Learn how Unknown Cyber used a combination of automated shared-code analysis and ChatGPT to uncover a potential supply chain attack.

Marrying Unknown Cyber and LLM to Detect Supply Chain Attacks

August 6, 2024

Table of Contents

Executive Summary

A few weeks ago, UC received a NEW DLL from a customer using our services for software supply chain audit. Our system, UC MAGIC, which uses code-similarity based search, detected that this DLL was 0.9886-level similar to a DLL known to be malicious (MALWARE DLL). MAGIC measures similarity on a scale from 0 to 1 by comparing the 'genome' extracted from the disassembled code of binaries. A genomic difference analysis between the NEW and MALWARE DLLs revealed that the MALWARE DLL contained 21 additional functions that were not present in the NEW DLL.

An analysis of these 21 additional functions by ChatGPT 4o uncovered capabilities corresponding to nine Tactics Techniques & Procedures (TTPs) in the MITRE ATT&CK framework (see Appendix B). By tracing the code connections in our shared-code repository, we discovered 12 other DLLs containing functions identified as malicious by ChatGPT 4o. Additionally, we created a Yara rule from the bytecode of these malicious functions and used it to retro hunt on Hybrid Analysis. The hunt identified 13 more DLLs containing the same malicious code.

As all the DLLs identified are compromised versions of popular DLLs, including 22 from Microsoft, we are likely looking at a supply chain attack similar to the SolarWinds incident [1,2]. Fortunately, unlike the compromised SolarWinds DLLs, these trojans are not signed. However, this is only a slight reassurance since, according to VirusTotal, 10 out of the 26 trojans are not detected as malicious by over 50% of its antivirus scanners.

Furthermore, the 26 trojans do not belong to a single malware family but are associated with various information stealers, including Agent Tesla, AsyncRAT, DCRat, Formbook, NjRAT, Raccoon, RedLine, Snake, and Vidar. This shared code among different malware families implies a sharing of resources between the threat actors behind them. These threat actors may either be managed by a single entity or use a common third-party tool or service to infect DLLs. In either case, our Yara rule can be used to scan for trojans infected by the same mechanism, including those not yet identified and reported to large malware repositories.

In summary, this case study demonstrates that by marrying automated shared-code analysis with LLMs it is feasible to cost-effectively audit software and software updates for the presence of malicious or otherwise unexpected capabilities. Starting with one NEW DLL, we identified 26 Trojan DLLs by utilizing UC's ability to rapidly search a large repository of executables based on code similarity, perform pairwise differences to identify common and differing code, analyze disassembled code for malicious behavior, identify specific malicious functions, trace shared code in the repository, and generate Yara rules from the bytecode of specific functions. All this was be done without using sandboxes or expertise in reverse engineering.

Recommendations to use the combined power of UC and LLM for post-release audits of software and Software Bill of Materials (SBOMs) are provided.

The Trigger

Global Data Systems (GDS), a Managed Security Service Provider, relies on Unknown Cyber (UC) for software supply chain audits to prevent SolarWinds-like supply chain attacks on their customers' networks. They employ state-of-the-art malware prevention technologies at all critical points where files can be downloaded, read, written, or executed. Despite these efforts, they encounter 70-100 files daily for which these technologies cannot definitively determine if they are benign or malicious even after detonating in a sandbox. Files with an 'unknown' verdict are sent to UC for further analysis to resolve these uncertainties.

In June 2024, the GDS uploaded a DLL named Newsoft.JSON.DLL "(first SHA256 hash in Table 1)". This popular .NET DLL is used for serializing and deserializing JSON data. The DLL was intercepted by GDS on a developer's machine when the developer activated the related package in Visual Studio. As this was the first use of this DLL by the developer, it was not on the MSSP's 'permitted programs' list and received an 'unknown' verdict from their anti-malware system.

UC employs a code-proximity based search to automatically identify programs in its repository that share a significant amount of code with the unknown program. The reputation of these similar files is retrieved from VirusTotal. If any of the files are malicious, an in-depth analysis is triggered.

Figure 1 diagrammatically represents the process of determining potentially malicious code using code proximity. UC's code-proximity based search found nine programs in its repository with a 0.7 or higher level of code similarity to the NEW DLL. UC computes code similarity as a measure between 0 and 1 based on the amount of code shared between two binaries. Table 1 presents the results of the search, with the first SHA256 hash being that of the NEW DLL. The remaining hashes are listed in decreasing order of similarity.

Figure 1
Checking for Potentially Malicious Code using Code Proximity

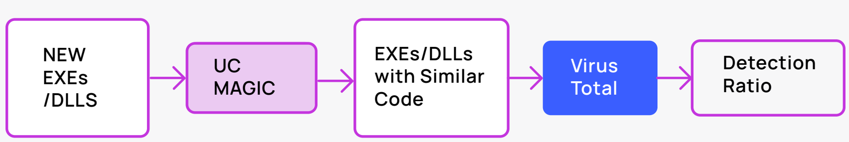


Table 1 also shows the AV Detection Ratio for each hash as reported by VirusTotal. None of the 74 scanners (0/74) flagged NEW DLL as malicious. Similarly, seven other files are not flagged malicious by any scanner. However, two hashes, second and ninth, are flagged as malicious by 51/74 and 35/71 scanners. The DLL in the second row, henceforth referred to as MALWARE DLL, also has the highest code similarity with NEW DLL.

