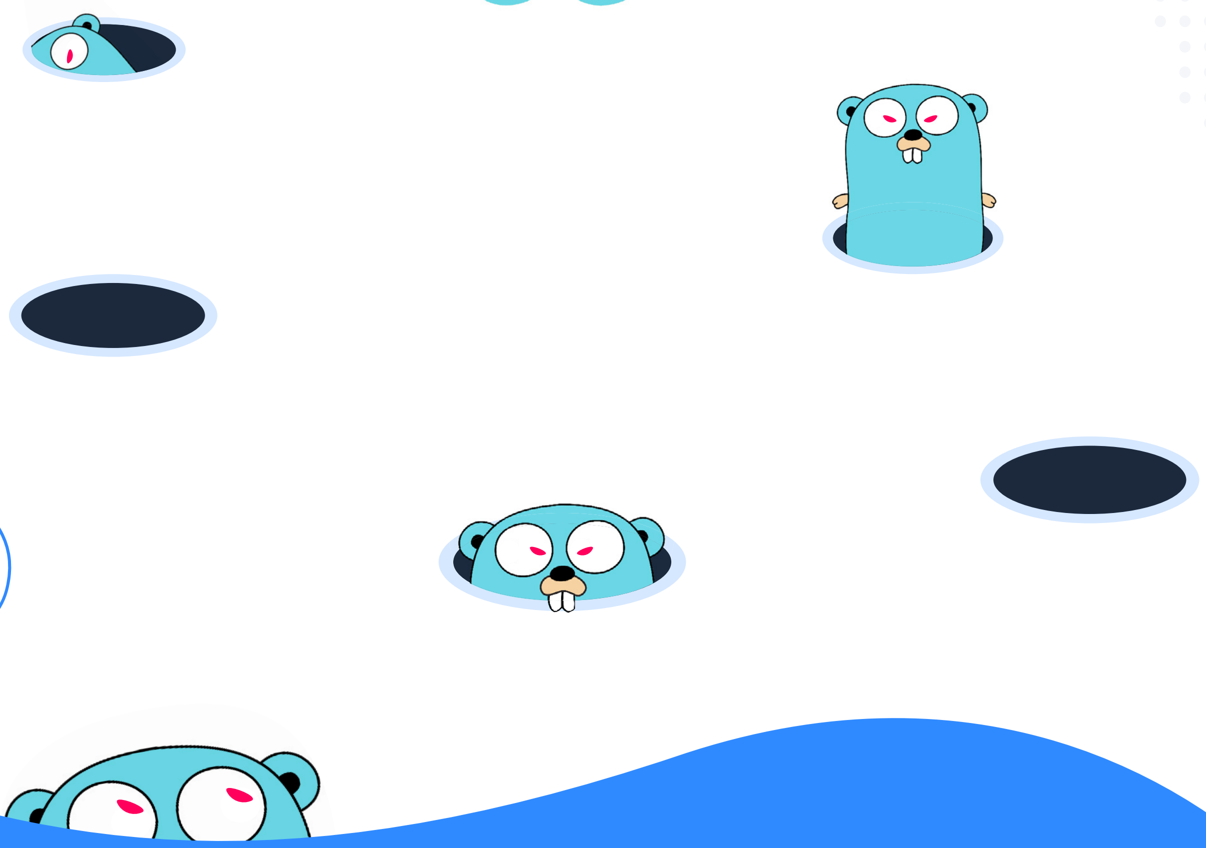
INTEZER

Year of the Gopher

A 2020 Go Malware Round-Up



Executive Summary

Malware written in Go has been steadily increasing in the last few years. During this time we have seen both nation state-backed and non-nation state threat actors adopt Go into their toolset. This report outlines the uses of Go malware by these threat actors during 2020.

Before 2020, a Russian nation state-backed threat actor had been using their Go variant of Zebrocy. In 2020, we saw a return of this malware in targeted attacks against Eastern European countries. Another Russian nation state-backed threat actor was attributed to a malware called WellMess that is written in Go. According to the UK's National Cyber Security Centre (NCSC), it was used in attacks against organizations that were part of COVID-19 vaccine research. WellMess has been around for a couple of years but before the NCSC's report, no attribution to a known threat actor had been made in the public. A Chinese nation state-backed threat actor utilized a new loader written in Go to execute their malware in some campaigns, and a new threat actor included two malware written in Go in attacks against Tibetan individuals.

On the non-nation state-backed front, crypters, stealers, remote access trojans (RATs), botnets, and ransomware were used. Some botnets active before 2020 were still active while some new ones emerged. Some started to target Linux environments. For example, new samples of IPStorm were discovered attacking Linux machines instead of Windows ones. The majority of botnets targeting Linux machines installed cryptominers or commandeered the machine to take part in distributed denial of service (DDoS) botnets. Windows machines were targeted by ransomware written in Go. During the year, some new ransomware were used in so-called *Big Game Hunting* attacks. Nefilim and EKANS infected Whirlpool and Honda respectively.

It's likely that the number of Go malware will continue to increase. We have seen threat actors targeting multiple operating systems with malware from the same Go codebase. Traditional Antivirus programs have had a hard time identifying Go malware due to many factors. A detection method based on code reuse has shown to be effective, especially when it comes to detecting when malware families are targeting new platforms. It's likely that attacks from Go malware against cloud environments will increase as more valuable assets are moved to the cloud. Using the security features provided by the hosting providers will not be enough and specialized runtime protection solutions will be needed.

Introduction

Go is an open-source programming language that was developed in 2007 by Robert Griesemer, Rob Pike and Ken Thompson at Google. It was released to the public in November of 2009. The motivation for developing a new language stemmed from the frustrations of working with current programming languages. As CPUs weren't getting faster by increasing the number of clock-cycles anymore. Instead more speed started to be obtained by adding more cores, allowing for more execution in parallel. This evolution in hardware had not been reflected well in the common programming languages. While languages such as C, C++ and Java provide functionality for executing things in parallel on multiple cores, they provide programmers with little help to do it efficiently and safely.

The programmers at Google set out to design a new programming language that would provide "first class" support for concurrency or parallelism easily and safely. The goal was also to combine the ease of programming in an interpreted language with the efficiency and safety of a statically typed and compiled language. As it was designed at Google to be used for network services running as part of Google's infrastructure, network support was also important.

To provide the feeling of programming in an interpreted language, Go uses garbage collection and handles all memory management. All Go binaries include an extensive library called the *runtime*, this results in Go binaries being larger in size compared to a similar program written in C that has been statically linked. The library handles the garbage collection, scheduling of execution threads and all other critical features of the language. While it is called the *runtime*, it is more like *libc* for C that has been statically compiled with the binary than for example the Java runtime. Go binaries are compiled to native machine code, but can also be compiled to JavaScript and WebAssembly.

Versions 1.4 and earlier of the mainline compiler were implemented in C but with the version 1.5 release in 2015, the compiler was fully written in Go and self-hosting. The move to a self-hosting compiler brought a huge improvement to the user experience with regards to cross-compiling. Before when the C based compiler was used, you needed to have a C compiler for the target operating system and architecture installed on the machine where the code was compiled. Very much in the same way as when cross-compiling C code for different targets. From release 1.5 and onwards, cross-compiling to different operating systems and architectures is achieved by just indicating to the compiler what it should compile for. No special compiler for the target is needed. Go can achieve this by not being dependent on libraries on the host machine to perform for example syscalls. The functionality otherwise provided by libc is provided and handled by the standard library for Go. There is one limitation to this ease of cross-compilation and it is when Go programs need to interact with libraries written in C via its *foreign function interface* (FFI). The new features and solutions have resulted in programmers adopting Go for new projects. In 2016, TIOBE awarded Go the "Programming Language of the Year", an award given to the language that has had the highest rise in rating. As software developers have started to adopt the use of Go for its features, it should not come as a surprise that malware authors have also started.

In July 2019, Palo Alto Networks' Unit 42 released an analysis of malware found in the wild that was written in Go. The study found that between 2017 and 2019 an increase of 1944% more samples were found in the wild. This quantified a trend that was easy to spot. Before 2019, spotting malware written in Go was more a rare occurrence and during 2019 it became a daily occurrence. The majority, 92% of the malware analyzed in the report targeted Windows, while 4.5% targeted Linux and 3.5% macOS. One thing that has been observed is the adoption of Go by pen-testing teams for the development of their tooling. This was also prominent in the study by Unit 42. The most common type of malware family was either an open-source or pen-testing backdoor. This was followed by coinminer, stealer, and botnet. This report covers the activity of known malware written in Go that were active during the year 2020.

Nation State-Backed Threat Actors

The adoption of Go by nation state-backed threat actors hasn't been as prominent as can be seen with non-nation state-backed threat actors. It may be due to them staying with what they believe is true and tested and the advantages Go provides are not needed by the groups. With a more focused targeting, the need for malware that can be compiled for varied architecture and operating systems from the same code base is not needed. While few APT groups have included Go in their toolset, there are some exceptions. During 2020, at least three new nation state-backed threat actors were attributed using malware written in Go.

APT28 - Zebrocy

It may not have come as a surprise when the first Zebrocy written in Go was discovered in May 2018. The Zebrocy malware, known to be used by Sofacy, has been implemented in AutoIT, C++, C# and Delphi to name a few programming languages before. The Go version was used in multiple waves throughout 2018 and 2019.

In November 2020, we discovered two new samples of the Go implementation of Zebrocy. The samples had overlapping code with older Zebrocy samples, as can be seen in Figure 1.

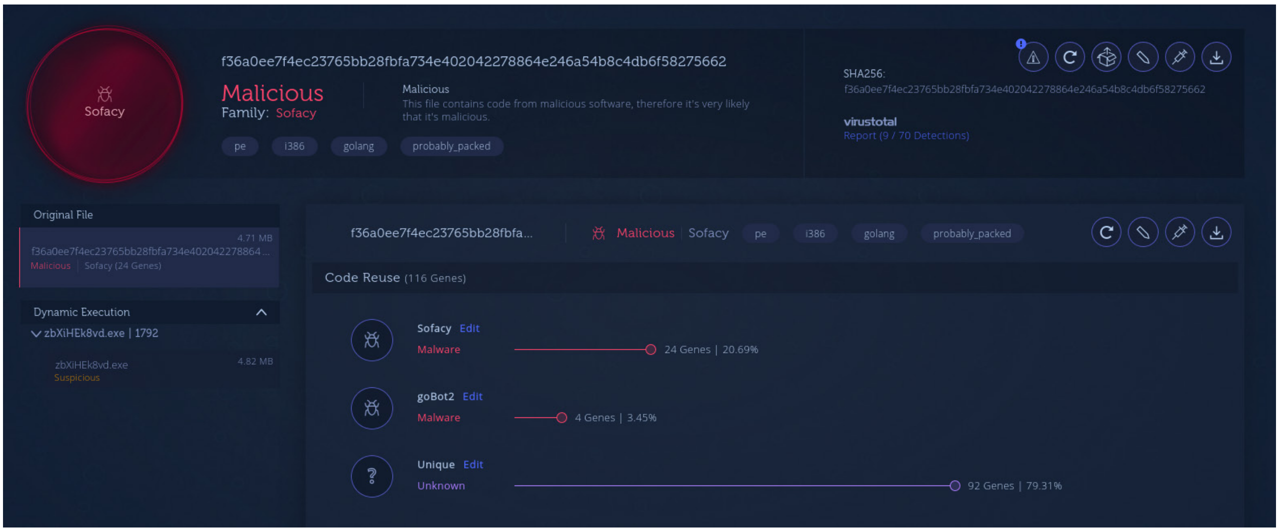


Figure 1: Code similarity detected by Intezer Analyze between the new Zebrocy sample and older samples.

Earlier in October, a government ministry of Kazakhstan had been targeted in a spear-phishing attack that was used to deliver a Delphi version of Zebrocy within a Virtual Hard Drive (VHD) image. A few weeks later, the same VHD image was uploaded to VirusTotal from the same country containing a Go version of Zebrocy. One possible scenario is that the initial attempt did not work and the threat actor switched from the Delphi version, which had been used throughout the Fall, to the Go version as they believed it might have been detected by Antivirus software.

While the initial Go version of Zebrocy wasn't obfuscated, the later ones have been. Below is a representation of the source code layout for the version used against Kazakhstan. As can be seen in the snippet, most of the function names have been obfuscated. The code-snippet below has been generated by the tool redress from the *Go Reverse Engineering Toolkit*. For more information on how this data is produced, see Appendix A.

```
Package main: _/C_/Users/user/CB3
File: main.go
    init Lines: 77 to 78 (1)
    cwnreihj Lines: 91 to 128 (37)
    v_kgzbniqbeinp Lines: 128 to 136 (8)
    DeleteDC Lines: 136 to 143 (7)
    CreateDIBSection Lines: 143 to 155 (12)
    DeleteObject Lines: 155 to 162 (7)
    BitBlt Lines: 162 to 177 (15)
    cshdqhuwymy Lines: 177 to 270 (93)
    _kwhj_z Lines: 270 to 282 (12)
    jhk Lines: 282 to 288 (6)
    mialo Lines: 288 to 316 (28)
    ictkqimqqf Lines: 316 to 338 (22)
    vmfcvvvab Lines: 338 to 346 (8)
    opjuugcz Lines: 346 to 363 (17)
    ygzwpop Lines: 363 to 394 (31)
    ipfhf Lines: 394 to 406 (12)
    main Lines: 406 to 412 (6)
```

A few weeks later a similar phishing lure was discovered that appeared to have been used to target a government ministry of Azerbaijan. This lure also consisted of a VHD file containing the Go version of Zebrocy and a PDF document. The PDF documents were presentation slides about *Sinopharm International Corporation*. Sinopharm is one of the companies in China that during the phishing campaign was working on a vaccine for COVID-19. Their vaccine was undergoing phase three clinical trials but had already been given to nearly one million people.

APT29 - WellMess and WellMail

WellMess is a malware that was first reported by LAC and JPCERT back in 2018. It was reported that attacks had been observed in Japan and that both samples targeting Windows and Linux machines had been observed. At the time, no attribution of the attacks was given. For nearly two years, WellMess was just a "bot" that was being observed arriving in waves. The attention to WellMess changed in July 2020, when the National Cyber Security Centre (NCSC) in the UK attributed the malware to APT29. As part of the report, NCSC disclosed that the malware had been used, together with a similar malware called WellMail, to target organizations involved in COVID-19 vaccine development. The report did not include any proof or data to support the attribution to APT29 and security vendors had a hard time validating the connection. For example, Kaspersky Labs findings point more in the direction of a new Chinese threat actor while Cisco Talos published that it could see weak links to either APT28 or DarkHotel.

A month later, PwC released a follow up report. By using the Program Database (PDB) paths in some of the WellMess samples, they found another sample from the same developer that had been uploaded to VirusTotal. The sample acts as an intermediate C2 server for WellMess. The intermediary receives commands from threat actors and the responses from the WellMess backdoor. The data is stored on disk until the intended receiver connects and requests it. The technique has some similarity with a previous malware used by APT29 called Seaduke.

Holy Water/Storm Cloud APT - Target Tibetan Individuals

In March 2020, both Kaspersky Labs and Volexity reported on a new threat actor, named Holy Water and Storm Cloud by the vendors, targeting Tibetan individuals via watering hole attacks. The compromised website was serving a drive-by download of a fake Adobe Flash update to highly selective visitors of the website. As part of the investigations of the group's toolset, two new malware written in Go were uncovered. The first malware is named intelsync and operates as a stager. It obtains persistence and downloads the other Go malware named Godlike12. Godlike12 is a backdoor and has a unique C2 communications channel. It's controlled via Google Drive. The malware interacts with Google Drive via an open-source library. Commands are added to a drive folder as a file that the malware downloads. The commands from the operator and responses from the backdoor are transmitted as encrypted files that are shared via Google Drive.

