vb 2025 BERLIN

24 - 26 September, 2025 / Berlin, Germany

ARACHNID ALERT: LATRODECTUS LOADER CRAWLS THROUGH DEFENCES

Albert Zsigovits

VMRay, Hungary

azsigovits@vmray.com

www.virusbulletin.com

ABSTRACT

Meet the cunning Latrodectus loader, which first emerged in 2023 and has become a 'go-to' tool for cybercriminals in the past year. Mainly functioning as a downloader, it employs advanced anti-analysis, evasion techniques, and encryption schemes, which all have hardened the loader to evade traditional malware detection.

Since its inception, the loader has undergone rapid development, evolving from version v1.1 to its current iteration, version v2.1, demonstrating the malware authors' commitment with its constant stream of improvements. With strong ties to the now-defunct IcedID loader, Latrodectus is gradually filling the void left by its predecessor's take-down due to Operation Endgame, carried out by Europol (EC3). This paper explores the inner workings of Latrodectus, analysing its anti-analysis features, decoding its malware configuration settings, providing guidance on its mitigation, and conclusions drawn from its usage of BruteRatel C2, the controversial red-team tool, deployed in recent Latrodectus delivery chains.

The loader also employs many sandbox evasion techniques, some of which were most likely designed to evade traditional sandbox solutions. One gripping functionality is its self-deletion mechanism, adapted from a public proof-of-concept *GitHub* repository, showcasing the malware authors' ability to repurpose open-source tools, which is becoming more of a problem due to its simple adaptability to meet malicious ends. As each Latrodectus campaign is denoted by a distinct group ID, we believe the loader may be gravitating towards a malware-as-a-service (MaaS) model.

By examining its rapid development cycle and potential trajectory during 2024 and 2025, we offer insights into its growing popularity as a preferred downloader in the cybercrime ecosystem.

OVERVIEW

First identified in October 2023, Latrodectus malware has since evolved significantly, becoming a key player in the cybercriminal ecosystem [1]. The malware works mainly as a loader/downloader. Latrodectus malware has strong ties with the former, infamous IcedID loader [2], which was taken down in May 2024, thanks to the efforts of Operation Endgame, an international operation led by Europol and EC3 [3]. Since Operation Endgame, IcedID has gone under and Latrodectus can be seen slowly taking its place in the cybercriminal ecosystem. Interestingly, Latrodectus includes a specific C2 command that can download a sample of IcedID loader [4].

Recently, the developers of Latrodectus have released multiple new versions in quick succession [5], likely as a means to stay ahead in the constant 'cat-and-mouse' game between defenders and attackers. As a result of this rapid iteration, these new versions consist primarily of small changes, including the removal of existing features. The previous pace of development would suggest that Latrodectus malware will continue iterating with new versions.

HOW LATRODECTUS MALWARE IS DISTRIBUTED AND CONTINUALLY EVOLVING

Latrodectus malware is distributed in a chain of JavaScript → MSI droppers, finally ending in the core DLL payload. The DLL payload often has four unique-looking exports, using the same address, and eventually running the same main logic when all four exports are tested.

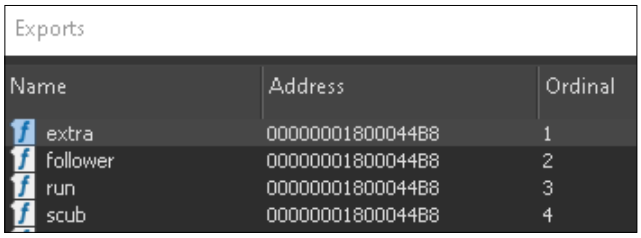

Figure 1: Latrodectus exhibiting four exports with the same export address.

Over time, the loader has undergone several iterations. At the time of writing, the most up-to-date version is v2.1. Initially, early versions began to surface in late September 2023, while samples of the most recent version were compiled at the end of May 2025.

