

Botnet

DDG的新征程——自研P2P协议构建混合P2P网络



JiaYu

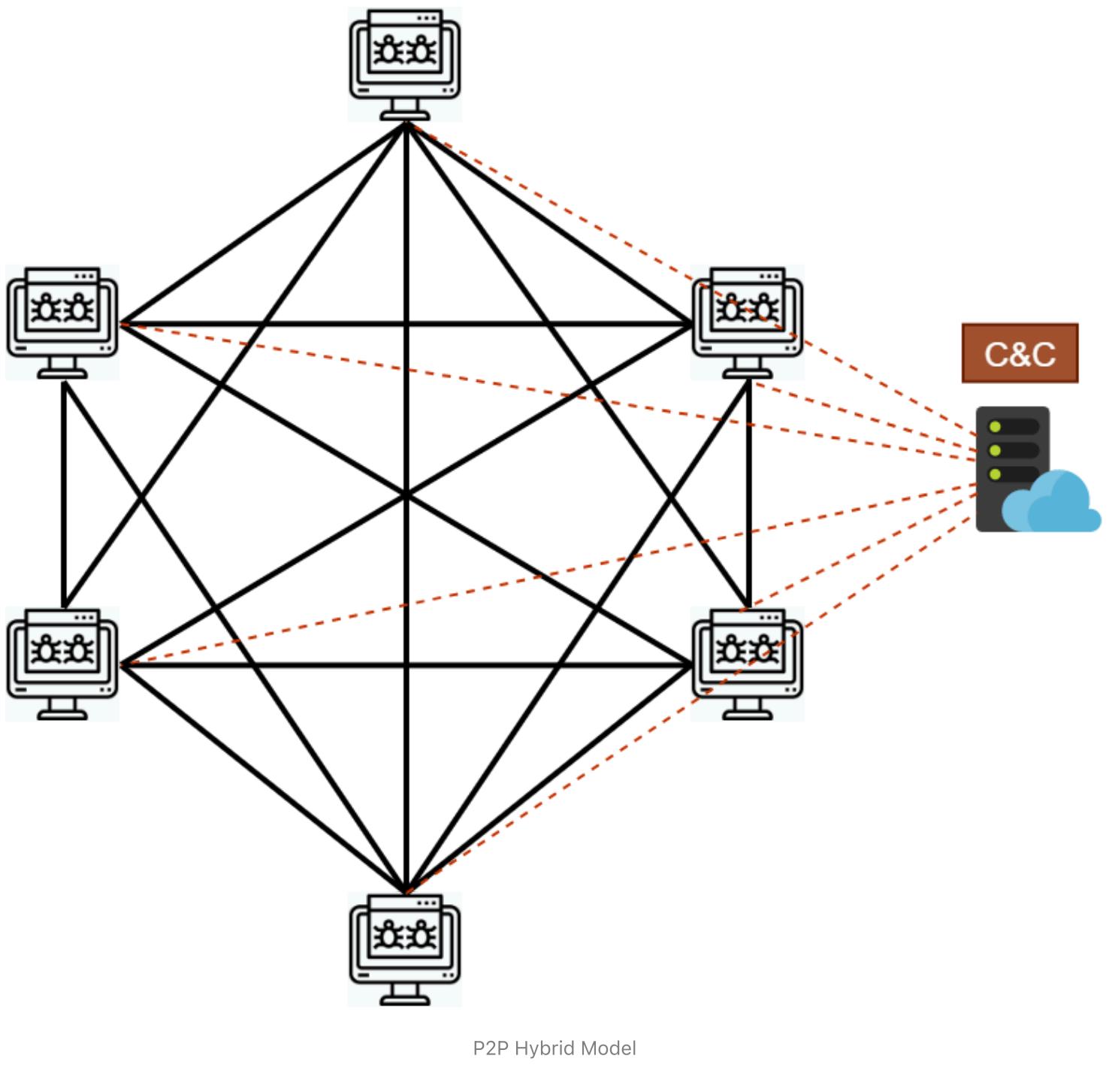
May 7, 2020 • 20 min read

1. 概述

DDG Mining Botnet 是一个活跃已久的挖矿僵尸网络，其主要的盈利方式是挖 XMR。从 2019.11 月份至今，我们的 Botnet 跟踪系统监控到 DDG Mining Botnet 一直在频繁跟新，其版本号和对应的更新时间如下图所示：



