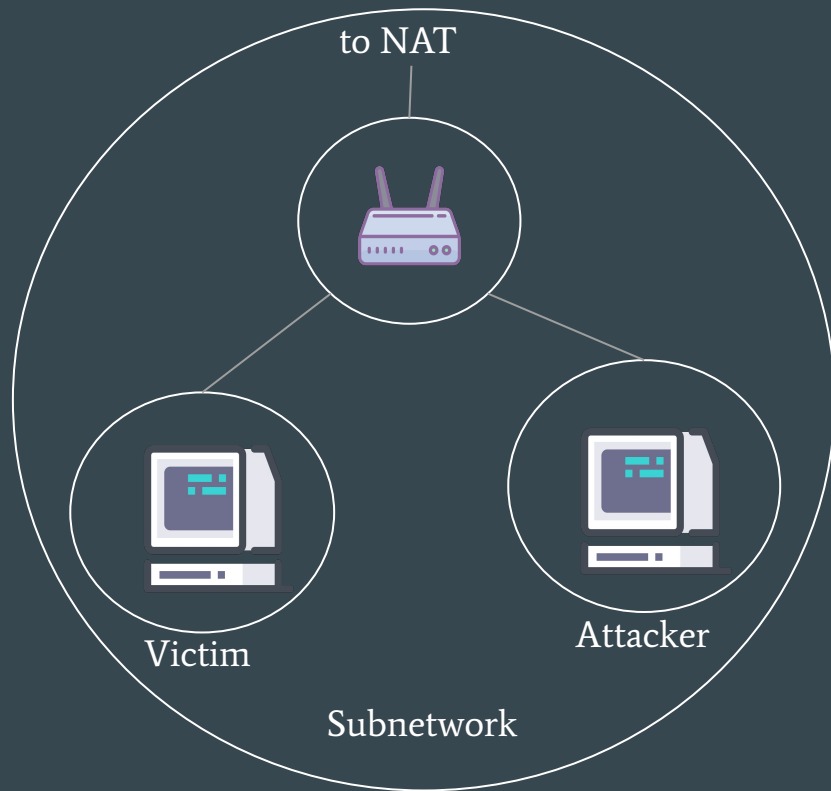
Finally: Platform must be really good at browser simulation. A huge opening was vagrant, however it was not enough time to study it properly. Now ova is available on Google Drive:
<https://drive.google.com/file/d/lit9XutX5brvca6FAJa4zgMAfTiEu5Lxa/view?usp=sharing>

Set up the environment

Created script to set up one machine as a router and provide internet to connected machines.

- Quagga - a network routing software suite
- DHCP

Used even now.



Play with MITM:

MITM Framework choice - Bettercap

Why bettercap?

Based on:

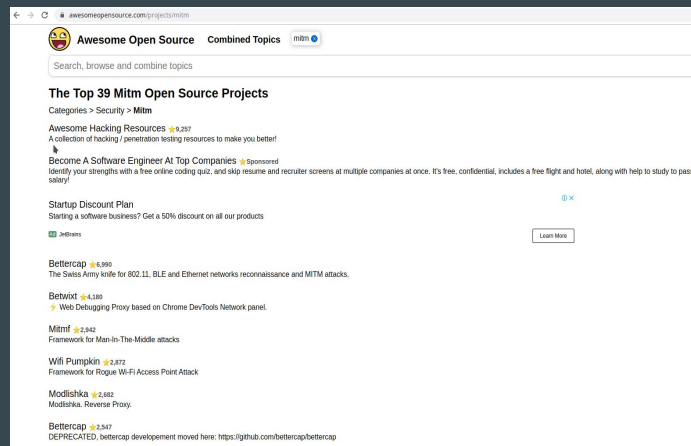
- Most recent commit
- Stars
- Number of features

Based on info got from:

<https://awesomeopensource.com/projects/mitm>

My table with comparison:

[Google Doc](#)



