REPUBLIC OF ALBANIA
NATIONAL CYBER SECURITY AUTHORITY
DIRECTORATE OF CYBERSECURITY ANALYSIS

Phishing campaign by Muddy Water
File Analysis - AteraAgent.exe

Version: 1.0
Date: 10.07.2024

TABLE OF CONTENTS

TABLE OF FIGURES

This report has limitations and should be interpreted with caution!

Some of these restrictions include:

The first phase: Sources of information: The report is based on information found at the time of its preparation. Meanwhile, some aspects may be different from current developments.

Second phase: Analysis details: Due to resource limitations, some aspects of the malignant details may not have been analyzed in depth. Any additional unknown information may reflect changes in the report.

The third phase: Information Security: To protect confidential resources and information, some details may be redacted or not included in the report. This decision was made to maintain the integrity and security of the data used.

National Cyber Security Authority (NCSA) reserves the right to change, update, or change any part of this report without prior notice.

This report is not a definitive document (extraction of additional details of the malicious actors will be made available to you at a later time).

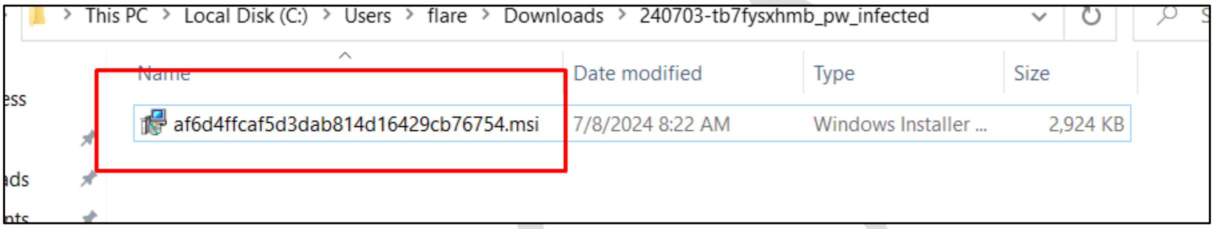
The findings of the report are based on the information available at the time of the investigation and analysis. There are no guarantees regarding possible changes or updates to the information reported during the following period. The authors of the report assume no responsibility for the misuse or consequences of any decision-making based on this report.

The report emphasizes the need for vigilance and proactive measures against sophisticated cyber threats, highlighting the importance of regular updates and the implementation of recommended security practices to protect critical infrastructure.

Technical Information

A Phishing campaign by malicious actors has been identified, wherein they exploit a legitimate application by modifying its source code to execute malicious actions on infected computers and systems. The analysis of the Atera file connects these activities to recent Phishing attacks associated with the Iranian group, **MuddyWater.**

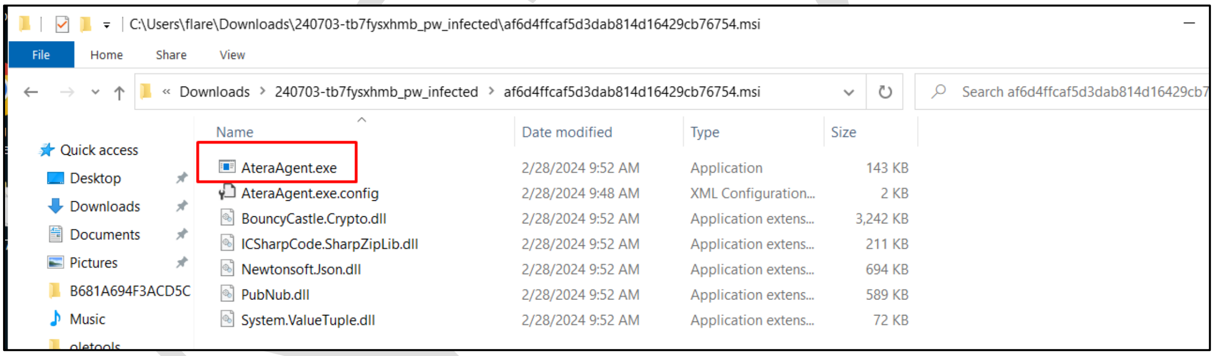
The file with **HASH** value **sha256**: **55AF6A90AC8863F27B3FCAA416A0F1E4FF02FB42AA46A7274C6B76AA000AACC2** is a file in **.msi format (Microsoft Windows Installer)** which is run by the user himself and installed on his computer. For such file formats we change the file extension from **.msi** and add the suffix **.7z** and try to extract it



