Botnet

A new botnet Orchard Generates DGA Domains with Bitcoin Transaction Information

daji, suqitian
Aug 5, 2022 • 13 min read

DGA is one of the classic techniques for botnets to hide their C2s, attacker only needs to selectively register a very small number of C2 domains, while for the defenders, it is difficult to determine in advance which domain names will be generated and registered.

360 netlab has long focused on the research of botnet attack and defense technology, we maintain a free DGA feed and share the research results with the industry.

Recently we discovered a new botnet that uses Satoshi Nakamoto's Bitcoin account transaction information to generate DGA domain name. Because of the uncertainty of Bitcoin transactions, this technique is more unpredictable than using the common time-generated DGAs, and thus more difficult to defend against.

The technique was discovered in a family of botnets we called Orchard. Since February 2021, this botnet has gone through 3 versions, and even switched programming languages in between.

Key points are as follow:

- Orchard is a botnet family that uses DGA technology with the core function of installing various malware on the victim's machine.

- From February 2021 to the present, we have detected 3 versions of Orchard samples, all using the DGA technique.
- Orchard's DGA algorithm has remained unchanged, but the use of dates has been changing, and the latest version also supports the use of bitcoin account information to generate separate DGA domains.
- In addition to DGA, Orchard also hardcodes C2 domains.
- Orchard is still active and dedicated to Monroe coin mining.

Overview

Orchard uses a redundant C2 mechanism of "hardcoded domain + DGA", and each version hardcodes a unique DuckDNS dynamic domain name as C2.

```
v1, orcharddns.duckdns.org
v2, orchardmaster.duckdns.org
v3, ojena.duckdns.org
```

A timeline is as follow

```
* 2021.3, v1, use C++. Combined with historical data, we found that the earliest appe
* 2021.9, v2, use Golang and C++
* 2022.7, v3, use C++
```

All three versions support propagation by infecting USB disks, much like traditional viruses, as described in the later section of "USB Infection Logic". Theoretically, Orchard can be spread in other ways as well.

Using our graph system in combination with PDNS and other dimensional data, we found that there is a clear case of shared IPs between the C2 of v1 and v2, as shown in the figure below.

The graph system helped us find more C2 IPs and domains, and the domains here are characterized by all ending in duckdns.org. v3 is relatively new and has fewer associated domains, and here is the active situation of v3 domains.

We can see that it was launched in May this year, and then gradually became active, and it should still be in the active period.

Based on our PDNS data, we evaluated the infection scale of the three versions, among which v1 and v2 have thousands of nodes, and v3 has less because of its late appearance, and the following is the detailed resolution number of each version of domain name to specific IP(note the numbers do not reflect all the bots as as our PDNS visibility focus mainly in China)

```
# v1, orcharddns.duckdns.org
37, 45.61.185.36
413,  45.61.186.52
1301,  45.61.187.240
207, 205.185.124.143

# v2, orchardmaster.duckdns.org
45,  45.61.185.36
104, 45.61.186.52
659,  45.61.187.240

# v3, ojena.duckdns.org
418, 45.61.185.231
```

Sample Analysis

Loader is used for counter analysis and self-protection. Currently Orchard loader is not fixed, even a single version can have a variety of loaders, for example, v1 version of Orchard is base64 encoded in the loader, v2/v3 version of the sample in the form of resource files stored in the loader in some cases. Each version has also used virtualization packers such as VMP, Enigma, etc. to protect itself. In general, Orchard's workflow can be summarized in the following diagram.

The functions of all three versions of Orchard are basically the same and include:

- Uploading device and user information
- Responding to commands/downloading to execute the next stage of the module
- Infecting USB storage devices

The core functions of each of the 3 versions of Orchard are analyzed below in several dimensions, such as DGA algorithm, C2 communication and host behavior.

v1 version

The analysis of this version is based on the sample with MD5=5c883ff8539b8d04be017a51a84e3af8. It first releases the embedded PE file to the self-boot directory at runtime, and the released PE is base64 decoded in memory to get the orchard data, and then the PE uses any exe under System32/SysWOW64 as a puppet process to run the saved orchard code. The overall logic of this version of Orchard is as follows, mainly divided into two parts:

network communication and USB infection, the final function depends on the specific module issued by C2, so orchard itself can be considered as a Downloader.