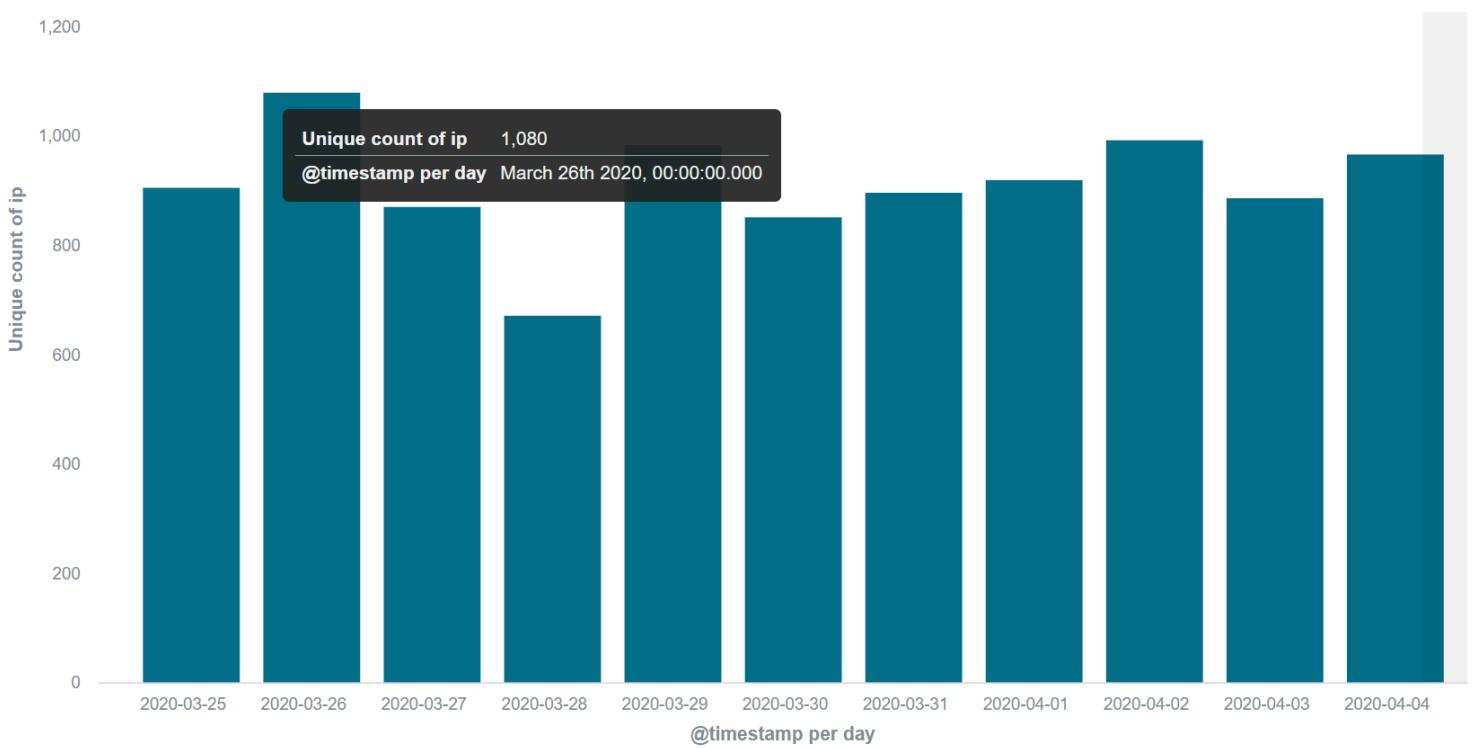
其中，v4005~v4011 版本最主要的更新是把以前以 Hex 形式硬编码进样本的 HubList 数据，改成了 [Gob 序列化](#)的方式；v5009 及以后的版本，则摒弃了以前[基于 Memberlist 来构建 P2P 网络](#)的方式，改用自研的 P2P 协议来构建[混合模式 P2P 网络](#)。简化的网络结构如下：



右边服务器是 C&C Server，DDG 中称它为 **xhub** 节点，是一个超级节点，除了与各 Bot 相同的 P2P 通信功能之外，还具有以下功能：

- 统计各 Peer 信息(Peer 会向它注册，上传自身信息)
- 协助各 Peer 寻找到对方，xhub 节点里保存了大量的 P2P Peers 列表，它会在 Bot 向其注册的时候把一部分列表分享给 Bot；而每个 Bot 最多保存 200 条 Peers <ip:port> 列表；
- 承载原始的恶意 Shell 脚本、主样本和其他组件的下载服务。

我们的 Botnet 跟踪系统追踪到 DDG 当前的一部分 P2P Nodes(失陷主机)，最近每天平均可以追踪到 900 个 Nodes，其中可验证存活的比例达 98%：



根据我们合作伙伴提供的统计数字，中国大陆境内活跃的 Bot 有 17,590 个。根据我们的追踪数据，中国大陆境内的 Bot 数量约占总量的 86%，以此反推，DDG 目前在全球范围内的 Bot 数量约 20,000。