This PC > Local Disk (C:) > Users > flare > Downloads > 240703-tb7fysxhmb_pw_infected

Name	Date modified	Type	Size
af6d4ffcaf5d3dab814d16429cb76754.msi	7/8/2024 8:22 AM	Windows Installer ...	2,924 KB

Figure 1: File.msi



C:\Users\flare\Downloads\240703-tb7fysxhmb_pw_infected\af6d4ffcaf5d3dab814d16429cb76754.msi

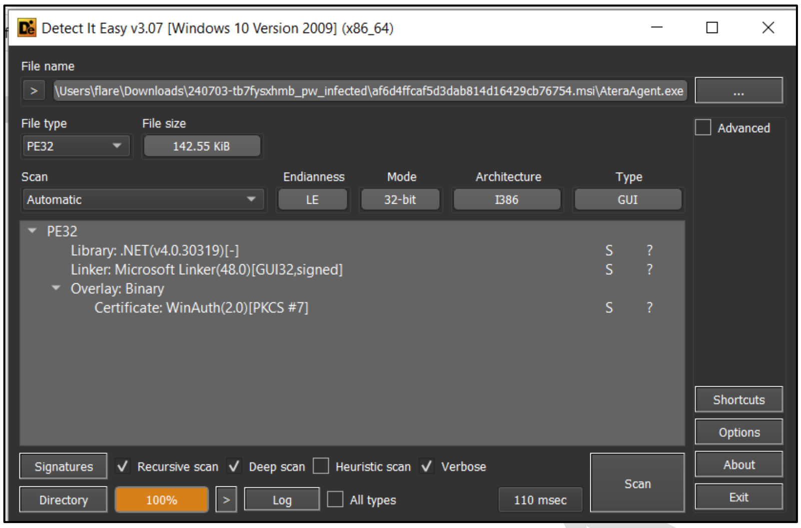
« Downloads > 240703-tb7fysxhmb_pw_infected > af6d4ffcaf5d3dab814d16429cb76754.msi

Search af6d4ffcaf5d3dab814d16429cb7

Name	Date modified	Type	Size
AteraAgent.exe	2/28/2024 9:52 AM	Application	143 KB
AteraAgent.exe.config	2/28/2024 9:48 AM	XML Configuration...	2 KB
BouncyCastle.Crypto.dll	2/28/2024 9:52 AM	Application extens...	3,242 KB
ICSharpCode.SharpZipLib.dll	2/28/2024 9:52 AM	Application extens...	211 KB
Newtonsoft.Json.dll	2/28/2024 9:52 AM	Application extens...	694 KB
PubNub.dll	2/28/2024 9:52 AM	Application extens...	589 KB
System.ValueTuple.dll	2/28/2024 9:52 AM	Application extens...	72 KB

Figure 2: AteraAgent.exe

From the extraction stage, it turns out that we have several sub-files where **AteraAgent.exe** is identified. From the name itself it is understood that we are dealing with a **RAT (Remote Access Trojan).** During the analysis, it is evident that the file is created in **.NET** and with **C#** programming language.



Detect It Easy v3.07 [Windows 10 Version 2009] (x86_64)

File name: \Users\flare\Downloads\240703-tb7fysxhmb_pw_infected\af6d4ffcaf5d3dab814d16429cb76754.msi\AteraAgent.exe

