REPUBLIC OF ALBANIA
NATIONAL CYBER SECURITY AUTHORITY
CYBER SECURITY ANALYSIS DIRECTORATE

Analysis
Spear-Phishing Campaign
Malicious File *OneDrive-latest-v3.zip*

Version: 1.0
Date: 26/06/2026

Street “Papa Gjon Pali II” nr.3 Tiranë;
web: www.aksk.gov.al E-mail: info@aksk.gov.al
Tel./Fax : 04 2221 039

CONTENT

List of Figures:

This analysis has limitations and should be interpreted with caution!

Some of these limitations include:

Phase One:

Information sources: The analysis is based on the information available at the time it was prepared. Certain aspects may differ from current developments.

Phase Two:

Scope and depth of analysis: Due to resource limitations, certain aspects of the malicious files may not have been examined in depth. Any additional information that was not previously available may result in changes to the analysis.

Phase Three:

Information security: To protect sources and confidential information, certain details may have been generalized or omitted from the analysis. This decision was taken to preserve the integrity and security of the data used.

AKSK reserves the right to amend, update, or otherwise revise any part of this analysis without prior notice.

This analysis is not a final document.

The findings are based on the information available at the time of the investigation. No guarantee is provided regarding possible changes or future updates to the reported information. The authors accept no liability for misuse of this analysis or for the consequences of any decision made on its basis. This analysis is not a final document.

Technical Information

A spear-phishing campaign has been identified that uses a copy of **Microsoft OneDrive** and fraudulent functionality to deceive users and obtain initial access to computers and systems. Threat actors distribute phishing emails from compromised email accounts, including **mnezeriti@mou.gr**, with the subject line **Document 6-17-2026**. The message body contains a URL that redirects users to SharePoint at **hxxps://sharepoint[.]epa[.]gov[.]gr/custom/,** where they are prompted to perform an update.

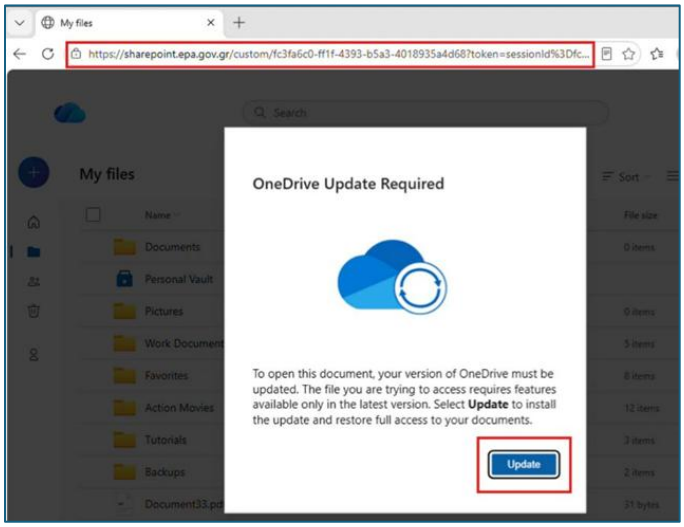


Figura 1. Portali pas aksesimit të URL .

Spear-Phishing Scheme Analysis

After the user clicks the **Update** button, the malicious archive **OneDrive-latest-v3.zip** is downloaded automatically.

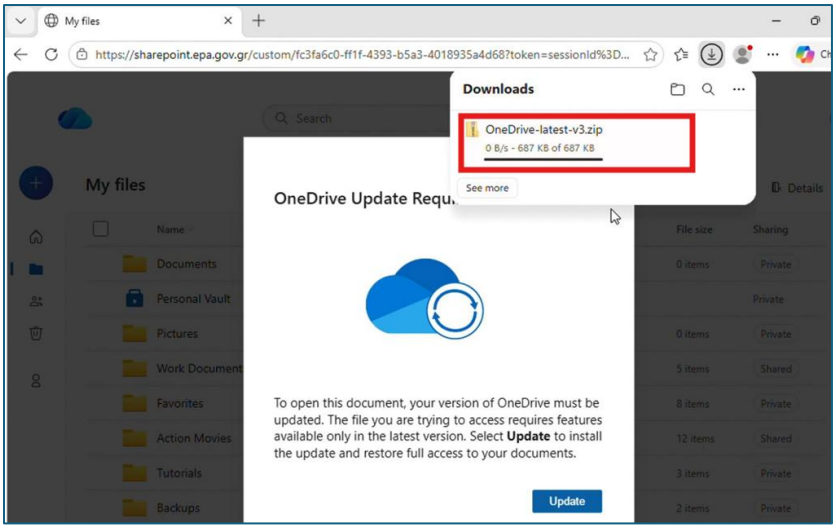


Figure 2. Download of the malicious file.

After the archive is extracted, four additional files are displayed. These files are analyzed in the

sections below.

