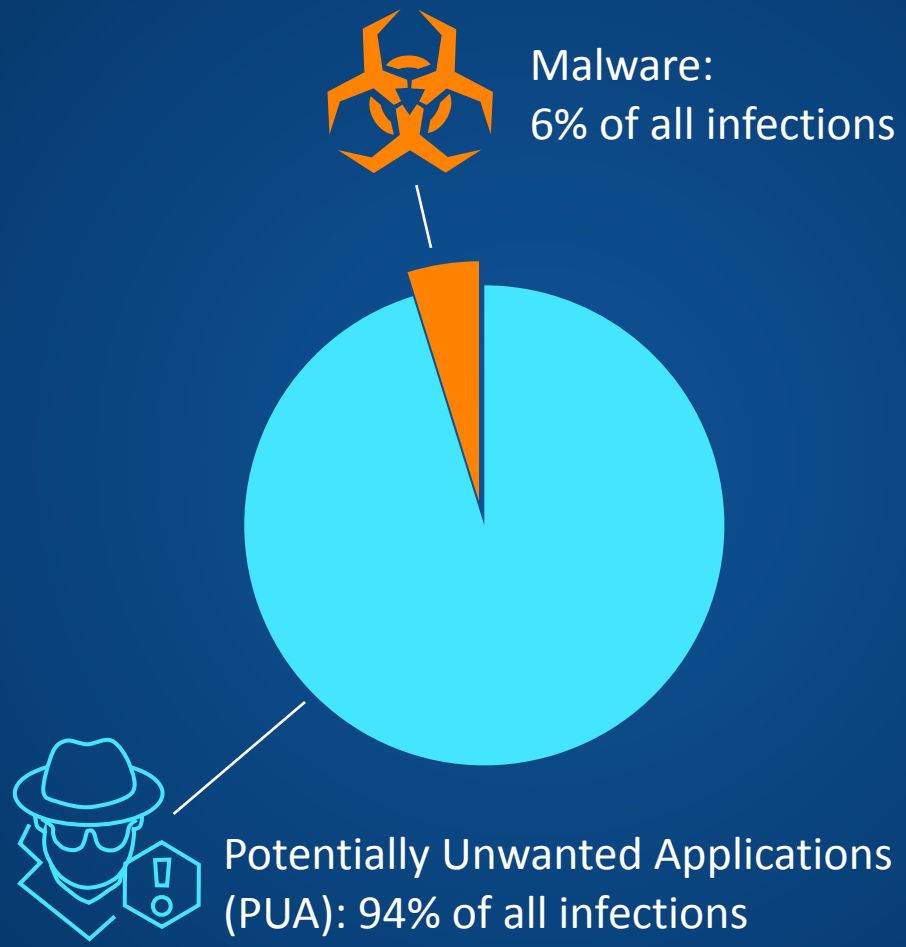
Never Before
Had Stierlitz Been So Close To Failure

