# The DGA of Ranbyus

bin.re/blog/the-dga-of-ranbyus/

Ranbyus is a trojan that steals banking information — among other personal data. End of April 2015 I first noticed samples of Ranbyus that use a Domain Generation Algorithm (DGA) to generate the domains for its command and control (C2) traffic:

```
hcfoopojnuqxho.su
undrdsbhivryqn.tw
dkehliueofdued.net
mpuakxjqpscfpj.com
eelolbwmfmtkae.pw
noppsmyiijqujh.in
joxrsxwdybbgqb.me
(...)
```

In this post I show how the DGA works by reversing the following sample:

**filename**

`_RANDOMNUM_6_11__vozvrat.exe`

**filetype**
PE32 executable (GUI) Intel 80386, for MS Windows

**md5**
fa57f601402aab8168dea94c7c5f029f

**sha256**
dc4f3340ca8e623a5a77eb95411696fc25a7e6f5ef657ac9fd76eb4bc11c16b4

**malwr**
link

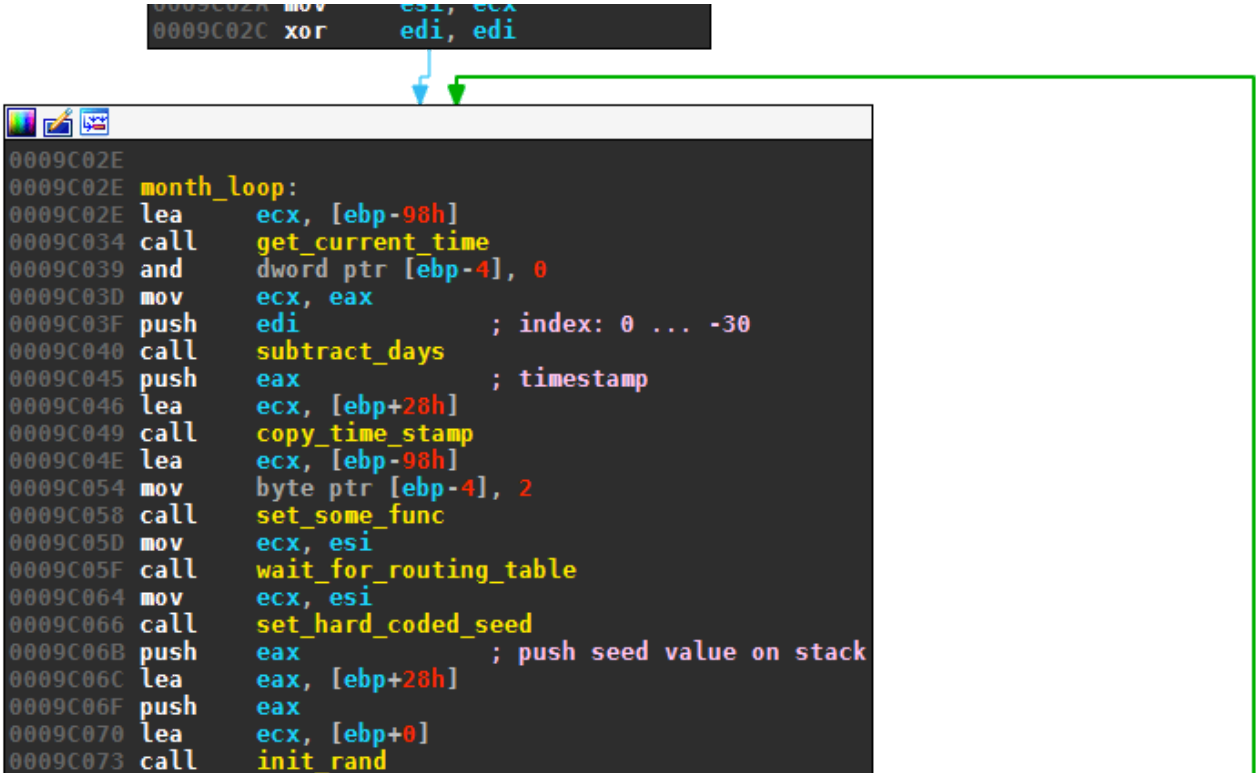
I focused my efforts exclusively on the domain generation part of Ranbyus. Refer to the blog posts of Aleksandr Matrosov here and here for an in-depth analysis of Ranbyus.

