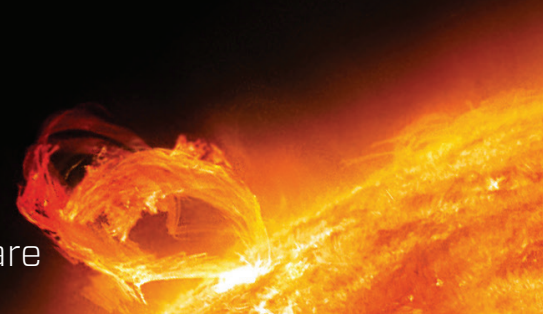
PREVASIO

Sunburst Backdoor

A deeper look into The SolarWinds' Supply Chain Malware



By Sergei Shevchenko
December 15, 2020

Summary

As earlier reported[1] by FireEye, the attackers have orchestrated a supply chain attack by trojanizing SolarWinds Orion business software updates to distribute malware.

The attackers have leveraged multiple techniques to evade detection and to obscure their activity.

The initial aim of this research was to uncover their operation. During a detailed analysis and reverse engineering of the trojanised SolarWinds Orion business software, Prevasio has uncovered a wide range of public and private organizations around the world that may have fallen the victims of this attack.

Prevasio is the first company in the world that has fully reconstructed the list of victims of this attack.

Prevasio has decided to publish this list to raise the level of awareness of this attack, and to provide administrators of the targeted companies with IOCs to help them to locate and to eradicate an infection that may have potentially affected their corporate infrastructure.

SUNBURST Backdoor

At the time of writing, the malicious update `SolarWinds-Core-v2019.4.5220-Hotfix5.msp` is still available for download.

The malicious component `SolarWinds.Orion.Core.BusinessLayer.dll` inside the MSP package is a non-obfuscated .NET assembly. It can easily be reconstructed with a .NET disassembler, such as ILSpy, and then fully reproduced in C# code, using Microsoft Visual Studio. Once reproduced, it can be debugged to better understand how it works.

In a nutshell, the malicious DLL is a backdoor. It is loaded into the address space of the legitimate SolarWinds Orion process `SolarWinds.BusinessLayerHost.exe` or `SolarWinds.BusinessLayerHostx64.exe`.

The critical strings inside the backdoor's class `SolarWinds.Orion.Core.BusinessLayer.OrionImprovementBusinessLayer` are encoded with the *DeflateStream* Class of the .NET's *System.IO.Compression* library, coupled with the standard base64 encoder.

1 https://www.fireeye.com/blog/threat-research/2020/12/evasive-attacker-leverages-solarwinds-supply-chain-compromises-with-sunburst-backdoor.html

Initialisation

Once loaded, the malware checks if its assembly file was created earlier than 12, 13, or 14 days ago. The exact number of hours it checks is a random number from 288 to 336.

Next, it reads the application settings value `ReportWatcherRetry`. This value keeps the reporting status, and may be set to one of the states:

- New (4)
- Truncate (3)
- Append (5)

When the malware runs the first time, its reporting status variable `ReportWatcherRetry` is set to `New (4)`.

The reporting status is an internal state that drives the logic. For example, if the reporting status is set to `Truncate`, the malware will stop operating by first disabling its networking communications, and then disabling other security tools and antivirus products.

In order to stay silent, the malware periodically falls asleep for a random period of time that varies between 30 minutes and 2 hours.

At the start, the malware obtains the computer's domain name. If the domain name is empty, the malware quits.

It then generates a 8-byte User ID, which is derived from the system footprint. In particular, it is generated from MD5 hash of a string that consists from the 3 fields:

- the first or default operational (can transmit data packets) network interface's physical address
- computer's domain name
- UUID created by Windows during installation (machine's unique ID)

Even though it looks random, the User ID stays permanent as long as networking configuration and the Windows installation stay the same.

Domain Generation Algorithm (Part I)

The malware relies on its own CryptoHelper class to generate a domain name. This class is instantiated from the 8-byte User ID and the computer's domain name, encoded with a substitution table: **"`rq3gsalt6u1iyfzop572d49bnx8cvmkewhj`"**.

For example, if the original domain name is *"domain"*, its encoded form will look like: *"n2huov"*.

To generate a new domain, the malware first attempts to resolve domain name *"api.solarwinds.com"*. If it fails to resolve it, it quits.

The first part of the newly generated domain name is a random string, produced from the 8-byte User ID, a random seed value, and encoded with a custom base64 alphabet **"`ph2eifo3n5utg1j8d94qrvbmk0sal76c`"**.

Because it is generated from a random seed value, the first part of the newly generated domain name is random.

For example, it may look like "*fivu4vjamve5vfrt*" or "*k1sdhtslulgqoagy*".

To produce the domain name, this string is then appended with the earlier encoded domain name (such as "*n2huov*") and a random string, selected from the following list:

- `.appsync-api.eu-west-1[.]avsvmcloud[.]com`
- `.appsync-api.us-west-2[.]avsvmcloud[.]com`
- `.appsync-api.us-east-1[.]avsvmcloud[.]com`
- `.appsync-api.us-east-2[.]avsvmcloud[.]com`

For example, the final domain name may look like:

```
fivu4vjamve5vfrtn2huov[.]appsync-api.us-west-2[.]avsvmcloud[.]com
```

or

```
k1sdhtslulgqoagyn2huov[.]appsync-api.us-east-1[.]avsvmcloud[.]com
```

Next, the domain name is resolved to an IP address, or to a list of IP addresses. For example, it may resolve to `20.140.0.1`.

