Latrodectus: A year in the making

vmray.com/latrodectus-a-year-in-the-making/

October 21, 2024

VIEW VMRAY’S ANALYSIS REPORT

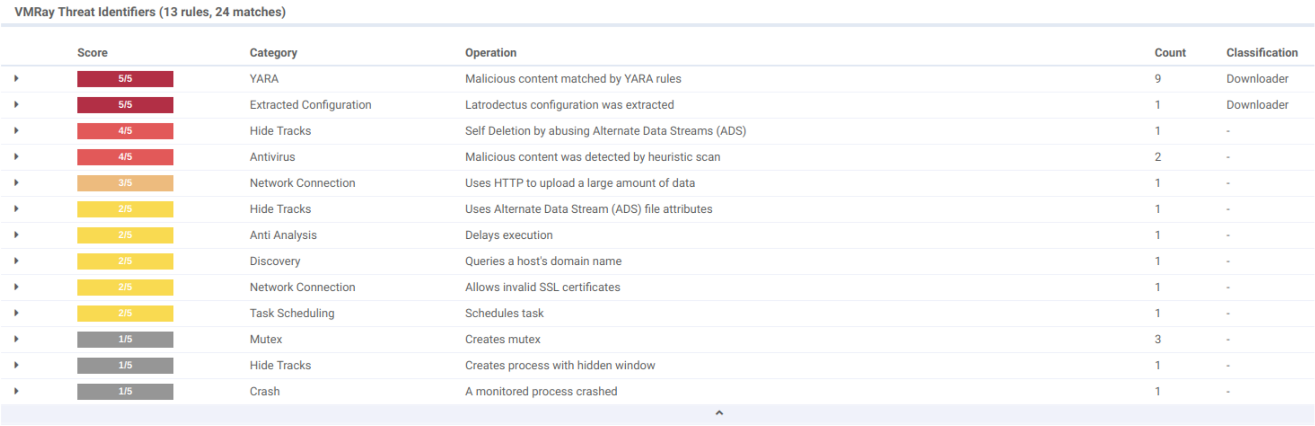
Overview

Latrodectus was first discovered by researchers in October 2023 and has been in heavy development ever since. The malware works mainly as a loader/downloader. Latrodectus has strong ties with the former, infamous loader IcedID, which was taken down in May 2024, thanks to the efforts of an international operation led by Europol and EC3. Since Operation Endgame, IcedID went under and Latrodectus is seen slowly taking its place in the cybercriminal ecosystem. Interestingly, Latrodectus also includes a specific C2 command, which can download a sample of IcedID loader.

Recently, the developers of this malware family have started on an iteration rampage, where multiple new versions were released in a relatively short time, perhaps in an effort to get ahead of the evergreen “cat-and-mouse” game between defenders and threat actors. These new versions only consist of small changes, even removal of existing features. The previous pace of development would suggest that Latrodectus will keep on iterating with new versions. Due to the prevalence of the malware family, we felt that adding malware configuration extraction support for all the recent versions was the best move forward for producing high-quality IoCs for our customers.

Furthermore, in this short blogpost, we would like to go over some of the most important features of the malware.

You can read our analysis report here (see Figure 1 for an overview).



VMRay Threat Identifiers (13 rules, 24 matches)

Score	Category	Operation	Count	Classification
5/5	YARA	Malicious content matched by YARA rules	9	Downloader
5/5	Extracted Configuration	Latrodectus configuration was extracted	1	Downloader
4/5	Hide Tracks	Self Deletion by abusing Alternate Data Streams (ADS)	1	-
4/5	Antivirus	Malicious content was detected by heuristic scan	2	-
3/5	Network Connection	Uses HTTP to upload a large amount of data	1	-
2/5	Hide Tracks	Uses Alternate Data Stream (ADS) file attributes	1	-
2/5	Anti Analysis	Delays execution	1	-
2/5	Discovery	Queries a host's domain name	1	-
2/5	Network Connection	Allows invalid SSL certificates	1	-
2/5	Task Scheduling	Schedules task	1	-
1/5	Mutex	Creates mutex	3	-
1/5	Hide Tracks	Creates process with hidden window	1	-
1/5	Crash	A monitored process crashed	1	-

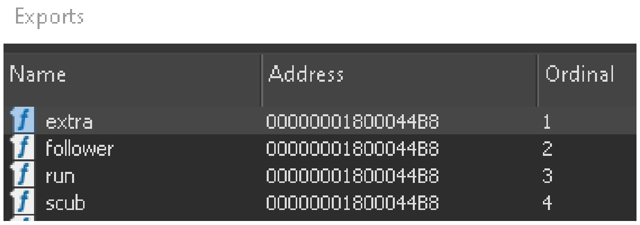
Figure 1: VMRay Platform's dynamic analysis reveals the malicious behavior of Latrodectus

Watch Video At:

https://youtu.be/qjyq3Cb2ioQ

Distribution and evolution

Latrodectus is distributed in a chain of JavaScript → MSI droppers, finally ending in the core DLL payload. The DLL payload usually has 4 unique-looking exports, utilizing the same export address, eventually running the same core logic when one decides to try all 4 exports.



Exports

Name	Address	Ordinal
extra	00000001800044B8	1
follower	00000001800044B8	2
run	00000001800044B8	3
scub	00000001800044B8	4

Figure 2: Latrodectus exhibiting 4 exports with the same export address

The loader has gone through several iterations: at the time of writing this post, the most up-to-date version is v1.8. Early versions have started to surface at the end of September 2023, while samples of the most recent version were just compiled at the end of September 2024. Effectively, we are talking about an evolution spanning over a year now.