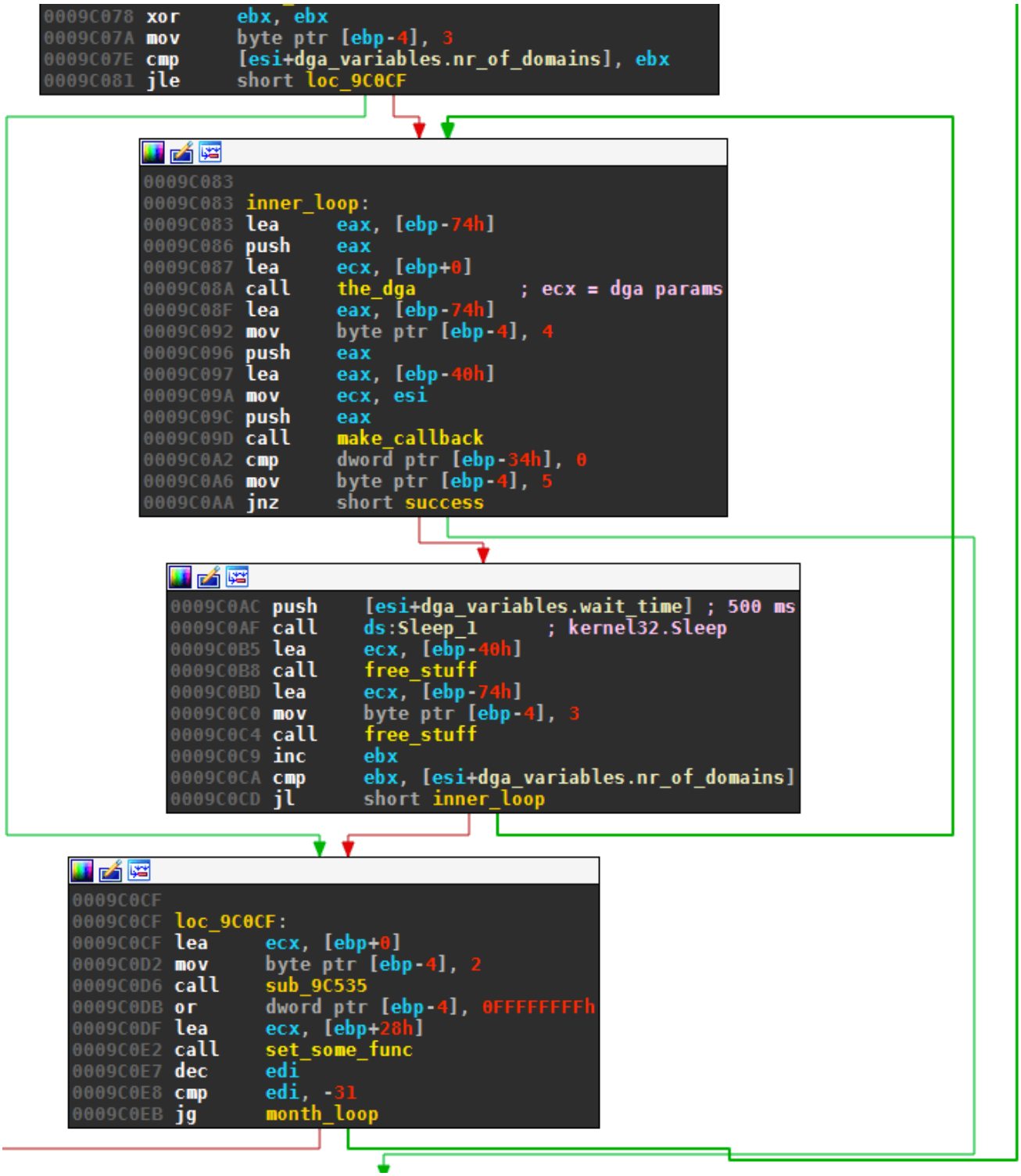
## Algorithm

This section shows the algorithm behind the domains of Ranbyus and its seeding and parametrization.