Here we mainly describe its network communication process (the logic of USB infection is the same for all three versions, see the section on USB infection for details).

C2 communication process is relatively simple, the bot in the check-in process will contact C2 to send the collected host information, and then wait for C2 response. The response data format is generally "command + data", the function of the command is specified by the command code.

The information collected by v1 version includes: volume serial number (HWID), computer name, user name, operating system name, system version, installed capture driver name, antivirus information, parent process file modification time, top window name and window title, etc. These information are separated by " [o.r.c.h.a.r.d]". " as a separator and then sent as shown in the following figure.

The example of C2 response data is as follows, where "[&&]" stands for instruction code 2, which represents downloading and execution, and the specific processing is divided into 2 types: if the response data is a URL, the PE corresponding to the URL is downloaded and executed; if it is a base64 encoded content, the decoded data is decoded first and then executed. The response data here is actually the new version of base64-encoded PE file, which is equivalent to upgrade, which also indicates that the old version may have been deprecated.

The v1 version defines a total of 8 instructions, and the correspondence between the instruction code and the instruction string is as follows.

```
1 \[=]
2 \[&&]
3 \[##]
4 \[###]
5 \[%%]
6 \[%%%]
7 \[#\_#]
8 \[\_\_\] \[>>] \[<<] \[^^] \[\*\] \[\~\] \[@] \[!] \[#\*\#\] \[#@#]
```

Due to the nulling of some instructions, the eight instructions actually correspond to three operations (subsequent versions are similar).

- Instruction code 1 and 2: determine whether the response data is URL or PE, if it is URL, then download and execute, if it is PE, then create a process to execute (CreateProcess to create a process, puppet process, remote thread injection, etc.).
- Instruction code 3, 4, 8: terminate the current process to delete the original file, or restart.
- Instruction code 7: collect C2, port, PID, file name information again to send to C2, example: orcharddns.duckdns.org[o.r.c.h.a.r.d]5890[o.r.c.h.a.r.d]2260[o.r.c.h.a.r.d]stage-3_.exe[o.r.c .h.a.r.d]

DGA Algorithm

v1's DGA takes the date string (e.g. "2022/07/05") as input, calculates its MD5 value, then divides the MD5 string into four 8-byte substrings, and splices them with the 4 suffixes of .com, .net, .org, .duckdns.org in turn to get the daily 4 groups of 16 DGA domain names, the algorithm is implemented as follows.

```
# 2021/04/15
import datetime
import hashlib

days=30
for i in range(0, days):
    datex = (datetime.datetime.now() - datetime.timedelta(days=i)).strftime('%Y/%m/%
    print("seed: ", datex)
    md5 = hashlib.md5(datex.encode()).hexdigest()
    print('md5: ', md5)

    dga_list = []
    dga_list.append(md5[:8])
    dga_list.append(md5[8:16])
    dga_list.append(md5[16:24])
    dga_list.append(md5[24:32])
    for j in range(len(dga_list)):
        print(dga_list[j] + '.com')
        print(dga_list[j] + '.net')
        print(dga_list[j] + '.org')
        print(dga_list[j] + '.duckdns.org')
```

Sample domains are as follow:

```
seed:  2022/07/05
md5:  91ac64d29f78281ad802f44648b2137f
91ac64d2.com
91ac64d2.net
91ac64d2.org
91ac64d2.duckdns.org
9f78281a.com
9f78281a.net
9f78281a.org
9f78281a.duckdns.org
d802f446.com
d802f446.net
d802f446.org
d802f446.duckdns.org
```

```
48b2137f.com
48b2137f.net
48b2137f.org
48b2137f.duckdns.org
```

v2 version

The v2 version appears as a sample of two programming language implementations, Golang and C++, but with the same functionality. The analysis here takes the Golang sample with MD5=f3e0b960a48b433bc4bfe6ac44183b74 as an example, and its C2 initialization function is shown below, which can obviously see the hard-coded C2 domain names.

