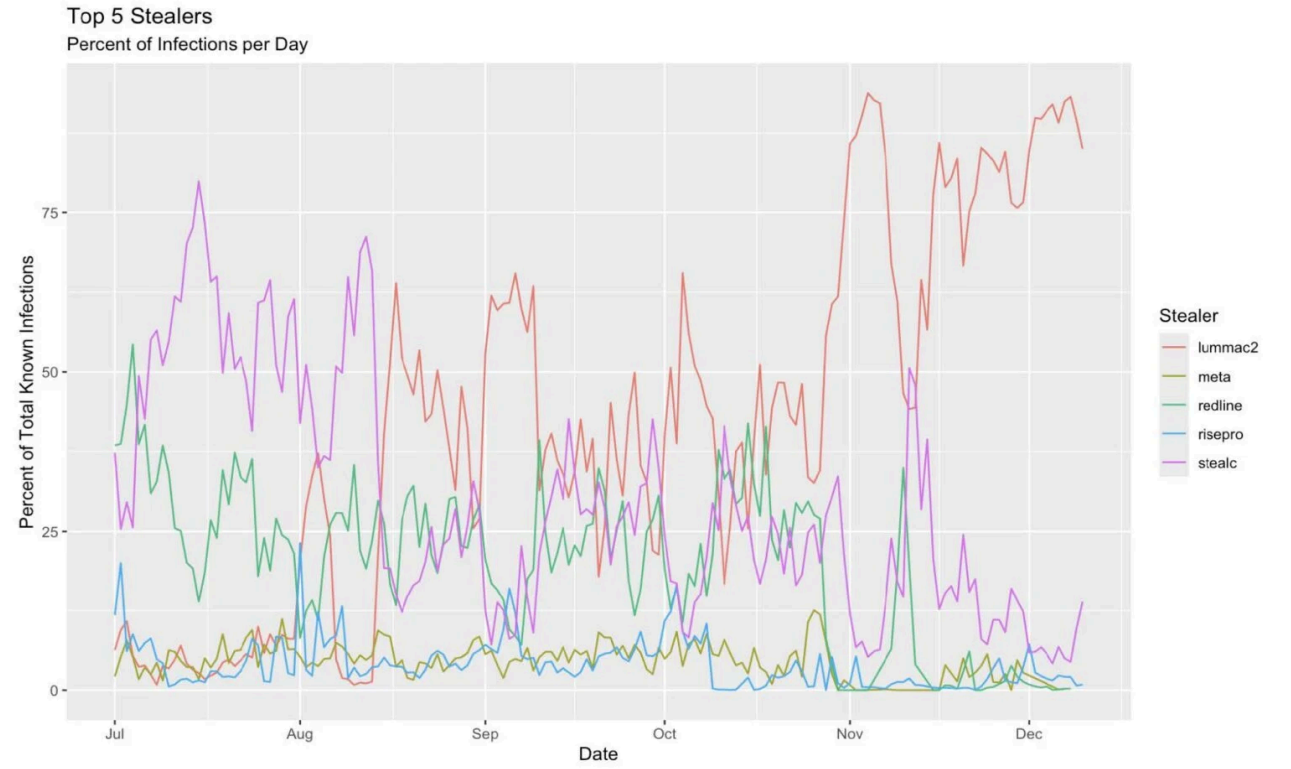
LummaC2 Revisited: What's Making this Stealer Stealthier and More Lethal

By James

Published: 2024-12-19 · Archived: 2026-04-05 22:00:02 UTC

It's been about a year since our last analysis of LummaC2, and SpyCloud analysts have been hard at work tracking the changes to LummaC2 infostealer malware that have occurred since then – and there have been many, including changes to its:



A graph comparing LummaC2 infections to other prevalent malware family infections. Around November 2024, Lumma infections skyrocket.

Here's what we found when we looked back under the hood.

