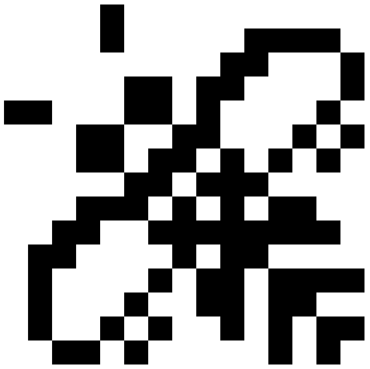
A few notes on SECONDDATE's C&C protocol

laanwj.github.io/2016/09/17/seconddate-cnc.html



Laanwj's blog

Randomness

Blog About
SECONDDATE is the most well-known of the spy toys in the EQGRP dump. It is a Man-in-the-Middle attack tool that is installed on intermediate routes. It can be used to fake DNS replies as well as inject HTTP redirects. This intercept article does a great job of explaining the operational side.

SECONDDATE (abbreviated as *SD* from here on) is a module in all of the leaked implant frameworks:

- BLATSTING
- BUZZDIRECTON
- BANANAGLEE
- TURBO (PANDAROCK, POLARSCORE, POLARTX)

Of both the BLATSTING and BUZZDIRECTION modules it is the module with the largest code size. This alone puts up some challenges for reverse-engineering. I have mainly looked at the BLATSTING module as I'm furthest with analyzing that framework, but at first sight it appears that the SD modules for all these frameworks are based on the same underlying code - much of the binary and its data matches.

The primary reason for its large size appears to be that it contains a copy of the open source regular expression library PCRE1 as well as a weighty part of a C library required to support that, statically linked.

Pretty soon after examining the dump I noticed that many of the SD Listening Post executables still had some debugging symbols. After dumping these with dwarf_to_c, a tool I wrote a few years back for a completely different reason, I surmised that the whole C&C protocol was already revealed there.

But only recently after completing the BLATSTING simulation and experimenting with it a it I could know this for sure.

Header

The SD C&C protocol uses encrypted UDP packets with a fixed payload size of `1060` bytes, contents defined by:

```
typedef struct
{
  int munge; /* +0x0 */
  int magic; /* +0x4 */
  int checkSum; /* +0x8 */
  /* De-munging and checksum XORing starts here */
  unsigned char encCounter[8]; /* +0xc */
  /* Decryption starts here */
  int encMagic; /* +0x14 */
  int sequenceNum; /* +0x18 */
  unsigned int type; /* +0x1c */
  int errCode; /* +0x20 */
  unsigned int logTime; /* +0x24 */
  union
  {
    RULE_TYPE rule;
    SD_INFO info;
    LOG_ENTRY log[15];
  }; /* +0x28 */
} SD_PDUTYPE;
```

Commands

Of these header fields `type` specifies the type of packet, which in turn specifies which of the union fields should be used to access the parameters. These were not part of the debug information but fairly easy to discover playing with the LP program:

type	description	parameter
0x01	ping	*N/A*
0x02	set rule	rule
0x03	enable rule	rule
0x04	disable rule	rule

type	description	parameter
0x05	get rule	rule
0x06	get log	log
0x07	get info	info
0x08	uninstall	*N/A*
0x09	hello	*N/A*
0x0a	clear log	*N/A*

