The Defective Domain Generation Algorithm of BazarLoader

johannesbader.ch/blog/the-buggy-dga-of-bazarbackdoor/

Edit 2020-07-19: Cybereason published an excellent article A Bazar of Tricks: Following Team9's Development Cycles. The article shows that the DGA is part of Bazar Loader, which will try to download Bazar Backdoor. I therefore renamed most instances of BazarBackdoor to BazarLoader.

When I analyzed the domain generation algorithm of BazarLoader, I noticed a sample that generates bizarre "domains":

```
^efggkzjhggm.bazaar
]`egkjzeggkl.bazaar
_`eigkzegigm.bazaar
^`ggilzeigin.bazaar
bceeijbhgeil.bazaar
_acgkjzfegkl.bazaar
a`gggkaeiggm.bazaar
`cehimzhghio.bazaar
``ceikzeeeim.bazaar
`edgjlzjfgjn.bazaar
_ccghjzheghl.bazaar
a`eijjaegijl.bazaar
^aegjkzfggjm.bazaar
a`geikaeieim.bazaar
_dghhkziihhm.bazaar
```

Two things are obviously wrong:

1. There is no top level domain `.bazaar`. There is a Persian tld .بازار which translates to bazaar, but that won't work of course.
2. Some second level domains contain special characters which makes them invalid too.

The first error is easy to explain: the authors meant to use `.bazar`, which is a valid EmerDNS domain. The second mistake is more interesting. The authors must have noticed the occasional special characters too. But they probably couldn't find the root cause and instead programmed a workaround that fixes some, but not all, characters.

Here is the sample with the broken DGA that I looked at:

MD5
18d635a8ca7caefb4f4513650a31efc9

SHA1
d555233122a277fb89797ab2293efbe2a0c75f7f

SHA256
2e99ed535a9f73bafab151ec409de04c953a0187cb8e4063317617befa09068d

Size
377 KB (386224 Bytes)

