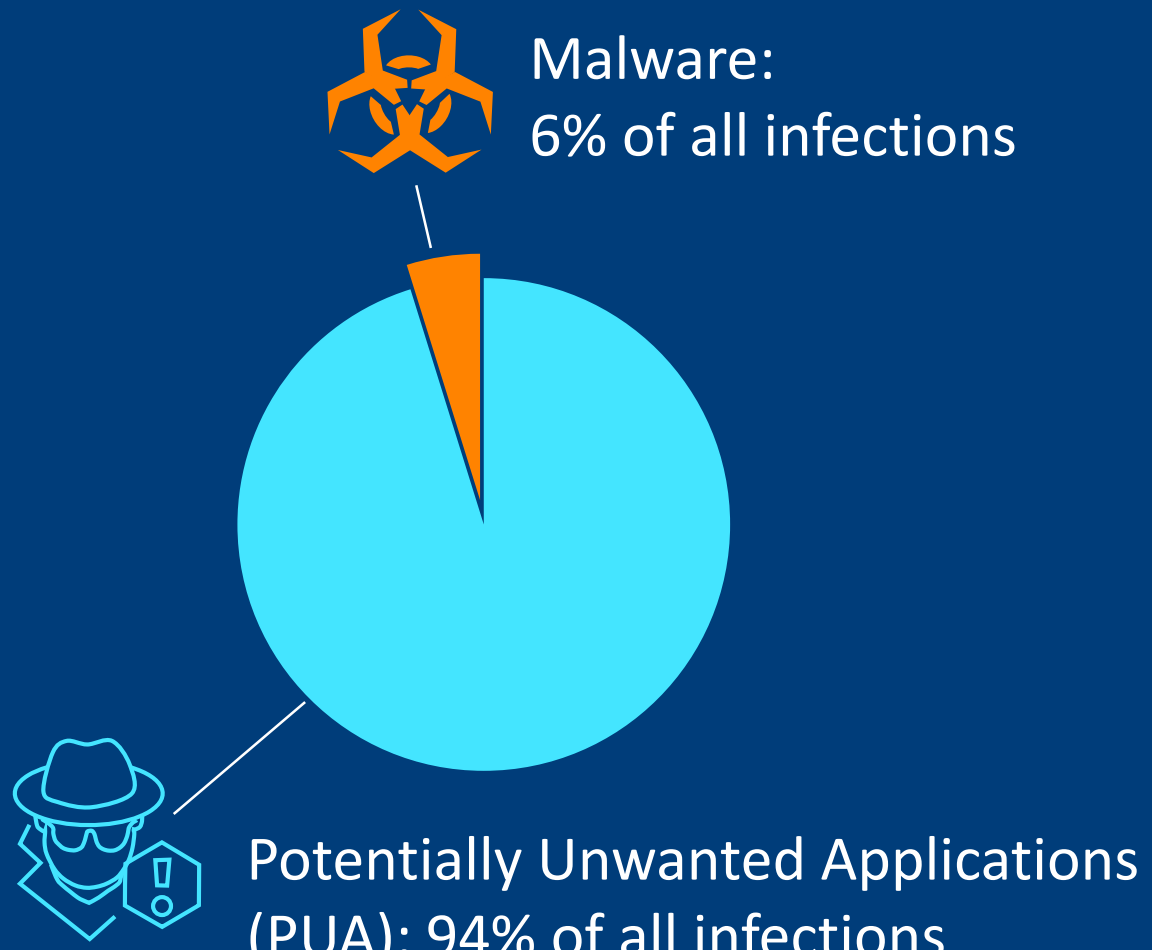
Never Before Had Stierlitz Been So Close To Failure