This discrepancy was the trigger. Why are the two files, NEW DLL and MALWARE DLL, assigned opposite reputations—one benign (or more precisely, not flagged as malicious) and the other malicious—when they share 0.9886 level similar code? There are three possibilities:

1. UC is wrong:

NEW DLL and MALWARE DLL do not actually share code at any significant level.

2. VirusTotal is wrong:

The reputation assigned by VirusTotal to one of the files is incorrect. If so, which one?

3. Neither UC nor VirusTotal are wrong:

MALWARE DLL is a trojanized version of NEW DLL.

Table 1
The UC MAGIC matches that triggered this analysis

SHA256	Similarity	AV Detection Ratio
b624949df8b0e3a6153fdfb730a7c6f4990b6592ee0d922e1788433d276610f3 (NEW)	self	0/74
15bad895c6afb47d3dbf662a5743d49ce0bba45b110b494645d92b2db423ac4c (MALWARE)	0.9886	51/74
5110b8934e6db5a5f990829c445829df09e29c5e0cd9fb6253709344e9d1a5d3	0.9399	0/66
7ea00ce56000a486b59d5f411791af562c2c2f7d2c9de05930d97f4efbcb373d	0.9399	0/64
c5c83bbc1741be6ff4c490c0aee34c162945423ec577c646538b2d21ce13199e	0.9301	0/71
e1e27af7b07eeedf5ce71a9255f0422816a6fc5849a483c6714e1b472044fa9d	0.9297	0/70
22c649f75fce5be7c7ccda8880473b634ef69ecf33f5d1ab8ad892caf47d5a07	0.9047	0/48
0d3d349ba4887068a012cc4dc16dc1e7ca11245816a01fb254009e5c8958b829	0.8377	0/69
8d29d1cb1bb450bfee7b3e9b1dfb00372e25fb6dc88d9bfa33bdc3d78adfd0eb	0.7927	35/71
d52dc9db3cfa3131926fcb6dedd68d0a8be3413ec38210b262def777b1f3cf7c	0.7794	0/70

Confirm the Presence of Supply Chain Attack

One way to resolve the reason for the discrepancies in the reputation assignment of the two DLLs with similar code is to review their metadata, as shown in Figure 2.

The first possibility, "UC is wrong," can be quickly dismissed (or at least, probabilistically put to rest) from the metadata of these files. Both the NEW DLL and MALWARE DLL share two significant attributes: they are authored by James Newton King and have the filename NewtonSoft.JSON.DLL. UC's similarity assessment does not utilize metadata, so the odds that it randomly identified two files with the same author and filename from millions in VirusTotal's database when these files do not share code, are astronomically low. Moreover, this isn't limited to just the NEW DLL and MALWARE DLL—the remaining eight files also share the same author and filename. Thus, the probability of UC being wrong is negligibly small.

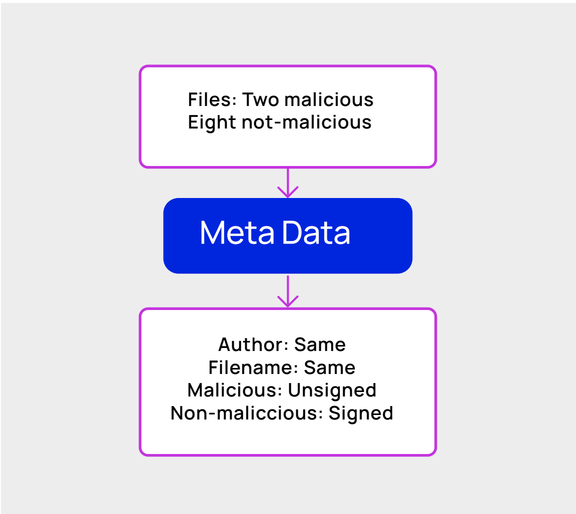


Figure 2
Quick Resolution of Hypotheses using Metadata

The metadata also provides evidence that counters the second possibility, "VT is wrong." It shows that while all eight files with zero detections are signed with valid signatures, the two files flagged as malicious are not signed. What is the likelihood that a company known for releasing signed DLLs would also release two unsigned DLLs? There are several scenarios where this might happen, but one plausible explanation is that the company did not release these unsigned DLLs—someone else did. Thus, VT is most likely right that the NEW DLL is not malicious.

This leads to the third possibility, "MALWARE DLL is a trojanized version of NEW DLL." The fact that the two malicious DLLs in Table 1 have very similar code to legitimate DLLs and have the same filename and author suggests an attempted supply chain attack akin to the notorious Sunburst attack. In that incident, a legitimate SolarWinds program used by many large enterprises worldwide was compromised and distributed through normal software update channels.

The data thus far appears to point to an attempt at a supply chain attack, albeit not on the MSSP customer's network. It appears that someone may be leveraging NewSoft.JSON.DLL's popularity to penetrate a supply chain. That the malicious DLLs were not signed indicates that the attack may not be as sophisticated as Sunburst. It could be that the attack is delivered through pirated software downloads or other unsigned programs.

What Does the Trojan Code Do?