Mustang Panda - Go Loader

In November 2020, Proofpoint released a report on the observed activity of Mustang Panda where a new loader written in Go was used to load PlugX. The loader was delivered as part of a self-extracting RAR archive. The archive also included a legitimate application from Adobe. The legitimate application is vulnerable to DLL side-loading and Mustang Panda uses this vulnerability to side-load the loader. This is a well-known technique that threat actors have used well in the past. The archive also includes another file with its content encrypted.

The loader decrypts the file and executes it from memory. The decrypted payload is a version of PlugX, a malware commonly used by Mustang Panda.

The loader is relatively small and consists of less than 250 lines of code. The CEFProcessForkHandlerEx function in the snippet below is exported and called by the legitimate application as part of the DLL side-loading attack.

```
Package main: .
File: _cgo_gotypes.go
    _cgoexpwrap_feb1cfd10781_nghthbdbieenvrn Lines: 46 to 54 (8)
    _cgoexpwrap_feb1cfd10781_CEFProcessForkHandlerEx Lines: 59 to 61 (2)
File: hex.go
    init Lines: 15 to 16 (1)
    getKeyString Lines: 23 to 62 (39)
    getDatString Lines: 62 to 79 (17)
    VirtualProtect Lines: 79 to 94 (15)
    getFileSize Lines: 94 to 111 (17)
    getFileSizefunc1 Lines: 100 to 102 (2)
    readFile Lines: 111 to 142 (31)
    ProcessStart Lines: 142 to 216 (74)
    CEFProcessForkHandlerEx Lines: 216 to 231 (15)
    main Lines: 231 to 239 (8)
```

Non-Nation State-Backed Threat Actors

Loaders/Crypters

The binaries produced by the Go compiler are relatively large when compared to binaries produced by other languages. For example, a *Hello World* binary has over 1700 functions. With so much common code in the binary, it can be like finding a needle in a haystack when looking for suspicious code. This may be one of the reasons why malicious Go binaries sometimes aren't detected by Antivirus engines. This has led to some threat actors developing crypters in Go and using them to deliver other older and well-detected malware. This technique can reduce the detection and even sometimes make the malware fully undetected. Embedding other binaries within a Go binary is relatively easy. There are plenty of open-source libraries that have solved this problem. Below is a list of some of them.

- https://github.com/gobuffalo/packr
- https://github.com/rakyll/statik
- https://github.com/GeertJohan/go.rice
- https://github.com/UnnoTed/fileb0x
- https://github.com/mjibson/esc
- https://github.com/kevinburke/go-bindata
- https://github.com/lu4p/binclude
- https://github.com/omeid/go-resources
- https://github.com/pyros2097/go-embed
- https://github.com/wlbr/mule
- https://github.com/miscing/embed
- https://github.com/kyioptr/gassets

Most of these have been designed to allow the embedding of static assets for web services but the use case isn't limited to this. The functionality of embedding files has been so well regarded that earlier this year a proposal was suggested to add the functionality directly to the Go compiler. The proposal has been accepted and should be released with version 1.16 that is scheduled for release in February 2021.

In October 2020, Palo Alto Networks Unit 42 released a report on some updates to the threat actor TeamTNT. One of the samples in the report has been encrypted with an open-source tool called Ezuri. Ezuri encrypts the original binary with AES and appends it to the end of a stub file. The stub has two functions in addition to the main function as can be seen in the code snippet. When executed, it decrypts the original memory and writes it to a memory-backed file that is executed. This results in no artifacts being written to disk.

```
Package main: /root/ezuri/stub
File: <autogenerated>
    init Lines: 1 to 1 (0)
File: main.go
    runFromMemory Lines: 20 to 50 (30)
    aesDec Lines: 50 to 58 (8)
    main Lines: 58 to 59 (1)
```

Another open-source loader is the Go shellcode LoaDer. It also encrypts the payload with AES. It decrypts the payload and uses *ZwProtectVirtualMemory* to mark the decrypted buffer as *Read/Execute* before it is executed. Malware samples that are using this loader have been observed in the wild. For example, CobaltStrike beacons.

```
Package main: C:/Users/xy/Desktop/🐎/gld-master
File: temp.go
    main Lines: 10 to 11 (1)
Package gld/util: C:/Users/xy/Desktop/🐎/gld-master/util
File: util.go
    newAead Lines: 8 to 30 (22)
    D Lines: 30 to 34 (4)
Package gld/detect: C:/Users/xy/Desktop/🐎/gld-master/detect
File: detect.go
    ContinueRun Lines: 13 to 38 (25)
    checkNic Lines: 38 to 74 (36)
    checkResource Lines: 74 to 106 (32)
    detectDBG Lines: 106 to 109 (3)
Package gld/loader: C:/Users/xy/Desktop/🐎/gld-master/loader
File: loader.go
    X Lines: 22 to 24 (2)
```

We have also observed threat actors writing their own crypters and loaders. For example, we have seen a loader called *gocrypter* used to encrypt commodity malware; mostly RATs and keyloggers. The stub, shown in the code snippet below, is less than 100 lines of code. The payload has been encrypted with AES and is stored as a base64 encoded blob inside the binary. The crypter decodes it into bytes and decrypts it before it is written to disk and executed.

```
Package main: C:/Users/tarik/Desktop/gocrypter/stub
File: main.go
    main Lines: 19 to 58 (39)
    RandStringBytes Lines: 58 to 66 (8)
    TempFileName Lines: 66 to 70 (4)
    DecodeMalware Lines: 70 to 83 (13)
```

Following is a list of a few gocrypter samples with different malware as the payload.

DarkComet

- 04d4423c595966a453f6b80e250f6f8597e28f955ab83d5d624d28ab5fe68280
- 2626c7c34f928a16b15d7b071dd65b7e462cab9ceebb725d16e1ce01375124f7
- 54f45ca1b511bfe2bb416d76747bc5c3c3b2be0e226644ced863c47383f405bd
- 88c469239563e3f8efe2ad40440e962ac09319523bb252ffb5f5ec875eb17633
- 8c497ffdea3cd42dc209f4004e0957f109cfe1c6c0d2f379aa752d73688e5c00
- 9b65dbe770a0eec691b0f979b338655c004a43bb336b922d29c172a11fddad32

AsyncRAT

- 194270766c8afe4cdd99c8f1ebdbc18321bd79ac6f2f3e0c0638ea93ffe8aaf6

NanoCore

- 562fc3234e53c14974bd59e5008f264438e67849ddaf11f06c4687fdd2da5311
- d1be12bb6204e7fb0afd0756f5732337270c06a4c991644420f5cf7b5f44e354
- 3a9754cc818ca7c9952d35e7df35a63d8f5608069fba06a2f868780fb79408a5
- 613b460226664675128 94f7766671c5fac3010ae279bc47fba40280efa1c5fe2
- 69a5fc527781aa2715ffb1b9e85a45f07ff5275a34c6a743882a068ce3546638

njRAT

- a22b9193e29d49924f9948810266ca406280584a123a8dad4f3a6e413df4cc79
- 11f168910ecea89c9fe01894f96b037296c4c8c4dc5bb58523addcc19234a252
- 444a40b41e354a58c9a88747e9bd8c2b523c8cfa98f62758b607b036538585b8
- 6295901df5dbce4ecf0c8ddae05680c990b9af9a792090c7aa83c2cb443ced0f
- 81ace7aca877a08537deaa3ca9976d9b55eb449a32632723054a77a9629ed511
- cdb385d8ddc171b17103c5ff6a26152d082459b6116c94d4d363d7007992349a
- d5b509fbc1a34e81649fb9553aa8176099998ae20772a4d1ddefa0d73414764b

HAKOPS Keylogger

- 37ab3f612e8cc8d6b5e8da786ebbc293006ca2f71802e0a1320300a0a8f322b0

Glupteba

- 3a1e89ed0885a82bbc04f972c07c19f7ad4bd6a5047b47a76e1514d45a65d86a
- 60f9d229df06b465f8ada059c981f956974f1d87c3446bd6f8c866670f8f418b
- a3c820c7842c50b28fa880675950c85ebebd220043b7394a9a80ddd57c9f4387

Quasar RAT

- 3d4c2b19110d7a9909bfaebc8319d1151b925e736f20d85310cba27168556242

Rebhip

- a1463f6380fc82c1fa765949c4944367ac7e4d71b9b248cf6af889ce0cec96e0
- e039b160b23db31475cb0d145abccbd97953d045442a3b80a5e95bcff80ab913

Ardamax

- b89c79d9961077867d372a407ec29d9be517c4771aa19e3edfff4c86977ae78c

RATs

Go has a very well-written networking stack that is easy to to work with. Go has become one of the programming languages for the cloud with many cloud-native applications written in it. For example, Docker, Kubernetes, InfluxDB, Traefik, Terraform, CockroachDB, Prometheus and Consul are all written in Go. This makes sense given that one of the reasons behind the creation of Go was to invent a better language that could be used to replace the internal C++ network services used by Google. That Remote Access Trojans (RATs) have been written in Go is not much of a surprise. After all, they very much function as network services.

Throughout the year, both new RATs emerged and old ones kept being used. Back in August 2020, we discovered a Linux version of a backdoor used by the Carbanak threat actor. The sample was compiled using version 1.8 of the compiler that was released in February 2017. The same compiler version and build environment were used for the initial Windows sample that was part of the RSA report in 2017. It is likely the Linux version was also used during that time frame when someone finally uploaded a sample to VirusTotal.

Glupteba is a malware that has been around since 2011 but in September of 2019, a new version rewritten in Go was discovered. Throughout 2020, more of this new version has been seen in the wild. The malware tries to install a root-kit when it infects a machine. To by-pass protections in Windows that prevent installing kernel drivers, the malware exploits a vulnerable VirtualBox driver. The malware installs the driver, which Windows will allow since it is signed, and uses it to execute code in Ring-0 to disable Kernel Patch Protection (KPP). This technique is not new, it was first used by the APT group Turla. In addition to this, the malware tries to spread within the local network by exploiting EternalBlue.

Windows is not the only operating system that has been targeted with RATs written in Go. In October 2020, Bitdefender released a discovery of a new RAT that targets Linux. Bitdefender's researchers believe it might be related to *PowerGhost* activity from 2019. The threat actor targets WebLogic servers that are vulnerable to CVE-2019-2725. The RAT appears to be named *NiuB* by the author. The malware consists of two binaries, the *main* malware, and a *guard* malware. The structure of the *main* malware is shown in the code snippet below.

```
Package main: C:/Users/john/Desktop/NiuB/Linux&C#/src/Linux/Main
File: Main.go
    LoadEnv Lines: 49 to 54 (5)
    init0 Lines: 54 to 71 (17)
    main Lines: 71 to 125 (54)
    ReceiveSignal Lines: 125 to 141 (16)
    MakeOnlineInfo Lines: 141 to 176 (35)
    Encrypt Lines: 176 to 186 (10)
    Md5Sum Lines: 186 to 202 (16)
    killLockFile Lines: 202 to 217 (15)
    killRepeat Lines: 217 to 246 (29)
    MakeUUID Lines: 246 to 268 (22)
    GetOsInfo Lines: 268 to 308 (40)
    GetIPs Lines: 308 to 343 (35)
    InitStart Lines: 343 to 351 (8)
    Download Lines: 351 to 387 (36)
    ExecShell Lines: 387 to 400 (13)
    ChangeSelfTime Lines: 400 to 411 (11)
    SetTimeOut Lines: 411 to 418 (7)
    MsgProcess Lines: 418 to 444 (26)
```

The malware collects information about the infected machine and sends it to the C2 server. It can execute shell commands and download and execute other binaries.

The *guard* part downloads and installs the malware. It ensures that the RAT is running. The snippet below shows the function names for the guard binary.

```
Package main: C:/Users/john/Desktop/NiuB/Guard/src/Main
File: GuardMain.go
    init Lines: 29 to 31 (2)
    LoadEnv Lines: 79 to 94 (15)
    ReceiveSignal Lines: 94 to 108 (14)
    init0 Lines: 108 to 137 (29)
    InstallStartUp Lines: 137 to 167 (30)
    copyToFile Lines: 167 to 197 (30)
    ChangeSelfTime Lines: 197 to 207 (10)
    ExecShell Lines: 207 to 218 (11)
    Download Lines: 218 to 250 (32)
    GetVersion Lines: 250 to 292 (42)
    killLockFile Lines: 292 to 303 (11)
    killRepeat Lines: 303 to 338 (35)
    run Lines: 338 to 428 (90)
    IsDir Lines: 428 to 439 (11)
    GuardProc Lines: 439 to 470 (31)
    Md5Sum Lines: 470 to 486 (16)
    Update Lines: 486 to 514 (28)
    main Lines: 514 to 518 (4)
    mainfunc1 Lines: 518 to 528 (10)
```

In January 2020, FireEye released a report on attacks targeting NetScaler devices. The attacks were exploiting the CVE-2019-19781 vulnerability. As part of the attack, the threat actor used a new malware that had not been seen before. FireEye named the malware *NOTROBIN*. As can be seen in the code snippet, *NOTROBIN* has a few functionalities.

```
Package main: /tmp/b/.tmpl_ci
File: 94d79f20885ddbbfb0af34caea4a8971.go
    doFile Lines: 72 to 92 (20)
    remove_bds Lines: 92 to 110 (18)
    install_itself Lines: 110 to 147 (37)
    install_cron Lines: 147 to 150 (3)
    xrun Lines: 150 to 176 (26)
    main Lines: 176 to 205 (29)
    mainfunc1 Lines: 205 to 207 (2)
File: <autogenerated>
    init Lines: 1 to 27 (26)
```

It was written in Go and compiled to run on *BSD, the underlying operating system used by NetScaler. One interesting functionality is that the malware blocks other malware from exploiting the same vulnerability by scanning for new NetScaler template files that may have been added as part of an exploit attempt and removes them. It opens a UDP listener on port 18634 but ignores the data sent to it. It essentially acts as a mutex to ensure only one copy of the malware is running on the infected machine.

Stealers

There have been a few *stealers* that have been written in Go. In 2019, Malwarebytes reported on a stealer called CryptoStealer.Go. It is designed to steal cryptocurrency wallets and data stored in browsers, such as credit card information. The function names have been obfuscated, as can be seen in the snippet below. Throughout 2020 threat actors have continued to use this stealer.

```
Package main: C:/gInstall/380645620
File: <autogenerated>
    init Lines: 1 to 29 (28)
File: main.go
    得k艾йэр泰也艾卡愿a Lines: 170 to 180 (10)
    伊g斯赛л沙euen佩o Lines: 180 to 194 (14)
    斯了册bpu泰肯x夏哈e Lines: 194 to 202 (8)
    给fф哟斯gиrб尼罗l Lines: 202 to 210 (8)
    艾ю和姆ю艾o斯n恩t贝 Lines: 210 to 233 (23)
    т亚阿了热bp图perт Lines: 233 to 265 (32)
    佩xgъfyc可贝ь艾ь Lines: 265 to 279 (14)
    лк了伊路бмы迪y夏v Lines: 279 to 297 (18)
    вco弗罗пщsяkt路 Lines: 297 to 329 (32)
    (*DATA_BLOB)ToByteArray Lines: 329 to 335 (6)
    м可u哈斯ebm亚д斯ы Lines: 335 to 348 (13)
    用пмюfй哟jя佩泰给 Lines: 348 to 397 (49)
    非eu艾n给cz和lb罗 Lines: 397 to 441 (44)
    лкn贝克克非x和щ亚m Lines: 441 to 571 (130)
    佩亚lo艾щr佩ъb阿贝 Lines: 571 to 614 (43)
    佩亚lo艾щr佩ъb阿贝func1 Lines: 578 to 583 (5)
    佩亚lo艾щr佩ъb阿贝func2 Lines: 583 to 588 (5)
    佩亚lo艾щr佩ъb阿贝func3 Lines: 588 to 593 (5)
    佩亚lo艾щr佩ъb阿贝func4 Lines: 593 to 598 (5)
    佩亚lo艾щr佩ъb阿贝func5 Lines: 598 to 603 (5)
    佩亚lo艾щr佩ъb阿贝func6 Lines: 603 to 616 (13)
    热y卡贝xбm卡жлп迪 Lines: 614 to 628 (14)
    热y卡贝xбm卡жлп迪func1 Lines: 616 to 623 (7)
    热y卡贝xбm卡жлп迪func2 Lines: 623 to 630 (7)
    ъm肯чдь得мд艾图阿 Lines: 628 to 657 (29)
    ъm肯чдь得мд艾图阿func1 Lines: 630 to 652 (22)
    ъm肯чдь得мд艾图阿func2 Lines: 652 to 659 (7)
    得q迪н切н图伊u迪b用 Lines: 657 to 692 (35)
    得q迪н切н图伊u迪b用func1 Lines: 659 to 687 (28)
    得q迪н切н图伊u迪b用func2 Lines: 687 to 694 (7)
    dq泰saбтk昊д尼夏 Lines: 692 to 713 (21)
```

Also during 2020, a clipboard stealer written in Go was found. It appears to have been active since 2019. Based on filenames for samples uploaded to VirusTotal, the stealer is masquerading as hacking tools, suggesting it is used to target other threat actors.