Sergei Shevchenko
Threat Research Manager

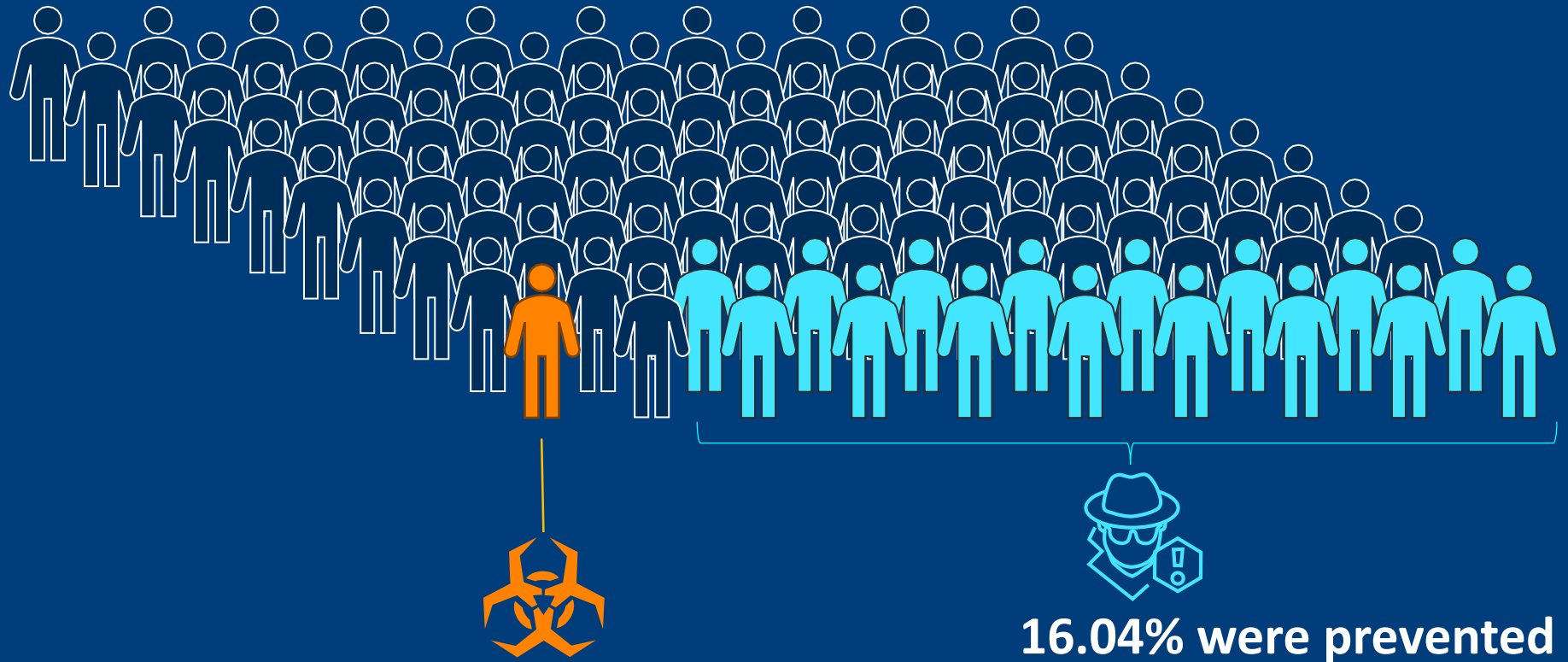
SOPHOS DISCOVER 2019
EVOLVE

macOS Threat Reports



macOS Potential Threat Exposure Rate

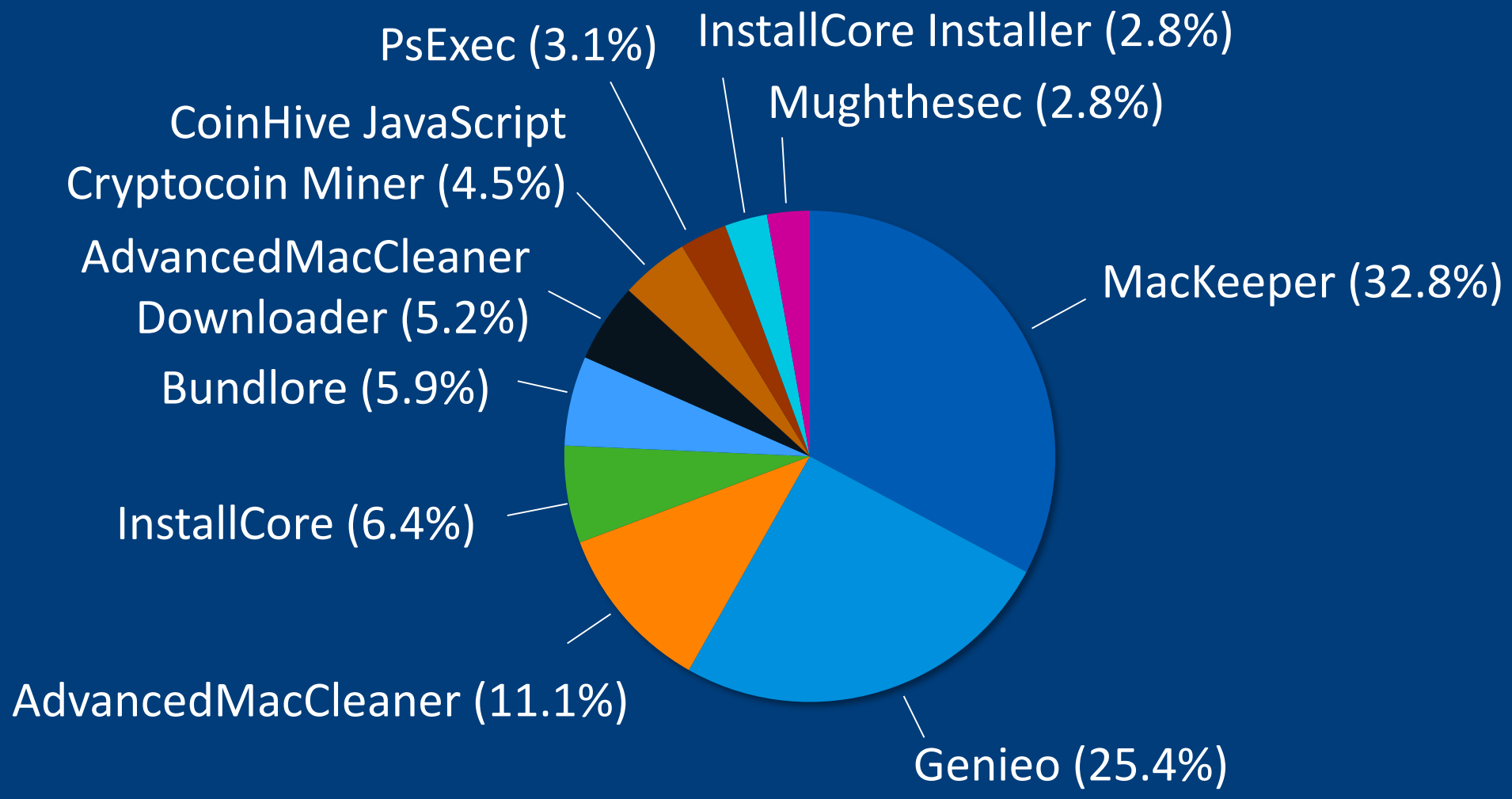
Percentage of users across our macOS Customer Base that were attacked with malware or PUA. 100% of attacks were detected and blocked.