File type: PE32 File size: 142.55 KiB

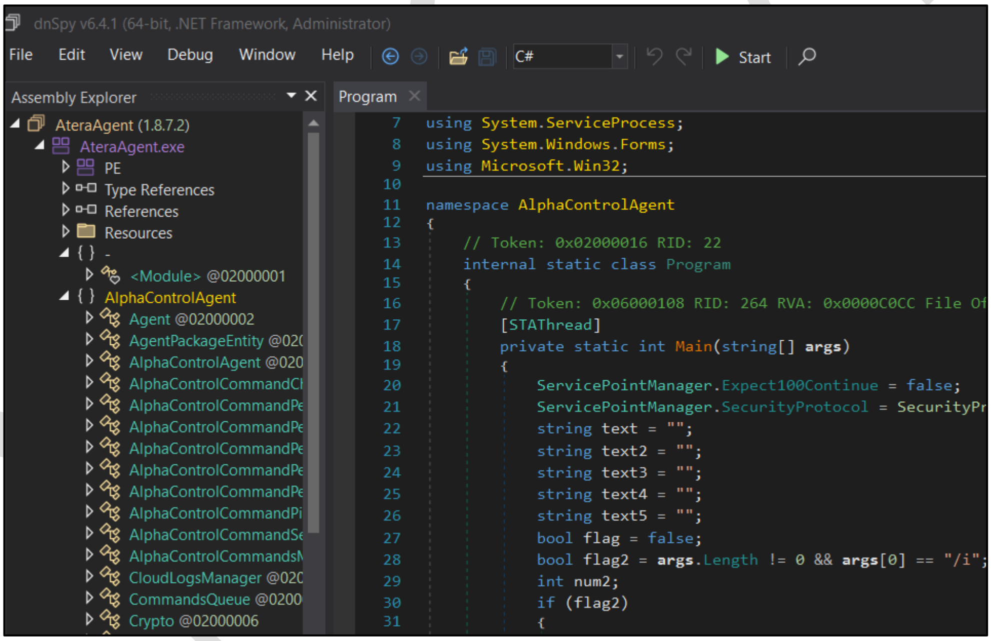
Scan: Automatic Endianness: LE Mode: 32-bit Architecture: I386 Type: GUI

```
PE32
    Library: .NET(v4.0.30319)[-]                    S    ?
    Linker: Microsoft Linker(48.0)[GUI32,signed]    S    ?
    Overlay: Binary
        Certificate: WinAuth(2.0)[PKCS #7]          S    ?
```

☐ Advanced ✓ Recursive scan ✓ Deep scan ☐ Heuristic scan ✓ Verbose ☐ All types 100% 110 msec

Figure 3: .NET framework.

We try to import this file as a project and it is evident that this RAT offers the most diverse functionalities.

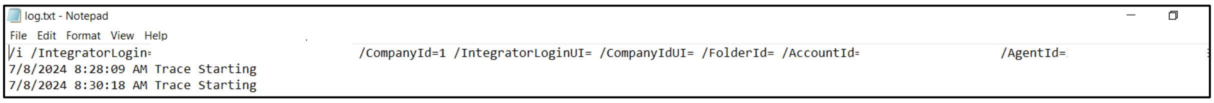


```
using System.ServiceProcess;
using System.Windows.Forms;
using Microsoft.Win32;

namespace AlphaControlAgent
{
    // Token: 0x02000016 RID: 22
    internal static class Program
    {
        // Token: 0x06000108 RID: 264 RVA: 0x0000C0CC File Of
        [STAThread]
        private static int Main(string[] args)
        {
            ServicePointManager.Expect100Continue = false;
            ServicePointManager.SecurityProtocol = SecurityPr
            string text = "";
            string text2 = "";
            string text3 = "";
            string text4 = "";
            string text5 = "";
            bool flag = false;
            bool flag2 = args.Length != 0 && args[0] == "/i";
            int num2;
            if (flag2)
            {
```

