Latest TeslaCrypt Ransomware Borrows Code From Carberp Trojan

researchcenter.paloaltonetworks.com/2015/10/latest-teslacrypt-ransomware-borrows-code-from-carberp-trojan/

Josh Grunzweig October 9, 2015

By Josh Grunzweig

October 9, 2015 at 5:00 AM

Category: Malware, Ransomware, Threat Prevention, Unit 42

Tags: AutoFocus, Carberp, CryptoLocker, CryptoWall, TeslaCrypt, Trojan, WildFire

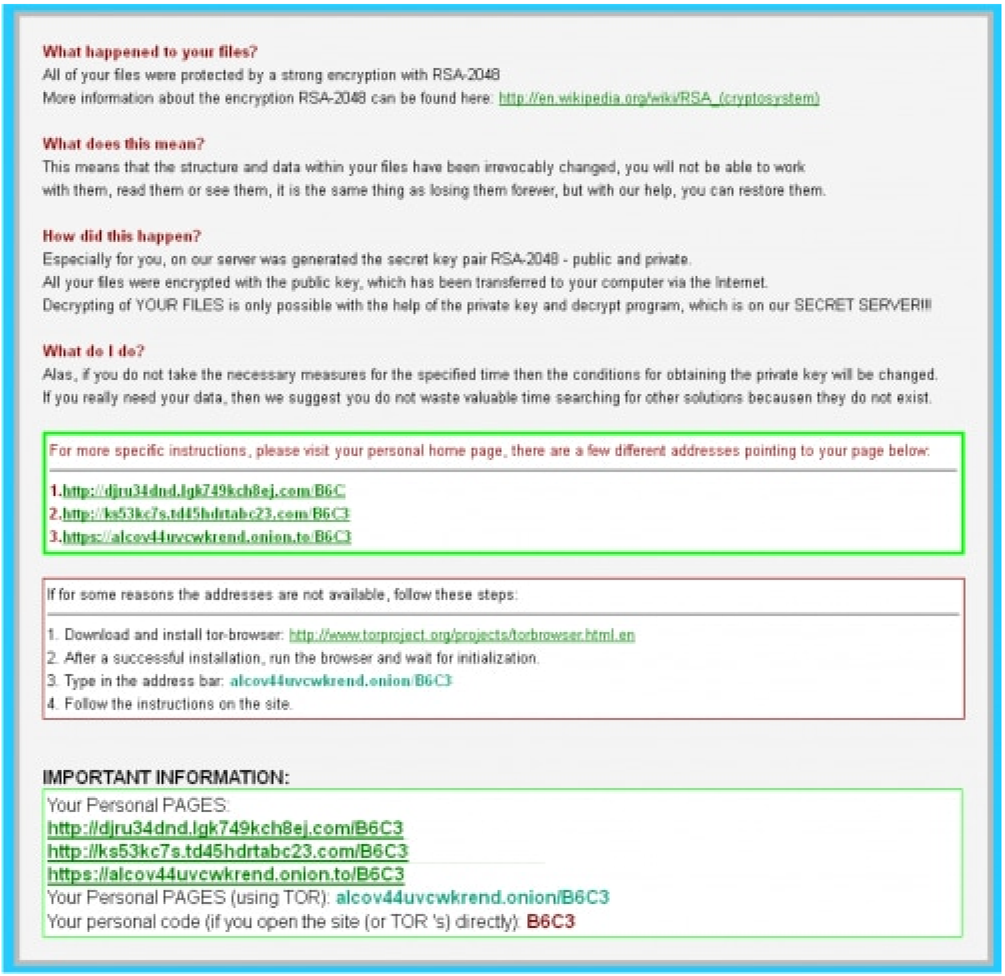
In recent weeks, we have noticed changes in the TeslaCrypt ransomware malware family's code base. OpenDNS recently discussed some of these changes regarding the encryption techniques in this newest variant. While reverse engineering the underlying code of these samples we discovered that the author of of TeslaCrypt borrowed code from the Carberp malware family in order to obfuscate strings and dynamically load libraries/functions.

