Tofsee Spambot features .ch DGA - Reversal and Countermesaures

govcert.ch/blog/tofsee-spambot-features-.ch-dga-reversal-and-countermesaures/

GovCERT.ch

- Homepage
- GovCERT.ch Blog

Close

Blog Posts

Recently published blog posts:

- 16.09.2022Unflattening ConfuserEx .NET Code in IDA
- 12.12.2021Zero-Day Exploit Targeting Popular Java Library Log4j
- 09.03.2021Exchange Vulnerability 2021

Blog Archive

Go to the blog archive and browse all previous blog posts we have published so far.

RSS Feed

Subscribe to the GovCERT.ch blog RSS feed to stay up to date and get notified about new blog posts.

- Whitepapers

Close

Whitepapers

Recently published whitepapers:

- 16.09.2022Unflattening ConfuserEx .NET Code in IDA
- 23.09.2019Trickbot - An analysis of data collected from the botnet
- 03.03.2017Scripting IDA Debugger to Deobfuscate Nymaim

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 NCSC

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

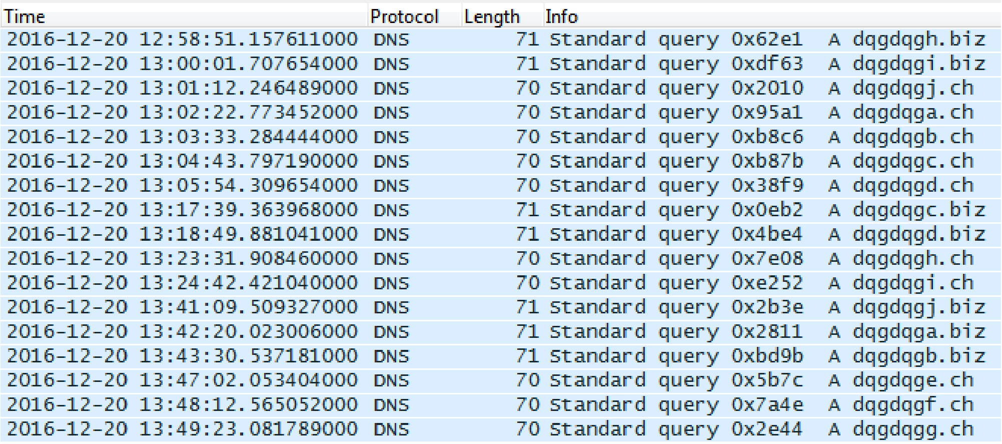
Encrypted Email

GovCERT.ch PGP-Key (preferred)
Alternative GovCERT.ch PGP Key (for older versions of PGP without Curve25519 support)
GovCERT.ch SMIME

- Statistics

Today we came across an interesting malware sample that appeared in our malware zoo. The malware, which we identified as Tofsee, has tried to spam out hundreds of emails within a couple of minutes. However, this wasn't the reason why it popped up on our radar (we analyze thousands of malware samples every single day, many of which are spambots too). The reason why this particular sample caught our attention were the domains queried by the malware. The domains appear to be algorithmically generated, and about half of the domains use the country code top level domain (ccTLD) of Switzerland:

Close



Time	Protocol	Length	Info
2016-12-20 12:58:51.157611000	DNS	71	Standard query 0x62e1 A dqgdqgh.biz
2016-12-20 13:00:01.707654000	DNS	71	Standard query 0xdf63 A dqgdqgi.biz
2016-12-20 13:01:12.246489000	DNS	70	Standard query 0x2010 A dqgdqgj.ch
2016-12-20 13:02:22.773452000	DNS	70	Standard query 0x95a1 A dqgdqga.ch
2016-12-20 13:03:33.284444000	DNS	70	Standard query 0xb8c6 A dqgdqgb.ch
2016-12-20 13:04:43.797190000	DNS	70	Standard query 0xb87b A dqgdqgc.ch
2016-12-20 13:05:54.309654000	DNS	70	Standard query 0x38f9 A dqgdqgd.ch
2016-12-20 13:17:39.363968000	DNS	71	Standard query 0x0eb2 A dqgdqgc.biz
2016-12-20 13:18:49.881041000	DNS	71	Standard query 0x4be4 A dqgdqgd.biz
2016-12-20 13:23:31.908460000	DNS	70	Standard query 0x7e08 A dqgdqgh.ch
2016-12-20 13:24:42.421040000	DNS	70	Standard query 0xe252 A dqgdqgi.ch
2016-12-20 13:41:09.509327000	DNS	71	Standard query 0x2b3e A dqgdqgj.biz
2016-12-20 13:42:20.023006000	DNS	71	Standard query 0x2811 A dqgdqga.biz
2016-12-20 13:43:30.537181000	DNS	71	Standard query 0xbd9b A dqgdqgb.biz
2016-12-20 13:47:02.053404000	DNS	70	Standard query 0x5b7c A dqgdqge.ch
2016-12-20 13:48:12.565052000	DNS	70	Standard query 0x7a4e A dqgdqgf.ch
2016-12-20 13:49:23.081789000	DNS	70	Standard query 0x2e44 A dqgdqgg.ch