Figure 4: Functions of AteraAgent.exe

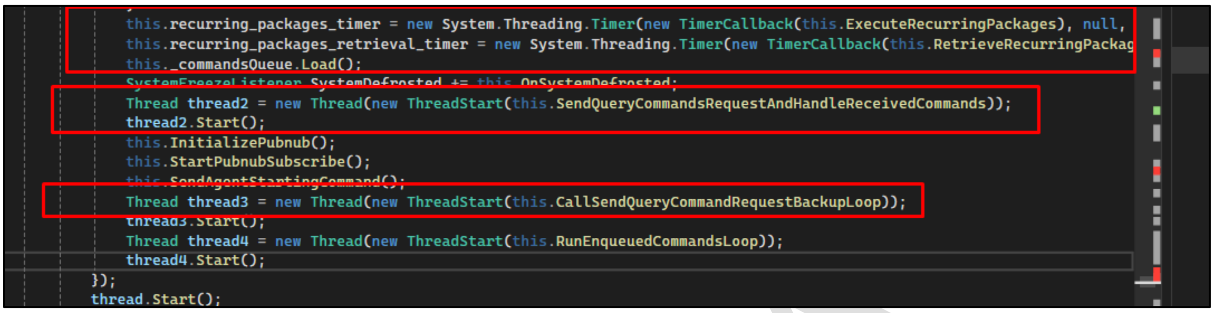
The **main()** function is the main function that starts in the Program.**cs** class. That's where the main implementation starts and the initialization of the **Agent** class which seems to use the **Singleton Design Pattern**. In the main function, the configuration part starts from where it reads as parameters the arguments passed to the agent during execution, this is done in order to make the connection with the remote *command and control server* (C2). This is done through a compromised email, agent parameter id, etc.



```
/i /IntegratorLogin=                    /CompanyId=1 /IntegratorLoginUI= /CompanyIdUI= /FolderId= /AccountId=                    /AgentId=
7/8/2024 8:28:09 AM Trace Starting
7/8/2024 8:30:18 AM Trace Starting
```

Figure 5: Parameters that are passed as arguments

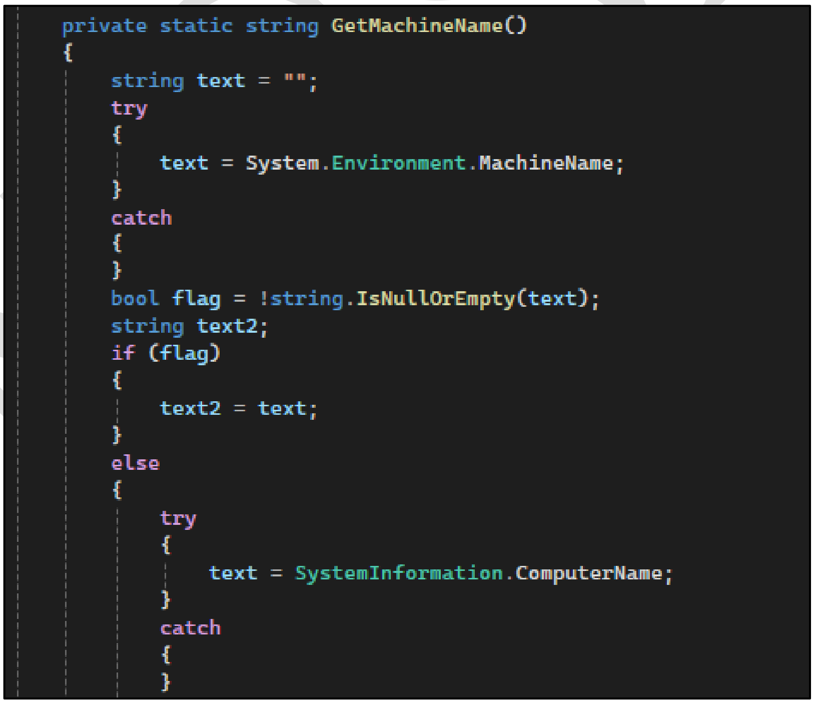
In the **Agent.cs** class, the function **SendQueryCommandsRequestAndHandleReceivedCommands** is highlighted, which is passed as a parameter to the constructor of the new ThreadStart class. If we check the subfunctions of this function, the call of the **SendQueryCommandsRequest** function is evidenced, which has implemented the logic of receiving and sending Remote commands.



```
            this.recurring_packages_timer = new System.Threading.Timer(new TimerCallback(this.ExecuteRecurringPackages), null,
            this.recurring_packages_retrieval_timer = new System.Threading.Timer(new TimerCallback(this.RetrieveRecurringPackag
            this._commandsQueue.Load();
            SystemFreezeListener.SystemDefrosted += this.OnSystemDefrosted;
            Thread thread2 = new Thread(new ThreadStart(this.SendQueryCommandsRequestAndHandleReceivedCommands));
            thread2.Start();
            this.InitializePubnub();
            this.StartPubnubSubscribe();
            this.SendAgentStartingCommand();
            Thread thread3 = new Thread(new ThreadStart(this.CallSendQueryCommandRequestBackupLoop));
            thread3.Start();
            Thread thread4 = new Thread(new ThreadStart(this.RunEnqueuedCommandsLoop));
            thread4.Start();
        });
        thread.Start();
```

