The Faulty Precursor of Pykspa's DGA

Archived: 2026-04-05 20:43:24 UTC

Pyskpa is a worm that spreads over Skype. The malware has been relying on a *domain generation algorithm* (DGA) to contact its command and control targets since at least October 2013. Even though the C2 infrastructure seems to be long abandoned, there are still many infected clients. Virustracker, who has been tracking Pyskpa since March, shows that the DGA is still used by well over 50`000 infected clients. You can find a description of the underlying DGA here.

A few days ago, Daniel Plohmann at Fraunhofer FKIE discovered new Pyskpa domains within the ShadowServer feeds. He kindly provided me the sample, from which I reversed the algorithm behind the newly found Pykspa domains. This short post first shows the algorithm, then examines its properties in comparison with the other DGA version.

The characteristics and spread of the emerged DGA variant lead me to believe that it is the predecessor of the other version. As shown later, the algorithm behaves in strange ways that were probably not intended by the malware authors. I therefore refer to the DGA in this post as the *Precursor DGA*, and call the other DGA the *Improved DGA*.

The Precursor DGA

The precursor DGA generates sets of 5000 distinct hostnames. It is seeded with the current unix timestamp divided by 172800, which corresponds to a granularity of two days. Here’s an implementation of the DGA in Python:

```
 from datetime import datetime
import argparse
from time import mktime

def get_sld(sld_len, r):
    a = sld_len ** 2
    sld = ""
    for i in range(sld_len):
        x = i*(r % 4567 + r % 19) & 0xFFFFFFFF
        y = r % 123456
        z = r % 5
        p = (r*(z + y + x)) & 0xFFFFFFFF
        ind = (a + p) & 0xFFFFFFFF
        sld += chr(ord('a') + ind % 26)
        r = (r + i) & 0xFFFFFFFF
        r = r >> (((i**2) & 0xFF) & 31 )
        a += sld_len
        a &= 0xFFFFFFFF
```

```
    return sld

def dga(seed, nr_domains = 5000):
    tlds = ["biz", "com", "net", "org", "info", "cc"]
    r = seed
    for domain_nr in range(nr_domains):
        r = int(r ** 2) & 0xFFFFFFFF
        r += domain_nr
        r &= 0xFFFFFFFF
        domain_length = (r % 10) + 6
        sld = get_sld(domain_length, r)
        tld = tlds[r % 6]
        domain = "{}.{}".format(sld, tld)
        print(domain)


def generate_domains(date, nr):
    unix_timestamp = mktime(date.timetuple())
    seed = int(unix_timestamp // (2*24*3600) )
    date_range = []
    for i in range(2):
        ts = (seed+i)*2*24*3600
        date_range.append(datetime.fromtimestamp(ts).strftime("%Y-%m-%d %H:%M"))
    t = "pykspa domains valid through {} - {}".format(*date_range)
    print("{}\n{}".format(t, "*"*len(t)))
    dga(seed, nr)

if __name__=="__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("-d", "--date", help="date for which to generate domains")
    parser.add_argument("-n", "--nr", help="nr of domains to generate", type=int, default=5000)
    args = parser.parse_args()
    if args.date:
        d = datetime.strptime(args.date, "%Y-%m-%d")
    else:
        d = datetime.now()
    generate_domains(d, args.nr)
```

For example, these are the 20 first domains active at 2015-07-19 00:00:

```
 ./dga.py -n 20 -d 2015-07-19
pykspa domains valid through 2015-07-18 02:00 - 2015-07-20 02:00
************************************************************
kmambodsholapet.com
siaiheiq.biz
oagsesiugkeq.net
```

```
jshbzafox.org
xfawpafox.cc
kspongeoya.net
cmvccqeoya.info
sbtrssdsholapet.com
aumarenansnan.cc
yeuwwiiugkeq.biz
aolmbo.info
dxnydafox.cc
qmzmtufqbex.org
skxsiyeoya.net
ichnvyeoya.info
lopevkn.com
zjrckkn.com
oitykueoya.net
megsoeiq.net
zfdthsn.cc
```

Properties and Comparison with the Improved DGA

The following table compares some the properties of the precursor DGA to the improved DGA.

Precursor DGA	**Improved DGA**
seeding	

Granularity is 2 days. Seed corresponds to divided timestamp:
`seed = int(unix_timestamp // (2*24*3600))`
	Granularity of 20 days. Seed corresponds to divided timestamp, passed through a cryptographic function:
	`index = int(unix_timestamp//(20*3600*24))`
	`seed = some_cryptographic_function(index)`

noise domains	
no noise domains	Interleaves the usable domains with noisy domains that are generated by the same DGA, but with an unpredictable seed.
nr domains per run	
5000	200 usable domains + 800 noisy domains
next random number	
Uses repeated squaring, which lets random number converge to two values.	Uses increasingly larger increments. No convergence.