The malware has a simple design. The extracted function names are shown in the snippet below. It installs itself under **AppData\Local\Support** and hides the file or the folder. It reads the clipboard and checks if it looks like a cryptocurrency address. If it does, the malware replaces the clipboard content with the threat actor's Bitcoin, Litecoin, Monero, or Ethereum wallet.

```
Package main: E:/Мои проекты/Golang/Clipper
File: <autogenerated>
    init Lines: 1 to 1 (0)
File: main.go
    ReturnCurrentDirrectory Lines: 23 to 32 (9)
    HideFileOrDir Lines: 32 to 37 (5)
    CreateHiddenDir Lines: 37 to 42 (5)
    Copy Lines: 42 to 47 (5)
    UpdateApp Lines: 47 to 66 (19)
    MaskETH Lines: 66 to 72 (6)
    MaskBTC Lines: 72 to 78 (6)
    MaskLTC Lines: 78 to 84 (6)
    MaskXMR Lines: 84 to 90 (6)
    Safe Lines: 90 to 113 (23)
    main Lines: 113 to 137 (24)
```

The Bitcoin wallet address in the malware has been active since the Fall of 2018. As can be seen in Figure 2 below, it has received 534 transactions with the value of almost 11 BTC as of this writing.

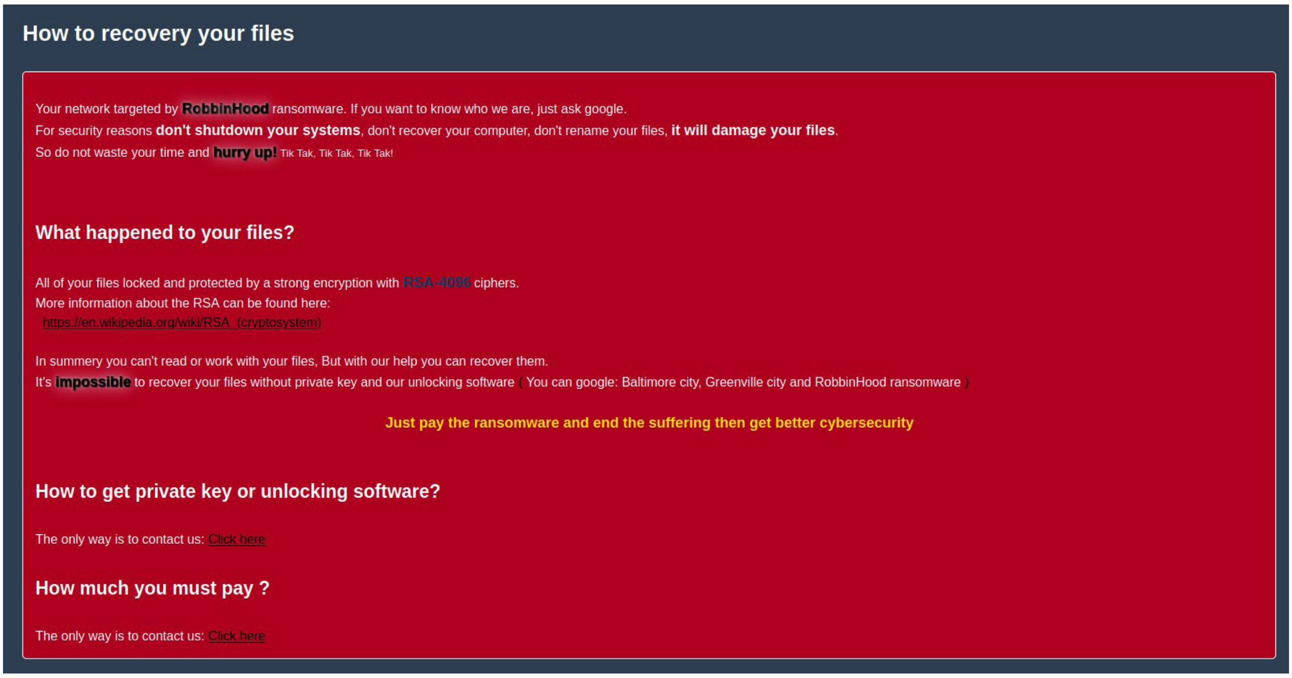
Address	1HLuWB9yEGV8q2XkhXo1nZEsDELFeXFLdo
Format	UNKNOWN (UNKNOWN)
Transactions	534
Total Received	10.97064564 BTC
Total Sent	10.97064564 BTC
Final Balance	0.00000000 BTC

Figure 2: Summary of the Bitcoin wallet used by the clipboard stealer.

Ransomware

Go's standard library provides a very robust set of crypto-libraries that allows developers to incorporate encryption in their applications without the need for using any third-party libraries. This, with the combination of the current increase in ransomware activity, is not a surprise that malware authors have started to use Go to write ransomware. During 2020 we saw both old and new ransomware families being used.

One of the ransomware that still had some activity in 2020 is RobbinHood. RobbinHood was discovered in the Spring of 2019 and got a lot of media attention when it was revealed the city of Baltimore had been attacked by the ransomware. Sophos released a report in February detailing some evolution by the threat actor. By exploiting a vulnerable driver from Gigabyte, the threat actor started to load an unsigned driver. Once the driver was loaded, it would kill processes to endpoint and tamper protection software to ensure the ransomware could encrypt the rest of the hard drive without being interrupted. Still, in November 2020, new samples were being found in the wild. As can be seen in Figure 3, the ransom note has not changed. A sample from November had the PDB string of C:/Users/User/go/src/Robbinhood7, suggesting it might be the 7th version of the ransomware according to the malware author.


How to recovery your files

Your network targeted by **RobbinHood** ransomware. If you want to know who we are, just ask google.
For security reasons **don't shutdown your systems**, don't recover your computer, don't rename your files, **it will damage your files**.
So do not waste your time and **hurry up!** Tik Tak, Tik Tak, Tik Tak!

What happened to your files?

All of your files locked and protected by a strong encryption with RSA-4096 ciphers.
More information about the RSA can be found here:
https://en.wikipedia.org/wiki/RSA_(cryptosystem)

In summery you can't read or work with your files, But with our help you can recover them.
It's **impossible** to recover your files without private key and our unlocking software (You can google: Baltimore city, Greenville city and RobbinHood ransomware)

Just pay the ransomware and end the suffering then get better cybersecurity

How to get private key or unlocking software?

The only way is to contact us: Click here

How much you must pay ?

The only way is to contact us: Click here

Figure 3: RobbinHood's ransom note used in November 2020.

Another older ransomware written in Go and still active is *Snatch*. Snatch was discovered in December of 2018 and to this day appears to still be used. The ransomware is used by Snatch Team and they target enterprise environments via remotely accessible services, for example, RDP. Once inside the network, the group tries to deploy the ransomware on all the machines and encrypt the files. The ransomware has an interesting technique when encrypting the files that was introduced to the ransomware in October of 2019. The ransomware installs itself as a service that can be started even if Windows boots into *Safe Mode*. Following this, the ransomware reboots Windows into Safe Mode allowing it to encrypt all the files on the hard drive without being blocked by any potential endpoint protection software installed. Given this functionality, the malware is relatively simple in design. A generated source code structure is shown below in the snippet.

```
Package main: /home/go/src/locker
File: config.go
    init Lines: 39 to 44 (5)
File: dirs.go
    scanDir Lines: 10 to 13 (3)
    scanDirfunc1 Lines: 11 to 14 (3)
File: files.go
    encryptFile Lines: 13 to 24 (11)
    encryptFilefunc1 Lines: 14 to 43 (29)
File: main.go
    main Lines: 24 to 91 (67)
    runInstance Lines: 91 to 99 (8)
File: misc.go
    decodeString Lines: 15 to 37 (22)
    makeFile Lines: 37 to 60 (23)
    makeFilefunc1 Lines: 43 to 68 (25)
    makeBatFile Lines: 60 to 67 (7)
    runBatFile Lines: 67 to 92 (25)
    runBatFilefunc1 Lines: 68 to 70 (2)
    isSafeBoot Lines: 92 to 115 (23)
    deleteShadowCopy Lines: 115 to 135 (20)
    selfRemove Lines: 135 to 158 (23)
    randomBatFileName Lines: 158 to 171 (13)
    Copy Lines: 171 to 174 (3)
File: queue.go
    (*Queue)Push Lines: 20 to 33 (13)
    (*Queue)Pop Lines: 33 to 45 (12)
    runWorkers Lines: 45 to 60 (15)
File: services.go
    (*myService)Execute Lines: 13 to 56 (43)
    getServicesNamesList Lines: 56 to 88 (32)
    stopServices Lines: 88 to 113 (25)
    setupServiceSafeBoot Lines: 113 to 138 (25)
    safeModeEnabled Lines: 138 to 161 (23)
    installService Lines: 161 to 188 (27)
    reboot Lines: 188 to 190 (2)
```

Nefilim is a ransomware that first emerged in March of 2020. It is the predecessor of another ransomware called Nemty. The original version was written in C++ but in July the malware was rewritten in Go. In addition to encrypting the files on the victim's machine, the threat actors behind *Nefilim* also steal the victim's data and use it for extortion. The group operates a leak site called *Corporate Leaks*, Figure 4.

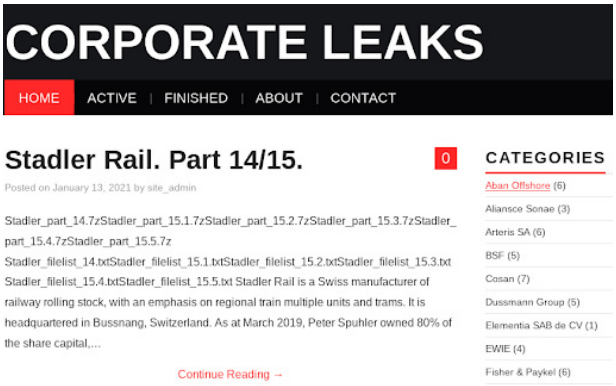


Figure 4: Screenshot of Nefilim's leak site.

According to their leak site, the threat group has at least compromised over 20 different entities and leaked some of their data. One of the biggest companies that has been compromised by this group is Whirlpool and their data started to become public in December of 2020.

Another ransomware that emerged in 2020 from a lineage of other ransomware is SatanCryptor. It first appeared in January 2020. According to security researcher Andrew Ivanov, the ransomware has its lineage from *Satan Cryptor 2.0* that first appeared in 2017. One interesting aspect with regards to the ransomware is that as it is encrypting the disk, it's providing feedback to the threat actor via a *Telegram* bot.

EKANS or also known as *Snake ransomware* is a malware that was first reported publicly in January 2020. It is a ransomware that is being used in highly targeted attacks against high-value targets. The first assumed victim reported in the public was The *Bahrain Petroleum Company* (BAPCO). The connection was made based on the email address left in the ransom note by the malware. At around the same time frame, BAPCO experienced a compromise where a new wiper named *Dustman* was used. Dustman is an updated version of the malware *ZeroCleare* that shares some similarities with *Shamoon*, a "cyber weapon" used by an Iranian threat actor. It is unclear if there is a link between the EKANS sample discovered and the Dustman compromise. The ransomware includes a list of processes that it tries to kill before it encrypts the files on the disks. The list includes some processes that are used in Industrial Control Systems (ICS). This is not unique to EKANS and it appears that the list has a significant overlap with another ransomware's kill list, the ransomware is known as MegaCortex.

On May 4th, the ID Ransomware saw a spike in EKANS samples submitted. This was later correlated to a ransomware attack that affected *Fresenius Group*, a hospital provider in Europe. According to CCN-CERT the malware tries to resolve the domain name **ADS.FRESENIUS.COM** and compares the returned IP address to a known private IP address. If the returned IP address doesn't match what the malware expects it to be, it doesn't perform anything malicious. Essentially, limiting the damage to the Fresenius environment only.

The method has been used by later versions of the ransomware to limit the spread of the malware. In June 2020, two new samples were uploaded to VirusTotal that both resolved two different internal domains. One tried to resolve **MDS.HONDA.COM** while the other tried to resolve **ENELINT.GLOBAL**. A day later, Honda released a statement on Twitter that "Honda Customer Service and Honda Financial Services are experiencing technical difficulties and are unavailable." Edesur Argentina, which is part of Enel Argentina, also released a similar statement. Not many attacks with EKANS have been reported in the public so it appears that the threat actor behind the ransomware is very selective and only uses it or targets high-value entities.

On October 19th, 2020, Malwarebytes discovered a phishing attack targeting staff at the University of British Columbia. The phishing campaign was used to deliver a new ransomware not seen before called *Vaggen*. The ransomware's function names are named in Swedish, as can be seen in the code snippet below. The files are encrypted with a static key that is “du_tar_mitt_hjart_mina_pengarna0”. Translated from: "You take my heart my money". The ransomware asked for a low fee of 80 USD in Bitcoin to decrypt the files.

```
Package main: /home/agent/barracks/op-lucifer
File: cryptme.go
    SPRINGA Lines: 82 to 117 (35)
    SPRINGAfunc1 Lines: 83 to 93 (10)
    LAMNARDETTA Lines: 117 to 143 (26)
    HIDDENBERRIES Lines: 143 to 167 (24)
    ELDBJORT Lines: 167 to 207 (40)
    DUVETVAD Lines: 207 to 218 (11)
    FOLOJVAG Lines: 218 to 224 (6)
    main Lines: 224 to 236 (12)
```

Using code reuse analysis, see Figure 5, other malware written by the same author were found. In addition to using Swedish words for function names, they also reference op-lucifer in the PDB path as can be seen in the code snippet below.

