PlugX: some uncovered points

airbus-cyber-security.com/plugx-some-uncovered-points

January 6, 2014

by Fabien Perigaud

PlugX (or Korplug, or Gulpix) is a well-known RAT involved in many APT cases. Some excellent write-ups about this malware have already been published by the CIRCL, Sophos and AlienVault. Since we met it on an incident response case back in 2012, we followed its evolution to improve our knowledge, rules and tools. We're planning to release details about this malware in a small serie of blog posts, to cover some points which have not been published yet.

This first post will cover some internals of the original PlugX malware and we'll deal with its evolution in the next one.

Internal Versions

Some PlugX samples found in the wild are debug releases, still containing the full path to the "XPlug.h" file. This full path allows to identify several versions of PlugX, ranging from 2.0 to 8.0. This kind of information has been used by AlienVault to identify the potential author of the RAT, via the leaked username. Here is a list of all the debug paths we found in the wild:

```
i:\work\plug2.0(.....)\shellcode\shellcode\
i:\work\plug2.0\shellcode\shellcode\
d:\work\plug2.0\shellcode\shellcode\
d:\work\plug2.5\shellcode\shellcode\
c:\users\whg\desktop\plug2.5(nzqk)\shellcode\shellcode\
c:\users\whg\desktop\plug2.5(rose)\shellcode\shellcode\
c:\users\whg\desktop\plug3.0\shellcode\shellcode\
d:\work\plug3.0(gf)\shellcode\shellcode\
d:\work\plug3.0(gf)udp\shellcode\shellcode\
d:\work\plug3.0(lyt)\shellcode\shellcode\
d:\work\plug3.0\shellcode\shellcode\
d:\work\plug3.1(icesword)\shellcode\shellcode\
d:\work\plug4.0(....)(......)\shellcode\shellcode\
d:\work\plug4.0(cammute)\shellcode\shellcode\
d:\work\plug4.0(msidb)(lyt)\shellcode\shellcode\
d:\work\plug4.0(nvsmart)(....)(7.0)\shellcode\shellcode\
d:\work\plug4.0(nvsmart)(hrb)\shellcode\shellcode\
d:\work\plug4.0(nvsmart)(mrxy)(675960)\shellcode\shellcode\
d:\work\plug4.0(nvsmart)(sxl)\shellcode\shellcode\
d:\work\plug4.0(nvsmart)\shellcode\shellcode\
d:\work\plug4.0(shellcode)(....)\shellcode\shellcode\
d:\work\plug4.0(shellcode)(hrb)(gf)\shellcode\shellcode\
d:\work\plug4.0(shellcode)(hrb)\shellcode\shellcode\
d:\work\plug4.0\shellcode\shellcode\
d:\work\plug5.0(3f)(zxf)(360)(9022863)(scldr3.0)\shellcode\shellcode\
d:\work\plug5.0(hrb)\shellcode\shellcode\
d:\work\plug5.0\shellcode\shellcode\
d:\work\plug6.0(360)(gadget)(....)\shellcode\shellcode\
d:\work\plug6.0(360)(gadget)(........)(....)\shellcode\shellcode\
d:\work\plug6.0(360)(hkcmd)(xts)(scldr3.0)\shellcode\shellcode\
d:\work\plug6.0(360)(hkcmd)(xts)\shellcode\shellcode\
d:\work\plug6.0(360)(mcinsupd)(....)\shellcode\shellcode\
d:\work\plug6.0(360)(mcinsupd)(48846669)\shellcode\shellcode\
d:\work\plug6.0(360)(mcoemcpy)(hhhtwy)(scldr3.0)\shellcode\shellcode\
d:\work\plug6.0(360)(minidownloader)\shellcode\shellcode\
d:\work\plug6.0\plug6.0(minidownloader)\shellcode\shellcode\
d:\work\plug6.0\plug6.0(rstray)\shellcode\shellcode\
d:\work\plug7.0(....)(3..)\plug7.0(oleview)(....3)(........)\shellcode\shellcode\
d:\work\plug7.0(arotutorial)(ykcai)(2)\shellcode\shellcode\
d:\work\plug7.0(bdreinit)(....)(360)\shellcode\shellcode\
d:\work\plug7.0(mcappcfg)(gf)(....)\shellcode\shellcode\
d:\work\plug7.0(mcvsmap)(fking)(....)\shellcode\shellcode\
d:\work\plug8.0(hkcmd)(....)\plug6.0(360)(mcoemcpy)(hhhtwy)
(scldr3.0)\shellcode\shellcode\
d:\work\plug8.0(mcoemcpy)(lyt)\shellcode\shellcode\
```

As you can see, it seems that PlugX has evolved quite fast, upgrading from “version” 2.0 to 8.0 in a few months (compilation dates from 03/2012 to 04/2013).

