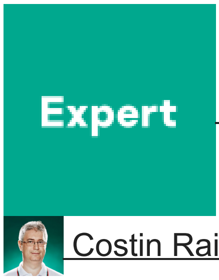
Sunburst: connecting the dots in the DNS requests

SL securelist.com/sunburst-connecting-the-dots-in-the-dns-requests/99862/

Authors

- 

Igor Kuznetsov
-
Costin Raiu

On December 13, 2020 FireEye published important details of a newly discovered supply chain attack. An unknown attacker, referred to as UNC2452 or DarkHalo planted a backdoor in the SolarWinds Orion IT software. This backdoor, which comes in the form of a .NET module, has some really interesting and rather unique features.

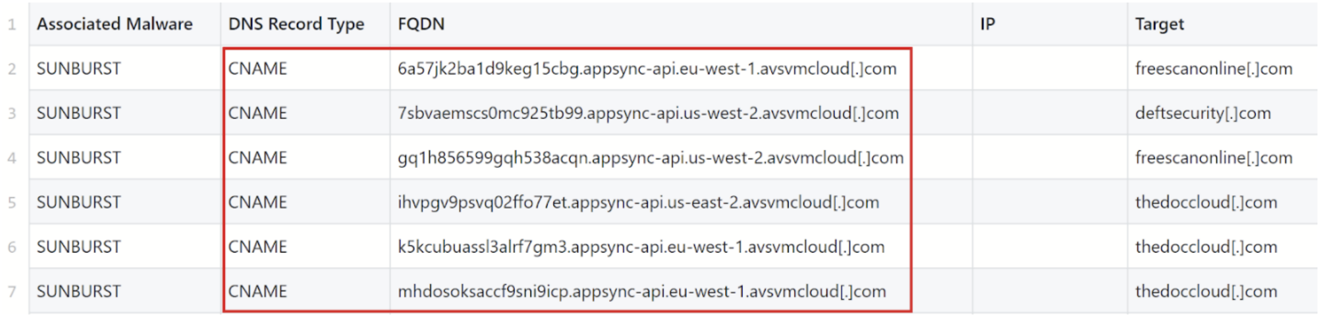
We spent the past days checking our own telemetry for signs of this attack, writing additional detections and making sure that our users are protected. At the moment, we identified approximately ~100 customers who downloaded the trojanized package containing the Sunburst backdoor. Further investigation is ongoing and we will continue to update with our findings.

Now, several things really stand out for this incident. This supply chain attack was designed in a very professional way – kind of putting the “A” in “APT” – with a clear focus on staying undetected for as long as possible. For instance, before making the first internet connection to its C2s, the Sunburst malware lies dormant for a long period, of up to two weeks, which prevents an easy detection of this behavior in sandboxes. Other advanced threat groups are also known to adopt similar strategies, for instance with hardware or firmware implants, which “sleep” for weeks or months before connecting to their C2 infrastructure. This explains why this attack was so hard to spot.

One of the things that sets this apart from other cases, is the peculiar victim profiling and validation scheme. Through the SolarWinds Orion IT packages, the attackers reached about 18,000 customers, according to the SolarWinds alert. Yet, out of these 18.000, it would appear that only a handful were interesting to them. Considering the fact that having the resources to manually exploit 18,000 computer networks is probably outside the reach of most if not all the attackers out there, this leads to the point that obviously some of those would have been a higher priority. Finding which of the 18,000 networks were further exploited, receiving more malware, installing persistence mechanisms and exfiltrating data is likely going to cast some light into the attacker’s motives and priorities.

In the initial phases, the Sunburst malware talks to the C&C server by sending encoded DNS requests. These requests contain information about the infected computer; if the attackers deem it interesting enough, the DNS response includes a CNAME record pointing to a second level C&C server.