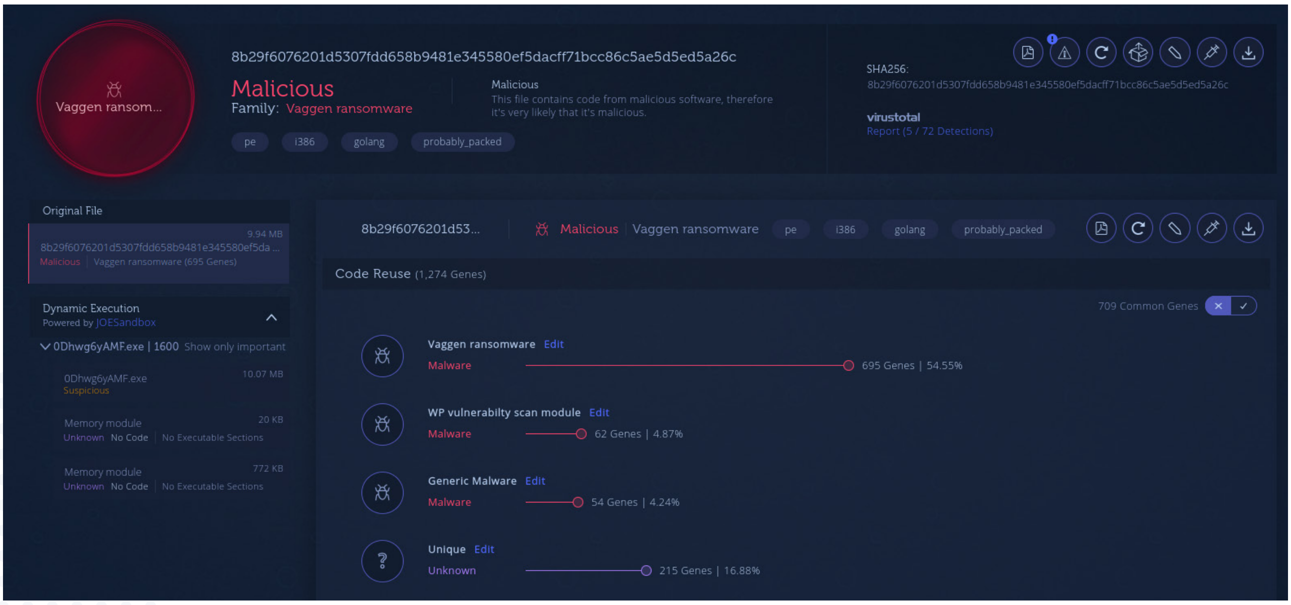


Figure 5: Code reuse analysis between Vaggen ransomware and another malware written by the same threat actor.

```
Package main: /home/agent/barracks/op-lucifer
File: cleaner.go
        KILLME Lines: 29 to 56 (27)
        TVINGAMIG Lines: 56 to 79 (23)
        RINGAHUSET Lines: 79 to 90 (11)
        main Lines: 90 to 110 (20)
```

One of the malware written by the threat actor appears to be a DNS reflection tool. The function names are shown in the code snippet below.

```
Package main: /home/yakubets/jobbet
File: cleaner.go
        FUNCTION_213 Lines: 51 to 63 (12)
        DNSreflection Lines: 63 to 114 (51)
        FUNCTION_281 Lines: 114 to 120 (6)
        FUNCTION_285 Lines: 120 to 208 (88)
        FUNCTION_21 Lines: 208 to 222 (14)
        FUNCTION_224 Lines: 222 to 237 (15)
        FUNC_66 Lines: 237 to 250 (13)
        FUNCT_0150 Lines: 250 to 320 (70)
        FUNCT_0150func1 Lines: 258 to 260 (2)
        FUNCT_1150 Lines: 320 to 339 (19)
        FUNCTION_123 Lines: 339 to 346 (7)
        main Lines: 346 to 360 (14)
```

Since Go provides a simple way of cross-compiling binaries for different architectures and operating systems, it is not a surprise that it is being used for RaaS ransomware. It allows the threat actor to use a single code base and produce binaries targeting different operating systems with a very low effort. Go has already been used in RaaS, as described earlier with Shifr. During the Spring of 2020, a new RaaS was announced called Smaug. Smaug is a relatively simple ransomware but it provides "users" of the service ransomware for Windows, Linux, and macOS. It can operate in an "enterprise" mode where one key is used for all machines or one key per machine mode. For the trouble, the operator of the RaaS takes a 20% cut.

That Go can produce binaries for other operating systems and architectures allows threat actors to easily target different types of devices, for example, embedded systems as such. In the Summer of 2019, we discovered QNAPCrypt, which is also known as eCh0raix, a ransomware that targeted QNAP NAS devices. It was later also used to target Synology NAS devices. In 2020, a new ransomware targeting QNAP devices was discovered. The new ransomware is called AgeLocker after its use of the open-source encryption tool and library age.

Other ransomware written in Go that were discovered during 2020 include: Betasup discovered in January, Sorena is also known as HackForLife and Vash discovered in February, GoGoogle discovered in March.

Bots

With Go having the support of many network protocols as part of the standard library and the ease to compile binaries for different architectures, it is not surprising more and more bots are being written in Go. The addition that the binary includes everything it needs to run properly provides the authors of the code more reassurance that it will run on for example different Linux distributions. It doesn't have to worry that a library is already installed on the machine or not. Because what it needs, it brings with it. There are also plenty of third-party libraries that provide the functionality to access other services.

For example here is a list of some bot libraries that can be used to develop bots for different services.

- https://github.com/go-joe/joe
- https://github.com/bot-api/telegram
- https://github.com/shomali11/slacker
- https://github.com/go-chat-bot/bot
- https://github.com/frodsan/fbot
- https://github.com/go-telegram-bot-api/telegram-bot-api
- https://github.com/tucnak/telebot

With open-source bot libraries available, it is not uncommon for them to be misused by malware authors. One example is IRCFlu. IRCFlu is an IRC bot that is hosted on GitHub. Figure 6 shows the project listing on GitHub. The library provides functionality to execute arbitrary code on the machine hosting the bot, which allows threat actors to use this bot to remote control multiple infected machines.

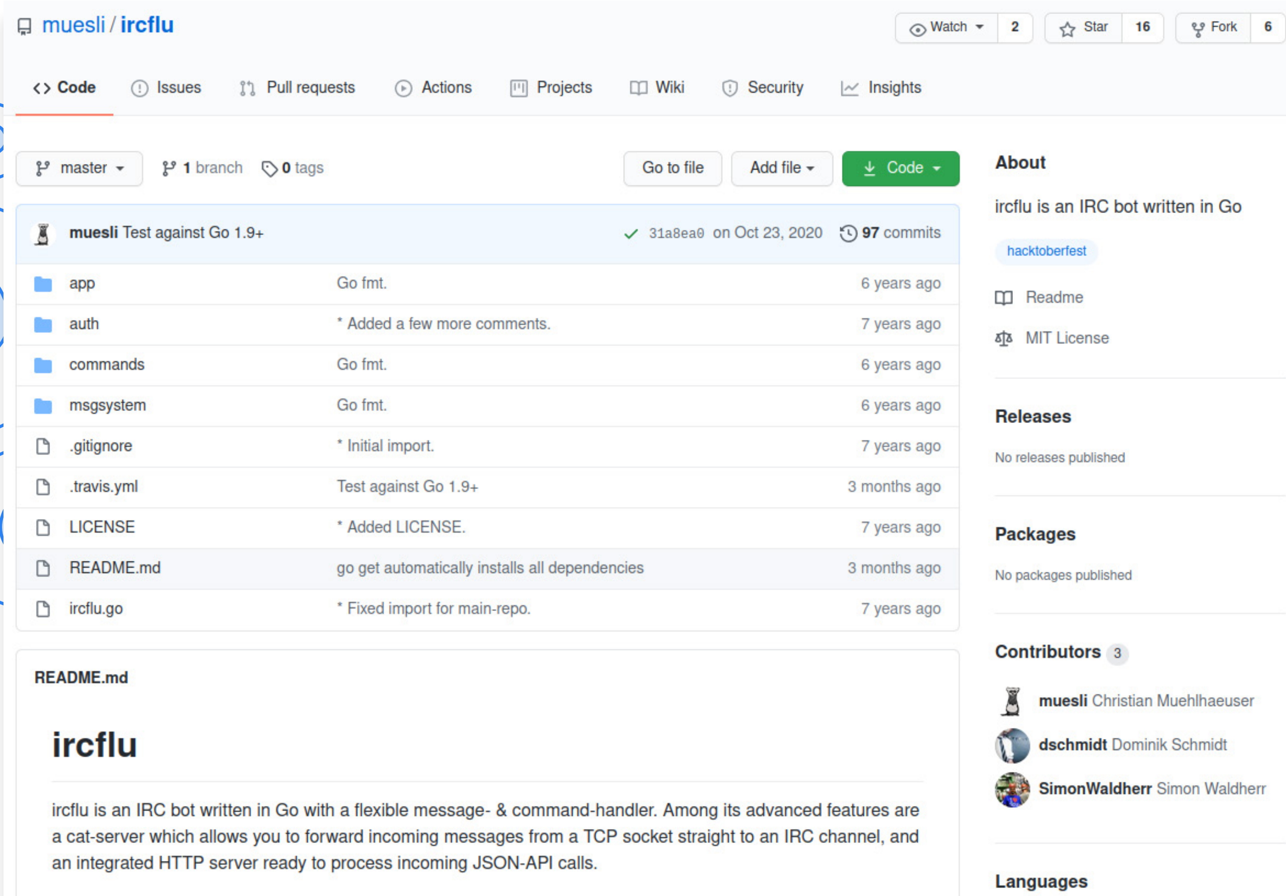


Figure 6: Screenshot of IRCFlu's source code repository hosted on GitHub.

In addition to open-source projects being misused, 2020 also saw the continuation of attacks from old and well-known bot networks. The botnet known as *ddg* was first reported on by Netlab at 360. They detected attacks against servers hosting OrientDB in October 2017 from the botnet. The goal for the botnet was to install Monero miners. In 2020, the botnet was updated to be more resilient against take-downs by adding a p2p network backed C2 infrastructure. The hybrid p2p network infrastructure allows the threat actor to maintain control over the bots if the normal C2 servers go down. In April of 2020, the botnet had about 20,000 bots with the majority of them being located in China.

Another older botnet that still is active is *StealthWorker*, also known as *GoBrut*. StealthWorker was first reported on by Malwarebytes in February 2019. It is a bot that is sold on dark web forums under the name of *Stealth Bomber* and is used to gain access to network services via credential brute force attacks. By analyzing the function names, shown in the code snippet below, we can see the services the bot has support for. The list includes web services such as Drupal and WordPress but also the e-commerce platforms Magento and OpenCart that if the threat actor gains access can be used to launch Magecart style attacks.

The botnet *r2r2* is another bot spreading via brute-forcing credentials. It was first discovered back in 2018. It randomly generates IP addresses and tries to gain access to services running SSH with weak credentials. Once it has gained a foothold, it installs a cryptominer on the machine. The bot has very limited functionality, it consists of less than 200 hundred lines of code as can be seen in the code snippet below. The malware has not changed much since 2019, Figure 7 shows similarities between a sample found in the wild in 2020 to samples indexed in 2019. The code overlap is 100%.

```
Package main: C:/Users/host/Desktop/Need/upd
File: rrr.go
	_2POST Lines: 20 to 64 (44)
	_send_good Lines: 64 to 73 (9)
	_ssh_brute Lines: 73 to 141 (68)
	_ssh_brutefunc1 Lines: 101 to 157 (56)
	_random Lines: 141 to 144 (3)
	_one_tread Lines: 144 to 153 (9)
	_start Lines: 153 to 172 (19)
	_startfunc1 Lines: 157 to 176 (19)
	main Lines: 172 to 175 (3)
	init Lines: 176 to 176 (0)
```

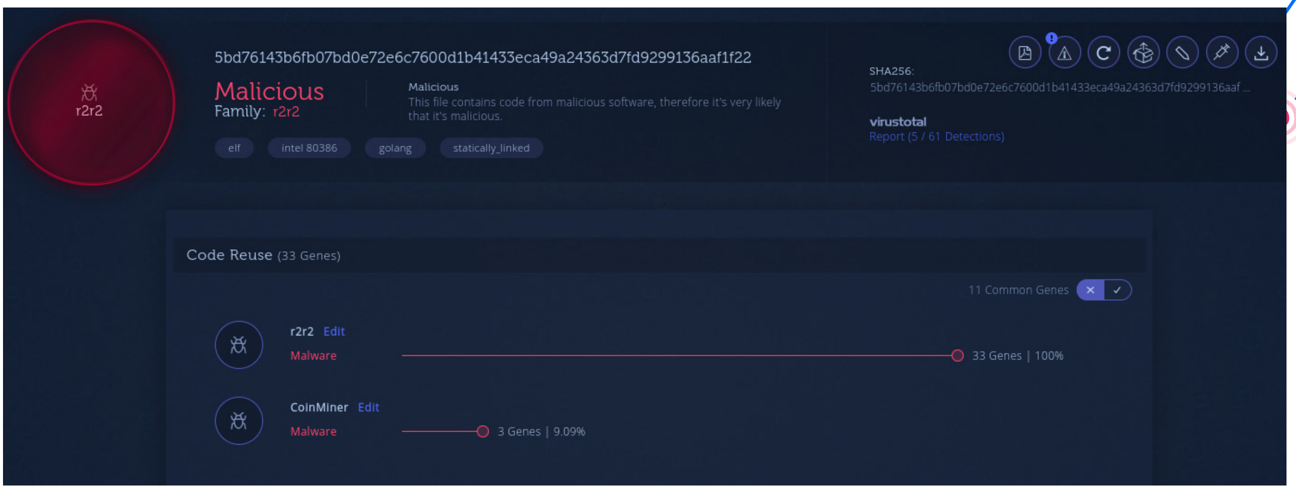


Figure 7: Similarity between 2020 and 2019 samples of r2r2 bot. The similarity shows a 100% match.

Other botnets have evolved to increase their potential targeting. In 2020 the Orthrus, also known as Golang, evolved to also target *Windows* servers. The bot was first reported on by Antiy in June 2019. It has mainly targeted *Redis* servers that are either unprotected or have weak credentials. Once it gains remote code execution, it installs a set of binaries. One is a scanner for other vulnerable services, a watchdog service, and a cryptominer. The scanner tries to compromise other network services with known vulnerabilities. For example, Weblogic, Elasticsearch, and Drupal are targeted. In 2020, the malware also added targeting against Microsoft SQL Servers. It tries to gain access via brute-forcing the credentials. The malware includes a list of nearly 3000 passwords that it only uses against SQL servers. In December we uncovered another cross-operating system mining bot that we call *XMRig Miner Dropper*. It targets servers running MySQL Tomcat and Jenkins with weak credentials, or vulnerable WebLogic. Depending on the underlying operating system, the bot delivers either a payload for executing a shell script or a PowerShell script. Once it has compromised the machine, it installs a cryptominer and tries to exploit other servers.