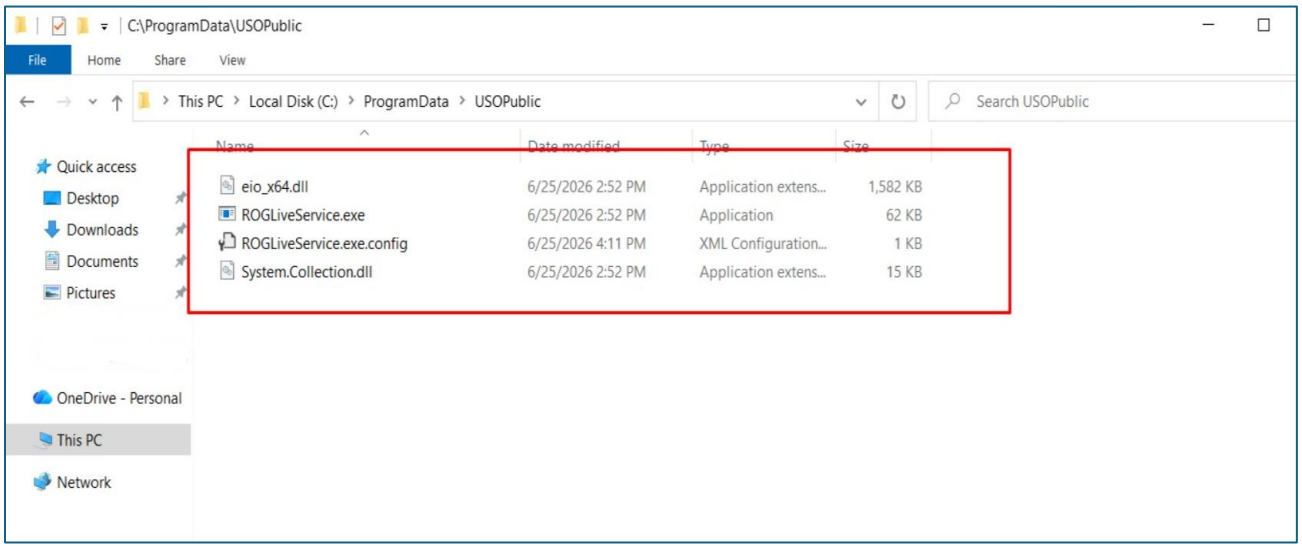


Figure 3. Extraction of the archived file.

OneDrive-latest-v3.exe is an executable compiled using **.NET**. The file itself appears clean and legitimate.

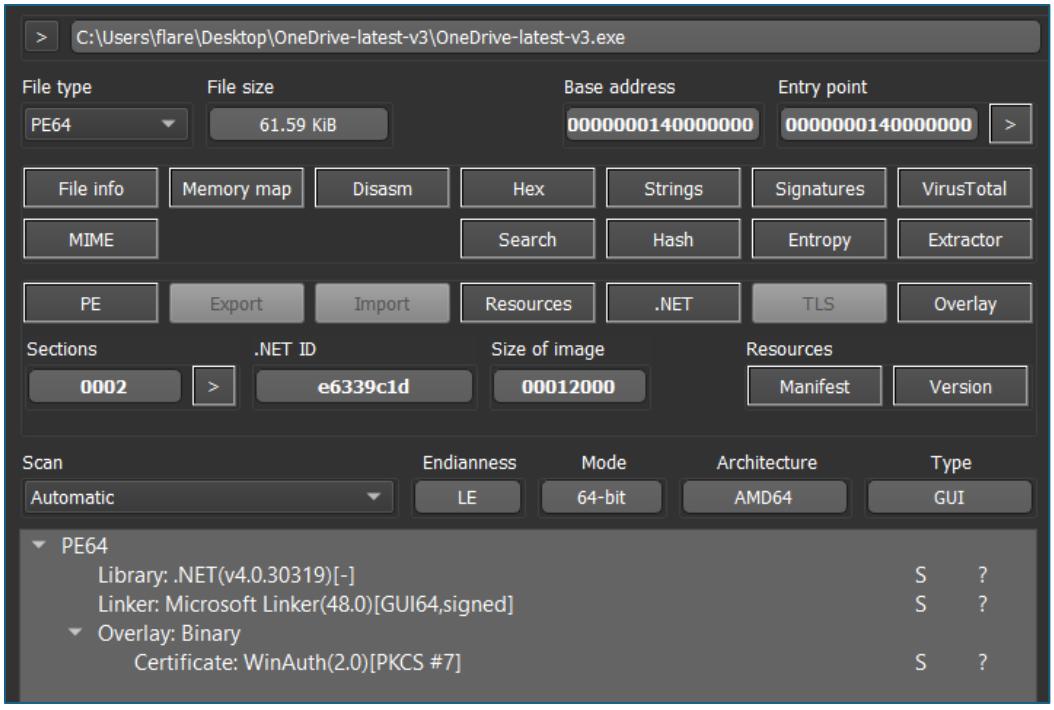


Figure 4. .NET compilation.