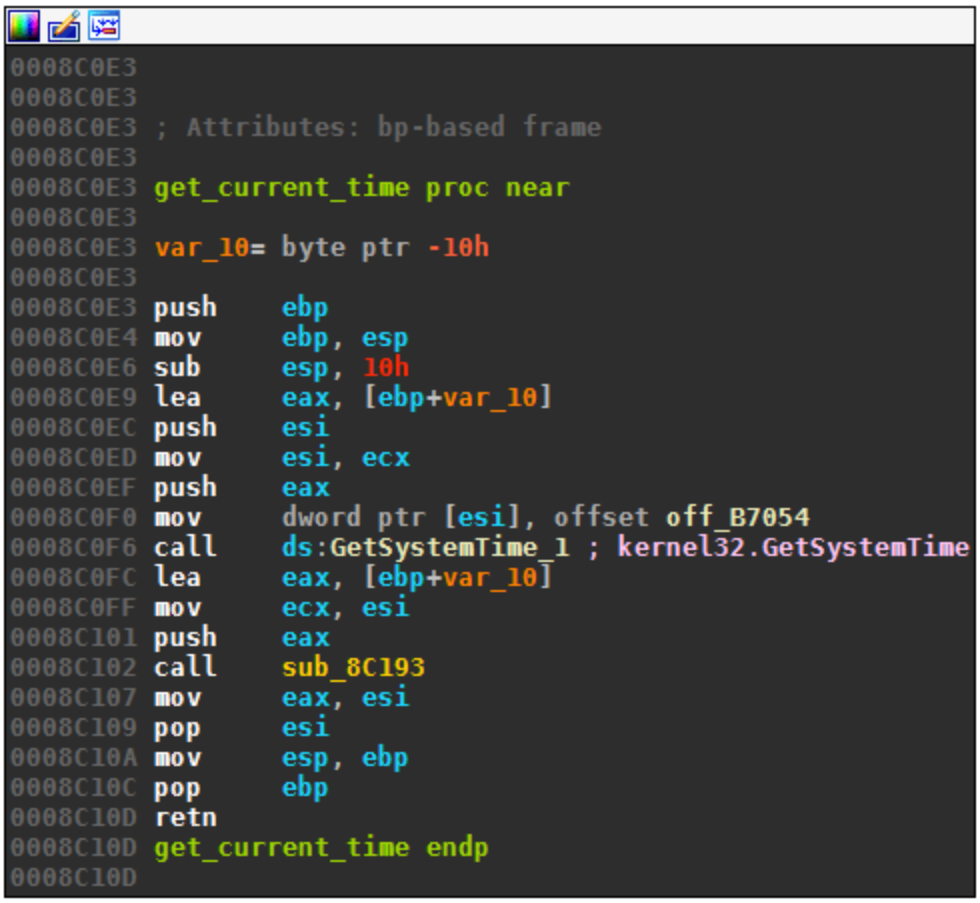
### Callback Loop

The next image represents the part of the Ranbyus that tries to find a valid C2 target. It consists of an outer loop (`month_loop`) and an inner loop. The register `edi` holds the index of the outer loop. It runs from 0 down to -30. The number of iterations for the inner loop is specified by a parameter of the DGA (set to 40 in all analysed samples):






The first act of the outer loop is to get the current time:



```
0008C0E3
0008C0E3
0008C0E3 ; Attributes: bp-based frame
0008C0E3
0008C0E3 get_current_time proc near
0008C0E3
0008C0E3 var_10= byte ptr -10h
0008C0E3
0008C0E3 push    ebp
0008C0E4 mov     ebp, esp
0008C0E6 sub     esp, 10h
0008C0E9 lea     eax, [ebp+var_10]
0008C0EC push    esi
0008C0ED mov     esi, ecx
0008C0EF push    eax
0008C0F0 mov     dword ptr [esi], offset off_B7054
0008C0F6 call    ds:GetSystemTime_1 ; kernel32.GetSystemTime
0008C0FC lea     eax, [ebp+var_10]
0008C0FF mov     ecx, esi
0008C101 push    eax
0008C102 call    sub_8C193
0008C107 mov     eax, esi
0008C109 pop     esi
0008C10A mov     esp, ebp
0008C10C pop     ebp
0008C10D retn
0008C10D get_current_time endp
0008C10D
```

Ranbyus then subtracts days from the current date according to the index of the outer loop:


The resulting date will be used to seed the DGA with a granularity of one day. In the first iteration, the DGA uses the current date. In the next iteration — when the index is -1 — yesterday's date is used. This continues up to 30 days in the past if need be. So even though the DGA generates a fresh set of domains every day, it also checks the domains of past days. This gives the DGA the benefit of fast changing domains in case domains get blocked or sinkholed, while at the same time enabling older domains to be used for up to one month if they still work.