In September 2016 the source code for Mirai was released. This has resulted in many new botnets derived from the *Mirai* source code. While the bot code was written in C++, the release of the code has served as a blueprint for other malware authors to write similar bots in different languages. In January 2020, Bitdefender released a report on a new Mirai-inspired botnet written in Go that they named LiquorBot. The botnet is essentially a reimplementation of Mirai in Go and targets Linux devices running on ARM (both 32 and 64 bit), x86 (32 and 64 bit), and MIPS. As can be seen in the code snippet below, the bot spreads by brute-forcing SSH credentials and by exploiting known vulnerabilities in routers. Once it has gained access to a device, it tries to infect others and also installs a Monero cryptominer.

```
Package main: /home/woot/Рабочий стол/webliquor
File: <autogenerated>
    init Lines: 1 to 1 (0)
File: brute.go
    (*SSHBrute)Start Lines: 26 to 31 (5)
    (*SSHBrute).Startfunc1 Lines: 35 to 89 (54)
    (*SSHBrute).Start.func11 Lines: 36 to 38 (2)
File: command.go
    processCommand Lines: 9 to 14 (5)
File: config.go
    initializers Lines: 11 to 12 (1)
File: exploit.go
    (*Exploit)LinksysExploit Lines: 14 to 32 (18)
    (*Exploit)HnapExploit Lines: 32 to 50 (18)
    (*Exploit)DlinlkExploit Lines: 50 to 66 (16)
    (*Exploit)Dlink930L Lines: 66 to 85 (19)
    (*Exploit)Dlink815 Lines: 85 to 104 (19)
    (*Exploit)NetgearMulti Lines: 104 to 117 (13)
    (*Exploit)Netgear7000 Lines: 117 to 134 (17)
    (*Exploit)SendPayload Lines: 134 to 143 (9)
File: main.go
    (*Bot)knock Lines: 88 to 148 (60)
    (*Bot).knockfunc1 Lines: 89 to 149 (60)
    (*Bot)report Lines: 148 to 157 (9)
    (*Bot).reportfunc1 Lines: 149 to 158 (9)
    (*Bot)SendExploit Lines: 157 to 173 (16)
    (*Bot).SendExploitfunc1 Lines: 158 to 160 (2)
    init0 Lines: 173 to 205 (32)
    main Lines: 205 to 222 (17)
    spoofedMain Lines: 222 to 235 (13)
File: scanner.go
    (*Scanner)Start Lines: 31 to 84 (53)
    (*Scanner).Startfunc1 Lines: 33 to 98 (65)
    (*Scanner)_scan Lines: 84 to 90 (6)
    (*Scanner)._scanfunc1 Lines: 98 to 112 (14)
    (*Scanner)._scanfunc2 Lines: 112 to 113 (1)
File: utils.go
    (*Utils)DownloadFile Lines: 27 to 53 (26)
    (*Utils)rndIP Lines: 53 to 102 (49)
    (*Utils)rnd Lines: 102 to 107 (5)
    (*Utils).rndfunc1 Lines: 104 to 104 (0)
```

LiquorBot is not the only Mirai-inspired botnet. In April we identified Kaiji, a botnet targeting Linux servers and IoT devices via SSH brute-forcing. In addition to brute-forcing weak credentials, the bot also tries to use local SSH keys found on the infected machine to spread to other machines within the enterprise. In a similar vein as Mirai, Kaiji allows the botmaster to launch DDoS attacks against any infrastructure of their choosing. The attacks include two TCPFlood implementations (one with raw sockets), two UDPFlood implementations (one with raw sockets), IPSpoof attack, SYNACK attack, SYN attack, and ACK attack. The malware has, as can be seen in the code snippet below, some function names that are Latin letter representations of Chinese words which suggests the malware is written by a Chinese speaker.

```
Package main: /root
File: 2.go
    Link Lines: 24 to 82 (58)
    quhuiduankou Lines: 82 to 93 (11)
    doTask Lines: 93 to 117 (24)
    runmain Lines: 117 to 123 (6)
    main Lines: 123 to 205 (82)
    timeall Lines: 205 to 224 (19)
    timead Lines: 224 to 238 (14)
    runganran Lines: 238 to 245 (7)
    addtime Lines: 245 to 251 (6)
    Getjiechi Lines: 251 to 280 (29)
    rundingshi Lines: 280 to 293 (13)
    runghost Lines: 293 to 315 (22)
    runprofile Lines: 315 to 340 (25)
    Jiechixieru Lines: 340 to 349 (9)
    Jiechigo Lines: 349 to 356 (7)
    runkshell Lines: 356 to 395 (39)
    runshouhu Lines: 395 to 411 (16)
    runkaiji Lines: 411 to 436 (25)
```

In June 2020, Kaiji expanded its targeting method to include servers with exposed API sockets. The malware started to scan the internet for hosts with port 2375 exposed. If it found one it would try to deploy a rogue Docker container and execute Kaiji in the container.

Kaiji is not the only botnet that has been targeting exposed Docker APIs. In November 2020, NetLab 360 reported a discovery of a new malware called Blackrota. Kinsing, also known as h2Miner, has been known to target Docker APIs. Kinsing was first reported on by the researchers at Alibaba Cloud in January 2020. The botnet was using *masscan* to find machines exposing Hadoop Yarn, Redis, and Docker. When it found a server running any of these services, it would try to exploit known vulnerabilities in the services to spread itself further. In May, we observed Kinsing exploiting *CVE-2020-11651* and *CVE-2020-11652*, two vulnerabilities in *SaltStack*, to spread. The malware also started to use an *LD-PRELOAD* userland rootkit to hide its process.

SSH brute-force has become one of the staple attacks employed by botnets written in Go. We found a new Linux variant of IPStorm that included this attack vector. IPStorm is a peer-to-peer (p2p) botnet that was first discovered in May 2019. It uses the open-source project IPFS as its network backbone. In addition to original Windows variants, we also found as part of the Linux variant, variants targeting Android and IoT devices. Unlike other botnets in this report, IPStorm's goal is not to install a miner. Instead, the botnet appears to provide a proxy network. This proxy network is sold as an anonymous proxy network on the internet.

IPStorm is not the only Go written p2p network that was active in 2020. In August 2020, Guardicore released a report on a new p2p botnet they had been tracking since January the same year. The botnet was named *FritzFrog* and infected machines by brute-forcing weak credentials. According to Guardicore, the botnet has successfully breached over 500 servers that included "known high-education institutions in the U.S. and Europe, and a railway company". The malware does not include a persistent mechanism to survive reboots of machines; the binary is deleted by itself after it has been executed. Instead, the botnet employs persistence via reinfection. A public SSH key is added to the *authorized keys* for the machine, allowing the threat actor to gain access to the machine via SSH and reinfect it.

Future Predictions

While the amount of malware written in Go is relatively small compared to malware written in other languages, the increase on a year-over-year basis is significant. This rate of increase is very likely to continue, meaning malware written in Go will become much more frequent. For malware targeting Linux environments, the fraction written in Go is greater than for malware targeting Windows. This will likely lead to, based on total malware targeting the specific system, the fraction of malware targeting Linux systems will likely become greatest.

A big part of the current Linux malware written in Go are bots that either are used for DDoS or installing cryptominers. This trend is likely to continue. Other types may also become more frequent. We have already seen Go ransomware targeting Linux systems and it is possible that more will emerge with the goal of stealing and encrypting valuable data.

This aligns with Proofpoint's prediction for 2021 that ransomware threat actors will start to focus more on attacking the cloud. This means companies should adopt cloud-focused detection and prevention products, such as Intezer Protect, to ensure their cloud environments are protected. Many traditional Antivirus and endpoint protection solutions have been designed to protect Windows environments while Linux environments have become more of a "second class citizen".

According to CrowdStrike's report on incidents from 2020, in 40% of all incidents, the malware was not detected by Antivirus products. In addition to this, Go malware has been hard to detect by Antivirus products so it's likely this trend will continue. We have seen threat actors pivot and target different operating systems with the same code base for the malware, resulting in low or undetected malware samples. Since the malware is derived from the same codebase, detection methods that use code genes are very effective. It's likely we will see more malware targeting multiple operating systems in the future since programming languages like Go provide an easy way for malware authors to cross-compile their malware.

On the Windows side, many threat actors have used Go for ransomware. It is likely this trend will continue in the future. With the emergence of more RaaS offerings, it's not unlikely that the ransomware will be written in Go. With the ability to easily cross-compile, the RaaS operators can offer broader targeting to their "customers".

As we have seen an increase of non-nation state-backed threat actors adopting Go into their tooling, we will also start to see more nation state-backed threat actors do the same. This adoption will first be focused on the Windows side with the rise of new loaders and RATs. Once the tooling has become well established in the Windows environment, it will be used in Linux environments too.

Some notable ransomware written in Go is *Nefilim*, *EKANS*, and *RobbinHood*, ransomware used in what's called *big game hunting* attacks.

It is not just non-nation state-backed threat actors that have adopted Go, nation state-backed threat actors have too. In 2020 we saw a return of the Go variant of Zebrocy that was used against targeted attacks against foreign affairs-focused entities in Eastern Europe. The UK's NCSC attributed attacks against entities working on the COVID-19 vaccine to APT29, a Russian nation state-backed threat actor. As part of the report, they attributed the WellMess malware to the threat actor. WellMess has been around for a few years but had not before this been attributed to a specific threat actor. Chinese nation state-backed threat actors also started to use Go malware as part of their toolset. Malware written in Go was used against targeted attacks against Tibetan individuals and we also saw Mustang Panda introduce a new loader written in Go that was used to execute PlugX.

With more adoption of Go in toolsets for both non-nation state-backed and nation-state-backed threat actors seen in 2020, it is likely this trend will continue in the future. Traditional Antivirus solutions still appear to struggle detecting malware written in Go. Newer techniques that not only determine maliciousness based on code reuse but also classify the threat have seen greater success as they can handle similarities even between Linux and Windows binaries. While malware written in Go may still be in its infancy, it may soon reach adolescence resulting in a considerable increase.

Conclusion

Go is an open-source programming language that was developed inside Google to take advantage of advancements done in hardware over the last few decades. It is designed to make it easy for developers to produce fast, safe, network-focused code that takes advancement on today's multicore CPUs. This has resulted in a great adoption of the language, especially in the cloud environment. Developers are not the only ones that have adopted Go. In the last few years, almost an increase of 2000% of new malware written in Go has been found in the wild. Many of these malware are botnets targeting Linux and IoT devices to either install crypto miners or enroll the infected machine into DDoS botnets. Also, ransomware has been written in Go and appears to become more common.

Appendix A -
Go Reverse Engineering Toolkit

The *redress* software is a tool for analyzing stripped Go binaries compiled with the Go compiler. It extracts data from the binary and uses it to reconstruct symbols that can be used to perform analysis. It essentially tries to "dress" a "stripped" binary. It is open-source and can be downloaded from its GitHub page.

Binaries compiled with the Go compiler include a large set of metadata. This metadata can be used to assist static analysis of stripped binaries. Stripped binaries, especially statically compiled, are hard to analyze. Since the symbols have been removed and the huge number of subroutines—hello world binary in Go has thousands of subroutines—it can be very time-consuming. Fortunately, the metadata in the binary can be used to reconstruct symbols and also recover information about the source code layout. One of the elements that can be recovered is function information. All this metadata can be found via the *moduledata* structure.

The *moduledata* structure is a data table that was first introduced in version 1.5 of Go. It is a structure that holds important information that is needed when you statically analyze Go binaries. It records information about the layout of the executable. For ELF binaries, the structure can be found in the ".noptrdata" section. In PE files it is much harder to find. Sometimes it is located in the ".text" section and sometimes in the ".data" section. If the binary has been stripped there is no symbol pointing to the structure. In these scenarios, a brute-force search is needed.

The first entry in the structure is the "pclntable". This table holds mappings between source code line numbers and the program counters. This table is used to produce meaningful stack traces during a panic. The table is not removed if the binary is stripped and if debugging information is removed. Since it holds information that maps back to the source code file, this table can be a gold mine for malware analysts. The information includes the full file path, the name of the source file at compile time and name of the function. Also using the program counter to source code line number mapping, it is possible to estimate the source code lines of functions. Since this table holds a lot of information, it is one of the big contributors to the large file size of Go binaries, particularly in larger applications.

The current implementation version in Go was released with version 1.2. The proposal both outlines the original reason for the table and some of the possible uses. The quote below states that it was added to enhance the stack traces.

> To get accurate crash-time stack traces years ago, I moved the Plan 9 symbol and pc->line number (pcln) tables into memory that would be mapped at runtime, and then I put code to parse them in the runtime package. In addition to stack traces, the stack walking ability is now used for profiling and garbage collection.

It also states that it can be used for mapping the program counter to source code line number:

> There are many interesting uses of pc-value tables. The most obvious is mapping program counter to line number.

This means, by parsing the table it should be possible to recover function information.

The proposal suggests that the standard library has code that can parse this table. Instead of implementing a parser from scratch, it is much easier to use something that is already written. Searching the standard library source code for references to the *PCLNTAB* results in some matches found in the debug package. The debug package appears to be a good starting point.

With these tools, it is possible to reconstruct a theoretical source code layout. By using the *PCToLine* function, it is possible to get the source code line numbers the function was located between. Sorting this by file and with some formatting it is possible to produce a tree structure as shown below. It can be seen in the example that the main function has all its code located in the main.go file. This file was located in the folder C:/!Project/C1/ProjectC1Dec during compile time. The output also includes a file called <autogenerated>. This is for functions generated by the compiler. In this case, the coder did not use and implement the init function so a dummy version was added by the compiler.

```
$ shasum zebrocy && file zebrocy && redress -src zebrocy
2114fb25d5243f76c85c0df68fc4bf8e93cfb19d  zebrocy
zebrocy: PE32 executable (GUI) Intel 80386 (stripped to
external PDB), for MS Windows
Package main: C:/!Project/C1/ProjectC1Dec
File: <autogenerated>
        init Lines: 1 to 1 (0)
File: main.go
        GetDisk Lines: 27 to 37 (10)
        Getfilename Lines: 37 to 52 (15)
        CMDRunAndExit Lines: 52 to 58 (6)
        Tasklist Lines: 58 to 70 (12)
        Installed Lines: 70 to 82 (12)
        Session_List Lines: 82 to 97 (15)
        List_Local_Users Lines: 97 to 119 (22)
        systeminformation Lines: 119 to 124 (5)
        CreateSysinfo Lines: 124 to 137 (13)
        ParseData Lines: 137 to 153 (16)
        SendPOSTRequests Lines: 153 to 174 (21)
        Screen Lines: 174 to 192 (18)
        GetSND Lines: 192 to 215 (23)
        PrcName Lines: 215 to 222 (7)
        main Lines: 222 to 229 (7)
```

Appendix B - Indicators of Compromise

Zebrocy

- 61c2e524dcc25a59d7f2fe7eff269865a3ed14d6b40e4fea33b3cd3f58c14f19
- f36a0ee7f4ec23765bb28fbfa734e402042278864e246a54b8c4db6f58275662

WellMess

- 00654dd07721e7551641f90cba832e98c0acb030e2848e5efc0e1752c067ec07
- 0322c4c2d511f73ab55bf3f43b1b0f152188d7146cc67ff497ad275d9dd1c20f
- 03e9adae529155961f1f18212ff70181bde0e3da3d7f22961a6e2b1c9da2dd2e
- 0b8e6a11adaa3df120ec15846bb966d674724b6b92eae34d63b665e0698e0193
- 14e9b5e214572cb13ff87727d680633f5ee238259043357c94302654c546cad2
- 1fed2e1b077af08e73fb5ecffd2e5169d5289a825dcaf2d8742bb8030e487641
- 21129ad17800b11cdb36906ba7f6105e3bd1cf44575f77df58ba91640ba0cab9
- 2285a264ffab59ab5a1eb4e2b9bcab9baf26750b6c551ee3094af56a4442ac41
- 2daba469f50cd1b77481e605aeae0f28bf14cedfcd8e4369193e5e04c523bc38
- 49bfff6b91ee71bbf8fd94829391a36b844ffba104c145e01c92732ada52c8ba
- 4c8671411da91eb5967f408c2a6ff6baf25ff7c40c65ff45ee33b352a711bf9c
- 5ca4a9f6553fea64ad2c724bf71d0fac2b372f9e7ce2200814c98aac647172fb
- 797159c202ca41356bee18c5303d37e9d2a43ca43d0ce02e1fd9e7045b925d11
- 7c39841ba409bce4c2c35437ecf043f22910984325c70b9530edf15d826147ee
- 84b846a42d94431520d3d2d14262f3d3a5d96762e56b0ae471b853d1603ca403
- 8749c1495af4fd73ccfc84b32f56f5e78549d81feefb0c1d1c3475a74345f6a8
- 92a856a2216e107496ee086e1c8cfe14e15145e7a247539815fd37e5a18b84d9
- 93e9383ae8ad2371d457fc4c1035157d887a84bbfe66fbbb3769c5637de59c75
- 953b5fc9977e2d50f3f72c6ce85e89428937117830c0ed67d468e2d93aa7ec9a
- a03a71765b1b0ea7de4fbcb557dcfa995ff9068e92db9b2dada9dd0841203145
- a117b2a904c24df62581500176183fbc282a740e4f11976cdfc01fe664a02292
- a3ca47e1083b93ea90ace1ca30d9ef71163e8a95ee00500cbd3fd021da0c18af
- b75a5be703d9ba3721d046db80f62886e10009b455fa5cdfd73ce78f9f53ec5a
- bec1981e422c1e01c14511d384a33c9bcc66456c1274bbbac073da825a3f537d
- c1a0b73bad4ca30a5c18db56c1cba4f5db75f3d53daf62ddc598aae2933345f3
- d7e7182f498440945fc8351f0e82ad2d5844530ebdba39051d2205b730400381
- dd3da0c596fd699900cdd103f097fe6614ac69787edfa6fa84a8f471ecb836bb
- e329607379a01483fc914a47c0062d5a3a8d8d65f777fbad2c5a841a90a0af09
- e3d6057b4c2a7d8fa7250f0781ea6dab4a977551c13fe2f0a86f3519b2aaee7a
- f3af394d9c3f68dff50b467340ca59a11a14a3d56361e6cffd1cf2312a7028ad
- f622d031207d22c633ccec187a24c50980243cb4717d21fad6588dacbf9c29e9
- fd3969d32398bbe3709e9da5f8326935dde664bbc36753bd41a0b111712c0950