Sergei Shevchenko
Threat Research Manager

SOPHOS

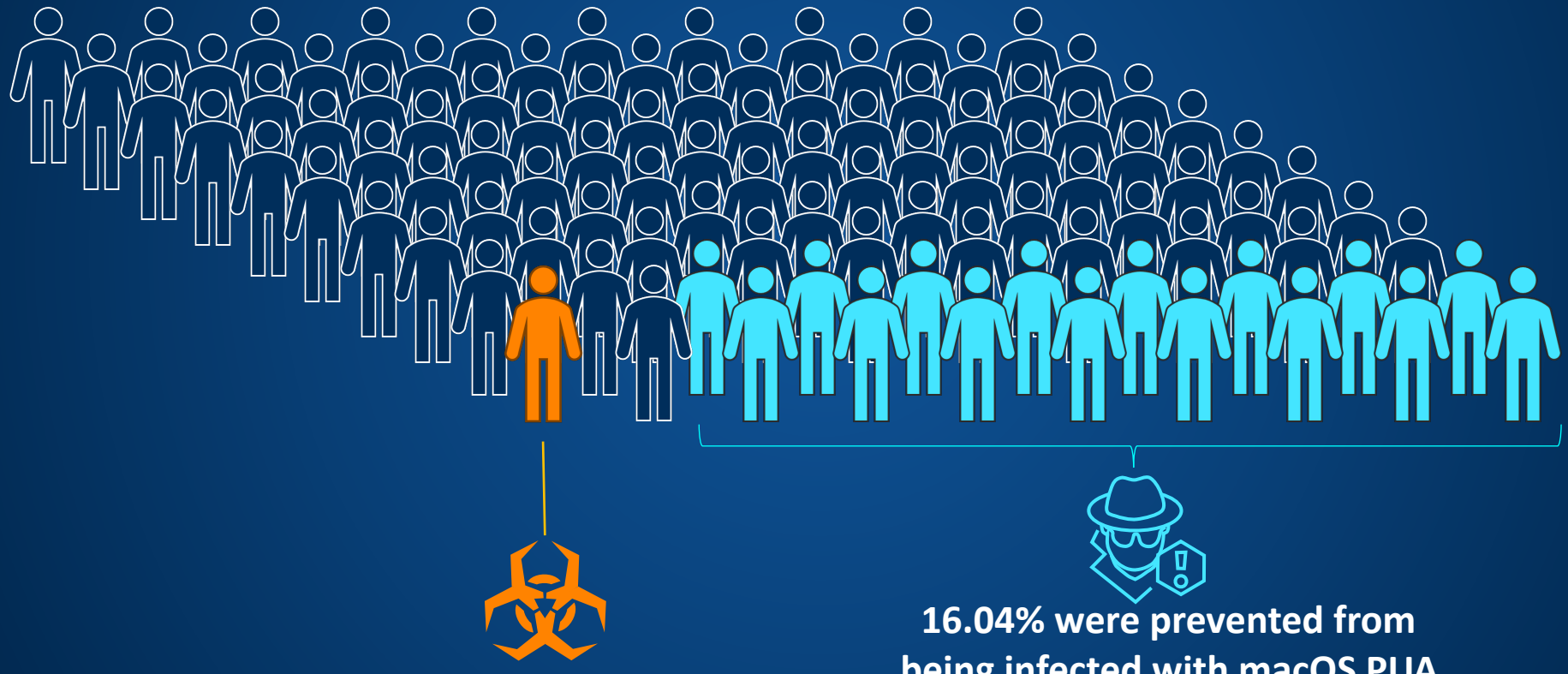
macOS Threat Reports



SOPHOS

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

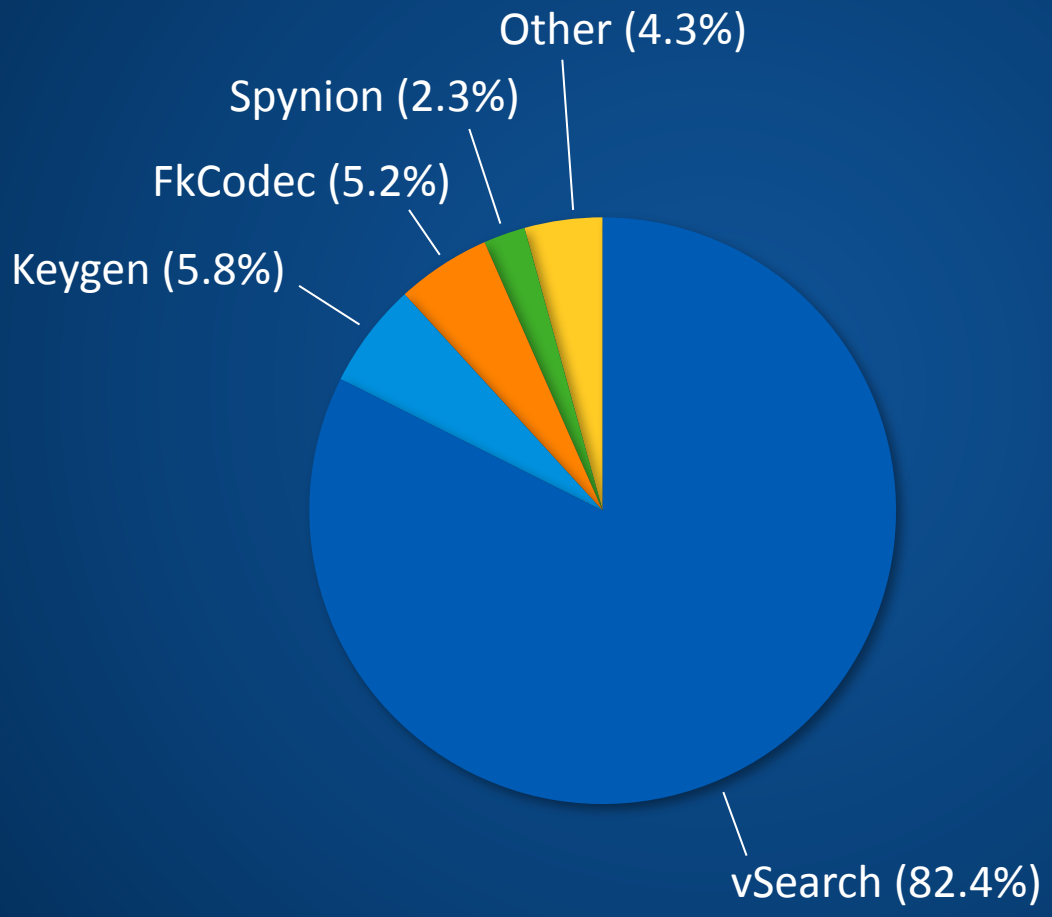
16.04% were prevented from being infected with macOS PUA