Our colleagues from FireEye published several DNS requests that supposedly led to CNAME responses on Github:
https://github.com/fireeye/sunburst_countermeasures/blob/main/indicator_release/Indicator_Release_NBIs.csv



Associated Malware	DNS Record Type	FQDN	IP	Target
SUNBURST	CNAME	6a57jk2ba1d9keg15cbg.appsync-api.eu-west-1.avsvmcloud[.]com		freescanonline[.]com
SUNBURST	CNAME	7sbvaemscs0mc925tb99.appsync-api.us-west-2.avsvmcloud[.]com		deftsecurity[.]com
SUNBURST	CNAME	gq1h856599gqh538acqn.appsync-api.us-west-2.avsvmcloud[.]com		freescanonline[.]com
SUNBURST	CNAME	ihvpgv9psvq02ffo77et.appsync-api.us-east-2.avsvmcloud[.]com		thedoccloud[.]com
SUNBURST	CNAME	k5kcubuassl3alrf7gm3.appsync-api.eu-west-1.avsvmcloud[.]com		thedoccloud[.]com
SUNBURST	CNAME	mhdosoksaccf9sni9icp.appsync-api.eu-west-1.avsvmcloud[.]com		thedoccloud[.]com

DNS CNAME request-response pairs (Copyright 2020 by FireEye, Inc.)

The goal

Knowing that the DNS requests generated by Sunburst encode some of the target’s information, the obvious next step would be to extract that information to find out who the victims are!

Our colleagues from QiAnXin Technology already published a Python script to decode the domain names (on Github, of course): https://github.com/RedDrip7/SunBurst_DGA_Decode/blob/main/decode.py . Unfortunately, that script cannot decode all the DNS requests, besides it is always good to practice in reverse engineering anyway, so let's look in the malware code.

TL;DR

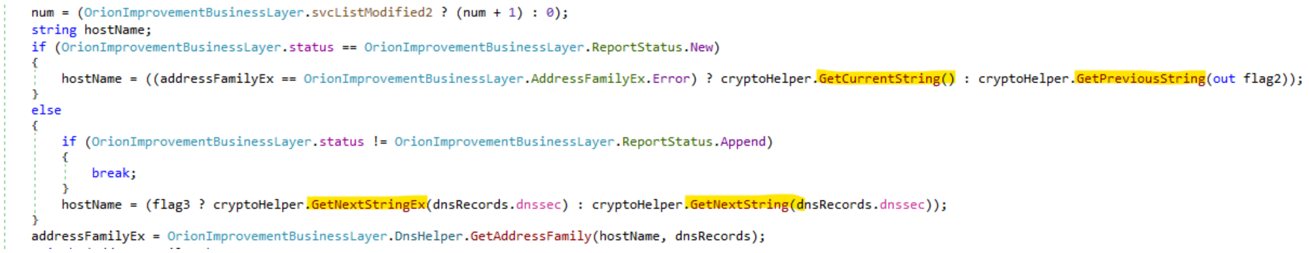
Three of the requests from FireEye's Github repository can be mapped to two domain names that belong to a government organization and a telecommunications company in the US.

Low-level details

The sample we'll be analyzing is **2c4a910a1299cdae2a4e55988a2f102e**. All we need is a .NET decompiler (dnSpy is a good one).

The logic that is interesting to us is located in *OrionImprovementBusinessLayer.Update*, which in turn calls one of the four functions: *GetCurrentString, GetPreviousString, GetNextStringEx* and *GetNextString*.