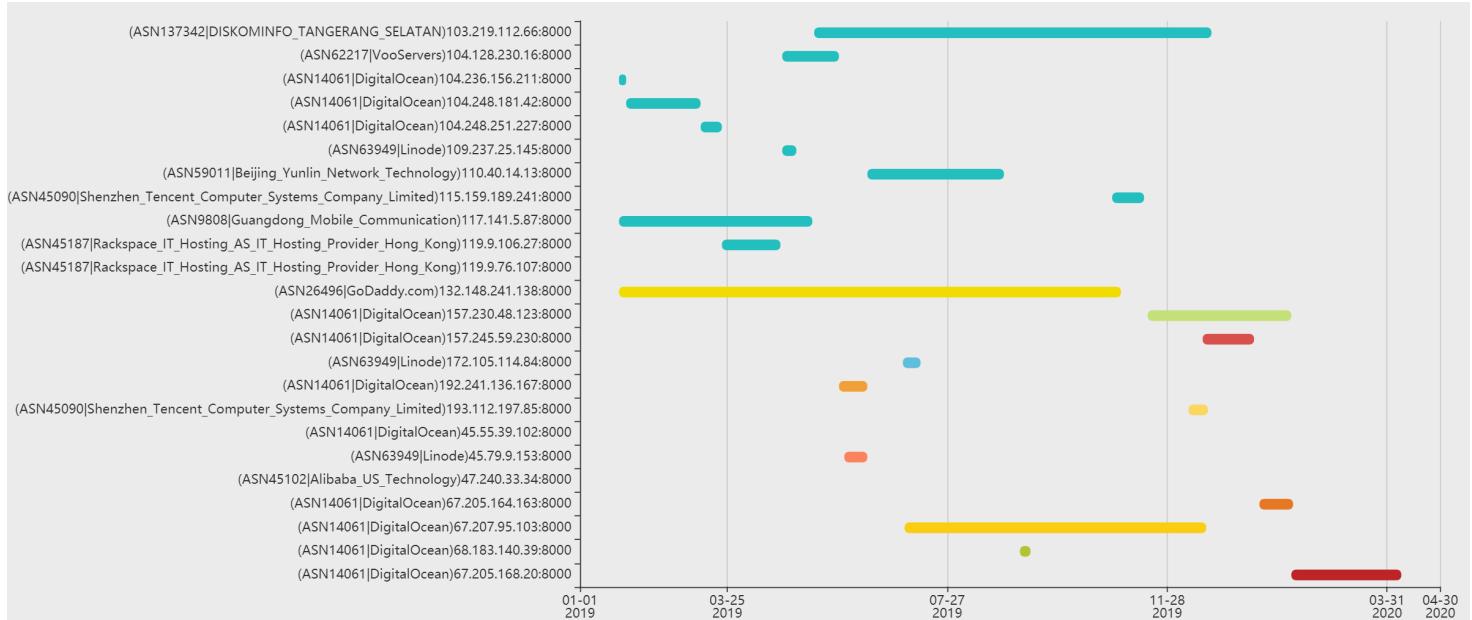
DDG 支持的传播途径，有以下 4 个：

- SSH 服务暴破；
- Redis 未授权访问漏洞；
- 针对 Nexus Repository Manager 3 RCE 漏洞(CVE-2019-7238)的利用；
- 针对 Supervisord RCE 漏洞(CVE-2017-11610)的利用。

根据我们的追踪数据，DDG 最近一段时间一直没有开启公网传播，而只利用 SSH 暴破在内网传播。其内网传播的开关，在 [slave config](#) 里设置：

```
"AAssh":{  
    "Id":2147,  
    "Version":5023,  
    "UrlPath":"/static/5023/ddgs",  
    "NThreads":100,  
    "Duration":"480h",  
    "IPDuration":"12h",  
    "GenLan":True,  
    "GenAAA":False,  
    "Timeout":"1m",  
    "Ports":[  
        22,  
        1987,  
        2222,  
        22222,  
        12222  
    ]  
},
```

另外，根据我们对 DDG 过去一年的追踪统计，它从 2019.1 月份至今共用了 24 个 C&C，每个 CC 的活跃时段统计如下：



最后，旧版本的 DDG Botnet Tracker 已经无法工作，我们现已将其开源：

<https://github.com/oxjiayu/DDGBotnetTracker/tree/master/v1>

并且公开最新版本 DDG 相关的运行日志、相关数据和相应解析工具，以及一个基于 P2P 机制的 Pre-Built Demo Tracker Program(ELF Binary file):

<https://github.com/oxjiayu/DDGBotnetTracker/tree/master/v2>

2. 样本关键行为分析

就 DDG 当前最新的 v5023 版本来看，DDG 有一些有趣的新特性：

- 用特定算法生成一个 4 字符的目录名，并在 `/var/lib/` 或 `/usr/local/` 下创建相应的隐藏目录作为工作目录，存放本地配置信息和从 C&C 或 P2P Peer 下载到的文件或数据；
- 自定义 Base64 编码的编码表(Alphabet)；
- 对关键数据和文件频繁使用编码(Base64/Adler32)、加密(AES)、压缩(Gzip)和数字签名(RSA/ED25519)手段，以对抗分析和数据伪造；
- 除了以前的 **slave** Config 来控制 Bot 执行挖矿和传播任务，又新增了一个 **jobs** Config 文件来对 Bot 进行更复杂的任务控制；

- 自研 P2P 协议，并以此在 Nodes 之间交换最新各自持有的 C&C、Nodes 列表和恶意任务指令配置等信息，还可以在 Peers 之间传播恶意组件。

以 DDG v5023 版本的样本为例，其主要的执行流程，从恶意 Shell 脚本文件 **i.sh** 开始，如下所示：

2.1 恶意 Shell 脚本 i.sh

DDG 每成功入侵一台主机，就会先执行这个 **i.sh** 脚本。**i.sh** 主要执行 3 个任务：

1. 篡改 Cron job，在 `/var/spool/cron/root` 和 `/var/spool/cron/crontabs/root` 写入恶意 Cron job，每 15 分钟下载 `http://67.205.168.20:8000/i.sh` 并运行；
2. 下载并执行最新的 DDG 主样本文件；
3. 检测目录 `/usr/local/bin/` | `/usr/libexec/` | `/usr/bin/` 是否可写。