Figure 6: Function SendQuerycommandRequests

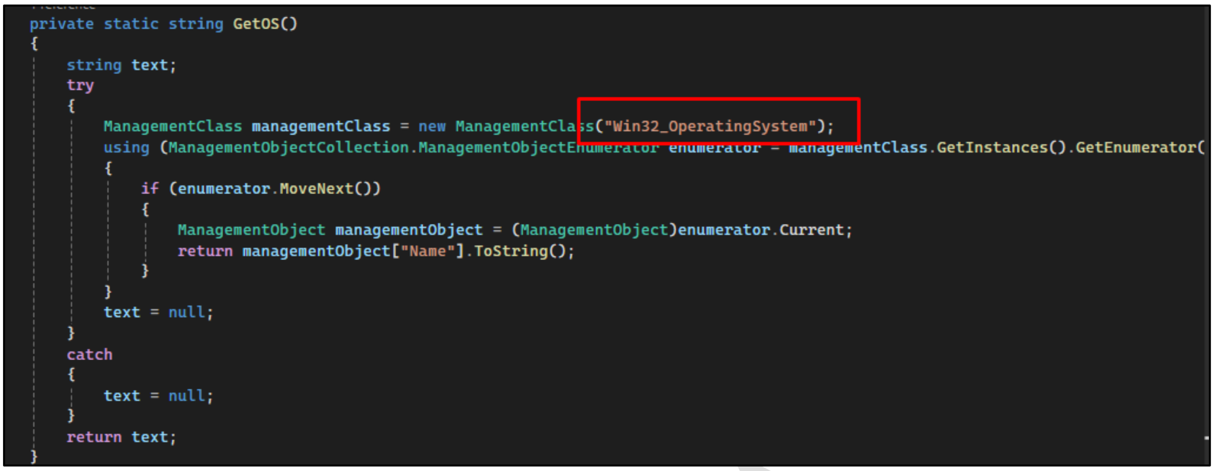
During the analysis, the functionality of obtaining information on the compromised computer through the **GetMachineName()** function is evidenced. If we look at the implementation of the function it is done through **System.Environment.MachineName.**



```
private static string GetMachineName()
{
    string text = "";
    try
    {
        text = System.Environment.MachineName;
    }
    catch
    {
    }
    bool flag = !string.IsNullOrEmpty(text);
    string text2;
    if (flag)
    {
        text2 = text;
    }
    else
    {
        try
        {
            text = SystemInformation.ComputerName;
        }
        catch
        {
        }
```

Figure 7: Function GetMachineName()