The resolved domain name will be returned into `IPAddress` structure that will contain an `AddressFamily`[1] field - a special field that specifies the addressing scheme.

If the host name returned in the `IPAddress` structure is different to the queried domain name, the returned host name will be used as a C2 host name for the backdoor.

Otherwise, the malware will check if the resolved IP address matches one of the patterns below, in order to return an 'address family':

IP Address	Subnet Mask	'Address Family'
10.0.0.0	255.0.0.0	Atm
172.16.0.0	255.240.0.0	Atm
192.168.0.0	255.255.0.0	Atm
224.0.0.0	240.0.0.0	Atm
fc00::	fe00::	Atm
fec0::	ffc0::	Atm
ff00::	ff00::	Atm
41.84.159.0	255.255.255.0	Ipx
74.114.24.0	255.255.248.0	Ipx
154.118.140.0	255.255.255.0	Ipx
217.163.7.0	255.255.255.0	Ipx

IP Address	Subnet Mask	'Address Family'
20.140.0.0	255.254.0.0	ImpLink
96.31.172.0	255.255.255.0	ImpLink
131.228.12.0	255.255.252.0	ImpLink
144.86.226.0	255.255.255.0	ImpLink
8.18.144.0	255.255.254.0	NetBios
18.130.0.0	255.255.0.0	NetBios
71.152.53.0	255.255.255.0	NetBios
99.79.0.0	255.255.0.0	NetBios
87.238.80.0	255.255.248.0	NetBios
199.201.117.0	255.255.255.0	NetBios
184.72.0.0	255.254.0.0	NetBios

1 https://docs.microsoft.com/en-us/dotnet/api/system.net.sockets.addressfamily?view=net-5.0

For example, if the queried domain resolves to `20.140.0.1`, it will match the entry in the table `20.140.0.0`, for which the returned 'address family' will be `ImpLink`.

The returned 'address family' invokes an additional logic in the malware.

Disabling Security Tools and Antivirus Products

If the returned 'address family' is `ImpLink` or `Atm`, the malware will enumerate all processes and for each process, it will check if its name matches one of the pre-defined hashes.

Next, it repeats this processed for services and for the drivers installed in the system. If a process name or a full path of an installed driver matches one of the pre-defined hashes, the malware will disable it.

For hashing, the malware relies on Fowler–Noll–Vo[1] algorithm.

For example, the core process of Windows Defender is `MsMpEng.exe`. The hash value of *"MsMpEng"* string is `5183687599225757871`. This value is specifically enlisted the malware's source under a variable name timeStamps:

```
timeStamps = new ulong[1]
{
    518368759922575787 1uL
}
```

The service name of Windows Defender is `windefend` - the hash of this string (`917638920165491138`) is also present[2] in the malware body. As a result, the malicioius DLL will attempt to stop the Windows Defender service.

In order to disable various security tools and antivirus products, the malware first grants itself `SeRestorePrivilege` and `SeTakeOwnershipPrivilege` privileges, using the native `AdjustTokenPrivileges()` API. With these privileges enabled, the malware takes ownership of the service registry keys it intends to manipulate.

The new owner of the keys is first attempted to be explicitly set to Administrator account. If such account is not present, the malware enumerates all user accounts, looking for a SID that represents the administrator account.

The malware uses Windows Management Instrumentation query *"Select * From Win32_UserAccount"* to obtain the list of all users. For each enumerated user, it makes sure the account is local and then, when it obtains its SID, it makes sure the SID begins with `S-1-5-` and ends with `-500` in order to locate[3] the local administrator account.

Once such account is found, it is used as a new owner for the registry keys, responsible for manipulation of the services of various security tools and antivirus products.

---

1 https://en.wikipedia.org/wiki/Fowler%E2%80%93Noll%E2%80%93Vo_hash_function
2 https://github.com/fireeye/sunburst_countermeasures/blob/main/fnv1a_xor_hashes.txt
3 https://devblogs.microsoft.com/scripting/how-can-i-determine-if-the-local-administrator-account-has-been-renamed-on-a-computer/

With the new ownership set, the malware then disables[1] these services by setting their `Start` value to `4` (Disabled):

```
registryKey2.SetValue("Start"), 4, RegistryValueKind.DWord);
```

HTTP Backdoor

If the returned 'address family' for the resolved domain name is `NetBios`, as specified in the lookup table above, the malware will initialise its `HttpHelper` class, which implements an HTTP backdoor.

The backdoor commands are covered in the FireEye write-up, so let's check only a couple of commands to see what output they produce.