Wireshark screenshot of Tofsee DNS queries (click to enlarge)

Domain generation algorithms (DGAs) that use the ccTLD for Switzerland are very rare. Gozi is currently the only malware covered by the DGArchive that uses .ch --- and only in 1 of over 90 different known configurations.

This blog post first describes the analysis of the DGA. We then give a reimplementation of the DGA in Python, as well as a list of the generated domains for the next 52 weeks. We conclude with measure that we took to deal with algorithmically generated .ch domains.

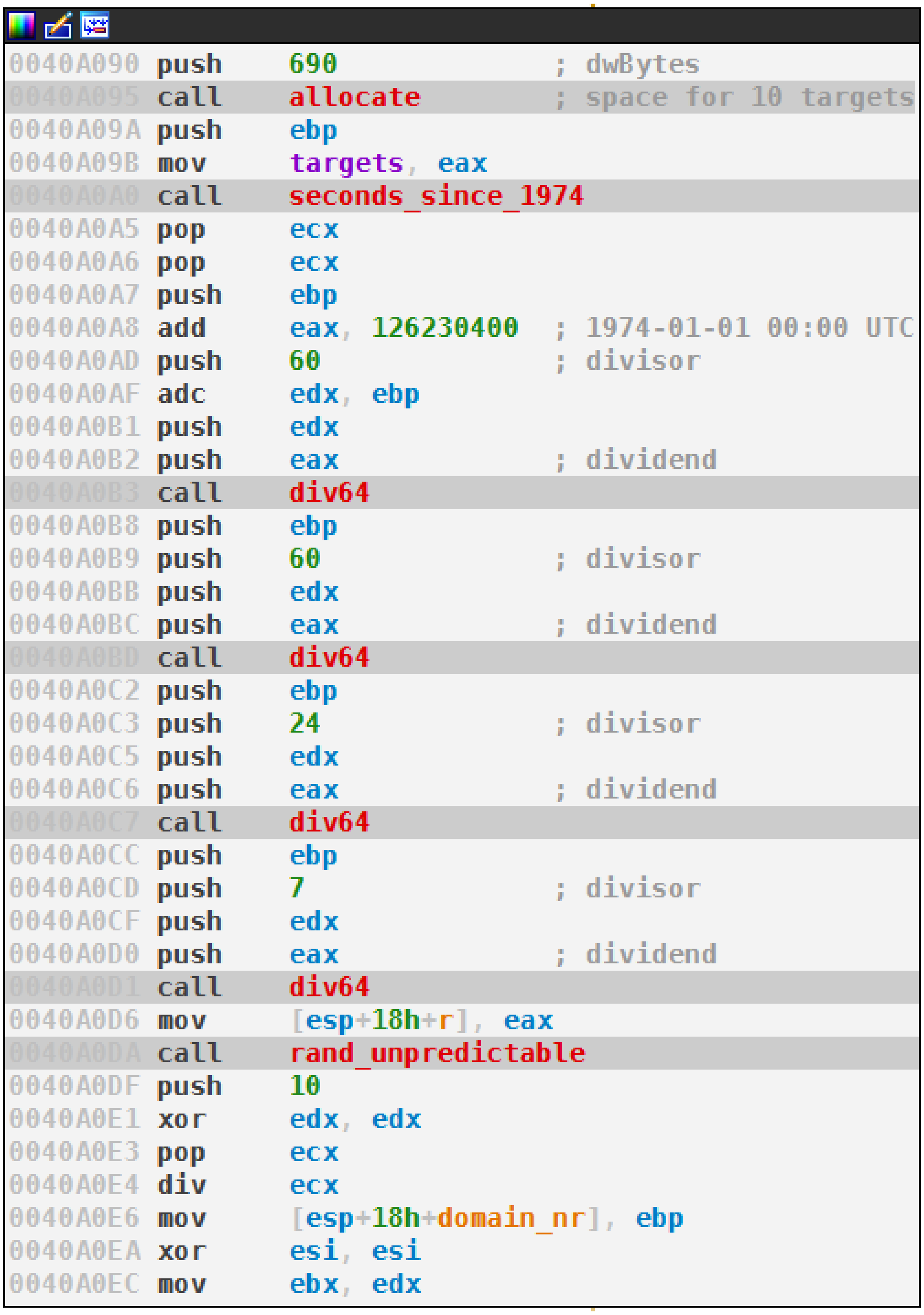
Analysis

We analyzed the following Tofsee sample, with a fairly recent compile timestamp of Fri, 16 Dec 2016 07:09:11:

```
md5     490e121113fbadd776ca270c6788c59a
sha     c294e79a5f0fbbffd535bb517f43bf69e9bbfb03
sha256  36704ec52701920451437a870e7d538eb409f50a4ae2f8231869500d1d6de159
```

Seeding

The following graph node show the seeding of the DGA. First, the number of seconds that have elapsed since 1 January 1974 are counted (offset 0x40A0A0). The difference between this date and the unix epoch --- 126230400 seconds --- is added to the result at 0x0040A0A8, effectively yielding the current unix time. It is unclear to us why the authors used this convoluted method to get the current time. The unix time then undergoes four integer division by 60,60,24 and 7, to get the number of weeks since epoch. This value is used as the seed for the upcoming domain generation algorithm. Domains are therefore valid for one week, starting on Thursday at midnight UTC.