The purpose of this executable is to load another **.NET** file, specifically a **DLL** named **System.Collection.dll**. This behavior is confirmed during decompilation, where the XML

configuration file **OneDrive-latest-v3.exe.config** specifies the DLL invocation.

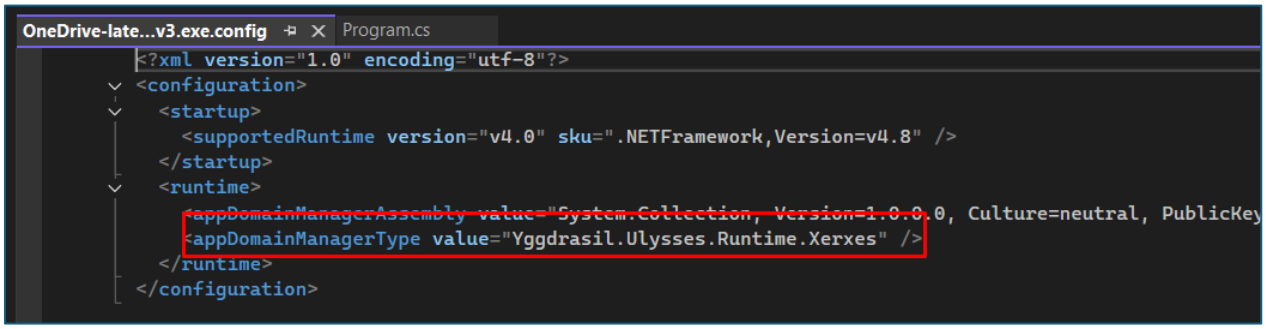


Figure 5. Malicious configuration file.

The objective is to initiate the second execution stage of the **Xerxes** class via the **InitializeNewDomain** function when the **.exe** file is run. The **System.Collection.dll** file inherits from the **AppDomainManager** class and triggers **InitializeNewDomain**, which implements several additional functions for specific purposes.

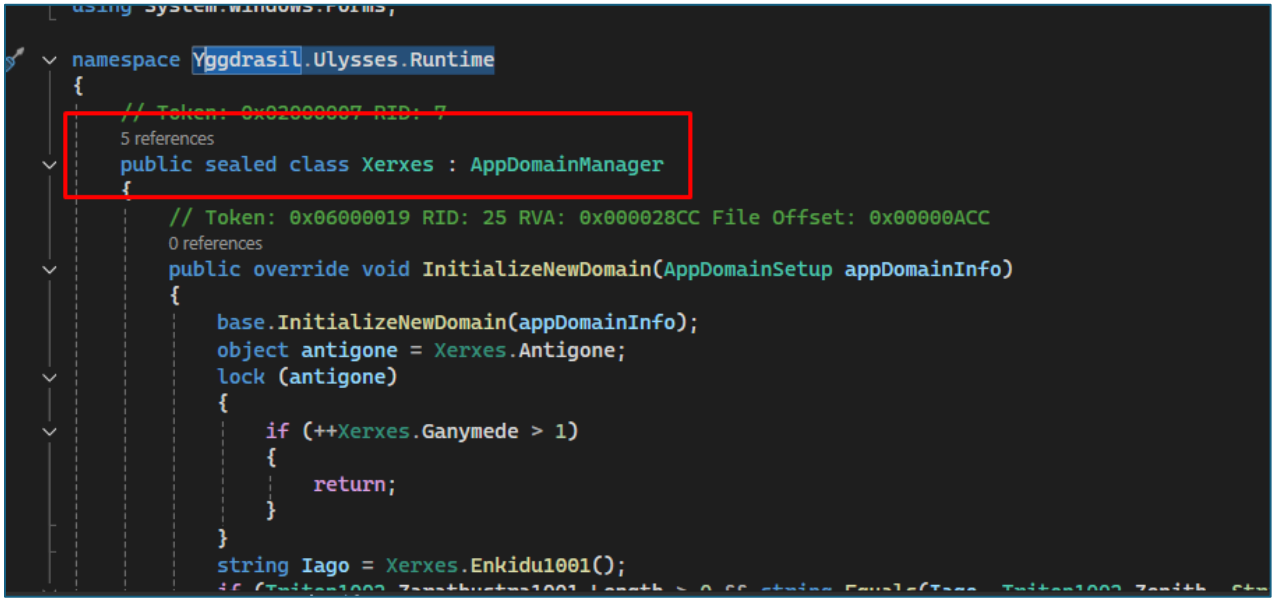
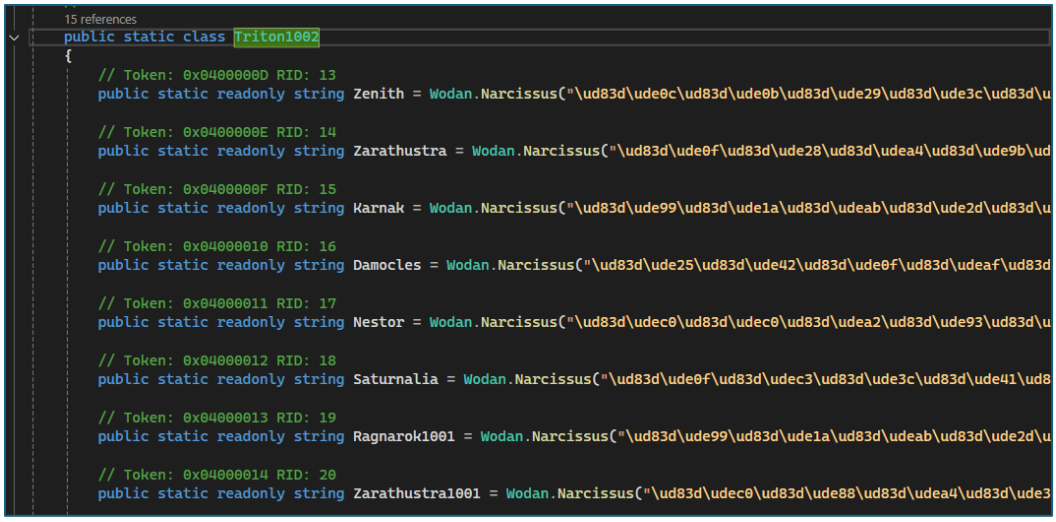


