MJoSHT
Malaysian Journal of Science, Health & Technology
[mjosht.usim.edu.my]

USIM

Article

Investigation of 0debug Stealer Using Assisted Static Analysis Method

Taqiyuddin Anas[1], Farida Ridzuan[1,2], Sakinah Ali Pitchay[1,2] and Charles Lim[3]

[1]*Faculty of Science and Technology, Universiti Sains Islam Malaysia, 71800 Nilai, Malaysia.*

[2]*Cybersecurity and Systems Research Unit, Faculty of Science and Technology, Universiti Sains Islam Malaysia, 71800 Nilai, Malaysia.*

[3]*Information Technology Dept, Swiss German University, Kota Tangerang, Banten 15143, Indonesia.*

Correspondence should be addressed to:
Farida Ridzuan; farida@usim.edu.my

Article Info
Article history:
Received:15 July 2025
Accepted: 5 February 2026
Published:15 Mac 2026

Academic Editor:
Nurzi Juana Mohd Zaizi

Malaysian Journal of Science, Health & Technology

MJoSHT2025, Volume 11, Special Issue on the 5th International Conference on Recent Advancements in Science and Technology (ICoRAST 2025): Responsible Artificial Intelligence – Advancing Science and Technology for Humanity
eISSN: 2601-0003

https://doi.org/10.33102/mjosht.512

***Abstract*— This study investigates the capabilities of the identified 0debug Stealer malware through qualitative static analysis and reverse engineering. This study primarily employs non-executional static techniques, such as fingerprinting, decompiling, and string extraction, with a bit of dynamic analysis used primarily for deobfuscation. Analysis confirms that obfuscation hinders direct examination, yet reveals core functionalities of 1) anti-tampering, anti-VM, and anti-sniffer to evade detection 2) data theft targeting important victim information, and 3) Command and Control exfiltration. Findings demonstrate 0debug Stealer's capabilities in bypassing traditional security measures, emphasising the critical role of static analysis in dissecting advanced threats. This paper contributes to the forensic analysis of emerging threats by providing a detailed examination of the 0debug Stealer's manner, revealing its evasion techniques and comprehensive data theft capabilities.**

***Keywords*— malware investigation; 0debug Stealer; obfuscation, assisted static analysis; forensic**

I. INTRODUCTION

0debug Stealer is a newly discovered malware variant that emerged in late November 2024, as reported. 0debug Stealer exhibits sophisticated anti-tampering and anti-analysis techniques, which complicates direct analysis. This research aims to investigate the 0debug Stealer using assisted static analysis methods to understand its behaviours and capabilities. Assisted static analysis examines a malware sample without executing it, while dynamic analysis monitors its behavior during execution. Assisted static analysis in this study, meaning using the dynamic analysis method to deobfuscate the sample utilizing a debugger. Analysis was then continued using a static analysis approach to study malware behavior.

This research focuses on the static analysis approach to uncover the inner workings of the 0debug Stealer. The emergence of 0debug reflects broader trends in stealer malware evolution, where threat actors increasingly target cryptocurrency wallets, browser data, and VPN configurations, a pattern documented in Europol's 2024 Internet Organized Crime Threat Assessment (IOCTA) [1].

Malware analysis is commonly conducted using two distinct methodologies: static and dynamic analysis [2]. Dynamic analysis involves monitoring malware's behaviour during execution, enabling real-time observation. However, its effectiveness is diminished when confronted with sophisticated threats, as many modern malware variants are engineered to detect and evade execution [3]. The 0debug Stealer, for example, integrates specific anti-analysis capabilities, including Anti-VM and Anti-Sniffer, which will break execution during the initial check. Consequently, this study adopts the static analysis approach to safely uncover the functions and capabilities of the 0debug Stealer [4]. A similar approach was used to analyse malware and infostealer using static analysis [5-7].

Evasion techniques, such as anti-VM, are not unique to 0debug but are characteristic of modern information stealers. For instance, Redline Stealer is well-documented to employ anti-VM and anti-debugging mechanisms to prevent automated sandbox analysis [5, 6]. However, the implementation of the anti-sniffer detection mechanism in 0debug is relatively uncommon and has received limited attention in prior studies of stealer malware.

This paper presents a comprehensive investigation of the 0debug Stealer. The rest of the paper is organised as follows: Section III presents the methodology, outlining the static analysis techniques, the secure laboratory environment, and the tools utilised for the examination. Section IV presents the results and a detailed discussion of the findings, including the deobfuscation process and a systematic breakdown of the malware's discovered functionalities, from evasion mechanisms to data exfiltration targets. Finally, Section V concludes the paper by summarizing the key findings, underscoring the critical role of static analysis in dissecting advanced threats, and suggesting potential directions for future work.

II. LITERATURE REVIEW

The administrative infrastructure of information-stealer malware, often accessible via web-based control panels, represents a critical component of the cybercriminal supply chain, enabling threat actors to manage campaigns, exfiltrate data, and monetize stolen information [8].

Analysis of these panels provides valuable forensic insights into attacker operational security (OPSEC), target demographics, and the commercial or collaborative models underpinning malware distribution. Figure 1 shows the panel for 0debug stealer.

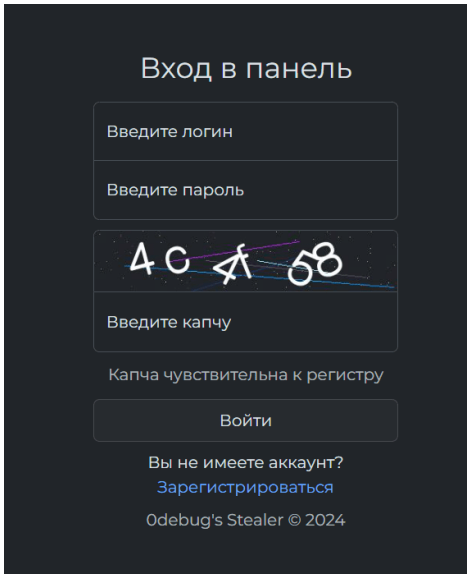


Figure 1. 0debug Stealer login panel

As shown in Figure 1, the login panel for the 0debug Stealer features a standardized authentication interface that requires a username, password, and a case-sensitive CAPTCHA, and uses distinctive Russian-language terminology.