Mustang Panda Go Loader

- 235752f22f1a21e18e0833fc26e1cdb4834a56ee53ec7acb8a402129329c0cdd
- e54b6319a2153b5a51a1422b97fd4bbc1665e8ef65a9739ab169428eab327a75

Holy Water

- 042619e01be411211ea84a3fb77049f83eb62948b27fedc59ce0fbc0180c38c3
- 5a11f3579cf078c494f241835ea3f68ac4a3585b046a53c78b50f9d11e76ea27
- 6501f16cfda78112c02fd6cc467b65adc0ef1415977e9a90c3ae3ab34f30cc29
- d9f51e0b5dda71db5af0326740dd60c420c9f22fdbc2a4852757fb35097e0201
- b658a0b0b5cce77ce073d857498a474044657daec50c3c246f661f3790a28b13
- 719d2201055fdda8281351c5069137b794d561ccee291852862234077ab37604

Ezuri

- 05be98eaeddfc932b1c527834edf2255fc35c72dafb927cddbd6b20c4857f7cc
- 0a569366eeec52380b4462b455cacc9a788c2a7883b0a9965d20f0422dfc44df
- 35308b8b770d2d4f78299262f595a0769e55152cb432d0efc42292db01609a18
- 637636632c4a3890f70f31ecf311bed09733acd88de305d268655e3ec6c62fe7
- 64fa109872ed922063979e2e3a1bd0a03d57e5130f92395dfaa3c0a58e2aff61
- 751014e0154d219dea8c2e999714c32fd98f817782588cd7af355d2488eb1c80
- 90b970d9fa76b089fcecc55e70f228a9a2ad1d1b1489880b3c27da0b9ade75ad
- b494ca3b7bae2ab9a5197b81e928baae5b8eac77dfdc7fe1223fee8f27024772
- ddbb714157f2ef91c1ec350cdf1d1f545290967f61491404c81b4e6e52f5c41f
- e15550481e89dbd154b875ce50cc5af4b49f9ff7b837d9ac5b5594e5d63966a3
- e1836676700121695569b220874886723abff36bbf78a0ec41cce73f72c52085

Go shellcode LoaDer

- 22832a2a1e82f18d74622fd93d3e527f21377a41d3e2bb307d1b5b3f2a29c73d
- 8e68ea25df2aff13a8e86ee063b1d33fc146f44dad30634b4e3e14b72e9a8126

Gocrypter

- 04d4423c595966a453f6b80e250f6f8597e28f955ab83d5d624d28ab5fe68280
- 0c01a6b672933dc759a29d43bc1090af2d1e6909b5d4f6c90da68019a5ab1757
- 0c95e2f250a4c66a12bb4a4c6a848a66b02a64851dc333b39ab474fe3ecdcc21
- 116f8a91832954fba21671769f613b64087344aab858d08058f246041dab57d8
- 11f168910ecea89c9fe01894f96b037296c4c8c4dc5bb58523addcc19234a252
- 194270766c8afe4cdd99c8f1ebdbc18321bd79ac6f2f3e0c0638ea93ffe8aaf6
- 1a97620462c979f1fae6b57482b42c41b9ba2625f707e493d67d10ad2c6941b3
- 1bf3fc14a5695054a484b6647936dd080240294757bf1a42542bf613cfc5623c
- 2626c7c34f928a16b15d7b071dd65b7e462cab9ceebb725d16e1ce01375124f7
- 29dc3a805656e59986842454a1fb1196c865cac9612212e91b8c1d90be2d73ed
- 37ab3f612e8cc8d6b5e8da786ebbc293006ca2f71802e0a1320300a0a8f322b0
- 3a1e89ed0885a82bbc04f972c07c19f7ad4bd6a5047b47a76e1514d45a65d86a
- 3a9754cc818ca7c9952d35e7df35a63d8f5608069fba06a2f868780fb79408a5
- 3d4c2b19110d7a9909bfaebc8319d1151b925e736f20d85310cba27168556242
- 444a40b41e354a58c9a88747e9bd8c2b523c8cfa98f62758b607b036538585b8
- 4dbe371e8514ec638acd82dead97ac29d3122889b7199438a79df640b6a10481
- 4ec2f044586038ff5d58c8ac4dd78b12768c9b4cfe510a279ef5f266a54d82a4
- 54f45ca1b511bfe2bb416d76747bc5c3c3b2be0e226644ced863c47383f405bd
- 562fc3234e53c14974bd59e5008f264438e67849ddaf11f06c4687fdd2da5311
- 60f9d229df06b465f8ada059c981f956974f1d87c3446bd6f8c866670f8f418b
- 613b46022666467512894f7766671c5fac3010ae279bc47fba40280efa1c5fe2
- 6295901df5dbce4ecf0c8ddae05680c990b9af9a792090c7aa83c2cb443ced0f
- 635c3e4b883aea6782c51b9dfa98e7d7bdd5d35c8b52e87dc6780df65d2d2efa
- 64e97a5844b89c1bcae6d8b880a0cc795080662376ff663579bb3949dec8f162
- 69a5fc527781aa2715ffb1b9e85a45f07ff5275a34c6a743882a068ce3546638
- 76a58e40b93e891d5b68b3967565e0cc764c0cac09bd743e9357a941e2a293b5
- 81ace7aca877a08537deaa3ca9976d9b55eb449a32632723054a77a9629ed511
- 842ced25753227f8f65e0d5400cf5284d84f0787fdd0c3e1e479443eef05a9b1
- 88c469239563e3f8efe2ad40440e962ac09319523bb252ffb5f5ec875eb17633
- 8c44426693941bca9ccef32029c3c6534d8c8a53cda49b59b18ad5f15b7571a8
- 8c497ffdea3cd42dc209f4004e0957f109cfe1c6c0d2f379aa752d73688e5c00
- 922fff4e94c57d6fda79d3befb15059a8e65e47a40c63a74717c85a1cbbce5e3
- 9b65dbe770a0eec691b0f979b338655c004a43bb336b922d29c172a11fddad32
- a1463f6380fc82c1fa765949c4944367ac7e4d71b9b248cf6af889ce0cec96e0
- a22b9193e29d49924f9948810266ca406280584a123a8dad4f3a6e413df4cc79
- a3c820c7842c50b28fa880675950c85ebebd220043b7394a9a80ddd57c9f4387
- b89c79d9961077867d372a407ec29d9be517c4771aa19e3edfff4c86977ae78c
- cdb385d8ddc171b17103c5ff6a26152d082459b6116c94d4d363d7007992349a

- d1be12bb6204e7fb0afd0756f5732337270c06a4c991644420f5cf7b5f44e354
- d38eb4ddde1887d3f55187cf4994c5042eb21976c7603af812008756c65a4caf
- d5b509fbc1a34e81649fb9553aa8176099998ae20772a4d1ddefa0d73414764b
- d7342b489bfd991d8e126ea5de6913029c6379d843ad35906d695e639598ca3e
- dd3895cd6e3ad2d5e4afbe051db2112da5ac7cedf5f695a657f642315a7d1dc8
- e039b160b23db31475cb0d145abccbd97953d045442a3b80a5e95bcff80ab913
- ef09170640e87bb7b5fe860726b9ed7b7b639c9bf9db72d6b5445c36f93448a8

Carbanak

- 2b03806939d1171f063ba8d14c3b10622edb5732e4f78dc4fe3eac98b56e5d46

Glupteba

- 04cfe913b80de7850c0f584a9a7641fef24f0f99b436dc6542735435b6158053
- 1e8ded4cd06163868aa6ed6e1c45f106ee1eaef3100fe824632e2e15dd3737d0
- 1fcbb5655c475264ec6a401fa916150d38fab859fb8e155d414eb0219bcd29fa
- 2c0374998473a5f9cb54fcf5db9ac3af224d01410fd9526bf53b7d10d956bfb9
- 3148e9b8a08ac683b358d35a56373ba924774ea187feb1b070a537b7be402a94
- 320dd11b54c418df35e125dc3d846aafacf2d5d42b8fc9e7e3120f9a11f522a5
- 401b225ad05edb2a22548571cf6c88940894357ac67c25390e331a687a5ee85b
- 42a912a389fd0ecb2923bbae3e88fccfcb482925059f1609cafe6dc412d69eea
- 4c1aa076f7b7d3886d9e1ba8f996208b2dea7f19fd6fa499507cc0e7435175ae
- 51d3f1b7ca995ed680c50219ef9991bc337931a3eb89bd64f372235c19611c28
- 585cb30550118597f0042a9ebeae42253c30fcdc8dce124297fa3a4c35e2905e
- 5a831ef6c4141d7703b7664bdf69759860cb8ab901f71d5bc4285042ad13f4e4
- 5c5d7b26614e27934ec6985e472230065bac2b7a838383b21450ed54c78b6c0a
- 71046c690b57651c084bd8d3d4eaf981e8f1899c2563c12d0714fbe7af9fd60d
- 77325d8a8cc1a3ff59ccd34c5c4cb92013ca8dc9eab2299dc1f08aa34cae7fb8
- 7831909988517043fcf04e86defd67576d48a2c9730cafc6e0f40ec0f9839d42
- 7d6262dff5f2825c30d1165aa7fb8c0679aa52c3d7cff586f6ce9ffd12d033f1
- 7e8f3239c7dcc7a845a140e2453fe2074f766634815c42b2dab38f548ce56f97
- 8721e3f8cece23345c71d461fe34a7b492428754d055f41b37da7c290ae0982d
- 89cda998570f32d2b0b968dabeec35fe61af6d68ef833be9972fb10648a5cf5b
- 8a3bd6f6e15c9945f68286da999527252596c762a607c559b1b5ad2adc60f442
- 8e33dae8b4a50c98b168d53811854226174f4d94b24bd479c193d01f645cde75
- 9395195e2be9585b1361b0b88aa671a31b0bd2b082fb346de113abe24928a57b
- 9e9be5823489c80f9038ffbbea2a406abace9f5545061385f49597779e3612ef
- a639817a72d18a164ee9bf82760f919f8e72487a73564bf7b921044ea17fb962
- ad7e9392d0d6214b298df74cd3343e02b909f88b5c8a45443305bbb08ced965d
- cac0ea62682fc3c30f67c796ec3a1ac7173a21a826e513961aeb1628c57eb3ba
- d16ae3123bf0d22787022e3123f6e756d6b6b03805f9317253ca0987c64772a4
- fbcdb2fb9938618fa4a1abf270ab5bd538c6cccea9ce54c46f99a87002d6c03e

NiuB

- 59fa110c24920aacbf668baacadce7154265c2a3dca01d968f21b568bda2130b
- 5b3370d125f7f93554331daf862f931f188f3de25c1003de51b7b461b3326bf0

NOTROBIN

- 2bb8e7dee899d91ac6300439a639e1e7e9590668bffc20cb2347a7f0a38d6da6
- 3bd77076be18ff772ccd0e4b78d25640d9e7a89b463ec131105c00e767e0276f
- 5b07f8a98e070a282e09c1d5762f16f01f12a925ebf514a633881ab72f75fe62
- 72595386070781bd49b1d34dc9c0b0c80443b8cb90f7cb7cccf493adc5bd1698
- add8983fcdc27cdfb3c4eebc38abd0ef7fdd4a135d9c630aa8008f55bcec02ab
- b5042ccae9c26fe1cbc17599b3f741704fbe65bbaf08de4100c7449d753c17c6
- e760b8526d944a7222b0913cf43eb63d1cefb85ed3f83ec2f9c1e8071e8f27b8

CryptoStealer.Go

- 311ac106c831b5356edf9a9f16cf965513ebd59d1d0f6f84151f4c454fa07c26

GoCryptoClipper

- 2a7736fd92bca67055c6dda10f1515dac4a18d397ccf952a97b4fdca9dff1650
- 6f712f5aa4aa5b6ac5a4b56f57f95febcc3cc7cce098a213221d704bcbbe038f

- a8b85dec3a7c78e9c9f10ee7c2751cd4d12e042dd3c55db114482c3d1832a500
- bd978ba0d723aea3106c6abc58cf71df5abe4d674d0d1fc38b37d4926d740738
- c2e079ae6c52d8585e99c12f29d9cdc329e7a1a4b47a9d23954d3d1dba450a0d
- e687f3703756b594ca228ff9d75a58f8fe5b662629e5b5b137eecb775950bf40
- f8714d700b74ab9dc1167a82b5263dba0d479027c1b5574078244cdf97b9d4a8

RobbinHood

- 7c7ef3ab31ab91a7379bc2e3f32473dfa7adf662d0c640ef994103f6022a092b

Snatch

- 03b3f0667db1e9931367de89be0a9fc8f93a44010b1c24edff6b8b14e417f9de
- 06f3c13f256ab3d2643a7e587a64d9b7c7786cc2439b3d1f0efc39438b640f44
- 08a116b54b3a91173bcbc0cc611a5f58fea4b66a5e92cddd99b3289b818cef15
- 1b89b91f8ee67758afad086fa4c166c9aa7ae25782466a92724cc18f2c63e7fb
- 1eca4b0faff8e7d07886aea3d0aca257667b303e70af998722668dc38bc30842
- 2339489be44f93b61688ee3b65d0bd2fede5f6dd494588a6268602d2f006bd16
- 2ab1a5f640b4c6392c1f56626c924a565a60eb23c804a0e3acb1a7f950ee58c4
- 3c171813acc8438701dbf3171ef7e243eb01947e2b5c23065195673283fc707b
- 40a2ace8bb6e3d7d50bb346344789c2fafd25490fdaa982134aaacce6f971995
- 41a49b2e3846eb3b5355da8ebd55725270da7b5b5e8ae33fdf0c63be44bbaff7
- 42e978a513d1bce5d9b837029a3f280220d7cabb7be556c6ee2a9e8113fd0c92
- 4493cc54062453de7634fe35b5a85d78380b99ced465b5fbffc526883ea20e20
- 47fb0fef06446d2c193ece4375d10a425162811d6cb4ff1ab7dd5bb4e0a1af48
- 55f14151a30f90f34a3c7adbbdab5a1a27d66bc260e08a4d91e4df8aebb6392d
- 58d7ce0e035789dd723b1e3cec972b4abeb83b548bb42d3b829cd6e3e88d3a7a
- 602c753ee7337a5398df34b82238dd243d6afc9aa0f2d6e75f9d5a98cb609aa9
- 6149e7211f7032b7bac61cf7b0f862dd324225994b30dadebe8e12d86849f5b2
- 6402f50275eb98f65f9cf030c7092081fc47eb74366edd9ab08beed7f6117f53
- 65345794380b6dc8e138972bdad6535ace009b00fb058641f6265a82c8ada8a4
- 6622569611a8de2b2640c84567a3be39a47a1ccda037534713a272abffdc8ae9
- 68a952802edff99a6a41c57f0b57f56f80318d1a6770a3a9019a3b76ed7f3dec
- 68fc013bce308073365e3cb16e35d6b62a7a7945f583894cdf396b9fbba115ed
- 690be9e806f882774ab1347302bd92236a16499f160f37e59992c220466493c0
- 6a39d100fd95fe494812d5cce4d2ce52478a15bfbf9492edfc7433acdc708718
- 8a99a4c58ef5e27d112f0efb4719abe01bcddb5e516114711ee4991f7922ab5f
- 958dc01812972f5ea69366c18f1e4413b7064618cc4100b3aede6f7afce46857
- a6c7c19f35322e43127582be7a7a519dd3df507863cf760929dfd04a7687faf5
- a74e6812a6706b4ed4d7efe53898e2331b5c3b36377bb5dcd10400dffd53491a
- afa011cfadf84112b9c23327d3e4b5cffcbfae05c6b501291b7d090cbb9e9fb7
- bd5ebf9632ad17e9a39393ab94ef055307ec053486fe703e2a614de391bc4a65
- c694dc35c2df1af133658d92f87be5d6c92876461f3221d77fba611aff8aea22
- c970b478bba99d85943d1e84bded87b67224da030e49e73ca739425c4310fd65
- ce24899fc29e0d0edd387381e7987917f9e1b72aeedb090a257df2cdd956e1c5
- d9d5cee72141834dbf50fc91a7740bbddba2ff04b40c8607adfda1c522cf0f76
- e50d52fbac295a37f0ed0b724d4ddafca8c86bcd2391b4c481d51d5e279afd20
- e670bb774d5477ba3d725d74a873ebe94fe353935168a8a25c90f826ddd647da
- e6a0f996be6c362f2a0d21e89a763142a5445cbc8ee383cac8da31775f08f320
- ec31c99085e59c09107da36af1b890e80e8b4e099029b21efe157ed7013eed0d
- fefb3c7053a1332b03a4c0523862e8e387a5065b263cf10cb0d7f33f02afc646

