The new Domain Generation Algorithm of Nymaim

Archived: 2026-04-10 03:05:47 UTC

The Nymaim malware first appeared in 2013. It is mainly used as a downloader for other malware such as ransomware, but it later also started manipulating search results for click fraud.

Many great articles have been published on Nymaim, including its DGA, maybe because the malware used an effective and interesting obfuscation. The obfuscation lead to many imaginative tools to aid analysis of the malware, for example by the GovCERT.CH or the CERT Polska.

Apart from the obfuscation, Nymaim is interesting because it tries to protect itself against sinkholing by adding a checksum to the A resource records, and by transforming the IP addresses before using them, see "Nymaim revisited" by the CERT Polska, "Threat Spotlight: GozNym" by Talos and "Nymaim Origins, Revival and Reversing Tales" by Alberto Ortega.

This month a new version of Nymaim appeared with a few modifications to the above mentioned features:

- The obfuscation has been dropped entirely, apart from using a packer. On the contrary, the malware even uses helpful logging messages and a configuration with descriptive names.
- The IP transformation has been slightly changed, using different constants, but otherwise sticking to the same procedure as before.
- The DGA has been completely rewritten. It is now based on wordlists, like the DGA of *Matsnu*, *Suppobox*, or the close relative to Nymaim, *Gozi*.
- Apart from the DGA, Nymaim also has a list of hard-coded domains that follow the same pattern as the DGA domains, but which are tried before the time-dependent DGA domains.

This blog post focuses on the DGA and the IP transformation aspect of Nymaim. For example, these are the first ten domains for April 27, 2018:

```
 virginia-hood.top
shelter-downloadable.ps
tylerpreparation.sg
zolofteffectiveness.ch
stakeholders-looked.hn
williampassword.sc
thailandcool.re
thoughtsjazz.ec
recovery-hairy.ac
workshopsforms.hn
```

I analyzed the following sample from Virustotal:

MD5
: 30bce8f7ac249057809d3ff1894097e7

SHA-256
: 73f06bed13e22c2ab8b41bde5fc32b6d91680e87d0f57b3563c629ee3c479e73

SHA-1
: b629c20b4ef0fd31c8d36292a48aa3a8fbfdf09c

Filesize
: 484 KB

Compile Timestamp
: 2010-06-13 18:50:03 (*very likely faked*)

First Submission to Virustotal
: 2018-04-17 21:49:18

Virustotal Link
: link

I unpacked it to the following executable. All screenshots are taken from this sample loaded at address 0x400000:

MD5
: 379ba8e55498cb7a71ec4dcd371968af

SHA-256
: 3eb9bbe3ed251ec3fd1ff9dbcbe4dd1a2190294a84ee359d5e87804317bac895

SHA-1
: 5f522dda6b003b151ff60b83fe326400b9ed7716

Filesize
: 368 KB

Compile Timestamp
: 2018-03-02 23:12:20

First Submission to Virustotal
: 2018-04-26 12:19:41 (*by me*)

Virustotal Link
: link

Analysis

This Section describes the details of the DGA. If you are only interested in a Python reimplementation, please refer to Section DGA.

Seeding

The new DGA of Nymaim is seeded with three values:

1. A hard-coded, 32 character upper-case hexadecimal string, presumably a MD5 hash (`3138C81ED54AD5F8E905555A6623C9C9` in the analyzed sample). Nymaim calls this string **GenerationKey**.

2. The zero-based day of the year. For example, January 1st is 0. This value is lower by one than the ISO definition, which defines January 1st as day 1 of the year. From this value a counter is subtracted, which starts at 0 and counts up to **DayDelta** (10 in the sample). This means the DGA will revisit domains from up to 10 days into the past if necessary (except at the turn of year, see Sliding Window).
3. The last two digits of the current year.

These three values are concatenated into a string. This string is then MD5-hashed, with the result represented as a lower-case hexadecimal string. Please note that this is in contrast to the *GenerationKey*, which is all upper-case. The resulting string is the seed and basis for the ensuing pseudo-random number generations.