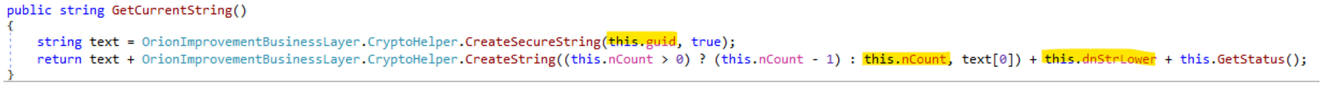
These four functions correspond to four different DNS-based communication methods.



```
num = (OrionImprovementBusinessLayer.svcListModified2 ? (num + 1) : 0);
string hostName;
if (OrionImprovementBusinessLayer.status == OrionImprovementBusinessLayer.ReportStatus.New)
{
    hostName = ((addressFamilyEx == OrionImprovementBusinessLayer.AddressFamilyEx.Error) ? cryptoHelper.GetCurrentString() : cryptoHelper.GetPreviousString(out flag2));
}
else
{
    if (OrionImprovementBusinessLayer.status != OrionImprovementBusinessLayer.ReportStatus.Append)
    {
        break;
    }
    hostName = (flag3 ? cryptoHelper.GetNextStringEx(dnsRecords.dnssec) : cryptoHelper.GetNextString(dnsRecords.dnssec));
}
addressFamilyEx = OrionImprovementBusinessLayer.DnsHelper.GetAddressFamily(hostName, dnsRecords);
```

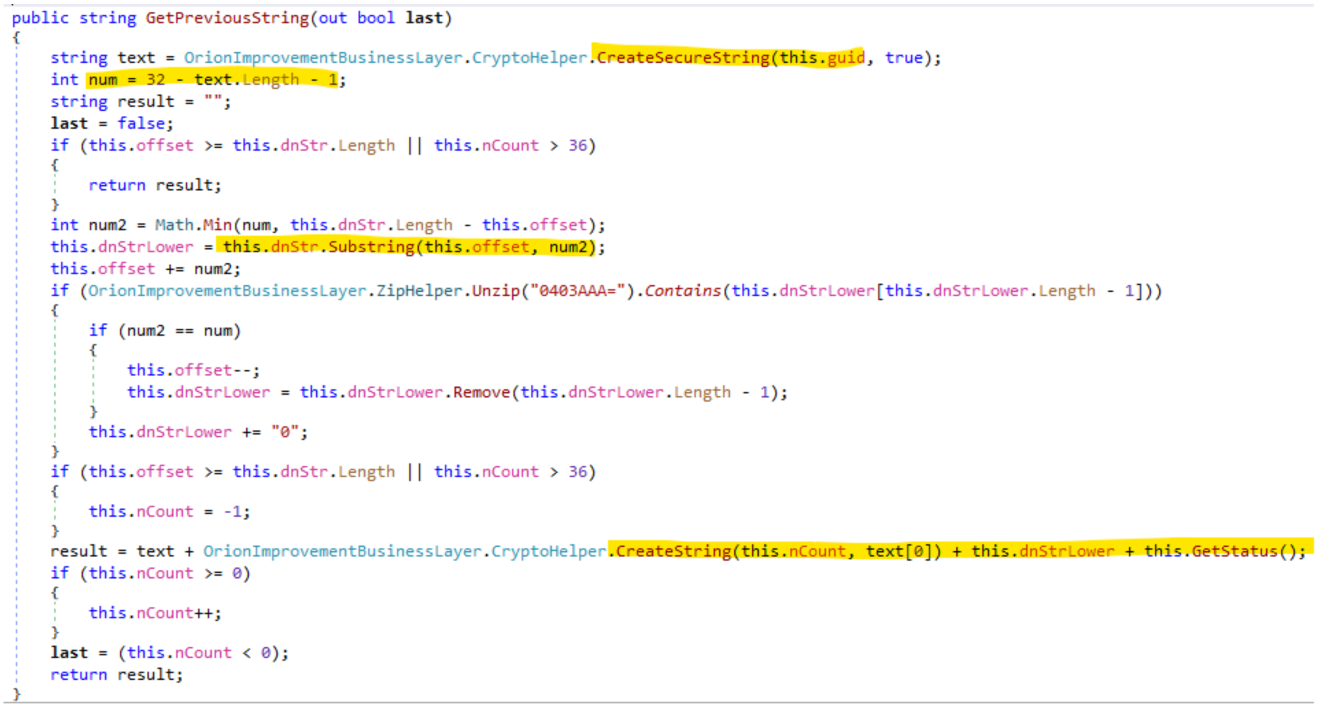
Now, to find out what kind of data is encoded in the DNS requests (the *hostname* variable) we need to look closer into each one of these functions.

