CYBR473 Assignment Three – Part Four Report
greenthom – 300536064

1. **Warmup**

1. a) Investigate the code and provide the following: 3 host-based indications of compromise by this backdoor (justify each of them).

- In Backdoor.cpp, the malware creates "C:\\Users\\" + win_username + "\\AppData\\Local\\SystemConnection" with data and app subfolders to store downloaded scripts and stolen data. This hidden folder indicates malware presence on the system.
- The malware copies its executable to explore.exe, renaming itself to avoid detection by users or security tools and to match a legitimate Windows process name explorer.exe.
- In Backdoor.cpp, it adds registry key modifications for persistence, adding itself to HKCU\\Software\\Microsoft\\Windows\\CurrentVersion\\Run with the name SysConnection ensuring the malware automatically runs at user login without requiring administrator privileges.

1. b) Investigate the code and provide the following: 1 network-based indication of compromise by this backdoor (with justification).

- The backdoor connects, ftp_connection.connect(), to an FTP server using hardcoded credentials, downloads command scripts (cmd.txt), uploads stolen data (output.txt), and interacts unusually over FTP instead of encrypted modern protocols.

2. **Training**

2. a) First, briefly describe the main idea and how this is possible.

- Mimikatz can extract plaintext passwords, hashes, and Kerberos tickets by accessing the memory of the LSASS (Local Security Authority Subsystem Service) process memory which contains credential structures. Windows stores authentication credentials in memory for user convenience. Mimikatz reads this memory space, either locally or through elevated privileges, allowing attackers or

penetration testers to retrieve credentials without cracking them. If credentials are encrypted it can obtain encryption key from memory to get plaintext.

2. b) Second, investigate the source code, and identify the function that implements this feature. Note that there is a directory inside that repository with the name of mimikatz: https://github.com/gentilkiwi/mimikatz/tree/master/mimikatz . That directory pertains to the Windows XP version (legacy). Your answer can be either for that version or the new version (https://github.com/gentilkiwi/mimikatz).

- The function is kuhl_m_sekurlsa_getLogonData(), which extracts stored credentials, including plaintext passwords, by reading memory from LSASS. LSASS stores credentials from providers like SSP and Kerberos. Mimikatz lists these providers in an array where the function then reads the in-memory credential structures of these, allowing recovery of usernames, hashes, sometimes plaintext passwords. kuhl_m_sekurlsa_enum_logon_callback_wdigest is a supporting function that extracts the plaintext of passwords specifically the WDigest provider.

3. Cool – down

3. (a) This RAT features a downloader and launcher.

i. What program does it use for downloading new binaries?

- Ghost RAT uses wget.exe to download new binaries. It drops a hidden wget.exe binary onto the victim's system, then uses it to silently download new payloads from specified URLs, saving them locally before execution. It runs wget in the background with no visible console window.

ii. Where does the RAT find the downloader program from? (where is the actual binary of the downloader executable located at?)

- The RAT retrieves the downloader program from a built-in resource labeled IDR_RCDATA1 of type RT_RCDATA. This resource is accessed directly in zombie.cpp using Windows API functions. The downloader binary (wget.exe) is not externally downloaded; it is embedded within the compiled RAT executable as a resource and extracted to disk at runtime using API calls (e.g. FindResource, LoadResource).

iii. What windows API or windows command does the RAT use to run the downloader program?

- The command used is CreateProcess (“wget.exe”, c_output, NULL, NULL, TRUE, CREATE_NO_WINDOW, NULL, NULL, &info, &processInfo). The RAT uses the Windows API function CreateProcess to run the downloader program wget.exe. It sets the CREATE_NO_WINDOW flag to ensure the process runs silently without displaying a command prompt window. This allows the downloader to operate in the background without alerting the user to any activity.

iv. Once the RAT uses the downloader to download a new binary (send by the Command and Control server), what does it use to launch the downloaded executable?

- The RAT uses the Windows API function ShellExecute to launch the downloaded executable. After the downloader retrieves the file. ShellExecute is called with the “open” operation, allowing the RAT to silently execute the new binary without requiring a visible console window or additional user interaction.

3. (b) This RAT also implements an encryption.

i. What encryption scheme is it?

- Ghost RAT implements a custom substitution cipher for encryption. It replaces characters based on two hardcoded arrays, c_original and c_encrypt. Each character from the input text is matched and swapped to an encrypted character. This is not a standard cryptographic scheme but a basic obfuscation method with no cryptographic strength.

ii. What is it used for?

- The custom encryption is used to obfuscate data sent between the RAT and its Command and Control (C2) server. Both incoming commands and outgoing responses are encrypted, making traffic harder to detect by simple network monitoring or analysis tools, even though the encryption itself is very weak.

iii. Where does the key come from?

- The key for the RAT’s encryption is hardcoded into the source code in encrypt.h via c_encrypt. It consists of two character arrays, c_original and c_encrypt, which

define the direct substitution mapping used to encrypt and decrypt communication between the RAT and its Command and Control (C2) server.

3. (c) This RAT obtains the filename of its executable and the path. What is the purpose of this? (This is, what does the RAT use these for?)

- Ghost RAT obtains its filename and path (GetModuleFileNameA()) to copy itself (fstream) into trusted locations like the Windows directory or temp folder. After copying, it relaunches itself (ShellExecute) and modifies and sets registry keys for persistence. This behavior helps the RAT hide from users and antivirus tools (SetFileAttributes (FILE_ATTRIBUTE_HIDDEN) while ensuring it starts automatically on reboot.

3. (d) The RAT uses three different registry keys for its persistence mechanism, in a specific order of preference.

i. What are these three places in the order of preference of the RAT?