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

Versions	Compiled time for first samples of respective versions
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
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

A few major changes worth highlighting across the versions:

- 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)
- From v1.4 on, the loader switched to AES-256 (CTR) string decryption with a hardcoded key and variable IV for each string
- Additional command IDs introduced for the command handler in v1.4, like the possibility of downloading an arbitrary file to %APPDATA%
- Some features that were previously incorporated are now removed from recent versions of samples, like the ADS self-deletion technique

Evasion techniques

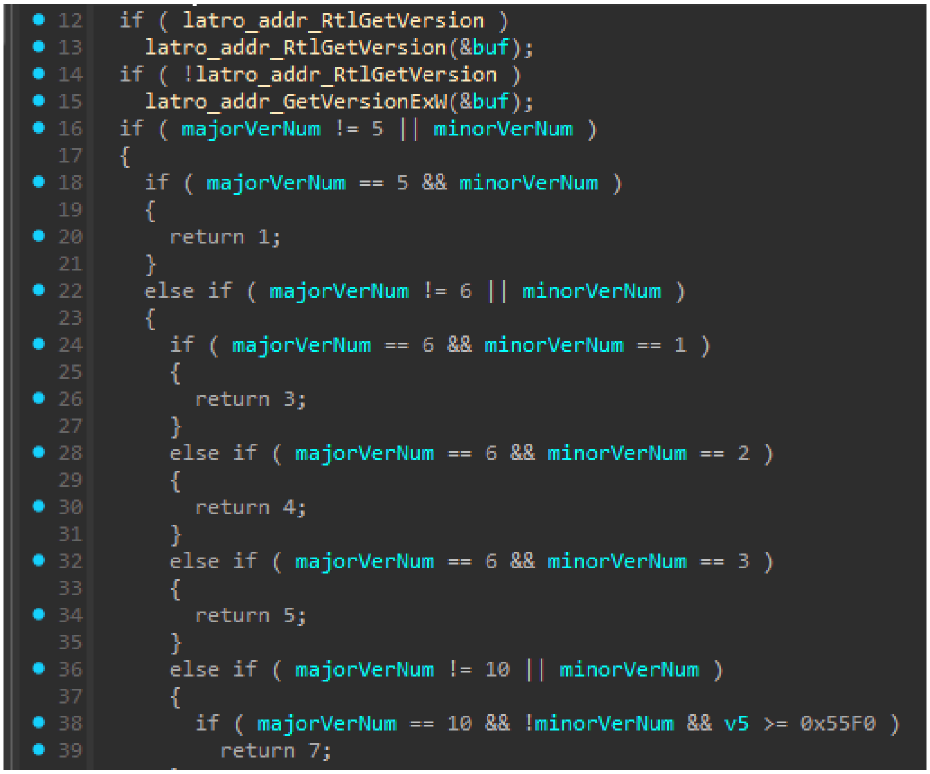
Overall, Latrodectus utilized 4 different anti-debugging and sandbox evasion techniques, these are as follows:

Process count check

This sanity process count check is most likely aimed at evading sandboxes as virtualized environments may not display the same number of installed and running applications as a real desktop environment would do.

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 needs at least 75 active processes to launch, otherwise it simply terminates. The other condition does the same check, just for Windows versions v6.3 or less (which would constitute Windows 8.1, Windows 8, Windows 7 and anything below). In this case, the loader needs at least 50 active processes to launch. This is to account for baseline levels for different versions of Windows OS.

The VMRay Platform allows customers to directly specify the amount of background processes during analysis time, successfully countering such sandbox evasion techniques.

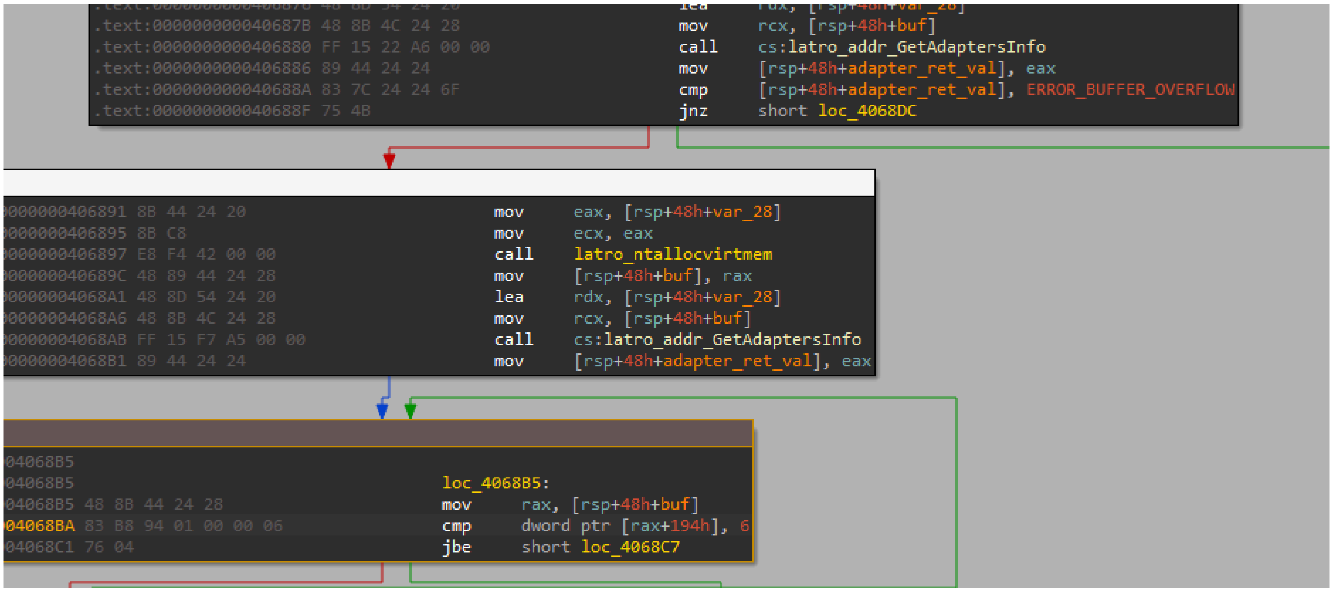


```
if ( latro_addr_RtlGetVersion )
  latro_addr_RtlGetVersion(&buf);
if ( !latro_addr_RtlGetVersion )
  latro_addr_GetVersionExW(&buf);
if ( majorVerNum != 5 || minorVerNum )
{
  if ( majorVerNum == 5 && minorVerNum )
  {
    return 1;
  }
  else if ( majorVerNum != 6 || minorVerNum )
  {
    if ( majorVerNum == 6 && minorVerNum == 1 )
    {
      return 3;
    }
    else if ( majorVerNum == 6 && minorVerNum == 2 )
    {
      return 4;
    }
    else if ( majorVerNum == 6 && minorVerNum == 3 )
    {
      return 5;
    }
    else if ( majorVerNum != 10 || minorVerNum )
    {
      if ( majorVerNum == 10 && !minorVerNum && v5 >= 0x55F0 )
        return 7;
```

Figure 3: Latrodectus enumerating Windows OS version

MAC address validity

The second evasion check enumerates the *_IP_ADAPTER_INFO* structure via the GetAdaptersInfo API function, then all hardware addresses of present network adapters are examined against the argument of 6. In the event, it does not equal to 6 bytes, the program will simply terminate. While MAC addresses have been standardized to 6 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 present in the BumbleBee loader as well.