The first function *GetCurrentString* generates strings that contain a supposedly unique target's identifier (*this.guid)*, the target's hostname (*this.dnStrLower*) and the rest of the hostname that will be in form of "*appsync-api.*.avsvmcloud[.]com"*. The encoding of the data is done by the functions *CreateSecureString* and *CreateString*.



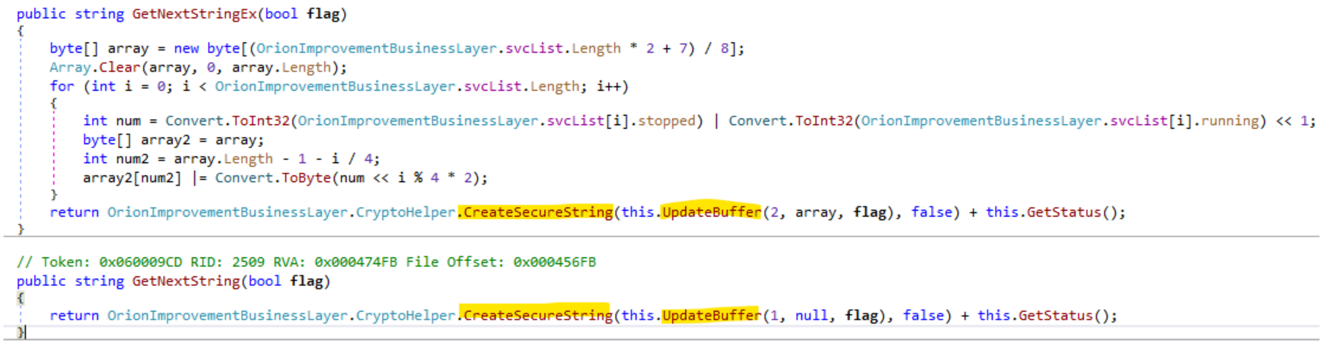
```
public string GetCurrentString()
{
    string text = OrionImprovementBusinessLayer.CryptoHelper.CreateSecureString(this.guid, true);
    return text + OrionImprovementBusinessLayer.CryptoHelper.CreateString((this.nCount > 0) ? (this.nCount - 1) : this.nCount, text[0]) + this.dnStrLower + this.GetStatus();
}
```

The function *GetPreviousString* is bigger, but produces a similar hostname for a DNS request. It includes a part of the target's hostname in the request, so that it would match the limitations on the request length. Each such request also includes the sequence number (*this.nCount*) that is the offset of the current substring from the beginning of the hostname.



```
public string GetPreviousString(out bool last)
{
    string text = OrionImprovementBusinessLayer.CryptoHelper.CreateSecureString(this.guid, true);
    int num = 32 - text.Length - 1;
    string result = "";
    last = false;
    if (this.offset >= this.dnStr.Length || this.nCount > 36)
    {
        return result;
    }
    int num2 = Math.Min(num, this.dnStr.Length - this.offset);
    this.dnStrLower = this.dnStr.Substring(this.offset, num2);
    this.offset += num2;
    if (OrionImprovementBusinessLayer.ZipHelper.Unzip("0403AAA=").Contains(this.dnStrLower[this.dnStrLower.Length - 1]))
    {
        if (num2 == num)
        {
            this.offset--;
            this.dnStrLower = this.dnStrLower.Remove(this.dnStrLower.Length - 1);
        }
        this.dnStrLower += "0";
    }
    if (this.offset >= this.dnStr.Length || this.nCount > 36)
    {
        this.nCount = -1;
    }
    result = text + OrionImprovementBusinessLayer.CryptoHelper.CreateString(this.nCount, text[0]) + this.dnStrLower + this.GetStatus();
    if (this.nCount >= 0)
    {
        this.nCount++;
    }
    last = (this.nCount < 0);
    return result;
}
```