One of the backdoor commands is `CollectSystemDescription`. As its name suggests, it collects system information. By running the code reconstructed from the malware, here is an actual example of the data collected by the backdoor and delivered to the attacker's C2 with a separate backdoor command `UploadSystemDescription`:

```
1. %DOMAIN_NAME%
2. S-1-5-21-298510922-2159258926-905146427
3. DESKTOP-VL39FPO
4. UserName
5. [E] Microsoft Windows NT 6.2.9200.0 6.2.9200.0 64
6. C:\WINDOWS\system32
7. 0
8. %PROXY_SERVER%

Description: Killer Wireless-n/a/ac 1535 Wireless Network Adapter #2
MACAddress: 9C:B6:D0:F6:FF:5D
DHCPEnabled: True
DHCPServer: 192.168.20.1
DNSHostName: DESKTOP-VL39FPO
DNSDomainSuffixSearchOrder: Home
DNSServerSearchOrder: 8.8.8.8, 192.168.20.1
IPAddress: 192.168.20.30, fe80::8412:d7a8:57b9:5886
IPSubnet: 255.255.255.0, 64
DefaultIPGateway: 192.168.20.1, fe80::1af1:45ff:feec:a8eb
```

NOTE: Field #7 specifies the number of days (0) since the last system reboot.

`GetProcessByDescription` command will build a list of processes running on a system. This command accepts an optional argument, which is one of the custom process properties enlisted here[2].

If the optional argument is not specified, the backdoor builds a process list that looks like:

1 http://smallvoid.com/article/winnt-services-regedit.html
2 https://weblogs.asp.net/dixin/query-operating-system-processes-in-c


```
[ 1720] svchost
[ 8184] chrome
[ 4732] svchost
```

If the optional argument is specified, the backdoor builds a process list that includes the specified process property in addition to parent process ID, username and domain for the process owner.

For example, if the optional argument is specified as "*ExecutablePath*", the `GetProcessByDescription` command may return a list similar to:

```
[ 3656] sihost.exe       C:\WINDOWS\system32\sihost.exe      1720 DESKTOP-VL39FPO\UserName
[ 3824] svchost.exe      C:\WINDOWS\system32\svchost.exe       992 DESKTOP-VL39FPO\UserName
[ 9428] chrome.exe       C:\Program Files (x86)\Google\Chrome\Application\chrome.exe      4600
DESKTOP-VL39FPO\UserName
```

Other backdoor commands enable deployment of the 2nd stage malware. For example, the `WriteFile` command will save the file:

```
using (FileStream fileStream = new FileStream(path, FileMode.Append, FileAccess.Write))
{
    fileStream.Write(array, 0, array.Length);
}
```

The downloaded 2nd stage malware can then the executed with `RunTask` command:

```
using (Process process = new Process())
{
    process.StartInfo = new ProcessStartInfo(fileName, arguments)
    {
        CreateNoWindow = false,
        UseShellExecute = false
    };
    if (process.Start())
    ...
```

Alternatively, it can be configured to be executed with the system restart, using registry manipulation commands, such as `SetRegistryValue`.

Domain Generation Algorithm (Part II)

As described in the first part of our analysis, the DGA (Domain Generation Algorithm) of the Sunburst backdoor produces a domain name that may look like:

```
fivu4vjamve5vfrtn2huov[.]appsync-api.us-west-2[.]avsvmcloud[.]com
```


The first part of the domain name (before the first dot) consists of a 16-character random string, appended with an encoded computer's domain name. This is the domain[1] in which the local computer is registered.

From the example string above, we can conclude that the encoded computer's domain starts from the 17th character and up until the dot (highlighted in yellow):

```
fivu4vjamve5vfrtn2huov
```

In order to encode a local computer's domain name, the malware uses one of 2 simple methods:

- *Method 1*: a substitution table, if the domain name consists of small letters, digits, or special characters '-', '_', '.'
- *Method 2*: base64 with a custom alphabet, in case of capital letters present in the domain name

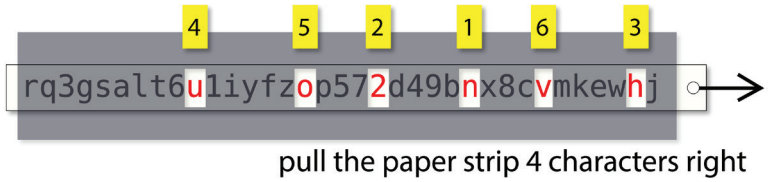
Method 1

In our example, the encoded domain name is **"n2huov"**. As it does not have any capital letters, the malware encodes it with a substitution table **"rq3gsalt6u1iyfzop572d49bnx8cvmkewhj"**.

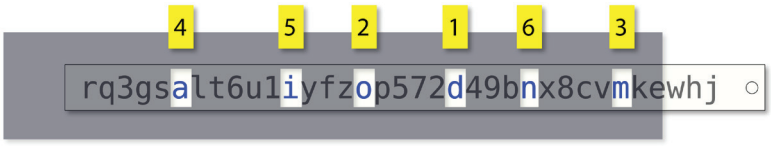
For each character in the domain name, the encoder replaces it with a character located in the substitution table four characters right from the original character.

In order to decode the name back, all we have to do is to replace each encoded character with another character, located in the substitution table four characters left from the original character.