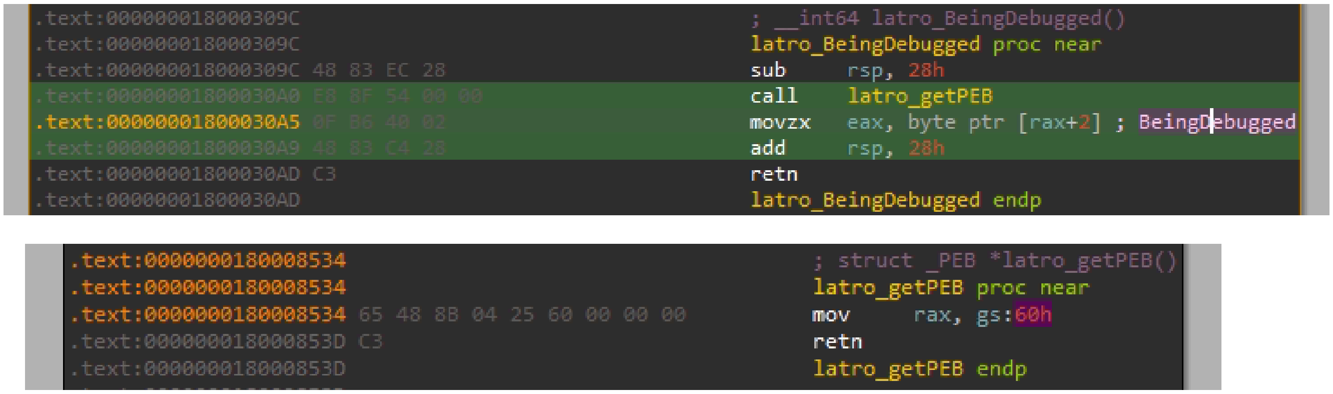


```
.text:0000000000406876 48 8D 54 24 20                    lea     rdx, [rsp+48h+var_28]
.text:000000000040687B 48 8B 4C 24 28                    mov     rcx, [rsp+48h+buf]
.text:0000000000406880 FF 15 22 A6 00 00                 call    cs:latro_addr_GetAdaptersInfo
.text:0000000000406886 89 44 24 24                       mov     [rsp+48h+adapter_ret_val], eax
.text:000000000040688A 83 7C 24 24 6F                    cmp     [rsp+48h+adapter_ret_val], ERROR_BUFFER_OVERFLOW
.text:000000000040688F 75 4B                             jnz     short loc_4068DC

000000406891 8B 44 24 20                 mov     eax, [rsp+48h+var_28]
000000406895 8B C8                       mov     ecx, eax
000000406897 E8 F4 42 00 00              call    latro_ntallocvirtmem
00000040689C 48 89 44 24 28              mov     [rsp+48h+buf], rax
0000004068A1 48 8D 54 24 20              lea     rdx, [rsp+48h+var_28]
0000004068A6 48 8B 4C 24 28              mov     rcx, [rsp+48h+buf]
0000004068AB FF 15 F7 A5 00 00           call    cs:latro_addr_GetAdaptersInfo
0000004068B1 89 44 24 24                 mov     [rsp+48h+adapter_ret_val], eax

04068B5
04068B5                               loc_4068B5:
04068B5 48 8B 44 24 28                mov     rax, [rsp+48h+buf]
04068BA 83 B8 94 01 00 00 06          cmp     dword ptr [rax+194h], 6
04068C1 76 04                         jbe     short loc_4068C7
```

Figure 4: A rare network card check to verify validity of MAC addresses

BeingDebugged

A 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 without calling the actual Windows API IsDebuggerPresent(), which may trigger some AV/EDR systems.



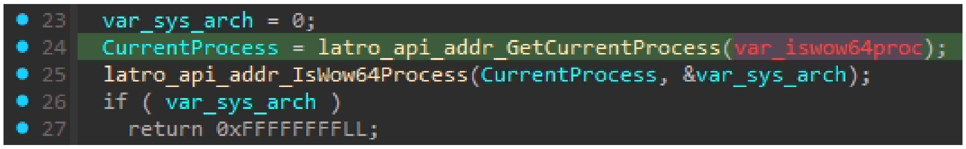
```
.text:000000018000309C                        ; __int64 latro_BeingDebugged()
.text:000000018000309C                        latro_BeingDebugged proc near
.text:000000018000309C 48 83 EC 28            sub     rsp, 28h
.text:00000001800030A0 E8 8F 54 00 00         call    latro_getPEB
.text:00000001800030A5 0F B6 40 02            movzx   eax, byte ptr [rax+2] ; BeingDebugged
.text:00000001800030A9 48 83 C4 28            add     rsp, 28h
.text:00000001800030AD C3                     retn
.text:00000001800030AD                        latro_BeingDebugged endp
```

```
.text:0000000180008534                              ; struct _PEB *latro_getPEB()
.text:0000000180008534                              latro_getPEB proc near
.text:0000000180008534 65 48 8B 04 25 60 00 00 00   mov     rax, gs:60h
.text:000000018000853D C3                           retn
.text:000000018000853D                              latro_getPEB endp
```

Figure 5: BeingDebugged flag being checked by walking the PEB

WOW64 process check

The next check is a validation of the current process, 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.



```
  var_sys_arch = 0;
  CurrentProcess = latro_api_addr_GetCurrentProcess(var_iswow64proc);
  latro_api_addr_IsWow64Process(CurrentProcess, &var_sys_arch);
  if ( var_sys_arch )
    return 0xFFFFFFFFLL;
```

Figure 6: Checking the running process against IsWow64Process

Encrypted strings

In order to make reverse engineering process harder, Latrodectus employs string encryption. The internal strings hold a significant amount of information on how the malware operates, what behavior 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 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 hardcoded key inside the sample and with a variable IV for each of encrypted strings.