This branding and structured access mechanism align with patterns observed in established stealer families. For instance, comparable panels for Redline and Vidar Stealers (Figure 2) share a similar functional layout for operator or affiliate access control, yet differ in visual branding and specific implementation details.

Scholarly and technical analyses of malware-as-a-service (MaaS) platforms, such as those conducted on Redline and Vidar, indicate that the design of these panels often reflects the malware's target audience, ranging from less technical criminals to more sophisticated actors, and can imply the level of service and support provided by the malware developers [5, 6, 11].

The presence of a CAPTCHA in the 0debug panel, a feature also noted in analyses of other private or semi-private stealers, suggests an attempt to deter automated scanning and unauthorized access attempts to the C2 server, thereby preserving the integrity of the stolen data repository and the attacker's infrastructure [11]. From this, it is evident that 0debug stealer is a distinct family of stealer based solely on its panel, thereby establishing a research gap. Therefore,

this study aims to fill a gap in understanding the behavior of the 0debug Stealer.

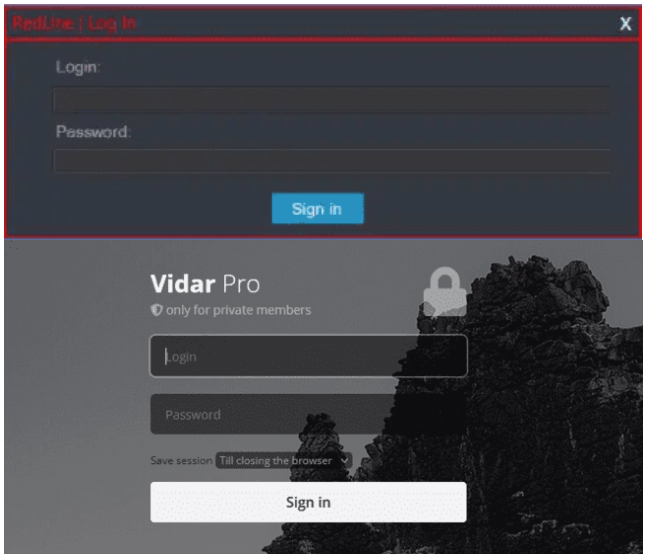


Figure 2. Redline Stealer login panel (top) and Vidar Stealer login panel (bottom) [9,10]

III. METHOD

Static analysis was employed to investigate the 0debug Stealer. This approach was deliberately chosen to facilitate a safe, controlled examination of the malware's binary without execution, an important step for bypassing anti-analysis and anti-tampering techniques triggered at runtime. The entire analysis was conducted within a secure, isolated virtualized environment to ensure containment and maintain analytical integrity. The step-by-step processes are executed as shown in Figure 3.

As shown in Figure 3, this study's methodology begins with the preparation system. A preparation system is a secure, isolated, virtualized environment configured for use with VMware Workstation to ensure containment of the host machine.

Next, in the binary sample download phase, the 0debug Stealer binary sample was downloaded from malware corpus VirusShare [12].

In the next step, the experiment proceeds to the deobfuscate binary sample phase. In this phase, entropy analysis confirmed that the sample was obfuscated, prompting the application of the de4dot tool to generate a deobfuscated executable, labelled cleaned-0debug.exe, for analysis.

Subsequently, the examination process followed a sequential step, including binary fingerprinting to verify integrity, file structure examination, string extraction, and decompilation to expose the program's logic. A critical step in this process was deobfuscating the binary sample, which involved using entropy analysis to detect obfuscation and generating a cleansed sample to enable deeper inspection.

Upon completing this examination, all artifacts and findings were systematically collated and documented in the Finish Reporting phase.

Finally, the evidence was thoroughly discussed in the Results section.

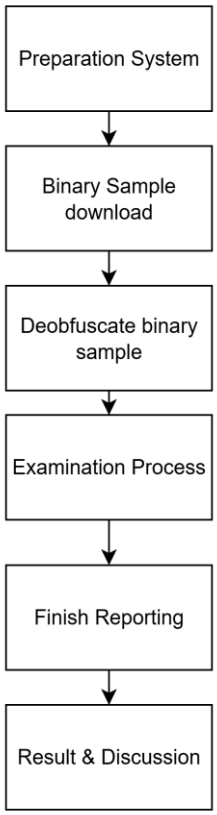


Figure 3. Methodology of static analysis examination process of 0debug Stealer

This methodology prioritized artifact dissection through reversible techniques, aligning with established static analysis principles for advanced malware forensics [13-15].

IV. RESULTS

Through static analysis, the deobfuscated 0debug Stealer sample revealed sophisticated evasion mechanisms and multi-stage data harvesting capabilities. Key findings demonstrate the malware's deployment of anti-tampering, anti-VM, and anti-sniffer techniques to prevent detection [16], followed by comprehensive reconnaissance targeting system information [11], browser data (Chromium/Gecko), cryptocurrency wallets, and a few application credentials.

A. Preparation System

The preparation system used in this research is to set up a virtual machine for malware analysis and to prepare a research object, namely, the 0debug Stealer malware sample. The virtual machine used is VMWare. The first preparation system to do is to install the operating system on the virtual machine. The operating system used in this study is Windows 10. System specifications used as an analysis lab are shown in Table I and Table II.

TABLE I. SPECIFICATION OF PERSONAL COMPUTER

Item	Value
OS Name	Microsoft Windows 11 Home
Language	English (United States)
Time Zone	(MYT +08:00) Malaysia
Processors	AMD Ryzen 7 7840HS
Memory	40 GB RAM
Total Storage	1500 GB

TABLE II. SPECIFICATION OF VIRTUAL MACHINE

Item	Value
VM Application	VMWare Workstation
OS Name	Microsoft Windows 10
Username	blue-box
Time Zone	(UTC-+08:00) Pacific Time (US & Canada)
Memory	8 GB RAM
Storage	100 GB

The tools used for each analysis technique in this study are shown in Table III.

