ADVANCED INFORMATION SECURITY - LAB 6

NAME: CANDICE RENEE ROMEO GOMES
ID: 1001427308

1812fabf2303ccbdc535ba3fd5404895

A few days ago, a user complained that some of her emails that she had never read were marked as read. She just kicked her computer because that usually makes things better. Well, today she did some online shopping only to find her credit cards were all maxed out. The Incident Response (IR) Team pulled this unknown executable off her system. Make sure it won't affect anyone else.

1. ***What does the malware drop to the disk?***

Ans: The malware drops a file to drive: "C:\Program Files\Google\Update\GoogleUpdate.exe". This file is present in the resource section: RC_DATA "DROP"

2. ***How does the malware achieve persistence? Why does this make a great host-based signature?***

Ans: Because a file is dropped in "C:\Program Files\Google\Update\GoogleUpdate.exe" the malware runs every time Google Updater is triggered. We find this is a good host-based signature because we can check the validity of this file.

3. ***How does the malware ensure more than one instance of itself isn't running on the system at any given time?***

Ans: The malware ensures this by using the mutex 'WODUDE'.

4. ***What are 2 ways the malware tries to hide its persistence from the user?***

Ans: The two ways malware hides persistence:

- Hiding the console window
- Replacing a trusted file/program

5. ***What are the 2 major WinAPI calls involved in enabling the keylogging?***

Ans: To enable keylogging, I found 2 major WinAPI calls:

- SetWindowsHookExW – It enables a callback function when a key is pressed.

- SetWinEventHook – It enables a callback function when the window focus is changed.

6. ***What are the names of the constants passed to each of these WinAPI calls?***

Ans: There are 3 constants passed to each of these WinAPI calls:

- WH_KEYBOARD_LL
- EVENT_SYSTEM_FOREGROUND
- WINEVENT_SKIPOWNPROCESS|WINEVENT_OUTOFCONTEXT

7. ***What does the malware do with the collected data?***

Ans: The malware writes a file and places it in the current directory in this case "C:\Program Files\Google\Update\<hostname>". This file has keylogged data based on the user's inputs.

e60d2bd0d371c89e7029687913336529

After your analysis of the previous sample, the IR Team devised a solution to make it impossible for anymore keyloggers to steal the users' passwords or personal data. The very next day, the entire IR team was fired because the CEO found very incriminating photos of him all over the Internet, and those photos were only stored on his Dropbox account.

1. ***Why does this malware use the Internet? Where does it connect to?***

Ans: This malware downloads the file present at "http://malcode.rpis.ec/update_defender" and uses the file to replace the file at "C:\Program Files\Mozilla Maintenance Service\maintenanceservice.exe". If that fails, it will replace that file with the DROP resource.

2. ***How does the malware achieve persistence? Why does this make a great host-based signature?***

Ans: Like the previous malware, this malware overwrites the update service for the Firefox browser. We find this as a great host based signature because we can verify this file to confirm presence of the malware.

3. ***Why is the second mutex necessary in this sample?***

Ans: A second mutex is required so that only one enumeration of child windows is done at one time. The first enumeration to run will secure the mutex, and the next enumerations will have to wait for this mutex to be released

4. ***Briefly describe what SendMessage does.***

Ans: SendMessage is used to send an Event/Message to a window or windows. This can be used for updates or triggers, e.g. mouse, keyboard

5. ***What are the names of the 3 constants used (as the 2nd argument to SendMessage) by this malware? What are their purposes?***

Ans: The 3 constants used in the 2nd argument of SendMessage is:

- 0xD2 - EM_GETPASSWORDCHAR - gets the character that an edit control message shows when a user is typing a password instead of showing the password.
- 0xC4 - EM_GETLINE - Copies a line of text from an edit control and places it in a specified buffer.
- 0xCC - EM_SETPASSWORDCHAR - sets the character that an edit control message shows when a user is typing a password instead of showing the password. In this case, the malware sends a parameter of 0 which means the control message will show the password plainly.

6. ***How does this malware steal passwords? How is it different from the last sample?***

Ans: This sample looks for password textboxes in foreground windows. Once it finds one it will remove the password mask using EM_SETPASSWORDCHAR, steal the password with EM_GETLINE, and then reset the password mask.
This differs from the last sample because the last sample hooked keyboard events to log all keystrokes. This sample specifically targets password fields.

7. ***What does the malware do with the collected data?***

Ans: All the collected data is written to a file in the current directory which is "C:\Program Files\Mozilla Maintenance Service\<hostname>".

eff5e85b1e0d7f892acac99475b1c324

1. Looking at the main function:

a. What is the address of the _main function?

Ans: 0x00403420

b. What imported functions does main call?

Ans: Imported functions:

- GetCommandLineA
- GetStartupInfoA
- GetModuleHandleA

c. What do these functions do?

Ans: Function properties

- GetCommandLineA: Gets the command string for the current process.
- GetStartupInfoA: Populates a STARUPINFO structure for the current process.
- GetModuleHandleA: When passed NULL (like it is here) returns a handle to the file that created the process.

2. Looking at the subroutine at 0x00402C6E:

a. What is the address of the encoding/decoding function?

Ans: The decoding function is at address 0x4012EC

b. What is being encoded/decoded?

Ans: It is decoding a bunch of null terminated strings in the .data section.

c. What is the very large basic block doing?

Ans: The very large basic block is doing a bunch of decoding to be used later, since there isn't any logic, just a lot of decoding, it makes the block large.

3. Looking at the subroutine at 0x004023D0:

a. What are all the GetProcAddress calls doing?

Ans: All the GetProcAddress calls are dynamically loading library imports after it has decoded them with the decode function at 0x4012EC.

b. What does this function do?

Ans: It loads all the functions it decoded, then if they all loaded successfully, it returns 1, otherwise it frees the library and returns 0.

4. What does this malware do? Be detailed in your description (at least 2 paragraphs)

Ans: The first thing it does after loading all the functions is to calculate the expected size of the exe due to headers, then read extra bytes from the end of the file that were not loaded in with the binary. It then takes this string and decodes it using the same function as before. Then it uses RtlDecompressBuffer on it, but for some reason this does not work at all, so an empty buffer is used. It would have done a bunch of operations based on this decompressed value, but we will have to skip those and look at what it would have done after this. Next it gets the executable path and appends a buffer to it (maybe as arguments) but that buffer appears to empty at this moment. It then creates a new process of this file with CreateProcess. It then seems like it would have read memory from that process and preform some sanity checks on it. It then also does a bunch of writing to this process' memory, possibly some sort of dynamic code modification.

It is very difficult to only use static analysis of these operations, but based on the unused but still decoded strings, I would make a guess that it does some checks to see if it is in a sandbox/vm ('sandbox' and 'vmware' strings). It may also do other redpill tests using GetUserNameA and reading registry values, since these functions were also decoded. Finally, it may try to swap the mouse buttons, due to the 'Control Panel //Mouse' and 'SwapMouseButtons' strings which were also unused.

Looking at the data that was at the end of the file, there may some sort of botnet kind of thing connected though IRC, due to the IRC commands there, however, there is not a lot of code in the binary, so that may be unlikely.