2.2 初始化

DDG 的主样本由 Go 语言编写，编译出来的原始 ELF 文件体积比较大。DDG 的作者就用 UPX Packer 把原始 ELF 文件加壳，一方面一定程度上可以对抗自动化分析，另一方面缩小了文件体积，便于网络传输。

DDG 的主样本里实现了多个工作模块，有一些后续用到的数据和全局变量，涉及相应的初始化工作。比如生成全局自定义 Alphabet 的 Base64 编解码句柄，解析样本内硬编码的 **xhub(C&C)** 相关数据和 **xSeeds(P2P Seed Nodes List)** 数据，解析样本内硬编码的弱口令字典(用于后续的 SSH 服务暴破)等等。

2.2.1 自定义 Alphabet 的 Base64 编码

在函数 `ddgs_common_init` 中，DDG 基于自定义的 Alphabet 创建了一个全局的 Base64 编解码句柄：

自定义的 Alphabet 为：

"eDy54SH1N2s-

Y7g3qnurvaTW_oBlCMJhfb6wtdGUcROXPAV9KEzIpFoi8xLjkmZQ"

生成的全局 Base64 编解码句柄，会在后续被用来：

- 解析内置硬编码的 **xhub** 和 **xSeeds** 数据；
- 解析后续从服务器下载到的文件 Sig、与 P2P Nodes 通信时的自定义 HTTP Header(**X-Hub** / **X-Seed** / **X-Sig**)
- 响应其他 P2P Peer 的请求时，编码自定义 HTTP Header(**X-Hub** / **X-Seed** / **X-Sig**)

2.2.2 解析内置 xhub/xSeeds

在 DDG 中，**xSeeds** 就是 P2P Seed Nodes 列表，每个 Bot 都持有一份 Seed Nodes List，里面内置了 200 个 P2P Node <ip:port> 列表。Bot 可以与他们通信交换数据。

xhub 即 C&C 服务器信息，C&C 服务器可以指定多个。

在函数 `ddgs_xnet_init_0` 中，DDG 解析了内置的 **xhub** 和 **xSeeds** 数据，还对 **xhub** 数据用 **ed25519** 算法校验是否被伪造。**xhub** 的解析、验证汇编代码如下：

DDG v5023 样本中内置的 **xhub** 数据为经过自定义 Alphabet Base64 编码过的字符串：

```
f0SIE4y3ZPcTuT7weiMSSr7-0-Vem5IfTxEbUirWGS9j5NsDJh2k54RsnK08lG-ECaHQ4ARiWy3mJs009HzBp
```

经过 Base64 解码后，还需要用 [msgPack](#) 进行一层解码，才能得到 **xhub** 的原始明文数据。解析上面的编码数据，可以得到两个关键数据：

- C&C 列表，目前只有一个：**67.205.168.20:8000**；
- ed25519 Signature：

```
0x8f, 0xfa, 0xca, 0x16, 0x49, 0x63, 0x63, 0x03, 0x77, 0x45, 0x15, 0x33, 0x4b, 0x64, 0
0xf4, 0x3c, 0xe0, 0x5b, 0x9c, 0x61, 0x9f, 0x74, 0xd7, 0x98, 0x5b, 0xfb, 0x0c, 0x82, 0
0xf2, 0x7c, 0x0c, 0x4a, 0x4a, 0x47, 0x06, 0x78, 0x6e, 0x62, 0xf1, 0x71, 0x51, 0xbf, 0
0x77, 0x5c, 0x23, 0xfd, 0x78, 0xa6, 0x6a, 0xbc, 0x6c, 0x9a, 0xd2, 0xc8, 0xb7, 0xb4, 0
```

样本中解出 C&C 列表和 ed25519 的 Signature 之后，就会用相反的步骤用 ed25519 算法来校验 C&C 列表是否被伪造。校验时用到的 RSA 公钥为：

```
0x20, 0x0A, 0x51, 0x81, 0x91, 0xE9, 0xF2, 0x54,
0x78, 0xFC, 0x1E, 0x66, 0x7B, 0x8F, 0x8D, 0xAC,
0xCF, 0x62, 0x28, 0x18, 0x46, 0xEC, 0x45, 0x7C,
0xF5, 0xC3, 0xBA, 0x4C, 0x86, 0xB0, 0xB5, 0x41
```

xhub 数据的解析方法：

https://github.com/oxjiayu/DDGBotnetTracker/blob/master/v2/tools/xsig_verify.go

xSeeds 包含 200 条 P2P Node <ip:port> 列表，数据量比较大，共 0x8E2 Bytes：

而且 DDG 样本对 xSeeds 解析步骤与 xhub 的解析略有不同，经过分析，xSeeds 经过以下 3 层处理，而不用 ed25519 校验：

1. msgpack 序列化编码；
2. gzip 压缩
3. 自定义 Alphabet 的 Base64 编码

xSeeds 数据的解析方法：

https://github.com/oxjiayu/DDGBotnetTracker/blob/master/v2/tools/dec_seeds.go

2.2.3 解析内置弱口令字典

在全局初始化函数 `ddgs_global_init` 中，DDG 调用了一个函数 `ddgs_global_decodePasswords`，在这个函数中解密并校验内置的弱口令字典，这些弱口令将在后续传播阶段被用来暴破 SSH 服务(暴破 SSH 服务时用的用户名为 **root**)：

