Maintenance Manual

Miguel Villanueva

May 2023

This Appendix would explain the structure of the source code and what to take on account when manipulating it. Several of the explanations here would overlap or redirect to the comments on the code and some information has already been explained in the implementation chapter. However, here it would also be explained the possible routes for improvement and expansion. In first place, this is the structure of the source code: Dir: SourceCode

- Dir:POC_Combination
    1. File: manifest.json
    2. File: background.js
    3. File: content_script.js
- Dir: Server
    1. Dir: Controlç
        (a) Dir: static
            i. File: edit.css
            ii. File: list.css
        (b) Dir: files
            i. File: Prueba.bat
        (c) Dir: templates
            i. File: List.html
            ii. File: Edit.html
            iii. File: Add.html
    2. Dir: DB
        (a) File: DB.sqlite
    3. File: app.py
    4. File: log.py
    5. File: README
    6. File: requirements.txt

- Dir: Short_Extensions
    1. Dir: Server
        (a) Dir: logs
            i. Files: All .csv files
        (b) File: app.py
        (c) File: log.py
        (d) File: README
    2. Dir: Short_Destructive
        (a) File: manifest.json
        (b) File: background.js
        (c) File: content_script.js
    3. Dir: Short_Detective
        (a) File: manifest.json
        (b) File: background.js
        (c) File: content_cript.js
    4. Dir: Short_Stealing
        (a) File: manifest.json
        (b) File: background.js
        (c) File: content_cript.js
    5. Dir: Short_Weak_Syste
        (a) File: manifest.json
        (b) File: background.js

Now, starting from top to bottom.

1 POC

The three main files of the POC, the manifest, the background (standard name for the Service Worker) and the content script have been explained on the Background chapter. Nevertheless, the internal structure as not been explained.

The manifest file structure is common to all the manifest files. Each element is declared, names, descriptions, extension version and manifest architecture version. Also, the service workers (they can be more than one), the content script (they can also be more than one) and the websites to which the content script apply and the files that it can access.

In the POC and in the rest of extensions there are only one of each that applies to all the websites, with access to the facebook.css and the content_script.js. Also, the Content script is run when the DOM has been fully loaded. This is because the phishing changes must be made after load for assuring, they are conserved.

Like said before, in the manifest all permissions are stated, to fully view the permissions please refear to: https://developer.chrome.com/docs/extensions/reference/

Finally, is declared host permissions for ¡all_urls¿, this is done for assuring future capacities when executing code. However, like said in the dissertation this is a reason for rejection by Google. If wanted to expand this POC to a real attack, it would be needed to make it more specific. For fully tailoring to your needs, please refer to: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/manifest.json/host_permissions

The Service Worker represented by the background.js file is fully commented. Note that background.js is the standard name, clear inheritance of the past Background component. In any case, here it would be explained some parts of the code. Mainly the 3 different phases that exists. The main variables and some important methods to know. Also, some of those methods could be modify such that less noisy is generated. The main phases are: Start, Listening, Attack.

"Start" corresponds to the first installation of the code, it would stablish UID and other values and never modify them unless uninstalled. This is done by just checking if those value already exist on memory, and it is important because each time the extension is update or turn on it would also activate the listener onInstalled.

The UID is generate by hashing the time of installation to the seconds. The rest are set by default to None and are presented as global variables in case other person would want to modify the default state. Finally, the uid is communicated to the C&C

The "Listening" phase refers to the recollection of information and communication with the C&C. As said in the implementation and the code this is done with the built-in function of the respective APIs used. This can be seen in the code in the form of the logSomething functions that are all called by the logSend function. Is important to note that in this case the functions each made their unique call to the server. If wanted to follow OOP standards this should be a unique method for the process of calling, but in that case is easy to fall into race condition and overlap of scopes in the execution and compilation of python given place to random errors. A good expansion of this function would be to modify the recollection of code and sent all in one package using an asynchronous system and a internal dictionary on the JSON for allow the server to easily separate the contents. To obtain the attack values Get request are used. A persistence system was needed; therefore it was used the detection of navigation to google and the storage of date for that purpose. Those values would be explained in more detain in the global variables. Finally, the last phase, the Attacks, corresponds to both code in the Service Worker as in the Content Script. This phase activates when the stored variable phase equals 1. The background.js manage completely the download hijack attack, with the

listeners that can be seen at the end of the file. Those listeners as explained in the implementation, react to new downloads, compared them with the stablish parameters and if true, would execute the attack by cancelling the download and starting a new one with the payload. The phishing attack code would be mentioned later.