<table>
<tr><th>Precursor DGA</th><th>Improved DGA</th></tr>
<tr><td><pre>r = int(r ** 2) (mod 2^32)
r += domain_nr (mod 2^32)</pre></td><td><pre>r += (r % (domain_nr + 1) + 1) (mod 2^32)</pre></td></tr>
<tr><td colspan="2">next second level domain length</td></tr>
<tr><td>Length of second level domain is between 6 and 15 characters:<pre>domain_length = (r % 10) + 6</pre></td><td>Length of second level domain is between 6 and 12 characters:<pre>domain_length = ((r + domain_nr) % 7) + 6</pre></td></tr>
<tr><td colspan="2">second level domain algorithm (all calculations mod 2^32)</td></tr>
<tr><td>Random number is right shifted after each letter, leading to convergence at the end of domain names.<pre>a = sld_len ** 2
sld = ""
for i in range(sld_len):
  index = (a + (r*(r % 5 + r % 123456 +
    i*(r % 4567 + r % 19))) +
    i*(r % 4567 + r % 19)))) % 26
  a += sld_len
  r = (r + i)
  r = r >> (((i**2) & 0xFF) & 31)
  sld += chr(ord('a') + index)</pre></td><td>No visible randomness decay for the later letters in domains, otherwise very similiar.<pre>a = sld_len ** 2
sld = ""
modulo = 541 * sld_len + 4
for i in range(sld_len):
  index = (a + (r*((r % 5) + (r % 123456) +
    i*((r & 1) + (r % 4567))))) % 26
  a += sld_len;
  r += (((7837632 * r * sld_len)) +
    82344) % modulo;
  sld += chr(ord('a') + index)</pre></td></tr>
<tr><td colspan="2">top level domain algorithm</td></tr>
<tr><td>Random pick from 6 top level domains:<pre>tlds = ["biz", "com", "net", "org",
    "info", "cc"]
tld = tlds[r % 6]</pre></td><td>Random pick from 4 top level domains, although 5 domains are hardcoded.<pre>tlds = ['com', 'net', 'org', 'info', 'cc']
tld = tlds[r % 4]</pre></td></tr>
</table>

Both DGAs are very similar. The precursor DGA has a defective random number generators though:

1. Within the main loop: changing the random number for each domains; the problem lies with `r = int(r ** 2)`.

2. Within the code to generate the second level domains: changing the random number for each letter; the culprit here is `r >> (i**2)`.

The defects cause a drastic loss of randomness the more random numbers are generated. For example, here are some of the domains whose second level domain has 15 characters:

```
 kmambodsholapet.com
sbtrssdsholapet.com
srjukodsholapet.org
izbqukdsholapet.com
ybdetodsholapet.com
mgesbsdsholapet.org
uafudkdsholapet.cc
wrsxuodsholapet.com
ycokbodsholapet.org
ckocgkdsholapet.org
yndccsdsholapet.cc
slwcnodsholapet.cc
anljvkdsholapet.cc
oznevadsholapet.org
ewgxsodsholapet.cc
wigiakdsholapet.com
mvomnkdsholapet.com
wvmqfodsholapet.cc
```

While the beginning of the domain is pretty divers, after the sixth letter always follows *“dsholapet”*. This is true for all 15 letter domains, making for a pretty solid network detection rule.

The other defect, i.e., repeatedly squaring the random number, is arguably even worse. It causes the random number to converge to one of two values, depending on the sign of the initial seed. Although the malware authors probably wanted to have a fresh set of 5000 domains every two days, the convergence causes all but the first 19 domains to be very consistent, only ever alternating between the same two sets of domains. The following image illustrates this behavior:

Every saturated color represents a new, yet unseen domain. The desaturated colors stand for revisited domains. As expected, every second day reuses the domains of the previous day, in accordance with the granularity of 2 days. The first 9 domains change after 48 hours, as desired. The later domains, however, increasingly revisit older domains, up to the point where no new domains are generated.

The next picture evaluates the domains over one year in increments of four days. A blue square represents a new domain, while a grey square represents a revisited domain. Clearly all domains after 19 stay the same. But already the second domain now and then reuses an older domain:

Conclusion

The precursor DGA is very similiar to the improved version, yet suffers from bad random number generators. The DGA still got deployed in the wild, as the following screenshot of Virustracker shows:

However, the number of hits is only about 600, compared to over 60'000 of the improved DGA:

Source: https://bin.re/blog/pykspas-inferior-dga-version/