Having established, albeit probabilistically, that the NEW DLL is not compromised, we still need to understand what the MALWARE DLL does. Since NEW DLL and MALWARE DLL have a 0.9886 similarity in code, it is essential to analyze the unique code present in MALWARE DLL but absent in NEW DLL.

We follow a two-step process to accomplish this, as shown in Figure 3. First, we identify the code that is only in MALWARE DLL. Second, we request an analysis from ChatGPT.

Figure 3 Identifying Trojan Code & Determining What it Does

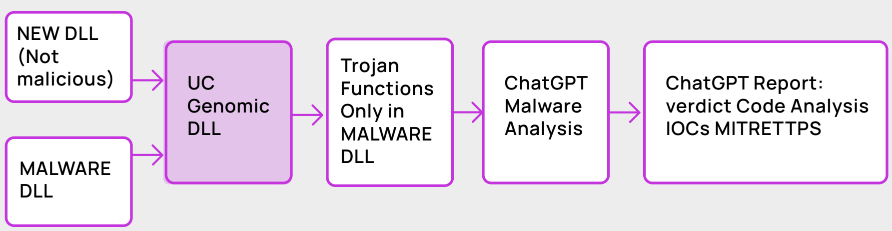


Table 2 Comparison of Functions Genomes of NEW DLL and MALWARE DLL

Functions	Count
In NEW DLL	1,826
In MALWARE DLL	1,847
Union of the two DLLs	1,847
Intersection of the two DLLs	1,826
Only in NEW DLL	0
Only in MALWARE DLL	21

We use UC's Genomic Diff to identify the functions unique to MALWARE DLL. Table 2 summarizes the results of the Genomic Diff of the two samples. The data shows that all the functions in NEW DLL are entirely subsumed in MALWARE DLL. There are no functions in NEW DLL that are not in MALWARE DLL, but there are 21 functions in MALWARE DLL that are not in NEW DLL. This strengthens the possibility that MALWARE DLL was created by inserting code into a copy of NEW DLL.

Table 3
Functions unique to MALWARE DLL

Table 3 lists the 21 functions unique to MALWARE DLL. It presents each function's Name, the Namespace, and the Class it belongs to. All the names follow a pattern: either a meaningful word, like `Main` and `get_ResourceManager`, or a meaningful word prefixed by three arbitrary letters, like `Yqnpressure` and `tndthemselves`. The 3-letter prefix often encodes information relevant to the compiler, while the meaningful word typically indicates the purpose of the namespace, class, or method. Words like `pressure`, `credit`, `drink`, `beer`, and `Sexrecipe`, used in Table 3, seem out of place in a library that serializes and deserializes JSON, suggesting an attempt to obfuscate the functions' purposes. Such obfuscation is commonly done programmatically when a DLL is bundled for shipping.

RVA	Namespace	Class	Name
0x37f70	Yqnpressure	Ujbexpert	.ctor
0x37fe0	Yqnpressure	Ujbexpert	Dmcmargin
0x38000	Yqnpressure	Ujbexpert	Akucotton
0x38030	Yqnpressure	Ujbexpert	Dppproducer
0x38060	Yqnpressure	Ujbexpert	Dwgsurely
0x380a0	Hdbcredit	Tndthemselves	Vpechairman
0x380f0	Epfgreatest.Ztdvoter	Static	Main
0x38160	Cllindeed	Lqphall	get_ResourceManager
0x381b0	Cllindeed	Lqphall	set_Culture
0x381d0	Epfgreatest.Properties	Sexrecipe	get_Default
0x381f0	Epfgreatest.Properties	Sexrecipe	.cctor
0x38220	Eixdrink	Ogksubstantial	.ctor
0x38230	Eixdrink	Ogksubstantial	Fmfcreate
0x38260	Eixdrink	Ogksubstantial	Hohadvice

0x382d0	Epfgreatest.UndoRedo	Mwztarget	.ctor
0x382f0	Epfgreatest.UndoRedo	Mwztarget	Ehgflee
0x38310	Epfgreatest.UndoRedo	Mwztarget	Vuuown
0x38330	Epfgreatest.UndoRedo	Mwztarget	Kcldrama
0x38420	Asvbeer	Isveventually	Plnarrive
0x384e0	Advinstruction	Fjyfact	Fikcut
0x38510	Advinstruction	Fjyfact	.ctor

Next, we gave ChatGPT 4o the disassembled code of the 21 functions in the above table along with the following prompt (taken verbatim from [3]).

"Act as a malware analyst by thoroughly examining this decompiled executable code. Methodically break down each step, focusing keenly on understanding the underlying logic and objective. Your task is to craft a detailed summary that encapsulates the code's behavior, pinpointing any malicious functionality. Start with a verdict (Benign or Malicious), then a list of activities including a list of Indicators of Compromise (IOCs) if any URLs, created files, registry entries, mutex, network activity, etc."

ChatGPT concluded that the 21 functions represented malware and provided the following summary:

The code exhibits characteristics typical of malware, including obfuscation, dynamic assembly loading, and state manipulation. The extensive use of late binding and reflection further suggests attempts to evade static analysis. Therefore, it is highly recommended to treat this executable as malicious and perform further dynamic analysis to uncover its full behavior.