Figure 6. Xerxes class.

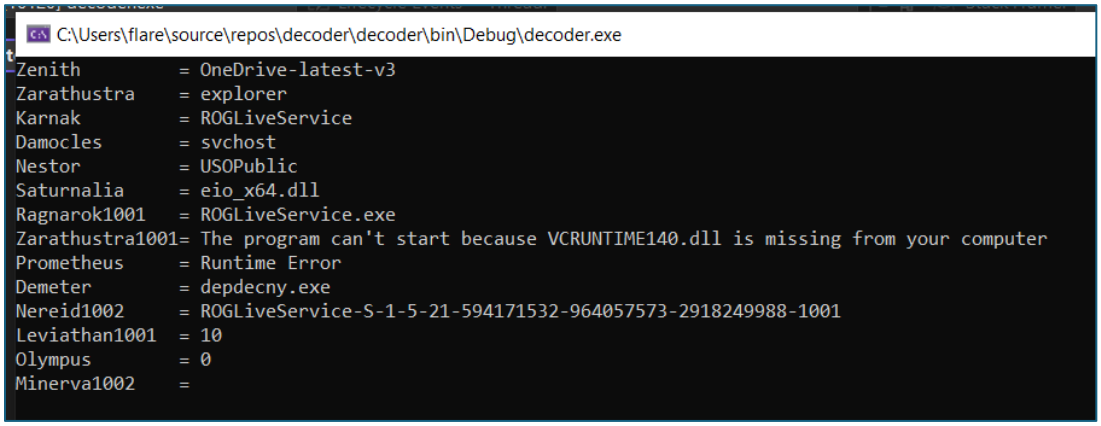
At the beginning of the analysis, several functions and strings with obfuscated names are identified in the code; their actual meaning becomes apparent during execution. The most important class is **Triton1002**, which contains the encoded and obfuscated values shown below.



```
15 references
public static class Triton1002
{
    // Token: 0x0400000D RID: 13
    public static readonly string Zenith = Wodan.Narcissus("\ud83d\ude0c\ud83d\ude0b\ud83d\ude29\ud83d\ude3c\ud83d\u

    // Token: 0x0400000E RID: 14
    public static readonly string Zarathustra = Wodan.Narcissus("\ud83d\ude0f\ud83d\ude28\ud83d\udea4\ud83d\ude9b\ud

    // Token: 0x0400000F RID: 15
    public static readonly string Karnak = Wodan.Narcissus("\ud83d\ude99\ud83d\ude1a\ud83d\udeab\ud83d\ude2d\ud83d\u

    // Token: 0x04000010 RID: 16
    public static readonly string Damocles = Wodan.Narcissus("\ud83d\ude25\ud83d\ude42\ud83d\ude0f\ud83d\udeaf\ud83d

    // Token: 0x04000011 RID: 17
    public static readonly string Nestor = Wodan.Narcissus("\ud83d\udec0\ud83d\udec0\ud83d\udea2\ud83d\ude93\ud83d\u

    // Token: 0x04000012 RID: 18
    public static readonly string Saturnalia = Wodan.Narcissus("\ud83d\ude0f\ud83d\udec3\ud83d\ude3c\ud83d\ude41\ud8

    // Token: 0x04000013 RID: 19
    public static readonly string Ragnarok1001 = Wodan.Narcissus("\ud83d\ude99\ud83d\ude1a\ud83d\udeab\ud83d\ude2d\u

    // Token: 0x04000014 RID: 20
    public static readonly string Zarathustra1001 = Wodan.Narcissus("\ud83d\udec0\ud83d\ude88\ud83d\udea4\ud83d\ude3
```

Figure 7. Obfuscated strings.

