The DGA in Alureon/DNSChanger

Archived: 2026-04-10 03:03:58 UTC

At least some of the famous DNSChanger malware samples use a domain generation algorithm (DGA) to generate five pseudo random domains. In contrast to most other uses of DGAs, the domains are never intended to be actually registered. Instead, by contacting the domains, the rogue name servers are informed immediately of newly infected clients. Failed DNS queries also reveal that the DNS changes did not succeed, or that the queries are blocked.

Because I couldn't find an implementation of the DGA, I post it here. Skip to Reimplementation and Domains to see the algorithm. I reversed the following sample:

md5
: 2563874010c5adf9009cb9b231c3d0fc

sha256
: 852c45049ec6e384dea40e3cc70479ad06015277646cf30da7ef9a038de9267e

source
: malwr

upload date
: 2014-10-21

Reverse Engineering

Context

The queries to the algorithmically generated domains are made after the DNS settings are changed. Five different domains are generated and tested, see offsets `0x1F1E08` to `0x1F1E13` in the following graph view clipping:

If the last DNS query fails, i.e., the malicious DNS servers can’t be reached, then DNSChanger determines the Windows version and calls some Windows 2000 specific remedies if applicable.

Generation and Resolving of Domains

The routine to generate and call the DGA-domains is very simple:

- The DGA routine is called. The routine takes the length as the only argument. The parameter is set to 10. The DGA returns a second level domain.
- The top level domain *.com* is appended to the second level domain by a `wsprintf` call.
- A DNS query is made for the domain by calling `gethostbyname`. The routine returns *True* if the DGA query succeeds, and *False* otherwise.

The DGA Routine

The DGA is kept simple. It generates a random string by repeatedly calling the `randint` routine to generate random lower case letters. The *length* parameter of the subroutine determines the length of the string.

Pseudo Random Number Generators and Seeding

The `randint` function is:

```
 if upper > lower
return lower + rand() % (upper - lower + 1)
else
return lower
```

The `rand` function is a linear congruential generator as implemented in C:

```
r = 214013*r + 2531011
```

```
return (r >> 16) & 0x7FFF
```

Seeding is done with the `srand` routine:

The seed itself is set to the sum of the current thread id and the approximate number of milliseconds since Windows last booted:

This number can’t be predicted precisely by the backers of DNSChanger. It shows that they don’t intend to actually register the DGA domains. The domains are used to test the changes to DNS settings and potentially to notify the operators of DNSChanger of newly infected systems.

Reimplementation and Domains

Python

Here is a reimplementation in Python. The script takes the seed as the one and only argument.

```
 import argparse
from ctypes import c_int

class Rand:

    def __init__(self):
         self.r = c_int()

    def srand(self, seed):
         self.r.value = seed

    def rand(self):
         self.r.value = 214013*self.r.value + 2531011
         return (self.r.value >> 16) & 0x7FFF

    def randint(self, lower, upper):
         return lower + self.rand() % (upper - lower + 1)

def dga(r):
    sld = ''.join([chr(r.randint(ord('a'), ord('z'))) for _ in range(10)])
    return sld + '.com'


if __name__=="__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("seed", type=int)
    args = parser.parse_args()
    r = Rand()
    r.srand(args.seed)
    for _ in range(5):
         print(dga(r))
```

For example:

```
 $ python dga.py 60000
xyinotlgjm.com
pzebjxgftf.com
jsgjmbtbdr.com
```

```
nhuknekrdg.com
jaebiztqia.com
```

Brute-Forcing

I also wrote a small C program that checks if a provided domain could stem from DNSChanger:

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

unsigned int r;

long myrand() {
    r = 214013*r + 2531011;
    return (r >> 16) & 0x7FFF;
}

long randint (int lower, int upper) {
    return lower + (myrand() % (upper - lower + 1));
}

void dga(unsigned long seed, char* url) {
    long int i;
    long int domain_len = 10;
    r = seed;
    for(i = 0; i < 10; i++) {
        url[i] = randint('a', 'z');
    }
    url[domain_len] = 0;
}

long int find(char* wanted) {
    char url[11];
    printf("searching %s\n", wanted);
    long int seed;
    long int maximum = 0xFFFFFFFF;
    for(seed = 0; seed < maximum; seed++) {
        dga(seed, url);
        if(strcmp(url, wanted) == 0) {
            printf("\r-> found seed %u\n", (unsigned int)seed);
            return 1;
        }
        if(seed % (2 << 22) == 0)
            printf("\r%.3f %%", (seed)/(double)maximum);
    }
```

```
    printf("\n");
    return 0;
}

int main (long int argc, char *argv[])
{
    if ( argc != 2 )
        printf( "usage: %s hostname (without tld)\n", argv[0] );
    else
        find(argv[1]);
}
```

Provide the second level domain to the program to find the corresponding seed. For example, to find the seed for the domain `xyinotlgjm.com`:

```
 $ ./reverse_dga xyinotlgjm
searching xyinotlgjm
-> found seed 60000
```

Domains

This file (about 9MB) lists the first DGA domain for seeds up to 600`000, i.e., roughly ten minutes after system reboot. The domains from sandboxes should fall within this time range.