Updates to LummaC2's theft capabilities

Since our last blog, LummaC2 has undergone several changes that upgrade its stealing capabilities. These changes include:

In addition to these specific changes, LummaC2's theft operation has also evolved some of its functionalities. **Instead of stealing information all at once, assembling it, and then exfiltrating it, LummaC2 now assembles**

and exfiltrates each newly obtained bit of information before a new function is executed.

This allows LummaC2 to be more resilient, and if it gets detected or something goes wrong during operation, it's possible that it will still be able to exfiltrate partial logs to the command and control server (C2) that threat actors can leverage.

New changes to LummaC2's browser theft techniques

In late July 2024, Google released an update to Chrome that introduced "App Bound Encryption," or ABE, a feature designed to limit illicit access to credentials like cookies. This feature now encrypts the cookies and stores them behind a device-specific ABE key, which is much more challenging for stealers to access.

LummaC2 has developed a bypass for this technique which scrapes Chromium process' internal memory for "chrome.dll" and finds the address to Chrome's CookieMonster library, used for manipulating cookies. Using this address, LummaC2 then interacts with Chrome's CookieMonster library to dump the cookies to a text file, which is then exfiltrated to the C2.

This process can be viewed in depth here, however, it should be noted that the obfuscated pattern string used by LummaC2 for matching is at the time of this writing:

- 9sdmLrTRuOE8????p4UMZQLB????jl7CKwIeGWvwDe3YvXN40wd763ssw7Cx????kdamAY3?PdE????6J?????7Qy6S04NP0R????k70a?oAj7a3????????K3smA????maSd?3l4

The string can be seen in use by the malware in **Image 1** below.

Image 1: Displays LummaC2 passing the obfuscated pattern string to the Chrome DLL memory searcher.

Additionally, LummaC2 now steals the victim's os_crypt.encrypted_key field, which can be used for further credential decryption. This key is stored in "dp.txt" in the browser exfil folder.

LummaC2's new approach to extension theft

In order to make extension theft more resilient and modular, the developers of LummaC2 have added a few JSON dictionary keys to their extension dictionary options, namely "ldb" and "ses", as observed in **Image 2**.

These two keys, set to a boolean value (true/false), indicate if there are additional behaviors that need to be performed or files that need to be stolen in order to properly steal the extension. For example, the "ldb" key indicates the presence of a .LDB file for LummaC2 to steal, which normally contains incredibly valuable information for LummaC2 such as recent transactions and wallet information. The "ses" key indicates that LummaC2 should additionally attempt to steal files from Chrome's "Sync Extension Settings" folder, which is a feature used by Chrome to allow users to share extensions across multiple devices.

Additionally, LummaC2 has implemented the ability to steal from Firefox extensions, which opens the door to a whole new avenue of extension theft through extensions that may be in use for Firefox but not Chrome.

Image 2: Displays the two updated entries for LummaC2's extension theft routines.

LummaC2's additional stealer capabilities

While LummaC2 has a dynamic config that it pulls down from the C2 infrastructure that instructs the bot on what to steal, it also has a few hardcoded theft functions. Some of these functions are fairly new, such as those for Discord, Steam, and Notepad++. Additionally, it has a new hardcoded C2 fallback functionality which is very unique in operation.

Discord user token theft

Among the newer hardcoded theft capabilities of LummaC2 is its ability to steal Discord user tokens. These tokens, which are normally base64 encoded, allow users and bots to log in and authenticate with servers. This capability allows LummaC2 customers to easily take over control of Discord accounts.

Steam profile information theft

Another one of LummaC2's new hardcoded theft capabilities is its ability to steal Steam profile information. This theft occurs in two parts, stealing from the Steam process memory as well as stealing Steam config files that allow for easy account takeovers.

Notepad++ text file theft

In recent versions of LummaC2, LummaC2's file theft routine focuses on stealing text files stored on the Desktop and in similar locations, and LummaC2's newest hardcoded theft capability is an upgrade of this routine. LummaC2 can now find Notepad++ session.xml files, which are created when a Notepad++ session is closed unexpectedly (or Notepad++ is open), scrape the files, and then extract the "filename=" field to find additional text files to steal and exfiltrate.