To illustrate this method, imagine that the original substitution table is printed on a paper strip and then covered with a card with 6 perforated windows. Above each window, there is a sticker note with a number on it, to reflect the order of characters in the word **"n2huov"**, where **'n'** is #1, **'2'** is #2, **'h'** is #3 and so on:



Once the paper strip is pulled by 4 characters right, the perforated windows will reveal a different word underneath the card: **"domain"**, where **'d'** is #1, **'o'** is #2, **'m'** is #3, etc.:



1 https://docs.microsoft.com/en-us/dotnet/api/system.net.networkinformation.ipglobalproperties.domainname?redirectedfrom=MSDN&view=net-5.0#System_Net_NetworkInformation_IPGlobalProperties_DomainName

A special case is reserved for such characters as **‘0’**, **‘-’**, **‘_’**, **‘.’**. These characters are encoded with **‘0’**, followed with a character from the substitution table. An index of that character in the substitution table, divided by 4, provides an index within the string **“0_-.”**.

The following snippet in C# illustrates how an encoded string can be decoded:

```
static string decode_domain(string s)
{
    string table = “rq3gsalt6u1iyfzop572d49bnx8cvmkewhj”;
    string result = “”;
    for (int i = 0; i < s.Length; i++)
    {
        if (s[i] != ‘0’)
        {
            result += table[(table.IndexOf(s[i]) + table.Length - 4) % table.Length];
        }
        else
        {
            if (i < s.Length - 1)
            {
                if (table.Contains(s[i + 1]))
                {
                    result += “0_-.”[table.IndexOf(s[i + 1]) % 4];
                }
                else
                {
                    break;
                }
            }
            i++;
        }
    }
```

Method 2

This method is a standard[1] base64 encoder with a custom alphabet **“ph2eifo3n5utg1j8d94qrvbmk0sal76c”**.

Here is a snippet in C# that provides a decoder:

```
public static string FromBase32String(string str)
{
    string table = "ph2eifo3n5utg1j8d94qrvbmk0sal76c";
    int numBytes = str.Length * 5 / 8;
    byte[] bytes = new Byte[numBytes];
```

1 https://stackoverflow.com/questions/641361/base32-decoding


```
    int bit_buffer;
    int currentCharIndex;
    int bits_in_buffer;
    if (str.Length < 3)
    {
        bytes[0] = (byte)(table.IndexOf(str[0]) | table.IndexOf(str[1]) << 5);
        return  System.Text.Encoding.UTF8.GetString(bytes);
    }

    bit_buffer = (table.IndexOf(str[0]) | table.IndexOf(str[1]) << 5);
    bits_in_buffer = 10;
    currentCharIndex = 2;
    for (int i = 0; i < bytes.Length; i++)
    {
        bytes[i] = (byte)bit_buffer;
        bit_buffer >>= 8;
        bits_in_buffer -= 8;
        while (bits_in_buffer < 8 && currentCharIndex < str.Length)
        {
            bit_buffer |= table.IndexOf(str[currentCharIndex++]) << bits_in_buffer;
            bits_in_buffer += 5;
        }
    }
    return System.Text.Encoding.UTF8.GetString(bytes);
}
```

When the malware encodes a domain using Method 2, it prepends the encrypted string with a double zero character: **"00"**.

Following that, extracting a domain part of an encoded domain name (long form) is as simple as:

```
static string get_domain_part(string s)
{
    int i = s.IndexOf(".appsync-api");
    if (i > 0)
    {
        s = s.Substring(0, i);
        if (s.Length > 16)
        {
            return s.Substring(16);
        }
    }
    return "";
}
```


Once the domain part is extracted, the decoded domain name can be obtained by using Method 1 or Method 2, as explained above:

```
if (domain.StartsWith("00"))
{
    decoded = FromBase32String(domain.Substring(2));
}
else
{
    decoded = decode_domain(domain);
}
```

Decrypting the Victims' Domain Names

To see the decoder in action, let's select 2 lists:

List #1

Bambenek Consulting has provided[1] a list of observed hostnames for the DGA domain.

List #2

The second list has surfaced in a Paste bin paste[2], allegedly[3] sourced from Zetalytics / Zonecruncher[4].

NOTE: This list is fairly 'noisy', as it has non-decodable domain names.

By feeding both lists to our decoder, we can now obtain a list of decoded domains, that could have been generated by the victims of the Sunburst backdoor.

DISCLAIMER

It is not clear if the provided lists contain valid domain names that indeed belong to the victims. It is quite possible that the encoded domain names were produced by third-party tools, sandboxes, or by researchers that investigated and analysed the backdoor.

The decoded domain names are provided purely as a reverse engineering exercise. The resulting list was manually processed to eliminate noise, and to exclude duplicate entries. Following that, we have made an attempt to map the obtained domain names to the company names, using Google search. Reader's discretion is advised as such mappings could be inaccurate.