We have tracked each version's earliest PE compiled time to give a rough estimation of its timeline of creation:

Versions	Compiled time for first samples of each version
v1.1a	29 Sep 2023 13:29:13 UTC
v1.1b	15 Feb 2024 10:10:37 UTC
v1.2	21 Mar 2024 16:27:39 UTC
v1.3	09 May 2024 11:08:17 UTC
v1.4	29 Jul 2024 10:07:54 UTC

Versions	Compiled time for first samples of each version
v1.5	30 Jul 2024 17:16:02 UTC
v1.7	16 Sep 2024 08:44:51 UTC
v1.8	25 Sep 2024 11:20:43 UTC

Key technical changes in Latrodectus malware versions include:

- Initially, the family used a PRNG seed with XOR algorithm for string decryption (v1.1a).
- Then Latrodectus developers decided to degrade it, and use a simpler rolling XOR method (v1.1b).
- Starting with version 1.4, the loader changed to AES-256 (CTR) string decryption with a fixed key and a variable initialization value (IV) for each string.
- Additional command IDs were introduced for the command handler in v1.4, such as the possibility of downloading an arbitrary file to %APPDATA%.
- Some features that were previously incorporated have been removed from recent versions of samples, such as the ADS self-deletion technique.

EVASION TECHNIQUES

Latrodectus employs four distinct anti-debugging and sandbox evasion techniques, which are outlined in the following sections.

Process count check

The sanity process count check is most likely aimed at evading sandboxes. Virtualized environments often lack the number of installed and running applications that would be found in a real desktop environment.

Latrodectus simply enumerates the *Windows* OS version via the API call `RtlGetVersion`, or via `GetVersionExW` if the Rtl version does not return data. If the routine detects *Windows 10* or *Windows 11* as the host OS, Latrodectus requires at least 75 active processes to launch, otherwise it simply terminates. The other condition does the same check for *Windows* versions v6.3 or lower (which would constitute *Windows 8.1*, *Windows 8*, *Windows 7* and anything below). In this case, the loader needs there to be at least 50 active processes to launch. This is to account for baseline levels for different versions of *Windows* OS.

```
12  if ( latro_addr_RtlGetVersion )
13    latro_addr_RtlGetVersion(&buf);
14  if ( !latro_addr_RtlGetVersion )
15    latro_addr_GetVersionExW(&buf);
16  if ( majorVerNum != 5 || minorVerNum )
17  {
18    if ( majorVerNum == 5 && minorVerNum )
19    {
20      return 1;
21    }
22    else if ( majorVerNum != 6 || minorVerNum )
23    {
24      if ( majorVerNum == 6 && minorVerNum == 1 )
25      {
26        return 3;
27      }
28      else if ( majorVerNum == 6 && minorVerNum == 2 )
29      {
30        return 4;
31      }
32      else if ( majorVerNum == 6 && minorVerNum == 3 )
33      {
34        return 5;
35      }
36      else if ( majorVerNum != 10 || minorVerNum )
37      {
38        if ( majorVerNum == 10 && !minorVerNum && v5 >= 0x55F0 )
39          return 7;
```

Figure 2: Latrodectus enumerating Windows OS version.

MAC address validity

The second evasion technique enumerates the `_IP_ADAPTER_INFO` structure via the `GetAdaptersInfo` API function, following which all hardware addresses of present network adapters are examined against the argument of 6. In the event an address does not equal six bytes, the program will simply terminate. While MAC addresses have been standardized to

six bytes for a long time now, some older networking technologies used different address lengths and certain specialized or proprietary systems might use non-standard MAC address formats. This same evasion check was also present in the BumbleBee loader.

Figure 3: A rare network card check to verify validity of MAC addresses.

BeingDebugged

The third evasion check is simply walking the Process Environment Block (PEB) data structure to query the `BeingDebugged` flag to detect any debugging attempts; this is a smarter way than calling the actual *Windows* API `IsDebuggerPresent()`, which may trigger some AV/EDR systems.

