Iteration V report

Daria Vaskovskaya

1 Intro

- **Student name**: Daria Vaskovskaya.
- **Topic**: Man In The Middle Attacks: Execution and Detection
- **Supervisor**: Rasheed Hussain
- **Iteration number**: 5.

2 Done during third iteration

It turned out that the `mitmengine` signature list did not contain signatures or contained not enough infromation (such that the normal browser request was percieved as MITM) for the firefox builds I tried. The process of both finding this out, and fixing the problem is described in further sections.

- hours spent 12 hours
- status Done
- results will be available below

3 Results

To start with, there was enough problems during this iteration, most of the problems involved wireshark and its documentation and mitmengine and its documentation (in both cases instructions from the documentation did not work, I did not find any solutions in the internet so I had to read sources).For example, in wireshark documentation it is advertised to use ./configure in order to compile wireshark with ssl support, however there is no configure file (cmake instead), moreover it was not easy to determine why some functions just did not work (there was no error outputs), so this iteration was more about problem fixing.

Documentation for `mitmengine` is not really elaborate about how does it work, I needed to read the source code (which was again rather convoluted).

In the end, the whole signature matching process is based on two strings - *UA fingerprint* and *TLS fingerprint*. The former is the formatted output of `uasurfer` library for go, which basically just parses the `User-Agent` into different fields.

The latter is built from the TLS client hello handshake part, it contains the TLS version, supported ciphers, supported curves, and some presented extensions.

The two previous scripts for capturing traffic both with and without `mitmproxy` were executed and their debug output carefully analyzed, to finally find out that `mitmproxy` does indeed offer different cipher suites from firefox's:

```
# this one is without mitmproxy
$ ./record_without_mitm.sh
...
    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,23,10,5,
        33,2b,d,2d,1c,15:1d,17,18,19,100,101:0::
...
# this one is with mitmproxy
$ ./set_up_transparent_proxy_with_pcap.sh
...
    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::
...
```

The cipher suite section is the one between the first and the second colon (i.e. `b` from `a:b:c:d:e`).

3.1 Fixing stuff

`mitmengine`, while written in go, uses a bunch of scripts in python and bash to perform the packet dissection (which also call `tshark`, the CLI for wireshark, to do the heavy lifting), and that has resulted in a bunch of problems.

First of all, the required python version was 3.7, and `apt` couldn't arrange a consistent 3.7 environment, so I was stuck with refactoring the code to use only 3.6 features.

Next up, and this is a more serious problem, my `tshark` version for some reason named all fields regarding TLS packets as `tls.<name>`, while the scripts expected them to be `ssl.<name>`. That was also the reason why the section with elliptic curves was empty in TLS fingerprints which were gathered automatically. In `tls.<name>` params they are part of the core packet info, while in `ssl.<name>` they were exposed as extensions, so the brute-force method of replacing `ssl` with `tls` in the script did not do the job completely.

I tried to install the same version as creators of mitmengine used during testing and development, however I did not find this exactly this version in wireshark archives. And, instructions did not work with the closest versions to the version used by the developers.

In the end, `mitmengine` was able to successfully recognize the firefox fingerprint, and successfully notify that `mitmproxy` did not match its expectations (due to `impossible_cipher`).

```
$ ./set_up_transparent_proxy_with_pcap.sh
...
    match:  impossible
    reason: impossible_cipher
...
```

```
$ ./record_without_mitm.sh
...
    match:  possible
...
```

In order for `mitmengine` to correctly identify `mitmproxy` in the final result (for it to write *"look, there is a mitm attack happening, alert!"*), I needed to manually add `mitmproxy` cipher suite fingerprint into the database, which turned out to be really simple, I could probably automate this thing better than the provided scripts.

```
$ ./set_up_transparent_proxy_with_pcap.sh
...
Request fingerprint matched known MITM signature:
    rq sig: <cipher signature>
    name:   mitmproxy
    type:   1
```

`type` here is just a dummy string, it is used internally to categorize MITM software (antivirus, malicious, content filter, etc.)

4 References to results

Link to patches: https://github.com/BananaAndBread/Project/blob/master/results/mitmengine.patch

5 Current issues

- spent so much time of fixing stuff

6 Planned for the next iteration

I can probably automate this manual work of adding signatures manually. Also, as planned in the beginning, I can start trying to circumvent this detection by careful configuration (and maybe manual patching) of mitmproxy.