Pseudo-Random Number Generator

The pseudo-random number generator (PRNG) uses the first 8 characters of the MD5 hash string and takes it as a big-endian hexadecimal representation of a 32-bit integer, which is the random number. The first 7 characters of the MD5 hash are discarded, and the rest is again hashed with MD5 and represented as a lower-case hexadecimal string. The first 8 characters from this string represent the next pseudo-random value. See the following illustration for the seeding procedure and the PRNG:

The DGA

The DGA uses four random values to pick strings from four lists:

1. A word from a first word list.
2. A separator character.
3. A word from a second word list.
4. A top level domain.

The four strings are then concatenated to form the domain. The words are chosen by using the remainder of dividing the random value by the length of the list to be picked from as the index into the list:

```
 CString *__thiscall dga(_DWORD *config, CString *szDomainName)
{
  dgaconfig *cfg; // esi@1
  int v3; // eax@2
  unsigned int nNumberOfFirstWords; // ecx@3
```

```
  randnrs objRandNrs; // [esp+Ch] [ebp-2Ch]@1
  int dgb2; // [esp+20h] [ebp-18h]@1
  int nr_random_values; // [esp+24h] [ebp-14h]@1
  char cstrDomainName; // [esp+28h] [ebp-10h]@3
  int dbg; // [esp+34h] [ebp-4h]@1

  dgb2 = 0;
  cfg = config;
  init_random_nrs(&objRandNrs);
  objRandNrs.self = &GetRuntimeClass;
  nr_random_values = 4;
  dbg = 1;
  do
  {
    v3 = rand_0(&cfg->random_hash);
    store_rand(objRandNrs.field_8, v3);
    --nr_random_values;
  }
  while ( nr_random_values );
  CString::CString(&cstrDomainName);
  nNumberOfFirstWords = cfg->nNumberOfFirstWords;
  LOBYTE(dbg) = 2;
  CString::operator+=(&cstrDomainName, cfg->rgFirstWords + 4 * (*objRandNrs.r % nNumberOfFirstWords));
  CString::operator+=(&cstrDomainName, cfg->rgSeparators + 4 * (*(objRandNrs.r + 4) % cfg->nNumberOfSeparators)
  CString::operator+=(&cstrDomainName, cfg->rgSecondWords + 4 * (*(objRandNrs.r + 8) % cfg->nNumberOfSecondWords
  CString::operator+=(&cstrDomainName, cfg->rgTLDs + 4 * (*(objRandNrs.r + 12) % cfg->nNumberOfTLDs));
  CString::CString(szDomainName, &cstrDomainName);
  dgb2 = 1;
  LOBYTE(dbg) = 1;
  CString::~CString(&cstrDomainName);
  LOBYTE(dbg) = 0;
  cleanup_0(&objRandNrs);
  return szDomainName;
}
```

The first word list contains 2450 words that start with letters R to Z. The shortest have four letters, the longest has 18 (*telecommunications*):

```
    "reaches",
   "reaching",
   "reaction",
   "reactions",
   "read",
   "reader",
   "readers",
   "readily",
```

```
    "reading",
    "readings",
    "reads",
    "ready",
    "real",
    "realistic",
    ...
    "zoom",
    "zoophilia",
    "zope",
    "zshops"
```

There are only two separators: the zero length string and the hyphen `-`. The third word list contains 4387 words starting with letters C to R. The last word is `reached`, which is probably the word just before the start of the first word list beginning with `reaches`. The shortest words have four letters, the longest have 18 (e.g., *pharmaceuticals*):

```
     "contamination",
    "contemporary",
    "content",
    "contents",
    "contest",
    "contests",
    "context",
    "continent",
    "continental",
    "continually",
    ...
    "ratios",
    "rats",
    "raymond",
    "rays",
    "reach",
    "reached"
```