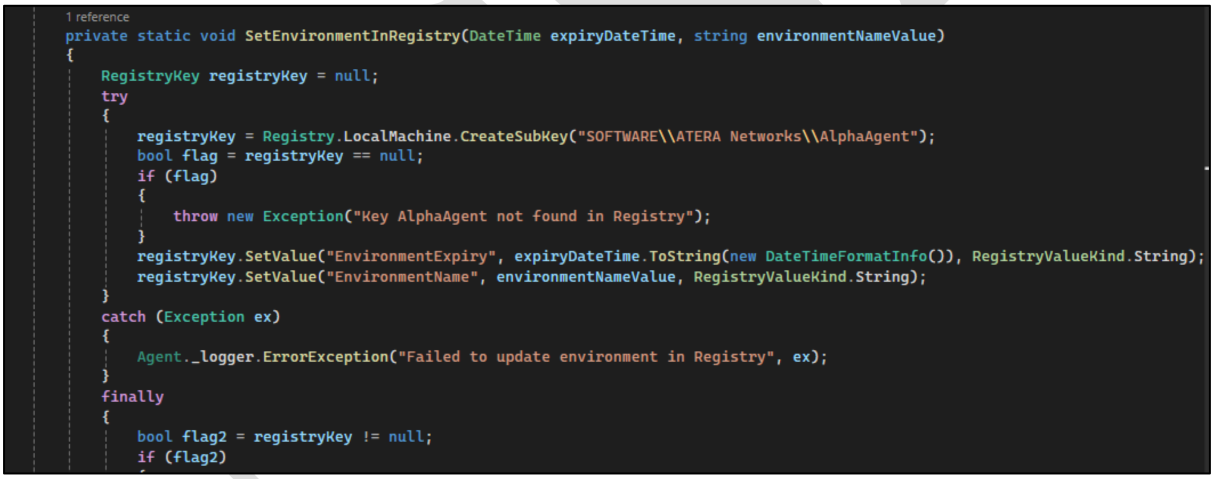
Getting information about the operating system is done through the **GetOS()** function, where the **ManagementClass** class is used and the **"Win32_OperatingSystem"** string is passed as a parameter.



```
private static string GetOS()
{
    string text;
    try
    {
        ManagementClass managementClass = new ManagementClass("Win32_OperatingSystem");
        using (ManagementObjectCollection.ManagementObjectEnumerator enumerator = managementClass.GetInstances().GetEnumerator(
        {
            if (enumerator.MoveNext())
            {
                ManagementObject managementObject = (ManagementObject)enumerator.Current;
                return managementObject["Name"].ToString();
            }
        }
        text = null;
    }
    catch
    {
        text = null;
    }
    return text;
}
```

Figure 8: Function GetOS()

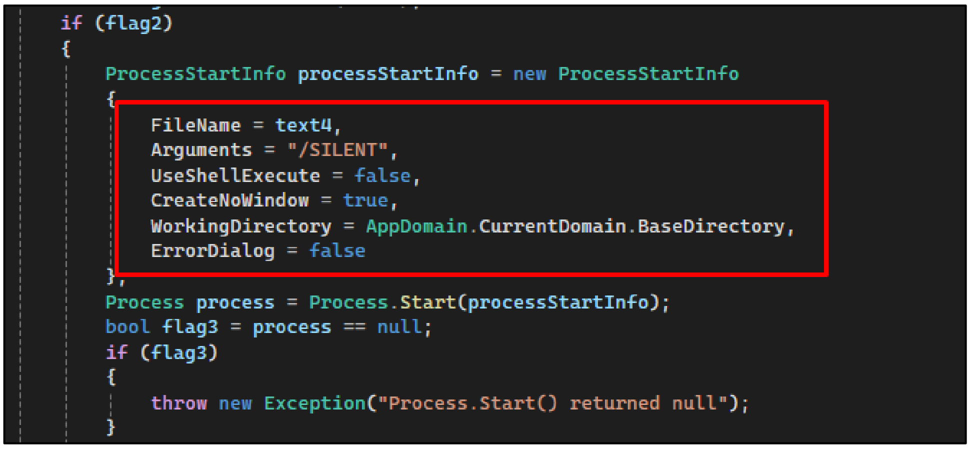
During the analysis of the code, the **SetEnvironmentInRegistry** function is identified, during the installation of the agent, a subfolder with the path **"SOFTWARE\ATERANETWORKS\AlphaAgent"** is created. This is done in order to achieve the persistence of the malicious file.



```
    1 reference
    private static void SetEnvironmentInRegistry(DateTime expiryDateTime, string environmentNameValue)
    {
        RegistryKey registryKey = null;
        try
        {
            registryKey = Registry.LocalMachine.CreateSubKey("SOFTWARE\\ATERA Networks\\AlphaAgent");
            bool flag = registryKey == null;
            if (flag)
            {
                throw new Exception("Key AlphaAgent not found in Registry");
            }
            registryKey.SetValue("EnvironmentExpiry", expiryDateTime.ToString(new DateTimeFormatInfo()), RegistryValueKind.String);
            registryKey.SetValue("EnvironmentName", environmentNameValue, RegistryValueKind.String);
        }
        catch (Exception ex)
        {
            Agent._logger.ErrorException("Failed to update environment in Registry", ex);
        }
        finally
        {
            bool flag2 = registryKey != null;
            if (flag2)
```