1.06% were prevented from being infected with macOS malware

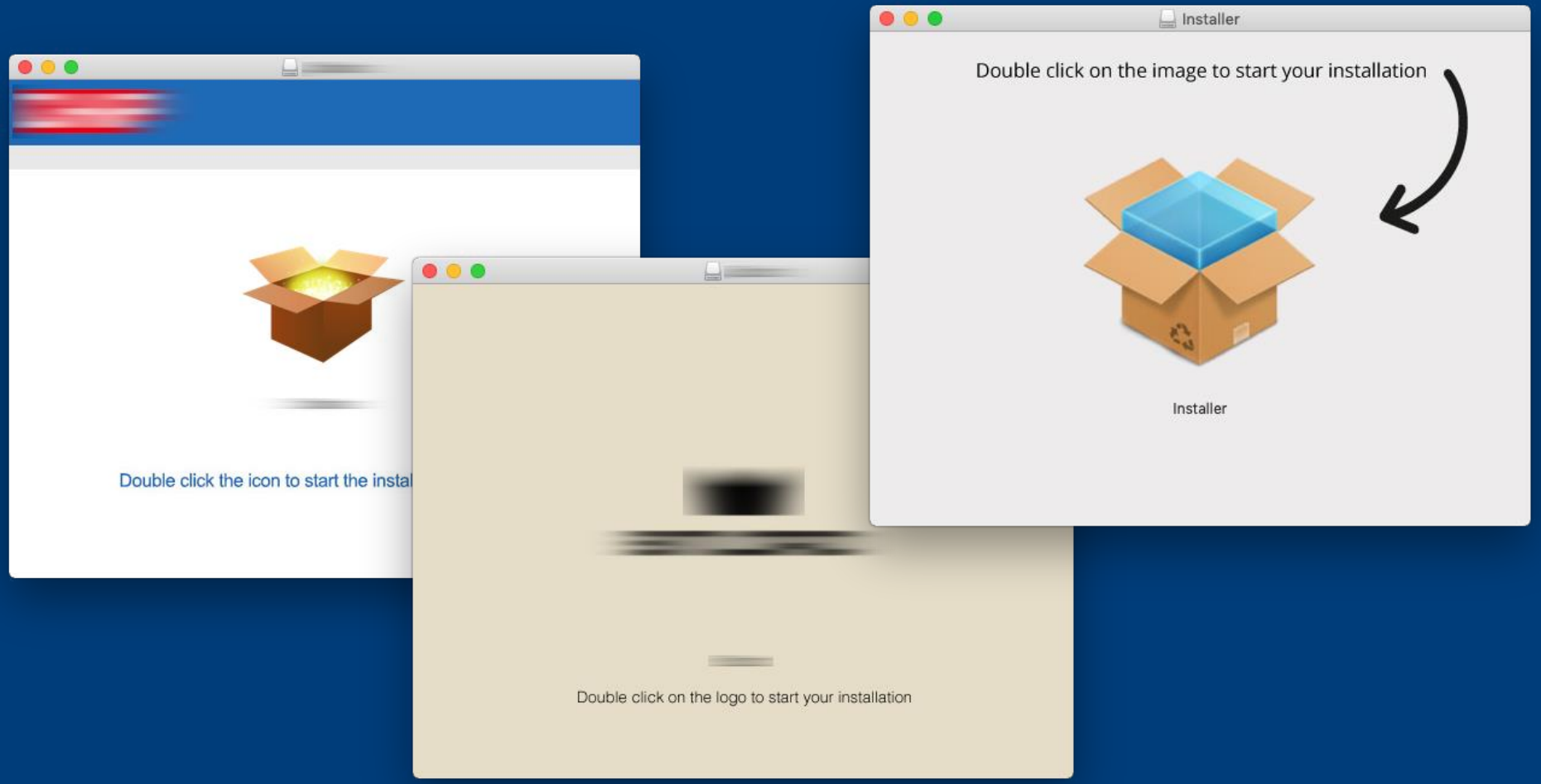
16.04% were prevented from being infected with macOS PUA

macOS Top PUA Threats

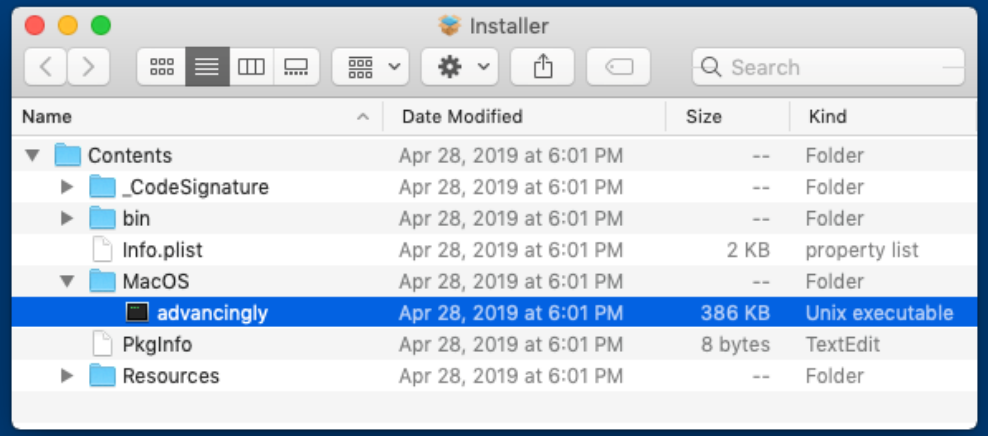


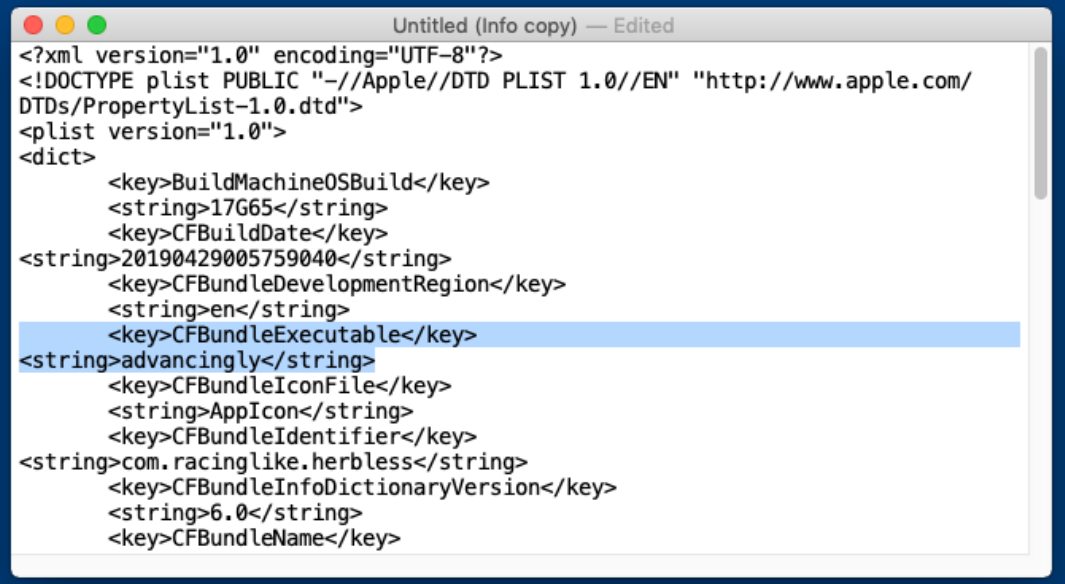
Installer is bundled with various forms of PUA