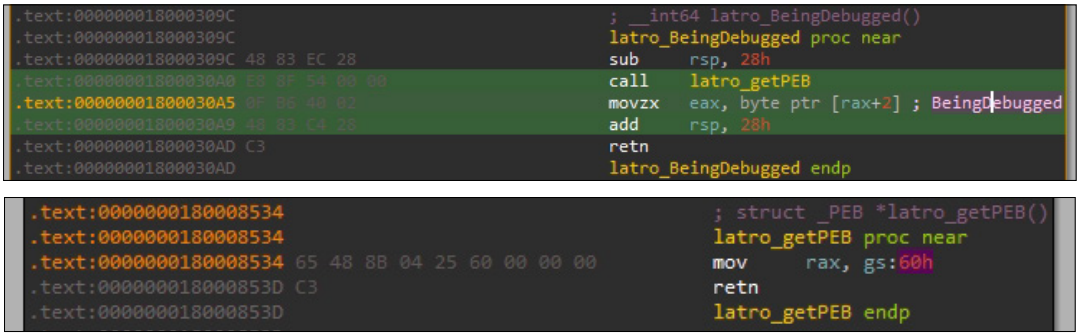

Figure 4: BeingDebugged flag being checked by walking the PEB.

WOW64 process check

The fourth check is a validation of the current process and whether it is running under WOW64 on *Windows*, which simply ascertains whether the malware process is running as a 32-bit process on the 64-bit OS. In this case, the malware will simply exit. Since all Latrodectus DLLs so far have been 64-bit DLLs, it is not fully clear what the intention of the threat actors was with this condition, since it will not return 32-bit in normal circumstances. This might be an attempt to detect certain emulation scenarios.

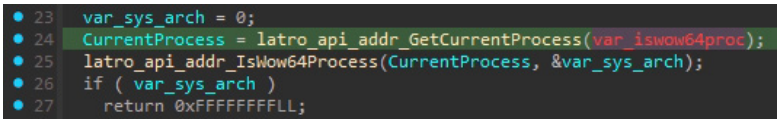

Figure 5: Checking the running process against IsWow64Process.

ENCRYPTED STRINGS

How Latrodectus encrypts strings

In order to make the reverse engineering process harder, Latrodectus employs string encryption. The internal strings hold a significant amount of information on how the malware operates and what behaviour it resembles. These internal strings

often serve as the base for malware configuration extraction as well. In early versions of samples, the malware family utilized a unique pseudo random number generator (PRNG) for seeding. Later, Latrodectus downgraded this functionality and simply opted to use an increment-based seed variable, which in essence turned the encryption process into a rolling XOR method. As of the most recent versions, the loader is now using AES-256 encryption with a hard-coded key inside the sample and with a variable IV for each of encrypted strings.

While the encryption algorithm has undergone several changes, as described earlier, the storage of encrypted strings has remained largely consistent. The prototype for these structures is simple: the encrypted strings are stored in the .data section of the DLL. In early versions of the loader, the first four bytes denote the XOR key and the delimiter bytes as well, the length of each strings are stored in the fifth and sixth bytes, and the remaining bytes are the actual encrypted data.

Storage of encrypted data across versions

In the recent versions, due to the introduction of the AES algorithm, a hard-coded key is burnt into the .text section of the samples. The data length still resides in the .data section in the first two bytes for each chunk, which is followed by the IV, taking up 16 bytes. The remaining data of each chunk is the actual encrypted data.

String encryption						
Version	**Algorithm**	**Key**	**Data length**	**IV**	**Data**	**Seed**
v1.1a	XOR	chunk[:4]	chunk[4:6]	Not applicable	chunk[6:6+data_length]	PRNG
v1.1b	Rolling XOR	chunk[:4]	chunk[4:6]	Not applicable	chunk[6:6+data_length]	Incrementer
v1.2	Rolling XOR	chunk[:4]	chunk[4:6]	Not applicable	chunk[6:6+data_length]	Incrementer
v1.3	Rolling XOR	chunk[:4]	chunk[4:6]	Not applicable	chunk[6:6+data_length]	Incrementer
v1.4	AES-256 (CTR mode)	hard coded	chunk[:2]	chunk[2:18]	chunk[18:18+data_length]	Not applicable
v1.5	AES-256 (CTR mode)	hard coded	chunk[:2]	chunk[2:18]	chunk[18:18+data_length]	Not applicable
v1.7	AES-256 (CTR mode)	hard coded	chunk[:2]	chunk[2:18]	chunk[18:18+data_length]	Not applicable
v1.8	AES-256 (CTR mode)	hard coded	chunk[:2]	chunk[2:18]	chunk[18:18+data_length]	Not applicable

RUNTIME API RESOLVING AND API HASHING

The loader again utilizes the Process Environment Block (PEB) structure to find the base addresses of kernel32.dll and ntdll.dll. Then Latrodectus continues to resolve other libraries, like user32.dll, wininet.dll and iphlpapi.dll by finding the files inside the \Windows\System32 folder, calculating the CRC32 checksums of the filenames, and then comparing them with the hard-coded hashed values in the sample. The last step is to call the `LoadLibraryW` function to finally load the library.