The remaining two functions, *GetNextStringEx* and *GetNextString,* generate a different type of DNS requests. They include only the target's UID, hashes of the running processes of interest and (only in the Ex version) the list and status of these processes. The target's UID is then encrypted, and the data is encoded with *CreateSecureString*.



```
public string GetNextStringEx(bool flag)
{
    byte[] array = new byte[(OrionImprovementBusinessLayer.svcList.Length * 2 + 7) / 8];
    Array.Clear(array, 0, array.Length);
    for (int i = 0; i < OrionImprovementBusinessLayer.svcList.Length; i++)
    {
        int num = Convert.ToInt32(OrionImprovementBusinessLayer.svcList[i].stopped) | Convert.ToInt32(OrionImprovementBusinessLayer.svcList[i].running) << 1;
        byte[] array2 = array;
        int num2 = array.Length - 1 - i / 4;
        array2[num2] |= Convert.ToByte(num << i % 4 * 2);
    }
    return OrionImprovementBusinessLayer.CryptoHelper.CreateSecureString(this.UpdateBuffer(2, array, flag), false) + this.GetStatus();
}

// Token: 0x060009CD RID: 2509 RVA: 0x000474FB File Offset: 0x000456FB
public string GetNextString(bool flag)
{
    return OrionImprovementBusinessLayer.CryptoHelper.CreateSecureString(this.UpdateBuffer(1, null, flag), false) + this.GetStatus();
}
```

The problem

The encryption used in the malware is just a simple XOR operation, and the encoding is either a substitution cipher or Base32 with a custom alphabet. However, if we reverse the sequence of operations of *GetPreviousString* or *GetCurrentString* for the known CNAME DNS requests published by FireEye, the resulting strings don't look like valid domain names!

A possible explanation is that the requests were generated by the third or fourth communication methods, described as *GetNextStringEx* or *GetNextString.* Indeed, they can be decoded without errors and the size of decoded data fits. However, these requests don't have the target’s name included!