For the developers who want to monetize their work

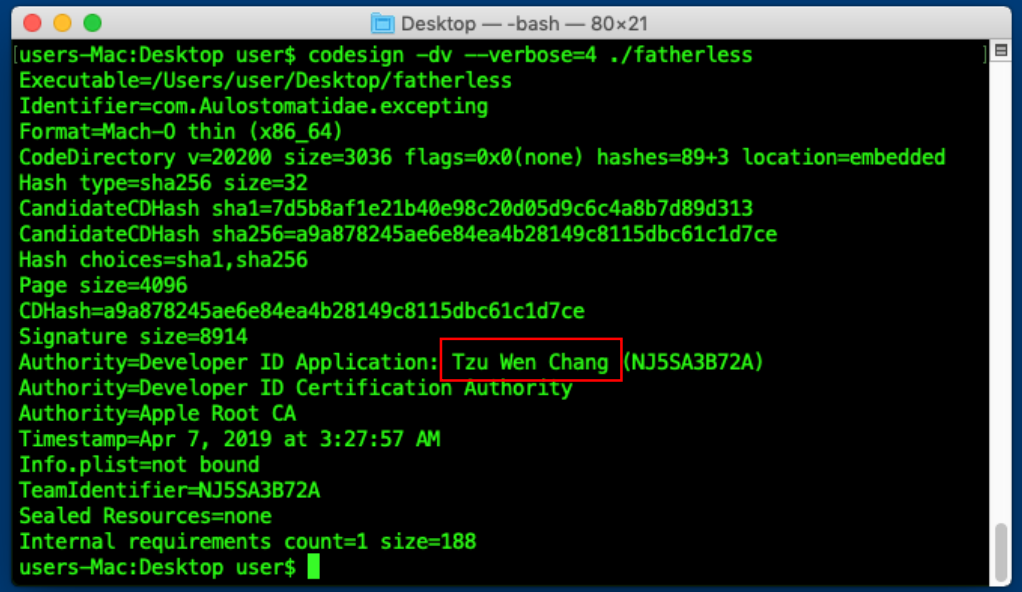


Main Executable: random name / signer





Various / random signers

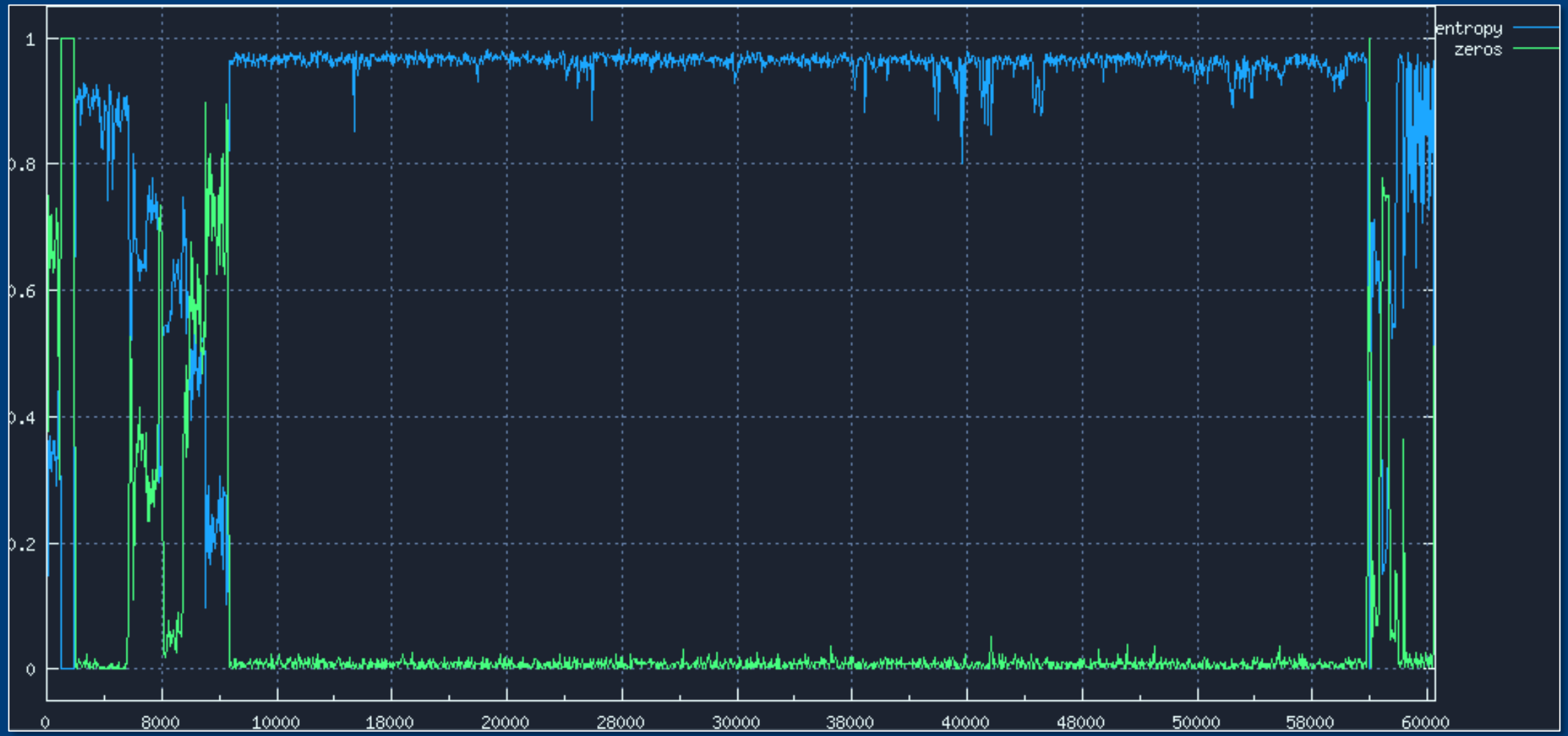


File name examples:

- fatherless
- senectitude
- sphenobasilic
- tryhouse
- entailment
- coconsecrator

…

Main Executable: Entropy



Disassembling Main Executable

Mach-O binary, relies on Objective-C runtime libobjc.dylib.
EP starts with ‘garbage’, no valid code to execute:

```
__text:0000000100001150 04      start   db      4
__text:0000000100001151 4A              db      4Ah ; J
__text:0000000100001152 3E              db      3Eh ; >
```

How is it executed without crashing?