Specifically, ChatGPT identified the following malicious behaviors:

- **Obfuscation and Decryption:** The `Asvbeer.Isveventually::Plnarrive` function suggests obfuscation techniques, involving base64 encoding and XOR operations to manipulate byte arrays, potentially indicating payload decryption.
- **Dynamic Assembly Loading:** The `Epfgreatest.Ztdvoter::Main` function dynamically loads assemblies and executes methods, which is a common technique used in malware to execute payloads at runtime.

The complete response from ChatGPT is provided in Appendix A.

Subsequently, we asked ChatGPT to map the malicious behaviors to the MITRE ATT&CK TTPs. ChatGPT identified nine MITRE ATT&CK TTPs, including dynamic component loading, obfuscation with base64 encoding and XOR mask, and process injection. The complete response from ChatGPT is provided in Appendix B.

Expand Search Through Sharing of Malicious Code

The fact that MALWARE DLL is unsigned suggests that James Newton King's infrastructure was unlikely compromised, indicating the attack may have occurred post-release. The trojan code, consisting of the 21 functions and associated data, could have been introduced using a .NET DLL editing tool like Resource Hack. If the trojan variant of NewSoft.JSON.DLL was created by injecting code into a legitimate copy of NewSoft.JSON.DLL, it's plausible that other DLLs could be similarly compromised at minimal additional cost. Consequently, we can expect to find trojan versions of other legitimate DLLs.

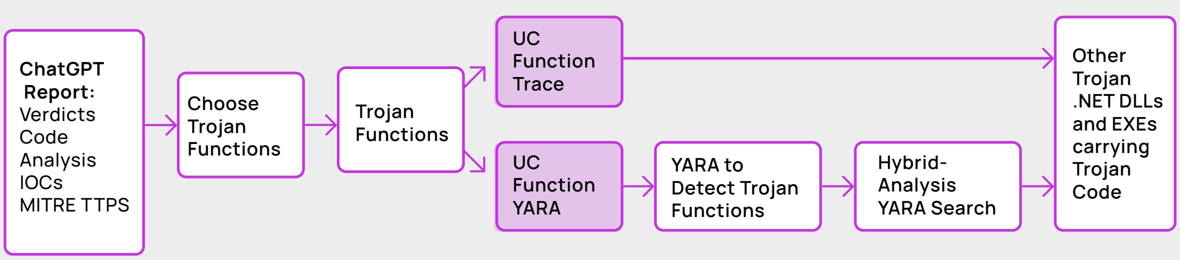
Our next step is to identify other DLLs containing the same trojan code. We do this by first identifying some interesting trojan code using ChatGPT's analysis and then using two parallel steps: search UC's repository and search third-party repositories. Figure 4 summarizes the process followed, with detailed steps presented below.

The ChatGPT report had identified the following to functions to a play a role in decoding and injecting the payload carried by the trojan:

- Asvbeer.Isveventually::Plnarrive (0x38420)
- Epfgreatest.Ztdvoter::Main (0x380f0)

Using UC Function Trace, we found other files in our repository containing these two trojan functions. Finding such DLLs in UC MAGIC's database is straightforward since the system maintains a bipartite graph with two classes of nodes: files and functions. A file in this graph is connected to all the functions it contains (and equivalently, a function is connected to all the files it is contained in). By traversing this graph, we found 13 other DLLs that shared the trojan functions of the original MALWARE DLL.

Figure 4
Identifying Trojan Code & Determining What it Does



In parallel, we used UC Function Yara to create a specialized Yara rule from the bytecode of the two trojan functions. The resulting Yara rule is presented in Figure 5. This rule was used to perform a retro hunt on Hybrid Analysis to find DLLs and EXEs in its repository.

Yara search on Hybrid Analysis using the rule yielded 26 .NET DLLs presented in Table 4 (Appendix C). These include the 13 DLLs we found through UC Function Trace. This table confirms the earlier hypothesis that if the compromise occurred outside of James Newton King's environment, then the attacker is unlikely to have singled out NewSoft.JSON.DLL. The 26 files are from different publishers, with the vast majority from Microsoft.

Table 4 reveals something particularly interesting and possibly unexpected. The "Family" column lists the malware family associated with each DLL. The intriguing part is that the 26 files do not all belong to the same family. Though the MALWARE DLL (fourth row) that triggered this investigation is classified under the Snake ransomware family, the others are associated with various families, most notably Agent Tesla (6), FormBook (8), and Vidar (5).

With this data, we can reasonably conclude that the analysis has revealed a potential supply chain attack since the trojan code from the first compromised DLL, MALWARE DLL, is found in 25 other DLLs, all from different applications. Although this attack may not be as sophisticated as the SolarWinds attack since the trojan DLLs are not signed, it remains worrisome.