TABLE III. LIST OF TOOLS USED

Analysis Technique	Tools Used
Fingerprint Malware	• Hashmyfile • Detect-it-Easy
File Type Identification	• CFF Explorer
String Extract	• Sysinternal Strings
Decompile	• dnSpy • Binary Ninja
Obfuscation Detect & Deobfuscation	• de4dot

The 0debug Stealer malware sample used in this study was downloaded from a secure repository. The sample malware is then extracted and renamed "samplemalware.exe".

B. Examination Process

The examination process is carried out on a virtual machine prepared during the preparation system process. The examination process consists of static analysis and dynamic analysis. Static analysis can yield valuable information for dynamic analysis. Therefore, the examination process in this study is carried out in the order of static analysis.

Static analysis is a method of analysing malware without running the malware. Static analysis consists of several stages, as shown in Figure 3.

The tool used at this stage is Hashmyfile. This stage aims to see the hash value of the malware sample file.

The hash value of the malware sample file is used to verify its authenticity by comparing the hash obtained from Hashmyfile with the hash from the secure repository. The results of the hash value examination using the Hashmyfile tool are shown in Figure 4.

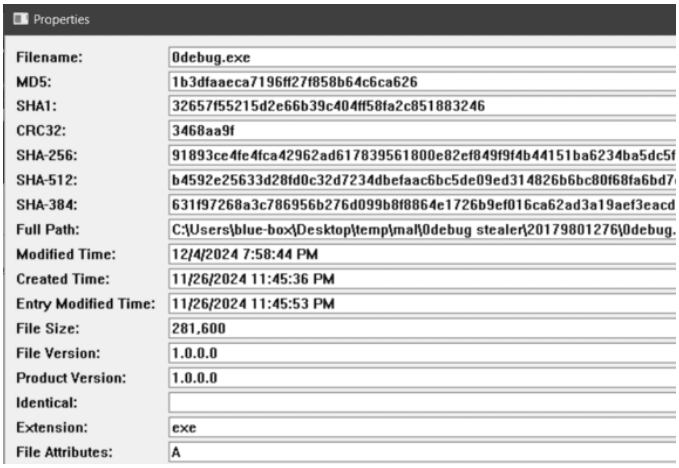


Figure 4. Hash value of malware sample using Hashmyfile tools

Figure 4 shows the result of hashing the file using Hashmyfile. The value generated by Hashmyfile matches the information from the secure repository, indicating that the file is original and unmodified. A comparison of hash values is shown in Table IV.

TABLE IV. COMPARISON OF HASH VALUE FILE MALWARE

Source	File Name	MD5
Local Files	0debug.exe	1b3dfaaeca7196ff27f858b64c6ca626
Virustotal	microsoft.exe	1b3dfaaeca7196ff27f858b64c6ca626

From Table IV, both hash values from different sources are identical. The only difference is the filename for the binary. The malware sample was then processed using the CFF Explorer tool, as shown in Figure 5.

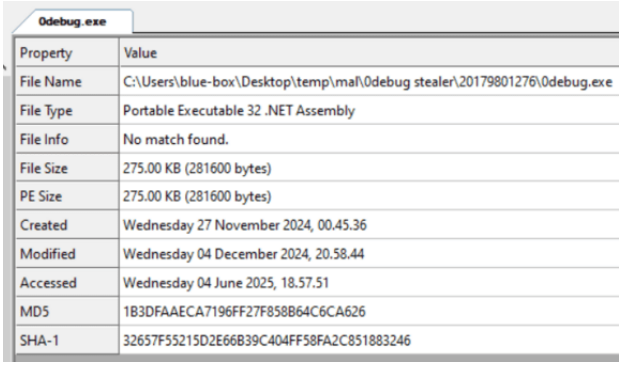


Figure 5. Detection of malware sample file type using CFF Explorer tools

Based on Figure 5, the information generated by this process indicates that the sample malware is a 32-bit Portable Executable (PE) file compiled with .NET, meaning it can run on Windows.

The analysis carried out at this stage involves extracting strings from the 0debug Stealer malware samples. The application used is Strings, developed by Sysinternals. The results of the string extraction performed on the 0debug Stealer malware sample file are shown in Figure 6.

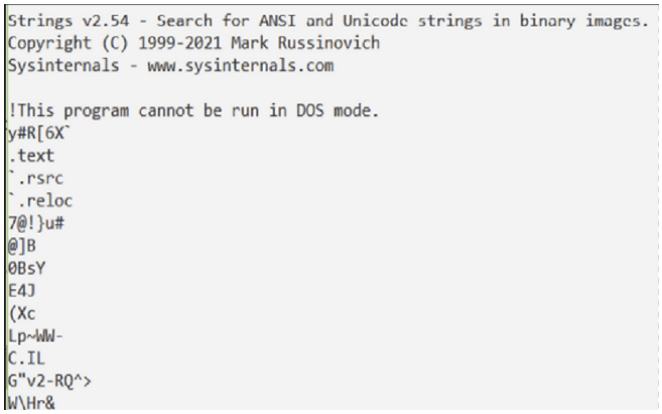


Figure 6. Strings extract of malware sample file types using Strings.

Figure 6 shows the results of the string extraction, which only returns random data. In other words, it does not produce any significant findings regarding the malware. The lack of information from this malware sample is due to the binary being compiled.

The decompilation process in this study uses the dnSpy tool, which converts compiled binary code into a high-level language for comprehension. The results of the decompilation process are shown in Figure 7.