Once Latrodectus has loaded all the necessary DLLs, it continues to resolve the APIs by comparing the CRC32 checksums of the exported functions with the target values. The open-source project hashdb [6] can save work here, as its Lookup Service can reverse the hash values and recreate the API names within an analysis.

```
.text:0000000000408F17 C7 44 24 48 D6 0D 1D 20          mov     [rsp+278h+var_230], 201D0DD6h
.text:0000000000408F1F 48 8D 05 9A 7F 00 00             lea     rax, latro_handle_user32_dll
.text:0000000000408F26 48 89 44 24 50                   mov     [rsp+278h+var_228], rax
.text:0000000000408F2B 48 8D 05 A6 7E 00 00             lea     rax, latro_addr_wsprintfW
.text:0000000000408F32 48 89 44 24 58                   mov     [rsp+278h+var_220], rax
.text:0000000000408F37 C7 44 24 60 87 B8 C9 D4          mov     [rsp+278h+var_218], 0D4C9B887h
.text:0000000000408F3F 48 8D 05 7A 7F 00 00             lea     rax, latro_handle_user32_dll
.text:0000000000408F46 48 89 44 24 68                   mov     [rsp+278h+var_210], rax
.text:0000000000408F4B 48 8D 05 8E 7E 00 00             lea     rax, latro_addr_wsprintfA
.text:0000000000408F52 48 89 44 24 70                   mov     [rsp+278h+var_208], rax
.text:0000000000408F57 C7 44 24 78 6C 1D C2 2E          mov     [rsp+278h+var_200], 2EC21D6Ch
.text:0000000000408F5F 48 8D 05 62 7F 00 00             lea     rax, latro_handle_wininet_dll
.text:0000000000408F66 48 89 84 24 80 00 00 00          mov     [rsp+278h+var_1F8], rax
.text:0000000000408F6E 48 8D 05 7B 7E 00 00             lea     rax, latro_addr_InternetOpenW
.text:0000000000408F75 48 89 84 24 88 00 00 00          mov     [rsp+278h+var_1F0], rax
.text:0000000000408F7D C7 84 24 90 00 00 00 F4 A5 4F C2 mov     [rsp+278h+var_1E8], 0C24FA5F4h
.text:0000000000408F88 48 8D 05 39 7F 00 00             lea     rax, latro_handle_wininet_dll
.text:0000000000408F8F 48 89 84 24 98 00 00 00          mov     [rsp+278h+var_1E0], rax
.text:0000000000408F97 48 8D 05 5A 7E 00 00             lea     rax, latro_addr_InternetConnectA
.text:0000000000408F9E 48 89 84 24 A0 00 00 00          mov     [rsp+278h+var_1D8], rax
```

Figure 6: CRC32-based API hashing in Latrodectus.

SETTING UP PERSISTENCE

File placement in %APPDATA% folder

To ensure persistence, the malware first checks whether it is running from under the %APPDATA% folder. If it is not, it copies itself to one of the following locations:

- `%APPDATA%\Custom_update\Update_XXXXXXXX.dll` (older versions)
- `%APPDATA%\falsify_steward\confrontation_XXXXXXXX.dll` (newer versions)

The part of the filename noted with `XXXXXXXX` gets filled up with the hardware ID, generated from the system's volume serial number and a hard-coded constant (described in the 'Hardware ID' section of this paper).

Leveraging COM interfaces for scheduled tasks

The creativity of the developers is again demonstrated at the next stage of persistence: rather than using common APIs or scheduler commands to create a scheduled task, Latrodectus uses the Component Object Model (COM) interface to stay active. Our function log clearly describes the behaviour and it is easy to follow the chain of events.

First, the sample calls the `CoCreateInstance` API to create and initialize an object, then it connects to the `ITaskService` object. A new task is created inside the root of the scheduler and the job is set to execute whenever the user logs on. The name of the scheduled task changes between 'Updater' and 'anxiety' between different versions of Latrodectus samples.