The v2 version of C2 communication starts to use json format, and the meaning of the fields is relatively clear. It collects roughly the same information as v1, including: volume serial number (HWID), computer name, user name, system version, antivirus information, active window information, etc. The new fields are: .net framework version (e.g. v2.0.50727), USB status, outgoing package type and its own version. The following is an actual observed version number information, Bot_Version=1.2/G may be interpreted as: version=v1.2, writing language=Golang.

The v2 version of the C++ language sample integrates the same C2, and the version information in the live package becomes "Bot_version:1/C", which collects the information shown in the figure below.

According to the code similarity analysis, the v2 C++ language sample is homologous with the later v3 C++ sample, which means that the latter is evolved from the former.

The v2 version has a total of two kinds of instructions.

- Instruction 1: terminate the current process to delete the original file, or restart it.
- Instruction 2: Determine whether the response data is URL or PE, if it is URL, then download and execute, if it is PE, then create process and execute (CreateProcess create process, puppet process, remote thread injection, etc.).

DGA Algorithm

v2 version of DGA algorithm is the same as v1, the difference lies in the processing of the date string. v2 will splice the hard-coded domain name "orchardmaster.duckdns.org" after the date string, such as "2022/07/05orchardmaster.duckdns.org", and then apply the DGA algorithm of v1 to generate the domain name.

v3 version

The development language of v3 is back to C++, which also includes C2 communication and USB infection functions. C2 communication logic runs in a thread, which also includes a secondary thread tied to XMRig mining, and when Orchard has received the XMrig program and created a puppet process to run, the secondary thread will send mining-related hardware information to C2 again. The purpose of trying to read the configuration of the mining software from C2 is to check if the XMRig runtime configuration needs to be dynamically modified (XMRig provides a set of HTTP api that supports dynamically reading and modifying the runtime mining configuration).
Taking the sample with MD5=cb442cbff066dfef2e3ff0c56610148f as an example, the C2 communication function is as follows.

The v3 version also uses json format to save host information in C2 communication, and the overall structure of the sent data is Byte_0x46+TotalLen+InfoLen+Info.json. Compared to v2, v3 adds several fields related to mining, and the collected information includes:

- Active_Window: the name of the currently active window
- Antivirus: antivirus information
- Authentiate_Type: Windows authentication type
- CPU_Model: CPU information
- Camera: whether a camera is present
- Elevated: whether it is administrator privileges
- GPU_Models: graphics card information
- Identity： HWID\username\computer name
- Operating_System: System version information
- Ram_Size： Running memory size
- System_Architecture： Number of processors
- Threads： Number of cores per processor

- Version： Orchard version

An example of a live package for v3 is shown below.

The body part of C2 response message is also in json format, and its structure is: `TotalLen.dword+ Byte0x46+TotalLen+RespDataLen+RespData.json`.
v3 supports 8 instructions, corresponding to 3 operations.

- Instruction 1: Collect host information/its own running status and send it to C2 (fields include Domain, In_Memory, Install_Path, Is_Patched, Message_Type, Patch_Name, Port, Power_SaverMode, Process_ID. (Process_Name, Process_Path, System_Idle, System_Uptime)
- Instruction 4, 6: terminate the current process to delete the original file, or restart.
- Instruction 7, 8: download & execute the miner program sent down

The following is an actual trace of the C2 response instruction.

Where Transfer_Port indicates that the host is expected to make another request to 2929, and Message_Type indicates the instruction code, whose value is 7, indicating download & execute.