The solution

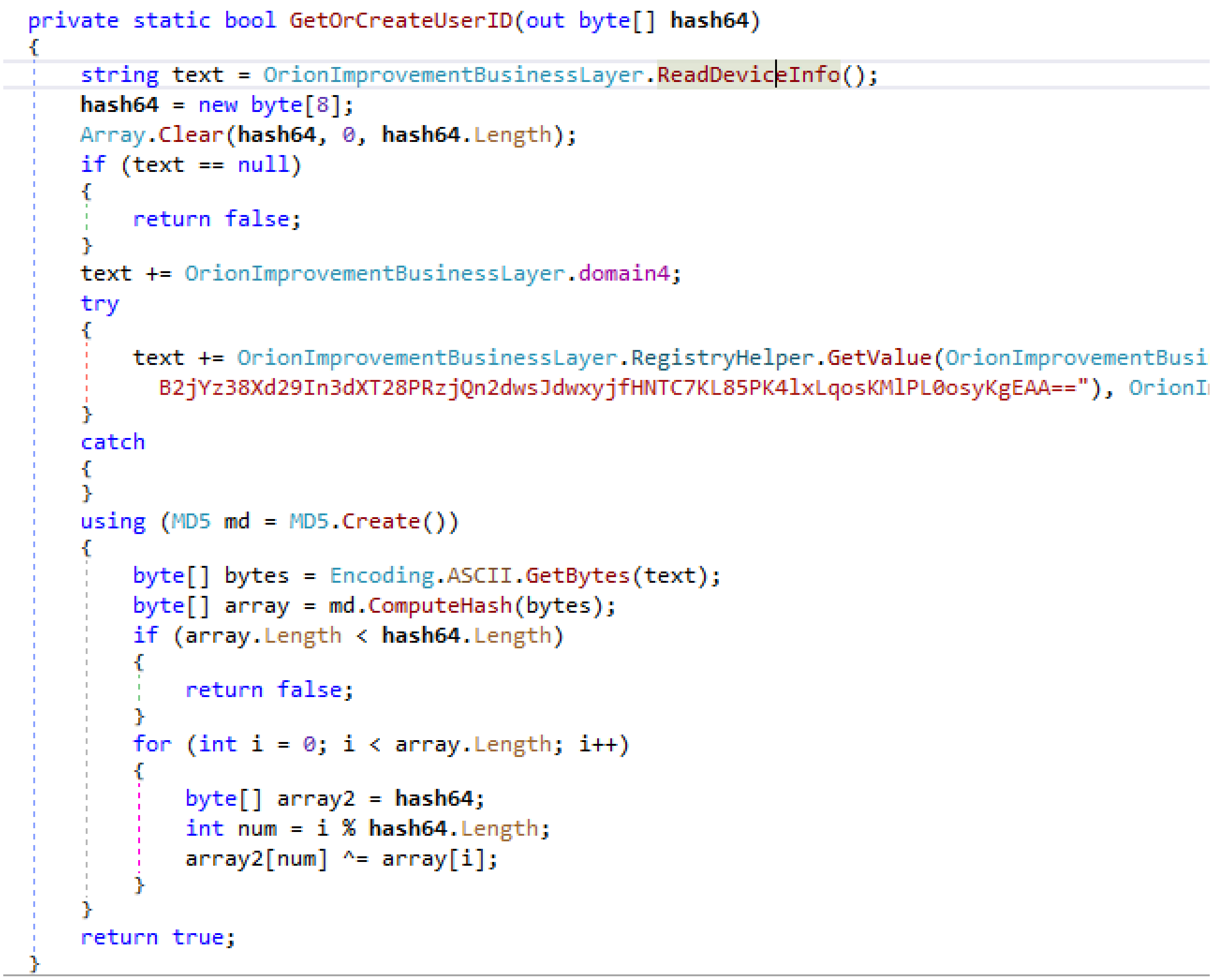
At this point, a question arises – can we match any of existing private and public DNS data for the malware root C2 domain, “avsvmcloud[.]com” with the CNAME records, to identify who was targeted for further exploitation?

A list of SUNBURST-generated domain names that include the domain names were kindly shared by John Bambenek on Github:
https://github.com/bambenek/research/blob/main/sunburst/uniq-hostnames.txt .

Here’s a few such examples:
nnbggtlr1iv0v3vfnfaddfe.appsync-api.us-west-2.avsvmcloud[.]com
nq97kdu88pn1qpv8f3t5.appsync-api.us-east-1.avsvmcloud[.]com
nr2ia9qfa349b0q2oi60bou6iuir02rn.appsync-api.us-east-1.avsvmcloud[.]com

We complemented John’s data with our own datasets as well as other publicly available pDNS databases. Each one of these DNS requests also has the Base32-encoded UID. Since the UIDs are also included in other types of requests (types 3 and 4) in encrypted form, this allows us to match the requests!