For these hardcoded values, each string is decoded through an implementation in the **Wodan** class and the **Narcissus** function. Each character sequence is passed to **Narcissus** as input and decoded at runtime. By debugging and modifying the source code, the output of each string can be obtained.



```
C:\Users\flare\source\repos\decoder\decoder\bin\Debug\decoder.exe
Zenith          = OneDrive-latest-v3
Zarathustra     = explorer
Karnak          = ROGLiveService
Damocles        = svchost
Nestor          = USOPublic
Saturnalia      = eio_x64.dll
Ragnarok1001    = ROGLiveService.exe
Zarathustra1001= The program can't start because VCRUNTIME140.dll is missing from your computer
Prometheus      = Runtime Error
Demeter         = depdecny.exe
Nereid1002      = ROGLiveService-S-1-5-21-594171532-964057573-2918249988-1001
Leviathan1001   = 10
Olympus         = 0
Minerva1002     =
```

Figure 8. Decoded string values.

After decoding, each variable is traced to determine its purpose. The key variable in the infection chain is **Saturnalia**, which resolves to **eio_x64.dll**. Following the call path in the **Jupiter1001.cs** class and the **Xanadu()** function reveals a call to **Aether**, which invokes the **LoadLibraryW** API from **kernel32.dll** to load the native DLL **eio_x64.dll**.

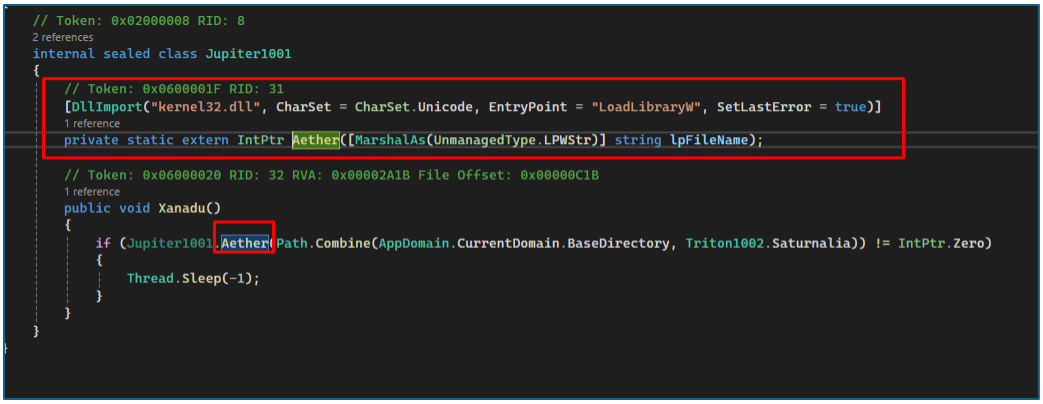


Figure 9. Loading of the native DLL eio_x64.dll.

The **Boreas()** function in **Xerxes.cs** creates two strings at runtime. The first variable stores the value **USOPublic** and combines it with **C:\ProgramData**. The second variable, **text2**, decodes to **RogLiveService** and is passed as a parameter to the **triton()** function of the **Chimera** class. These values are then passed as parameters to the **Triton** function.

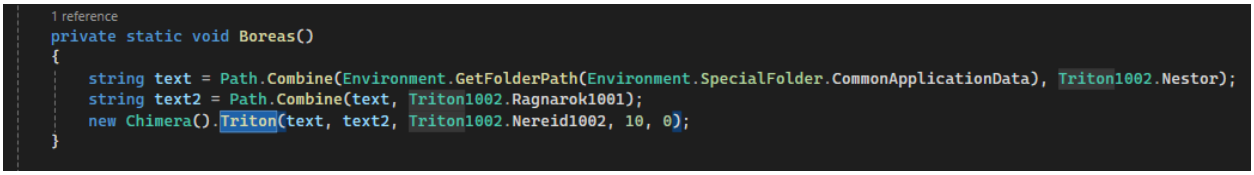


Figure 10. Boreas function.