Although, the encryption algorithm went through several changes as described previously, the storage of the encrypted strings remained almost similar. The prototype for these structures are simple: the encrypted strings are stored in the .data section of the DLL. In early versions of the loader, the first 4 bytes noted the XOR key and the delimiter bytes as well, the length of each strings are stored in the 5. and 6. bytes, and the remaining bytes are the actual encrypted data.

The recent versions, due to introducing the AES algorithm, a hardcoded key is burnt-in 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 again the actual encrypted data.

String encryption

Versions	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)	hardcoded	chunk[:2]	chunk[2:18]	chunk[18:18+data_length]	Not applicable
v1.5	AES-256 (CTR mode)	hardcoded	chunk[:2]	chunk[2:18]	chunk[18:18+data_length]	Not applicable
v1.7	AES-256 (CTR mode)	hardcoded	chunk[:2]	chunk[2:18]	chunk[18:18+data_length]	Not applicable

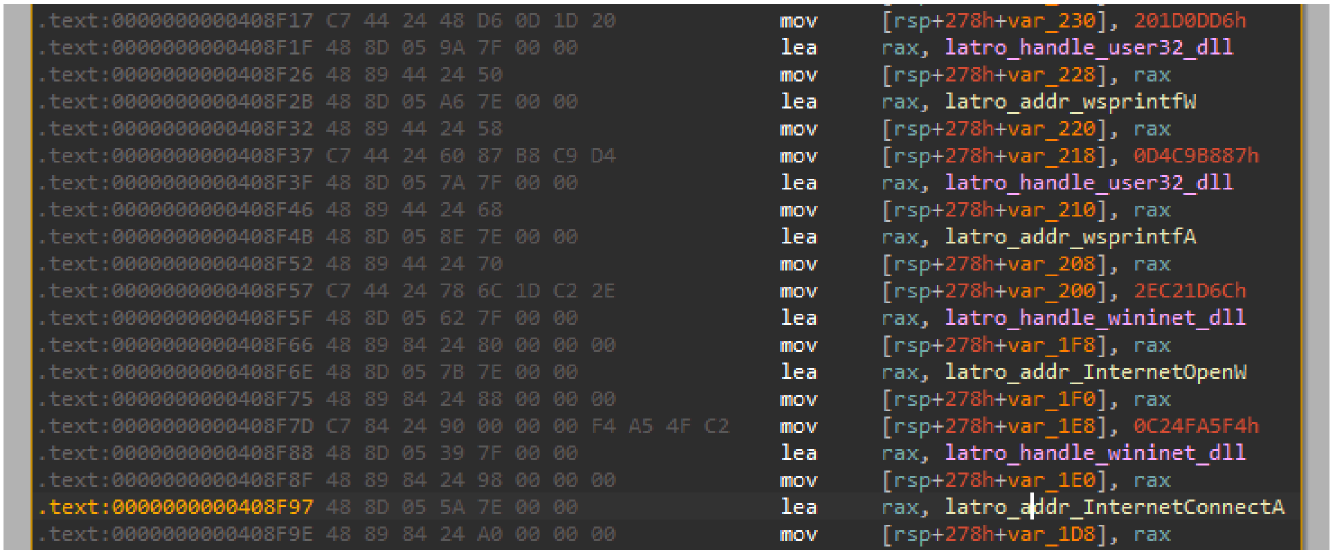
String encryption						
v1.8	AES-256 (CTR mode)	hardcoded	chunk[:2]	chunk[2:18]	chunk[18:18+data_length]	Not applicable

Figure 7: Encryption changes across versions

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, iphlpapi.dll via finding the files inside the \Windows\System32 folder, calculating the CRC32 checksums of the filenames and then comparing them with the hardcoded hashed values in the sample. The last step is then to call the LoadLibraryW function to finally load the library.

Once Latrodectus loaded all DLLs necessary, it continues to resolve the APIs by comparing the CRC32 checksums of the exported functions with the target values. The open-source project HashDB can help and save work here, as its Lookup Service can reverse the hash values and recreate the API names within an analysis. Reference: https://github.com/OALabs/hashdb



```
.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 8: CRC32-based API hashing in Latrodectus

Setting up persistence

Using a simple condition, the malware verifies if it is running from under the %APPDATA% folder: if that's not the case, it will copy itself to the location of either:

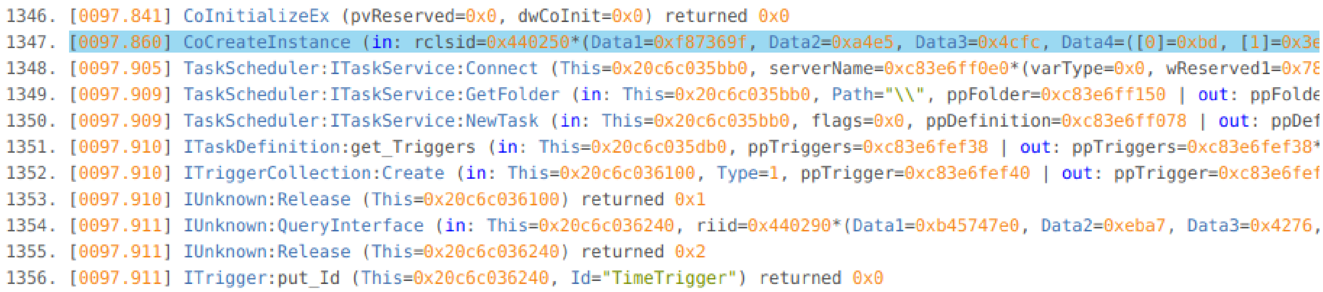
- %APPDATA%\Custom_update\Update_XXXXXXXX.dll (older versions)
- %APPDATA%\falsify_steward\confrontation_XXXXXXXX.dll (newer versions)

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 hardcoded constant described in the Hardware ID section of the blogpost.

The creativity of developers is again revealed at the next stage of persistence: instead of calling conventional APIs or scheduler commands, that would simply create a scheduled task, Latrodectus uses the Component Object Model (COM) interface to achieve persistence. Our function log clearly describes the behavior and it is easy to follow the chain of events. In the past, we have also taken a deep-dive into how the use of COM objects can blind malware analysis.

First, the sample calls the CoCreateInstance API to create and initialize an object, then 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 is changing between “Updater” or “anxiety” between different versions of Latrodectus samples.

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

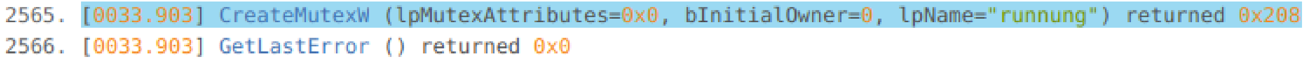