C2 fallback

In recent samples of LummaC2, we observed a C2 fallback routine, which connects out to a hardcoded URL to obtain an obfuscated C2. It then deobfuscates the C2 to reveal an additional C2 not included in its hardcoded config. LummaC2 only uses this routine if the C2s contained in LummaC2's hardcoded config are not responsive.

While having a fallback routine is not unique in itself, what *is* unique is how LummaC2 goes about it, leveraging Steam accounts that are named the URL, and ROT +11 caesar ciphers to obfuscate, as observed in **Image 3**:

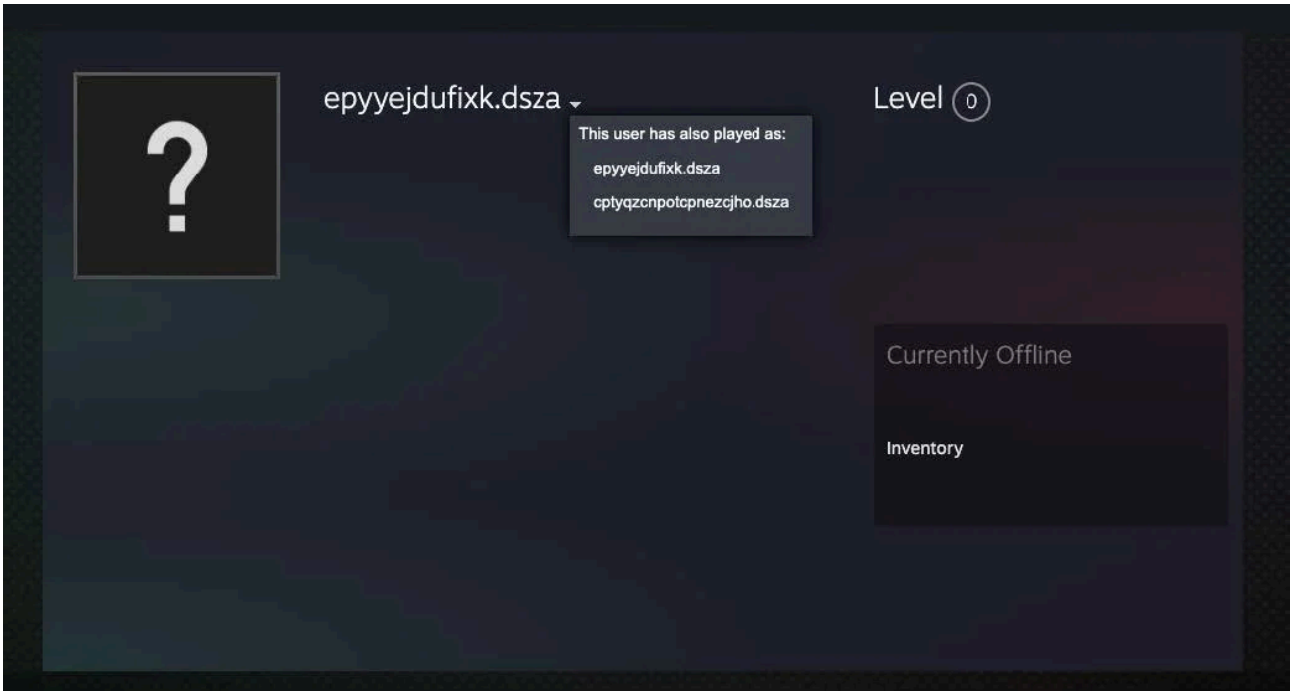


Image 3: One of LummaC2's fallback Steam profiles, along with the history used for the account.

In addition to the domain shown in Image 3, two of the fallback domains previously used by LummaC2 for C2 operations are:

- tenntysjuxmz[.]shop
- reinforcedirectorywd[.]shop

For a LummaC2 infection, stolen credentials are not the only way for criminals to monetize access. In fact, as observed in **Image 4**, using a collaboration with GhostSocks, LummaC2 now allows actors to infect victims with reverse proxy binaries to turn their victims into residential proxies.