弱口令数据是经过 **AES** 加密、**gzip** 压缩后内置在样本中的，密文数据共 **0x2BCFE** Bytes：

DDG 会首先对上面数据用 **gzip** 解压，解压后密文数据结构如下：

DDG 会先用 **ed25519** 对上述密文数据的 **Sha256** 值进行校验(公钥与前面校验 **xhub** 时用的公钥相同)，校验成功之后才会用 **AES** 算法将其解密，解密后得到一份 **17,907** 条密码的弱口令字典：

2.2.4 创建全局 **ed25519** 密钥对

在函数 `ddgs_global_init` 中，DDG 还有另外一项关键全局变量的初始化工作：创建一对全局 **ed25519** 密钥，用以在后续存取 BoltDB 中数据时对数据进行签名和校验：

创建密钥对的种子如下：

```
0x5C, 0x9E, 0xAE, 0xAE, 0x43, 0x26, 0xB7, 0xA2,  
0x52, 0xDC, 0x43, 0xF9, 0xBD, 0x3F, 0xD1, 0xA6,  
0xC8, 0xB0, 0x28, 0xE1, 0xDF, 0xA8, 0xB0, 0xF5,  
0xCF, 0x43, 0xE7, 0x82, 0xD1, 0x90, 0x11, 0x6B
```

2.3 创建工作目录

旧版本的 DDG 会直接把当前用户的 HOME 目录作为自己的工作目录，主要是在此目录下创建隐藏的 **BoltDB** 文件，文件中存放 Hublist 数据(旧版本的 P2P Node List)。现在新版本的 DDG 会用特定算法生成目录名，并在 `/var/lib/` 或 `/usr/local/` 目录下创建相应的隐藏目录。Go 语言实现的工作目录名生成算法如下(假设当前 DDG 二进制样本文件的路径为 `/root/ddgs_x64`)：

https://github.com/oxjiayu/DDGBotnetTracker/blob/master/v2/tools/gen_workdir_name.go

上面程序的执行结果是 **jsfc**，那么 DDG 就会尝试创建目录 `/var/lib/.jsfc`，后续工作目录的结构如下：

```
/var/lib/.jsfc  
├── 5023  
│   └── cache  
│       └── static  
│           ├── bb3  
│           │   ├── busybox.x86_64  
│           │   └── busybox.x86_64.sig  
│           └── wordpress  
│               └── wordpress.sig  
└── .local
```

其中 `.local` 文件即为 BoltDB 格式的文件，其它的还有从 C&C 服务器上下载到的恶意挖矿程序(**wordpress**)、编译好的 Busybox ELF 文件以及它们相应的数字签名。

2.4 BoltDB 文件

BoltDB 是一个基于内存的小型 KV 数据库，其数据内容可以落地到磁盘文件。DDG 旧版本中用 BoltDB 来存放明文 Hublist 数据。在 DDG 最新的 BoltDB 数据库中，有一个 **Bucket** 名为 **xproxy**。**xproxy** 里存放了 3 份数据：

1. **hubsig**: xhub 信息；
2. **seeds**: xSeeds 数据；
3. **port**: 本地在 (30000~65535) 范围内随机监听的 TCP 端口，用来相应其他 P2P Nodes 的通信请求。

DDG 对这 3 份数据，每一份都做了如前文内置弱口令字典数据同样的处理，经过 msgpack 序列化编码后再用 AES 加密，重组加密数据后再存入 BoltDB。重组的密文数据结构也与弱口令字典密文数据结构相同：

不同的是，DDG 对 BoltDB 中数据的签名和校验用到的全局密钥，是函数 **ddgs_global_init** 用样本中硬编码的种子数据生成的。

2.5 监控关键文件/目录

DDG 利用 **fsnotify** 框架监控以下文件，在运行期间防止被别的进程改动，用以保护自己的持久化配置：

- `/root/.ssh/authorized_keys`
- `/var/spool/cron/root`
- `/var/spool/cron/crontabs/root`

3. P2P 机制

DDG 执行完上述关键步骤，DDG Bot 就开始了与其他 Bot 之间的 P2P 通信。DDG 的 P2P 机制特性如下：

- Peers 首先会向 xhub 注册
- Peers 之间有特有的 Ping/Pong 机制
- Peers 之间可以共享各自持有的 C&C 和 Peers 列表数据；
- Peers 之间可以传播 **slave config** 和 **jobs config** 数据；
- Peers 之间可以传播其他 Payload 或组件，比如恶意挖矿程序、已编译的 Busybox 二进制文件