The inner loop generates the domains for the day with `the_dga` and makes the callback. In case of failure, Ranbyus sleeps for `wait_time` milliseconds (500 in my samples) and retries up to `nr_of_domains` (40) different domains.

## DGA Parameters and Seed

Apart from the current date, the DGA is seeded with a hardcoded magic number:

```
0009C311 push    0B6354BC3h      ; hardcoded seed
```

The number of domains per day is hardcoded to 40:

```
0009BEFD push    40              ; nr of domains
0009BEFF pop     eax
0009BF00 mov     [edi+39h], eax
0009BF03 mov     [edi+dga_variables.nr_of_domains], eax
```

The wait time after a failed callback attempt is set to 500 ms:

```
0008F7EC push    500             ; sleep time
```

Ranbyus also uses a hard-coded list of top level domains:

```
seg001:000B765C 2E 69 6E 00                     tld_in          dd 'ni.'
seg001:000B7660 2E 6D 65 00                     tld_em          dd 'em.'
seg001:000B7664 2E 63 63 00                     tld_cc          dd 'cc.'
seg001:000B7668 2E 73 75 00                     tld_us          dd 'us.'
seg001:000B766C 2E 74 77 00                     tld_tw          dd 'wt.'
seg001:000B7670 2E 6E 65 74 00 00 00 00 tld_net         dd 'ten.', 0
seg001:000B7678 2E 63 6F 6D 00 00 00 00 tld_com         dd 'moc.', 0
seg001:000B7680 2E 70 77 00                     tld_pw          dd 'wp.'
seg001:000B7684 2E 6F 72 67 00 00 00 00 tld_org         dd 'gro.', 0
```

The top level domains are: *.in*, *.me*, *.cc*, *.su*, *.tw*, *.net*, *.com*, *.pw*, and *.org*. The last domain *.org* is never used due to a bug of the DGA. The top level domains are tested one after another (except the last one), starting at a day-dependent offset:

```
0009C755 mov     esi, [ebx+seed_struct.nr_tlds]
0009C758 call    get_day
0009C75D lea     ecx, [esi-1]    ; len(tlds) - 1
0009C760 cdq
0009C761 idiv    ecx
0009C763 mov     ecx, [esp+74h]
0009C767 pop     esi
0009C768 mov     [ebx+seed_struct.index], edx
```

The error of subtracting 1 from the modulus is repeated also when picking the letters of the second level domain.