Nefilim

- 0bafde9b22d7147de8fdb852bcd529b1730acddc9eb71316b66c180106f777f5
- 21873b75c829aa37d30c87e1bc29bebd042f7f3594d5373749270c42ab7c042f
- 9e6be0a3bf10410a43c979902507647a4e4f4625a1470ad1ed90e460183b5995
- ab7ae6803a010e1f92189c956080c46eca38c6325c1fcc0d766dd491212cbcd6
- e3dfc0485c5ecbeeb9a71473a25a6a8cdf616b7f05d66788ed6e6ade76aaf1af

SatanCryptor

- 0afbc3c4c924b136695779459f627305eb87c3fa714c4da929114c2089a233be
- 41c059f4dfaa143cc75df07f38f50d7d6ac0c6416d3e21aac2e530683c037fdf
- cefa175129fb129cb756dbe9e0253f568838e808a616dd05345930c1cc7e2f52

EKANS

- 09133f97793186542546f439e518554a5bb17117689c83bc3978cc532ae2f138
- 2ed3e37608e65be8b6e8c59f8c93240bd0efe9a60c08c21f4889c00eb6082d74
- 595c1b38792afae6d6db5ec1e56d1911a0b480b1610032c3f28fdb04fe267dc4
- 8212514da96b371bbe6b6b0d84bf85fb526b5b099c672e400a1974fcff9ada8f
- 8a04cb48e356af5b5ea78c7eac1b91ef6a0516fe815f7709205e55eb89d1d2d6
- d4da69e424241c291c173c8b3756639c654432706e7def5025a649730868c4a1
- dc403cfef757e9bcb3eaa3cc89f8174fc8de5eef64a0e0ee5e5698991f0437f9
- dec853fe095f630fa4655dd6a3aff43bb1243eba9781238259bd2e97c470d69a
- e5262db186c97bbe533f0a674b08ecdafa3798ea7bc17c705df526419c168b60
- edef8b955468236c6323e9019abb10c324c27b4f5667bc3f85f3a097b2e5159a

Vaggen and Related

- 03420a335457e352e614dd511f8b03a7a8af600ca67970549d4f27543a151bcf
- 07f65751038f001d18f9bd0d038f2b4f72995f56911748418dd1b3f75c46e8af
- 0d5f933a38d4fb4f7f7e1bf0a6b28d0d04a4ef5482030a98a7c6ffb828b708b4
- 0f72e50077fe811d11ec1c08a5329391f93de43a0ed784c179d4faf6e194ee67
- 173c97d83bd988a8fbc9bf2e6e06781c41ed3e25b02b894c9e27ab90695dd017
- 2a8f6d82fcff3418f72b4a19ff7ec9be9b42b2571e09898897208bec6b549112
- 43c222eea7f1e367757e587b13bf17019f29bd61c07d20cbee14c4d66d43a71f
- 4558010f342bb4f6e33f380edab88ee488eb683524d92c15a3657bb463c094b1
- 471272527024f33d030f5438208aa44c61bf12bb277f9c5d5220081364d60647
- 8b29f6076201d5307fdd658b9481e345580ef5dacff71bcc86c5ae5d5ed5a26c
- 9f454b993587ecb9ef3defc2d638f794e742b39f1a4a1dcd00ef18d5b892427b
- 9f4b35afc734823f0c14ad8974bce429680ceefb3c2b5726784d83e2e95f4cc6
- 9f8c67a0fde04321be4d8a8370fb130e623e52b677efe1a1a2b0b1d400d7b5fb
- a2108d798ad421c2562b3195c036248e4a561c530eac7432ccdffaaccb84ad87
- e884225aedd3363644bc65d0866ae28f90c6971a2cb9a296718c14bf85985ee8

Smaug

- 1a8d5b23dfabb833873b67e5ac5fd0c88fcaae1319a880c6fc4a01ad4bfa6426
- 195883b48b6cba46a671aec7f995f91550f7ea96069745cb5a9a2fe5ecb07b9d
- de3d0fab26d04f5780b1f1037acfcbc51505c4e0cca3c6defba817bc36df9768
- b7ed4bc96fe43bd7a55a9bfb8d8e7f8a6b3cf968816cd4601fffbc42b79d04d7

AgeLocker

- 29c2f559a9494bce3d879aff8731a5d70a3789028055fd170c90965ce9cf0ea4
- bcc0a748732e3e8ac08edf26eb5c6350ab11301acfa63375af81ddb756f704d9

Betasup

- 5eb33afe516fe5a73f0773340f0a3d6c9d88342bb07b968ec2f759194bf430ee
- eb5853df85dc4757b030110d72a48d11e722dab62f04294a693c156fb423a9aa

Sorena, HackForLife, Vash

- 05bd4ef6b32a66ee100dae0dbf57a9d21fd6c9a57b41db3fc3bf553f2e982d3e
- 14480fa567237564c6a0eaf3215c356eaa87109c236dc58225fe64c56bee19ab
- 1557bcdf1e1c3913a166354e4d4b3878adc7503a5214179dbc8cef417ba3aee1
- 155f3c2b2fe4fe43a1795c220ceb309b5e68f22ccd3a2cc7f2e6e2df5644717b
- 1c57236d1e9eff4771ac687f5dddfdf902690e095b5877ecb2087effacec682d
- 1d6d9df4c9f7c6548d178baad3669cbb97777d4aedc6135f6ae436f8bce352d4
- 2ba2c20a826f51ed753f4f4dd78118d6f371a2fd5b4b0a2ff640c8f046d4fb55
- 35e8e113150b041416abda4a8d8952ab9dc4ce86f184847220ef0e964e0916fd
- 37015af4323378c6dacae8aebd77e5ceaf089787c4d4e3297035ec1203d1f5d4
- 3a9625c5ce576012999511a99003bec50887f47ca9176cb4de567ffd9419adac
- 3b6ea97f585054a55d21ade9f6940ee3cd0edb8d3eafbb3d2cfc4abf19f6d0a9
- 43179ce482ad41cc83090786d45dd064ef0a9f3c3ddb3a641cf5ffb63b0e9d9c
- 4cbe5b91719b3c08fce931472aeac4e999e1968434be091bb5ab770d96192a8a

- 4cdb0f8d00b8db665bf8686c7e61b583579d6b9e6f957b0aab6b2d7444402451
- 54b6d85202f1c540fb3c80f5e84cf48ef333219a93776eadee14acb3cddd42ff
- 5531b27552a5a74235a72d91d42fbf5b643806a0c1ca01c32a8e74d87c6af732
- 5c29aaddfd62548bd1bcc98dda867a9eab45742989b458dbaa670163fff279c0
- 6016b23e2b8611c6f8a44111e126d9419f1c88d5621ae479e92e90e9b70e6ce5
- 6845211002813319a52b6d80f970da3a1f21d1035fdd6fe6f05dd067a131253e
- 68d04f4b4dcfdd679fc625771dcb93311390a1d9cabf9a492d3058ae9d170416
- 6a889b99e2ddd3036d7bde02b03ea6ce3474dfb55fb8bc9ed0aa3bb342fa437a
- 73f0d428e02305b0cb80546d9c1357271be3f8e961f795de2ecc4777087a99a2
- 7937dd9a573ba7303010781d34f945936bde1eeccc3fccc8dff2ba22852311fa
- 80524ef85a8b932ff3d782663ba401b04e0d4baf17b2e4554464c7f436e48c6c
- 86d6584a8176d98afe841eb666de9cee4156c93b1ab3d5bd797ba3b017730cc0
- 891fb058c9d7a5ec0703cbfb62164c967d4d38f933beef99c77b2831ad2210ff
- 8b7cfc09f9edacb803fd9b3d15281aae96df0ff76f13383fc86a4624adac4464
- a0e6b3e35549e46d67717675dada1f3d2cc8631f369ddfacd4db61b18697575a
- a7c67294349feb719f3752d0a78c8e6e4605e55bef21ee8f88b3fff0521a886c
- a911c7f64d373ebd07e60c2e5a0242dac26492dd2220d8cb82e80e3223fd6f53
- be69a0f4cb9b3de25ed4277f2a85421f58f88981776ccb2876e464f4bec9bb9f
- c8df39d2803d496902e4cec802079de739b20ad8db68bd0a03eeb0261bb6783f
- cbeceb582607be7b546c6e4736cb952060cad1d63973dbe1746e1219225d8427
- f391c5fc696e839a4fe2d87701dad10fa1c22a29ddbea95306c93d0b6bc85be5
- f4873b38e0ab4e2abf7c76a00c52b395802d5d746ac725fa58d924a75b997a56

GoGoogle

- 03f1c514eaae77086d7ea479b1bf86c21cca7af44a586f474fddfb0f11b31ea2
- 4426f0b7ec5221865ec49cc973763503fba18a95920d69d27fb0a3b40a7050c8
- 4d1ec6a631579960b9815a843f4e14ef1536605d07527f373cbd2d70950a6425
- 58855d1b8f7f9110df5a3538de4e9cce3b55d5e797d710501a8c78114cf7d12e
- 6bbaf93ef6a9b0d02ea4764555734fa33c4bb28a377ec45a5533ec8933868dd1
- a898dde84468cd5c24e88922be8e6e009d13988df19fd89c31f993842beced85
- bda54b51aac10677af848e3fdf2fd119da89568fceb246b068e5ad20e83c4c69
- d5efc6118f723ecfc2322cfbae494fa9f8f0d35a2d3916121d924a3182046bb5

IRCFlu

- 0166399ab8b3870dde740cb9daa3b9f98873ce5d0af87f6629f2f618bdbee7d9
- 3a7111cc374bc9f90aa8ca8a64cbe3bd49a9b8823f5301bfc80196ff228ed194
- a83f97b5ae7e1b2ab0394d797981d381c55667c8eefeb0449831c62f5de28722
- c5e83acae1e1ba34e7c1e140987ad881fdb2b4d543419d7ee3393aeb584cc0c0
- f30cbeab6473bd919ce96004963b6f94c95c16f83600bf262525bb2bd0dd6d3c

Ddg

- 0e0897781e3ca4773de29d36101ea7c9360b67f461617afa50d19b76254d6bd1
- 2f52e5a0f9da244a2e44aa56dc70b86535f1ee1eb256839fcec04565e563226f
- 306e23b30a282b5d58c82fd267ba3bc8c292c2ee48384f976c3513437182f9be
- 59eff031cae3bfb263f02bbc5b1fd2b7a47055d98eec6ed96ebda1d4ed52eabc
- 5d427ec4fdcaf9af3fc28da84b654922416179a88404b98ca995b51392deea20
- 63928a9c248e5f6d682fc4ae78a96eea02ac17bdd05973221760687f0315ad65
- 7926771af5b9e6e5d67093ba36212c1fc09fc988c2dc4ad9918738a207095a64
- 8a1058544226fcb9e7371f029d126e30053cc2b1474864862f87fd1331f5e7c2
- 988fb3c2e422a0d31b95e007c0d179b629e389b7e420b15d7cc87600bf982f66
- a9a978923521645195b17d97117963040733fffb93eeaf718e793800ca152710
- b9df41c142b326c475607b3044c4e46835254fdae899f3d28b9829e4ebc41212
- bca9449c2b6092c256013a66a075fe2ca2420d9036be7ca603587a67b7c2cbf0
- c6fa6c7491690efd5a7ec7c57017a758fc99deae453d4bc8f5cd5da32d322442
- d4aa907e5e7cc2171dc9eeb610743046f744ea3d75f495ccfd9c24eb75132bf2
- fbd052433f0dc3a7d6bec758b06ef23ebb7cbec1667782ee93fcc7dcbfce8370