Compile Timestamp
2020-06-17 09:20:56 UTC

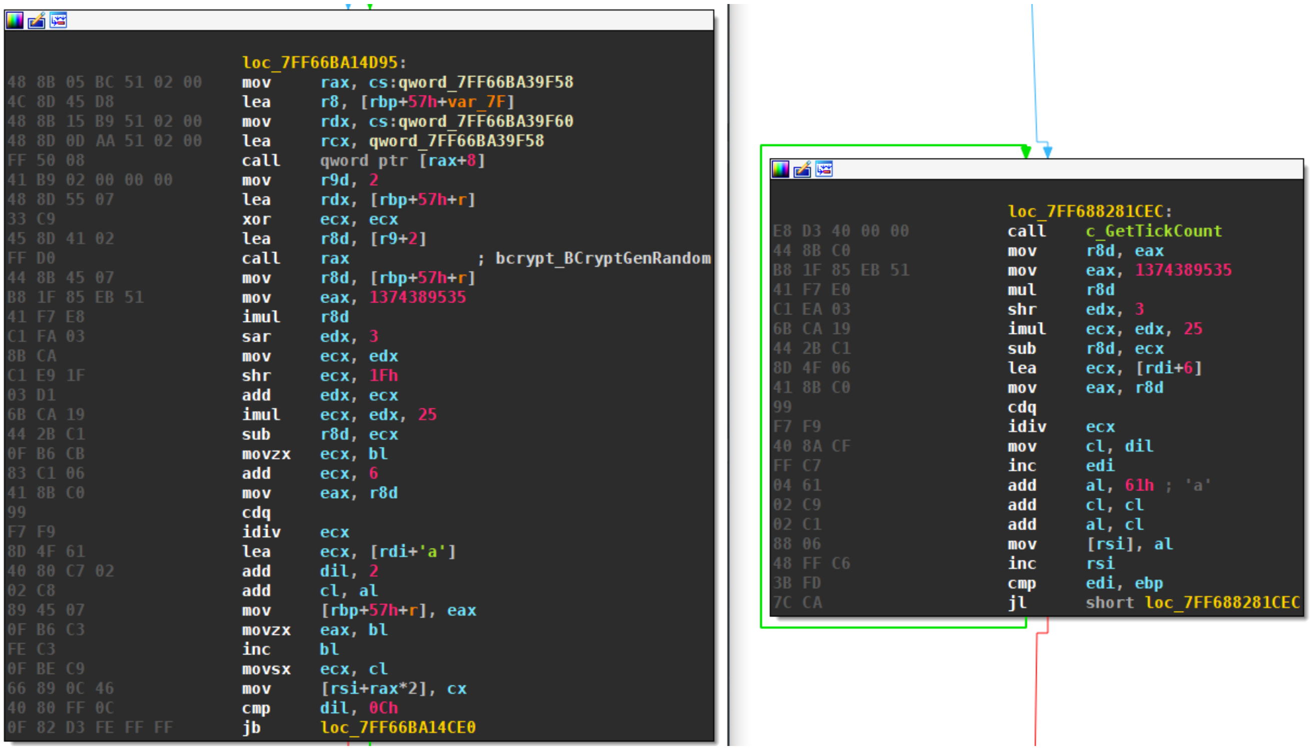
Links
VirusTotal

Filenames
DD45.exe, Preview_Report.exe (VirusTotal)

Detections

Virustotal: 41/76 as of 2020-07-09 13:47:45 - Trojan.Trickster.Gen (ALYac), Trojan.Win32.Mansabo.4!c (AegisLab), Trojan:Win32/Mansabo.e7acfbbd (Alibaba), Trojan/Win32.Mansabo (Antiy-AVL), Trojan.Mansabo (CAT-QuickHeal), Trojan.Win32.Mansabo.fef (Kaspersky), Trojan:Win32/Trickbot.A!Cert (Microsoft), TrojanSpy.Win64.TRICKBOT.ENJ (TrendMicro), TrojanSpy.Win64.TRICKBOT.ENJ (TrendMicro-HouseCall), Trojan.Mansabo (VBA32), Trojan.Win32.Mansabo.fef (ZoneAlarm)

The domain generation algorithm in this faulty version is the same as the one documented here. The only place that is different is shown in the following screenshot comparison. The faulty DGA is on the left, the fixed on the right. Can you spot the problem?



```
                                 loc_7FF66BA14D95:
48 8B 05 BC 51 02 00             mov     rax, cs:qword_7FF66BA39F58
4C 8D 45 D8                      lea     r8, [rbp+57h+var_7F]
48 8B 15 B9 51 02 00             mov     rdx, cs:qword_7FF66BA39F60
48 8D 0D AA 51 02 00             lea     rcx, qword_7FF66BA39F58
FF 50 08                         call    qword ptr [rax+8]
41 B9 02 00 00 00                mov     r9d, 2
48 8D 55 07                      lea     rdx, [rbp+57h+r]
33 C9                            xor     ecx, ecx
45 8D 41 02                      lea     r8d, [r9+2]
FF D0                            call    rax             ; bcrypt_BCryptGenRandom
44 8B 45 07                      mov     r8d, [rbp+57h+r]
B8 1F 85 EB 51                   mov     eax, 1374389535
41 F7 E8                         imul    r8d
C1 FA 03                         sar     edx, 3
8B CA                            mov     ecx, edx
C1 E9 1F                         shr     ecx, 1Fh
03 D1                            add     edx, ecx
6B CA 19                         imul    ecx, edx, 25
44 2B C1                         sub     r8d, ecx
0F B6 CB                         movzx   ecx, bl
83 C1 06                         add     ecx, 6
41 8B C0                         mov     eax, r8d
99                               cdq
F7 F9                            idiv    ecx
8D 4F 61                         lea     ecx, [rdi+'a']
40 80 C7 02                      add     dil, 2
02 C8                            add     cl, al
89 45 07                         mov     [rbp+57h+r], eax
0F B6 C3                         movzx   eax, bl
FE C3                            inc     bl
0F BE C9                         movsx   ecx, cl
66 89 0C 46                      mov     [rsi+rax*2], cx
40 80 FF 0C                      cmp     dil, 0Ch
0F 82 D3 FE FF FF                jb      loc_7FF66BA14CE0
```

```
                                 loc_7FF688281CEC:
E8 D3 40 00 00                   call    c_GetTickCount
44 8B C0                         mov     r8d, eax
B8 1F 85 EB 51                   mov     eax, 1374389535
41 F7 E0                         mul     r8d
C1 EA 03                         shr     edx, 3
6B CA 19                         imul    ecx, edx, 25
44 2B C1                         sub     r8d, ecx
8D 4F 06                         lea     ecx, [rdi+6]
41 8B C0                         mov     eax, r8d
99                               cdq
F7 F9                            idiv    ecx
40 8A CF                         mov     cl, dil
FF C7                            inc     edi
04 61                            add     al, 61h ; 'a'
02 C9                            add     cl, cl
02 C1                            add     al, cl
88 06                            mov     [rsi], al
48 FF C6                         inc     rsi
3B FD                            cmp     edi, ebp
7C CA                            jl      short loc_7FF688281CEC
```