Rule definitions

Rule definitions are the bread and butter of this software. They define what to match on (TCP or UDP packets), a regular expression on the packet contents, and a specification of what to inject in turn. They can be set using the `set rule` command, enabled and disabled using the `enable rule` and `disable rule` command respectively, and requested using `get rule`.

```
typedef struct
{
  unsigned char enabled; /* +0x0 */
  unsigned char checkHTTPGET; /* +0x1 */
  unsigned char checkPattern; /* +0x2 */
  unsigned char tcpFlags; /* +0x3 */
  unsigned char injectflag; /* +0x4 */
  unsigned int index; /* +0x8 */
  unsigned int start_index; /* +0xc */
  unsigned int stop_index; /* +0x10 */
  unsigned int tagOffset; /* +0x14 */
  unsigned int u_timestamp; /* +0x18 */
  unsigned int e_timestamp; /* +0x1c */
  unsigned int srcAddrFilter; /* +0x20 */
  unsigned int srcAddrFilterMask; /* +0x24 */
  unsigned int dstAddrFilter; /* +0x28 */
  unsigned int dstAddrFilterMask; /* +0x2c */
  unsigned int protocolFilter; /* +0x30 */
  short unsigned int srcPortFilter; /* +0x34 */
  short unsigned int dstPortFilter; /* +0x36 */
  unsigned int minInterval; /* +0x38 */
  unsigned int maxInjections; /* +0x3c */
  unsigned int injectWindow; /* +0x40 */
  unsigned int injectLen; /* +0x44 */
  unsigned int currentInjections; /* +0x48 */
  unsigned int totalInjections; /* +0x4c */
  unsigned int totalMisses; /* +0x50 */
  unsigned int nextInjectTime; /* +0x54 */
  unsigned int injectWindowEnd; /* +0x58 */
  unsigned char pattern[256]; /* +0x5c */
  unsigned char inject[512]; /* +0x15c */
} RULE_TYPE;
```

Info structure

Some basic information about the implant can be requested using the `get info` command:

```
typedef struct
{
  unsigned int version; /* +0x0 */
  unsigned int logEntries; /* +0x4 */
  unsigned int ruleCount; /* +0x8 */
  unsigned int timeStamp; /* +0xc */
} SD_INFO;
```

Log structure

The implant keeps a small log of the last matches and injections that it did (likely in a circular buffer). This can be requested using the `get log` command and wiped with the `clear log` command.

```
typedef struct LOG_ENTRY_
{
  unsigned int index; /* +0x0 */
  unsigned int srcAddr; /* +0x4 */
  unsigned int dstAddr; /* +0x8 */
  short unsigned int srcPort; /* +0xc */
  short unsigned int dstPort; /* +0xe */
  unsigned int timeStamp; /* +0x10 */
  unsigned int rule; /* +0x14 */
  char protocol; /* +0x18 */
  char dataBuffer[40]; /* +0x19 */
} LOG_ENTRY;
```

Decryption

Received UDPv4 packets go through the following flow:

- After receiving a packet first the size is checked against `1060` (like in BLATSTING, the ports don't matter, though the LP always uses source port `32768` and a random destination port).

- If that matches, the first two fields read as big-endian, `munge` and `magic` are checked via `munge + magic - 0x61e57cc6 == 0`. If these match it will proceed to the next steps.

- Then a simple checksum is computed and checked (simply XOR together all 4-byte words), and a XOR de-munging step is done. Munging is the term that the Equation Group uses for simple obfuscation. Anyhow a `xorkey = munge - 0x61e57cc6` is computed and the entire packet is XORed with the 4-byte word.

- Then, the packet is decrypted using RC6 in CTR (counter mode)3, instead of OFB mode as in BLATSTING C&C. The counter is part of every packet, as header field `encCounter`.

- `encMagic` is checked to be `0x9e 0x1a 0x83 0x3a`. If this matches, the packet is dispatched for command processing.

It does not look like the packets are authenticated in any way. There is no MAC. This is usually a bad idea, but even more with crypto modes such as CCM that use the underlying block cipher as a stream cipher. Bits can be flipped for a lot of fun as long as you don't forget to update the checksum.