TeslaCrypt was discovered in February 2015, and has been actively developed since its initial release. The TeslaCrypt family is known as ransomware—a type of malware that encrypts a victim's files then demands a form of payment in exchange for the decryption key. Ransomware has been very lucrative for attackers, and an ongoing challenge for consumers and businesses alike. Malware like TeslaCrypt is often delivered via spam emails or exploit kits. A recent takedown of multiple domains used by the popular Angler exploit kit estimated that as much as $60 million in revenue was generated annually by ransomware alone.

TeslaCrypt has historically been known to borrow code or other features from various ransomware families. Older variants used a notification screen that looked nearly identical to the one used by the CryptoLocker malware family.

Figure 1. Locker notification for old variants of TeslaCrypt

The latest versions of TeslaCrypt attempt to mimic the popular CryptoWall malware family.


What happened to your files?
All of your files were protected by a strong encryption with RSA-2048
More information about the encryption RSA-2048 can be found here: http://en.wikipedia.org/wiki/RSA_(cryptosystem)

What does this mean?
This means that the structure and data within your files have been irrevocably changed, you will not be able to work with them, read them or see them, it is the same thing as losing them forever, but with our help, you can restore them.

How did this happen?
Especially for you, on our server was generated the secret key pair RSA-2048 - public and private.
All your files were encrypted with the public key, which has been transferred to your computer via the Internet.
Decrypting of YOUR FILES is only possible with the help of the private key and decrypt program, which is on our SECRET SERVER!!!

What do I do?
Alas, if you do not take the necessary measures for the specified time then the conditions for obtaining the private key will be changed.
If you really need your data, then we suggest you do not waste valuable time searching for other solutions becausen they do not exist.

For more specific instructions, please visit your personal home page, there are a few different addresses pointing to your page below:

1.http://djru34dnd.lgk749kch8ej.com/B6C
2.http://ks53kc7s.td45hdrtabc23.com/B6C3
3.https://alcov44uvcwkrend.onion.to/B6C3

If for some reasons the addresses are not available, follow these steps:

1. Download and install tor-browser: http://www.torproject.org/projects/torbrowser.html.en
2. After a successful installation, run the browser and wait for initialization.
3. Type in the address bar: alcov44uvcwkrend.onion/B6C3
4. Follow the instructions on the site.

IMPORTANT INFORMATION:
Your Personal PAGES:
http://djru34dnd.lgk749kch8ej.com/B6C3
http://ks53kc7s.td45hdrtabc23.com/B6C3
https://alcov44uvcwkrend.onion.to/B6C3
Your Personal PAGES (using TOR): alcov44uvcwkrend.onion/B6C3
Your personal code (if you open the site (or TOR 's) directly): B6C3

Figure 2. Locker notification for new variants of TeslaCrypt

As we can see from the figures, the author of TeslaCrypt has no reservations about re-using code where possible. Starting in late September, the newest version of TeslaCrypt was introduced and it included multiple updates. One of these updates involved modifications to how the victims’ files were encrypted, which was discussed by OpenDNS in their blog post.