The task will point to the file previously dropped in the %APPDATA% folder.

```
1346. [0097.841] CoInitializeEx (pvReserved=0x0, dwCoInit=0x0) returned 0x0
1347. [0097.860] CoCreateInstance (in: rclsid=0x440250*(Data1=0xf87369f, Data2=0xa4e5, Data3=0x4cfc, Data4=([0]=0xbd, [1]=0x3e
1348. [0097.905] TaskScheduler:ITaskService:Connect (This=0x20c6c035bb0, serverName=0xc83e6ff0e0*(varType=0x0, wReserved1=0x78
1349. [0097.909] TaskScheduler:ITaskService:GetFolder (in: This=0x20c6c035bb0, Path="\\", ppFolder=0xc83e6ff150 | out: ppFolde
1350. [0097.909] TaskScheduler:ITaskService:NewTask (in: This=0x20c6c035bb0, flags=0x0, ppDefinition=0xc83e6ff078 | out: ppDef
1351. [0097.910] ITaskDefinition:get_Triggers (in: This=0x20c6c035db0, ppTriggers=0xc83e6fef38 | out: ppTriggers=0xc83e6fef38*
1352. [0097.910] ITriggerCollection:Create (in: This=0x20c6c036100, Type=1, ppTrigger=0xc83e6fef40 | out: ppTrigger=0xc83e6fef
1353. [0097.910] IUnknown:Release (This=0x20c6c036100) returned 0x1
1354. [0097.911] IUnknown:QueryInterface (in: This=0x20c6c036240, riid=0x440290*(Data1=0xb45747e0, Data2=0xeba7, Data3=0x4276,
1355. [0097.911] IUnknown:Release (This=0x20c6c036240) returned 0x2
1356. [0097.911] ITrigger:put_Id (This=0x20c6c036240, Id="TimeTrigger") returned 0x0
```

Figure 7: Function log reveals the setup of the scheduled task via the Component Object Model (COM) interface.

MUTEX

Latrodectus also tracks previously successful infections by creating a mutex on the target system. The hard-coded string 'runnung' has been consistent across all Latrodectus versions and it is checked before execution to prevent re-infecting already corrupted systems.

```
2565. [0033.903] CreateMutexW (lpMutexAttributes=0x0, bInitialOwner=0, lpName="runnung") returned 0x208
2566. [0033.903] GetLastError () returned 0x0
```

Figure 8: Function log showing the hard-coded mutex 'runnung'.

GROUP ID GENERATION

Enumerating the campaign name

So far, we have seen that each new version of the loader also introduces a new group ID. We suspect this may change in the future and there will be unique group IDs per version, if Latrodectus decides to switch to a malware-as-a-service model.

The group IDs are present in the initial C2 check-in traffic as '&group= parameter' and are represented as decimal numbers. They are also present in the malware sample as a string in an encrypted form. Since we have already discovered that a Fowler-Noll-Vo (FNV1a) hash is created based on the IDs, we can easily brute-force a reasonable number of potential group names if we don't have the decrypted campaign name string. Our approach was to create a wordlist of all possible combinations of the English alphabet (26 letters) and try to simply brute-force it. With a high-computing machine,

it is also reasonable to try mixed lowercase and uppercase variations, but for this short experiment, we stuck with just capitalizing the first letters.

Keep in mind that, since this is FNV1a 32-bit space, there could be multiple strings appearing under the same hash due to hash collisions. So in rare cases, there might be a slight chance that the script cannot find the original campaign name.

```
def generate_words(length):
    alphabet = 'abcdefghijklmnopqrstuvwxyz'
    words = []
    for combination in itertools.product(alphabet, repeat=length - 1):
        word = ''.join(combination).capitalize()
        words.append(word)
    return words

def write_words(words, file_path):
    with open(file_path, 'a') as f:
        for word in words:
            f.write(word + '\n')
```