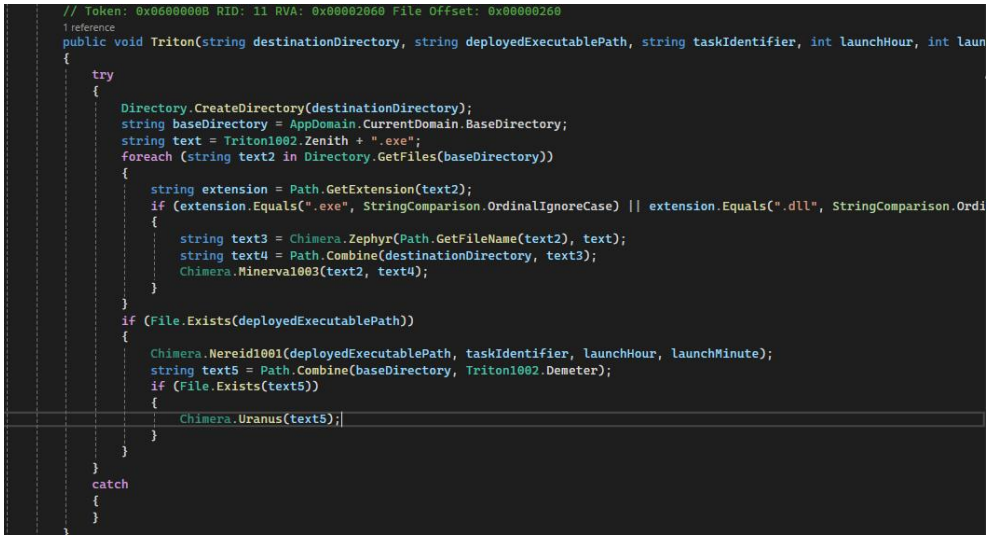
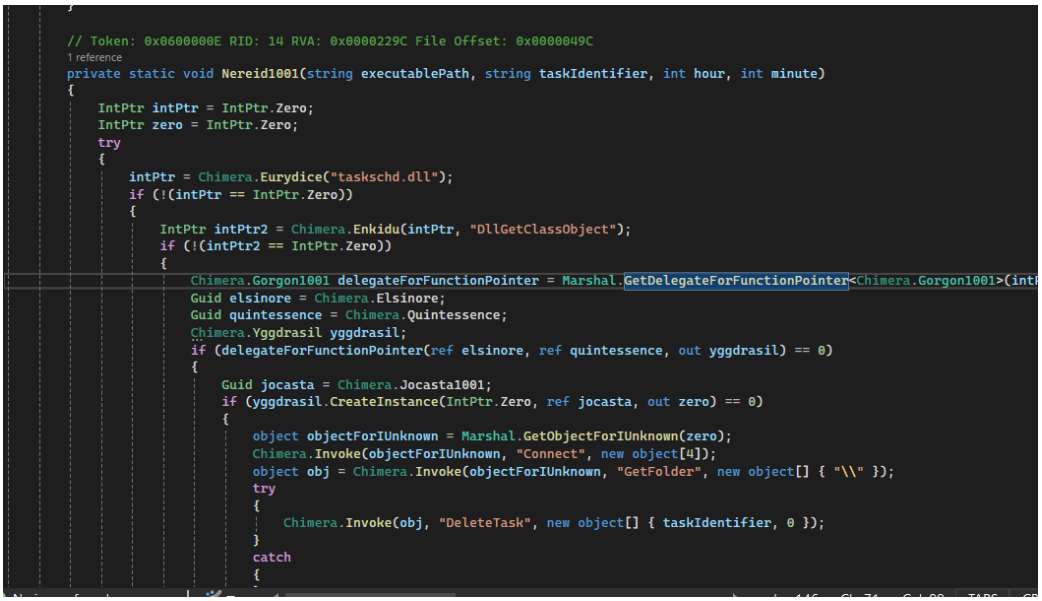


Figure 11. Triton function.

During execution, the previously supplied parameters create **RogLiveService.exe**, emonstrating a persistence technique in which malicious files are copied to **C:\ProgramData\USOPublic\RogLiveService.exe**. The **Triton** implementation also calls another function, **Nereid1001**.

During execution, the previously supplied parameters create **RogLiveService.exe**, demonstrating a persistence technique in which malicious files are copied to **C:\ProgramData\USOPublic\RogLiveService.exe**. The **Triton** implementation also calls another function, **Nereid1001**.



```
// Token: 0x0600000E RID: 14 RVA: 0x0000229C File Offset: 0x0000049C
1 reference
private static void Nereid1001(string executablePath, string taskIdentifier, int hour, int minute)
{
    IntPtr intPtr = IntPtr.Zero;
    IntPtr zero = IntPtr.Zero;
    try
    {
        intPtr = Chimera.Eurydice("taskschd.dll");
        if (!(intPtr == IntPtr.Zero))
        {
            IntPtr intPtr2 = Chimera.Enkidu(intPtr, "DllGetClassObject");
            if (!(intPtr2 == IntPtr.Zero))
            {
                Chimera.Gorgon1001 delegateForFunctionPointer = Marshal.GetDelegateForFunctionPointer<Chimera.Gorgon1001>(intP
                Guid elsinore = Chimera.Elsinore;
                Guid quintessence = Chimera.Quintessence;
                Chimera.Yggdrasil yggdrasil;
                if (delegateForFunctionPointer(ref elsinore, ref quintessence, out yggdrasil) == 0)
                {
                    Guid jocasta = Chimera.Jocasta1001;
                    if (yggdrasil.CreateInstance(IntPtr.Zero, ref jocasta, out zero) == 0)
                    {
                        object objectForIUnknown = Marshal.GetObjectForIUnknown(zero);
                        Chimera.Invoke(objectForIUnknown, "Connect", new object[4]);
                        object obj = Chimera.Invoke(objectForIUnknown, "GetFolder", new object[] { "\\" });
                        try
                        {
                            Chimera.Invoke(obj, "DeleteTask", new object[] { taskIdentifier, 0 });
                        }
                        catch
                        {
```

Figure 12. Nereid1001 function.