3.1 准备工作

在进行 P2P 通信之前，DDG 会有两项关键的准备工作：

- 生成一个随机域名
- 生成自己的 Peer UID

随机域名的生成规则是：`<RAND 3 LowChars>.<RAND [5-7] Lowchars>.com`，如 `kez.tirueg.com`。

UID 的形式：`VERSION_NUM.UID_HASH`，如 `5023.dd9b2f57af3be6b6276606d4f37e4a5b`。

UID_HASH 的生成规则，则是综合计算 Host information 和网卡信息的 MD5 值得出，Go 语言实现如下：

<https://github.com/oxjiayu/DDGBotnetTracker/blob/master/v1/lib/util.go#L40>

3.2 Peer <--> xhub

DDG 会首先向 xhub 如下 HTTP POST 请求：

以上 HTTP 通信中，

- Request Header **Host**, 即为事先随即生成的域名。值得一提的是，这个域名不会被 DNS 服务解析，因为以上 HTTP 通信是复用了已建立的 TCP **Socket Session**，在已有 **TCP Socket** 连接上 **Send/Recv** 数据并把数据用 **HTTP** 协议来解析而已。攻击者这样做，可能是为了逃避或混淆某些安全设备的流量检测；
- Request Header **X-Hub**, 即为 DDG 样本当前持有的硬编码在样本中的 **xhub** 信息，详见 2.2.2 节；
- Request Header **X-Port**, 即为 DDG 样本当前随即开启并监听的 P2P 通信端口；
- Request Header **X-Uid**, 即事先生成的 UID；
- Request Header **X-Relay**, 是 DDG 综合 **X-Uid** 和 **X-Port** 字段的值通过 **Adler** 算法算出来的校验值；
- Response Header **X-Seed**, 是对方从自己持有的 Peers 列表中随机选取的 20 个 Peers 地址列表信息，DDG 样本收到后会合入自身持有的 200 个 Peers 列表，总数不超过 200；
- Response Header **X-Hub**, 是对方持有的 **xhub** 信息，DDG 样本收到后会用它替换掉自身持有的 **xhub** 信息。

最后，HTTP 响应中的 **jobs config** 数据，是经过 msgpack 序列化编码后又用 AES 加密过的数据，数据的组织结构与 gzip 解压后的样本内置弱口令字典数据相同，解析过程也完全相同。最新解密后的 **jobs config** 数据见：

<https://github.com/oxjiayu/DDGBotnetTracker/blob/master/v2/data/jobs.json>

从 **jobs Config** 的内容来看，攻击者对 Bot 的行为控制的更为复杂精细，该配置文件的核心功能在流程图里已有说明，此处总结如下：

- 针对每个低级版本 DDG 都有不同的处理，或 Kill 或 Discard 或 Upgrade；
- 引入了 Busybox 执行更复杂的命令，主要用来干掉竞争对手；
- 干掉竞争对手的姿势复杂多样，杀进程、禁用服务、清除 SSH Key、删除 Cron Jobs，重置 Lock File 等等，最显眼的是通过篡改 `/etc/hosts` 文件

屏蔽一大批竞争对手需要访问的域名。

其中，最显眼的一部部分配置，是 DDG 通过篡改 `/etc/hosts` 文件屏蔽竞争对手要访问的域名，大多数都是 **LSDMiner** 和 **systemdMiner** 相关的域名：

```
LSDMiner      img.sobot.com
LSDMiner      lsdu.b-cdn.net
LSDMiner      thyrsi.com
LSDMiner      aliyun.one
systemdMiner   an7kmd2wp4xo7hpr.onion.sh
systemdMiner   an7kmd2wp4xo7hpr.tor2web.su
systemdMiner   aptgetgxqs3secda.onion.ly
systemdMiner   aptgetgxqs3secda.onion.pet
systemdMiner   dns.rubyfish.cn
systemdMiner   aptgetgxqs3secda.onion.in.net
systemdMiner   aptgetgxqs3secda.tor2web.fyi
systemdMiner   an7kmd2wp4xo7hpr.d2web.org
systemdMiner   an7kmd2wp4xo7hpr.timesync.su
```

3.3 Peer <--> Peer

在与 **xhub** 通信过后，DDG 样本就开始与自身持有的 200 个 Peers 进行通信。Peers 之间的通信有 4 个关键步骤：

1. Ping/Pong
2. 拉取对方的 **slave config** 和 **jobs config**；
3. 拉取对方的其他恶意组件，比如恶意矿机程序和 Busybox 二进制程序；
4. 响应(服务)别的 Peers 的以上 3 种请求

3.3.1 Ping/Pong

DDG 一旦成功与某个 Peer 建立连接，会至少经过 2 轮 Ping/Pong 通信，间隔 **3os**，对端响应的 Pong 包与 Ping 包完全相同，长度为 **3 Bytes**：