```
0040A090 push    690                 ; dwBytes
0040A095 call    allocate            ; space for 10 targets
0040A09A push    ebp
0040A09B mov     targets, eax
0040A0A0 call    seconds_since_1974
0040A0A5 pop     ecx
0040A0A6 pop     ecx
0040A0A7 push    ebp
0040A0A8 add     eax, 126230400      ; 1974-01-01 00:00 UTC
0040A0AD push    60                  ; divisor
0040A0AF adc     edx, ebp
0040A0B1 push    edx
0040A0B2 push    eax                 ; dividend
0040A0B3 call    div64
0040A0B8 push    ebp
0040A0B9 push    60                  ; divisor
0040A0BB push    edx
0040A0BC push    eax                 ; dividend
0040A0BD call    div64
0040A0C2 push    ebp
0040A0C3 push    24                  ; divisor
0040A0C5 push    edx
0040A0C6 push    eax                 ; dividend
0040A0C7 call    div64
0040A0CC push    ebp
0040A0CD push    7                   ; divisor
0040A0CF push    edx
0040A0D0 push    eax                 ; dividend
0040A0D1 call    div64
0040A0D6 mov     [esp+18h+r], eax
0040A0DA call    rand_unpredictable
0040A0DF push    10
0040A0E1 xor     edx, edx
0040A0E3 pop     ecx
0040A0E4 div     ecx
0040A0E6 mov     [esp+18h+domain_nr], ebp
0040A0EA xor     esi, esi
0040A0EC mov     ebx, edx
```

Seeding of the DGA (click to enlarge)

Seeding also calls a pseudo random number generator (PRNG) and takes the result modulo 10 to get a value between 0 and 9. The random number generator is the standard linear congruential algorithm used by the Borland C/C++ compiler:

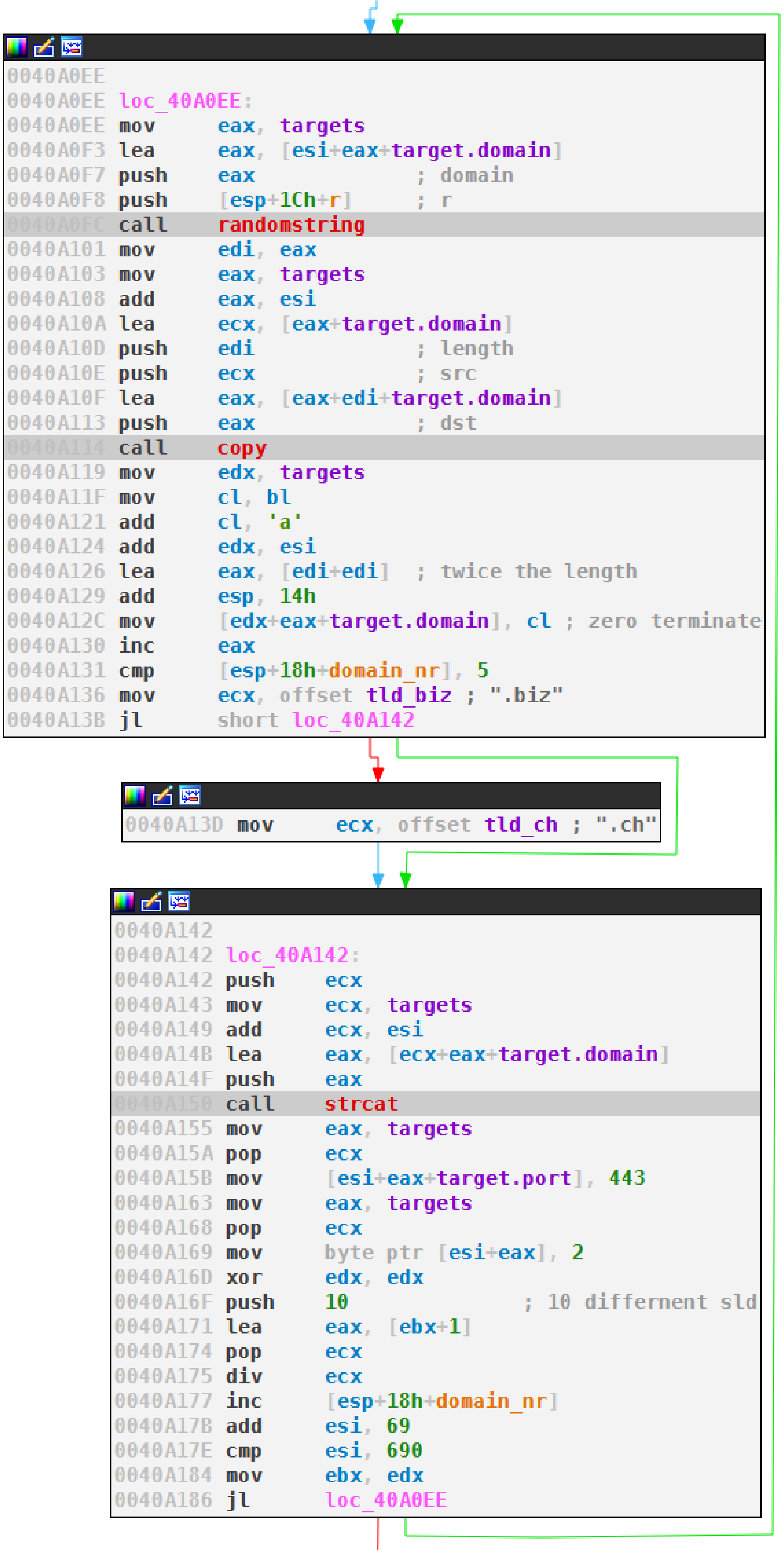
```
unsigned int rand()
{
    r2 = 22695477 * r2 + 1;
    return r2 >> 16;
}
```

The initial value of r2 is virtually unpredictable:

```
GetSystemTimeAsFileTime(&SystemTimeAsFileTime);
GetVolumeInformationA(0, 0, 4u, &VolumeSerialNumber, 0, 0, 0, 0);
r2 = (VolumeSerialNumber \
      ^ SystemTimeAsFileTime.dwHighDateTime \
      ^ GetTickCount()) & 0x7FFFFFFF;
```

DGA

The generated domains, along with the port 443 and an unknown constant 2, are stored 69 bytes apart in memory (called target in the next graph). In total 10 domains are generated:



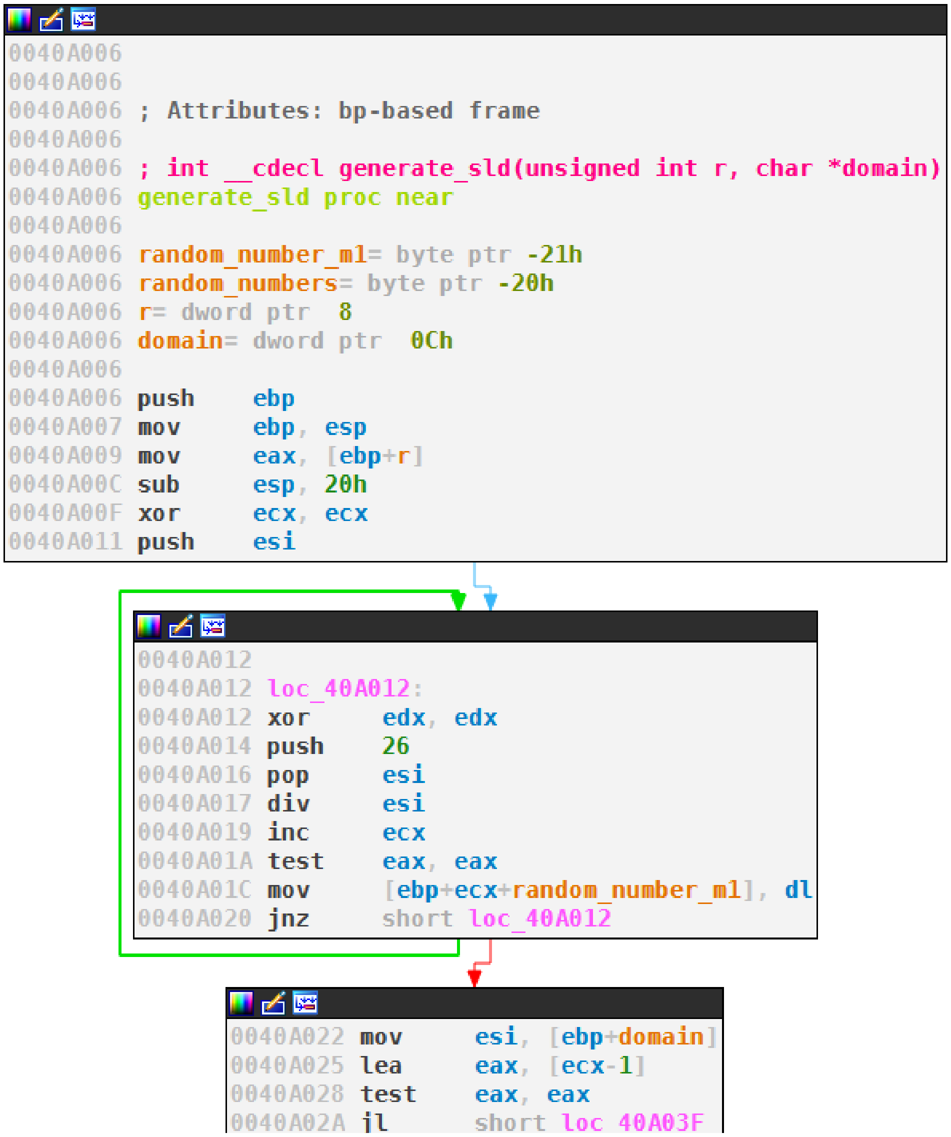
```
0040A0EE
0040A0EE loc_40A0EE:
0040A0EE mov     eax, targets
0040A0F3 lea     eax, [esi+eax+target.domain]
0040A0F7 push    eax                 ; domain
0040A0F8 push    [esp+1Ch+r]         ; r
0040A0FC call    randomstring
0040A101 mov     edi, eax
0040A103 mov     eax, targets
0040A108 add     eax, esi
0040A10A lea     ecx, [eax+target.domain]
0040A10D push    edi                 ; length
0040A10E push    ecx                 ; src
0040A10F lea     eax, [eax+edi+target.domain]
0040A113 push    eax                 ; dst
0040A114 call    copy
0040A119 mov     edx, targets
0040A11F mov     cl, bl
0040A121 add     cl, 'a'
0040A124 add     edx, esi
0040A126 lea     eax, [edi+edi]  ; twice the length
0040A129 add     esp, 14h
0040A12C mov     [edx+eax+target.domain], cl ; zero terminate
0040A130 inc     eax
0040A131 cmp     [esp+18h+domain_nr], 5
0040A136 mov     ecx, offset tld_biz ; ".biz"
0040A13B jl      short loc_40A142
```

```
0040A13D mov     ecx, offset tld_ch ; ".ch"
```

```
0040A142
0040A142 loc_40A142:
0040A142 push    ecx
0040A143 mov     ecx, targets
0040A149 add     ecx, esi
0040A14B lea     eax, [ecx+eax+target.domain]
0040A14F push    eax
0040A150 call    strcat
0040A155 mov     eax, targets
0040A15A pop     ecx
0040A15B mov     [esi+eax+target.port], 443
0040A163 mov     eax, targets
0040A168 pop     ecx
0040A169 mov     byte ptr [esi+eax], 2
0040A16D xor     edx, edx
0040A16F push    10                ; 10 different sld
0040A171 lea     eax, [ebx+1]
0040A174 pop     ecx
0040A175 div     ecx
0040A177 inc     [esp+18h+domain_nr]
0040A17B add     esi, 69
0040A17E cmp     esi, 690
0040A184 mov     ebx, edx
0040A186 jl      loc_40A0EE
```