Image 4: LummaC2's instructions for how to use the GhostSocks proxy plugin.

Using this feature, actors are able to easily leverage LummaC2's "Google Expired Token Refresh" feature in conjunction with the residential proxies to refresh expired Google tokens, even when a victim has changed their password. This is particularly concerning because other account protections that depend on same-device fingerprinting become trivial to bypass when traffic originates from the device that is fingerprinted.

Additionally, actors that work with ransomware teams (either directly or tangentially) can use these residential proxies to sell direct access to juicy victims to ransomware brokers. This feature gives actors an additional monetization route that would've been much harder to establish without. Ransomware brokers can then sell this access to teams that distribute ransomware, which results in easy access to environments to deploy ransomware.

Figure 1: Flowchart showing how a threat actor can use LummaC2's GhostSocks feature to turn a victim into a residential proxy for further illicit activity.

We've observed another change to LummaC2's use of a technique known as "dynamic import hashing" in order to obfuscate and hide its use of some Windows API calls. This technique of hashing the names of imports in order to

hide functionality from analysts/AV is pretty standard for malware and LummaC2 is no different.

In past versions of Lumma, the hashing algorithm used for this purpose was murmurhash32, an established hashing algorithm. In current versions of Lumma, LummaC2 has shifted to using FNV1A with a standard prime and modified offset basis, as observed in **Image 5**.

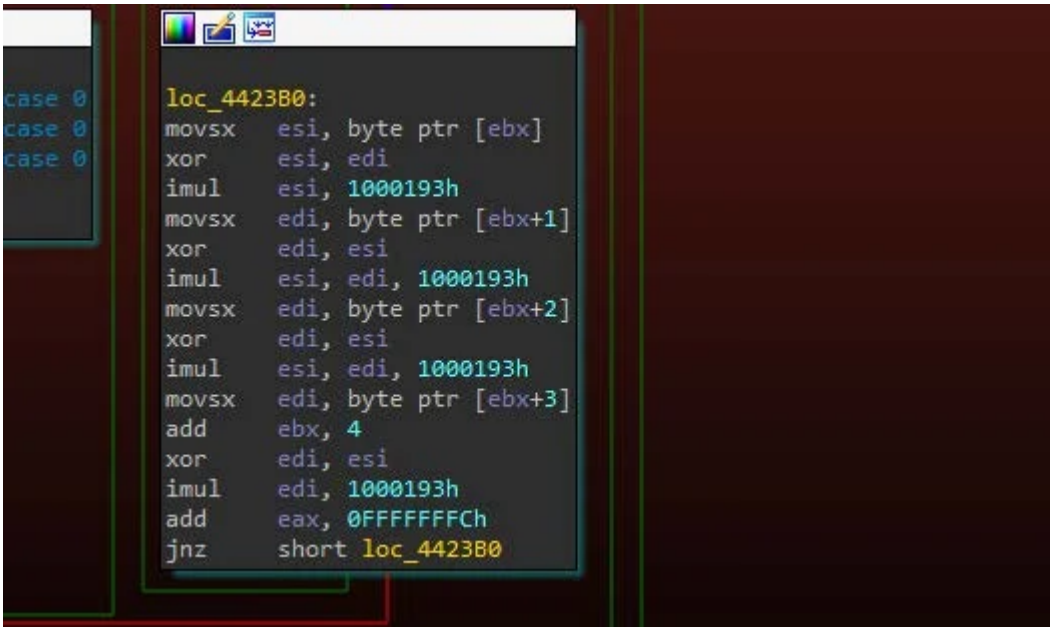


Image 5: LummaC2's implementation of FNV1A using the 0x1000193 Prime.

The offset basis used for LummaC2's dynamic import hashing implementation changes frequently, allowing LummaC2 to better hide from defenders and detection software, as function hashes are no longer detectable.