However, we also found that there is an internal version number in the malware’s code, used in the hello packet sent to the C&C. This version number is a dword which hexadecimal representation seems to be a date. More interestingly, we only found 4 different internal version numbers for all the aforementioned debug strings:

```
20120123
20121016
20121107
20121203
```

For the record, Destory, which is supposed to be PlugX ancestor, has a similar version number. In all the samples we collected, it is set to 20100921.

While one could think these different versions could indicate different communication protocols, the various HTTP headers and requests used could not be directly linked to a specific version...

Network Communications

In 90% of the samples we collected, the HTTP request matches the following pattern (request and headers):

```
POST /update?id=%8.8x
X-Session / X-Status / X-Size / X-Sn
```

However, some samples exhibited a different HTTP communication protocol. Here are the other variants we observed:

```
POST /%p/%p/%p
X-Session / X-Status / X-Size / X-Sn

POST /%p
X-Session / X-Status / X-Size / X-Sn

POST /%p%p/%p
CLR / CLA / CLC / CLX

POST /%p%p/
X801 / X824 / X851 / X879

POST /%d?%d?%d
AndrioA / AndrioB / AndrioC / AndrioD

POST /%p/%p/%p
HNS1 / HNS2 / HNS3 / HNS4

POST /%p/%p/%p
H0 / H1 / H2 / H3

GET /%8.8x/
XSID / XSTATE / XSIZE / XSN
```

The cipher algorithm has not changed through all these versions (more to come in the next post). For all the possible communication protocols (TCP/UDP/HTTP), the data is sent encrypted and/or compressed (using LZNT1 algorithm), with a 10-bytes header containing:

```
Cipher Key
Flags (encryption/compression enabled)
Command ID
Sizes of payload (compressed and uncompressed)
Error code
```

Configuration

The various versions can have a different configuration blob length. Versions before 3.0 (according to the debug string) have a configuration length of 0xbe4, whereas later ones are bigger (0x150c). This change is only due to the space expansion for storing strings (such as the name of the created service) from 0x80 to 0x200. Samples with the "20121107" internal version have even bigger configurations (0x1d18) because of new features in the malware (as Run key persistence and process injection).

The configuration is ciphered using the well-known PlugX algorithm (the key lies in the 4 first bytes):

```
v1=v2=v3=v4=key
i=0
out=""
while i<len(buff):
     v1 = (v1 + (v1 >> 3) - 0x11111111)&0xffffffff;
     v2 = (v2 + (v2 >> 5) - 0x22222222)&0xffffffff;
     v3 = (v3 + 0x33333333 - (v3 << 7))&0xffffffff;
     v4 = (v4 + 0x44444444 - (v4 << 9))&0xffffffff;
     mysum=(v1+v2+v3+v4)&0xff
     out=out+chr(ord(buff[i])^mysum)
     i=i+1
```

Let's focus on the 0x150c length configuration, as it is the most widespread one. The following information is available:

Flags (Offset 0x8, Length 0x2c)

There are 11 different flags available, but some of them are not currently implemented/used. Each flag is a DWORD which value can be 0 (disabled) or 0xffffffff (enabled). Some of these flags are:

- UAC bypass
- Self deletion
- Demo mode (the famous "THIS IS A DEMO VERSION!!!" message box)
- Hide service

Timers (Offset 0x34, Length 0x8)

Two timers are configurable, each one stored on a DWORD (4 bytes: days, hours, minutes and seconds):