However, when looking at the underlying code, a number of other changes caught our eye, including string obfuscation previously unseen in TelsaCrypt.

```
.text:0041DA40                 push    148             ; size_t
.text:0041DA45                 push    offset a9ommrpgdqsfovk ; "9omMRPGDQsfOvKM4fbGs56uwGHkwC6Sx14UW3sq"...
.text:0041DA4A                 push    esi             ; void *
.text:0041DA4B                 mov     [ebp+len], 148
.text:0041DA52                 call    _memcpy
.text:0041DA57                 lea     ecx, [ebp+len]
.text:0041DA5A                 push    ecx
.text:0041DA5B                 push    esi
.text:0041DA5C                 call    string_decrypt
.text:0041DA61                 mov     esi, c2_url_0
.text:0041DA67                 push    60h             ; size_t
.text:0041DA69                 push    offset aSs5glbid5scn0s ; "ss5G1BId5ScN0sLOkRx5aNCbwurD13kiiuii7eg"...
.text:0041DA6E                 push    esi             ; void *
.text:0041DA6F                 mov     [ebp+len], 96
.text:0041DA76                 call    _memcpy
.text:0041DA7B                 lea     edx, [ebp+len]
.text:0041DA7E                 push    edx
.text:0041DA7F                 push    esi
.text:0041DA80                 call    string_decrypt
.text:0041DA85                 mov     esi, c2_url_1
.text:0041DA8B                 push    54h             ; size_t
.text:0041DA8D                 push    offset aXkmFoRp6cq223f ; "xkM/FO+RP6cQ223ff6AvJTksja807U/gjvPTLUj"...
.text:0041DA92                 push    esi             ; void *
.text:0041DA93                 mov     [ebp+len], 84
.text:0041DA9A                 call    _memcpy
.text:0041DA9F                 lea     eax, [ebp+len]
.text:0041DAA2                 push    eax
.text:0041DAA3                 push    esi
.text:0041DAA4                 call    string_decrypt
.text:0041DAA9                 mov     esi, c2_url_2
```

Figure 3. TeslaCrypt string obfuscation

Upon further review, we discovered that these strings are encrypted using the RC2 cryptographic algorithm, using a static key of 'sdflk35jghs'. The initialization vector is generated by removing the first and last 4 characters, not counting the base64 padding characters. This process is shown below.

```
       Original String: ss5G1BId5ScN0sLOkRx5aNCbwurD13kiiuii7egAaW4cGinoQdicPlacsRlt7S0cVyv6JRWFECpuk3oBYlS5Se+XMwb126==
         First 4 Bytes: ss5G
          Last 4 Bytes: b126
 Initialization Vector: ss5Gb126
Base64 String to Decrypt: 1BId5ScN0sLOkRx5aNCbwurD13kiiuii7egAaW4cGinoQdicPlacsRlt7S0cVyv6JRWFECpuk3oBYlS5Se+XMw==
```

Figure 4. TeslaCrypt IV and data parsing

While examining the Carberp source code, we discovered this exact code. Carberp was a popular banking Trojan discovered in late 2011. Its main functionality included stealing online banking credentials, keystroke logging, and capturing data from various applications.

In mid-2013, the source code to Carberp was posted for sale on an underground Russian forum. A number of weeks following this posting, the source code was leaked to the general public. This allowed any individual to modify or copy the source code to this banking Trojan, which the author of TeslaCrypt appears to have done.

```
PCHAR RC2Crypt_PackEncodedBuffer(PCHAR Buf, DWORD BufSize, PCHAR IV)
{
  PCHAR Result = STR::Alloc(BufSize + 8);
  if (Result == NULL)
    return NULL;

  PCHAR P = Result;

  STR::Copy(IV, P, 0, 4);
  P += 4;

  PCHAR End = Buf + BufSize;
  while (End > Buf && *(End - 1) == '=') End--;

  STR::Copy(Buf, P, 0, End - Buf);
  P += End - Buf;

  STR::Copy(IV, P, 4, 4);
  P += 4;

  STR::Copy(End, P, 0, BufSize - (End - Buf));

  return Result;
}
```