Figure 5 YARA Generated from Bytecode of Trojan Functions in MALWARE DLL

```
rule UnknownCyber_NewsoftJsonTrojan
{
    meta:
        date = "July 06, 2024"
        author = "Unknown Cyber, Inc"
        description = "Created from trojan code in a compromised NewSofJSON.DLL"
        sample_hash = "15bad895c6afb47d3dbf662a5743d49ce0bba45b110b494645d92b2db423ac4c"
    strings:
        $_48b5c3dcbe9ec0dea2ea8738a8f634d3 = {
                00 73 ?? ?? ?? ?? 28 ?? ?? ?? ?? 6F ?? ?? ?? ??
                6F ?? ?? ?? ?? 28 ?? ?? ?? ?? 0A 73 ?? ?? ?? ??
                06 73 ?? ?? ?? ?? 28 ?? ?? ?? ?? 6F ?? ?? ?? ??
                6F ?? ?? ?? ?? 6F ?? ?? ?? ?? 28 ?? ?? ?? ?? 00
                2A }
        $_4c60e972512feee4eeea489799cf0f09 = {
                20 ?? ?? ?? ?? 8D ?? ?? ?? ?? 25 ?? ?? ?? ?? ??
                28 ?? ?? ?? ?? ?? ?? ?? ?? ?? 00 72 ?? ?? ?? ??
                ?? ?? ?? ?? ?? ?? 7E ?? ?? ?? ?? 0B 07 8E 69 1A
                5A 8D ?? ?? ?? ?? 0C 07 16 08 16 07 8E 69 1A 5A
                28 ?? ?? ?? ?? 00 28 ?? ?? ?? ?? 08 6F ?? ?? ??
                ?? 28 ?? ?? ?? ?? 0D 16 13 05 2B 21 00 09 11 05
                06 11 05 06 6F ?? ?? ?? ?? 5D 6F ?? ?? ?? ?? 09
                11 05 91 61 D2 9C 00 11 05 17 58 13 05 11 05 09
                8E 69 FE 04 13 06 11 06 2D D2 09 73 ?? ?? ?? ??
                73 ?? ?? ?? ?? 13 04 11 04 13 07 2B 00 11 07 2A }
    condition:
        all of them
}
```

VirusTotal flagging the 26 trojan DLLs as malicious is beneficial for threat research, but antimalware scanning on desktops and servers is typically done by one antimalware scanner, not a suite of 70 scanners like VirusTotal. Table 4 shows that, at the time of writing, 10 out of the 26 trojan variants of legitimate DLLs were detected by less than half of VirusTotal's antivirus scanner suite. In other words, more scanners do not detect 10 malicious DLLs than those that do. Despite this, attacks from these trojans can be thwarted if the system is locked down to disallow unsigned executables from executing. However, this may be easier said than done since many commercial and open-source programs are distributed unsigned.

[4-26]. The fact that the same trojan code is used to deliver different payloads suggests either a single threat actor group is behind these malware families, or different threat actors have obtained the DLL infection mechanism from a common third-party source.

This case study underscores the power of analyzing shared code at scale, as UC has done. We utilized code similarity to identify DLLs with shared code, genomic difference to locate trojan code, ChatGPT to analyze the trojan code, and bytecode-based Yara rules based on the trojan code to detect not just a single malware family but a malware delivery mechanism used by numerous malware families.

Summary

This case study demonstrates the feasibility and effectiveness of tracing shared code across large program repositories. By starting with a single DLL, even one that turned out to be non-malicious, we were able to uncover a potential supply chain attack through shared code analysis. The Yara rule produced by UC from shared code proved to be a highly accurate IOC.

Our analysis also uncovered intriguing connections among several seemingly unrelated malware families: Agent Tesla, AsyncRat, DCRat, Formbook, NjRat, Raccoon, RedLine, Snake, and Vidar

Challenge

For readers who enjoy Capture the Flag (CTF) challenges, we have a challenge for you. There is one hash notably absent from Table 4. Early indications suggest that investigating this missing hash could reveal a much more complex story. The first person to identify the missing hash is invited to join us in investigating it further.

Recommendations

Whether a software supply chain attack is executed by compromising the DevOps workflow of a vendor or by tampering with the code post-release, the following recommendations can help mitigate such risks. The first set of recommendations outlines basic best practices to limit the possibility of such attacks:

1. Limit Execution of Unsigned Code:

- Configure your environment to completely disallow the execution of unsigned code.
- When executing unsigned code is unavoidable, run it within additional security boundaries, such as isolating it in containers.

2. Allow Code Only from Approved Vendors:

- Implement strict controls to allow only signed code.
- Further restrict execution to signed code from a list of approved vendors.

However, as demonstrated by the SolarWinds attack, the existence of a valid code signature alone is not sufficient to guarantee that a program is uncompromised. Therefore, in line with the principle of "Trust but Verify," we recommend incorporating mechanisms to audit software updates:

3. Audit Software Updates for Deviation from Release Notes:

- Use code-differencing tools to compare successive updates and identify new and modified code.
- Use advanced tools like ChatGPT to assist in (a) identifying potential malicious code within the updates and (b) providing a report of capabilities introduced and modified in the update.
- Compare the ChatGPT produced analysis report against the vendor's release notes to identify any deviations or undocumented changes.

4. Audit Software Updates for Deviation from Software Bill of Material (SBOM).

- Identify components in delivered software by code-similarity with a catalogue of known software.
- Verify the SBOM against the components identified.

5. Audit Software Updates for Proximity to Malware:

- Using UC like code-similarity search find malware that share code with the update.
- Analyze the common code for maliciousness.
- Analyze the differing code in the malware to detect potential trojan code.
- Develop Yara signatures from trojan code for preventive defense.

6. Audit Software Updates for the Presence of Known Supply Chain Attacks:

- Maintain a repository of Yara signatures of known supply chain attacks.
- Search for these signatures within software updates.

By adopting these best practices and auditing mechanisms, organizations can significantly reduce the risk of software supply chain attacks and ensure a higher level of security for their software infrastructure.