1 https://github.com/bambenek/research/blob/main/sunburst/uniq-hostnames.txt
2 https://pastebin.com/6EDgCKxd
3 https://twitter.com/0xrb/status/1339199268146442241
4 https://zetalytics.com/

Decoded Domain	Mapping (Could Be Inaccurate)
hgvc.com	Hilton Grand Vacations
Amerisaf	AMERISAFE, Inc.
kcpl.com	Kansas City Power and Light Company
SFBALLET	San Francisco Ballet
scif.com	State Compensation Insurance Fund
LOGOSTEC	Logostec Ventilação Industrial
ARYZTA.C	ARYZTA Food Solutions
bmrn.com	BioMarin Pharmaceutical Inc.
AHCCCS.S	Arizona Health Care Cost Containment System
nnge.org	Next Generation Global Education
cree.com	Cree, Inc (semiconductor products)
calsb.org	The State Bar of California
rbe.sk.ca	Regina Public Schools
cisco.com	Cisco Systems
pcsco.com	Professional Computer Systems
barrie.ca	City of Barrie
ripta.com	Rhode Island Public Transit Authority
uncity.dk	UN City (Building in Denmark)
bisco.int	Boambee Industrial Supplies (Bisco)
haifa.edu	University of Haifa
smsnet.pl	SMSNET, Poland
fcmat.org	Fiscal Crisis and Management Assistance Team
wiley.com	Wiley (publishing)
ciena.com	Ciena (networking systems)
belkin.com	Belkin
spsd.sk.ca	Saskatoon Public Schools
pqcorp.com	PQ Corporation
ftfcu.corp	First Tech Federal Credit Union
bop.com.pk	The Bank of Punjab
nvidia.com	NVidia
insead.org	INSEAD (non-profit, private university)
usd373.org	Newton Public Schools
agloan.ads	American AgCredit
pageaz.gov	City of Page
jarvis.lab	Erich Jarvis Lab
ch2news.tv	Channel 2 (Israeli TV channel)
bgeltd.com	Bradford / Hammacher Remote Support Software
dsh.ca.gov	California Department of State Hospitals
dotcomm.org	Douglas Omaha Technology Commission
sc.pima.gov	Arizona Superior Court in Pima County

Decoded Domain	Mapping (Could Be Inaccurate)
itps.uk.net	IT Professional Services, UK
moncton.loc	City of Moncton
acmedctr.ad	Alameda Health System
csci-va.com	Computer Systems Center Incorporated
keyano.local	Keyano College
uis.kent.edu	Kent State University
alm.brand.dk	Sydbank Group (Banking, Denmark)
ironform.com	Ironform (metal fabrication)
corp.ncr.com	NCR Corporation
ap.serco.com	Serco Asia Pacific
int.sap.corp	SAP
mmhs-fla.org	Cleveland Clinic Martin Health
nswhealth.net	NSW Health
mixonhill.com	Mixon Hill (intelligent transportation systems)
bcofsa.com.ar	Banco de Formosa
ci.dublin.ca.	Dublin, City in California
siskiyous.edu	College of the Siskiyous
weioffice.com	Walton Family Foundation
ecobank.group	Ecobank Group (Africa)
corp.sana.com	Sana Biotechnology
med.ds.osd.mi	US Gov Information System
wz.hasbro.com	Hasbro (Toy company)
its.iastate.ed	Iowa State University
amr.corp.intel	Intel
cds.capilanou.	Capilano University
e-idsolutions.	IDSolutions (video conferencing)
helixwater.org	Helix Water District
detmir-group.r	Detsky Mir (Russian children's retailer)
int.lukoil-int	LUKOIL (Oil and gas company, Russia)
ad.azarthritis	Arizona Arthritis and Rheumatology Associates
net.vestfor.dk	Vestforbrænding
allegronet.co.	Allegronet (Cloud based services, Israel)
us.deloitte.co	Deloitte
central.pima.g	Pima County Government
city.kingston.	City of Kingston
staff.technion	Technion - Israel Institute of Technology
airquality.org	Sacramento Metropolitan Air Quality Management District
phabahamas.org	Public Hospitals Authority, Caribbean
parametrix.com	Parametrix (Engineering)
ad.checkpoint.	Check Point

Decoded Domain	Mapping (Could Be Inaccurate)
corp.riotinto.	Rio Tinto (Mining company, Australia)
intra.rakuten.	Rakuten
us.rwbaird.com	Robert W. Baird & Co. (Financial services)
ville.terrebonn	Ville de Terrebonne
woodruff-sawyer	Woodruff-Sawyer & Co., Inc.
fisherbartoninc	Fisher Barton Group
banccentral.com	BancCentral Financial Services Corp.
taylorfarms.com	Taylor Fresh Foods
neophotonics.co	NeoPhotonics (optoelectronic devices)
gloucesterva.ne	Gloucester County
magnoliaisd.loc	Magnolia Independent School District
zippertubing.co	Zippertubing (Manufacturing)
milledgeville.l	Milledgeville (City in Georgia)
digitalreachinc	Digital Reach, Inc.
deniz.denizbank	DenizBank
thoughtspot.int	ThoughtSpot (Business intelligence)
lufkintexas.net	Lufkin (City in Texas)
digitalsense.co	Digital Sense (Cloud Services)
wrbaustralia.ad	W. R. Berkley Insurance Australia
christieclinic.	Christie Clinic Telehealth
signaturebank.l	Signature Bank
dufferincounty.	Dufferin County
mountsinai.hosp	Mount Sinai Hospital
securview.local	Securview Victory (Video Interface technology)
weber-kunststof	Weber Kunststoftechniek
parentpay.local	ParentPay (Cashless Payments)
europapier.inte	Europapier International AG
molsoncoors.com	Molson Coors Beverage Company
fujitsugeneral.	Fujitsu General
cityofsacramento	City of Sacramento
ninewellshospita	Ninewells Hospital
fortsmithlibrary	Fort Smith Public Library
dokkenengineerin	Dokken Engineering
vantagedatacente	Vantage Data Centers
friendshipstateb	Friendship State Bank
clinicasierravis	Clinica Sierra Vista
ftsillapachecasi	Apache Casino Hotel
voceracommunicat	Vocera (clinical communications)
mutualofomahaban	Mutual of Omaha Bank

Passive DNS requests

Previously, we have described the most important parts of the Sunburst backdoor functionality and its Domain Generation Algorithm (DGA).