```
1346. [0097.841] CoInitializeEx (pvReserved=0x0, dwCoInit=0x0) returned 0x0
1347. [0097.860] CoCreateInstance (in: rclsid=0x440250*(Data1=0xf87369f, Data2=0xa4e5, Data3=0x4cfc, Data4=([0]=0xbd, [1]=0x3e
1348. [0097.905] TaskScheduler:ITaskService:Connect (This=0x20c6c035bb0, serverName=0xc83e6ff0e0*(varType=0x0, 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 9: VMRay Platform’s 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 hardcoded 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 10: VMRay Platform’s Function log showing the hardcoded 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 versions, if Latrodectus decides to switch to a “Malware-as-a-Service” model.

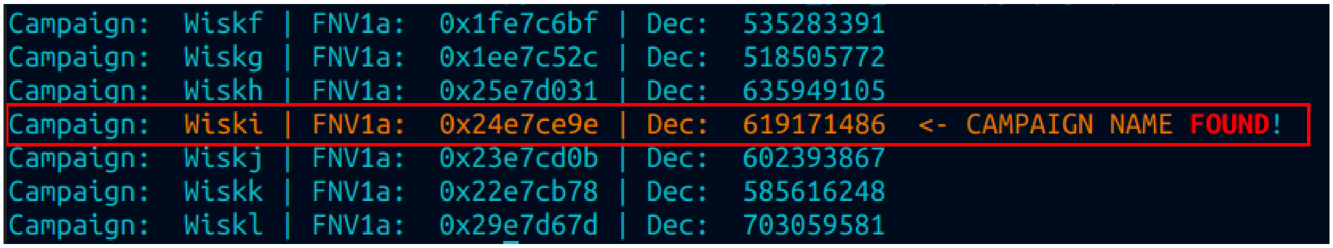
The group IDs are present in the initial C2 check-in traffic as &group= parameter and are represented as decimal numbers. They are also present in the malware sample as a string in an encrypted form. Since, we have already discovered that a Fowler–Noll–Vo (FNV1a) hash is created based off of the IDs, we can easily brute-force a reasonable amount of potential group names if we don’t have the decrypted campaign name string. Our approach was to create a word-list of all possible combinations of the English alphabet (26 letters) and try and 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 FNV-1a 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 gave enough time to the script to generate a massive (~ 130MB) wordlist (we kept it up to 7 letters), we can simply call 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))
```

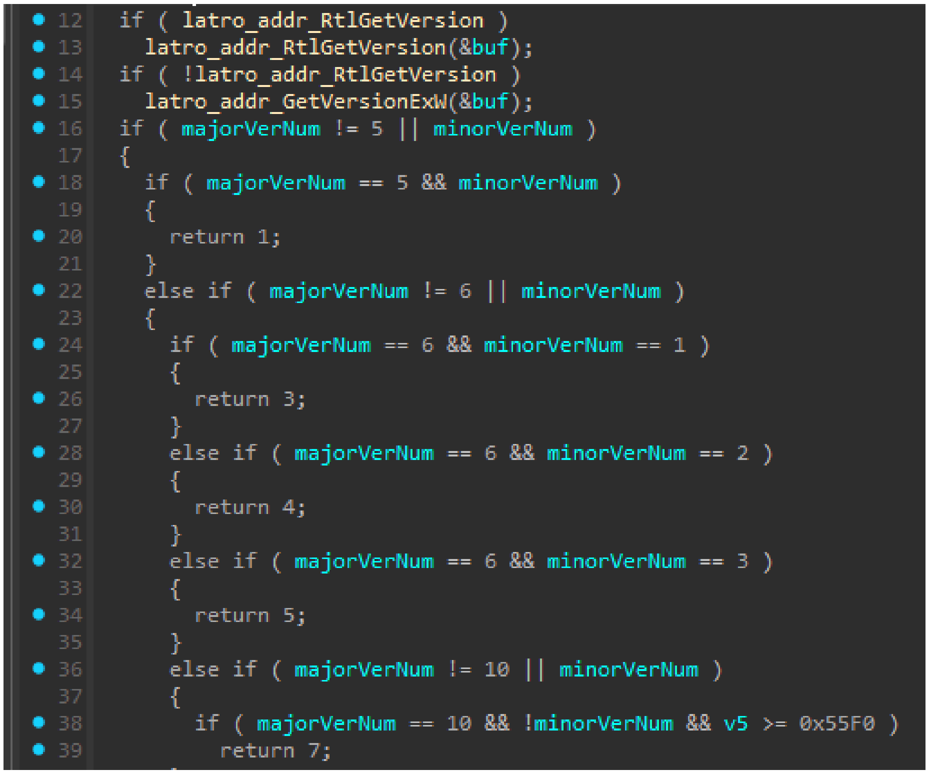


```
Campaign:  Wiskf | FNV1a:  0x1fe7c6bf | Dec:  535283391
Campaign:  Wiskg | FNV1a:  0x1ee7c52c | Dec:  518505772
Campaign:  Wiskh | FNV1a:  0x25e7d031 | Dec:  635949105
Campaign:  Wiski | FNV1a:  0x24e7ce9e | Dec:  619171486  <- CAMPAIGN NAME FOUND!
Campaign:  Wiskj | FNV1a:  0x23e7cd0b | Dec:  602393867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