3 Bytes 的 Ping 包生成规则如下：

- 第 1 字节为 `0x00`；

- 第 3 字节在 [0, 0xFF] 中随机产生；
- 第 2 字节为第 3 字节与 **0x42** XOR 运算的结果。

Go 语言代码描述如下：

```
// Generate ping packet bytes
func GenPingbytes(globalRand *rand.Rand) []byte {
    pkt := make([]byte, 3)
    pkt[0] = 0x00
    pkt[2] = byte((globalRand.Intn(0xFE) + 1) & 0xFF)
    pkt[1] = (pkt[2] ^ 0x42) & 0xFF

    return pkt
}
```

3.3.2 传播 slave/jobs config data

经过 2 轮的 Ping/Pong 通信，DDG 会随机选成功通信的 Peers 向对方请求拉取 **slave config** 或 **jobs config**，请求方式类似于上面向 **xhub** 请求 **jobs config**，以拉取对方 **slave config** 为例：

值得注意的两点：

1. DDG Peers 之间发送 HTTP Post 请求，HTTP Request Header 中相比请求 **xhub** 时少了一个 **X-Port**。
2. 发往 Peer 的 HTTP 请求，复用了前面 **Ping/Pong** 通信中用到的 **TCP** 连接，即同一个 TCP Session，刚开始用来 Ping/Pong 交互，后面直接基于这个 TCP Session 发送 HTTP 请求、接收 HTTP 响应，所以 HTTP Header 中 **Host** 字段里随机生成的域名并不会经过 **DNS** 解析。

对端响应的 **slave config** 数据，是经 msgpack 序列化编码过的二进制数据，数据格式与旧版本变化不大，解码方式可以参考 [以 P2P 的方式追踪 DDG](#)。最新解码后的 **slave config** 数据已上传到 Github：

<https://github.com/oxjiayu/DDGBotnetTracker/blob/master/v2/data/slave.json>

解码后的 **slave config** 数据包含自身数据的数字签名，DDG 样本会以此校验 **slave config** 数据，校验时用到的 RSA 公钥硬编码在样本中：

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEAAQCAQ8AMIIIBCgKCAQEA1+/izr0GcigBPC+oXnr2
S3JI76iXxqn7e90NAmNS+m5nLQx2g0GW44TFtMHhDp1lPdUIui1b1odu36i7Cf0g
31vdYi1i6nGfxXi7UkHMsLVkGkxEknNL1c1vv0qE1b2i2o4TlcXHKHWOPu4xrPYY
m3Fqjni0n5+cQ8IIcVyjkX70N0U1n8pQKRW0vrsPh06tvJnLckK0P1ycG0cgNiBm
sdA5WDjw3sg4xWCQ9EEpMeB0H1UF/nv7AZQ0etncMxhiWoBxamuPWY/KS3wZStUd
gsMB0A00pnbxL9N+II7uquQQkMm06HriXRmjw140mSBEoEcFMWF2j/0HPVas0cx2
xQIDAQAB
-----END PUBLIC KEY-----
```

3.3.3 下载恶意组件

没有被选中拉取 **slave config** 和 **jobs config** 的 Peer，DDG 会继续与他们的 Ping/Pong 通信。从某个 Peer 拉取的 **slave config** 中指定了恶意矿机程序 Miner 的下载 URI、本地存放路径和 MD5，DDG 接下来会随机选取另一个 Ping/Pong 中的 Peer，复用 **Ping/Pong** 的 **TCP Session**，通过 HTTP 协议向其发起 Miner 的下载请求：

下载到的恶意矿机程序，DDG 不仅会将其存放到 **slave config** 中指定的本地路径，还会连同 HTTP Response Header 中的 **X-Sig** 内容作为矿机程序的数字签名数据一起放到自己创建的工作目录中：