LummaC2's devs sell access to the malware on a tier-based system, including Corporate, Professional, and Enterprise tiers. As observed in **Image 6**, LummaC2 devs previously advertised "Heaven's Gate" functionality included in the builds for its corporate tier only, allowing it to execute functions in 64-bit memory space from a 32-bit application. This functionality leverages a known 64-bit handler embedded in 32-bit applications for compatibility purposes.

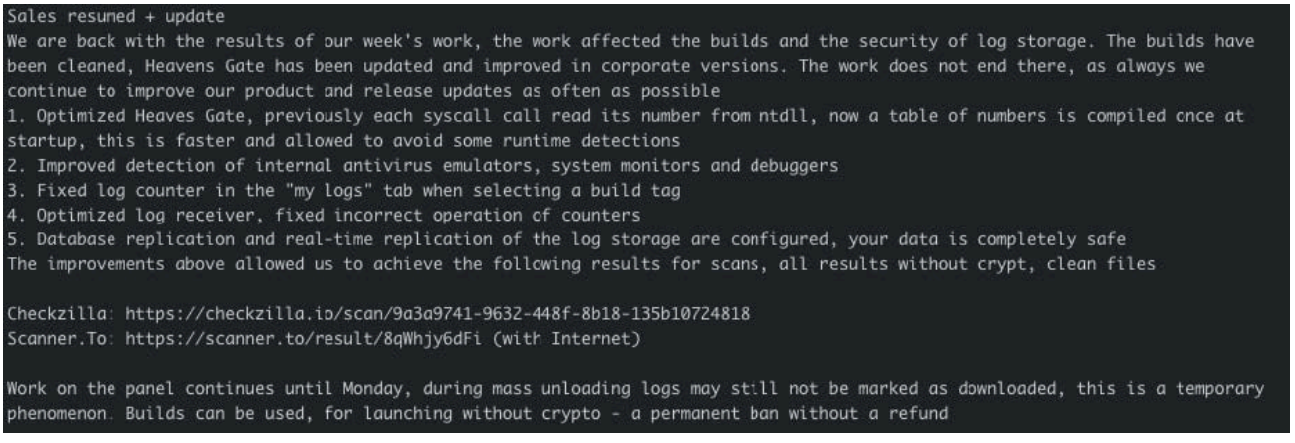


Image 6: One of LummaC2's older ads that mentions Heaven's Gate, from April 2023.

The technique, known as “Heaven’s Gate”, allows LummaC2 to better evade sandboxes and analyses, as the actual important function calls are heavily obfuscated and proxied into 64 bit memory space.

In recent builds of LummaC2, however, “Heaven’s Gate” has been included in the lower tiers, allowing all tiers of LummaC2 to access and leverage this technique.

In current builds, as observed in **Image 7**, LummaC2 first prepares a list of FNV1A hashed functions, as well as their corresponding ordinals in the libraries they exist in.