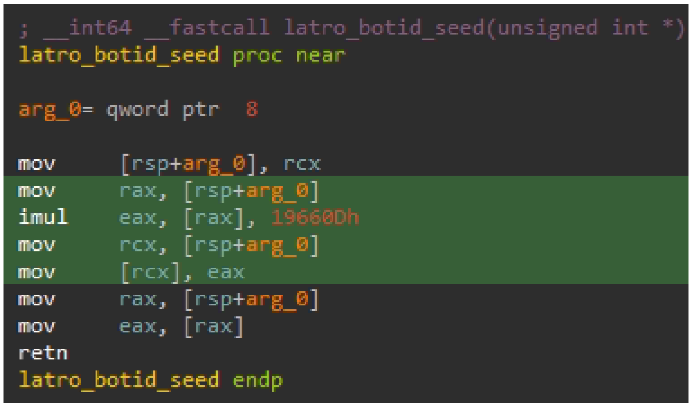
Figure 11: Successfully brute-forcing the campaign name based on the decimal value of the campaign ID

Hardware ID generation

The loader also generates a unique hardware ID for each target host. This ID is based off of the victim's Serial Volume ID and simply multiplied with a hardcoded 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.



```
if ( latro_addr_RtlGetVersion )
  latro_addr_RtlGetVersion(&buf);
if ( !latro_addr_RtlGetVersion )
  latro_addr_GetVersionExW(&buf);
if ( majorVerNum != 5 || minorVerNum )
{
  if ( majorVerNum == 5 && minorVerNum )
  {
    return 1;
  }
  else if ( majorVerNum != 6 || minorVerNum )
  {
    if ( majorVerNum == 6 && minorVerNum == 1 )
    {
      return 3;
    }
    else if ( majorVerNum == 6 && minorVerNum == 2 )
    {
      return 4;
    }
    else if ( majorVerNum == 6 && minorVerNum == 3 )
    {
      return 5;
    }
    else if ( majorVerNum != 10 || minorVerNum )
    {
      if ( majorVerNum == 10 && !minorVerNum && v5 >= 0x55F0 )
        return 7;
```



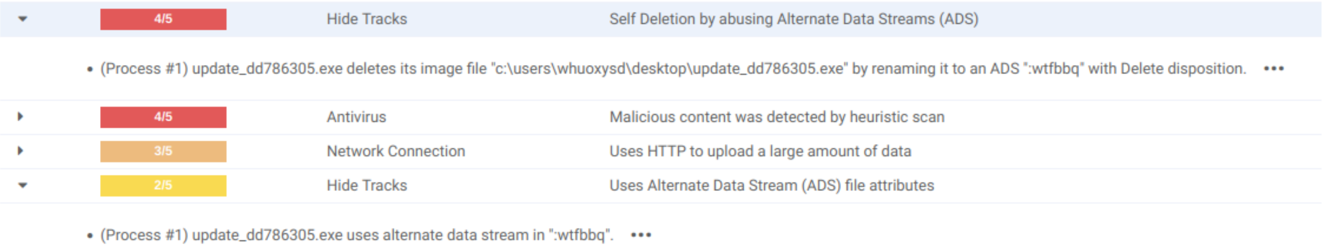
```
; __int64 __fastcall latro_botid_seed(unsigned int *)
latro_botid_seed proc near

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 12: Generating the hardware ID, using the Volume Serial Number and the hardcoded constant (0x19660Dh)

Self-deletion

The loader uses a rather fascinating self-deletion technique: besides Latrodectus, we have previously observed this technique in both 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: https://github.com/LloydLabs/delete-self-poc

We have – uniquely in the industry – tried creating a future-proof detection coverage specifically for this technique, which is now observable as a VMRay Threat Identifier (VTI).



▾	4/5	Hide Tracks	Self Deletion by abusing Alternate Data Streams (ADS)

- (Process #1) update_dd786305.exe deletes its image file "c:\users\whuoxysd\desktop\update_dd786305.exe" by renaming it to an ADS ":wtfbbq" with Delete disposition. •••

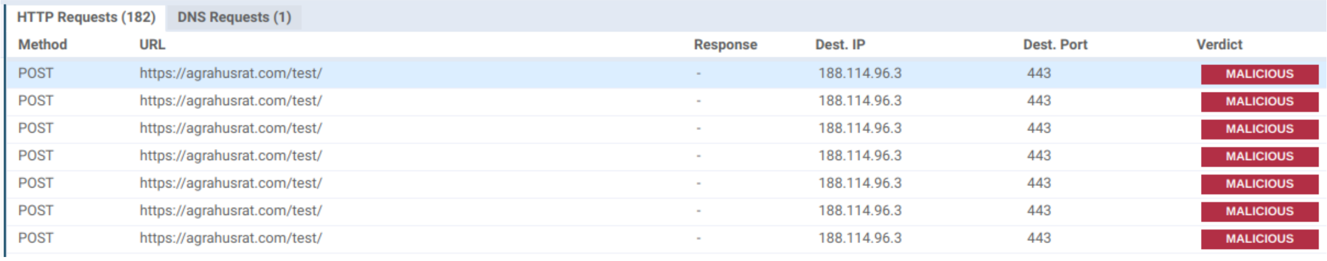
▸	4/5	Antivirus	Malicious content was detected by heuristic scan
▸	3/5	Network Connection	Uses HTTP to upload a large amount of data
▾	2/5	Hide Tracks	Uses Alternate Data Stream (ADS) file attributes

- (Process #1) update_dd786305.exe uses alternate data stream in ":wtfbbq". •••

Figure 13: The VMRay Platform triggering on ADS self-deletion technique via VTIs

Network C2

Upon successful infection, Latrodectus sends an initial check-in POST request with a hardcoded 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.

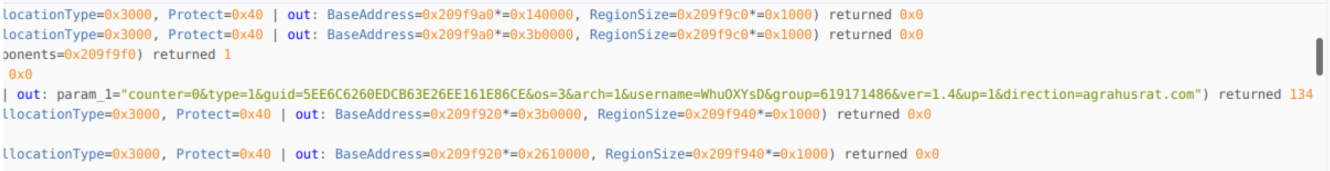


HTTP Requests (182) | DNS Requests (1)

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 14: VMRay Platform's 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 hardcoded values, stored in the sample that identifies the campaign and the sample version. These parameters are originally sent towards the C2 server RC4 encrypted and then base64 encoded, but the VMRay Platform easily captures the decrypted values in the function logs.



