

# The DGA of Symmi

Archived: 2026-04-05 21:13:40 UTC

Looking through the most recent reports on malwr.com, a [sample](#) sparked my interest because it suits [my current interest in domain generation algorithms \(DGA\)](#). Virus scanners label the sample as *Symmi*, other names for the same or similar malware family are *MewsSpy* and *Graftor*. The sample is very noisy. It tries to resolve many domains in a short period of time — only limited by the response time of the DNS server:

There are hardly any samples online that match the above DGA pattern; I only found the following four:

date	sha-256
2014-12-12	<a href="#">55f6945302a5baa49f32ef25425b793c</a>
2014-12-27	<a href="#">e0166446a676adb9e3160c9c06e56401</a>
2014-12-29	<a href="#">1ca728b9d0c64b1edfc47aeceb899b4</a>
2015-01-19	<a href="#">b75f00d7ae2857a3e1cc8f5eb4dc11b9</a>

The earliest sample is from December 12th, 2014, which could indicate that the DGA — or at least the used configuration — is fairly new.

## Configuration

The malware uses a large structure for its configuration settings, including most of the parameters that control the DGA:

```
config_data_struct struc
...
+7A8 seed_constant    ; char*
+7AC days_period      ; char*
+7B0 nr_of_domains    ; char*
+7B4 third_lvl_min_len ; char*
+7B8 third_lvl_max_len ; char*
...
```

All parameters are stored as XOR encrypted strings. The following table lists the purpose of the parameters along with the value from the examined sample. The full meaning of the parameters will become clear when discussing the individual parts of the DGA:

variable	value	purpose
seed_constant	42	A constant added to the seed. This allows for different sets of generated domains at any given point in time.
days_period	16	For how many days the DGA will produce the same domains. For example, the value 16 means the DGA produces one set of domains for the first 16 days of the month, and a new set for the second half.
nr_of_domains	64	How many different domains the DGA generates before restarting with the first domain.
third_lvl_min_len	8	How long the third level domain is at the least.
third_lvl_max_len	15	How long the third level domain is at most.

## The DGA

### Random Number Generator

The malware uses msvcrt's implementation of rand — a [linear congruential pseudo-random number generator](#) with multiplier 214013 and increment 2531011. The code is inlined — meaning each call to rand amounts about six lines of assembly:

```
100066CC mov ecx, [esi]
100066CE imul ecx, 343FDh
100066D4 add ecx, 269EC3h
100066DA mov [esi], ecx
...
100066E1 shr ecx, 10h
100066E4 and ecx, 7FFFh
```

In many cases the removal of the low order bytes and clipping to RAND\_INT is optimized away by the compiler. For example:

```
100066CC mov ecx, [esi]
100066CE imul ecx, 343FDh
100066D4 add ecx, 269EC3h
100066DA mov [esi], ecx
...
100066E1 test ecx, 1000000h
```

In contrast to the famous [Conficker.B](#), the rand function is used correctly and all random aspects of the DGA are truly pseudo-random.

## Seed

The random number generator is seeded with the current date and a hardcoded constant:

```
10008625 call ds:GetLocalTime ; kernel32.GetLocalTime
1000862B movzx eax, [esp+60h+day]
10008630 xor edx, edx
10008632 div [esp+60h+days_period] ; is 0x10
10008636 mov [esp+60h+domain_counter], 0
1000863E mov edx, eax
10008640 movzx eax, [esp+60h+month]
10008645 imul edx, 100
10008648 add edx, eax
1000864A movzx eax, [esp+60h+year]
1000864F imul edx, 10000
10008655 add edx, [esp+60h+seed_constant] ; is 0x2A
10008659 add eax, edx
```

which calculates:

$$\text{seed} = \left( \lfloor \frac{\text{day}}{\text{days\_period}} \rfloor \cdot 100 + \text{month} \right) \cdot 10000 + \text{year} + \text{seed\_constant}$$

The configurable `seed_constant` (42 for all four samples listed in the introduction) allows for different sets of domains at the same point in time. The parameter `days_period` controls how many days the DGA will produce the same domains. For the examined sample I got `days_period = 16`, meaning the domains will change twice a month.