The target’s UID is calculated in OrionImprovementBusinessLayer.GetOrCreateUserID by MD5-hashing the MAC address of the first online network adapter, then XORing it down to 64 bits.



```
private static bool GetOrCreateUserID(out byte[] hash64)
{
    string text = OrionImprovementBusinessLayer.ReadDeviceInfo();
    hash64 = new byte[8];
    Array.Clear(hash64, 0, hash64.Length);
    if (text == null)
    {
        return false;
    }
    text += OrionImprovementBusinessLayer.domain4;
    try
    {
        text += OrionImprovementBusinessLayer.RegistryHelper.GetValue(OrionImprovementBusi
          B2jYz38Xd29In3dXT28PRzjQn2dwsJdwxyjfHNTC7KL85PK4lxLqosKMlPL0osyKgEAA=="), OrionI
    }
    catch
    {
    }
    using (MD5 md = MD5.Create())
    {
        byte[] bytes = Encoding.ASCII.GetBytes(text);
        byte[] array = md.ComputeHash(bytes);
        if (array.Length < hash64.Length)
        {
            return false;
        }
        for (int i = 0; i < array.Length; i++)
        {
            byte[] array2 = hash64;
            int num = i % hash64.Length;
            array2[num] ^= array[i];
        }
    }
    return true;
}
```

The DNS requests published by FireEye on their GitHub have the following encrypted UIDs inside:

DNS request	UID (64 bit)
6a57jk2ba1d9keg15cbg.appsync-api.eu-west-1.avsvmcloud[.]com	0xEED328E059EB07FC
7sbvaemscs0mc925tb99.appsync-api.us-west-2.avsvmcloud[.]com	0x683D2C991E01711D
gq1h856599gqh538acqn.appsync-api.us-west-2.avsvmcloud[.]com	0x2956497EB4DD0BF9
ihvpgv9psvq02ffo77et.appsync-api.us-east-2.avsvmcloud[.]com	0xF7A37335B9E57DDB
k5kcubuassl3alrf7gm3.appsync-api.eu-west-1.avsvmcloud[.]com	0xA46E6E874771323C

mhdosoksaccf9sni9icp.appsync-api.eu-west-1.avsvmcloud[.]com	0xA46E6E874771323C

In total, we analyzed 1722 DNS records, leading to 1026 unique target name parts and 964 unique UIDs.

Matching the two lists we got the following data:
domain name part(0x2956497EB4DD0BF9)=central.****.g
domain name part(0x2956497EB4DD0BF9)=ov
domain name part(0x683D2C991E01711D)=central.****.g
domain name part(0x683D2C991E01711D)=ov
domain name part(0xF7A37335B9E57DDB)=***net.***.com
These steps effectively decoded 3 of the 6 CNAME records provided by FireEye into two possible domains:

*****net.***.com – a rather big telecommunications company from the US, serving more than 6 million customers**
central.*.gov – a governmental organization from the US**

Please note that for ethical reasons, we do not include these exact domain names here. We notified the two organizations in question though, offering our support to discover further malicious activities, if needed.

It should also be noted that there is no way to be sure that machines in these two domains were actually further exploited. This being a probabilistic puzzle, **we can assume with a high degree of certitude the two decoded domains were interesting to the attackers**, however, we cannot be 100% sure that associated organizations were the subject of further malicious activities.

To summarize our research, the UIDs we discovered match two domain names that belong to a US government organization and a large US telecommunications company. It is likely that other interesting targets were selected by the attackers for further exploitation. If you happen to have access to large DNS databases, including CNAME replies for any subdomain in “avsvmcloud[.]com”, please let us know! (contact: intelreports (at) kaspersky [dot] com)

In order to help the community to potentially identify other interesting targets for the attackers, we are publishing the source code for the decoder:
https://github.com/2igosha/sunburst_dga

Stay safe!

More details and mitigations about Sunburst, UNC2452 / DarkHalo are available to customers of Kaspersky Intelligence Reporting. Contact: intelreports (at) kaspersky [dot] com