Finally, there are 74 top level domains. The tld `.com` appears four times and `.net` appears three times, which increases the likelihood that `.com` or `.net` are picked. The full list of TLDs is: `.com`, `.com`, `.com`, `.net`, `.net`, `.net`, `.ac`, `.ad`, `.at`, `.am`, `.az`, `.be`, `.biz`, `.bt`, `.by`, `.cc`, `.ch`, `.cm`, `.cn`, `.co`, `.com`, `.cx`, `.cz`, `.de`, `.dk`, `.ec`, `.eu`, `.gs`, `.hn`, `.ht`, `.id`, `.in`, `.info`, `.it`, `.jp`, `.ki`, `.kr`, `.kz`, `.la`, `.li`, `.lk`, `.lv`, `.me`, `.mo`, `.mv`, `.mx`, `.name`, `.net`, `.nu`, `.org`, `.ph`, `.pk`, `.pl`, `.pro`, `.ps`, `.re`, `.ru`, `.sc`, `.sg`, `.sh`, `.su`, `.tel`, `.tf`, `.tj`, `.tk`, `.tm`, `.top`, `.uz`, `.vn`, `.win`, `.ws`, `.wtf`, `.xyz`, `.yt`.

Sliding Window

The DGA generates **MaxDomainsForTry** domains per day, which for the analysed sample is 64. After those 64 domains, the PRNG is reseeded with the seed from the previous day, by subtracting 1 from the day of the year. This way, up to 64*(10+1) = 704 domains are generated:

At the turn of the year, when the day of the year is smaller than the *DayDelta*, the offset day can become negative. For example, on January 3rd the sliding window leads to day of year values of 2, 1, 0,-1, …, -8. The negative value lead to new set of domains.

Hard-Coded Domains

Nymaim has a list of 46 hard-coded domains that follow the DGA pattern of two words separated by an optional hyphen. These domains are all with the `.com` TLD. The hard-coded domains are always tested first, before any DGA domains are calculated. For the sample at hand, the hard-coded domains are:

```
 sustainabilityminolta.com
theories-prev.com
starringmarco.com
seekerpostcards.com
threadsmath.com
recall-pioneer.com
waste-neighborhood.com
usage-maternity.com
standings-descriptions.com
volumedatabase.com
summaries-heading.com
stoppedmeaningful.com
singles-october.com
scottish-fact.com
weblogcourage.com
troycyber.com
reply-phantom.com
wagon-crime.com
sharp-louisiana.com
suitedminerals.com
saskatchewan-funds.com
sites-experts.com
techrepublicexemption.com
serbia-harbor.com
super-ideas.com
translationdoor.com
wildhelmet.com
shoefalse.com
remainedoxide.com
wild-motels.com
staticlesbian.com
valentinequeensland.com
travelling-mechanics.com
solelypersonal.com
resulting-museum.com
towndayton.com
workedforest.com
yorkshire-engineer.com
stockholm-effect.com
reynoldshydrogen.com
```

```
sluts-persistent.com
satisfaction-granted.com
slut-hentai.com
territoriesprayers.com
thumbnailsfragrance.com
undergraduategraphical.com
```

Nameserver Test

A distinctive feature of Nymaim is the DNS query for the name server record (NS). Nymaim checks if any of the answers contains a word from a list it calls **BlackNsWords**. These words are related to sinkholing:

```
 sinkhole
amazonaws
honeybot
honey
parking
domaincontrol
dynadot
```

If Nymaim finds any of those word in the NS resource record, it will not use the domain.

Preferred DNS Servers

Nymaim uses a list of dns servers called **PreferredDnsServers**, presumably because these are less likely to alter or block DNS requests.

IP	Company
8.8.8.8	Google
8.8.4.4	Google
156.154.70.1	Neustar Security
156.154.71.1	Neustar Security
208.67.222.222	OpenDNS
208.67.220.220	OpenDNS

IP-Transformation

Like the earlier version of Nymaim, the A resource records are **not** the C2 IPs. The real addresses are obtained by transforming the IPs with a sequence of easily reversible XOR and substraction steps. Talos intelligence wrote a detailed report in September 2017 that describes the algorithms.

The following graph view snippet shows the transformation of an IP:

A Python script to perform the IP transformation in both directions can be found at the end of this blog post.