SOPHOS

macOS Current Threat Map

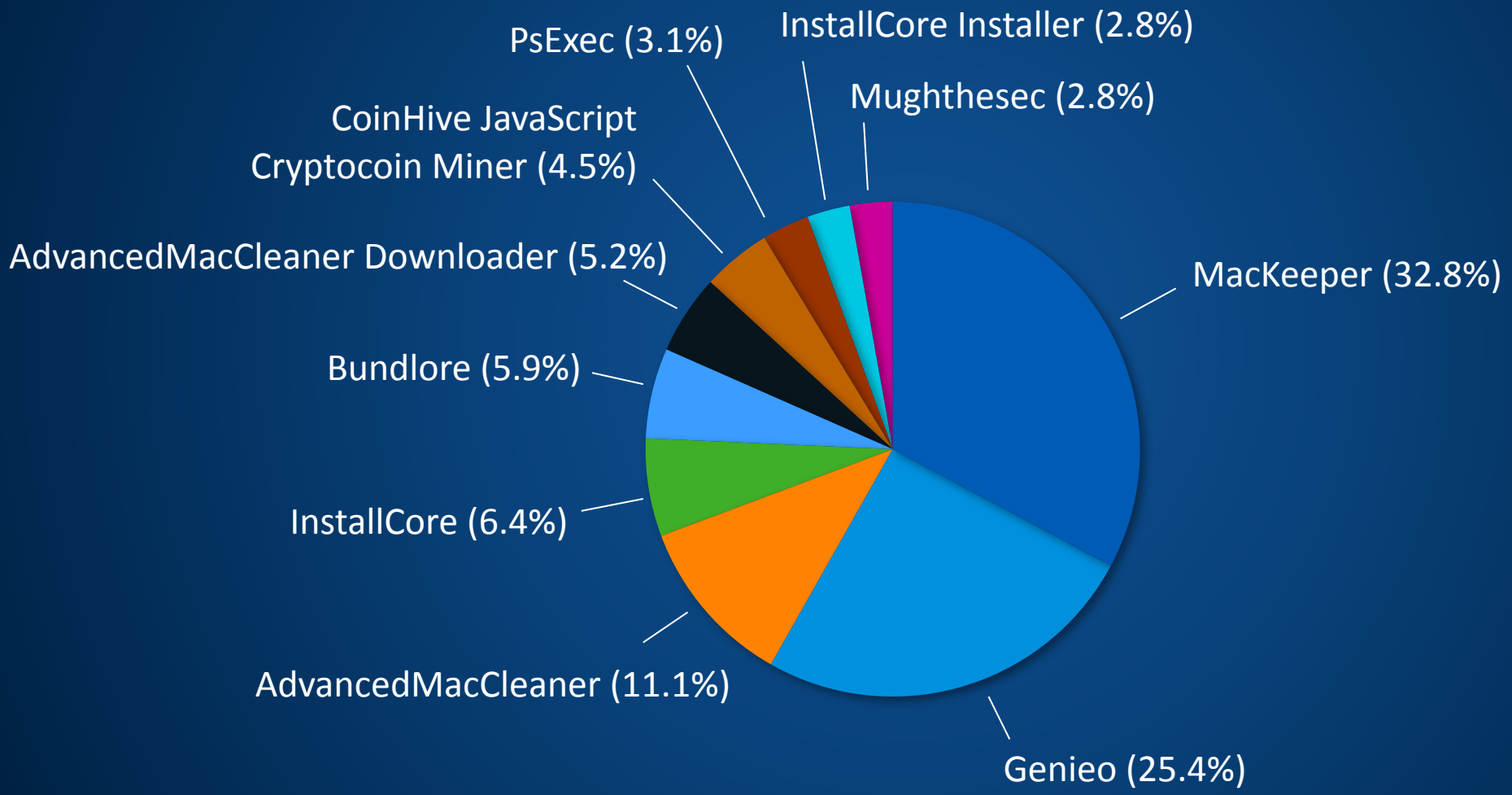
SOPHOS

macOS Top Malware Threats



SOPHOS

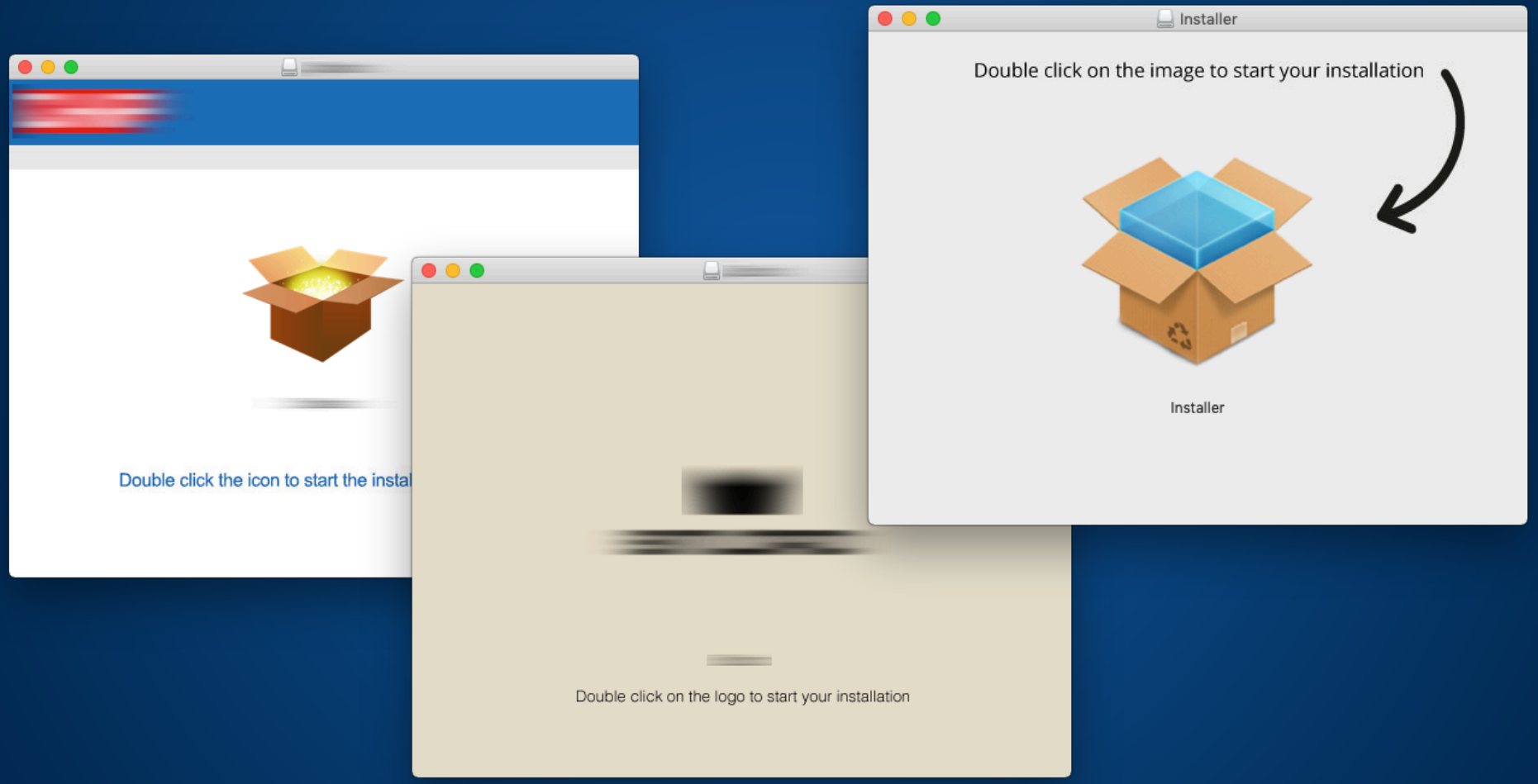
macOS Top PUA Threats



SOPHOS

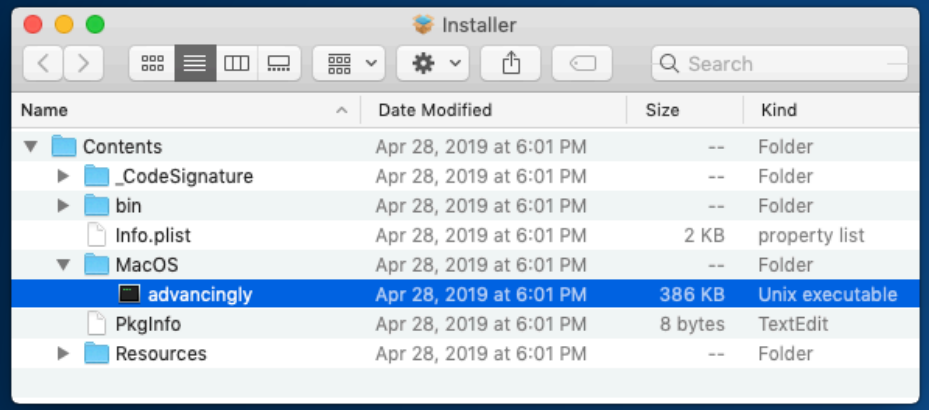
Installer is bundled with various forms of PUA