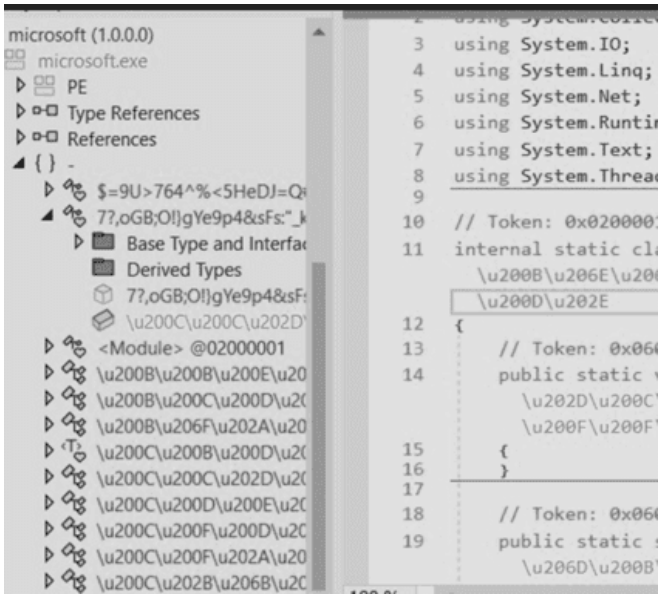


Figure 7. Result of decompilation for 0debug Stealer executable using DnSpy

After decompiling, the code's output appears similar in Figure 5, indicating that many Unicode characters cannot be understood directly. This is because the code originally was obfuscated. The obfuscation detection phase identified cryptographic and structural techniques employed by 0debug Stealer to hinder analysis. Using de4dot, as shown in Figure 8, the sample exhibited signatures consistent with .NET obfuscators [17].

Figure 8. Obfuscation detection on 0debug.exe using de4dot tools

Figure 8 shows that the binary is obfuscated. However, the type of obfuscator is not determined after using the de4dot tools. Another way to demonstrate that the binary is obfuscated is through entropy analysis, as shown in Figure 7.

This entropy drops to $H \approx 4.9$ post-deobfuscation in Figure 10, confirming successful payload recovery, enabling further string extraction and analysis.

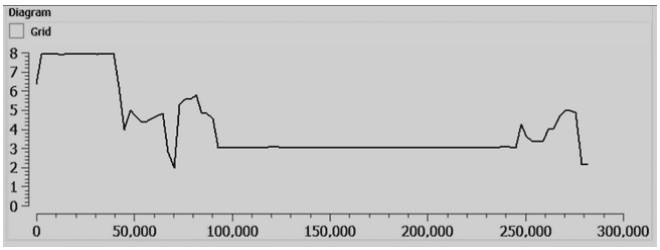


Figure 9. Entropy of obfuscated 0debug.exe executables using Detect-it-Easy tools

Entropy analysis, as shown in Figure 9, quantified this obfuscation, revealing a high entropy value of $H \approx 7.8$, which means the file's data is highly random, a strong indication that it has been obfuscated, corresponding to being packed or encrypted to hide its actual code and evade analysis. The entropy exceeds thresholds typical of benign .NET executables ($H < 6.2$) [18].

The binary sample was then extracted using a few NOP instruction implementations after obfuscated function execution, utilising debuggers in DnSpy. The result of the deobfuscation process is a file named "cleaned-0debug.exe", as shown in Figure 10.

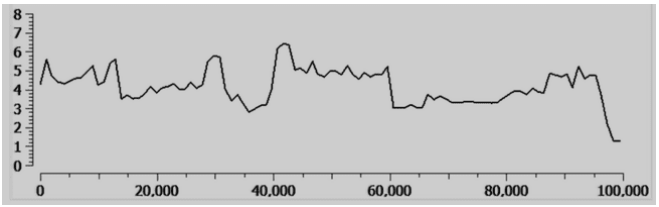


Figure 10. Entropy after deobfuscation of cleaned-0debug.exe

The entropy reduction in cleaned-0debug.exe in Figure 10 enabled deeper static dissection of 0debug Stealer's core logic. Some information changes regarding the original malware and the deobfuscated malware samples are shown in Table V.

TABLE V. COMPARISON OF ORIGINAL EXECUTABLES AND DEOBFUSCATED EXECUTABLES

Type	File Name	MD5
Original Files	0debug.exe	1b3dfaaeca7196ff27f858b64c6ca626
After deobfuscation	cleaned-0debug.exe	f0949f6146b7ee88e23c68174cf2cb8f

According to Table V, the binary hashes for both files differ. This means the sample file has undergone changes, specifically deobfuscation via binary extraction.

The deobfuscated malware sample files are then used to extract strings. Analysis revealed a phase operational flow as discussed in the Results & Discussion section.

C. Result & Discussion

After static analysis of cleaned-debug.exe, Figure 11 was produced to understand the flow of the 0debug stealer binary execution.