<https://awesomeopensource.com/projects/mitm>, accessed February 25

Name	Stars	Description	More info	Most recent commit	Comments
Bettercap	★6,911	The Swiss Army knife for 802.11, BLE and Ethernet networks reconnaissance and MITM attacks.	https://awesomeopensource.com/projects/mitm/bettercap	5 days ago	
Betwixt	★4,171	Web Debugging Proxy based on Chrome DevTools Network panel.	https://awesomeopensource.com/projects/mitm/betwixt	a year ago	Not for my purposes
Mitmf	🔥2,342	Framework for Man-In-The-Middle attacks	https://awesomeopensource.com/projects/mitm/mitmf	3 user ago	On this

Perform some MITM attacks on simulation platforms

ARP-spoofing

The screenshot displays a Linux Lite terminal window and a Wireshark network traffic capture. The terminal window shows the configuration of a network interface (enp0s3) for ARP-spoofing. The Wireshark window shows the captured traffic, including an ICMP Echo (ping) request and a TCP Reset (RST) packet, indicating a successful MITM attack.

Linux Lite Terminal:

```
valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq
group default qlen 1000
    link/ether 08:00:27:07:c5:56 brd ff:ff:ff:ff:ff:ff
    inet 192.168.50.50/24 brd 192.168.50.255 scope global dynamic
        valid_lft 530sec preferred_lft 530sec
    inet6 fe80::3e4f:47f4:705b:80a3/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
toor ➜ curl google.com
<HTML><HEAD><meta http-equiv="content-type" content="text/html
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="http://www.google.com/">here</A>.
</BODY></HTML>
toor ➜ curl google.com
<HTML><HEAD><meta http-equiv="content-type" content="text/html
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
```

Wireshark Network Traffic:

No.	Time	Source	Destination	Protocol	Length	Info
4	1.261778463	192.168.50.51	192.168.50.50	ICMP	102	Echo (ping) request
5	1.261800607	192.168.50.50	216.239.38.120	TCP	74	59162 → 80 [RST] Seq=1286561000
6	1.301380629	192.168.50.50	216.239.38.120	TCP	60	59162 → 80 [RST] Seq=1286561000
7	1.301409894	192.168.50.50	216.239.38.120	TCP	54	59162 → 80 [RST] Seq=1286561000
8	1.301437238	192.168.50.50	216.239.38.120	HTTP	128	GET / HTTP/1.1\r\nHost: google.com\r\nUser-Agent: curl/7.58.0\r\nAccept: */*\r\n

Wireshark Details:

- Internet Protocol Version 4, Src: 192.168.50.50, Dst: 216.239.38.120
- Transmission Control Protocol, Src Port: 59162, Dst Port: 80, Seq: 1, Ack: 1, Len: 74
- Hypertext Transfer Protocol
 - GET / HTTP/1.1\r\nHost: google.com\r\nUser-Agent: curl/7.58.0\r\nAccept: */*\r\n

Iteration II

II Iteration

Decision:

- Detect the presence of MITM attack **regardless of the mechanism used to launch it**

Goals:

- Read the papers about MITM detection
- Make short summaries about the noteworthy ones
- Pick paper/papers to investigate further during the 3rd iteration

II Iteration results

Noteworthy papers:

- Vesper: Using Echo-Analysis to Detect Man-in-the-Middle Attacks in LANs [
- The Security Impact of HTTPS Interception
- Detection of MITM Attack in LAN Environment using Payload Matching
- Client-Side Web Proxy Detection from Unprivileged Mobile Devices

Selected:

"The Security Impact of HTTPS Interception"

- Can not detect proxies which closely mimic the request of the client
- Can add this feature to proxy to bypass detection