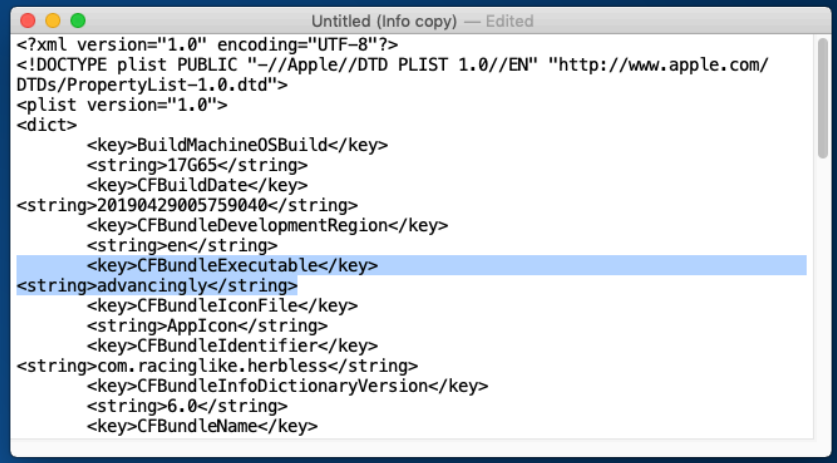
For the developers who want to monetize their work



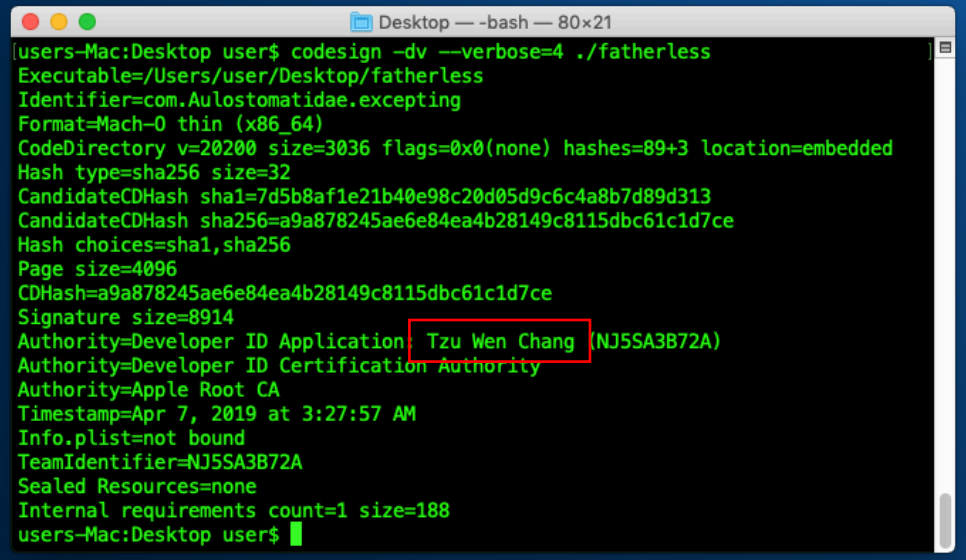
SOPHOS

Main Executable: random name / signer





Various / random signers



File name examples:

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

SOPHOS

Main Executable: Entropy

SOPHOS

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

SOPHOS

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]));
    }
}
```

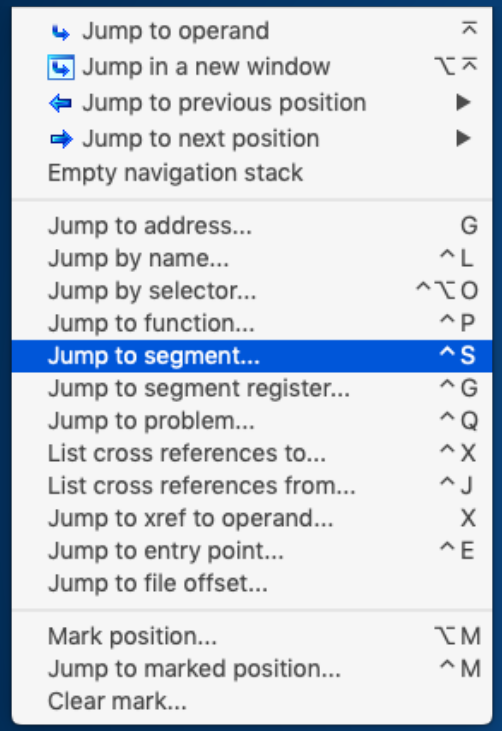
objc-file.mm

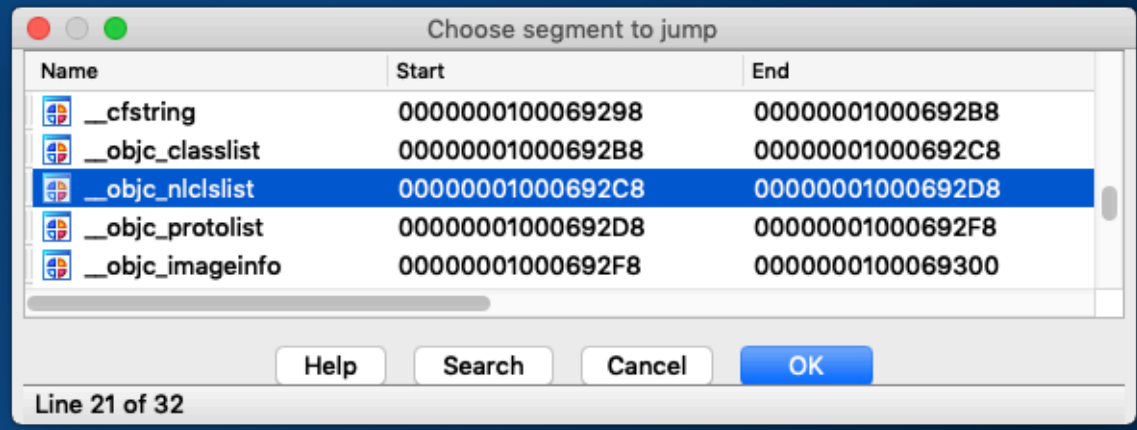
_getObjc2NonlazyClassList() collects non-lazy classes from the `__objc_nlclslist` data section

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

SOPHOS

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
```

SOPHOS

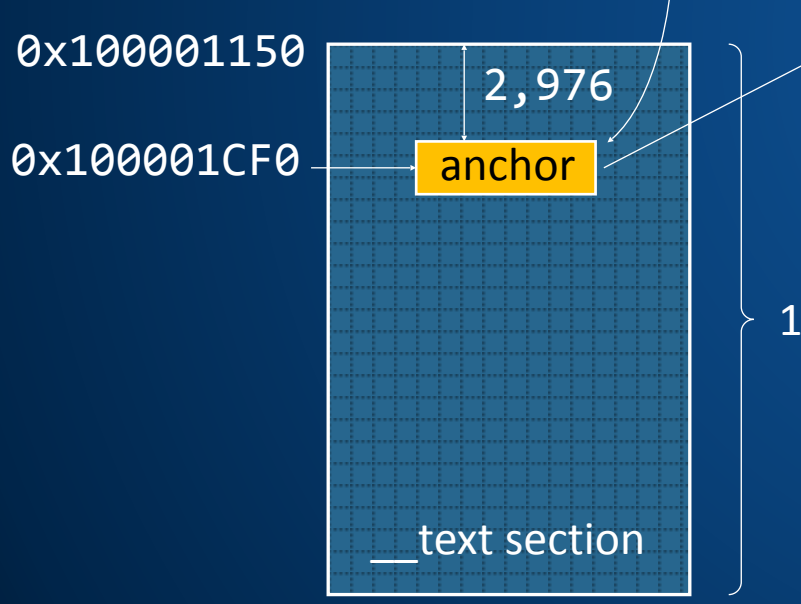
+[ListedUpaithric load]

```
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()
```

SOPHOS

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
```

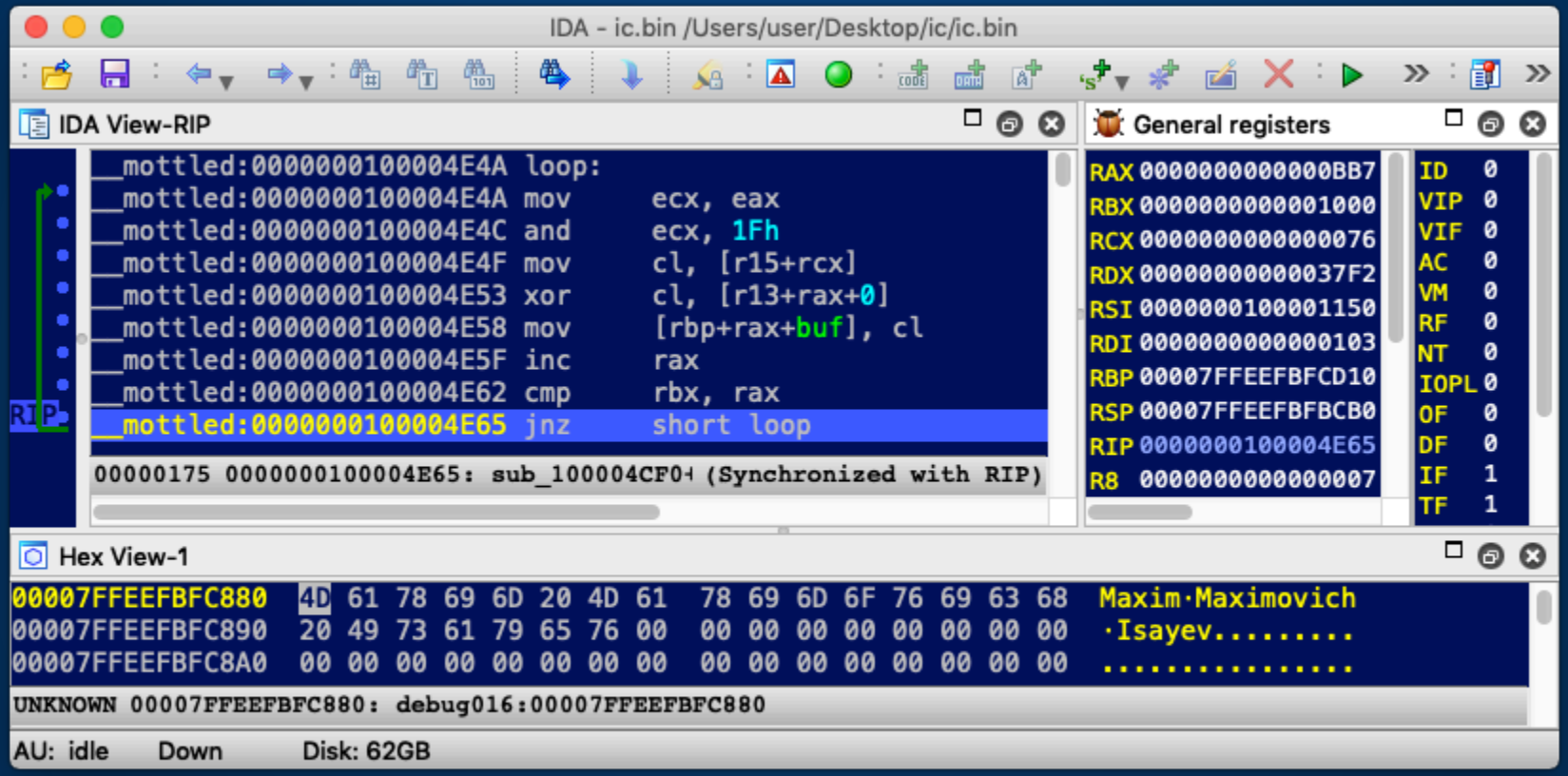
14,322 bytes

Decrypt with 32-byte XOR key:

```
nJvgccZUbkJMUaoapqPGcgEjPyGay6xx
```

SOPHOS

Decrypting __text code section



SOPHOS

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
```

SOPHOS

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.

Working as a deep undercover agent within SS, he diverts the German nuclear "Vengeance Weapon" research program into a fruitless dead-end.

SOPHOS

Never Before Had Stierlitz Been So Close To Failure

SOPHOS

String / API Encryption

```
__const:0000000100065BE0 xmmword_100065BE0 xmmword 'K@mqqthzyptgfTG]'
```

```
char decrypt(char ch, int index)
{
    return ch ^ (index + 0x13);
}
```

```
*(OWORD *)buf = xmmword_100065BE0;
*(WORD *)(buf + 16) = 0x244D;
buf[0] = decrypt(0x5D, 0);
index = 1;
do
{
    buf[index] = decrypt(buf[index], index);
    ++index;
} while (index != 17);
```

All the string decoding functions use different keys, but they implement one of the following 3 algorithms:

- simple XOR key
- simple key subtraction
- auto-incremented XOR key

1,228 encoded strings, decoded with 1,055 different functions

```
00007FFEEFBFFBE0  4E 53 41 70 70 6C 69 63  61 74 69 6F 6E 4D 61 69  NSApplicationMai
00007FFEEFBFFBF0  6E 24 00 00 0E 00 00 00  00 00 00 00 00 00 00 00  n$..............
```

SOPHOS

New String Obfuscation from April 2019

Each int number is encoded with a separate function, e.g. number 6 is encoded as:

```
get_6       proc near
            push    rbp
            mov     rbp, rsp
            mov     al, 3           ; al = 3
            shl     al, 2           ; al = 12
            movsx   ecx, al         ; ecx = 12
            mov     eax, 65         ; eax = 65
            xor     edx, edx        ; edx = 0
            idiv    ecx             ; 65 / 12, eax = 5
            mul     cl              ; eax = 60
            mov     cl, 65          ; cl = 65
            sub     cl, al          ; cl = 65 - 60 = 5
            inc     cl              ; cl = 6
            movsx   eax, cl         ; result = 6
            pop     rbp
            retn
get_6       endp
```

Hex-Rays Decompiler's output:

```
signed __int64 get_6()
{
    return 6;
}
```

SOPHOS

Dynamic Module Loading

Encrypted data stub (>300KB) stored in a separate section of the executable.

Data is read, validated (CRC32), decrypted and decompressed with *uncompress()* API from the loaded libz.1.dylib. The uncompressed data (>800KB) is data is loaded from memory as a plugin module with the help of *NSCreateObjectFileImageFromMemory()* and *NSLinkModule()* APIs.

```
(lldb) image list
…
[222] FABB97BC-...
```

```
->  0x100001ac6 <+1020>: callq  *%r13
```