After receiving the above instruction, bot makes another request to C2's TCP port 2929. Cuda is a parallel computing framework introduced by Nvidia that can only be used for its own GPU, and a Cuda_Version of 0 here means that Cuda is not supported.

C2 then responds with an XMRig miner program, which Client receives and saves and then injects XMRig into the puppet process according to instruction 7 to start performing mining work.

During the analysis we found that the v3 version has recently been continuously distributing an identical XMRig mining program, the latter integrated with the default mining configuration information, private mining pool address: 45.61.187.7:7733

DGA Algorithm

The DGA algorithm of v3 is unchanged, but the input is more variable. It actually generates two sets of DGA domains, the first set of domains is entered with a spelling algorithm of date string + "ojena.duckdns.org", shaped like "2022-08-02ojena.duckdns.org ". The second set of domain names is entered as the return result of the URL `https://blockchain.info/balance?active=1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa` . A typical return result is shown below.

```
{"1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa":"final_balance":6854884253,"n_tx":3393,"total_
```

The meaning of the relevant fields can be found in Blockchain's API manual

It is worth emphasizing that the v3 version does not parse the returned results, but directly feed it into the DGA algorithm as a whole to generate the domain name. Instead, the wallet address 1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa is said to be the Bitcoin Genesis address held by Satoshi Nakamoto himself. Over the past decade or so, small amounts of bitcoin have been transferred to this wallet on a daily basis for various reasons, so it is variable and that change is difficult to predict, so the balance information for this wallet can also be used as DGA input.

At the time of writing, we found that other researchers had recently noticed this use of bitcoin account transaction information as DGA input for v3. The results of their analysis agreed with ours, but they did not notice that Orchard had actually been around for a long time.

The complete v3 DGA algorithm is as follows.

```
# 2022/07/05
import datetime
import requests
import hashlib

# cluster 1
days = 30
for i in range(0, days):
    domains = ['ojena.duckdns.org', 'vgzero.duckdns.org']
    for do in domains:
        datex = (datetime.datetime.now() - datetime.timedelta(days=i)).strftime('%Y-%
        print("seed_1: %s" % datex)
        md5 = hashlib.md5(datex.encode()).hexdigest()
        print("md5: %s" % md5)

        dga_list = []
        dga_list.append(md5[:8])
        dga_list.append(md5[8:16])
        dga_list.append(md5[16:24])
        dga_list.append(md5[24:32])
        for j in range(len(dga_list)):
            print(dga_list[j] + '.com')
            print(dga_list[j] + '.net')
            print(dga_list[j] + '.org')
            print(dga_list[j] + '.duckdns.org')


# cluster 2
url = 'https://blockchain.info/balance?active=1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa'
res = requests.get(url)
wallet_info = res.text
print('seed_2: %s' % wallet_info)
md5 = hashlib.md5(wallet_info.encode()).hexdigest()
print('md5: %s' % md5)

dga_list = []
dga_list.append(md5[:8])
dga_list.append(md5[8:16])
dga_list.append(md5[16:24])
dga_list.append(md5[24:32])
for j in range(len(dga_list)):
    print(dga_list[j] + '.com')
    print(dga_list[j] + '.net')
    print(dga_list[j] + '.org')
    print(dga_list[j] + '.duckdns.org')
```

USB Infection

Orchard's file infection is not a traditional code insertion, but a file replacement. When a USB storage device is detected, Orchard will create a hidden directory under the root directory of the device, traverse all files for infection, and back up the files before and after infection to this hidden directory, the infected object is infected with the type attribute removed, and after infection all become exe type, and append the .exe suffix to become an executable file. Then the sample will copy itself to the infected directory and randomly named, the string is saved to the resources of the infected file. When the infected file in the device is executed by the user in the new system, it will launch the sample file in the hidden directory to achieve the purpose of spreading the infection.