Loop that generates 10 domains (click to enlarge)

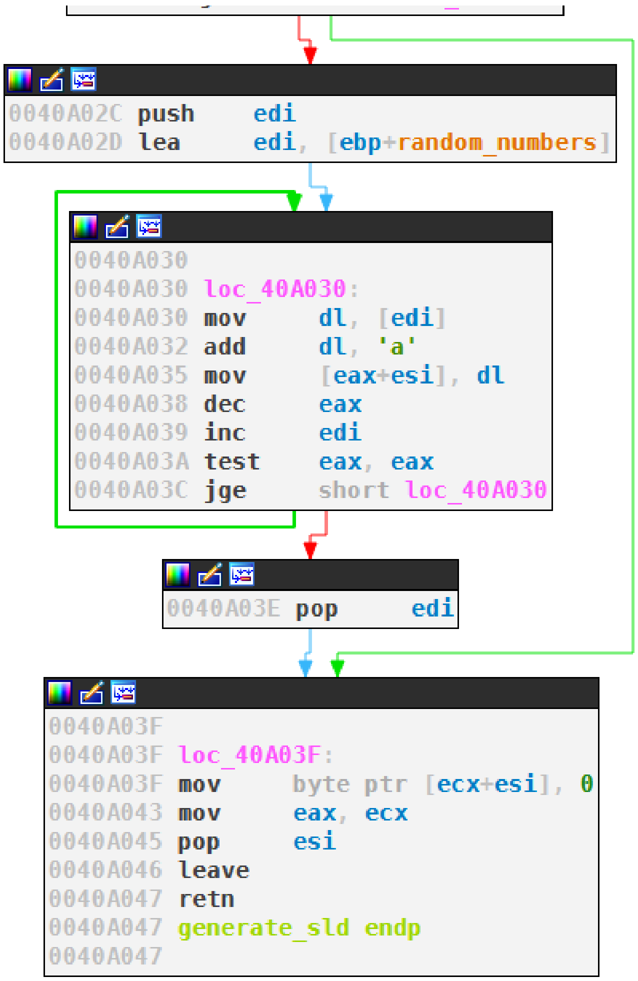
At offset 0x040A0FC a random string is generated based on the seed, i.e., number of weeks. We will come back to this routine later. The random string is repeated once at 0x040A114; so that, for example, drs becomes drsdrs.
A random letter between "a" and "j" is then appended to the string to complete the second level domain. The picked letter is based on the unpredictable PRNG shown in the seeding section above for the first generated sld. After that, the DGA picks the remaining letters in order (offsets 0x0040A16D to 0040A175). For example, if in the first iteration the letter "c" is appended, then the following iterations append "d","e","f","g","h","j","a" and finally "b".
The top level domain is set to .ch for the first five domains, and to .biz for the remaining five domains. For any given run of the malware this will result in exactly one tld per generated second level domain per run of the malware, although a second level domain will be paired with both top level domains through additional runs of the malware.
Finally, let's come back to the random string generation routine called at 0x040A0FC:



```
0040A006
0040A006
0040A006 ; Attributes: bp-based frame
0040A006
0040A006 ; int __cdecl generate_sld(unsigned int r, char *domain)
0040A006 generate_sld proc near
0040A006
0040A006 random_number_m1= byte ptr -21h
0040A006 random_numbers= byte ptr -20h
0040A006 r= dword ptr  8
0040A006 domain= dword ptr  0Ch
0040A006
0040A006 push    ebp
0040A007 mov     ebp, esp
0040A009 mov     eax, [ebp+r]
0040A00C sub     esp, 20h
0040A00F xor     ecx, ecx
0040A011 push    esi
```

```
0040A012
0040A012 loc_40A012:
0040A012 xor     edx, edx
0040A014 push    26
0040A016 pop     esi
0040A017 div     esi
0040A019 inc     ecx
0040A01A test    eax, eax
0040A01C mov     [ebp+ecx+random_number_m1], dl
0040A020 jnz     short loc_40A012
```

```
0040A022 mov     esi, [ebp+domain]
0040A025 lea     eax, [ecx-1]
0040A028 test    eax, eax
0040A02A jl      short loc_40A03F
```



```
0040A02C push    edi
0040A02D lea     edi, [ebp+random_numbers]
```

```
0040A030
0040A030 loc_40A030:
0040A030 mov     dl, [edi]
0040A032 add     dl, 'a'
0040A035 mov     [eax+esi], dl
0040A038 dec     eax
0040A039 inc     edi
0040A03A test    eax, eax
0040A03C jge     short loc_40A030
```

```
0040A03E pop     edi
```

```
0040A03F
0040A03F loc_40A03F:
0040A03F mov     byte ptr [ecx+esi], 0
0040A043 mov     eax, ecx
0040A045 pop     esi
0040A046 leave
0040A047 retn
0040A047 generate_sld endp
0040A047
```