This time, let's have a deeper look into the passive DNS requests reported by Open-Source Context[1] and Zetalytics[2].

The valid DNS requests generated by the malware fall into 2 groups:

- DNS requests that encode a local domain name
- DNS requests that encode data

- The first type of DNS requests allows splitting long domain names into separate requests. These requests are generated by the malware's functions *GetPreviousString()* and *GetCurrentString()*.

In general, the format of a DNS request that encodes a domain name may look like:

```
USER_ID.NUM.COMPUTER_DOMAIN[.]appsync-api.us-west-2[.]avsvmcloud[.]com
```

where:

- `USER_ID` is an 8-byte user ID that uniquely identifies a compromised host, encoded as a 15-character string
- `NUM` is a number of a domain name - either 0 or 1, encoded as a character
- `COMPUTER_DOMAIN` is an encoded local computer domain

Let's try decoding the following 3 DNS requests:

- `olc62cocacn7u2q22v02eu.appsync-api.us-west-2.avsvmcloud.com`
- `r1qshoj05ji05ac6eoip02jovt6i2v0c.appsync-api.us-west-2.avsvmcloud.com`
- `lt5ai41qh5d53qoti3mkmc0.appsync-api.us-west-2.avsvmcloud.com`

String #1

Let's start from the 1st string in the list:

```
olc62cocacn7u2q22v02eu.appsync-api.us-west-2.avsvmcloud.com.
```

In this string, the first 15-character string is an encoded `USER_ID`: **"`olc62cocacn7u2q`"**.

Once it is base-64 decoded, as explained previously, it becomes a 9-byte byte array:

```
86 7f 2f be f9 fb a3 ae c4
```

1 https://oscontext.com/
2 https://zetalytics.com/

The first byte in this byte array is a XOR key: `0x86`. Once applied to the 8 bytes that follow it, we get the 8-byte user ID - let's take a note and write it down, we will need it later:

```
f9 a9 38 7f 7d 25 28 42
```

Next, let's take the `NUM` part of the encoded domain: it's a character **"2"** located at the position #15 (starting from 0) of the encrypted domain.

In order to decode the `NUM` number, we have to take the first character of the encrypted domain, take the reminder of its division by `36`, and subtract the `NUM`'s position in the string **"0123456789abcdefghijklmnopqrstuvwxyz"**:

```
num = domain[0] % 36 - "0123456789abcdefghijklmnopqrstuvwxyz".IndexOf(domain.Substring(15, 1));
```

The result is `1`. That means the decrypted domain will be the 2nd part of a full domain name. The first part must have its `NUM` decoded as 0.

The `COMPUTER_DOMAIN` part of the encrypted domain is **"2v02eu"**. Once decoded, using the previously explained method, the decoded computer domain name becomes **"on.ca"**.

String #2

Let's decode the second passive DNS request from our list:

```
r1qshoj05ji05ac6eoip02jovt6i2v0c.appsync-api.us-west-2.avsvmcloud.com
```

Just as before, the decoded 8-byte user ID becomes:

```
f9 a9 38 7f 7d 25 28 42
```

The `NUM` part of the encoded domain, located at the position #15 (starting from 0), is a character **"6"**.

Let's decode it, by taking the first character (**"r" = 114**), take the reminder of its division by `36` (`114` % `36` = `6`), and subtracting the position of the character **"6"** in the **"0123456789abcdefghijklmnopqrstuvwxyz"**, which is `6`. The result is `0`. That means the decrypted domain will be the 1st part of the full domain name.

The `COMPUTER_DOMAIN` part of the encrypted domain is **"eoip02jovt6i2v0c"**. Once decoded, it becomes **"city.kingston."**

Next, we need to match 2 decrypted domains by the user ID, which is `f9 a9 38 7f 7d 25 28 42` in both cases, and concatenate the first and the second parts of the domain.

The result will be **"city.kingston.on.ca"**.

String #3

Here comes the most interesting part. Lets try to decrypt the string #3 from our list of passive DNS requests:

```
lt5ai41qh5d53qoti3mkmc0.appsync-api.us-west-2.avsvmcloud.com
```

The decoded user ID is not relevant, as the decoded **NUM** part is a number **-29**.

It's neither **0** nor **1**, so what kind of domain name that is? If we ignore the **NU**M part and decode the domain name, using the old method, we will get **"thx8xb"**, which does not look like a valid domain name.

Cases like that are not the noise, and are not some artificially encrypted artifacts that showed up among the DNS requests.

This is a different type of DNS requests. Instead of encoding local domain names, these types of requests contain data. They are generated by the malware's function GetNextStringEx().

The encryption method is different as well. Let's decrypt this request.

First, we can decode the encrypted domain, using the same base-64 method, as before.