Once we had given the script enough time to generate a massive (~ 130MB) wordlist (we kept it to seven letters), we simply called a FNV1a hash generator to iterate through the given words line by line:

```
fnv_prime_32 = 2**24 + 2**8 + 0x93
offset_basis_32 = 0x811c9dc5

def fnv1a_hash_32(bs):
    r = offset_basis_32
    for b in bs:
        r = r ^ b
        r = (r * fnv_prime_32) & 0xffffffff
    return r

if __name__ == '__main__':
    with open('wordlist.txt', 'rb') as file:
        wordlist = file.readlines()
    for words in wordlist:
        print("Campaign: " + Fore.YELLOW, wordbytes.decode('ascii'),"| FNV1a: ",
hex(fnv1a_hash_32(wordbytes)),"| Dec: ",int(hex(fnv1a_hash_32(wordbytes)), 16))
```

```
Campaign:  Wiskf | FNV1a:  0x1fe7c6bf | Dec:  535283391
Campaign:  Wiskg | FNV1a:  0x1ee7c52c | Dec:  518505772
Campaign:  Wiskh | FNV1a:  0x25e7d031 | Dec:  635949105
Campaign:  Wiski | FNV1a:  0x24e7ce9e | Dec:  619171486  <- CAMPAIGN NAME FOUND!
Campaign:  Wiskj | FNV1a:  0x23e7cd0b | Dec:  602393867
Campaign:  Wiskk | FNV1a:  0x22e7cb78 | Dec:  585616248
Campaign:  Wiskl | FNV1a:  0x29e7d67d | Dec:  703059581
```

Figure 9: Successfully brute-forcing the campaign name based on the decimal value of the campaign ID.

HARDWARE ID GENERATION

The loader also generates a unique hardware ID for each target host. This ID is based on the victim's Serial Volume ID and simply multiplied with a hard-coded constant. This constant is consistent so far in all observed Latrodectus versions: 0x19660D. The generated GUID is present in the initial C2 check-in request as '&guid= parameter'.

```
void *latro_botid_seed_volumeid()
{
  int VolumeInfo; // [rsp+40h] [rbp-448h]
  __int64 FirstVolume; // [rsp+48h] [rbp-440h]
  char lpFileSysFlags[4]; // [rsp+50h] [rbp-438h] BYREF
  char lpMaxComponentLength[12]; // [rsp+54h] [rbp-434h] BYREF
  char lpszVolumeName[528]; // [rsp+60h] [rbp-428h] BYREF
  char lpFileSysNameBuffer[536]; // [rsp+270h] [rbp-218h] BYREF

  latro_free_buffer((__int64)lpszVolumeName, 0x208uLL);
  latro_free_buffer((__int64)lpFileSysNameBuffer, 0x208uLL);
  FirstVolume = latro_api_addr_FindFirstVolumeW(lpszVolumeName, 260LL);
  if ( FirstVolume == -1 )
    return 0LL;
  VolumeInfo = latro_api_addr_GetVolumeInformationW(
                 lpszVolumeName,
                 0LL,
                 260LL,
                 &VolSerialNum,
                 lpMaxComponentLength,
                 lpFileSysFlags,
                 lpFileSysNameBuffer,
                 260,
                 0);
  latro_api_addr_FindVolumeClose(FirstVolume);
  if ( VolumeInfo )
    return &VolSerialNum;
  else
    return 0LL;
}
```

```
; __int64 __fastcall latro_botid_seed(unsigned int *)
latro_botid_seed proc near

arg_0= qword ptr  8

mov     [rsp+arg_0], rcx
mov     rax, [rsp+arg_0]
imul    eax, [rax], 19660Dh
mov     rcx, [rsp+arg_0]
mov     [rcx], eax
mov     rax, [rsp+arg_0]
mov     eax, [rax]
retn
latro_botid_seed endp
```

Figure 10: Generating the hardware ID, using the Volume Serial Number and the hard-coded constant (0x19660Dh).

SELF-DELETION

The loader uses a rather fascinating self-deletion technique: besides Latrodectus, we have previously observed this technique in DarkSide, Dark Power, HelloXD and other malware families. Ultimately, this method can delete a locked, or a currently running executable from disk. It uses the `SetFileInformationByHandle` *Windows* API to rename the executable's primary data stream and then facilitates the `DeleteFile` flag in the `FileDispositionInfo` class to trigger the disposition. There is a publicly available proof-of-concept code for this method on *GitHub* [7].