Non-lazy ('eager') and lazy ('on-demand') implementation of Objective-C classes:

- Non-lazy classes are realised when the program starts up. These classes will always implement +load method
- Lazy classes (classes without +load method) do not have to be realised immediately, but only when they receive a message for the first time

Objective-C Runtime realizes non-lazy classes

objc-runtime-new.mm

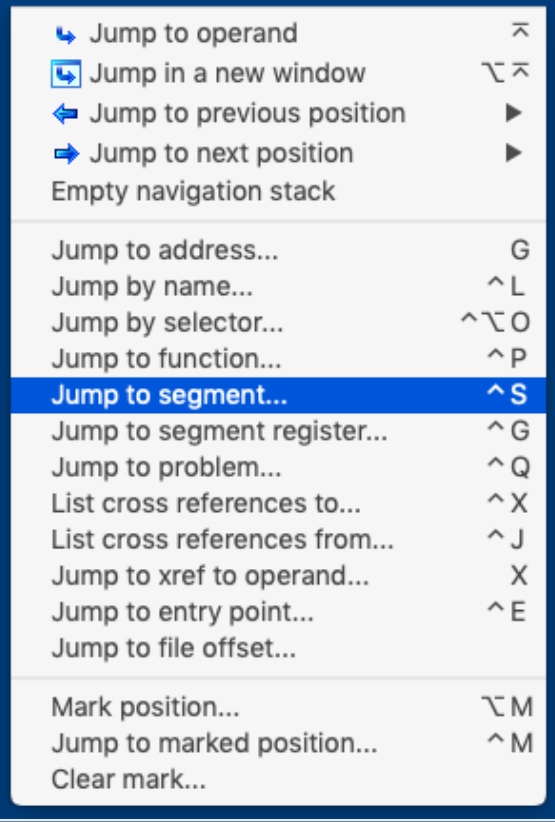
```
// Realize non-lazy classes (for +load methods and static instances)
for (EACH_HEADER) {
    classref_t *classlist = _getObjc2NonlazyClassList(hi, &count);
    for (i = 0; i < count; i++) {
        realizeClass(remapClass(classlist[i]));
    }
}
```

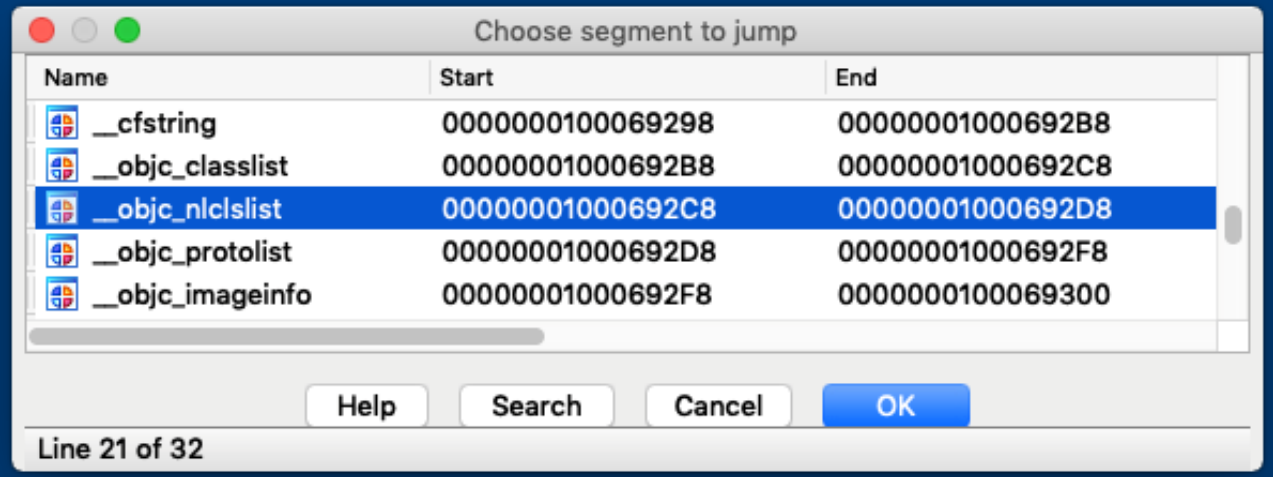
objc-file.mm

_getObjc2NonlazyClassList() collects non-lazy classes from the __objc_nlclslist data section

```
//        function name          | content type | section name
GETSECT(_getObjc2NonlazyClassList, classref_t, "__objc_nlclslist");
```

Jumping into __objc_nlclslist segment

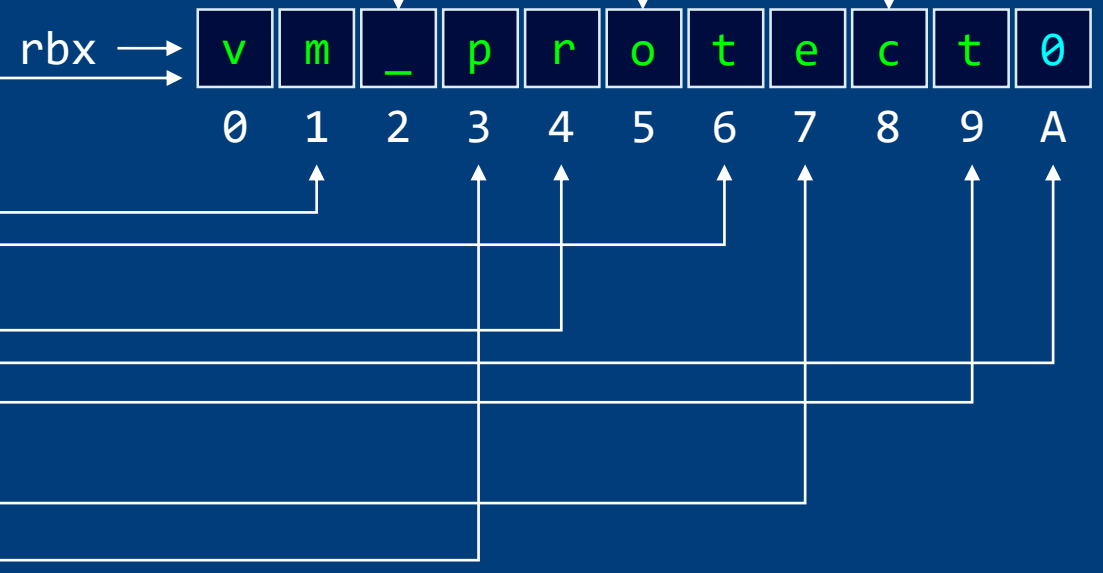





```
__objc_nlclslist:0001000692C8   __objc_nlclslist segment para public 'DATA' use64
__objc_nlclslist:0001000692C8         dq offset _OBJC_CLASS_$_ListedUpaithric
__objc_nlclslist:0001000692D0         dq offset _OBJC_CLASS_$___ARCLite__
__objc_nlclslist:0001000692D0   __objc_nlclslist ends
```