References

For the benefit of future research, provided below are articles discussing the various malware families that have been found to have utilized the trojan DLL delivery mechanism unearthed in this study.

[Sunburst]

1. FireEye, SolarWinds Supply Chain Attack Uses SUNBURST Backdoor, Mandiant, December 13, 2020
2. Microsoft Threat Intelligence, Analyzing Solorigate, the compromised DLL file that started a sophisticated cyberattack, and how Microsoft Defender helps protect customers, Microsoft, Publish Date unknown, Last accessed July 15, 2024.

[Google AI]

3. Bernardo Quintero, From Assistant to Analyst: The Power of Gemini 1.5 Pro for Malware Analysis, Google Cloud Blog, April 29, 2024.

[Agent Tesla]

4. Bill Toulas, Google ads push ‘virtualized’ malware made for antivirus evasion, Bleeping Computer, February 2, 2023.
5. James Arndt, Understanding Agent Tesla: Notorious Keylogger, Cofense, February 21, 2023
6. Xiaopeng Zhang, New Agent Tesla Variant Being Spread by Crafted Excel Document, FortiGuard Labs, September 4, 2023.
7. Bernard Bautista, Agent Tesla’s New Ride: The Rise of a Novel Loader, Trustwave Spiderlab Blogs, March 26, 2024
8. David Brunsdon, Agent Tesla Unmasked: Revealing Interrelated Cyber Campaigns, HYAS, April 29, 2024.

[AsyncRAT]

9. Pedro Tavares, Unmasking AsyncRAT Malware: In-depth Analysis & Prevention Tips I Infosec, Infosec Institute, January 10, 2023.
10. Lakshya Mathur & Vignesh Dhatchanamoorthy, Unmasking AsyncRAT New Infection Chain, McAfee Labs, November 3, 2023.

[DCRat]

11. The BlackBerry Research and Intelligence Team, How DCRat (AKA Dark Crystal) Works, Blackberry, May 9, 2022.
12. Muhammad Hasan Ali, A deep dive into DCRAT/DarkCrystalRAT malware. August 30, 2023.

[Formbook]

13. Xiaopeng Zhang, Deep Analysis: New FormBook Variant Delivered in Phishing Campaign – Part I ,Fortiguard Labs, April 12, 2021.

[NjRAT]

14. Aaron Stratton, njRAT Malware Analysis. Introduction I by Aaron Stratton, InfoSec Write-ups, October 7, 2022.

15. Splunk Research Team, More Than Just a RAT: Unveiling NjRAT's MBR Wiping Capabilities, Splunk, October 31, 2023.

[Raccoon]

16. Pierre Le Bourhis, Quentin Bourgue and Sekoia TDR, Raccoon Stealer v2 - Part 2: In-depth analysis, Sekoia.io Blog, June 29 2022.

17. Noel Anthony Llimos, Raccoon Stealer Announce Return After Hiatus, Cyberint, August 15, 2023.

[RedLIne]

18. A Detailed Analysis of the RedLine Stealer, Security Scorecard, April 1, 2022.

19. Monika Grigutytė, What is RedLine Stealer, and how does it work?, NortVPN, July 13, 2023.

20. New RedLine Stealer Variant Disguised as Game Cheats Using Lua Bytecode for Stealth, The Hacker News, April 21, 2024.

[Snake]

21. Evident CERT Team, Snake Malware - Taken down by the FBI after 20 years of existence, Atos, March 17, 2023.

22. Cybersecurity Advisory, Hunting Russian Intelligence "Snake" Malware, CISA, May 9, 2023.

23. Cybereason Global SOC Team, THREAT ANALYSIS REPORT: Snake Infostealer Malware, Cyberreason, Publish date unknown, Last accessed July 15, 2024

[Vidar]

24. Vidar - Information-Stealing Malware Malvertises on Google, Darktrace, January 30, 2023.

25. GridinSoft, Vidar Stealer Malware Detailed Analysis 2024, Gridinsoft, February 15, 2024.

26. Vidar Stealer: An In-depth Analysis of an Information-Stealing Malware - CYFIRMA, CYFIRMA, June 3, 2024.

Appendix A: Maliciousness Analysis

The following presents the verbatim output of ChatGPT in response to the prompt presented earlier to analyze the 21 unique functions of MALWARE DLL.

Verdict: Malicious

Summary of Activities

1. Class Initialization and Component Binding

- The Yqnpressure.Ujbexpert class constructor initializes window components and binds certain commands and mouse events.
- Functions dppproducer, akucotton, and dmcmargin are called, indicating various initialization and event handling routines.

2. Command and Event Handling

- The akucotton function handles execution of commands, particularly related to undo operations.
- The dmcmargin function deals with mouse button events, specifically checking for left mouse button clicks and storing state accordingly.

3. State Management and Undo Mechanism

- The Dppproducer function initializes certain visibility properties and loads application components, suggesting manipulation of UI elements.
- The Dwgsurely function sets up the sgranimal field, indicating interaction with a canvas or drawing component.
- The Epfgreatest.UndoRedo namespace contains classes like Mwztarget responsible for managing undo/redo operations, storing and restoring states.

4. Resource Management

- The Cllindeed.Lqphall functions manage resource localization, handling resource manager and culture settings.