The string will be decoded into 14 bytes:

```
7c a5 4d 64 9b 21 c1 74 a6 59 e4 5c 7c 7f
```

Let's decode these bytes, starting from the 2nd byte, and using the first byte as a XOR key. We will get:

```
7c d9 31 18 e7 5d bd 08 da 25 98 20 00 03
```

In this array, the bytes marked in yellow are an 8-byte User ID, encoded with a XOR key that is selected from 2 bytes marked in red.

Let's decode User ID:

```
for (int i = 0; i < 8; i++)
{
    bytes[i + 1] ^= bytes[11 - i % 2];
}
```

The decoded byte array becomes:

```
7c f9 a9 38 7f 7d 25 28 42 25 98 20 00 03
```

The User ID part in marked in yellow. Does it look familiar? Indeed, it's the same User ID we've seen before, when we decoded **"city.kingston.on.ca"**.

The next 3 bytes marked in red are: **25 98 20**.

- `2`
- `0x59820`

The first number 2 stands for the size of data that follows - this data is **`00 03`** (selected in green).

The number **`0x59820`**, or **`366,624`** in decimal, is a timestamp. It's a number of 4-second periods of time since 1 January 2010.

To obtain the real time stamp, we need to multiple it by 15 to get minutes, then add those minutes to 1 January 2010:

```
var date = (new DateTime(2010, 1, 1, 0, 0, 0, DateTimeKind.Utc)).AddMinutes(timestamp * 15);
```

For the number **`0x59820`**, the time stamp becomes 16 July 2020 12:00:00 AM - that's the day when the DNS request was made.

The remaining 2 bytes, **`00 03`**, encrypt the state of 8 security products, to indicate whether each one of them is running or whether it is stopped.

The 8 security products are:

- Windows Live OneCare / Windows Defender
- Windows Defender Advanced Threat Protection
- Microsoft Defender for Identity
- Carbon Black
- CrowdStrike
- FireEye
- ESET
- F-Secure

2 states for 8 products require 2 * 8 = 16 bits = 2 bytes.

The 2 bytes **`00 03`** in binary form are:

```
00 00 00 00 00 00 00 11
```

Here, the least-significant bits **`11`** identify that the first product in the list, Windows Live OneCare / Windows Defender, is reported as **`'running'`** (**`1`**) and as **`'stopped'`** (**`1`**).

Now we know that apart from the local domain, the trojanised SolarWinds software running on the same compromised host on **`"city.kingston.on.ca"`** domain has also reported the status of the Windows Defender software.

What Does it Mean?

As explained in the first part of our description, the malware is capable of stopping the services of security products, be manipulating registry service keys under Administrator account.

It's likely that the attackers are using DNS queries as a C2 channel to first understand what security products are present. Next, the same channel is used to instruct the malware to stop/deactivate these services, before the 2nd stage payload, TearDrop Backdoor, is deployed.

Armed with this knowledge, let's decode other passive DNS requests, printing the cases when the compromised host reports a running security software.

NOTES:

- As a private case, if the data size field is **0** or **1**, the timestamp field is not followed with any data. Such type of DNS request is generated by the malware's function *GetNextString()*. It is called 'a ping' in the listing below.
- If the first part of the domain name is missing, the recovered domain name is pre-pended with **'*'**.
- The malware takes the time difference in minutes, then divides it by 30 and then converts the result from *double* type to *int* type; as a result of such conversion, the time stamps are truncated to the earliest half hour.

```
2D82B037C060515C SFBALLET

          Data:
          Windows Live OneCare / Windows Defender [running] 11/07/2020 12:00:00 AM

          Pings:
          12/07/2020 12:30:00 AM

70DEE5C062CFEE53 ccscurriculum.c

          Data:
          ESET [running] 17/04/2020 4:00:00 PM

          Pings:
          20/04/2020 5:00:00 PM

AB902A323B541775 mountsinai.hospital

          Pings:
          4/07/2020 12:30:00 AM

9ACC3A3067DC7FD5 *ripta.com

          Data:
          ESET [running] 12/09/2020 6:30:00 AM

          Pings:
          13/09/2020 7:30:00 AM
          14/09/2020 9:00:00 AM
```