+[ListedUpaithric load]

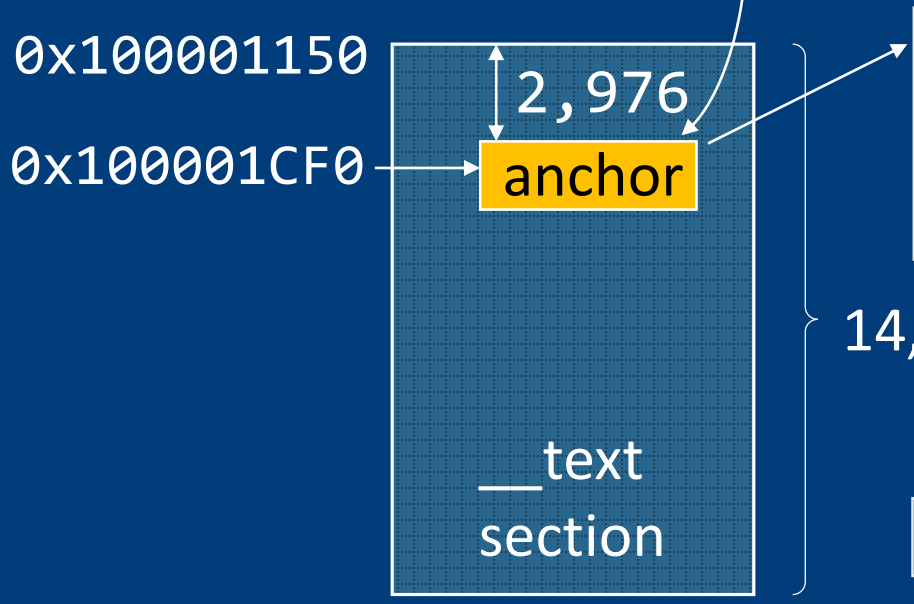
```
mov     al, 'c'
mov     [rbx+8], al
mov     byte ptr [rbx+2], '_'
mov     byte ptr [rbx+5], 'o'
mov     byte ptr [rbx+0Ah], 0
mov     byte ptr [rbx+4], 'r'
mov     r13b, 'm'
mov     [rbx+1], r13b
mov     al, 't'
mov     [rbx+6], al
mov     byte ptr [rbx], 'v'
mov     al, [rbx+6]
mov     [rbx+9], al
mov     al, 'e'
mov     [rbx+7], al
mov     byte ptr [rbx+3], 'p'
mov     rdi, 0FFFFFFFFFFFFFFFFh ; handle
mov     rsi, rbx                ; symbol
call    _dlsym                  ; vm_protect()
```




vm_protect

Decrypting __text code section

```
vm_protect(mach_task_self(),          // own task
           (char *)&anchor - 2976,    // 0x100001150 -> start of the __text section
           14322,                     // size of the entire __text section
           0,                         // maximum protection = FALSE
           VM_PROT_ALL)               // assign read, write, and execute access rights
```



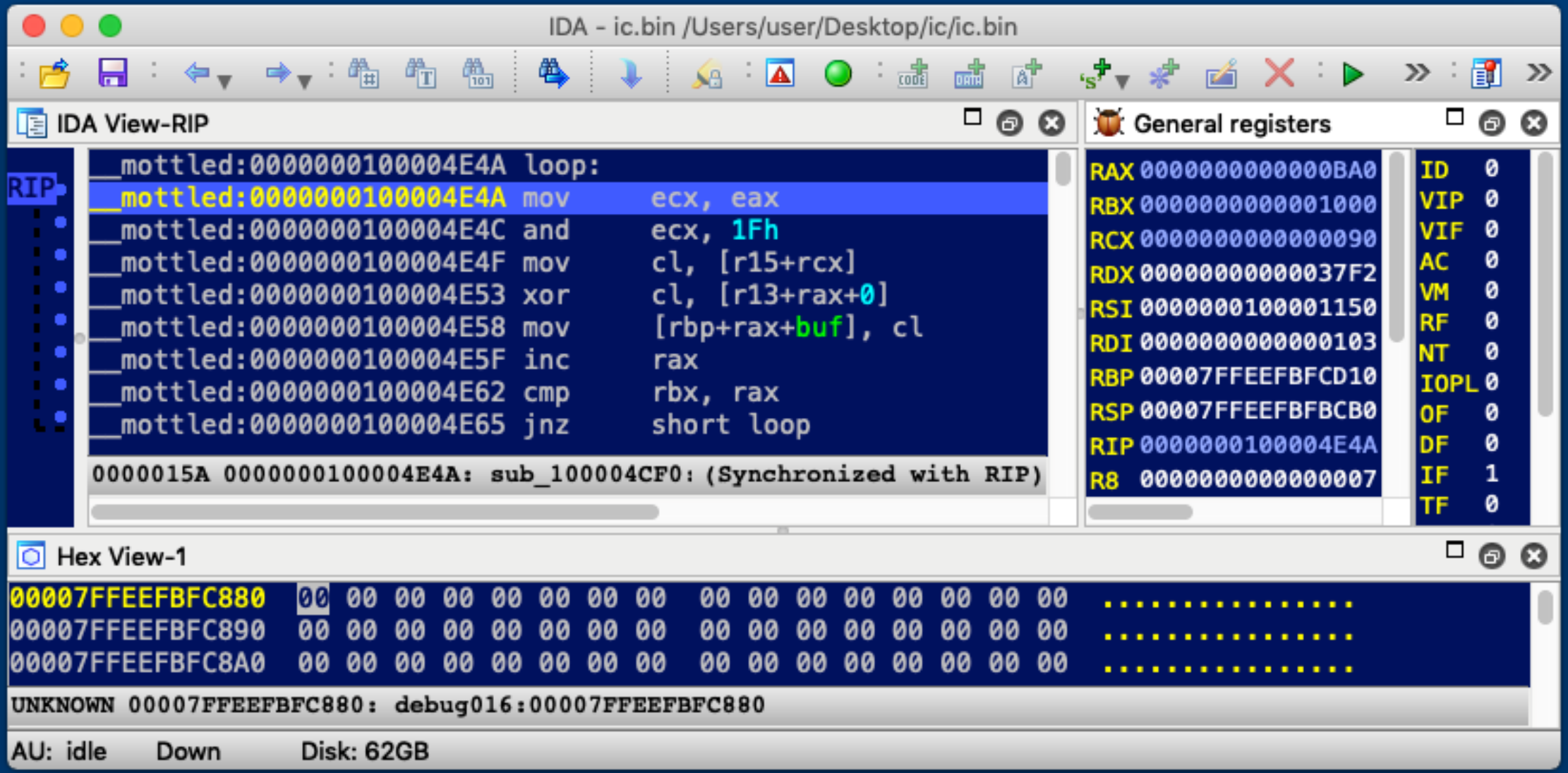

```
__text:000100001CF0    23    anchor    db  23h ; #
__text:000100001CF1    2B              db  2Bh ; #
__text:000100001CF2    0E              db  0Eh
__text:000100001CF3    0E              db  0Eh
```