Disassembly of the sld generation (click to enlarge)

This routine uses the seed r, i.e., the number of weeks since January 1st, 1970, to generate a random string as follows:

```
string = ""
DO
    string += r % 26 + 'a'
    r = r / 26            // integer division
WHILE r
string = reverse(string)
```

For example, as of December 20, 2016, the number of week since epoch is 2450. Dividing by 26 results in 94 with a remainder of 6, so the first letter is g. Dividing 94 by 26 results in 3 with remainder 16, so the second letter is q. Dividing 3 by 26 results in 0, so we append letter d and exit the loop. The random string gqd is reversed resulting in dqg.

Reimplementation

The following reimplementation of the DGA in Python prints all 20 potential domains for any given date. Please note that each actual run of the malware will only generate and test one domain per second level domain.

```
from datetime import datetime
import time
import argparse

def dga(r):
    domain = ""
    while True:
        domain += chr(r % 26 + ord('a'))
        r //= 26
        if not r:
            break
    for i in range(10):
        for tld in ['.biz', '.ch']:
            yield 2*domain[::-1] + chr(i + ord('a')) + tld

def printdomains(date):
    unixtimestamp = time.mktime(date.timetuple())
    seed = int(unixtimestamp // 60 // 60 // 24 // 7)

    for domain in dga(seed):
        print(domain)


if __name__=="__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("-d", "--date", help="date for which to generate
domains")
    args = parser.parse_args()
    if args.date:
        d = datetime.strptime(args.date, "%Y-%m-%d")
    else:
        d = datetime.now()
    printdomains(d)
```

List of Domains

The following table lists all generated domains of the next 52 weeks. The domains are given as brace expansions, so dqgdqg{a..j}.{ch,biz} stands for 20 different domains. All times are given in CET.

start	end	domains
2016-12-15 01:00:00	2016-12-22 00:59:59	dqgdqg{a..j}.{ch,biz}
2016-12-22 01:00:00	2016-12-29 00:59:59	dqhdqh{a..j}.{ch,biz}
2016-12-29 01:00:00	2017-01-05 00:59:59	dqidqi{a..j}.{ch,biz}
2017-01-05 01:00:00	2017-01-12 00:59:59	dqjdqj{a..j}.{ch,biz}

start	end	domains
2017-01-12 01:00:00	2017-01-19 00:59:59	dqkdqk{a..j}.{ch,biz}
2017-01-19 01:00:00	2017-01-26 00:59:59	dqldql{a..j}.{ch,biz}
2017-01-26 01:00:00	2017-02-02 00:59:59	dqmdqm{a..j}.{ch,biz}
2017-02-02 01:00:00	2017-02-09 00:59:59	dqndqn{a..j}.{ch,biz}
2017-02-09 01:00:00	2017-02-16 00:59:59	dqodqo{a..j}.{ch,biz}
2017-02-16 01:00:00	2017-02-23 00:59:59	dqpdqp{a..j}.{ch,biz}
2017-02-23 01:00:00	2017-03-02 00:59:59	dqqdqq{a..j}.{ch,biz}
2017-03-02 01:00:00	2017-03-09 00:59:59	dqrdqr{a..j}.{ch,biz}
2017-03-09 01:00:00	2017-03-16 00:59:59	dqsdqs{a..j}.{ch,biz}
2017-03-16 01:00:00	2017-03-23 00:59:59	dqtdqt{a..j}.{ch,biz}
2017-03-23 01:00:00	2017-03-30 01:59:59	dqudqu{a..j}.{ch,biz}
2017-03-30 02:00:00	2017-04-06 01:59:59	dqvdqv{a..j}.{ch,biz}
2017-04-06 02:00:00	2017-04-13 01:59:59	dqwdqw{a..j}.{ch,biz}
2017-04-13 02:00:00	2017-04-20 01:59:59	dqxdqx{a..j}.{ch,biz}
2017-04-20 02:00:00	2017-04-27 01:59:59	dqydqy{a..j}.{ch,biz}
2017-04-27 02:00:00	2017-05-04 01:59:59	dqzdqz{a..j}.{ch,biz}
2017-05-04 02:00:00	2017-05-11 01:59:59	dradra{a..j}.{ch,biz}
2017-05-11 02:00:00	2017-05-18 01:59:59	drbdrb{a..j}.{ch,biz}
2017-05-18 02:00:00	2017-05-25 01:59:59	drcdrc{a..j}.{ch,biz}
2017-05-25 02:00:00	2017-06-01 01:59:59	drddrd{a..j}.{ch,biz}
2017-06-01 02:00:00	2017-06-08 01:59:59	dredre{a..j}.{ch,biz}
2017-06-08 02:00:00	2017-06-15 01:59:59	drfdrf{a..j}.{ch,biz}
2017-06-15 02:00:00	2017-06-22 01:59:59	drgdrg{a..j}.{ch,biz}
2017-06-22 02:00:00	2017-06-29 01:59:59	drhdrh{a..j}.{ch,biz}
2017-06-29 02:00:00	2017-07-06 01:59:59	dridri{a..j}.{ch,biz}