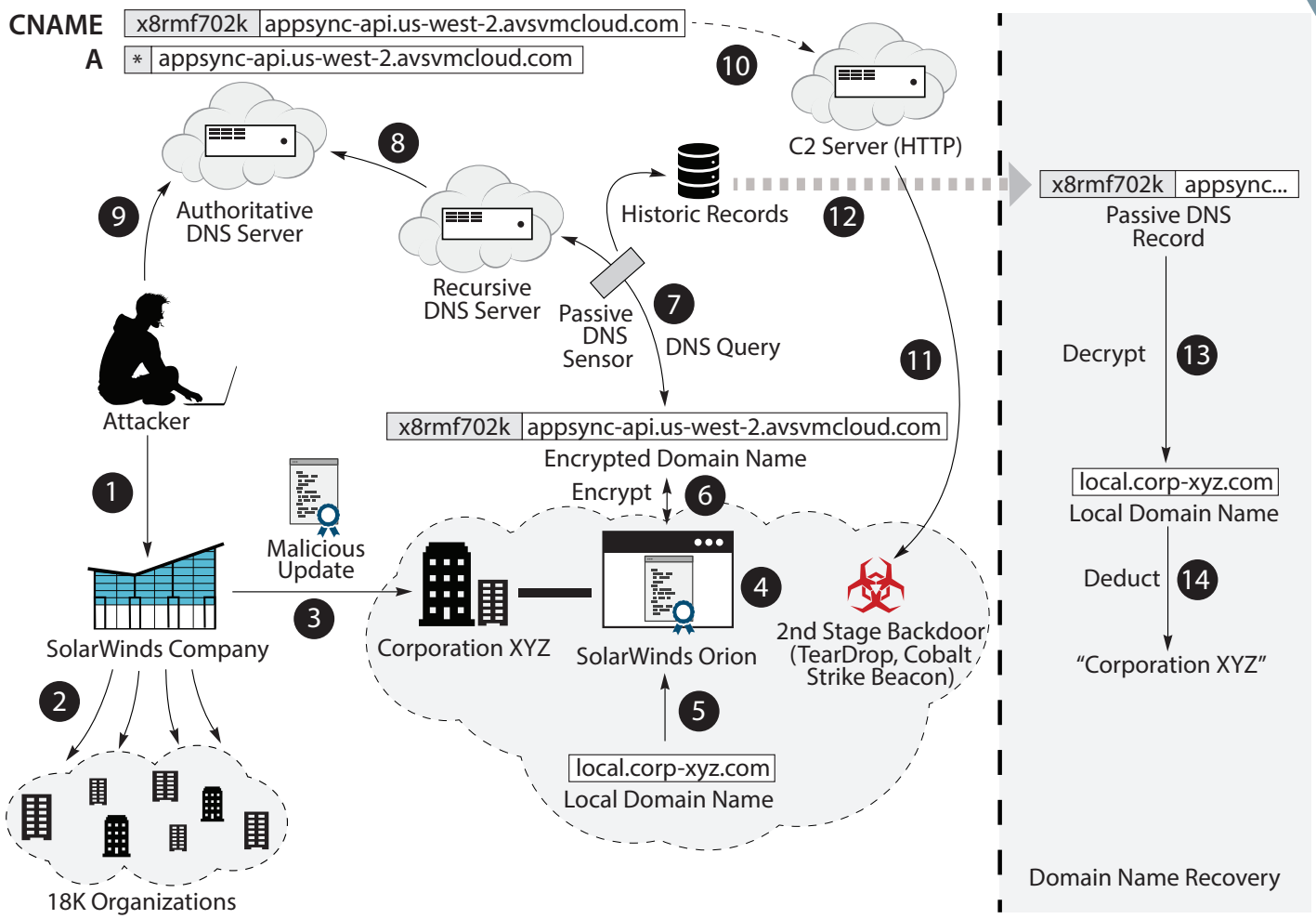
```
CB34C4EBCB12AF88 DPCITY.I7a

                Data:
                ESET [running] 26/06/2020 5:00:00 PM

                Pings:
                27/06/2020 6:30:00 PM
                28/06/2020 7:30:00 PM
                29/06/2020 8:30:00 PM
                29/06/2020 8:30:00 PM

E5FAFE265E86088E *scroot.com

                Data:
                CrowdStrike [running] 25/07/2020 2:00:00 PM

                Pings:
                26/07/2020 2:30:00 PM
                26/07/2020 2:30:00 PM
                27/07/2020 3:00:00 PM
                27/07/2020 3:00:00 PM
```

Full list of decoded pDNS requests can be found here[1].
An example of a working implementation is available at this[2] repo.

For a high-level illustration of the DNS Tunneling method used in the SolarWinds supply chain attack, please see Addendum A.

1 https://pastebin.com/xLs5H10C
2 https://github.com/sysopfb/open_mal_analysis_notes/tree/master/sunburst_dga

Addendum A. DNS Tunneling In The SolarWinds Supply Chain Attack



1. An Attacker compromises SolarWinds company and trojanizes a DLL that belongs to its software.
2. Some of the customers receive the malicious DLL as an update for the SolarWinds Orion software.
3. *"Corporation XYZ"* receives the malicious and digitally signed DLL via update.
4. SolarWinds Orion software loads the malicious DLL as a plugin.
5. Once activated, the DLL reads a local domain name *"local.corp-xyz.com"* (a fictious name).
6. The malware encrypts the local domain name and adds it to a long domain name.
7. The long domain name is queried with a DNS server (can be tapped by a passive DNS sensor).
8. The recursive DNS server is not authorized to resolve ***avsvmcloud[.]com***, so it forwards the request.
9. An attacker-controlled authoritative DNS server resolves the request with a wildcard A record.
10. The Attacker checks the victim's name, then adds a CNAME record for the victim's domain name.
11. The new CNAME record resolves the long domain name into an IP of an HTTP-based C2 server.
12. The malicious DLL downloads and executes the 2nd stage malware (TearDrop, Cobalt Strike Beacon).
13. A Threat Researcher accesses the passive DNS (pDNS) records.
14. One of the long domain names from the pDNS records is decrypted back into *"local.corp-xyz.com"*.
15. The Researcher deducts that the decrypted local domain name belongs to *"Corporation XYZ"*.