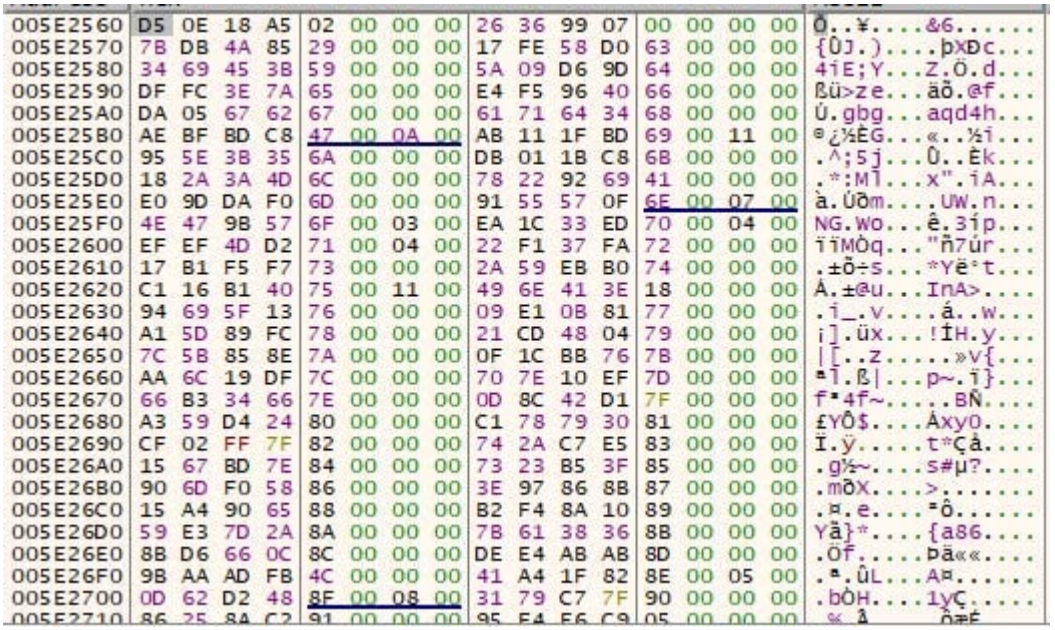


Image 7: The buffer of hashed functions and their corresponding ordinals.

When LummaC2 wants to call specific functions, it locates the hash in the list and then passes the ordinal to the Heaven’s Gate proxy function that allows access to 64-bit memory space, as observed in **Image 8**.