## The DGA

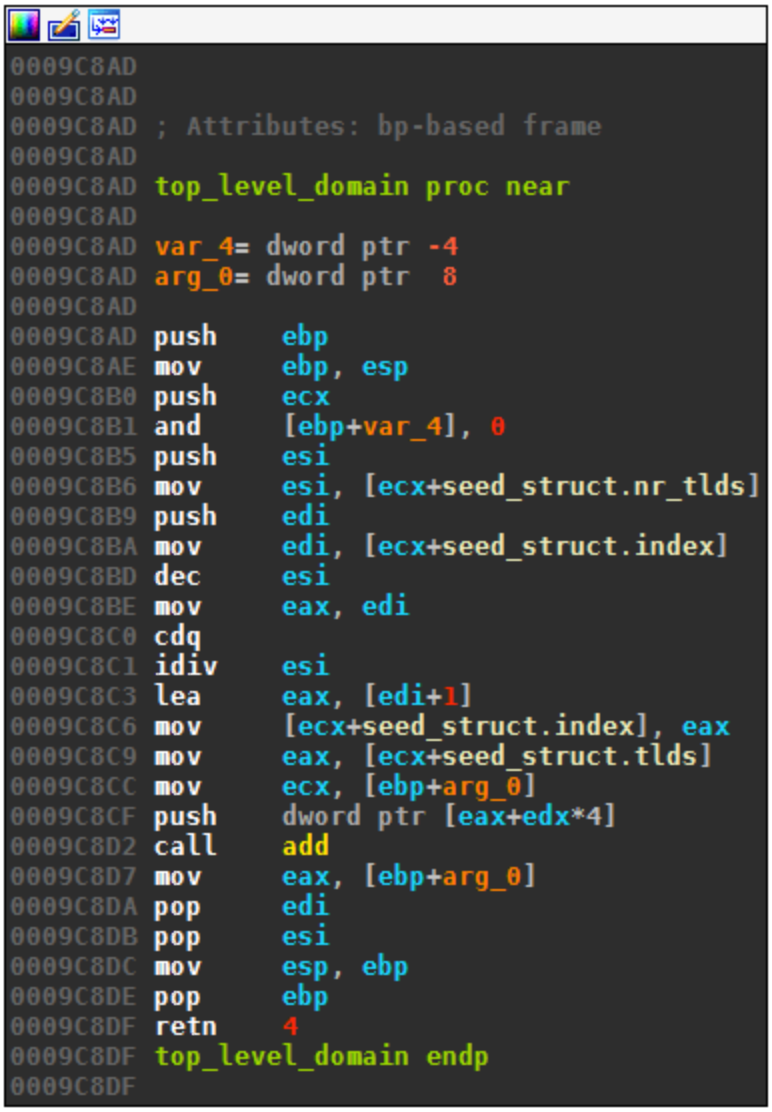
This is the disassembly of the DGA routine:

```
0009C848
0009C848
0009C848
0009C848 the_dga proc near
0009C848 mov     eax, offset loc_B4696
0009C84D call    stack_unrolling
0009C852 sub     esp, 6Ch
0009C855 and     dword ptr [ebp-10h], 0
0009C859 lea     eax, [ebp-78h]
0009C85C push    esi
0009C85D push    edi
0009C85E push    eax
0009C85F mov     edi, ecx
0009C861 call    top_level_domain
0009C866 mov     esi, eax
0009C868 and     dword ptr [ebp-4], 0
0009C86C lea     eax, [ebp-44h]
0009C86F push    eax
0009C870 mov     ecx, edi
0009C872 call    second_level_domain
0009C877 push    esi
0009C878 push    eax
0009C879 push    dword ptr [ebp+8]
0009C87C mov     byte ptr [ebp-4], 1
0009C880 call    sub_852F4
0009C885 add     esp, 0Ch
0009C888 lea     ecx, [ebp-44h]
0009C88B call    free_stuff
0009C890 lea     ecx, [ebp-78h]
0009C893 call    free_stuff
0009C898 mov     ecx, [ebp-0Ch]
0009C89B mov     eax, [ebp+8]
0009C89E pop     edi
0009C89F pop     esi
0009C8A0 mov     large fs:0, ecx
0009C8A7 mov     esp, ebp
0009C8A9 pop     ebp
0009C8AA retn    4
0009C8AA the_dga endp
0009C8AA
```

The subroutine generates domains in two independent parts:

1. the top level domain is picked from the hardcoded list shown above
2. the second level domain is generated.

The following disassembly shows how the top level domain is picked:



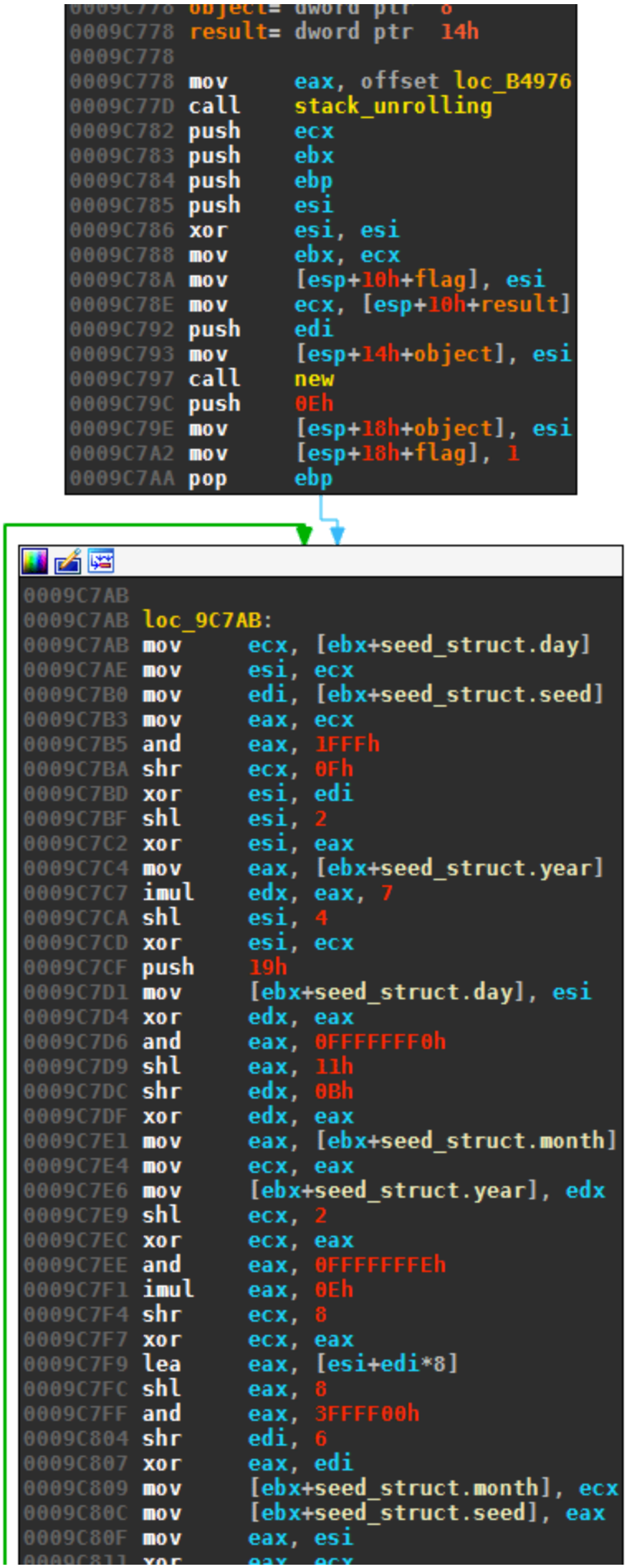
```
0009C8AD
0009C8AD
0009C8AD ; Attributes: bp-based frame
0009C8AD
0009C8AD top_level_domain proc near
0009C8AD
0009C8AD var_4= dword ptr -4
0009C8AD arg_0= dword ptr  8
0009C8AD
0009C8AD push    ebp
0009C8AE mov     ebp, esp
0009C8B0 push    ecx
0009C8B1 and     [ebp+var_4], 0
0009C8B5 push    esi
0009C8B6 mov     esi, [ecx+seed_struct.nr_tlds]
0009C8B9 push    edi
0009C8BA mov     edi, [ecx+seed_struct.index]
0009C8BD dec     esi
0009C8BE mov     eax, edi
0009C8C0 cdq
0009C8C1 idiv    esi
0009C8C3 lea     eax, [edi+1]
0009C8C6 mov     [ecx+seed_struct.index], eax
0009C8C9 mov     eax, [ecx+seed_struct.tlds]
0009C8CC mov     ecx, [ebp+arg_0]
0009C8CF push    dword ptr [eax+edx*4]
0009C8D2 call    add
0009C8D7 mov     eax, [ebp+arg_0]
0009C8DA pop     edi
0009C8DB pop     esi
0009C8DC mov     esp, ebp
0009C8DE pop     ebp
0009C8DF retn    4
0009C8DF top_level_domain endp
0009C8DF
```

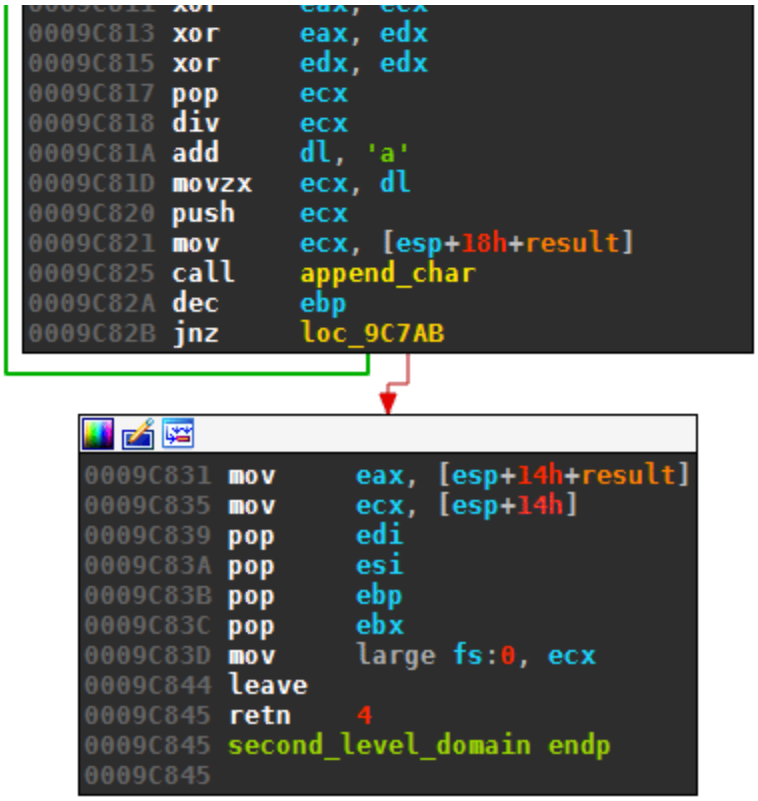
Starting at the day dependent offset determined earlier, the algorithm picks the domains in a circular fashion, omitting the last domain because the DGA wrongly subtracts one from the modulus.

```
[".in", ".me", ".cc", ".su", ".tw", ".net", ".com", ".pw", ".org"][offset % (9-1)]
offset++
```