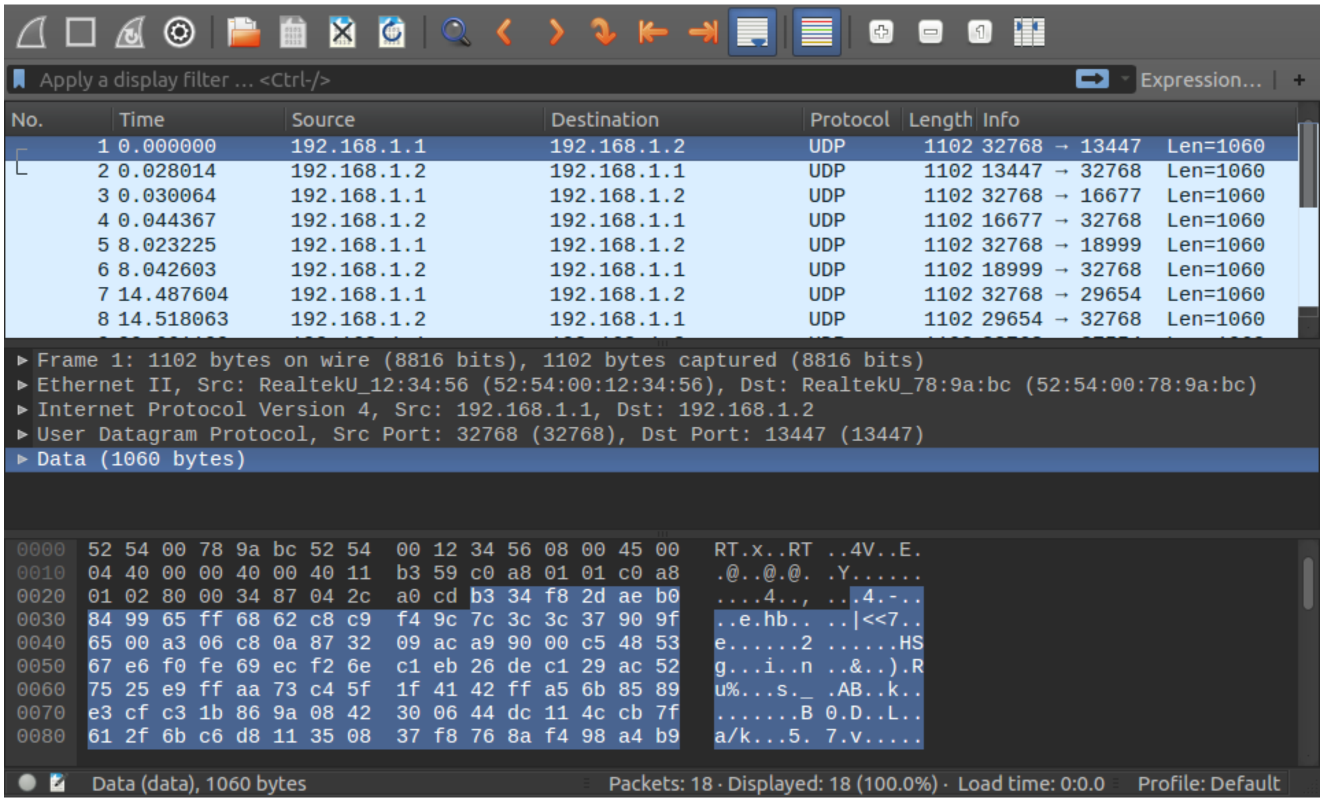
Again, if you were hoping to find evidence of unworldly cryptographic magic here, you'll be disappointed.

Hardcoded key

Curiously all the samples in the dump use a hardcoded RC6 key: `e2be1a91aed43cf0398acdd13f7bc3f9` 2. I have only verified this for sure with the BLATSTING implementation but this key can be found (in binary, not as hex string) in all implants of SD as well as all LP executables to control it, except for BUZZDIRECTION.

There appears to be no way to change or override the key externally, although there is of course the possibility that the "keying system", which is missing from this dump for all of the frameworks, changes the key in the binary.