Iteration III

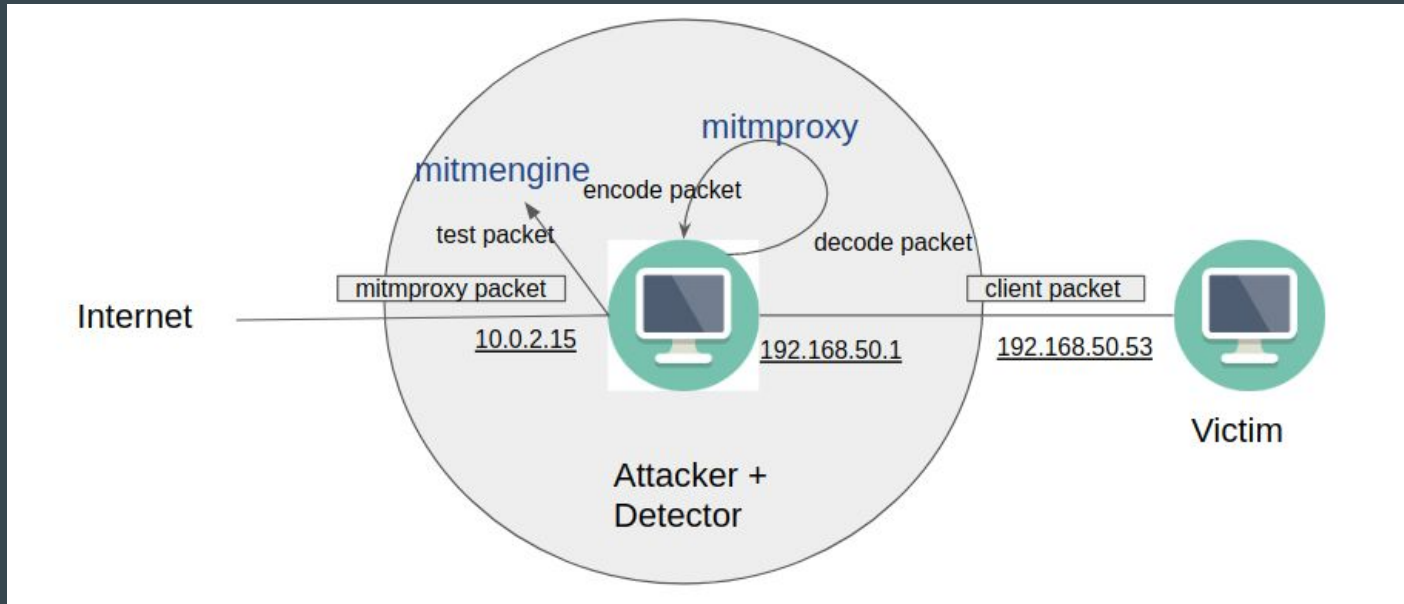
III Iteration

Goal:

Test and experiment with detection tool based on "The Security Impact of HTTPS Interception" paper - mitmengine