```
/var/lib/.jsfc
├── 5023
│   └── cache
│       └── static
│           ├── bb3
│           │   ├── busybox.x86_64
│           │   │   └── busybox.x86_64.sig
│           │   └── wordpress
│           │       └── wordpress.sig
└── .local
```

DDG 从 Peer 下载已编译的 Busybox 程序的过程同上。

3.3.4 响应其他 Peers 的请求

P2P 协议中，Peers 之间的功能、角色是对等的。DDG 样本既然可以从其他 Peer 那里拉取数据和文件，自然也可以响应其他 Peer 的对等请求。

当 Peer 来请求 **slave config** 或 **jobs config** 时，DDG 样本会从内存中整理好一份自己持有的数据，经过与前面阐述的解码、解密相反的编码、加密处理，返回给 Peer。

当 Peer 来请求下载矿机程序(比如上面的 **wordpress** 文件)或已编译好的 Busybox 程序时，DDG 样本会检查自己的工作目录的 **cache** 子目录中是否已经缓存了相应文件，如果缓存了相应文件并且签名有效，就会返回给 Peer。

另外的问题是，很多 DDG 控制的失陷主机都在内网，不一定可以穿透 NAT 对外提供这种服务。所以跟踪程序无法通过 P2P 机制跟踪到所有的 Bot。

3.3.5 Peers 之间的 Proxy 特性

DDG 的 P2P 机制中，还有一个有意思特性：Peer 的 Proxy 功能。

当某个 Peer 来请求下载矿机程序或 Busybox 程序时，如果 DDG 经过检查发现自己工作目录中暂时不存在相应文件，它就会把自己作为一个 Proxy，向自己成功连接的其他 Peer 随机发送相应的下载请求。如果成功下载，就会返回给向自己请求下载文件的 Peer。

4. 总结

DDG 经过两年多的发展，从最初简单的挖矿木马，到借用第三方协议框架构建简单的 P2P 网络，到现在自研 P2P 协议，已经演化成了一个复杂的 P2P 僵尸网络。可能这是第一个 P2P 结构的挖矿僵尸网络。如果你怀疑自己的主机被 DDG 入侵，建议从以下几个方面排查、处置：

1. 检查 `/var/spool/cron/root` 和 `/var/spool/cron/crontabs/root` 是否存在恶意 Cron Jobs

2. 检查 `~/.ssh/authorized_keys` 文件中是否存在恶意 SSH Public Key
3. 检查 `/var/lib/`, `/usr/local/` 目录下的隐藏目录, 是否存在 DDG 工作目录
4. 检查 `/usr/bin/`, `/usr/local/bin/` 目录下是否存在可疑 ELF 文件
5. 检查 `/etc/hosts` 文件是否被写入恶意内容

5. IoCs

C&C:

67.205.168.20:8000

URLs:

<http://67.205.168.20:8000/i.sh>
<http://67.205.168.20:8000/static/wordpress>
http://67.205.168.20:8000/static/bb3/busybox.x86_64
<http://67.205.168.20:8000/static/bb3/busybox.i686>

MD5s:

5e2e0564ee03c743b50e4798c1041cea 5009/ddgs.i686
ea3d5d224ed7474159936f727db7555d 5009/ddgs.x86_64
05d7e2c36d5c58b26b00ca80ee7d8abe 5012/ddgs.i686
38fb3221d43d743a0de12d494ad60669 5012/ddgs.x86_64
91217bdbcc9f5663aac47b9fe803d4c7 5013/ddgs.i686
02e645c3bdd84d7a44b7aefc0f6d9e74 5013/ddgs.x86_64
5f4587df10ba4b3e7b46eb8b46d249bd 5014/ddgs.i686
e956e5b97cd0b73057980d735ee92974 5014/ddgs.x86_64
8e44e7a361c4d91c670072e049e6d729 5015/ddgs.i686
7f87c72701576da704056b38f6fae1ce 5015/ddgs.x86_64
6c164f25cabbcdc112192b1409ae73c5 5016/ddgs.i686
f84a0180ebf1596df4e8e8b8cfcedf63 5016/ddgs.x86_64
c2480ce231cc84130d878cb42bd072dd 5017/ddgs.i686
c8b416b148d461334ae52aa75c5bfa79 5017/ddgs.x86_64
66cd0c4c13670c32f43b0fd3304b0bf6 5018/ddgs.i686
dc87e9c91503cc8f2e8e3249cd0b52d7 5018/ddgs.x86_64
58c9a8561584dd1b70fbcb68b458f293 5019/ddgs.i686
682f839c1097af5fae75e0c5c39fa054 5019/ddgs.x86_64
28b2ee07f7a611d353efd8e037973bca 5020/ddgs.i686

```
495dfc4ba85fac2a93e7b3f19d12ea7d 5020/ddgs.x86_64
ffe204b87c5713733d5971e7479c0830 5021/ddgs.i686
d2a81a0284cdf5280103bee06d5fe928 5021/ddgs.x86_64
e2430bbeb49a11bfa30c6b01a28362c7 5022/ddgs.i686
e64b247d4cd9f8c58aedc708c822e84b 5022/ddgs.x86_64
d3a203cb0aa963529c0e4f8eccbf8c56 5023/ddgs.i686
2c4b9d01d2f244bb6530b48df99d04ae 5023/ddgs.x86_64

d146612bed765ba32200e0f97d0330c8 miner_1
d146612bed765ba32200e0f97d0330c8 miner_2
```

0 Comments

1 Login ▾



Start the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS [?](#)

Name



Share

Best [Newest](#) [Oldest](#)

Be the first to comment.

Subscribe

Privacy

Do Not Sell My Data

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

Botnet



Import 2022-11-30 11:16
**双枪团伙新动向，
借云服务管理数十
万僵尸网络**

Botnet
**The LeetHozer
botnet**

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 →](#)

本文作者: jinye, JiaYu, suqitian, 核心安全部研究员 THL 概述 近日，我们的域名异常监测系统 DNSMon 捕捉到域名 pro.csocools.com 的异常活动。根据数据覆盖度估算，感染规模超过100k。我们通过告警域名关联到一批样本和 C2，分析样本后发现是与双枪恶意程序相关的团伙开始新的大规模活动。近年来双枪团伙屡次被安全厂商曝光和打击，但每...

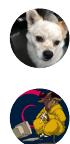


May 23, 21 min

2020 read



Background On March 26, 2020, we captured a suspicious sample 11c1be44041a8e8ba0 5be9df336f9231. Although the samples have the word mirai in their names and most antivirus engines identified it as Mirai, its network traffic is totally new, which had got our attention. The sample borrowed some of Mirai's...



Apr 27, 11 min

2020 read