## Third Level Domain Length

Right after the DGA is seeded, the length of the third level domain is determined:

```

10008666      mov     ebx, [esp+60h+third_lvl_max_len]
1000866A      sub     ebx, [esp+60h+third_lvl_min_len]
1000866E      inc     ebx
1000866F      jmp     short loc_10008675

10008671 ; -----
10008671
10008671 loc_10008671:                ; CODE XREF: calls_create_next_url+B00j
10008671      mov     eax, [esp+60h+rand]
10008675
10008675 loc_10008675:                ; CODE XREF: calls_create_next_url+9FFj
10008675      imul  eax, 343FDh
1000867B      add     eax, 269EC3h
10008680      mov     [esp+60h+rand], eax
10008684      shr     eax, 10h
10008687      and     eax, 7FFFh
1000868C      xor     edx, edx
1000868E      div     ebx
...
1000869B      add     edx, [esp+68h+third_lvl_min_len]
1000869F      push   edx                ; third_lvl_len

```

The above code sets the domain length to a value picked uniformly at random between `third_lvl_min_len` and `third_lvl_max_len` (including the boundaries):

```

span = third_lvl_max_len - third_lvl_min_len + 1
third_lvl_len = third_lvl_min_len + r.rand() % span

```

## Generating the Domain

Next follows the core of the DGA algorithm: generating the actual domain inside `create_domain`. Apart from the domain length, the subroutine also takes a type and the third and top level string as parameters.

```

10008690 mov     ecx, [esp+60h+second_and_top_lvl]
10008694 push   1                ; type
10008696 push   ecx                ; second_and_top_lvl
10008697 lea   eax, [esp+68h+rand]
1000869B add     edx, [esp+68h+third_lvl_min_len]
1000869F push   edx                ; third_lvl_len
100086A0 call   create_domain

```

The `create_domain` routine includes two similar types of DGA, selected by the `type` parameter passed to the routine. The examined sample only used type 1 to generate the domains, so I didn't investigate the other type option. The routine is rather long (over 300 lines of disassembly), which is mostly attributed to the fact that the

compiler inlined `rand` and `strcat` . I therefore only show the Python version of the DGA here, you can find the underlying disassembly on [Github Gist](#):

```
def next_domain(r, second_and_top_lvl, third_lvl_domain_len):
    letters = ["aeiouy", "bcdfghklmnpqrstvwxyz"]
    domain = ""

    for i in range(third_lvl_domain_len):
        if not i % 2:
            offset_1 = 0 if r.rand() & 0x100 == 0 else 1
            s = r.rand()
            offset = (offset_1 + i) % 2
            symbols = letters[offset]
            domain += symbols[s % (len(symbols) - 1)]

    return domain + second_and_top_lvl
```

The DGA picks random letters from either “*aeiouy*” (vowels) or “*bcdfghklmnpqrstvwxyz*” (consonants). The letter “*j*” is missing. Also, because of a bug in the DGA, the last characters of either character class, i.e., “*y*” or “*z*” can’t be picked. The DGA randomly chooses either the vowel or consonant class for letters with even index (0, 2, 4, ...); the subsequent letter is then always from the other character class. For example:

```
p <- consonant
i <- vowel

v <- consonant
u <- vowel

o <- vowel
g <- consonant

u <- vowel
s <- consonant

o <- vowel
d <- consonant

t <- consonant
o <- vowel

k <- consonant
u <- vowel
```

This will lead to domain names that are somewhat pronounceable. After the third level domain is generated, the DGA appends the configured second and top level string, for this sample “.ddns.net”