R13 -> pointer to *NSLinkModule()*

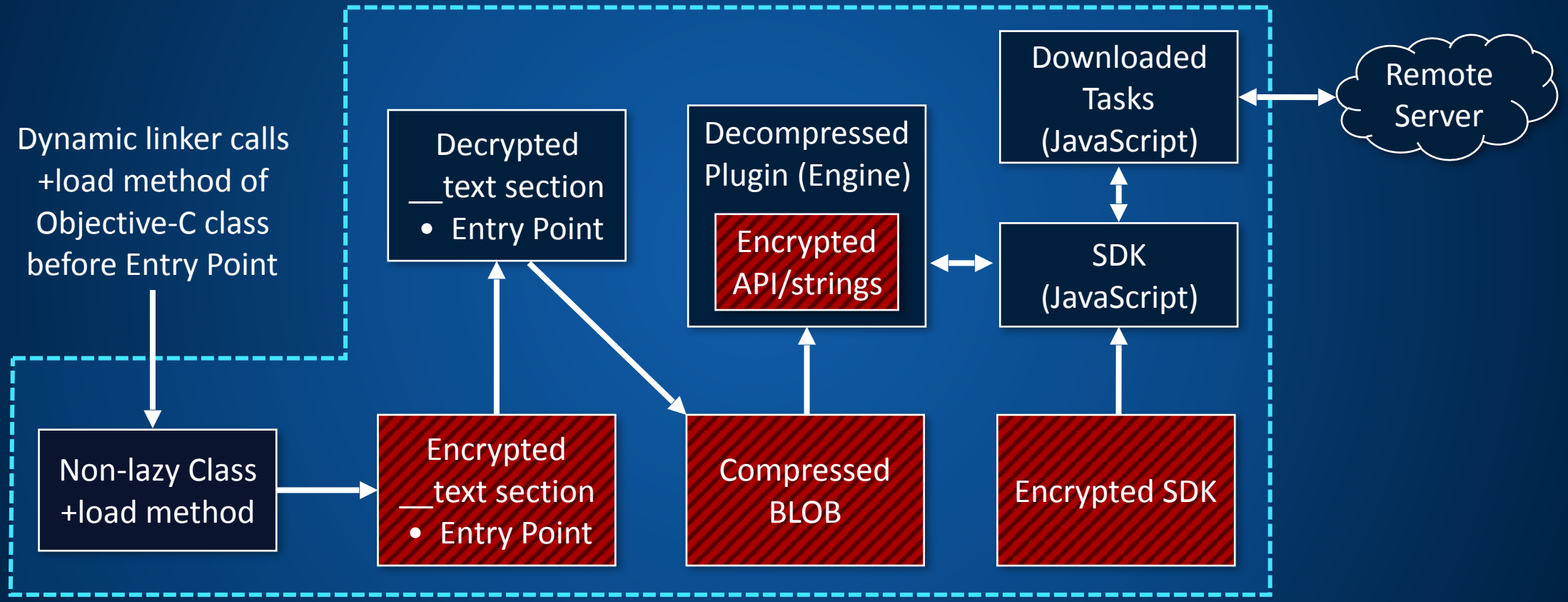
```
[223] C5F8F084-D151-3D02-9058-905A19117A90 0x0000000101a00000 image (0x0000000101a00000)
…
[265] B16080FC…

(lldb) mem read 0x0000000101a00000
0x101a00000: cf fa ed fe 07 00 00 01 03 00 00 00 08 00 00 00  ????............
0x101a00010: 1e 00 00 00 58 12 00 00 85 80 01 00 00 00 00 00  ....X...........
```

SOPHOS

The Engine

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



SOPHOS

Anti-Debugging

The anti-debugging defence is provided with *ptrace()* request named PT_DENY_ATTACH (0x1F), called from:

```
ptrace = 0x515D5A5D;                          // encrypted 'ptrace' string: 5D 5A 5D
51
ptrace_plus_4 = 0x5752;                       //                            52 57
ptrace_plus_6 = 0x33;                         //                            33
ptrace[0] = add_2D_xor(0x5D, 0);              // decrypt 1st char (5D ^ (2D + 0))
i = 1;                                        // start loop from the 2nd char
do
{                                             // decrypt the rest
    ptrace[i] = add_2D_xor(ptrace[i], i);     // ptrace[i] ^= 2D + i
    i++;
}
while (i != 6);                               // 6 chars from the 2nd char, incl /0
fn_ptrace = dlsym(RTLD_NEXT, &ptrace);        // get proc addr from the linked dylibs
return fn_ptrace(PT_DENY_ATTACH, 0, 0, 0); // call ptrace() by pointer, deny
tracing
```

If the process is being debugged, it will exit with the exit status of ENOTSUP (45), *'error, not supported'*. Otherwise, it sets a flag that denies future traces – an attempt to debug it with this flag set will result in a segmentation violation exception.

SOPHOS

Anti-Debugging

```
mac:/ user$ sudo lldb /Users/user/Installer/Installer.app
(lldb) target create "/Users/user/Installer/Installer.app"
Current executable set to '/Users/user/Installer/Installer.app' (x86_64).
(lldb) r
Process 1280 launched: '/Users/user/Installer/Installer.app/Contents/MacOS/
radiosurgical' (x86_64)
Process 1280 exited with status = 45 (0x0000002d)
```

By stepping over the deny_attach() call (or NOP-ing the 5 bytes of the call),
the anti-debugging trick above can be easily circumvented:

```
->  0x103dd1ff5 <+25>: callq  0x103e30cd3        ; call the function with ptrace()
      0x103dd1ffa <+30>: callq  0x103de03aa       ; ICCrashLogger::sharedLogger()
      0x103dd1fff <+35>: movq   %rax, %rdi
(lldb) re w pc `$pc+5`       ; step over deny_attach() by adding 5 bytes to $pc
(lldb) x/2i $pc              ; now $pc (pseudo-name for RIP) points to next instr
->  0x103dd1ffa: e8 ab e3 00 00  callq  0x103de03aa ; ICCrashLogger::sharedLogger()
     0x103dd1fff: 48 89 c7        movq   %rax, %rdi
```

SOPHOS

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

SOPHOS

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

SOPHOS

Crash Logs

The crash logger sends GET request to a remote script, disguised as a PNG file:

```
http[://][removed].us-west-2.compute.amazonaws[.]com/black.png
```

The stats it submits to the remote script are encoded as URL parameters:

- `crash=1`
- `os=mac`
- `appkit=%APP_KIT%`
- `ver=%VERSION%`
- `ldebug=%LIVE_DEBUG%`
- `backtrace=%CALL_BACKTRACE%`

SOPHOS

Config Files: 1/2

The installer uses 2 configuration files.

The first one is dynamically extracted from its own body.