NETWORK C2

Upon successful infection, Latrodectus sends an initial check-in POST request with a hard-coded `User-Agent string: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Tob 1.1)`. This User-Agent header is consistent across all Latrodectus versions so far.

HTTP Requests (182) | DNS Requests (1)

Method	URL	Response	Dest. IP	Dest. Port	Verdict
POST	https://agrahusrat.com/test/	-	188.114.96.3	443	MALICIOUS
POST	https://agrahusrat.com/test/	-	188.114.96.3	443	MALICIOUS
POST	https://agrahusrat.com/test/	-	188.114.96.3	443	MALICIOUS
POST	https://agrahusrat.com/test/	-	188.114.96.3	443	MALICIOUS
POST	https://agrahusrat.com/test/	-	188.114.96.3	443	MALICIOUS
POST	https://agrahusrat.com/test/	-	188.114.96.3	443	MALICIOUS
POST	https://agrahusrat.com/test/	-	188.114.96.3	443	MALICIOUS

Figure 11: Network capture displays POST requests to the C2 server.

The POST request data includes parameter values collected from the system and also consists of a few hard-coded values, stored in the sample that identify the campaign and the sample version. These parameters are originally sent to the C2 server RC4 encrypted and then Base64 encoded.

We have seen Latrodectus C2 servers using the `/test/`, `/work/` and `/drive/` URI resources.

Figure 12: VMRay function log revealing the parameters being filled with values.

Parameter breakdown

Parameter	Value description
counter	C2 request throttling for evasion, (default = 0)
type	Type of request (check-in = 1)
guid	Hardware ID, seeded by the volume serial number, multiplied by the hard-coded value of 0x19660D
os	*Windows* OS version
arch	*Windows* architecture version
username	Username of the infected host
group	Campaign ID in decimal representation
version	Sample version
up	Potential sub-version number/update package
direction	C2 server
mac	Network card MAC address
computername	Hostname of infected host
domain	Host domain

C2 COMMAND HANDLER

Once an infection has taken place, the malicious process can receive further commands from the C2 server. Four different commands are available:

Directives	Description
CLEARURL	Clears the C2 table
URLS	Sends a new C2 URL to be stored in the C2 table
COMMAND	The command handler to other functionalities
ERROR	Sends an error message to the host

The COMMAND handler is the most interesting of these, as it can receive further sub-commands from the C2, as shown in Figure 13.

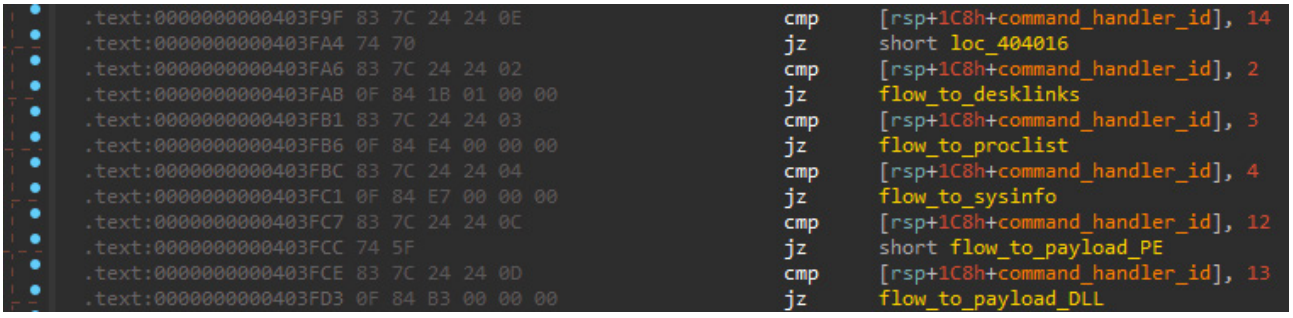

Figure 13: Command handler IDs for more functionalities.