14,322 bytes

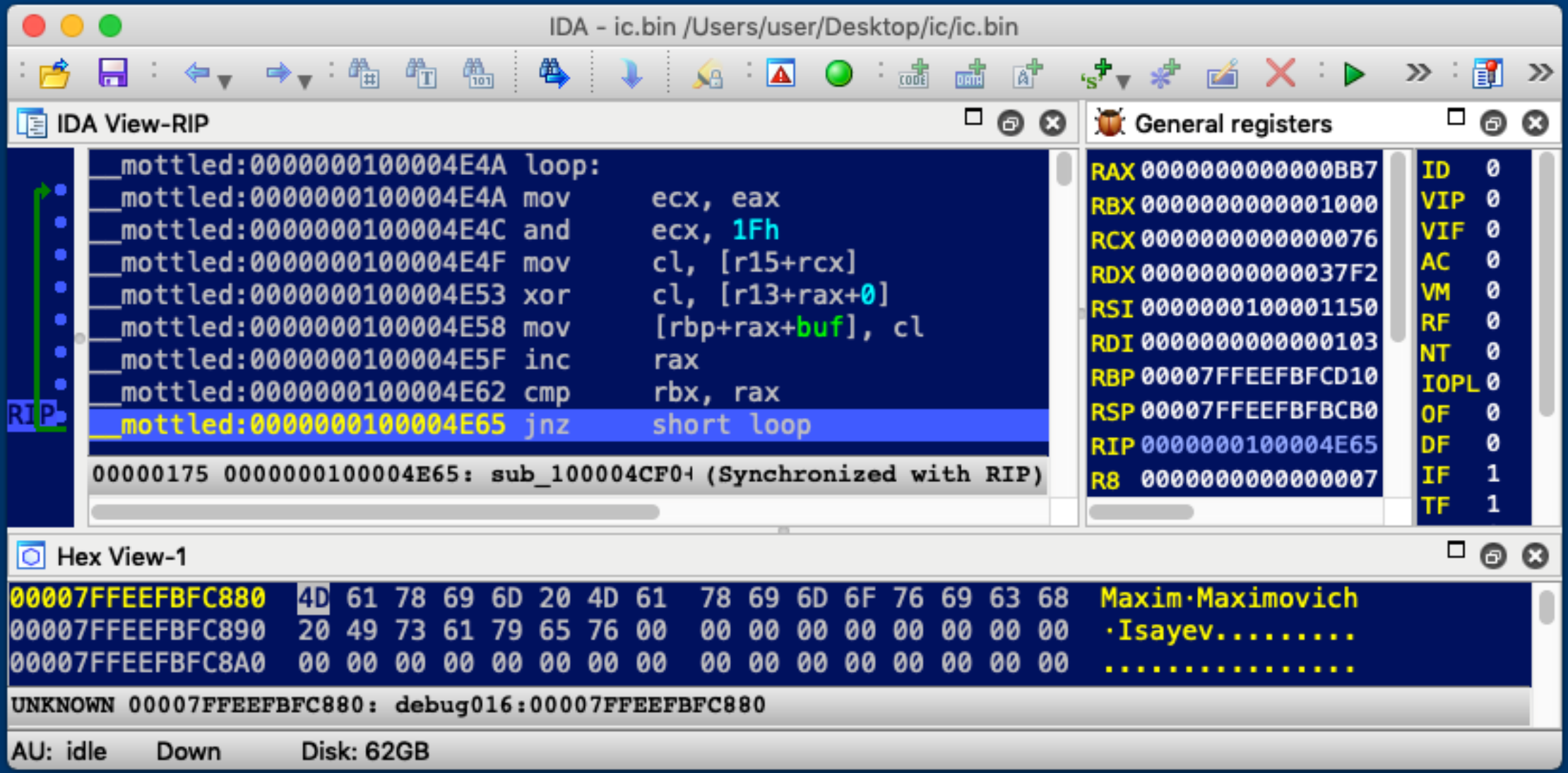
Decrypt with 32-byte XOR key:

```
nJvgccZUbkJMUaoapqPGcgEjPyGay6xx
```

Decrypting __text code section



Decrypting __text code section



Decrypted __text code section

Decrypted code section:

```
__text:000100001150                 public start
__text:000100001150    start        proc near
__text:000100001150                 push  0
__text:000100001152                 mov   rbp, rsp
__text:000100001155                 and   rsp, 0FFFFFFFFFFFFFFF0h
__text:000100001159                 mov   rdi, [rbp+8]
```

Anchor within encrypted section:

```
__text:000100001CF0    anchor       db  23h ; #
__text:000100001CF1                 db  2Bh ; +
__text:000100001CF2                 db  0Eh
__text:000100001CF3                 db  0Eh
```

Anchor within decrypted section:

```
__text:000100001CF0    anchor       db 'Maxim Maximovich Isayev',0
```