The divisions by invariant multiplication are hard to read, but notice the right site being much shorter even tough the calculation is basically the same. This is because compiler optimization was able to strip some minor corrections that are only necessary for large numbers. Here the decompiled code after some renaming and cleaning up:

```
j_1 = 0;
i_1 = 0;
do
{
  r = 0;
  [...]
  bcrypt_BCryptGenRandom(0i64, &r, 4i64);
  offset_letter = i_1 + 'a';
  i_1 += 2;
  character = r % 25 / (j_1 + 6) + offset_letter;
  r = r % 25 / (j_1 + 6);
  j_2 = j_1++;
  *(szDomain + 2 * j_2) = character;
} while ( i_1 < 12u );
```

This is the same code as for the fixed DGA, except for how the random numbers are generated:

1. The faulty DGA generates 4 random bytes using a call to `BCryptGenRandom`.
2. The fixed DGA generates a random value with a call to `GetTickCount`, and extracting the lowest 15 bits.

The problem with the first approach is, that the number will be `0x80000000` or larger in 50% of the cases. Since it is a signed number, it becomes negative. And the remainder of a negative number for a positive divisor is negative. The fixed version doesn't have this problem, because the integer overflow does not happen. When extending the random number ranges to the negative, we get these character sets:

index	random number range	potential characters
0	-4–4	]^_`abcde
1	-3–3	`abcdef
2	-3–3	bcdefgh
3	-2–2	efghi
4	-2–2	ghijk
5	-2–2	ijklm

The malware authors used the following patch instead of fixing the integer overflow.

```
l = 6i64;
do {
    c = *(&szSeedStr[-6] + wDomain - a2) + *(wDomain - 6) - '0';
    *wDomain = c;
    if ( c < 'a' )
      *wDomain = 'z';
    ++wDomain;
    --l;
} while ( l );
```

The patch is an if condition that replaces characters below “a” — that includes all special characters generated by the faulty DGA — with “z”. This resolves the problem for the last half of the second level domain (in particular, the 7th and 8th letter, the rest are not affected by the bug). However, the first half of the second level domain remains unmodified.

The following Python reimplementation generates all possible domains for a given date. Note that due to the extended random ranges, there are about 55000 domains per month instead of 2160 for the fixed version. So even if the correct tld would have been used, then the number of domains would have been a problem for the attackers — as they have no way of predicting which ones are used in what order.

```python
import argparse
from datetime import datetime
from itertools import product


def dga(date):
    month = date.month
    year = date.year
    date_str = "{0:02d}{1:04d}".format(12-month, year-18)

    valid_chars = [
      "]^_`abcde",
      "`abcdef",
      "bcdefgh",
      "efghi",
      "ghijk",
      "ijklm"
    ]
    valid_chars = [list(_) for _ in valid_chars]
    for part1 in product(*valid_chars):
        domain = "".join(part1)
        for i, c in enumerate(part1):
            r = ord(c) + int(date_str[i])
            if r < ord('a'):
                domain += 'z'
            else:
                domain += chr(r)
        domain += ".bazaar"
        yield domain



if __name__=="__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("-d", "--date", help="date when domains are generated, e.g.,
2020-06-28")
    args = parser.parse_args()
    if args.date:
        d = datetime.strptime(args.date, "%Y-%m-%d")
    else:
        d = datetime.now()
    for domain in dga(d):
        print(domain)
```