The USB infection process involves two embedded PE files, the first one is a DLL file that will be released to the %LocalAppData% directory, this DLL is called CGO_Helper by Orchard and is mainly used to extract and replace the icon of the infected file, its MD5 is The second file is an exe file with MD5 of f3c06399c68c5fdf80bb2853f8f2934b, which is used as a template file to store the infected code, and all the data of the infected file will be replaced with the data of this template file. The function of this template is to find the corresponding exe in the hidden directory to start execution according to the exe name in the resource, so the resource of the infected file is saved with the name of the backup Orchard sample.

USB infection case example is as follows, the name of Orchard sample is saved in the resource of the infected file, when the user clicks on the infected exe, it will start the Orchard sample file in the hidden directory.

Summary

Orchard is a botnet family that uses DGA technology. The latest version is dedicated to mining and has started using more unpredictable information like transaction information of bitcoin accounts as input to DGA, making detection more difficult. In over 1 year, Orchard has appeared in 3 different versions with changes in programming language and DGA implementation, indicating that Orchard is a botnet family that is still active and deserves our vigilance. We expect more variants to emerge subsequently, for which we will continue to keep an eye on, and will continue to disclose new findings.

Contact us

Readers are always welcomed to reach us on twitter or email us to **netlab[at]360.cn**.

IOCs

C2

```
orcharddns.duckdns.org
orchardmaster.duckdns.org
ojena.duckdns.org
vgzero.duckdns.org
victorynicholas.duckdns.org
zamarin1.duckdns.org

45.61.185.36
45.61.186.52
45.61.187.240
205.185.124.143
45.61.185.231
```

MD5

```
5c883ff8539b8d04be017a51a84e3af8
f3e0b960a48b433bc4bfe6ac44183b74
9cbe4bd27eba8c70b6eddaeb6707659b
cb442cbff066dfef2e3ff0c56610148f
10D42F5465D5D8808B43619D8266BD99
f3c06399c68c5fdf80bb2853f8f2934b
19159280736dbe6c11b7d6a57f6bb7b9
```

```
b5a6f78d5575a60316f4e784371d4f8c
3c20ba851edecd28c198691321429883
2b244a39571ab27f7bb4174d460adeef
ae1e9b3621ee041be6ab5e12bff37c53
00b1620f89b7980b34d53737d9e42fd3
4d2445a43591d041cabbbf3dfca6dfbd
```

Private mining pool

```
45.61.187.7:7733
```

Contact us

Readers are always welcomed to reach us on Twitter or email us to netlab[at]360.cn.

— 360 Netlab Blog - Network Security Research Lab at 360 —

Botnet

僵尸网络911 S5的数字遗产

Heads up! Xdr33, A Variant Of CIA’s HIVE Attack Kit Emerges

警惕：魔改后的CIA攻击套件Hive进入黑灰产领域

See all 114 posts →

loader

PureCrypter Loader持续活跃，已经传播了10多个其它家族

在我们的日常botnet分析工作中，碰到各种loader是常事。跟其它种类的malware相比，loader的特殊之处在于它主要用来“推广”，即在被感染机器上下载并运行其它的恶意软件。根据我们的观察，大部分loader是专有的，它们和推广的家族之间存在绑定关系。而少数loader家族会将自己做成通用的推广平台，可以传播其它任意家族，实现所谓的malware-as-...

· Aug 29, 2022 · 14 min read

Botnet

DGA家族Orchard持续变化，新版本用比特币交易信息生成DGA域名

DGA是一种经典的botnet对抗检测的技术，其原理是使用某种DGA算法，结合特定的种子和当前日期，定期生成大量的域名，而攻击者只是选择性的注册其中的极少数。对于防御者而言，因为难以事先确定哪些域名会被生成和注册，因而防御难度极大。 360 netlab长期专注于botnet攻防技术的研究，维护了专门的DGA算法和情报库，并通过订阅情报的方...

· Aug 5, 2022 · 18 min read