Checksum Test

Nymaim also still uses the checksum test of A resource records. For example, here are the IPs for a C2 domain that was operational at the time of writing:

```
 > dig @8.8.8.8 +short -t A sustainabilityminolta.com
127.33.12.14
127.33.12.15
```

```
192.202.176.55
126.56.117.50
```

The following table lists these four IPs (first column) and the transformed, real IP (second column). The third column shows the integer representation:

IP	**IP'**	**value**
127.33.12.14	127.0.0.0	0x0000007F
127.33.12.15	127.0.0.1	0x0100007F
192.202.176.55	192.42.116.41	0x29742AC0
126.56.117.50	**190.43.116.42**	**0x2A742BBE**

Nymaim will check all integer values to see if they are the sum of the remaining values. In the above example, the bold “IP” 190.43.116.42 is the result of transforming the A RR 126.56.117.50. It has a little endian integer representation of 0x2A742BBE. This corresponds to the checksum obtained by adding up the integer representation of the remaining IPs, i.e.,0x2A742BBE = 0x0000007F + 0x0100007F + 0x29742AC0.

The IP that matches the checksum is removed from the list, it only serves as the checksum for the other IPs. Nymaim will then one after another test the transformed IPs:

1. A DNS request for the NS resource of `sustainabilityminolta.com` is made to check for signs of a sinkhole. The response `dns100.ovh.net` does not contain one of the `BlackNsWords` and Nymaim proceeds to query the A records.
2. The DNS request for the A records returns four transformed IPs. Because the fourth IP is a checksum for the remaining three IPs, Nymaim goes on to contact the IPs in order.
3. The first non local IP address, 192.42.116.41, is contacted with an HTTP POST requests to `http://192.42.116.41/index.php`.

Request

The actual C2 requests are HTTP POSTs. The content is AES encrypted with a session key, which is protected with asymmetric encryption. The first C2 requests are around 900 bytes. The URL filename is hard-coded to `index.php`:

```
http://192.42.116.41/index.php
```

Characteristics

The following tables summarizes the properties of the Nymaim DGA.

property	value
type	TDD (time-dependent-deterministic)
generation scheme	MD5 based PRNG
seed	generation key + current date
domain change frequency	daily, with a 11 day sliding window
domains per day	46 hardcoded domains + 64 new DGA domains + 640 old DGA domains
sequence	sequential
wait time between domains	None
top level domains	69 different domains, `.com` and `.net` favored
second level characters	two words from wordlists with optional hyphen as separator
second level domain length	8 (e.g., *realrays.kr*) – 34 (e.g., *telecommunications-pharmaceuticals.name*)

Results

In this section you find a Python reimplementation of the DGA, and a script for the IP transformation of Nymaim. Please also refer to the Github page for current versions of the scripts.

DGA

The DGA needs the large wordlists words.json, place it in the same directory as the DGA script. You can generate the domains for a specific day with `-d` or `--date`, for example:

```
> python dga.py -d 2018-04-27
```

```
import json
import argparse
from datetime import datetime
import hashlib
```

```
class Rand:

    def __init__(self, seed, year, yday, offset=0):
        m = self.md5(seed)
        s = "{}{}{}".format(m, year, yday + offset)
        self.hashstring = self.md5(s)

    @staticmethod
    def md5(s):
        return hashlib.md5(s.encode('ascii')).hexdigest()

    def getval(self):
        v = int(self.hashstring[:8], 16)
        self.hashstring = self.md5(self.hashstring[7:])
        return v

def dga(date):
    with open("words.json", "r") as r:
        wt = json.load(r)

    seed = "3138C81ED54AD5F8E905555A6623C9C9"
    daydelta = 10
    maxdomainsfortry = 64
    year = date.year % 100
    yday = date.timetuple().tm_yday - 1

    for dayoffset in range(daydelta + 1):
        r = Rand(seed, year, yday - dayoffset)
        for _ in range(maxdomainsfortry):
            domain = ""
            for s in ['firstword', 'separator', 'secondword', 'tld']:
                ss = wt[s]
                domain += ss[r.getval() % len(ss)]
            print(domain)

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()
    dga(date)
```