5. Obfuscation and Decryption

- The Asvbeer.Isveventually::Plnarrive function suggests obfuscation techniques, involving base64 encoding and XOR operations to manipulate byte arrays, potentially indicating payload decryption.

6. Dynamic Assembly Loading

- The Epfgreatest.Ztdvoter::Main function dynamically loads assemblies and executes methods, which is a common technique used in malware to execute payloads at runtime.

7. Late Binding and Reflection

- Functions make extensive use of late binding and reflection (microsoftvisualbasiccompilerservicesnewlatebinding::latecall, systemreflectionassembly::load), which can be used to dynamically invoke methods and load assemblies, often seen in obfuscated malware.

Indicators of Compromise (IOCs)

1. URLs:

- No explicit URLs were found in the decompiled code.

2. Created Files:

- The code does not explicitly mention file creation, but dynamic assembly loading implies that temporary files or in-memory assemblies may be involved.

3. Mutex:

- No mutex creation detected, but further dynamic analysis might reveal synchronization mechanisms.

4. Network Activity:

- The code does not explicitly mention network activities, but dynamic assembly loading and obfuscation suggest potential communication with external servers.

5. Other Artifacts:

- Extensive use of obfuscation techniques (base64 encoding, XOR operations).
- Dynamic assembly loading and method invocation.

Appendix B: MITRE ATT&CK Framework

The following presents the verbatim response of ChatGPT when asked to map the malicious behavior to MITRE ATT&CK TTPs.

Based on the detailed analysis of the dec ompiled code, several MITRE ATT&CK tactics, techniques, and procedures (T TPs) can be associated with the observed malicious behaviors. Here is a mapping t o relevant MITRE ATT&CK T TPs:

1. Dynamic Component Loading	**T1127 -Trusted Developer Utilities Proxy Execution:** The code dynamically loads assemblies and executes methods, a common technique to execute payloads at runtime without directly embedding them in the initial code.
2. Obfuscation	**T1127 -Trusted Developer Utilities Proxy Execution:** The code employs base64 encoding and XOR operations to obfuscate payloads, making it difficult for static analysis tools to detect malicious content.
3. Command and Scripting Interpreter	**T1059 - Command and Scripting Interpreter:** The use of reflection and late binding (microsoftvisualbasiccompilerservicesnewlatebinding::latecall, systemreflectionassembly::load) suggests manipulation of code and command execution at runtime.
4. Process Injection	**T1055 - Process Injection** Dynamic assembly loading and manipulation may involve injecting malicious code into the memory space of legitimate processes.
5. Inhibit System Recovery	**T1490 - Inhibit System Recovery** The Epfgreatest.UndoRedo namespace manages undo/redo operations, which might be used to tamper with system states and inhibit recovery processes.

6. File and Directory Discovery	**T1083 - File and Directory Discovery:** While not explicitly mentioned, the initialization and interaction with various UI components suggest potential enumeration and manipulation of files and directories.
7. Data Encoding	**T1132 - Data Encoding:** The use of base64 encoding to encode payloads or other data for transmission.
8. Software Packing	**T1045 - Software Packing:** Obfuscation techniques including encoding and runtime decryption are indicative of packing strategies to evade detection.
9. System Information Discovery	**T1059 - Command and Scripting Interpreter:** Interaction with system resources and dynamic component loading suggests potential gathering of system information to tailor further actions.

Appendix C: Trojan Versions of Legitimate DLLs

Table 4
Malicious .NET DLL containing malicious code found in MALWARE DLL

Nr	SHA256	Family
01	0703f29cc8457988de8ab0e0a8b187521fce8812bbc3bb216e29f8907ffe4c66	AgentTesla
02	126345195badc693f032508364890a81f914d05b08f925ca4b48161e2ddf2ef8	Njrat
03	1568cf8a9d6742a6d840c479a58d9d2b0613394d8065b1a2d2e934706a43df3d	Vidar
04	15bad895c6afb47d3dbf662a5743d49ce0bba45b110b494645d92b2db423ac4c (MALWARE DLL)	Snake
05	1e5f4297bced52e5ea9c305b82f7aa6d2305930fb93c3018f6e4d6d6a2927764	DCRat
06	2ffaee43775fc2cd680d702a6076ac1d8bb8bf17eab0305a4290cb2ac66a865b	Vidar
07	315ea8af1371ec210937876305db47f1116a5e75b8a74ffab043267d55c9a46b	AsyncRat
08	32ca6952f2d306c5229317314380f71455d772ec84cfff2feca9d74a2364d036	Vidar
09	34c16889e439ea444e969c6bc6fef3296dff405444f130ed9f3a82f47f6f1575	AgentTesla
10	39c887e8c10540bd9ac9145877cd3c5e86040b75fbdda1e65204bd67ad531db2	Formbook
11	6569e6fd303dc3e667758a528e55d94fa7548687efde5deeda81887b43b6be8d	Formbook
12	9262f758cc7ded37f9e1362d7e07a2a2603790da6ee9ce22366e105e2d9591d9	Formbook
13	9bd924c7e94a5e95dc52fe98173b92816ccd1854a9612a9896f0d1f4b4ac53ae	Formbook
14	aef3536e125a558e68c2add19613754b67df5d965fe630e645ee16f90e9ca12a	RedLine
15	b6c377b6acf9d822885bd750074103a2468bd47ce77d1e72303f053dba49f743	Formbook
16	c2aa3b42821e22b2ad2c61981ffcbfbc9c24c2bc9c61899c6337155f3688d79a	Formbook
17	c8c56bc8efd9d68a8a854f608c5c1a90369f92dfc6f2cda2bb5f6a27ad2f6710	Snake
18	cb1b1d4e4f463115da0a4934d1d38d451a2a90c3fc83a4657b2efd4feb5f66b0	AgentTesla
19	cf6c3743ab5657d587fbd8e98b804b11e749cd11e9f0e4ec9644104b240437aa	Formbook
20	d0c16fc8e7e0701da3fcd850c6834f772dd9d6daab6a0ea5f507baba97d39913	AgentTesla
21	d407dea31b4e55d9955bdea84e990205f7bbea67fd39e82bb61f942dc20e9b54	Raccoon
22	dc6a963a037578ef826cb4bc2cac32781f2ea15eb8126e40fd809b2a3d7a33cd	Formbook
23	df809087a49af7316955ba3bc0dc35d086529be98bd82df7ab4ccfa22c38e97d	AgentTesla
24	ec9a5f5033e8421fe260cb9ad30f2cb4d83ff5bb09a9a45593f2267d9bb495eb	AgentTesla
25	ef6ff8755172436f00bece971170bd7e680ab77e88e510078d8cd0744d4d2d3e	Vidar
26	f6d87ecd12a6724e65e0d2be0f8d787b5184c16ba178e9c14913b23ea226300c	Vidar