This configuration is encrypted with AES-128 algorithm. To locate the encrypted config, the installer module parses the contents of the file.

For each pair of bytes, it subtracts one byte from another, until if locates a specific signature that consists of 7 64-bit integers.

Decrypted config specifies the URL of an application to download and install:

```
PRODUCT_TITLE = [removed]
PRODUCT_DESCRIPTION = [removed]
DOWNLOAD_URL = http%3A%2F%2F[removed]-Installer.dmg
PRODUCT_LOGO_URL = http%3A%2F%2F[removed].png
ROOT_IF_INSTALLED = [removed]
```

SOPHOS

Config Files: 1/2

```
if ( ptr != (_BYTE *)&FEEDFACF + 1 )
{
  found = 0LL;
  do
  {
    prev = ptr[(_QWORD)index - 2];
    curr = ptr[(_QWORD)index - 1] - prev;
    if ( curr < 0 )
      curr = ptr[(_QWORD)index - 1] - prev + 256;
    if ( curr == signature[found] )
    {
      if ( ++found == 7 )
        goto found_inj;
    }
    else
    {
      found = 0LL;
    }
    --ptr;
  }
  while ( ptr != (_BYTE *)&FEEDFACF + 1 );
}
ptr = (_BYTE *)(&FEEDFACF + 1);
```

```
HEADER:0000000000000000 FEEDFACF dd 0FEEDFACFh
```

```
__const:00000000000B7430 signature dq 0Fh, 9, 3Eh, 23h, 7, 86h, 0Ch, 0
```

SOPHOS

Config Files: 2/2

The 2nd configuration file is provided as a JavaScript file, and is decrypted with the other SDK files from the app's Resources directory.

This configuration defines multiple operational parameters, such as report and ad servers:

```
var appInfo = {
    report: 'http://rp.[removed].com',
    ad_url: 'http://os.[removed].com/[removed]',
    requires_root: false,
    root_if_installed: [''],
    skip_vm_check: false,
   ...
```

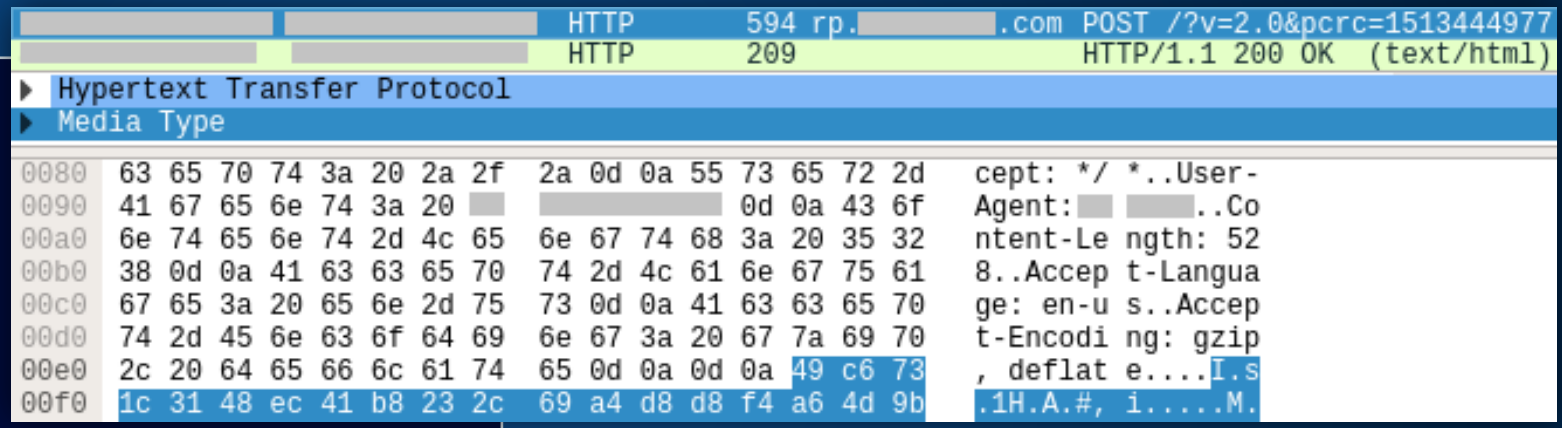
SOPHOS

Report Server

The report server from the configuration is used to receive posted reports.

For example, an example below demonstrates what data is posted to the report server:

PROD_TITLE = [REMOVED]
schemeName = MacDarwenDLM
OSName = OSX
OSVer = 10.12
OSLang = en
_makeDate = 201811091722
BRW = Safari
OSPlat = 2
MAC_L = [REMOVED]000000000000%3A127.0.0.1%3A24%3A0
hddSize = 107374182400
_makerver = total20181107115116
Isuseradmin = 1
isVmDef = 1
inst_flv = no_injection_106.1712
QuitPage = welcomePage



The collected data is assembled into a text, then encrypted with AES-128, and posted to the server

SOPHOS

Remote Tasks

Remote tasks are received encrypted from the ad server:

```
POST http://[removed].com/[removed]
USER-AGENT: ICMAC
Response:
    Header: X-ICSCT-SERVER-NAME: [removed]
    Data: 85,368 bytes binary [6c ec 6c 99...]
```

When the received task is decrypted, its data is split into named sections. Each section is surrounded with the following comments:

```
var namestartstr = '<!--SECTION NAME="';
var nameendstr = '"-->';
var sectionendstr = '<!--/SECTION-->';
```

The parser extracts JavaScript code from those sections. That code will then rely on APIs exposed by the SDK, to drive the engine that exposes its own API interface to the SDK.

An analysis of the tasks received from the ad server reveals no malicious activity.

SOPHOS

Engine Capabilities

The bundleware's engine consists of the several 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

SOPHOS

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

SOPHOS

Security made simple.