- The RAT first tries HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon (Shell key). If that fails, it attempts HKLM\Software\Microsoft\Windows\CurrentVersion\Run (WinUpdateSched). If that also fails due to permission errors, it uses HKCU\Software\Microsoft\Windows\CurrentVersion\Run (WinUpdateSched).

ii. What is the justification for this order of preference?

- Ghost RAT first uses Winlogon\Shell because it ensures execution even in Safe Mode. If administrative access prevents modifying this key, it falls back to HKLM\Run for machine/system-wide persistence. If that also fails, it finally uses HKCU\Run for per-user startup without requiring administrator privileges.

4. Post-Workout Shake/Smoothie/Overstretched Metaphor!

4. Here are the features of this RAT listed by the author:

- **Runtime loading of DLLs**
- **Anti-virus evasion via simple strings obfuscation**
- **Deletes itself and runs from temporary folder**
- **Anti-debugging via rdtsc timing**
- **Anti-sandbox via process enumeration**

4. (a) About the first claim:

i. What specifically in the code shows the use of "runtime loading of DLLs"?

- RATwurst shows runtime loading of DLLs by using the Windows API function LoadLibraryA to load libraries like kernel32.dll and ws2_32.dll, and GetProcAddress to find specific functions dynamically. This technique hides API usage from static analysis and helps evade signature-based detection methods during malware scans. Defines API names in character arrays, reducing visibility during static string scans.

ii. How the code would have been different if it would be using the usual (i.e., load-time) dynamic linking as opposed to runtime dynamic linking?

- With usual load-time dynamic linking, the code would directly call API functions like CreateFileA and GetComputerNameA without manually loading DLLs. The compiler would link against the required libraries (e.g. kernel32) during build time, and function addresses would be resolved automatically when the program loads into memory.

iii. What is the purpose of using runtime dynamic linking of DLLs (i.e. runtime loading of DLLs) here?

- Runtime dynamic linking allows the RAT to hide which Windows API functions it uses. Since function names are not visible in the program's import table, static analysis and antivirus software have a harder time detecting or flagging the malware based on known suspicious API usage patterns.

4. (b) About the second claim:

i. What specific string obfuscation has this RAT implemented?

- Implements string obfuscation by defining sensitive strings (like DLL and function names) as character arrays instead of string literals. For example, CreateFileA is stored as ('C', 'r', 'e', 'a', 't', 'e', 'F', 'I', 'l', 'e' ,'A'). This prevents these strings from appearing in plain text in the binary, making it harder for static analysis tools and antivirus scanners to detect malicious indicators.

ii. What part of the code is responsible for that?

- Obfuscation of function and DLL names is implemented throughout the code using character array definitions for strings that are normally used as literals. These arrays are then passed to GetProcAddress() or LoadLibraryA() to resolve the actual functions at runtime. Ensures that no readable strings like “CreateFileA” or “kernel32.dll” are present in the compiled binary’s string table, which is the core obfuscation technique used.

4. (c) About the third claim:

i. What section of the code is doing the file deletion?

- function is responsible for file deletion. It builds a command using ping 127.0.0.1 & del <path> to delay and then delete the original executable after copying itself to a new temporary location, allowing the RAT to clean up its initial presence from the system. Also has a flag FILE_FLAG_DELETE_ON_CLOSE in a handle object in the ReceiveCmdCommand() method that occurs after calling gf_CloseHandle(handle), which is an alternative way of deleting.

ii. How does this RAT achieve persistence if it deletes its own execution file?!

The RAT copies itself to a new file in the system’s temporary folder before deleting the original executable. It then sets a persistence mechanism by writing a registry key, HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run, pointing to the new file, ensuring the RAT runs automatically after system reboots or user logins.

4. (d) Anti-debugging claim: the author is claiming that it is using “rdtsc timing”. In the author’s blog post (https://www.accidentalrebel.com/making-a-rat.html), we further read: *Anti-debugging checks are littered throughout the code which checks the amount of time it takes to reach from one part of the code to the next. This is used to detect if someone is stepping through the code, increasing the delay between code execution. When no shenanigans is detected then it’ll proceed to work as intended.* Via analysing the code, find out what specific condition on the timing is used for this purpose, and whether it is a reasonable check.

The RAT uses QueryPerformanceCounter to measure execution time between two points. If the time difference exceeds 3 seconds (or 20 seconds during reconnection attempts), it assumes debugging activity and exits. This is simple but reasonable anti-debugging check, though advanced debuggers can bypass or adjust for it.

4. (e) Finally, the "Anti-sandbox" claim. From the author's blog: *When RATwurst is first executed. It does some anti-sandbox checks via process enumeration. It checks if there are more than 15 process[es] that are running and if there are virtualization tools present like vmware.exe. It will also setup an auto-run registry entry for persistence. And also move the executable to Windows temp folder and re-run it from there.*

i. **What does comparison with 15 signify? (What is the justification of this comparison for the claimed anti-sandbox feature of this RAT?) Is this justified?**

The comparison with 15 processes checks if the system is likely a sandbox, since sandboxes typically run fewer processes than real machines. If fewer than 15 are detected, the RAT assumes it is inside a sandbox and exits. This method is simple but not fully justified, as it may result in false positives.

ii. **Besides vmware.exe, this RAT looks for presence of another process as well, what is this?**

Besides vmware.exe, the RAT also checks for the process vmtoolsd (vbox.exe), which belongs to VMware Tools, a common utility in virtual machines. By identifying vmtoolsd, the RAT improves its detection of virtualized or sandboxed environments. This makes it harder for malware analysts to safely study the RAT's behavior without detection.