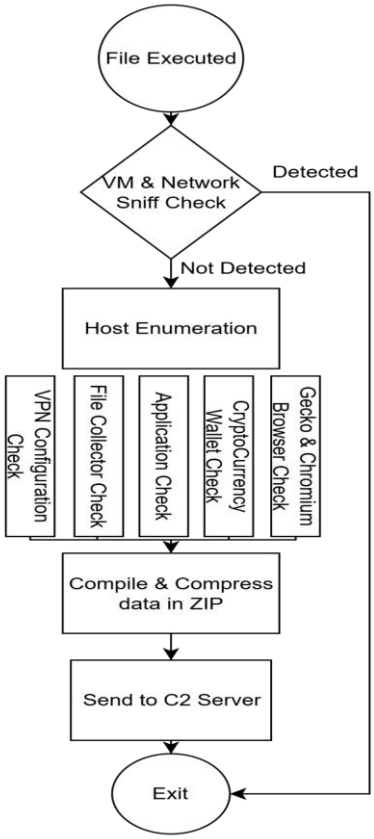


Figure 11. Flow of 0debug Stealer execution and its capabilities

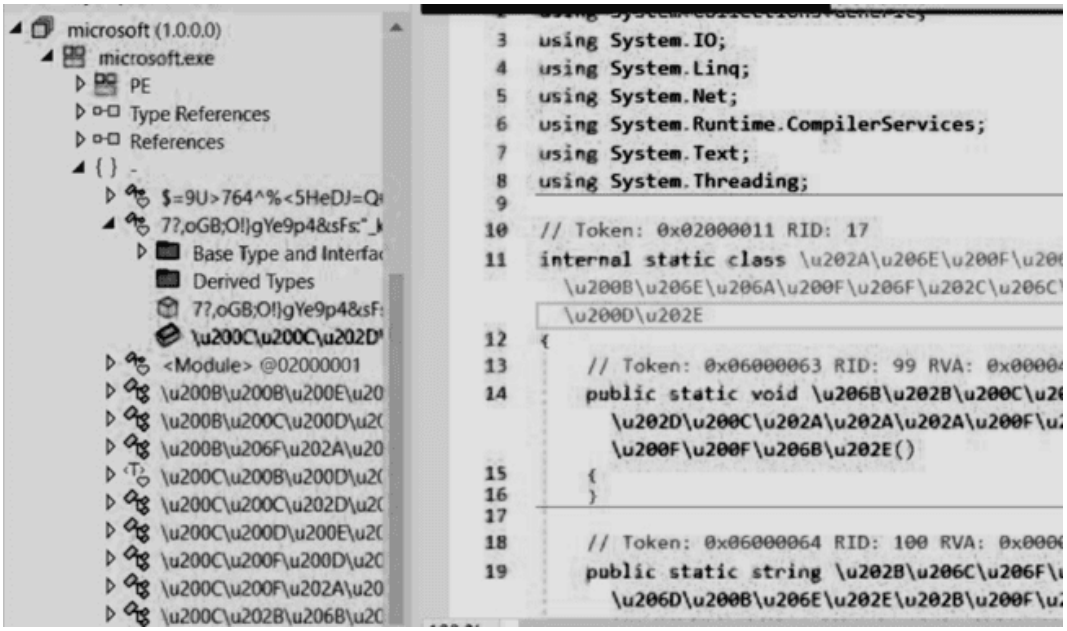


Figure 12. Anti-Tampering Mechanism of 0debug.exe, applied before deobfuscation

From Figure 11, the 0debug Stealer appears to be obfuscated, as indicated in Figure 12, and the binary's flow cannot be produced. The file exhibits a self-modifying section and unknown executable characteristics, suggesting the use of a packer or protector. The obfuscated function uses Unicode to obfuscate the binary. After deobfuscation, the execution flow of the debug stealer is identical to that shown in Figure 12.

1)VM & Network Sniff Check: Upon removal of the Anti-Tampering mechanisms and deobfuscation of the code, the program begins execution by invoking the VirtualProtect function, as shown in Figure 13.

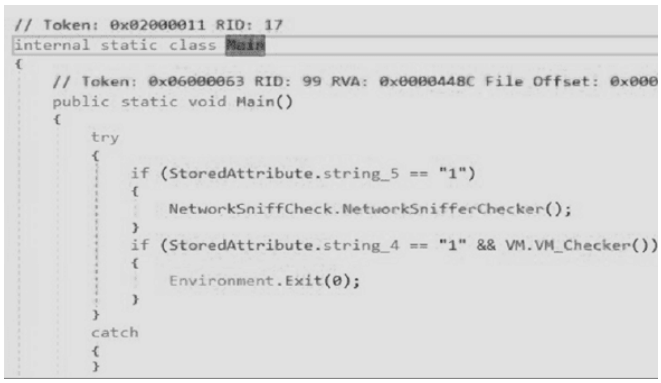


Figure 13. Implementation of Anti-Sniffer and Anti-VM on 0debug Stealer from the main function

From Figure 13, the observed function likely modifies memory permissions, allowing the code to execute self-modifying instructions or dynamically decrypt additional payloads.

The initial checks include:

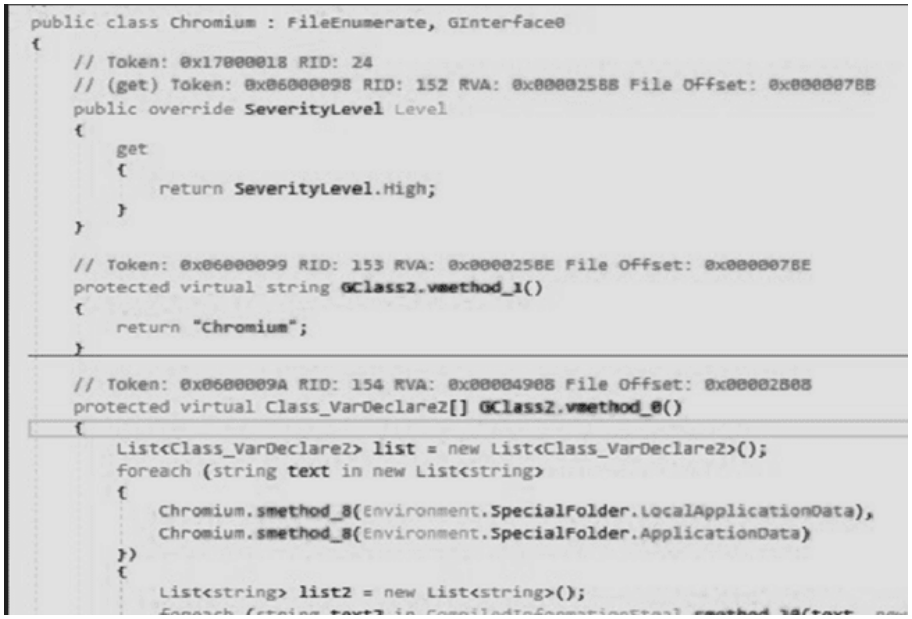
- **Anti-Sniffer Detection**: NetworkSniffCheck.NetworkSnifferChecker() is triggered if StoredAttribute.string_5 equals "1". This function is designed to detect network sniffers, tools that capture and analyze network traffic. By detecting and preventing the use of such tools, the malware aims to remain undetected during the transmission of sensitive data. The detection mechanism leverages predefined attributes to identify potential network sniffing activities, ensuring the malware cannot be captured by network monitoring tools.
- **Anti-VM Checks**: If StoredAttribute.string*4 equals "1" and the Virtual Machine detection function VM.VM*Checker() returns true, the program forcibly exits via Environment.Exit(0). This check is crucial to ensure the malware does not run in a virtual environment, which is commonly used for malware analysis and reverse engineering. The VM.The VM_Checker() function examines various system attributes, such as manufacturer names and model identifiers, to determine if the system is running in a virtualized environment. Common indicators include terms like "Microsoft Corporation" combined with "VIRTUAL" in the model's name, or explicit mentions of "VMware" or "VirtualBox". These terms are commonly associated with virtualization environments, such as VMware Workstation and VirtualBox.