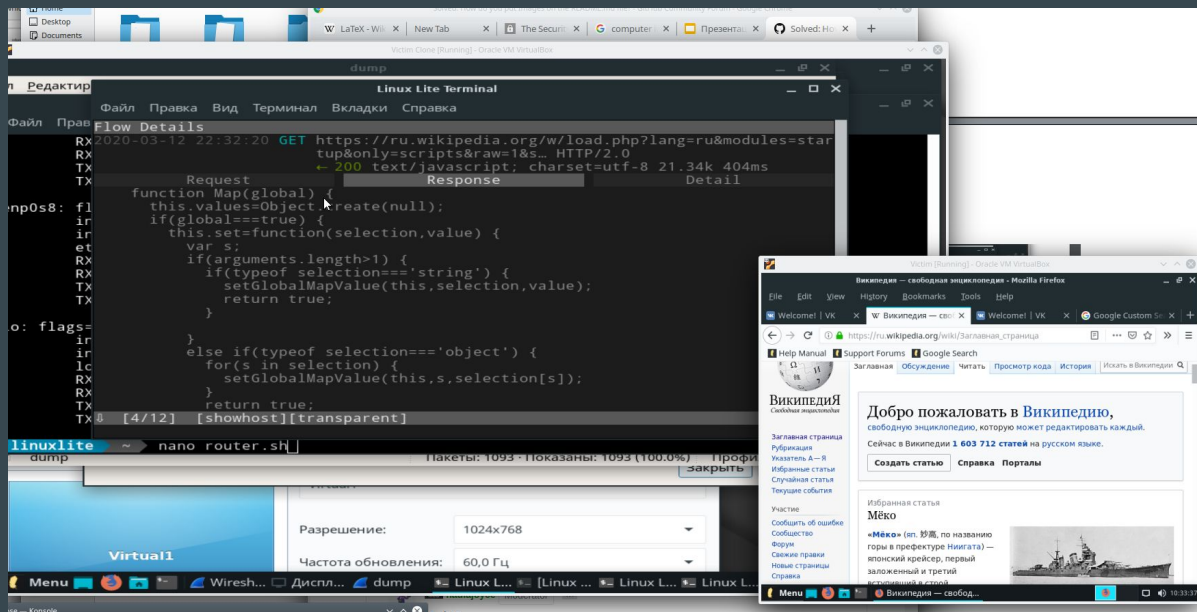
III Iteration

- Set-up:



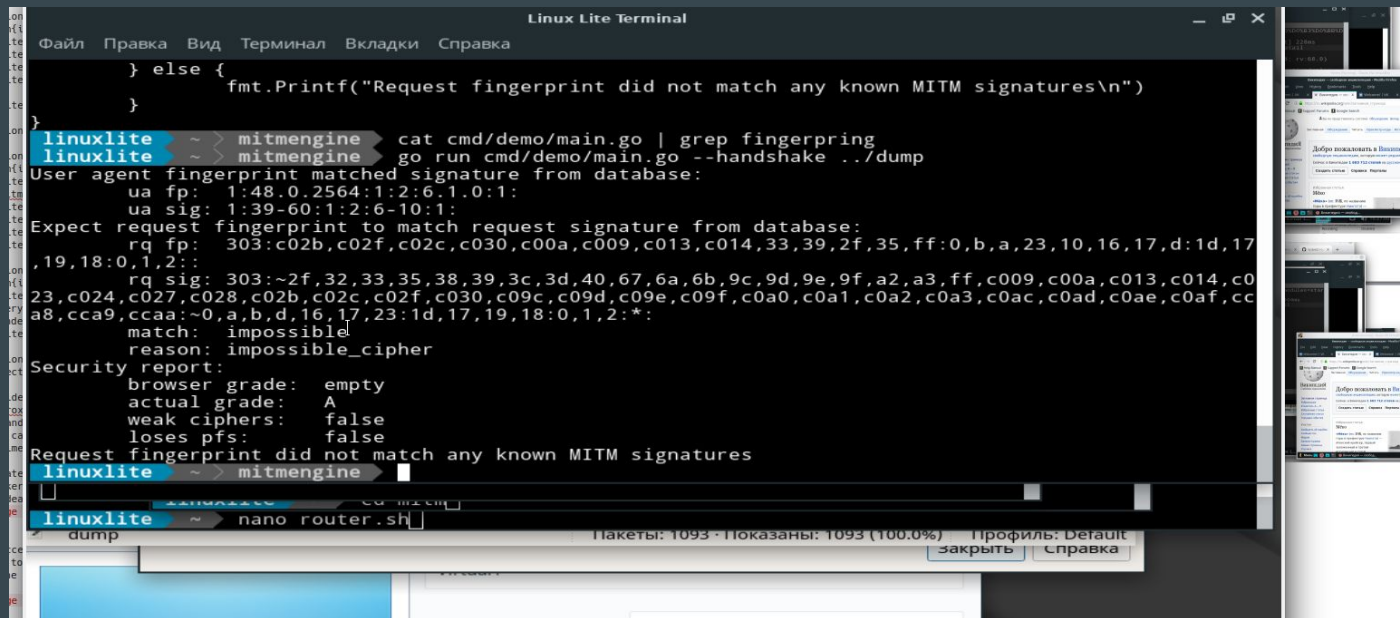
III Iteration

- Installed and ran mitmproxy to sniff HTTPs packets



III Iteration

- Installed detection tool in order to catch man in the middle
- Ran tests



```
Linux Lite Terminal
Файл Правка Вид Терминал Вкладки Справка

    } else {
        fmt.Printf("Request fingerprint did not match any known MITM signatures\n")
    }
}

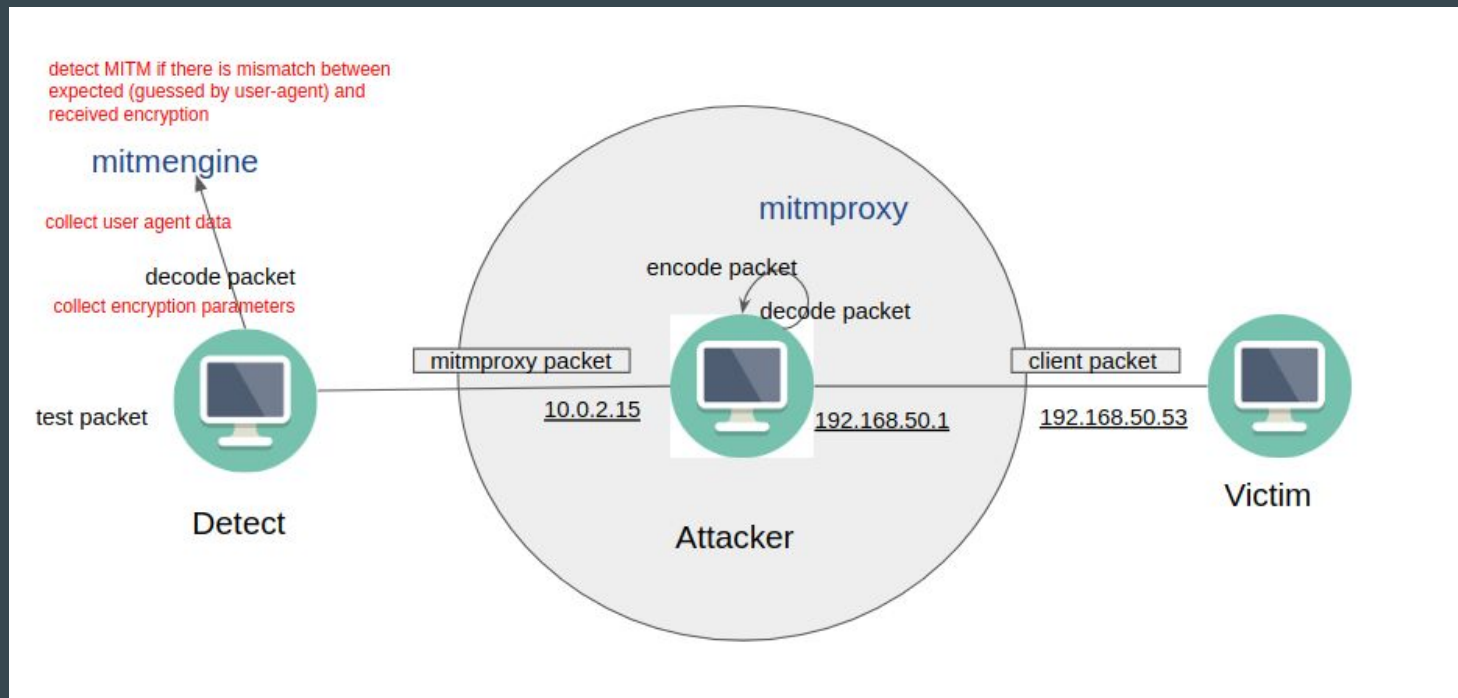
linuxlite ~ > mitengine cat cmd/demo/main.go | grep fingerpring
linuxlite ~ > mitengine go run cmd/demo/main.go --handshake ../dump
User agent fingerprint signature from database:
ua fp: 1:48.0.2564:1:2:6.1.0:1:
ua sig: 1:39-60:1:2:6-10:1:
Expect request fingerprint to match request signature from database:
rq fp: 303:c02b,c02f,c02c,c030,c00a,c009,c013,c014,33,39,2f,35,ff:0,b,a,23,10,16,17,d:1d,17,19,18:0,1,2:~
rq sig: 303:~2f,32,33,35,38,39,3c,3d,40,67,6a,6b,9c,9d,9e,9f,a2,a3,ff,c009,c00a,c013,c014,c023,c024,c027,c028,c02b,c02c,c02f,c030,c09c,c09d,c09e,c09f,c0a0,c0a1,c0a2,c0a3,c0ac,c0ad,c0ae,c0af,cca8,cca9,ccaa:~0,a,b,d,16,17,23:1d,17,19,18:0,1,2:*:
match: impossible
reason: impossible_cipher
Security report:
browser grade: empty
actual grade: A
weak ciphers: false
loses pfs: false
Request fingerprint did not match any known MITM signatures
linuxlite ~ > mitengine
linuxlite ~ > nano router.sh
dump
Пакеты: 1093 - Показаны: 1093 (100.0%) Профиль: Default
[заккрыть] [справка]
```