start	end	domains
2017-07-06 02:00:00	2017-07-13 01:59:59	drjdrj{a..j}.{ch,biz}
2017-07-13 02:00:00	2017-07-20 01:59:59	drkdrk{a..j}.{ch,biz}
2017-07-20 02:00:00	2017-07-27 01:59:59	drldrl{a..j}.{ch,biz}
2017-07-27 02:00:00	2017-08-03 01:59:59	drmdrm{a..j}.{ch,biz}
2017-08-03 02:00:00	2017-08-10 01:59:59	drndrn{a..j}.{ch,biz}
2017-08-10 02:00:00	2017-08-17 01:59:59	drodro{a..j}.{ch,biz}
2017-08-17 02:00:00	2017-08-24 01:59:59	drpdrp{a..j}.{ch,biz}
2017-08-24 02:00:00	2017-08-31 01:59:59	drqdrq{a..j}.{ch,biz}
2017-08-31 02:00:00	2017-09-07 01:59:59	drrdrr{a..j}.{ch,biz}
2017-09-07 02:00:00	2017-09-14 01:59:59	drsdrs{a..j}.{ch,biz}
2017-09-14 02:00:00	2017-09-21 01:59:59	drtdrt{a..j}.{ch,biz}
2017-09-21 02:00:00	2017-09-28 01:59:59	drudru{a..j}.{ch,biz}
2017-09-28 02:00:00	2017-10-05 01:59:59	drvdrv{a..j}.{ch,biz}
2017-10-05 02:00:00	2017-10-12 01:59:59	drwdrw{a..j}.{ch,biz}
2017-10-12 02:00:00	2017-10-19 01:59:59	drxdrx{a..j}.{ch,biz}
2017-10-19 02:00:00	2017-10-26 01:59:59	drydry{a..j}.{ch,biz}
2017-10-26 02:00:00	2017-11-02 00:59:59	drzdrz{a..j}.{ch,biz}
2017-11-02 01:00:00	2017-11-09 00:59:59	dsadsa{a..j}.{ch,biz}
2017-11-09 01:00:00	2017-11-16 00:59:59	dsbdsb{a..j}.{ch,biz}
2017-11-16 01:00:00	2017-11-23 00:59:59	dscdsc{a..j}.{ch,biz}
2017-11-23 01:00:00	2017-11-30 00:59:59	dsddsd{a..j}.{ch,biz}
2017-11-30 01:00:00	2017-12-07 00:59:59	dsedse{a..j}.{ch,biz}
2017-12-07 01:00:00	2017-12-14 00:59:59	dsfdsf{a..j}.{ch,biz}

Actions taken

To prevent that the Tofsee botnet operators are able to abuse the Swiss domain name space (ccTLD .ch) for hosting their botnet Command&Control infrastructure (C&C), we have discussed further actions with the registry of ccTLD .ch (SWITCH). Together with SWITCH and the Registrar of Last Resort (RoLR), all possible DGA domain name combinations have been set to non registrable at the registry level. It is therefore not possible to register any of the DGA domain names for the next 12 months.