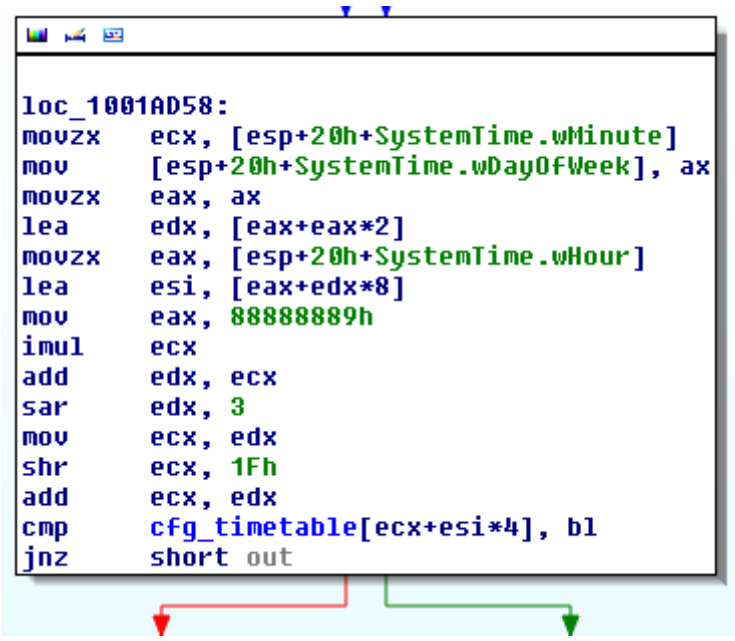
- Time between two connection attempts to the C&C
- Sleep time

Timetable (Offset 0x3c, Length 0x2a0)

This timetable defines when the malware is allowed to contact its C&C. There are 672 bytes, each one representing fifteen minutes, from Monday 00:00 AM to Sunday 11:45 PM. The malware is only allowed to communicate when the byte value is not zero.

This is a very interesting feature to hide malicious traffic under the amount of legitimate traffic during a normal working day, by allowing the malware to only communicate between 08:00 AM and 05:00 PM from Monday to Friday.

Practically, a function is called before any attempt to reach the C&C, and waits until the good time slot is reached.



Custom DNS servers (Offset 0x2dc, Length 0x10)

Four custom DNS servers can be provided in the configuration to resolve the C&C domain names instead of the system configured DNS. Various samples we encountered contained the following four DNS addresses, which seem to be a default value in the builder:

- 8.8.8.8 (Google)
- 61.139.2.69 (ChinaNet)
- 202.98.96.68 (ChinaNet)
- 205.252.144.228 (Beyond the Network America)

C&C Servers (Offset 0x2ec, Length 0x110)

Four different C&C servers can be configured, each one represented by the following structure:

```
struct cc {
        WORD type;
        WORD port;
        CHAR address[0x40];
}
```

The type is a bit field indicating which protocols to use for the communication (Raw TCP / Raw UDP / HTTP).

URLs (Offset 0x3fc, Length 0x200)

Four URLs are available to provide a fallback mechanism in case the four C&C servers are not available anymore. These URLs are therefore joined by the malware, which will parse the web page, looking for DZKS and DZJS patterns. The string between these patterns is in fact a structure describing a new C&C server, each nibble being encoded in base16 (using characters from A to P).

Let's take the following string as an example:

```
DZKSHAAAJDADBDCDHDOCADOCADOCBDDZJS
```

The encoded string is HAAAJDADBDCDHDOCADOCADOCBD. The decoding can be performed by taking the characters as pairs and substracting 0x41 to each ascii value. The decoded structure is:

```
07 00 39 30 31 32 37 2e 30 2e 30 2e 31              ..90127.0.0.1

Type = 0x7
Port = 0x3039 (12345)
C&C = 127.0.0.1
```

Proxys (Offset 0x5fc, Length 0x310)

Four proxys can also be configured to ensure the RAT can reach its C&C servers. The following structure is used:

```
struct proxy {
        WORD type;
        WORD port;
        BYTE proxy[0x40];
        BYTE user[0x40];
        BYTE passwd[0x40];
}
```

Persistence (Offset 0x90c, Length 0x800)

The following four parameters are present (strings of 0x200 bytes):

- Installation directory
- Service name
- Service display name
- Service description

Currently unused strings (Offset 0x110c, Length 0x400)

Two currently unused strings are present in each configuration file. In all the samples we found, these two strings were set to "1234". We'll describe these strings more precisely in an upcoming blog post.

Evolution

The PlugX RAT has not evolved so much from early 2012 to Q1 2013. However, a new version has been spotted in the wild since Q2 2013, and this will be the topic of my next blog post. Stay tuned

Back to Blog