The Anti-VM method involves checking specific system attributes, such as the Manufacturer and Model fields, to identify common indicators of virtual machines. If any of these conditions are met, indicating the presence of a virtual machine, the function returns true, signalling that the malware is running in a sandbox or virtual analysis setup. This ensures that the malware remains undetectable and unanalyzable in such environments. After passing both checks, the malware will declare the C2 address that will be used at the end for the Info payload submission. This declaration is essential for the malware to communicate with its Command and Control (C2) server once it has gathered the required information.

2)Host Enumeration: 0debug Stealer then performs extensive reconnaissance and data collection on compromised systems to gather critical information. It targets both system-level details (such as OS version, running processes, and installed antivirus programs) and browser-specific data, application (Telegram, Discord & Steam), cryptocurrency wallets, file documents, and VPN configuration.

Like generic stealers, 0debug Stealer was designed to collect detailed data about the compromised system. This includes critical information such as the username, hostname, IP address, PC name, and organization name, which helps the malware identify and fingerprint the target machine. Additionally, it gathers system-level details, such as the OS version, Windows build, processor, video card, and PC HWID (Hardware ID), to build a comprehensive system profile. All this information is bundled into a structured output, often saved as a file (e.g., Computer Info.txt) for later exfiltration. This structured format facilitates efficient data handling and analysis by the malware operators.

3)Browser Enumeration: It appears that the 0debug Stealer targets both Chromium and Gecko-based browsers. The 0debug Stealer will have access to Chrome, Edge, and Opera, all of which use the Chromium engine. This behavior can be seen in evidence of the code and its execution, as shown in Figure 14.



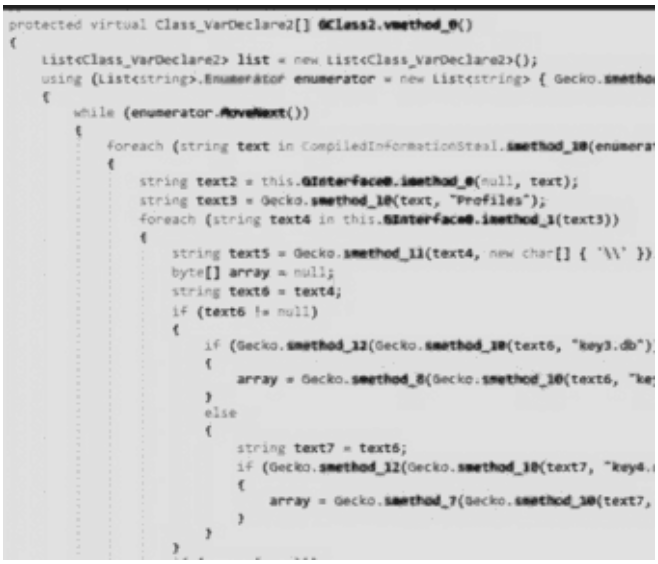
```
public class Chromium : FileEnumerate, GInterface0
{
    // Token: 0x17000018 RID: 24
    // (get) Token: 0x06000098 RID: 152 RVA: 0x0000258B File Offset: 0x0000078B
    public override SeverityLevel Level
    {
        get
        {
            return SeverityLevel.High;
        }
    }

    // Token: 0x06000099 RID: 153 RVA: 0x0000258E File Offset: 0x0000078E
    protected virtual string GClass2.vmethod_1()
    {
        return "Chromium";
    }

    // Token: 0x0600009A RID: 154 RVA: 0x00004908 File Offset: 0x00002B08
    protected virtual Class_VarDeclare2[] GClass2.vmethod_0()
    {
        List<Class_VarDeclare2> list = new List<Class_VarDeclare2>();
        foreach (string text in new List<string>
        {
            Chromium.smethod_8(Environment.SpecialFolder.LocalApplicationData),
            Chromium.smethod_8(Environment.SpecialFolder.ApplicationData)
        })
        {
            List<string> list2 = new List<string>();
```

Figure 14. Chromium code implementation enumeration in 0debug Stealer

As shown in Figure 14, the malware employs techniques such as running a headless browser to retrieve cookies from Chromium browsers. This approach allows the malware to gather sensitive data without a visible browser window, thereby reducing the risk of detection. After that, the 0debug Stealer triggers and collects wallet data from browser extensions. This process involves extracting data stored in browser extensions, which often contain valuable information such as login credentials and financial details.

0debug Stealer malware also targets Gecko-based browsers, specifically Mozilla Firefox, to extract sensitive user information. It begins by locating browser profiles in the ApplicationData directory and enumerates the key files used for data collection. The example is shown in Figure 15.

The malware targets critical files such as key3.db or key4.db, which store encryption keys used to decrypt saved credentials. It also extracts cookies from cookies.sqlite and autofill or form history data from formhistory.sqlite. Extracted data, including passwords, cookies, and autofill information, is organized in structured formats and often saved in text files for exfiltration. This structured format ensures that the data can be easily processed and analyzed by the malware operators.



```
protected virtual Class_VarDeclare2[] GClass2.vmethod_0()
{
    List<Class_VarDeclare2> list = new List<Class_VarDeclare2>();
    using (List<string>.Enumerator enumerator = new List<string> { Gecko.smetho
    {
        while (enumerator.MoveNext())
        {
            foreach (string text in CompiledInformationSteal.smethod_10(enumera
            {
                string text2 = this.GInterface0.imethod_0(null, text);
                string text3 = Gecko.smethod_10(text, "Profiles");
                foreach (string text4 in this.GInterface0.imethod_1(text3))
                {
                    string text5 = Gecko.smethod_11(text4, new char[] { '\\' })
                    byte[] array = null;
                    string text6 = text4;
                    if (text6 != null)
                    {
                        if (Gecko.smethod_12(Gecko.smethod_10(text6, "key3.db")
                        {
                            array = Gecko.smethod_8(Gecko.smethod_10(text6, "ke
                        }
                        else
                        {
                            string text7 = text6;
                            if (Gecko.smethod_12(Gecko.smethod_10(text7, "key4.
                            {
                                array = Gecko.smethod_7(Gecko.smethod_10(text7,
                            }
                        }
                    }
```

Figure 15. Gecko code implementation enumeration in 0debug Stealer

4)Crypto Wallet Enumeration: 0debug Stealer also demonstrates its capability to target cryptocurrency wallets by scanning specific directories associated with popular wallet applications, such as Atomic, Electrum, Exodus, Guarda as shown in Figure 16.