The globals variables that can be found in the background are the following:

1. DeltaA / DeltaB: The time in seconds that should pass between sending and requesting information
2. TimeA / TimeB: the last time that has been send/receive information to / from the C&C
3. Phase: The state of the attack phase
4. ObjDow: the string we search in the url of the downloads that define the objective
5. TarDown: the URL for downloading the payload
6. NamDown: The name that the payload would have
7. PhiChild: the element to change with the content script for the phishing attack
8. PhiParent: The parent element of PhiChild
9. PhiContents: The contents of the new button that will be generate for the Phishing attack
10. UID: The unique identifier of the user that will be send with the data to recognize the user
11. IP: The IP of the server
12. IP_Send: The route for sending data

2 Content Script

The Content Script injects two elements to all pages, the facebook.css that clones a facebook button and the content_script.js that injects the code that also execute when wanted the phishing attack. The other important thing that does is collect the OS and geolocation data. The main thing that it could be improved is the obfuscation of the code and where the css is introduced. Because, while inline script is forbidden, inline style is not and therefore the css file could be also send with the PhiContent variable, however for testing would have been more cumbersome.

The entire of the POC does not have more requirements then using Google Chrome.

3 C&C

The C&C (the first Server directory) requires of the use of Flask and SQLite packages for python, and the simple server (the second Server directory) requires the pandas library, as mentioned before. The App.py as the entire server in both cases and the log.py as the corresponding logging functions to store the data on the database. The actual structure of code expects that each data arrives by different routes already pre-sorted for storing the information. If modify the sent of information in the extensions, the C&C should be modified accordingly.

The actual utility of the extensions gives little space for extension or improvement in the backend point of view. However, extensive work could be done to improve the UI experience and the visualization of data, which at this point is good enough for showing and proving concepts, but horrible for analysing. For extending the analysing capabilities a modification of the log.py file and the structure of the database would be needed, such that all information is stored correctly and ready to use. Nevertheless, due to how standardized is the code it should not be more complicated than modifying some strings.

Note: if a some point the extension stops interacting with the C&C restart the flask server.

4 Database

The architecture of the database can be seen in the implementation. With the reasons of why and how already explained. Therefore, for avoid repetition, please go to Section**??**.

What is not commented there, is the possibilities of improvement. They are little due to the fact of being a simple database. Obviously creating a real server database would be good for a real attack, in the same way that correctly separating each parameter of each table instead of storing the data as a long text would help with the improvement of the UX of the C&C and the analysis of the data.

5 Short Extensions and Simple Server

None of this component should be worked further, they existed for proving ideas and constructing concepts in accumulative way for the POC. However, it would be explained the main components for debugging purposes as well as making sure no damages is done to any computer.

Stealing and Detective extension are well explained before and are like the Listening phase on the POC. Please refer to the information there or to the commented code for working on them (see **??**,**??**,**??**,**??**). The Weakening extensions is also explained on the implementation (see **??** and **??**). Nevertheless, is not explained that the only real way to return to the default privacy settings is

by turn off the extension itself. That is because the listeners are not killed by the extension at any point and the settings would change by themselves to the correct setting.

When using the Destructive extension, as explained before, be careful. It starts working immediately and will download several files at once without having to ask nor stop. Also, in the content script you will find a Fibonacci function that kills every tab by excess of computation and in the content script the rules that block any subsequent URL. If wanted to test without the infinite loop, delete or comment the last three lines in the background.js