Figure 9: Function SetEnvironmentInRegistry

We also have a control which tries to execute a **silent** process where it is realized through the **ProcessStartInfo** class and in the construction part it receives the relevant parameters.



```
if (flag2)
{
    ProcessStartInfo processStartInfo = new ProcessStartInfo
    {
        FileName = text4,
        Arguments = "/SILENT",
        UseShellExecute = false,
        CreateNoWindow = true,
        WorkingDirectory = AppDomain.CurrentDomain.BaseDirectory,
        ErrorDialog = false
    };
    Process process = Process.Start(processStartInfo);
    bool flag3 = process == null;
    if (flag3)
    {
        throw new Exception("Process.Start() returned null");
    }
```

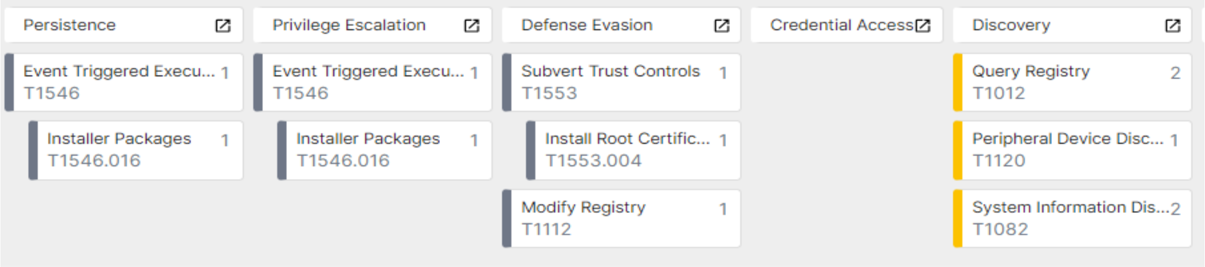
Figure 10: Creating a silent process

Indicators of compromise

HASH

55af6a90ac8863f27b3fcaa416a0f1e4ff02fb42aa46a7274c6b76aa000aacc2
8fbd374d4659efdc5b5a57ff4168236aeaab6dae4af6b92d99ac28e05f04e5c1
c152b0c74d704054e2962e2a6198195dc96b4de92f97d8243519e7dcbfca4bd3

Techniques of MITRE ATT&CK



Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery
Event Triggered Execu... 1 T1546	Event Triggered Execu... 1 T1546	Subvert Trust Controls 1 T1553		Query Registry 2 T1012
Installer Packages 1 T1546.016	Installer Packages 1 T1546.016	Install Root Certific... 1 T1553.004		Peripheral Device Disc... 1 T1120
		Modify Registry 1 T1112		System Information Dis...2 T1082

Recommendations

NCSA recommends:

- Immediate blocking of the above-mentioned Indicators of Compromise in your security devices.
- Continuous analysis of logs coming from SIEM (Security information and Event Management).
- Training of non-technical staff about "Phishing" attacks and ways to avoid being infected by them.
- Installation of network perimeter devices that perform deep traffic analysis based not only on access list rules but also on its behavior (NextGen Firewalls).
- The identified systems should be segmented into different VLANs, applying "Access

control list for the entire perimeter of the network", webservices should be separated from their Database, Active Directory should be in a separate VLAN.

- Application and use of the LAPS technique for Microsoft systems, for managing passwords of Local Administrators.
- Apply traffic filters in the case of remote access to hosts (employees/third parties/customers).
- Implement solutions that filter, monitor and block malicious traffic between Web applications and the Internet, Web Application Firewall (WAF).
- Conduct traffic analysis at the "behaviour" level for end devices, application of EDR, XDR solutions. This brings the analysis of malicious files not only at the signature level but also at the behavior level.
- To design the "Identity Access Management" user access management solution to control the identity and privileges of users in real time according to the "zero-trust" principle.