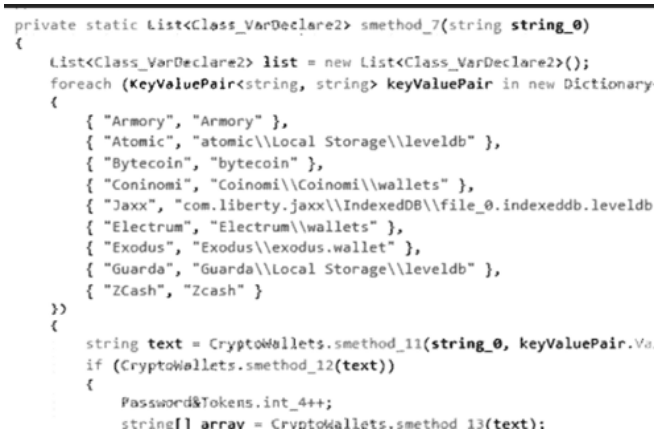


Figure 15. Popular crypto wallet dictionary for wallet enumeration function

From Figure 16, 0debug Stealer's function of Crypto wallet enumeration iterates through a predefined dictionary of wallet storage paths and attempts to locate the relevant files. Upon identifying the targeted files, the malware extracts sensitive information, including wallet keys, configuration data, and potentially stored credentials. It also employs decryption methods and functions to retrieve the data, which is subsequently added to an organized list for exfiltration. This process ensures that the malware can access and extract sensitive financial information from cryptocurrency wallets.

5)Application Enumeration: Additionally, 0debug Stealer collects data from Discord & Telegram software that are being used on the victim's computer, where it dumps chat application databases and their configurations. This data-collection strategy enables the malware to gather communication logs and metadata, which can be valuable for social engineering or further exploitation.

0debug Stealer can also gather files of interest from compromised systems. The malware's file-grabbing capabilities extend to well-known applications, including FileZilla, an FTP client widely used for file transfers. In FileZilla, the malware searches for configuration files such as sitemanager.xml and recentservers.xml, which contain FTP server credentials, usernames, and connection histories. The 0debug Stealer also employs recursive directory enumeration to identify and process target files. It specifically targets file extensions such as *.txt, *. seed, *.dat, *.mafile, and *.key, which are often used for credentials, configuration files, and sensitive data, such as cryptographic keys. Once located, the malware retrieves and processes file contents, storing the extracted data for further exfiltration.

6)VPN Enumeration: 0debug Stealer incorporates techniques to gather VPN configuration data from compromised systems [19], specifically targeting popular VPN applications such as Surfshark, ProtonVPN, and OpenVPN.

These actions allow the malware to compromise VPN credentials, potentially exposing encrypted network traffic or enabling unauthorized access to private VPN tunnels. For Surfshark, the malware searches within the application data directory for configuration files with a .dat extension. If the relevant files are found, the malware extracts and processes them, storing the decrypted data for exfiltration.

Similarly, for ProtonVPN, the stealer navigates to the local application data and iterates through potential configuration files, such as user.config. The report shows that ProtonVPN has details of memory protection flaws in ProtonVPN and Proton Pass, exposing sensitive data to credential-stealing malware [19]. Once located, the malware attempts to decrypt the files using memory management techniques and stores the obtained credentials or configuration details.

For OpenVPN, the malware specifically searches the OpenVPN Connect directory in the application data folder, focusing on files under the "profiles" directory, and uses memory manipulation to decrypt user.config and similar structured data [20]. While earlier families like Raccoon primarily focused on financial data, modern variants like Vidar and Redline have expanded to include VPN credentials, given their high value for identity masking and network intrusion [11]. 32% of enterprise VPN breaches involve theft of .ovpn files, with OpenVPN being the most frequently targeted [21]. Once identified, it iterates through the potential configuration data, particularly targeting files with the. ovpn extension, which typically stores OpenVPN connection settings.

A comparative analysis of core exfiltration capabilities, as summarized in Table VI.

TABLE VI. DIFFERENT CAPABILITES COMPARED ACROSS DIFFERENT TYPE STEALER [5-6,20-23]

Stealing Capabilities	**0debug**	**Redline**	**Vidar**
Host Information	✓	✓	✓
Browser Data & Credential	✓	✓	✓
Browser Wallet Extensions	✓	✓	✓
Crypto Wallet	✓	✓	✓
File Collector Config	✓	✓	✓
VPN Config	✓	✓	✓
Telegram Data	✓	✗	✓
Discord Data	✓	✓	✓
Steam Data	✓	✓	✓
Cloud Credentials	✗	✗	✓

Table VI reveals that 0debug Stealer shares a foundational feature set with modern stealers, such as Redline and Vidar [9-10, 22-23]. All three families systematically harvest host information, browser credentials and wallets, cryptocurrency wallet data, application configurations (including Discord and Steam), and VPN settings. However, note that there are some differences in its capabilities on Telegram and Cloud Credentials because it was being built by different operators.

V. CONCLUSIONS

This study used assisted static analysis to investigate the 0debug Stealer, effectively dissecting its binary without execution by using fingerprinting, decompilation, and deobfuscation. The analysis verified that the malware used advanced obfuscation and preliminary anti-analysis checks (Anti-VM, Anti-Sniffer) to obstruct the investigation. System information, login credentials from Chromium/Gecko browsers, cryptocurrency wallets, data

from apps (Telegram, Discord, Steam), and configuration files from VPN clients such as OpenVPN, ProtonVPN, and Surfshark were all found to have been extensively harvested.

The results have obvious practical and forensic implications, even though the creation of specific detection signatures (such as YARA rules) is outside the scope of this work. The thorough mapping of 0debug's evasion strategies, obfuscation tactics, and specific data targets gives forensic analysts and threat hunters useful information.

The research underscores the indispensable role of static analysis in the foundational dissection of such malware. Future work should focus on dynamic analysis of the deobfuscated sample to observe runtime behavior and network activity, which will be crucial for developing more robust, behavior-based detection and mitigation strategies.

CONFLICT OF INTEREST

The authors declare that there is no conflict of interest regarding the publication of this paper.