Iteration IV

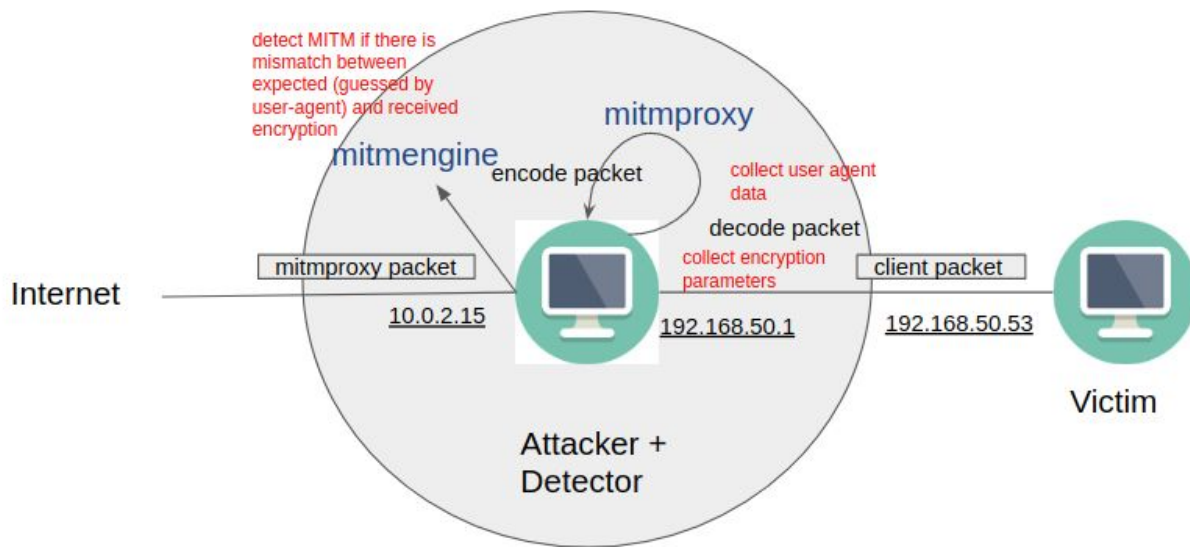
Iteration IV

- Digged into the mitmengine code
- Better understood how it works, especially input and output formats
- Made scripts for testing mitmengine with and without MITM
- Tested mitmengine with those scripts