## Number of Domains

The malware is very aggressive in that it doesn't wait when it tries to resolve a NXDOMAIN, but instead immediately moves on to the next domain — first determining the length, then building the url with `create_domain`. The DGA only waits when it can either resolve a domain or if `nr_of_domains` (in my sample 64) domains are consumed:

```
10008B38 mov     ecx, [esp+60h+nr_of_domains]
10008B3C mov     cl, [eax+ecx]
10008B3F mov     [edx+ebx], cl
10008B42 inc     eax
10008B43 inc     edx
10008B44 cmp     eax, esi
10008B46 jnb     short loc_10008B5E
```

The DGA will finally sleep after all `nr_of_domains` domains are tested without success. When all domains were NXDOMAINS, the sleep time is a mere thirty seconds; If the C2 sites could not be reached for another reason (flag `dword_100393D8` set), the DGA sleeps for five minutes:

```
seg129:10008D74      cmp     ds:dword_100393D8, 0
seg129:10008D7B      jz     short loc_10008D8F
seg129:10008D7D      push   300000          ; unsigned __int32
seg129:10008D82      call   __sleep_1
seg129:10008D87      add     esp, 4
seg129:10008D8A      jmp     loc_10007C87
seg129:10008D8F ; -----
seg129:10008D8F
seg129:10008D8F loc_10008D8F:
seg129:10008D8F      push   300000          ; unsigned __int32
seg129:10008D94      call   __sleep_1
seg129:10008D99      add     esp, 4
seg129:10008D9C      jmp     loc_10007C87
seg129:10008D9C calls_create_next_url endp
```

## Python Code of DGA

The following Python code generates 64 domains for a given date (or today, if no date is provided):

```
import argparse
from datetime import datetime

seed_const = 42
days_period = 16
nr_of_domains = 64
```

```
third_lvl_min_len = 8
third_lvl_max_len = 15

class Rand:

    def __init__(self, seed):
        self.seed = seed

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

def next_domain(r, second_and_top_lvl, third_lvl_domain_len):
    letters = ["aeiouy", "bcdfghklmnpqrstvwxyz"]
    domain = ""

    for i in range(third_lvl_domain_len):
        if not i % 2:
            offset_1 = 0 if r.rand() & 0x100 == 0 else 1
        s = r.rand()
        offset = (offset_1 + i) % 2
        symbols = letters[offset]
        domain += symbols[s % (len(symbols) - 1)]

    return domain + second_and_top_lvl

def dga(seed, second_and_top_lvl, nr):
    r = Rand(seed)
    for i in range(nr):
        span = third_lvl_max_len - third_lvl_min_len + 1
        third_lvl_len = third_lvl_min_len + r.rand() % span
        print(next_domain(r, second_and_top_lvl, third_lvl_len))

def create_seed(date):
    return 10000*(date.day//days_period*100 + date.month) + date.year + seed_const

if __name__=="__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("-d", "--date", help="as YYYY-mm-dd")
    args = parser.parse_args()
    date_str = args.date
    if date_str:
        date = datetime.strptime(date_str, "%Y-%m-%d")
    else:
        date = datetime.now()
```

```
seed = create_seed(date)
dga(seed, ".ddns.net", nr_of_domains)
```

You also find the script on [GitHub Gist](#).

## DGA Characteristics

The following table summarizes the properties of the DGA

property	value
seed	based on date and configurable constant, set to change twice a month in analysed sample
domains per seed	unlimited
tested domains	configurable, set to 64 in analysed sample
sequence	one after another, restarting with first domain when no success
wait time between domains	none, before restarting with first domain 30 second or 5 minute wait time
top and second level domain	.ddns.net
second level characters	all letters except “z”, “y” and “j”
second level domain length	uniformly distributed between configurable bounds, for the analysed sample 8 to 15 characters

## Past and Upcoming Domains

For the seed constant “42” and the day period “16”, these are the first three domains generated between November of last year and the upcoming 2015.