This function creates a Windows Scheduled Task directly through **taskschd.dll**. It accepts the location of an executable as a parameter, creates a task using the supplied name, schedules it to run every day at the specified time, and then starts it immediately.
Before creating the task, the function deletes any existing task with the same name. The task is registered for the current user, assigned the description **"Daily startup"**, enabled, and linked to the executable previously copied to **C:\ProgramData**.
In practical terms, the code installs a mechanism that automatically launches the program every day. Although this is a legitimate technique for software updates or services, it is also frequently used by malicious files to establish persistence.

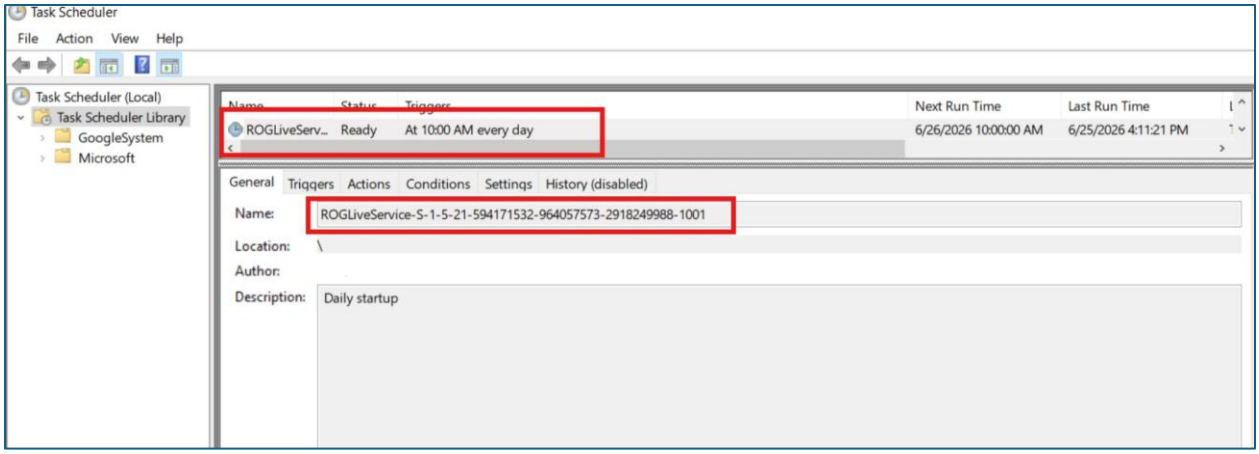


Figure 13. Scheduled Task created after execution of the executable.

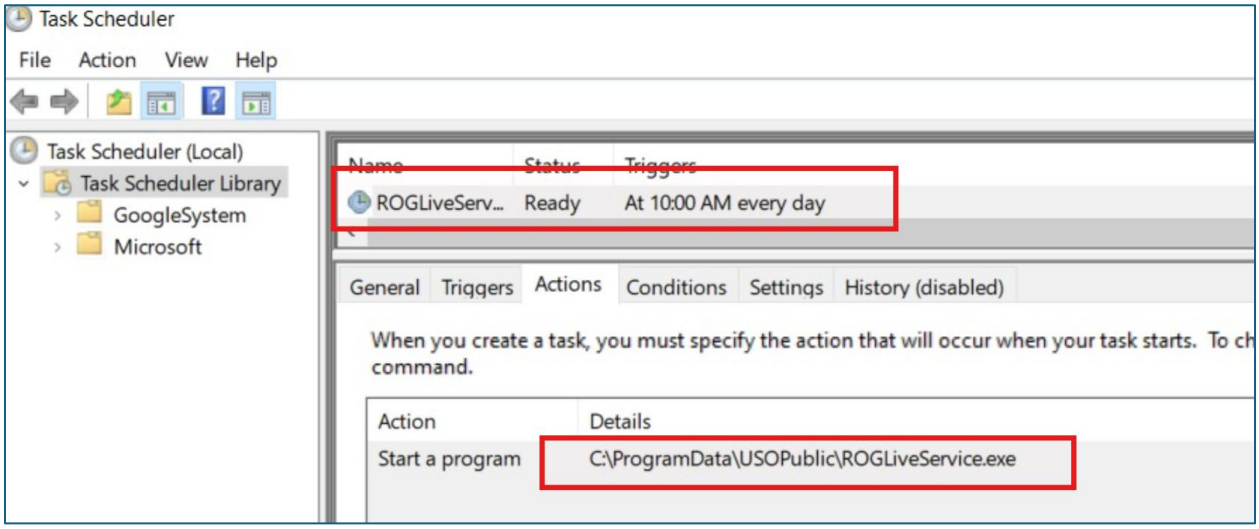


Figure 14. Scheduled Task process and the program launched according to schedule.

Debugging of the native DLL and the executable revealed communication with the domain **DietHealths[.]com** through the **RDX** register. Continuous **POST** requests are sent to this domain with parameters such as an **ID token**, and a validation process is performed to determine whether authentication has succeeded.