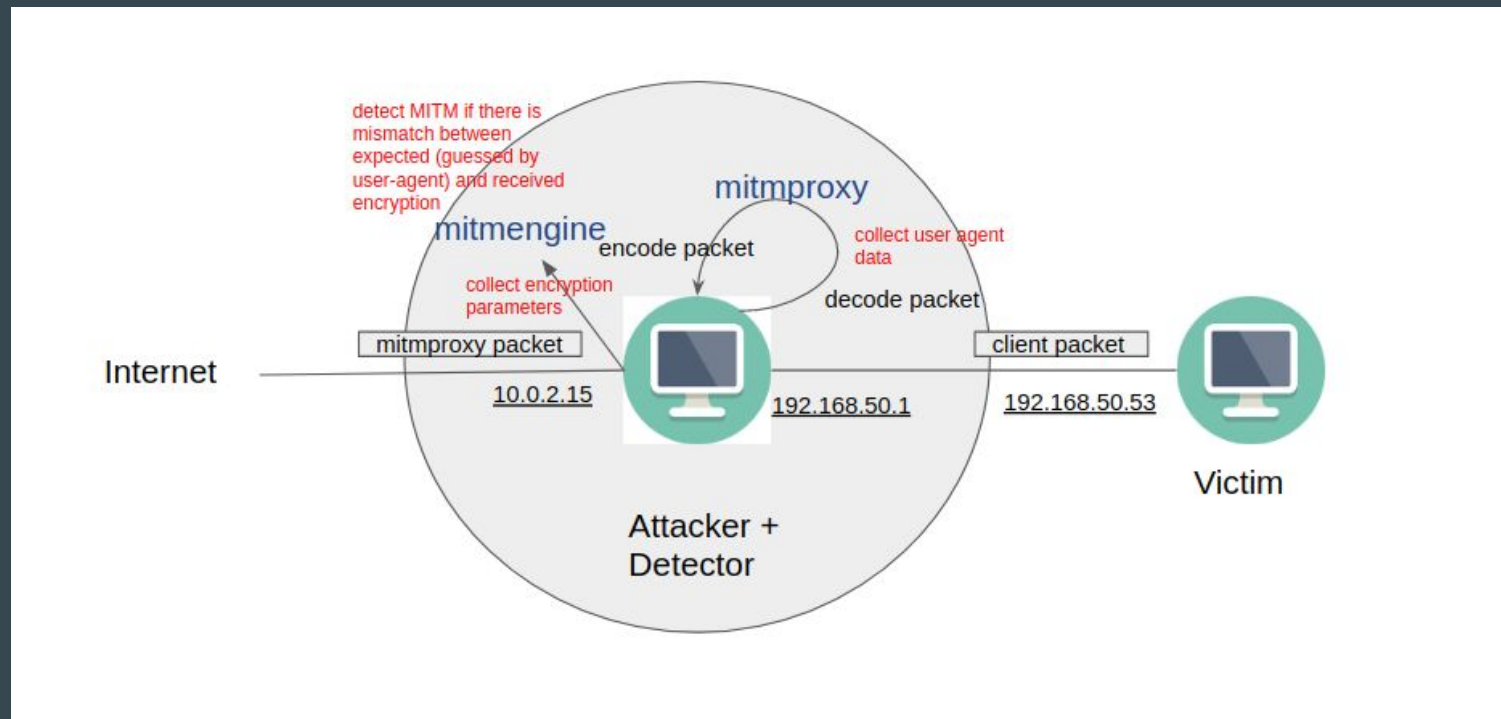
In real life:



My scheme: without MITM



My scheme: with MITM



Results

```
my output
Mozilla/5.0 (X11; Linux x86_64; rv:62.0) Gecko/20100101 Firefox/62.0BrowserFirefox
62.0.0
User agent fingerprint matched signature from database:
  ua fp: 4:62.0.0:3:10:0.0.0:1:
  ua sig: 4:62:3:10:0:1:
Expect request fingerprint to match request signature from database:
  rq fp: 303:1301,1303,1302,c02b,c02f,cca9,cca8,c02c,c030,c013,c014,2f,35,a:0,17,ff01,a,b,23,10,5,33
0::
  rq sig: 303:cca8,cca9,c02f,c030,c02b,c02c,c013,c009,c014,c00a,9c,9d,2f,35,c012,a:0,5,a,b,d,ff01,12:
  match: impossible
  reason: impossible_cipher
Security report:
  browser grade: A
  actual grade: A
  weak ciphers: false
  loses pfs: false
Request fingerprint did not match any known MITM signatures
root / > home > linuxlite > Downloads > iptables -S
```

```
my output
Mozilla/5.0 (X11; Linux x86_64; rv:62.0) Gecko/20100101 Firefox/62.0BrowserFirefox
62.0.0
User agent fingerprint matched signature from database:
  ua fp: 4:62.0.0:3:10:0.0.0:1:
  ua sig: 4:62:3:10:0:1:
Expect request fingerprint to match request signature from database:
  rq fp: 303:c02b,c02f,c02c,c030,c013,c014,2f,35,ff:0,b,a,23,10,16,17,d:1d,17,19,18:0,1,2::
  rq sig: 303:cca8,cca9,c02f,c030,c02b,c02c,c013,c009,c014,c00a,9c,9d,2f,35,c012,a:0,5,a,b,d,ff01,12:1d,17,18,19:0:*:
  match: impossible
  reason: impossible_cipher
Security report:
  browser grade: A
  actual grade: A
  weak ciphers: false
  loses pfs: true
Request fingerprint did not match any known MITM signatures
```