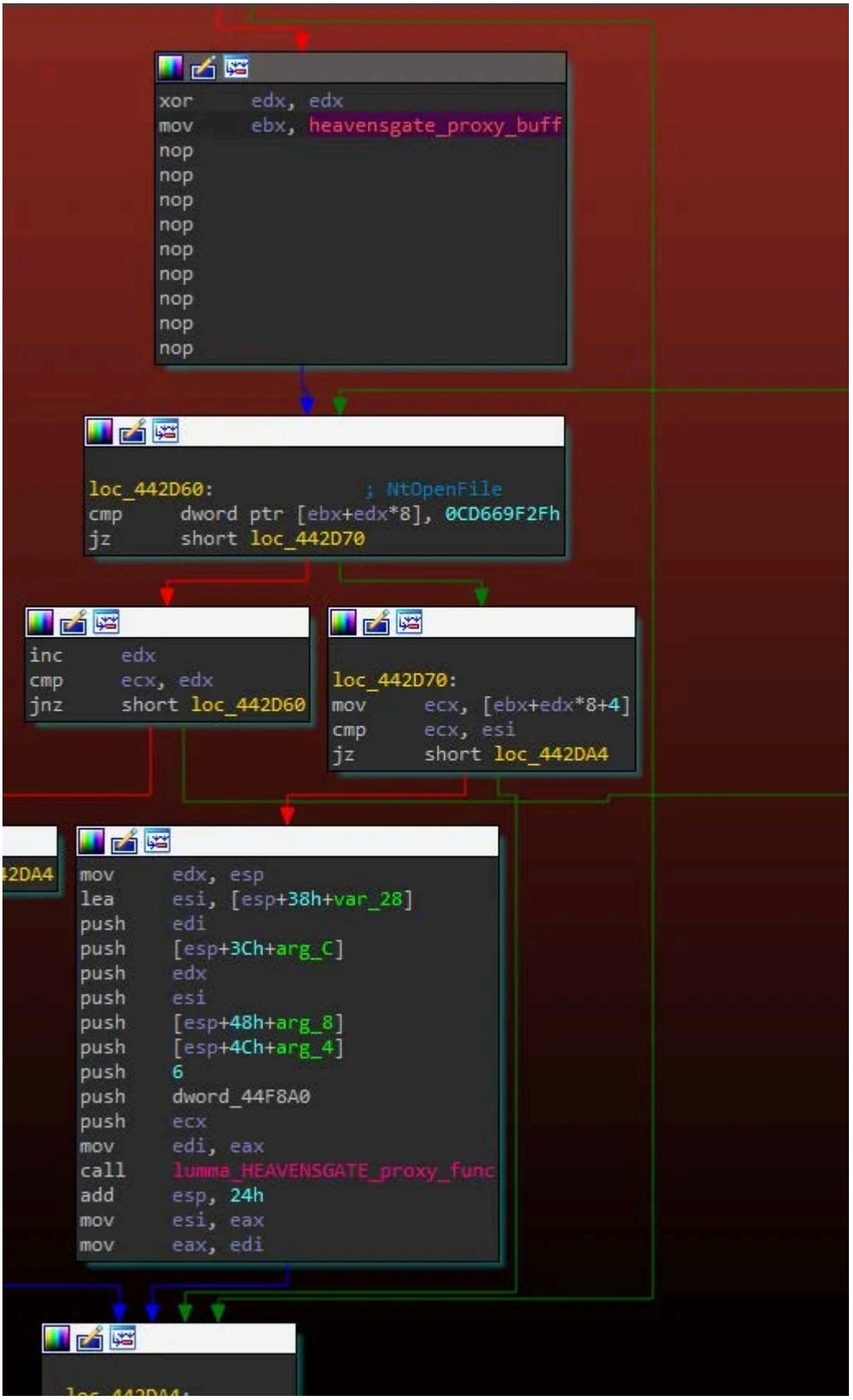


Image 8: LummaC2 locating the hash of NtOpenFile and then executing it with the Heaven's Gate proxy function.

LummaC2's 64-bit injected code can be observed in **Image 9** below:

Image 9: The disassembly of LummaC2's Heaven's Gate injection.

This change allows LummaC2 to more easily evade runtime detections that would otherwise detect the function calls, as many sandboxes do not hook into 64-bit memory space for 32-bit processes.

Last but not least in the long list of changes, LummaC2's recent versions have released a code flattener that makes static analysis challenging.

Code flattening, also known as control flow flattening, is an obfuscation technique that tries to obfuscate a control flow by "flattening" it – essentially putting all functions, jumps, conditional loops, and all other code branches into one big loop with various switch case statements or "jumps" to handle the flow.

Normally, these jumps are very clearly defined. LummaC2, however, first flattens its code before splitting it into chunks separated by jumps, and then also hides the next chunk of code using obfuscated addresses, which are then deobfuscated to reveal the next code chunk using a math algorithm. This offset-calculating algorithm can be observed in **Image 10**. This makes it challenging for static analysis, but trivial for dynamic runtime.

Image 10: One of LummaC2's obfuscated code chunk footers.

As observed in Image 10, the steps to calculate a new code block offset are as follows:

This particular method of moving through code is very effective at breaking "graph views" for disassemblers like IDA, which makes analysis challenging but not impossible – as demonstrated in this blog.

LummaC2's developers have been busy this past year. There have been many updates to the malware, and while many of the changes aren't surprising given adapting detection abilities, several are pretty novel – and certainly concerning for defenders who should be aware of these evolved capabilities.

The new browser, extension, and third-party data theft mean **LummaC2 is capturing more victim data than ever** – all of which can be used against individuals and businesses in identity-based attacks.

The modified exfiltration routine enables data theft **even if the malware is stopped mid-execution**.

The evolved hashing, function, and code flattening features make LummaC2 **tougher for defenders to spot and analyze**.

The capability that has us most concerned – and shouldn't go unnoticed – is the LummaC2 and GhostSocks collaboration that **transforms a victim machine into a residential proxy** after infection, giving bad actors an easy button for further illicit activity. Security teams *have* to monitor for and be able to identify employees, customers, vendors, and contractors infected with infostealers like LummaC2 and fully remediate infections to avoid threats from escalating to attacks like ransomware.

As always, we'll continue monitoring developments of Lumma's capabilities to better understand exfiltration trends and will share updates to our research when available.

The SpyCloud Post-Infection Remediation Guide offers an in-depth look at a critical addition to malware infection response.

Source: https://spycloud.com/blog/lummac2-malware-stealthier-capabilities/