Hidden Marker

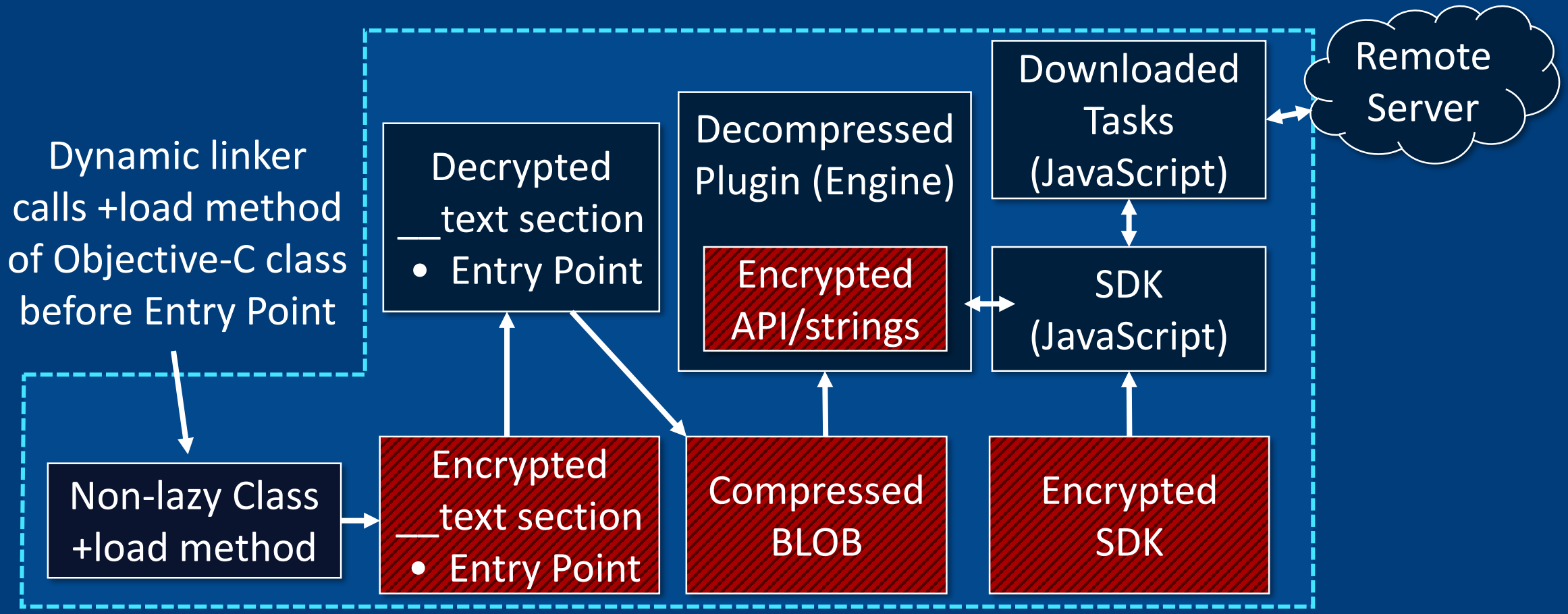
Maxim Maximovich Isayev (Максим Максимович Исаев) is a real name of Max Otto von Stierlitz, the lead character in a popular Russian book series written in the 1960s.

A Soviet James Bond, Stierlitz takes a key role in SS Reich Main Security Office in Berlin during World War II.

Never Before Had Stierlitz Been So Close To Failure

The Engine

The loaded module represents itself an engine driven by the JavaScript files.



VM Detection

The engine is able to detect the presence of VM through the method *checkPossibleFraud()*. This method is exposed to JavaScript, where it can be called as:

```
var isVm = system.checkPossibleFraud()>0 ? 1 : 0;
```

The engine compiles so called 'fraud' report that consists of the following details:

`vmVendor` Check if the MAC address starts from an address that is common for a given VM manufacturer. For example, “00:1C:42*” is for Parallels VM. Recognises over 35 VMs by known MAC prefixes:

- Virtualtek. Co. Ltd
- VMware, Inc.
- Microsoft Corporation (was: Connectix)
- Microsoft Corp.
- Microsoft Network Load Balancing Service Heartbeat
- Microsoft XCG
- Oracle Corporation (was: Virtual Iron Software)
- Oracle Corporation (was: Xsigo Systems, Inc.)
- Oracle Corporation (was: Sun Microsystems, Inc)
- CADMUS COMPUTER SYSTEMS
- Parallels ID.
- Egenera, Inc.
- First Virtual Corporation
- linux kernal virtual machine (kvm)
- Virtual Iron Software, Inc. (was: Katana Technology)
- Paravirtual Corporation (was: Accenia, Inc.)
- Virtual Conexions
- Virtual Computer Inc.
- virtual access, ltd.
- Virtual Instruments

VM Detection

`MAC_L`	MAC and IP addresses for all network interfaces
`Host UUID`	*gethostuuid()*
`hddName`	*DADiskCreateFromBSDName()* for `'/dev/disk0'` device
`usbFraud`	`ioreg -l | grep -e 'USB Vendor Name'`
`dispRats`	display ratio
`lastMove`	mouse position since the last mouse movement event
`lastRbt`	system up-time, since last reboot
`dmgLoc`	full path filename of the DMG file, in case it's executed by a sandbox under a generic name, i.e. a file hash
`fromDMG`	
`wndPos`	position and size of the app's window
`msePos`	mouse position, to see if mouse is in use

to recognise fingerprints of the common sandboxes

Engine Capabilities

The bundleware's engine consists of components, capable of doing the following:

- Browser manager
  - terminate browser process
  - set new home page
- Screenshot controller
  - take full screen snapshot with the mouse location
- Task manager
  - download and execute new tasks
  - create authorization for tasks, using given creds
- System controller
  - collect system OS version
  - collect all cookies from browsers
  - collect the list of all installed / running applications
  - check the presence of VM
  - add/remove applications to/from dock
  - get info about connected iOS devices:
  - device class, ID, serial number (iPod/iPad/iPhone)
  - search for files in the specified directory
  - terminate specified applications
  - read key values from user defaults
  - add an app to dock as persistent item
  - read text files
  - copy given directory to a new location
  - delete the specified directory
  - run specified script with '/bin/sh', as root
  - get detailed HDD information
  - collect network information
  - download files
  - display alerts
  - launch tasks/applications as root
  - copy/move files
  - save data to files
  - create/delete directories

Conclusions

- A popular bundleware product conceals a very powerful engine
- The engine resembles a backdoor as it unlocks full access to the system
- Memory injection is described in the “The Mac Hacker's Handbook”
- The engine is driven by symmetrically encrypted remote tasks
- A disturbing trend we’re witnessing – the continued ‘spill’ of the traditional Windows malicious techniques, such as run-time packing, strings/API obfuscation, memory injection into the world of Mac

SOPHOS

Cybersecurity made simple.