Results explanation

Mitmengine still does not show expected output. It detects MITM in both cases

Mitmengine did not know enough signatures

Iteration V

Iteration V

- Read nearly ALL of the mitmengine code
- Tried to fix mitmengine code in order to make it work with newer software
- Tried to make environment the same with one developers had
- Finally, added signature manually

Results:

Without MITM:

```
ua sig: 4:76:0:0:3:10:0:0:1:
Expect request fingerprint to match request signature from database:
rq fp: 303:1301,1303,1302,c02b,c02f,cca9,cca8,c02c,c030,c00a,c009,c013,c014,33,39,2f,35,a:0,17,ff01,a,b,10,5,33,2b,d,1c,15:1d,17,18,1
9,100,101:0::
rq sig: 303:1301,1303,1302,c02b,c02f,cca9,cca8,c02c,c030,c00a,c009,c013,c014,33,39,2f,35,a:0,17,ff01,a,b,10,5,33,2b,d,1c,15:1d,17,18,1
9,100,101:0::
match: possible
Security report:
```

With MITM:

```
My output
Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.138 Safari/537.36BrowserChrome
:81.0.4044
User agent fingerprint matched signature from database:
ua fp: 1:81.0.4044:3:10:0:0:0:1:
ua sig: 1:81.0.4044:3:10:0:0:0:1:
Expect request fingerprint to match request signature from database:
rq fp: 303:c02b,c02f,c02c,c030,c013,c014,9c,9d,2f,35,ff:0,b,a,23,10,16,17,d:1d,17,19,18:0,1,2::
rq sig: 303:1301,1302,1303,c02b,c02f,c02c,c030,cca9,cca8,c013,c014,9c,9d,2f,35,a,a:0,17,ff01,a,b,23,10,5,d,12,33
d,17,18,18:0::
match: impossible
reason: impossible_cipher
Security report:
```

Iteration VI

Iteration VI

- Version of mitmproxy from the master branch contained only a part of ciphers from the OpenSSL spec
- Solution: ran an openssl command to extract the ciphers with their ids:

```
$ openssl ciphers -V ALL
0xC0,0x30 - ECDHE-RSA-AES256-GCM-SHA384
0xC0,0x2C - ECDHE-ECDSA-AES256-GCM-SHA384
0xC0,0x28 - ECDHE-RSA-AES256-SHA384
0xC0,0x24 - ECDHE-ECDSA-AES256-SHA384
0xC0,0x14 - ECDHE-RSA-AES256-SHA
```

- Injected these into the source code (mitmproxy)
- However, it there were still differences in ciphers, because of SSL libs mismatch.

Iteration VI

- Downgrading the library to some lower version would probably resolve the problem. Final solution should include dynamic load of multiple TLS libraries based on some client-based heuristic (need info about many TLS libraries and its versions)

What is next?

What happened after iteration VI?

How to find which SSL/TLS libraries can support ciphers which victim supports?

- Downgrading mitmproxy
- Upgrading firefox => (no TLS_EMPTY_RENEGOTIATION_INFO_SCSV) .
Firefox is vulnerable?
- Upgrading Google Chrome => hard to teach mitmengine what Chrome does