The disassembly for the second level domain looks as follows. It generates 14 different letters based on the DGA's seed, and the value of `day`, `month` and `year`. Note that these names are misleading: although theses values are initialized with the current or past dates, the values are modified by each call to the routine.

```
0009C778
0009C778
0009C778
0009C778 second_level_domain proc near
0009C778
0009C778 flag= dword ptr -4
```


0009C778 object= dword ptr  8
0009C778 result= dword ptr  14h
0009C778
0009C778 mov eax, offset loc_B4976
0009C77D call stack_unrolling
0009C782 push ecx
0009C783 push ebx
0009C784 push ebp
0009C785 push esi
0009C786 xor esi, esi
0009C788 mov ebx, ecx
0009C78A mov [esp+10h+flag], esi
0009C78E mov ecx, [esp+10h+result]
0009C792 push edi
0009C793 mov [esp+14h+object], esi
0009C797 call new
0009C79C push 0Eh
0009C79E mov [esp+18h+object], esi
0009C7A2 mov [esp+18h+flag], 1
0009C7AA pop ebp
0009C7AB
0009C7AB loc_9C7AB:
0009C7AB mov ecx, [ebx+seed_struct.day]
0009C7AE mov esi, ecx
0009C7B0 mov edi, [ebx+seed_struct.seed]
0009C7B3 mov eax, ecx
0009C7B5 and eax, 1FFFh
0009C7BA shr ecx, 0Fh
0009C7BD xor esi, edi
0009C7BF shl esi, 2
0009C7C2 xor esi, eax
0009C7C4 mov eax, [ebx+seed_struct.year]
0009C7C7 imul edx, eax, 7
0009C7CA shl esi, 4
0009C7CD xor esi, ecx
0009C7CF push 19h
0009C7D1 mov [ebx+seed_struct.day], esi
0009C7D4 xor edx, eax
0009C7D6 and eax, 0FFFFFFF0h
0009C7D9 shl eax, 11h
0009C7DC shr edx, 0Bh
0009C7DF xor edx, eax
0009C7E1 mov eax, [ebx+seed_struct.month]
0009C7E4 mov ecx, eax
0009C7E6 mov [ebx+seed_struct.year], edx
0009C7E9 shl ecx, 2
0009C7EC xor ecx, eax
0009C7EE and eax, 0FFFFFFFEh
0009C7F1 imul eax, 0Eh
0009C7F4 shr ecx, 8
0009C7F7 xor ecx, eax
0009C7F9 lea eax, [esi+edi*8]
0009C7FC shl eax, 8
0009C7FF and eax, 3FFFF00h
0009C804 shr edi, 6
0009C807 xor eax, edi
0009C809 mov [ebx+seed_struct.month], ecx
0009C80C mov [ebx+seed_struct.seed], eax
0009C80F mov eax, esi




This pseudo-code summarizes the algorithm:

```
FOR i = 0 TO 13
    day = (day >> 15) ^ 16 * (day & 0x1FFF ^ 4 * (seed ^ day))
    year = ((year & 0xFFFFFFF0) << 17) ^ ((year ^ (7 * year)) >> 11)
    month = 14 * (month & 0xFFFFFFFE) ^ ((month ^ (4 * month)) >> 8)
    seed = (seed >> 6) ^ ((day + 8 * seed) << 8) & 0x3FFFF00
    int x = ((day ^ month ^ year) % 25) + 'a'
    domain[i] = x;
```

The malware authors repeated their modulus error: like for the tld, the modulus needed to be increased by one. As it stands, ‘z’ is no reachable. Ranbyus shares this bug with the DGAs of Ramnig and Necurs.