StealtWorker

- 00c64303d8a4223cfc3cd78e8b4c908f434702247cff11626a9d18a2afb564da
- 04013f6b12df021a5b60adf99748a25f12748049961cf85901b796fc7ccf60ae
- 06ca26d431a929cfd719458da758b7ac404b5865ae720e9ed2b4b3a9f0187280
- 06f70c14fb5ef90f7d3b755ba3f0375265dd5c5514b1cd9619f97477643b19bd
- 0d8f0596d278ee601eff8afb6dd8b0c722a483bd70617e2d9ea0df293734463e
- 0e5561f3f7a6bc6c49281237076558b7994c1cf045f502d9ff9e453dfd6dc39f
- 0fb38404d39e0b34f5cf366a39e73521d6e91b34f89357fc8901a78475ae9d31
- 15b48cc9e7b9d31bcd7d76f9c01a3cfee6a70532ef2e98e2e67e4de53195739a
- 19e70ec36daf08be1cafca0eb592eaf5a61362c678d77a3ed9820d760d5b277f
- 1d09f2f7ef5f062a70d91b86ceffcdaac71139c8a8bd2a15ab06e8844316f54f
- 1d3ce8fe024540de0c52dbb395f519df698e6f9916efb34f5e410083780cf359
- 1e1f4e03af6e2863f543b8b6e6c2e266de5762a2b4b7600e1efbbe3acfd85668
- 255a59f16670cb5d959da0435e053c32036999638d22875dcbdaacea7365b0fc
- 2b0b679ab6e903225cc7420ea67d1e94dcee852df0ecae891f3f5c6c5c8e3a4b
- 34cfe96b16cb252fb113e9ca183485bef70fba96ca8c3382d6efdd4f4907a6df
- 3638168e4e5e18c74075ba9b3952761156dc077ab3af4b88878cf528bf257133
- 36a59d0abc832b7fc921bf902fd73635609497c0e2b65266540d8266b77469f1
- 3f24fd22eb90c020241780b2a193d1a868658bf493230d84fd01c6b8b3c5be4b
- 41808ec2e6eb0771d7786604be6e325806f99ac67295c2f96d19c131c145981b
- 454e48eb0dae78f691f64de35128a90122ade56837d0e460524d438a9b23041b
- 4584ca1c7a9117bf6ed76b2bc95f37b1f51f67ff94a5092cafe5368415369b05
- 4a3384e65f82229bde7bc08c7a999340b74a306b9b79b5cc17793ba803eeeb46
- 4d241a9932f37571e37d47f641872b07c3d631a29300e5978fa5c490d4e7aba7
- 4d364b214c85170c6595605c5c09c0faf90328f65fadaa8b1ed25bab14821f0d
- 582a3e35366d98b6c316c6eb86ad6ffa62405be41d8adcc8435bb0df4783d4af
- 5a7c2b0e4ed773d7052a3c2594576f13de25abe51aef210ee9e1f7b494f32dad
- 5c1171253f75b81c5536206cfeb04da30d691fcb4627dabc474ef7326b613ac1
- 67d2bfef654117e1bd11383060e890f3804dc6656b0633e54c6b6fe3a59ddd32
- 6b463e18e01f47269d6cd7509aa6d2788cc2e7473bc9b8169bdb003f2f61c913
- 6e107005a29b2adfe5745079534d14aa1443b3beaf57124b25786e8dce3312a7
- 70d6c4ef45c0b14a42d6b9a8de68bb795d76fef04a91ba434da66a4402086de0
- 7209316ee8ffb2c0e4a5f9a2f4cf1b455dc1c480a1ff11feb2e85cc8c48b1786
- 7d1577de785d89762ebe1d698fd0b43b8f9e6ee121685d37fda05f7cf3033555
- 84dfa212005b4035401a1fb2f4595550302aaf237f8306700dc597dde046e78b
- 91ddaec1f0c644990ce51b6d013058fb28ad3b36640b6a04bd8f6ff8934b701c
- 973bc1fdffb38b8342096719957a15c8075692fc856e7dfa1d123b46a5f07279
- 9c1b9dbfb9715276003a36d3c0f9b9f917044ec663276944f188786783075504
- a329d34d9023c1adcd17f5432a48a3af886da6f63a335a10e95474c6813f4258
- a4fe359f2605850b92e083b25463af91a489d65f573d9802523d0622521d3c05
- a66c80161a102c5d87ef207bbac7a5c687b114749dbabed5e771f12bc028580e
- a9b09277842527ec321921d734717e6c6d3affd903aa9fc9ecb6b74e08f25ce6
- ae6dd2dcb0dc781d599acd25d1273f2ffe8420479ceebf58f5b8af86db9bc70a
- aeb25d0fbe6ffc5aad7868ff1456d4ab890199884bf0a846ff207f9a91146b06
- bb5ea85fb28fcef83bfc5538a485c09743c1faac54946671e691b78b121a562c
- c16e04ca8889c6a9e8a015d00ab2f55babdb4cb5a5d9498ea3bf85606b86257f
- c65a7d788a5810c717ed2a5f60b57d585502179f1e8e05b5f731b85d9767a014
- d284d58f5f2afc5931577247cc7944e683cec7813d61ef90ad46c100822d962e
- d28c3ffc5abe65aee11354523ce34cb57f007559384eb71d35ef4bdf2b4ec791
- d70511f773f0b825b3e3217276fc4c05b55d1f5dc10ac5b5078fc26f82ffed50
- da78e99dce3fc85715c6cf10cb4c926f0f95e2b6787f6bc9d3837ad80362fe16
- e2361db75d61bc42fc656394ae96c2a8f442e29f27ecfdcc05103bb1e693efaf
- e41c6f2e199d734f799e08a9d3ed35650abc28c74166dcaa25cf3ddab91b86de

- e6e3a37f053bd43bb78e7c8d9c5ad26c71aab6e91f7df1005ee8ffaddaade14a
- e6fbaffdd3a7386cd7f9b200d28b904195e1c70a5e3797f41639aa0fab0fedfa
- f495b30f961f5864c1c8db0422eead3e91c5548518721b47133565fef2146344
- f66201a415fa0a6d30dde90d7678331e5a9578b177bfd1ed5eb6921f859505c5
- f860822a794f347d9898438e211093e8691fb244d1a9c361e507df2412a85c3f
- f860822a794f347d9898438e211093e8691fb244d1a9c361e507df2412a85c3f
- fa7bdf7418e4555f081c5d94ac8514b1e9cc53bf92e74ddf07e428bccafc4e30

r2r2

- 5bd76143b6fb07bd0e72e6c7600d1b41433eca49a24363d7fd9299136aaf1f22

Orthrus

- 233b7d48a4f00cdf93e6dad2e76f441519d36fd0f95158d099881b882b44a81f
- 33dd0a46f93cd24d4b35b3bc8ac0469fae28de2b1d1c49b21dd8d6fa88004383
- 40c40bd3aa61d09b704748ecbb9ecbd9f34b0e4afb175036b12906900ce2cbde
- 40f634077d37fe1b6bcf149e3e56c8ba668b3e7781515966ea676d035759eb21
- 5dffdd711bfcb5e8b63efffa2839c5a0ea405820f72e424515f52c9e85297c64
- 79602bc786edda7017c5f576814b683fba41e4cb4cf3f837e963c6d0d42c50ee
- 8129d9f08314ccf6ff7a978a272ff43e1515cd0a28f1f1aac7008a36dde3618b
- 87157021b468dea4cf873dd58a7215b5407a15ea3345e172fe108e85a2110125
- a0e2768ff05867446de916d32d3b9e77fd27ac0a80391d9d31bafc752797e4c1
- a98e81b9fd8a8a78d7977ddb9c37f36bee0a51695898d0ae8065a3ea3af3057a
- b0cece02ba3d25d0161ca6daf75c3491f457197e1dc2faf73919a32e37bc8d56
- b7818c458f0e6e9a5d2080626c7cede1eac573fa2c4dae65d91607eeaa23bdf0
- bceee7d9ace363ef2bfb1494a9784a6377fe14c4c5fefa0c180fcec33a5d1716
- e476165c01bbb981ede35e0d0b17361aa453f835c59f76bf18a61cb1cdbea8f7
- ea55a206f7047f54a9e97cc3234848dfd3e49d0b5f9569b08545f1ad0e733286
- f373d4a035aa64f9c4e30a483ee1bcc997cfba1333036331af9e425df53cba15
- ff98edb0f8be57fff101c1fd2068452f5148ed6b68aa814d48d26cfdb50775be

LiquorBot

- 23ecc306ee6f73f2774b05be96a57959d075e3c9f809693aef48c50c8ed8fd63
- 41dc20fd00aef444af5b14c42425b628d0d5a9e5281ce68373ecb20e956d56b0
- 774b54e55462e232ae15b3ea1ca6f62361ed94532c6b5b74b3bf3e961fe4b84e
- 87fa9f39e32b609322908f56d537f22e2455a56f516a5d378c978912afc5f5c8
- c685e849fe0ecd90d19ac79d4c7c8c571958f16a91a7021e479f3cf5fbe764d4

Kaiji

- 03ae6417e3831bef53c33728e32f844d1d73198d8b4de7005c77fc09050cac6c
- 0900406fca4156dd9044007436e3e2bc0c69bdf46df17386a5de7b2aa61461b6
- 0b8b6bc7f036fa5116878b3ce9d128370b24ee108523c27919fe8731a800858c
- 0f150e80b4435d723f28252b91a27dcc811f61cfe9c841c7493b1db29da1d89a
- 134c60c410f055449d05548d42053241e2d90aeaff86a49e6cd6043c243be14c
- 2045a0d9f69d29f8f12f9283651b858f12c172c3256eca6da559d4ad152d8216
- 2c2571b749ba82959d21c67441fa8ebe882dc3a414e950e0c714858651f4f1b6
- 31fc57f59b53e0d747c53e81faed9d8037ad07e9807863dd45dc5a7089d4c5d6
- 3bde736dd78efc553a0e7cd6e2f14436fe012a12b8c8cba5e0d92d3f79e948a1
- 41b8b6ba7bdb710fae38d6c4d00d3f02b92daae7bc225de0cc2e0485174a6eee
- 42db0f94dbe2bf8c49589918a586ebb85521e0d03fd70b2f29d9e798ce25e770
- 4e8d4338cd3b20cb027a8daf108c654c10843e549c3f3da6646ac2bb8ffbe24d
- 509bede7ad9b7ec9258cc830d687e6ea66b5916d8572e6aadc48c108009fc1fb
- 541fd2005b7cc6ebb54a588b2d8cf6cc59ccc37eec9d69ada4d1c6e4cddadde1
- 543b14d808f57295612767ea9add7d58eacea69435b1dfab21e15de458de0f6c
- 5e1c475215b4ec81db24c7b25ba9cfd3aec0e02226fc699422a9f44be19fe7dc
- 640b73013fe9ced59d01c413727fc46907744ca3c7a542b3ece338334644ce36
- 649acf403fc906c78566a0ae5ad9be59f09286c3226398c07ab6e6c94ae746c7
- 68128a0006d9e0f89517f4cac220f80506026f540b6bffb98d4803dff72e7dda
- 6c32c160fa92bfae2a979b179f8fad111ff3f01a441ab285d45e30d2ac8687b5

- 73eeb8bbe6cc74b1660cf58702064f50a448bc92e12d57f2ad70b172a835bb50
- 81d6756a5b7808f45104b9499d425a2ec6f2fee538aede532eca98572fa6aa33
- 9198853b8713560503a4b76d9b854722183a94f6e9b2a46c06cd2865ced329f7
- 933750ab6f805ac51d310d7053c00eedb0c9ce38384bcee139b0558b3e1a7217
- a4514ddaf55e441340782910ef5e2e8bc854b140367ebb17f095eb7b190736e3
- ab0de56489cd033083e7d1ff6a4731f1d1c80b28b213cb0bf0a2a40ef45dba73
- be88633c5cdecb4f5f40f82303af19ee2ba95f7f6e0f7e7d5b11b1bf7d43e582
- c2912feb4dbac64fe8476efc86585d8e0c9f9b5df6ff13a3dd3dc8eeb7a1341c
- c4455457af7be7ff0c160a58896dc14b35788d1b986ff7476ff539b768fd35f9
- cb2470cf4f13e60f8a7cd59c6fa87807d6ecb70beb87db6b9f8a5b9588b52c9f
- cb64370c9e6540017fbb59bd078eaad348ba32f1b89f30d97da2b3bb421ccccf
- d68e25e05827d24bfebeedbaae76a8d948d9119c5a3b5436d8e64a8571d8bb1a
- e31eb8880bb084b4c642eba127e64ce99435ea8299a98c183a63a2e6a139d926
- e8fb3c211d6515251642f79be989e424392585fabe1f3bae37dc2e0b4434a3fc
- e9bd07bdf3bc80e3f9c9fd8a9f03a93f3b488f4438cd0dabcf6bece1dde758dd
- edd8ae236bdae31118d46604abbcde05129cf8ca08a1d35f18a6dbaa89285160
- f1f26eb4f7332005eedcd1c13d3a11707b4ef9ddad971f8d1eeb2cddac6bb327
- f251a0b754548c702511eb7077b76e220507143c63a6c8207400f95644fc5080

Blackrota

- a6fdc85b5406097edf795f6a9dcea8ecb85095cfc17cc970a5d80764b052c363

Kinsing

- 5f1e0e3cc38f7888b89a9adddb745a341c5f65165dadc311ca389789cc9c6889
- 9352b5831a078d46d6b2b8754e74303d2f6967a99222d33cac05d70b6fcffa22

IPStorm

- 087f2ec8bbcee4091241e5ad30d449a1aecd0b9879338d072638c7d0ed6b30da
- 10403f0d7dc7e1d401ec0f303ced137b672743298f062d41940dacc6b3c0fd61
- 146a72d735ecf1e48bce1ce5cae69e3fc6912c434414d202968a385149ce5025
- 1996927b41960a2af8e49cf745ed6668bc5b8d7855c2bb116f98104163e29000
- 2f6f44e3e2baf701ae1ee3826986f89df4e5314c8ba50615fb6580f1ef54c830
- 4635fcde4ecba736b84a2ce842aa43e93e2ab56b6d82cd3700de7dfe6cdaf499
- 4f0add8eadb24a134b5cab6052920f576eec1bb39232c9548286a66883dcab82
- 5098a4ae39d3ff51d4a35a6e80c95acb7de79335d22d28330d078e5a5e96f003
- 5103133574615fb49f6a94607540644689be017740d17005bc08b26be9485aa7
- 522a5015d4d11833ead6d88d4405c0f4119ff29b1f64b226c464e958f03e1434
- 52f215521ba59cb6a51314bd1527f1c540ffc04df924ad971ca2160405879778
- 5d39e8aaf75042aadd5683431f9fca75013a06639fcb5db010aff1ef5ab8b518
- 64abc2cf5866e932b0731a6deb487aa3d9756724250de26bac2fb930cd478dc0
- 658638c6bef52e03e6aea4b6c1b2b3b8d81ad40144b56b2122d96e6957c33117
- 73a63ed199df345cb6125a1fc318eea512ce89763cb841d2cb174f781b236eb4
- 79ec318a968679f94d2ab0ba15daaeeb71406d2f744eb0cd1b314c4bb403114d
- aa7639b11f7c852005110e5ac34c9a2c94c562bcc95dbf6f60a1a7192cf8ea77
- aff33374ec4f0368e513867b48c2f2a04df095ec4f19b0c3dbfb0a1c3e118e0b
- c374486a5ed513505ee36f15298f6bf9d2de602f034981fdf8ba6429f156471c
- c378c6758c1a062960f679127d8dac1c86cbd548ae6fc6a0f5b778ed9adb665d
- d233c37f2d49badbf53d054bce7fb8e787c9973067e8dcd79835d7816aacfa43
- db9c95bdc4247ff6cdaf8a8e47b4add21a730461d8f6e2693136aecd346b3fb5
- dfeecdd23f28f80e42e58c87c9a4858648964b3100dfb899c61b54aed7856cf7
- f540e6ddbf054da5b21fcc80ae64a8d7caba50d3619c976b870cfec6dffdd6f1
- f9790d9c6f89a5315d84b3c357a6d43bfb1264d99b3ca0b133e627eaa786640a
- fbd5e48ee691df949e0dd3687755c80cc5b9d1a1a89e7dc486694370697de893

FritzFrog

- 103B8404DC64C9A44511675981A09FD01395EE837452D114F1350C295357C046
- 30C150419000D27DAFCD5D00702411B2B23B0F5D7E4D0CC729A7D63B2E460A01
- 9384B9E39334479194AACB53CB25ACE289B6AFE2E41BDC8619B2D2CAE966B948
- 39AB194DC7A7BA65615A30D99ED8845EE00AD19F2AC1236FBD71A671F7FA4C5A

- D1E82D4A37959A9E6B661E31B8C8C6D2813C93AC92508A2771B2491B04EA2485
- 0AB8836EFCAA62C7DAAC314E0B7AB1679319B2901578FD9E95EC3476B4C1A732
- 5FB29FB0136978B9CCF60750AF09CEC74A257A0CA9C47159CA74DBBA21FBCC59
- D9E8D9187FDD5A6682CC3B55076FE48BC8743487EB4731F669A7D591D3015CE5
- 041BC20CA8AC3161098CBC976E67E3C0F1B672AD36ECBE22FD21CBD53BCAA742
- 7745B070943E910E8807E3521AC7B7A01401D131BF6C18A63433F8177ED539A6
- 8C094313B1D4236AE3F630D93E8037B0A9D38DF716D5D7AED5B871DEA0DC1445
- 90B61CC77BB2D726219FD00AE2D0ECDF6F0FE7078529E87B7EC8E603008232D5
- 985FFEE662969825146D1B465D068EA4F5F01990D13827511415FD497CF9DB86
- 23E390E6531623C1E9E09B1EAF807D501D1A01E45184B7D3FFC4EEED955B0C6D
- 6FE6808B9CFE654F526108EC61CB5211BB6601D28E192CADF06102073B54F69C
- 453468B86856665F2CC0E0E71668C0B6AAC8B14326C623995BA5963F22257619
- 7F18E5B5B7645A80A0D44ADF3FECDAFCBF937BFE30A4CFB965A1421E034996DD
- 001EB377F0452060012124CB214F658754C7488CCB82E23EC56B2F45A636C859
- 2378E76ABA1AD6E0C937FB39989217BF0DE616FDAD4726C0F4233BF5414CDE86
- 3205603282A636979A55AA1E1BE518CD3ADCBBE491745D996CEB4B5A4DECE0C5