```
locationType=0x3000, Protect=0x40 | out: BaseAddress=0x209f9a0*=0x140000, RegionSize=0x209f9c0*=0x1000) returned 0x0
locationType=0x3000, Protect=0x40 | out: BaseAddress=0x209f9a0*=0x3b0000, RegionSize=0x209f9c0*=0x1000) returned 0x0
ponents=0x209f9f0) returned 1
 0x0
| out: param_1="counter=0&type=1&guid=5EE6C6260EDCB63E26EE161E86CE&os=3&arch=1&username=WhuOXYsD&group=619171486&ver=1.4&up=1&direction=agrahusrat.com") returned 134
llocationType=0x3000, Protect=0x40 | out: BaseAddress=0x209f920*=0x3b0000, RegionSize=0x209f940*=0x1000) returned 0x0

llocationType=0x3000, Protect=0x40 | out: BaseAddress=0x209f920*=0x2610000, RegionSize=0x209f940*=0x1000) returned 0x0
```

Figure 15: VMRay Platform's Function Log revealing the parameters being filled with values

Each of these parameters serve a specific purpose:

Parameter	Value description
counter	C2 request throttling for evasion, (default = 0)
type	Type of the request (check-in = 1)
guid	Hardware ID, seeded by the volume serial number, multiplied by the hardcoded value of 0x19660D
os	Windows OS version
arch	Windows architecture version

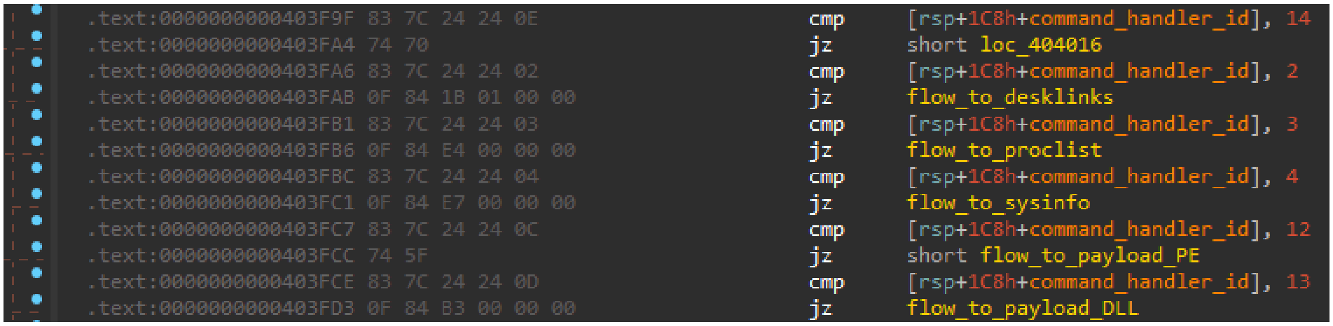
Parameter	Value description
username	Username of the infected host
group	Campaign ID in decimal representation
version	Sample version
up	Potential sub-version number/update package
direction	C2 server
mac	Network card MAC address
computername	Hostname of infected host
domain	Host domain

C2 command handler

Once an infection took place, the malicious process can receive further commands from the C2 server, 4 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 one as it can receive the following further sub-commands from the C2:



```
.text:0000000000403F9F 83 7C 24 24 0E                    cmp     [rsp+1C8h+command_handler_id], 14
.text:0000000000403FA4 74 70                             jz      short loc_404016
.text:0000000000403FA6 83 7C 24 24 02                    cmp     [rsp+1C8h+command_handler_id], 2
.text:0000000000403FAB 0F 84 1B 01 00 00                 jz      flow_to_desklinks
.text:0000000000403FB1 83 7C 24 24 03                    cmp     [rsp+1C8h+command_handler_id], 3
.text:0000000000403FB6 0F 84 E4 00 00 00                 jz      flow_to_proclist
.text:0000000000403FBC 83 7C 24 24 04                    cmp     [rsp+1C8h+command_handler_id], 4
.text:0000000000403FC1 0F 84 E7 00 00 00                 jz      flow_to_sysinfo
.text:0000000000403FC7 83 7C 24 24 0C                    cmp     [rsp+1C8h+command_handler_id], 12
.text:0000000000403FCC 74 5F                             jz      short flow_to_payload_PE
.text:0000000000403FCE 83 7C 24 24 0D                    cmp     [rsp+1C8h+command_handler_id], 13
.text:0000000000403FD3 0F 84 B3 00 00 00                 jz      flow_to_payload_DLL
```

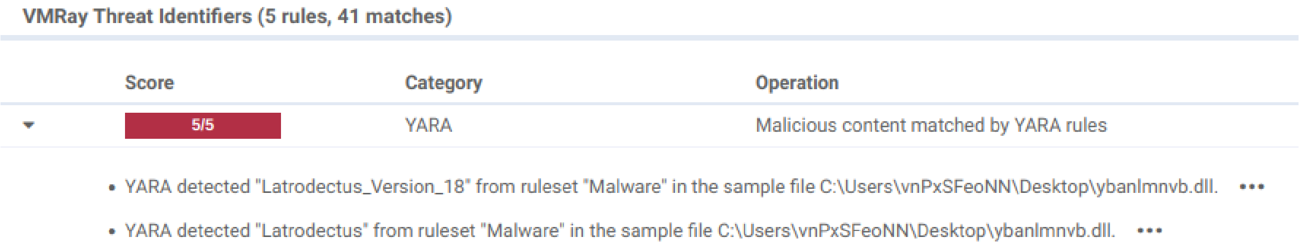
Figure 16: Command handler IDs for more functionalities

Command ID	Description
2	Grabs filelist from the Desktop folder
3	Gets host process list

Command ID	Description
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 execute
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%

YARA coverage

We have introduced several forward-looking YARA signatures to detect all versions of the family. We also provide version-based signatures to aid customers with up-to-date information on the exact version of Latrodectus in question.



VMRay Threat Identifiers (5 rules, 41 matches)

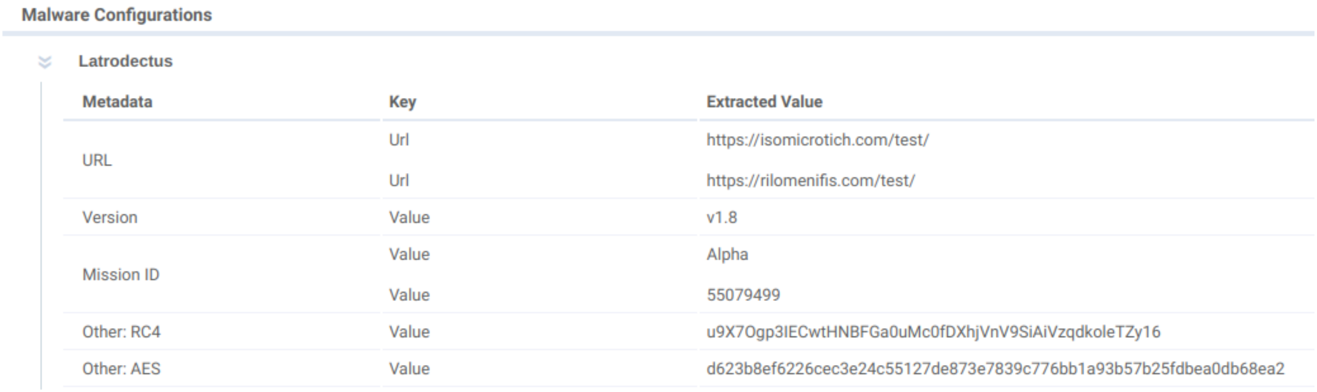
Score	Category	Operation
5/5	YARA	Malicious content matched by YARA rules

- YARA detected "Latrodectus_Version_18" from ruleset "Malware" in the sample file C:\Users\vnPxSFeoNN\Desktop\ybanlmnvb.dll. •••
- YARA detected "Latrodectus" from ruleset "Malware" in the sample file C:\Users\vnPxSFeoNN\Desktop\ybanlmnvb.dll. •••

Figure 17: VMRay Platform’s report showing YARA detection signatures on Latrodectus samples

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 v1.8).



Malware Configurations

Latrodectus

Metadata	Key	Extracted Value
URL	Url	https://isomicrotich.com/test/
	Url	https://rilomenifis.com/test/
Version	Value	v1.8
Mission ID	Value	Alpha
	Value	55079499
Other: RC4	Value	u9X7Ogp3IECwtHNBFGa0uMc0fDXhjVnV9SiAiVzqdkoleTZy16
Other: AES	Value	d623b8ef6226cec3e24c55127de873e7839c776bb1a93b57b25fdbea0db68ea2

Figure 18: VMRay Platform’s successful malware configuration extraction for Latrodectus v1.8

Conclusion

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 substantial major change. We suspect the prevalent loader will enter into version 2.0 soon, as the previous pace of development seems to indicate even more updates incoming. Interestingly, we have seen that subversions even removed certain features from the loader, perhaps in an attempt to refactor some of the internal structures. As this threat is still prevalent today, we will make sure to follow-up on future changes to aid customers with proper detection coverage and precise malware configuration extraction.

The VMRay Platform currently detects all Latrodectus versions up to v1.8 and can acquire malware configuration from all working samples.

References

IoCs

C2 URLs	hxxps://antyparkov[.]site/live/ hxxps://aytobusesre[.]com/live/ hxxps://carflotyup[.]com/live/ hxxps://drifajizo[.]fun/live/ hxxps://coolarition[.]com/live/ hxxps://finjuiceer[.]com/live/ hxxps://grebiunti[.]top/live/ hxxps://grunzalom[.]fun/live/ hxxps://illoskanawer[.]com/live/ hxxps://jertacco[.]com/live/ hxxps://saicetyapy[.]space/live/ hxxps://scifimond[.]com/live/ hxxps://skinnyjeanso[.]com/live/ hxxps://stratimasesstr[.]com/live/ hxxps://stripplasst[.]com/live/ hxxps://titnovacrion[.]top/live/ hxxps://trymeakafr[.]com/live/ hxxps://winarkamaps[.]com/live/ hxxps://workspacin[.]cloud/live/ hxxps://worlpquano[.]com/live/ hxxps://zumkoshapsret[.]com/live/ hxxps://minrezviko[.]com/test/ hxxps://pomaspoteraka[.]com/test/ hxxps://finilamedima[.]com/test/ hxxps://restoreviner[.]com/test/ hxxps://peronikilinfer[.]com/test/ hxxps://rilomenifis[.]com/test/ hxxps://isomicrotich[.]com/test/
Mutex	runnung
Scheduled task name	Updater anxiety
Persistence location	%APPDATA%\falsify_steward\confrontation_XXXXXXXX.dll %APPDATA%\Custom_update\Update_XXXXXXXX.dll
RC4 keys	12345 2sDbsEUXvhgLOO4Irt8AF6eI3jJ0M1MowXyao00Nn6ZUjtjXwb u9X7Ogp3IECwtHNBFGa0uMc0fDXhjVnV9SiAiVzqdkoleTZy16 eNIHaXC815vAqddR21qsuD35eJFL7CnSOLI9vUBdcb5RPcS0h6 EhAyPSHvva9CvL6OIddDJvDXHJjoMsqXyjraKyYmXFqDGdAYyO 9edoY7pK6eQfntcLBNU1WSkauwf1sHj4I8vTuAddXvPwYbJPeP v9JrWM4aDsviWsTfSCgX5Ed98pH6kMpQr1VWWj5LTMiC5C5Lna k2C0I3yY0ZDMCy4zFZDFnCD3mzc4fFdEMw5uF1n6u59eGG2NDN xkxp7pKhnkQxUokR2dl00qsRa6Hx0xvQ31jTD7EwUqj4RXWtHwELbZFbOoqCnXI8

Group/Campaign IDs	Alpha (v1.8) Alpha (v1.7) Ceres Compati Delta (v1.5) Electrol Facial Jupiter Liniska Littlehw Mars Mercury Neptun Novik Olimp Supted Trusted Venus Wiski (v1.4)
Versions	v1.1a v1.1b v1.2 v1.3 v1.4 v1.5 v1.7 v1.8