Conclusion



```
No.  Time         Source          Destination     Protocol  Length  Info
  1  0.000000     192.168.1.1     192.168.1.2     UDP       1102    32768 → 13447  Len=1060
  2  0.028014     192.168.1.2     192.168.1.1     UDP       1102    13447 → 32768  Len=1060
  3  0.030064     192.168.1.1     192.168.1.2     UDP       1102    32768 → 16677  Len=1060
  4  0.044367     192.168.1.2     192.168.1.1     UDP       1102    16677 → 32768  Len=1060
  5  8.023225     192.168.1.1     192.168.1.2     UDP       1102    32768 → 18999  Len=1060
  6  8.042603     192.168.1.2     192.168.1.1     UDP       1102    18999 → 32768  Len=1060
  7  14.487604    192.168.1.1     192.168.1.2     UDP       1102    32768 → 29654  Len=1060
  8  14.518063    192.168.1.2     192.168.1.1     UDP       1102    29654 → 32768  Len=1060

▸ Frame 1: 1102 bytes on wire (8816 bits), 1102 bytes captured (8816 bits)
▸ Ethernet II, Src: RealtekU_12:34:56 (52:54:00:12:34:56), Dst: RealtekU_78:9a:bc (52:54:00:78:9a:bc)
▸ Internet Protocol Version 4, Src: 192.168.1.1, Dst: 192.168.1.2
▸ User Datagram Protocol, Src Port: 32768 (32768), Dst Port: 13447 (13447)
▸ Data (1060 bytes)

0000  52 54 00 78 9a bc 52 54  00 12 34 56 08 00 45 00   RT.x..RT ..4V..E.
0010  04 40 00 00 40 00 40 11  b3 59 c0 a8 01 01 c0 a8   .@..@.@. .Y......
0020  01 02 80 00 34 87 04 2c  a0 cd b3 34 f8 2d ae b0   ....4.., ...4.-..
0030  84 99 65 ff 68 62 c8 c9  f4 9c 7c 3c 3c 37 90 9f   ..e.hb.. ..|<<7..
0040  65 00 a3 06 c8 0a 87 32  09 ac a9 90 00 c5 48 53   e......2 ......HS
0050  67 e6 f0 fe 69 ec f2 6e  c1 eb 26 de c1 29 ac 52   g...i..n ..&..).R
0060  75 25 e9 ff aa 73 c4 5f  1f 41 42 ff a5 6b 85 89   u%...s._ .AB..k..
0070  e3 cf c3 1b 86 9a 08 42  30 06 44 dc 11 4c cb 7f   .......B 0.D..L..
0080  61 2f 6b c6 d8 11 35 08  37 f8 76 8a f4 98 a4 b9   a/k...5. 7.v.....

Data (data), 1060 bytes    Packets: 18 · Displayed: 18 (100.0%) · Load time: 0:0.0    Profile: Default
```

Screenshot of a Wireshark dump of SECONDDATE C&C packets. Would be nice to have a dissector… Here's the accompanying seconddate.cap file.

I hope this article gives enough information to detect SD C&C usage in the wild. I may do a LP transcript or demo of using the implant later on. Also I think the rule structure deserves more attention.

Footnotes

1. I found at least `regexp_compile` and `regexp_match` with the same API as documented by PCRE. It is used to look for the for the famous string `ace02468bdf13579` (see the Intercept article linked at the beginning) match HTTP in TCP streams with `^GET.*(?:/ |\\.(?:htm|asp|php)).*\\r\\n` , as well as look for user-defined patterns.

2. No hosts on the internet respond to SD packets under this key. That has been checked. Also it’s all out in the open, no one knows how long this leak has been floating around prior to publication, and I doubt that I’m the first one to discover this. So it is safe to reveal. But it is fun (or scary) to think that once this may have been the key to the castle for controlling a significant part of the internet.

3. Details: It is standard 20-round RC6, with blocksize 128 bits. Example C implementation. Counter is padded to input block as `<8b counter (big endian)><8b 0>`, then encrypted to generate keystream. Initial value of counter is in the packet, after each block the counter is increased with 1.

Written on September 17, 2016

Tags: eqgrp malware
Filed under Reverse-engineering