ACKNOWLEDGEMENT

This research was funded by the Ministry of Higher Education (MOHE) Malaysia under the Fundamental Research Grant Scheme (FRGS/1/2020/ICT02/USIM/02/1). The authors would like to express their gratitude to Universiti Sains Islam Malaysia (USIM) and MOHE for the support and facilities provided.

REFERENCES

[1] Europol, Internet Organised Crime Threat Assessment (IOCTA) 2024, Europol Public Information, 2024, doi: 10.2813/442713.

[2] N. Saurabh, "Advance malware analysis using static and dynamic methodology," *2018 International Conference on Advanced Computation and Telecommunication (ICACAT)*, pp. 1–5, Dec. 2018, doi: 10.1109/icacat.2018.8933769.

[3] A. Afianian, S. Niksefat, B. Sadeghiyan, and D. Baptiste, "Malware dynamic Analysis Evasion techniques," *ACM Computing Surveys*, vol. 52, no. 6, pp. 1–28, Nov. 2019, doi: 10.1145/3365001.

[4] R. Nair, K. R. Dodiya, and P. Lakhalani, "A Static Approach for Malware Analysis: A guide to analysis tools and techniques," International Journal for Research in Applied Science and Engineering Technology, vol. 11, no. 12, pp. 1451–1474, Dec. 2023, doi: 10.22214/ijraset.2023.57649.

[5] N. N. Widiyasono, N. S. R. Selamat, N. A. Sinjaya, N. Rianto, N. R. Rizal, and N. M. Praseptiawan, "Investigation of malware Redline Stealer using static and dynamic analysis method Forensic," *Journal of Advanced Research in Applied Sciences and Engineering Technology*, vol. 48, no. 2, pp. 49–62, Jul. 2024, doi: 10.37934/araset.48.2.4962.

[6] F. Ramadan and I. R. Hikmah, "Redline Stealer Malware Analysis with Surface, Runtime, and Static Code Methods," *IEEE International Conference on Cryptography, Informatics, and Cybersecurity (ICoCICs) 2023*, pp. 205–211, Aug. 2023, doi: 10.1109/icocics58778.2023.10276709.

[7] C. Rathnayaka and A. Jamdagni, "An Efficient Approach for Advanced Malware Analysis Using Memory Forensic Technique," *IEEE Trustcom/BigDataSE/ICESS*, Aug. 2017, doi: 10.1109/trustcom/bigdatase/icess.2017.365.

[8] A. K. Sood, S. Zeadally, and R. Bansal, "Cybercrime at a Scale: A Practical study of deployments of HTTP-Based Botnet command and control Panels," IEEE Communications Magazine, vol. 55, no. 7, pp. 22–28, Jan. 2017, doi: 10.1109/mcom.2017.1600969.

[9] R. Team, "Redline Stealer," *Cyberint*, Aug. 09, 2023. https://cyberint.com/blog/research/redline-stealer/

[10] G. Tubin, "CyOPS Lighthouse: Vidar Stealer," *Cynet Unified, AI-Powered Security Platform for MSPs & SMEs*, Nov. 19, 2025. https://www.cynet.com/blog/cyops-lighthouse-vidar-stealer/

[11] Stealer Malware Evolution: 2024 Threat Landscape, ENISA Tech. Brief, Sept. 2024. [Online] Available: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2024

[12] VirusShare.com, "VirusShare - Because Sharing is Caring," VirusShare, [Online]. Available: https://virusshare.com/. [Accessed: Jan. 21, 2026].

[13] E. Carrera and G. Erdélyi, Digital Code Forensics. Cham, Switzerland: Springer, 2020. [Online]. Available: https://link.springer.com/book/10.1007/978-3-030-44701-4. ISBN: 978-3-030-44700-7

[14] de4dot, ".NET Deobfuscator," GitHub Repository, 2020, [Online], Available: https://github.com/de4dot/de4dot

[15] M. Sikorski and A. Honig, Practical Malware Analysis, Ch. 4, No Starch Press, 2012, ISBN: 978-1-59327-290-6. [Online]. Available: https://nostarch.com/malware

[16] R. González Arias, J. Bermejo Higuera, J. J. Rainer Granados, J. R. Bermejo Higuera, and J. A. Sicilia Montalvo, "Systematic Review: Anti-Forensic Computer Techniques," Applied Sciences, vol. 14, no. 12, Art. 5302, 2024. doi: 10.3390/app14125302

[17] [23] M. Eckardt, ".NET Deobfuscation Techniques and Tooling," cyber.wtf, Apr. 7, 2025. [Online]. Available: https://cyber.wtf/2025/04/07/dotnet-deobfuscation

[18] M. Egele et al., "A Survey on Automated Malware Analysis," ACM Computing Surveys, vol. 44, 2008. doi: 10.1145/2089125.2089126

[19] A. Mishra, "500 Million Proton VPN & Pass Users at Risk Due to Memory Protection Vulnerability," GBHackers on Security, Jan. 30, 2025. [Online]. Available: https://gbhackers.com/500-million-proton-vpn-pass-users

[20] Guru Baran, "OpenVPN Connect Vulnerability Let Attackers Access Users' Private Keys," Cybersecurity News, Jan. 7, 2025. [Online]. Available: https://cybersecuritynews.com/openvpn-connect-private-key

[21] Microsoft Threat Intelligence, "Chained for Attack: OpenVPN Vulnerabilities Discovered Leading to RCE and LPE," Microsoft Security Blog, Aug. 8, 2024. [Online]. Available: https://www.microsoft.com/en-us/security/blog/2024/08/08/chained-for-attack-openvpn-vulnerabilities-discovered-leading-to-rce-and-lpe

[22] Trend Micro - United States (US), "Fast, Broad, and Elusive: How Vidar Stealer 2.0 Upgrades Infostealer capabilities," *Trend Micro*, Oct. 21, 2025. https://www.trendmicro.com/en_us/research/25/j/how-vidar-stealer-2-upgrades-infostealer-capabilities.html

[23] "Technical analysis of the RedLine Stealer," *CloudSEK*, Aug. 21, 2025. https://www.cloudsek.com/blog/technical-analysis-of-the-redline-stealer