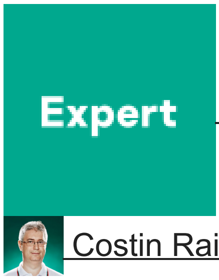
Sunburst / UNC2452 / DarkHalo FAQ

1. **Who is behind this attack? I read that some people say APT29/Dukes?**
   At the moment, there are no technical links with previous attacks, so it may be an entirely new actor, or a previously known one that evolved their TTPs and opsec to the point where they can't be linked anymore. Volexity, who previously worked on other incidents related to this, named the actor DarkHalo. FireEye named them "UNC2452", suggesting an unknown actor. While some media sources linked this with APT29/Dukes, this appears to be either speculation or based on some other, unavailable data, or weak TTPs such as legitimate domain re-use.
2. **I use Orion IT! Was I a target of this attack?**
   First of all, we recommend scanning your system with an updated security suite, capable of detecting the compromised packages from SolarWinds. Check your network traffic for all the publicly known IOCs – see https://github.com/fireeye/sunburst_countermeasures. The fact that someone downloaded the trojanized packages doesn't also mean they were selected as a target of interest and received further malware, or suffered data exfiltration. It would appear, based on our observations and common sense, that only a handful of the 18,000 Orion IT customers were flagged by the attackers as interesting as were further exploited.
3. **Was this just espionage or did you observe destructive activities, such as ransomware?**
   While the vast majority of the high-profile incidents nowadays include ransomware or some sort of destructive payload (see NotPetya, Wannacry) in this case, it would appear the main goal was espionage. The attackers showed a deep understanding and knowledge of Office365, Azure, Exchange, Powershell and leveraged it in many creative ways to constantly monitor and extract e-mails from their true victims' systems.
4. **How many victims have been identified?**
   Several publicly available data sets, such as the one from John Bambenek, include DNS requests encoding the victim names. It should be noted that these victim names are just the "first stage" recipients, not necessarily the ones the attackers deemed interesting. For instance, out of the ~100 Kaspersky users with the trojanized package, it would appear that none were interesting to the attackers to receive the 2nd stage of the attack.
5. **What are the most affected countries?**
   To date, we observed users with the trojanized Orion IT package in 17 countries. However, the total number is likely to be larger, considering the official numbers from SolarWinds.
6. **Why are you calling this an attack, when it's just exploitation? (CNA vs CNE)**
   Sorry for the terminology, we simply refer to it as a "supply chain attack". It would be odd to describe it as a "supply chain exploitation".

7. **Out of the 18,000 first stage victims, how many were interesting to the attackers?**
   This is difficult to estimate, mostly because of the lack of visibility and because the attackers were really careful in hiding their traces. Based on the CNAME records published by FireEye, we identified only two entities, a US government organization and a telecommunications company, who were tagged and “promoted” to dedicated C2s for additional exploitation.
8. **Why didn’t you catch this supply chain attack in the first place?**
   That’s a good question! In particular, two things made it really stealthy. The slow communication method, in which the malware lies dormant for up to two weeks, is one of them. The other one is the lack of x86 shellcode; the attackers used a .NET injected module. Last but not least, there was no significant change in the file size of the module when the malicious code was added. We observed two suspicious modules in 2019, which jumped from the usual 500k to 900k for SolarWinds.Orion.Core.BusinessLayer.dll. When the malicious code was first added, in February 2020, the file didn’t change size in a significant manner. If the attackers did this on purpose, to avoid future detections, then it’s a pretty impressive thing.
9. **What is Teardrop?**
   According to FireEye, Teardrop is malware delivered by the attackers to some of the victims. It is an unknown memory-only dropper suspected to deliver a customized version of the well-known CobaltStrike BEACON. To date, we haven’t detected any Teardrop samples anywhere.
10. **What made this such a successful operation?**
    Probably, a combination of things – a supply chain attack, coupled with a very well thought first stage implant, careful victim selection strategies and last but not least, no obvious connections to any previously observed TTPs.

- Backdoor
- Malware Descriptions
- Malware Technologies
- Sunburst
- Supply-chain attack
- Targeted attacks

Authors

- 

  Igor Kuznetsov
- Costin Raiu

Sunburst: connecting the dots in the DNS requests

Your email address will not be published. Required fields are marked *