Seed the end of this blog post for a C-implementation of the DGA.

## Observed Seeds

The following tables lists some of the samples from malwr.com that are Ranbyus with the described DGA. All samples use the same parametrization, only the seed varies.

| md5 | seed |
|---|---|
| 4b04f6baaf967e9c534e962c98496497 | 65BA0743 |
| 087b19ce441295207052a610d1435b03 | 65BA0743 |

| md5 | seed |
|---|---|
| 28474761f28538a05453375635a53982 | 65BA0743 |
| b309eab0277af32d7a344b8a8b91bd73 | C5F128F3 |
| 4c7057ce783b2e4fb5d1662a5cb1312a | C5F128F3 |
| b309eab0277af32d7a344b8a8b91bd73 | C5F128F3 |
| 7cbc671bcb97122e0ec5b448f0251dc0 | C5F128F3 |
| 437028f94ceea4cab0d302d9ac6973eb | C5F128F3 |
| b309eab0277af32d7a344b8a8b91bd73 | C5F128F3 |
| 6378b7af643e87c063f69ddfb498d852 | B6354BC3 |
| fa57f601402aab8168dea94c7c5f029f | B6354BC3 |
| 9f2c89ad17e9b6cf386028a8c9189264 | 0478620C |

## Summary

### DGA Characteristics

The characteristics of Ranbyus' DGA are:

| property | value |
|---|---|
| seed | magic number and current date |
| granularity | 1 day, with a 31 day sliding window |
| domains per seed and day | 40 |
| domains per sliding window | 1240 |
| sequence | sequential |
| wait time between domains | 500 ms |
| top level domains | .in, .me, .cc, .su, .tw, .net, .com, .pw |
| second level characters | lower case letters except 'z' |
| second level domain length | 14 letters |

### Decompiled Code

The following C code generates the domains for a given day and seed. In order to generate all domains that the malware can generate for any given seed and date, one would also need to run the code for all dates going 30 days in the past.

**Edit 23.5.2015:** The following code had contained a bug that led to a wrong sequence of top level domains, thanks to Anthony Kasza for sharing that with me.

```
#include <stdio.h>
#include <stdlib.h>

char* dga(unsigned int day, unsigned int month, unsigned int year,
        unsigned int seed, unsigned int nr)
{
    char *tlds[] = {"in", "me", "cc", "su", "tw", "net", "com", "pw", "org"};
    char domain[15];
    int d;
    int tld_index = day;
    for(d = 0; d < nr; d++)
    {
        unsigned int i;
        for(i = 0; i < 14; i++)
        {
            day = (day >> 15) ^ 16 * (day & 0x1FFF ^ 4 * (seed ^ day));
            year = ((year & 0xFFFFFFF0) << 17) ^ ((year ^ (7 * year)) >> 11);
            month = 14 * (month & 0xFFFFFFFE) ^ ((month ^ (4 * month)) >> 8);
            seed = (seed >> 6) ^ ((day + 8 * seed) << 8) & 0x3FFFF00;
            int x = ((day ^ month ^ year) % 25) + 97;
            domain[i] = x;
        }
        printf("%s.%s\n", domain, tlds[tld_index++ % 8]);
    }
}

main (int argc, char *argv[])
{
    if(argc != 5) {
        printf("Usage: dga <day> <month> <year> <seed>\n");
        printf("Example: dga 14 5 2015 b6354bc3\n");
        exit(0);
    }

    dga(atoi(argv[1]), atoi(argv[2]), atoi(argv[3]),
            strtoul(argv[4], NULL, 16), 40);
}
```

## Archived Comments

***Note:*** *I removed the Disqus integration in an effort to cut down on bloat. The following comments were retrieved with the export functionality of Disqus. If you have comments, please reach out to me by Twitter or email.*

Sandor Nemes Aug 26, 2015 07:42:01 UTC
In the C code the strtoul function should be used instead as strtol will limit the seed to 0x7fffffff.

Johannes Bader Aug 26, 2015 09:36:08 UTC
Thanks, fixed it.