Command ID	Description
2	Grabs filelist from the Desktop folder
3	Gets host process list
4	Collects sysinfo
12	Downloads and executes a next-stage PE
13	Downloads and executes a next-stage DLL
14	Downloads and executes a next-stage shellcode
15	Updates and restarts the bot
17	Terminates itself
18	Downloads IcedID loader and executes
19	Increases timeout
20	Resets the counter value
21	Executes a stealer module
22	Downloads and executes shellcode via Base64 function
25	Downloads a file to %APPDATA%

MALWARE CONFIGURATION EXTRACTION

The *VMRay Platform* currently extracts all important malware configuration information from the samples. These would include the C2 URLs, the exact version, mission ID, and any potential encryption keys that are used for the string encryption or C2 communication – namely, the RC4 key and the AES key (from v1.4 up to v1.9).

Figure 14: Successful malware configuration extraction for Latrodectus v1.8.

OPERATION ENDGAME 2.0

As of the time of writing this paper, law enforcement entities have made a significant effort in bringing down key, malicious infrastructure under Operation Endgame 2.0 [8]. Authorities have disrupted Latrodectus, and taken down around 300 servers worldwide overall. Looking through our telemetry, we have also noticed a significant drop in Latrodectus activity – however, new versions of the malware have surfaced, perhaps indicating that the original developers behind the family are still at large, and planning to revive the loader in the long run.

ENTERING VERSION 2.0

Just as we were wrapping up this paper, our livehunt alerts started to trigger on some potentially new Latrodectus samples. After a quick triage, we determined that the family has entered version 2.0 and we also spotted one v2.1 sample. We believe – at this time – that Latrodectus is in full-force to come back after the takedown and that its developers have implemented a few smaller, changes in its payload. One interesting detail we noticed is that the first-stage DLLs are now utilizing 400-500, randomly named DLL exports to confuse and derail analysis.

CONCLUSION

With its consistent updates and strategic adaptations, Latrodectus malware continues to challenge security measures worldwide.

The threat actors behind the malware family seem to iterate versions in a speedy fashion, perhaps to wear defenders out or potentially to prepare for a major change. Looking ahead, we suspect the prevalent loader will enter version 2.2 soon. Given the previous pace of development, it seems likely that even more updates are on the horizon. We noticed that some subversions even removed features from the loader, likely to reorganize its internal design. As this threat is still prevalent today, even after the takedown, we will continue to follow-up on future changes to have proper detection coverage and precise malware configuration extraction, as we expect Latrodectus to return.

SAMPLE HASHES

5cecb26a3f33c24b92a0c8f6f5175da0664b21d7c4216a41694e4a4cad233ca8 [9]

b45136abdb4284ac6d0096a237cb4f146decd034a556ff41356e666c3ce46910 [10]

b268f63bd11b98f249f9e2ec02184bf048fcffdc888a28c66cdcf4b5d7d4b65c [11]

REFERENCES

[1] Malpedia. Latrodectus. https://malpedia.caad.fkie.fraunhofer.de/details/win.latrodectus.

[2] Proofpoint. Latrodectus: This Spider Bytes Like Ice. 4 April 2024. https://www.proofpoint.com/us/blog/threat-insight/latrodectus-spider-bytes-ice.

[3] Europol. Largest ever operation against botnets hits dropper malware ecosystem. https://www.europol.europa.eu/media-press/newsroom/news/largest-ever-operation-against-botnets-hits-dropper-malware-ecosystem.

[4] Stepanic, D.; Bousseaden, S. Spring Cleaning with LATRODECTUS: A Potential Replacement for ICEDID. Elastic Security Labs. 16 May 2024. https://www.elastic.co/security-labs/spring-cleaning-with-latrodectus.

[5] MalwareBazaar. https://bazaar.abuse.ch/browse/signature/latrodectus/.

[6] OALabs / hashdb. https://github.com/OALabs/hashdb.

[7] LloydLabs / delete-self-poc. https://github.com/LloydLabs/delete-self-poc.

[8] Europol. Operation ENDGAME strikes again: the ransomware kill chain broken at its source. https://www.europol.europa.eu/media-press/newsroom/news/operation-endgame-strikes-again-ransomware-kill-chain-broken-its-source.

[9] VMray. https://www.vmray.com/analyses/_vt/5cecb26a3f33/report/overview.html.

[10] VMray. https://www.vmray.com/analyses/_tf/b45136abdb42/report/overview.html.

[11] VMray. https://www.vmray.com/analyses/_tf/b268f63bd11b/report/overview.html.