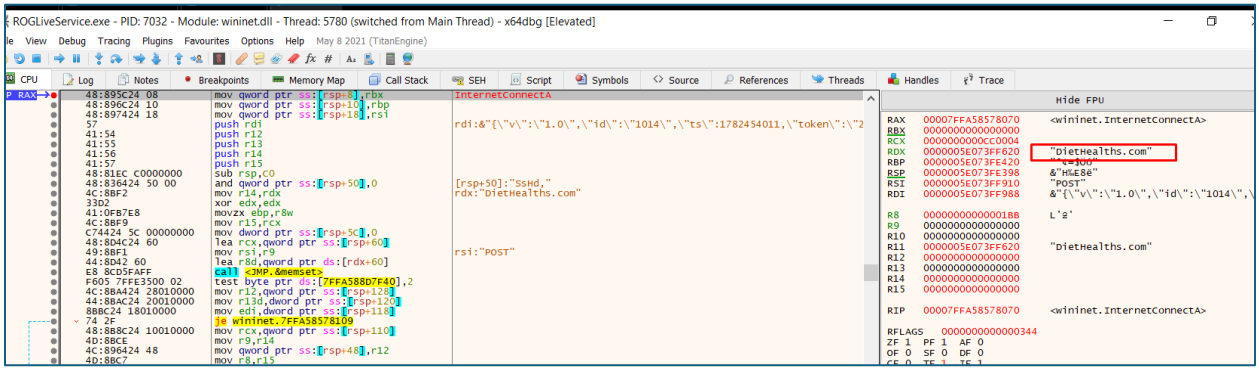


Figure 15. Identified active C2 domain.

Figure 16. Path used for the C2 POST request: /api/v2/validate.

Indicators of Compromise

OneDrive-latest-v3[.]zip	**9a778782dc1237f814b6a9e583c2b47203704d3d1d04d0cfd13d1499a3b31a49**
OneDrive-latest-v3.exe[.]config ROGLiveService.exe[.]config	**1103b254c247903ab1d39525e6066bf5a306f1faf8bcafb9ad39e04ec2635a29**
System.Collection[.]dll	**a29ea16355de86e5f58c993ceafdfa0dad0a607278986b94f5d73e5d6cc23dcd**
eio_x64[.]dll	**81aa2b88cbe5bd2e0567a341321cb2008d99a0d64294532f9bca6769ed775637**
Directory	**C:\ProgramData\USOPublic**
Domain	**diethealths[.]com**
URL	**https://sharepoint.epa[.]gov[.]gr/custom/**
Sender	**mnezeriti@mou.gr**

MITRE ATT&CK

Tactic MITRE ATT&CK	ID	Technique	Description / Observed Activity
Initial Access	T1566.001	Phishing: Spearphishing Attachment	Delivered a malicious ZIP archive containing sideloaded payloads through a document-themed spear-phishing campaign sent from a compromised email account.
Execution	T1204.002	User Execution: Malicious File	Relied on the victim to extract the archive and run the malicious executable.
Execution	T1574.001	Hijack Execution Flow: DLL	Employed DLL sideloading to execute malicious code through a legitimate signed binary.
Persistence	T1053.005	Scheduled Task/Job: Scheduled Task	Created a Windows Scheduled Task ("Daily startup") to establish persistence and automatically execute the malware daily at a predefined time.
Defense Evasion	T1036	Masquerading	Disguised malicious payloads as legitimate Microsoft OneDrive software components and updates.
Defense Evasion	T1027.015	Obfuscated Files or Information: Compression	Delivered the malicious payload within a compressed ZIP archive, and used obfuscated/encoded strings within the .NET code.

Recommendations

The National Cyber Security Authority recommends:

- Immediately block the Indicators of Compromise listed above on your security controls and protective devices.
- Review Windows Task Scheduler for new or unauthorized tasks created by this file.
- Inspect the following location: **C:\ProgramData\USOPublic**
- Provide continuous phishing-awareness training to non-technical staff, including guidance on how to avoid infection.
- Deploy network-perimeter devices capable of deep traffic inspection based not only on access-control rules, but also on behavioral analysis, such as next-generation firewalls.
- Implement solutions that filter, monitor, and block malicious traffic between web applications and the internet, such as a Web Application Firewall (WAF).
- Continuously analyze logs collected by the Security Information and Event Management (SIEM) platform.
- Segment the identified systems into separate VLANs and apply access-control lists across the network perimeter. Web services should be separated from their databases, and Active Directory should be placed in a dedicated VLAN.
- Implement and use Microsoft Local Administrator Password Solution (LAPS) to manage local administrator passwords.
- Apply traffic filters when hosts are accessed remotely by employees, third parties, or clients.
- Perform behavior-based traffic analysis for endpoints and deploy EDR/XDR solutions. This enables malicious files to be analyzed not only by signature, but also by behavior.
- Design and implement an Identity and Access Management (IAM) solution to control user identities and privileges in real time in accordance with the zero-trust principle.