Nr	Filename	VT Detection Ratio
01	Microsoft.IdentityModel.Clients.ActiveDirectory.dll	37/67
02	Microsoft.IdentityModel.Clients.ActiveDirectory.dll	44/68
03	Rebex.Net.Ssh.dll	27/67
04	Newtonsoft.Json.dll	51/74
05	Microsoft.VisualStudio.Services.CodeReview.Discussion.WebApi.dll	53/71
06	Microsoft.VisualStudio.DiffBase.Controls.dll	42/66
07	Microsoft.VisualStudio.TestPlatform.Client.dll	23/67
08	No name	42/68
09	Microsoft.VisualStudio.WindowsAzure.CommonAzureTools.Contracts.Internal.dll	22/66
10	Microsoft.WebTools.Languages.Html.dll	32/66
11	Microsoft.Ingestion.Shared.WebApi.Client.dll	29/52
12	Rebex.Net.SecureSocket.dll	41/68
13	Microsoft.PythonTools.IronPython.Interpreter.dll	45/70
14	SQLitePCLRaw.core.dll	35/68
15	Microsoft.DiaSymReader.Converter.dll	41/70
16	NuGet.Frameworks.dll	21/66
17	Microsoft.WindowsAzure.Storage.dll	25/67
18	Microsoft.Diagnostics.MemoryGraph.dll	33/67
19	Microsoft.AspNet.Scaffolding.12.0.dll	43/71
20	Microsoft.VisualStudio.Workspace.Extensions.dll	27/66
21	Microsoft.VisualStudio.Utilities.Internal.dll	35/66
22	Microsoft.VisualStudio.TeamFoundation.Lab.dll	33/66
23	SQLitePCLRaw.core.dll	15/67
24	Microsoft.TeamFoundation.Core.WebApi.dll	36/67
25	Microsoft.VisualStudio.Services.Search.Shared.WebApi.dll	40/68
26	ServiceHub.Host.CLR.exe	39/68

Authors

Dr. Arun Lakhotia

@DrArunL

CTO, Unknown Cyber Inc

Professor of Computer Science,
University of Louisiana at Lafayette

Bob Miller

@RobertDMiller

COO, Global Data Systems

About Unknown Cyber

Unknown Cyber is an In-Q-Tel Portfolio Company that is founded on technology developed at the University of Louisiana at Lafayette under the DARPA Cyber Genome Project to detect, hunt, and attribute unknown malware through their code. Our many unique innovations automate the capability to track the evolution of code through generations of malicious and benign programs, identify novel variants of malware rapidly and at scale, and automatically create very precise Yara rule from the bytecode of functions unique to a malware. The technology has been validated by very experienced professionals in MSSPs, Enterprise SOCs, and intelligence agencies.

About Global Data Systems

Global Data Systems is a leading full-service Managed Services Provider (MSP)/Managed Security Services Provider (MSSP), offering innovative IT solutions and services grounded in the principles of "Connect. Collaborate. Protect." With 37 years of industry expertise, we have become the trusted partner for midsize and large enterprises, providing a comprehensive suite of IT services that adapt to the dynamic needs of modern businesses. From network management and cloud solutions to cybersecurity and communication technologies, our customer-centric approach tailors services to address the unique challenges of each organization. We excel in enabling robust connections, promoting collaboration through innovative tools, and prioritizing security with cutting-edge cybersecurity measures. Our dedicated team of experts puts their knowledge to work for our clients, assisting them in solving their IT challenges. This commitment to excellence propels Global Data Systems to higher levels of success and client satisfaction each year, ensuring impeccable service delivery from solution design to simplified billing, all with 24x7x365 support.

For more information

Phone: 1-888-678-6992

Email: info@UnknownCyber.com

Website: www.UnknownCyber.com