IP Transformation Script

The following Python script can be used to transform Nymaim IP addresses in both directions, and to see if a list of IP addresses fulfills the checksum requirement:

```
 import argparse


def iptoval(ip):
    els = [int(_) for _ in ip.split(".")]
    v = 0
    for el in els[::-1]:
        v <<= 8
        v += el
    return v


def valtoip(v):
    els = []
    for i in range(4):
        els.append(str(v & 0xFF))
        v >>= 8
    return ".".join(els)


def step(ip, reverse=False):
    v = iptoval(ip)
    if reverse:
        v ^= 0x18482642
        v = (v + 0x78643587) & 0xFFFFFFFF
        v ^= 0x87568289
    else:
        v ^= 0x87568289
        v = (v - 0x78643587) & 0xFFFFFFFF
        v ^= 0x18482642
    return valtoip(v)


def transform(ip, iterations=16, reverse=False):
    for _ in range(iterations):
        ip = step(ip, reverse=reverse)
    return ip


def checksum(pairs, index):
    checksum = 0
    for i, p in enumerate(pairs):
```

```
        if i == index:
            continue
        checksum += iptoval(p[1])
    return checksum & 0xFFFFFFFF


def findip(pairs):
    for i, p in enumerate(pairs):
        c = checksum(pairs, i)
        if c == iptoval(p[1]):
            return p[0]


if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("ip", nargs="+")
    parser.add_argument("-r", "--reverse", help="reverse transformation",
                        action="store_true")
    parser.add_argument("-c", "--checksum", help="test checksum",
                        action="store_true")
    args = parser.parse_args()

    pairs = []
    for ip_src in args.ip:
        ip_dst = transform(ip_src, reverse=args.reverse)
        pair = (ip_src, ip_dst)
        d = "-->"
        if args.reverse:
            pair = pair[::-1]
            d = "<--"
        pairs.append(pair)
        if not args.checksum:
            print("{} {} {}".format(ip_src, d, ip_dst))

    fmt = "| {:4} | {:15} | {:15} | {:10} |"
    fmt2 = "| {:4} | {:15} | {:15} | 0x{:08X} |"
    if args.checksum:
        print(fmt.format("", "IP", "IP'", "value"))
        print(fmt.format(*4 * ["---"]))
        ok_ip = findip(pairs)

        for ip, ipp in pairs:
            if ip == ok_ip:
                continue
            print(fmt2.format("", ip, ipp, iptoval(ipp)))

        for ip, ipp in pairs:
```

```
            if ip != ok_ip:
                continue
            print(fmt2.format("x", ip, ipp, iptoval(ipp)))

        if not ok_ip:
            print("No IP matches checksum")
        else:
            print("The IP marked x matches the checksum of remaining IPs, "
                  "it is removed.")
```

For example, one of the A RR of `sustainabilityminolta.com` is 192.202.176.55. The real IP can be found with:

```
 > python3 transform.py 192.202.176.55
192.202.176.55 --> 192.42.116.41
```

To reverse the transformation, use `-r` or `--reverse`:

```
 > python3 transform.py 192.42.116.41 --reverse
192.42.116.41 <-- 192.202.176.55
```

To check if the A resource records satisfy the checksum, add all IPs as arguments and add `-c` or `--checksum`:

```
 > python3 transform.py 127.33.12.14 127.33.12.15 192.202.176.55 126.56.117.50 --checksum
|      | IP              | IP'             | value      |
| ---  | ---             | ---             | ---        |
|      | 127.33.12.14    | 127.0.0.0       | 0x0000007F |
|      | 127.33.12.15    | 127.0.0.1       | 0x0100007F |
|      | 192.202.176.55  | 192.42.116.41   | 0x29742AC0 |
| x    | 126.56.117.50   | 190.43.116.42   | 0x2A742BBE |
The IP marked x matches the checksum of remaining IPs, it is removed.
```

If an IP matches the checksum, it is marked with an `x`.

Source: https://johannesbader.ch/2018/04/the-new-domain-generation-algorithm-of-nymaim/