Figure 5. Carberp string parsing prior to decryption

Looking further into the underlying code of TeslaCrypt, we found that the author has also implemented dynamic library and function loading.

```
.text:00412A0D                 push    edi
.text:00412A0E                 mov     edi, [ebp+var_4]
.text:00412A11                 push    0CEBF17E6h      ; dwProcNameHash
.text:00412A16                 push    2               ; dwModule
.text:00412A18                 push    0               ; Dll
.text:00412A1A                 call    GetProcAddressEx
.text:00412A1F                 add     esp, 0Ch
.text:00412A22                 push    ebx
.text:00412A23                 push    esi
.text:00412A24                 push    0
.text:00412A26                 push    1
.text:00412A28                 push    0
.text:00412A2A                 push    edi
.text:00412A2B                 call    eax
.text:00412A2D                 test    eax, eax
.text:00412A2F                 js      short loc_412A37
.text:00412A31                 mov     ecx, [ebx]
.text:00412A33                 mov     byte ptr [esi+ecx], 0
.text:00412A37
.text:00412A37 loc_412A37:                             ; CODE XREF: sub_4129F0+3F↑j
.text:00412A37                 mov     edi, [ebp+var_4]
.text:00412A3A                 push    0D4B3D42h       ; dwProcNameHash
.text:00412A3F                 push    2               ; dwModule
.text:00412A41                 push    0               ; Dll
.text:00412A43                 call    GetProcAddressEx
.text:00412A48                 add     esp, 0Ch
.text:00412A4B                 push    edi
.text:00412A4C                 call    eax
.text:00412A4E                 mov     edi, [ebp+iv]
.text:00412A51                 push    72760BB8h       ; dwProcNameHash
.text:00412A56                 push    2               ; dwModule
.text:00412A58                 push    0               ; Dll
.text:00412A5A                 call    GetProcAddressEx
.text:00412A5F                 add     esp, 0Ch
.text:00412A62                 push    0
.text:00412A64                 push    edi
.text:00412A65                 call    eax
```

Figure 6. Dynamic function loading in TeslaCrypt

Sure enough, this code was also copied from Carberp's source code. Hashes used to identify function are generated via the following algorithm:

```
ULONG CalcHash(char *str)
{
    ULONG hash = 0;
    char *s = str;

    while (*s)
    {
        hash = ((hash << 7) & (ULONG)-1) | (hash >> (32 - 7));
        hash = hash ^ *s;

        s++;
    }

    return hash;
}
```

Figure 7. Hashing algorithm

In order to assist analysts and reverse-engineers working on the latest version of TeslaCrypt, please refer to the script shown in Figure 8 that will attempt to automatically convert API hashes to their actual function names.

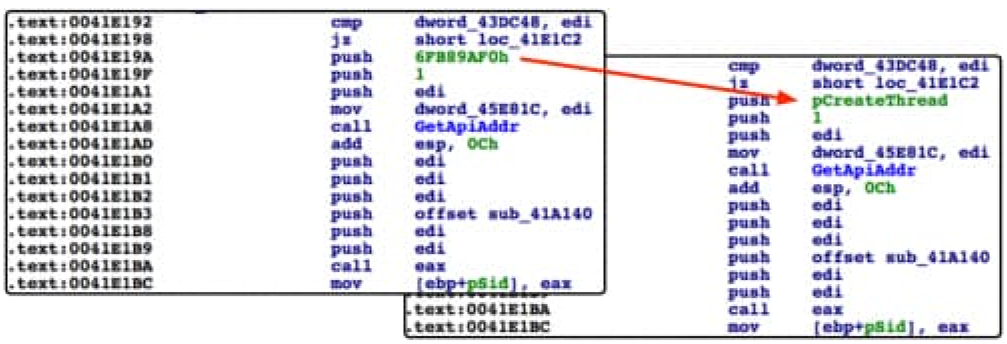


Figure 8. Results of running IDAPython script

Overall, it appears that the author of TeslaCrypt has continued their history of re-using code and functionality from other malware families. By using the string obfuscation and dynamic API loading functionality from Carberp, it makes reverse-engineering and simple static analysis slightly more difficult. However, as the Carberp source code is so widely known by the security community, the author may have inadvertently made detection of these samples easier. This is the tradeoff of re-using code from other malware families. It's certainly quicker and easier to do, but may result in easier detection by security software.

All new variants of the TeslaCrypt malware family samples are properly classified as malicious by Palo Alto Networks WildFire. AutoFocus users can find more information on samples and indicators related to this attack by viewing the TeslaCrypt tag.