start date	end date	first three domains
2014-11-01	2014-11-15	vitevecaasbaim.ddns.net, buxotopelah.ddns.net, doefruevtan.ddns.net
2014-11-16	2014-11-30	tevalurii.ddns.net, ufrasequcoequidi.ddns.net, qularivafou.ddns.net
2014-12-01	2014-12-15	urasahrenaheen.ddns.net, xoegfeima.ddns.net, niubsacaosuce.ddns.net
2014-12-16	2014-12-31	leuvuftet.ddns.net, obneifqumea.ddns.net, bodihemouxk.ddns.net
2015-01-01	2015-01-15	uwuhuhawidb.ddns.net, exdihasuhes.ddns.net, axtoomov.ddns.net

<b>start date</b>	<b>end date</b>	<b>first three domains</b>
2015-01-16	2015-01-31	pivuogusodtoku.ddns.net, onogibuluremg.ddns.net, geevheuqsemaif.ddns.net
2015-02-01	2015-02-15	obwihecidik.ddns.net, umashadilauru.ddns.net, irmatexoitn.ddns.net
2015-02-16	2015-02-28	opacutemadufu.ddns.net, ercehuhowi.ddns.net, uwsioqxua.ddns.net
2015-03-01	2015-03-15	eqosenrealq.ddns.net, duwugunuwaqauk.ddns.net, loopowakm.ddns.net
2015-03-16	2015-03-31	efrubatoiketxa.ddns.net, loliqooq.ddns.net, tixufaheurvo.ddns.net
2015-04-01	2015-04-15	haaxicuconx.ddns.net, xaguuswibuoqope.ddns.net, ukwoubapgi.ddns.net
2015-04-16	2015-04-30	aginemkiroacus.ddns.net, haerbugoviosmu.ddns.net, vireacvio.ddns.net
2015-05-01	2015-05-15	wewateikho.ddns.net, oviceosweub.ddns.net, siriomilfomu.ddns.net
2015-05-16	2015-05-31	vabuibofqouxog.ddns.net, tubiebigceli.ddns.net, olkaerxedus.ddns.net
2015-06-01	2015-06-15	muavosecit.ddns.net, uxefilka.ddns.net, deivekmiuwoxe.ddns.net
2015-06-16	2015-06-30	looxnaaluhotw.ddns.net, feohpoaqpeheuw.ddns.net, gabouhlat.ddns.net
2015-07-01	2015-07-15	ucsauhdune.ddns.net, deohupivoco.ddns.net, haufidasu.ddns.net
2015-07-16	2015-07-31	bovuugodvuecf.ddns.net, cuopxeudu.ddns.net, muarocavhaqe.ddns.net
2015-08-01	2015-08-15	edehgogoep.ddns.net, uciluswaaqnieb.ddns.net, ugxeicbeveudusu.ddns.net
2015-08-16	2015-08-31	ogovugtuipawi.ddns.net, afowkaupbabe.ddns.net, ipkureleakm.ddns.net
2015-09-01	2015-09-15	rocaexesti.ddns.net, veonwient.ddns.net, axgoxevikupoxa.ddns.net
2015-09-16	2015-09-30	uhrixaloduuse.ddns.net, gecoohocali-fluw.ddns.net, ecunxoorokonw.ddns.net
2015-10-01	2015-10-15	huosinamu.ddns.net, udebliena.ddns.net, imewgiopaqexacb.ddns.net
2015-10-16	2015-10-31	gouhumuvelcua.ddns.net, decouqunic.ddns.net, eretodweqee.ddns.net
2015-11-01	2015-11-15	hiwosoofa.ddns.net, oxacuhuvanoxxo.ddns.net, tahimoteev.ddns.net
2015-11-16	2015-11-30	vebiabipkilo.ddns.net, hunoikuxibi.ddns.net, imugoqsoakiqahi.ddns.net
2015-12-01	2015-12-15	ociqusdal.ddns.net, xiupfisuaw.ddns.net, mivibicoruq.ddns.net
2015-12-16	2015-12-31	veeswaehsisu.ddns.net, uhbacoim.ddns.net, baugkoosdui.ddns.net

Source: <https://bin.re/blog/the-dga-of-symmi/>