```
on isSearch(myURL)
    set isGL to ((myURL contains "q=") and (myURL contains "google.com") and (myURL contains "/search") and (myURL contains
        "channel=mac_bm") and (myURL does not contain "&ei=") and (myURL does not contain "oi=ddle"))
    set isY to ((myURL contains "yahoo.com") and (myURL contains "/search") and (myURL contains "ei=utf-8&fr=aaplw&p="))
    set isB to ((myURL contains "bing.com") and (myURL contains "/search") and (myURL contains "form=APMCS1"))
    if (isGL or isY or isB) then
        return true
    else
        return false
    end if
end isSearch
on addOne()
    set OurSearchUrl to "$1"
    if application "Safari" is running then
        tell application "Safari"
            set theURL to URL of current tab of window 1
            set isSrch to isSearch(theURL) of me
            if isSrch is equal to true then
                set s to theURL
                set AppleScript's text item delimiters to {"q=", "p="}
                set urlQuery to text item 2 of s
                set s to urlQuery
```

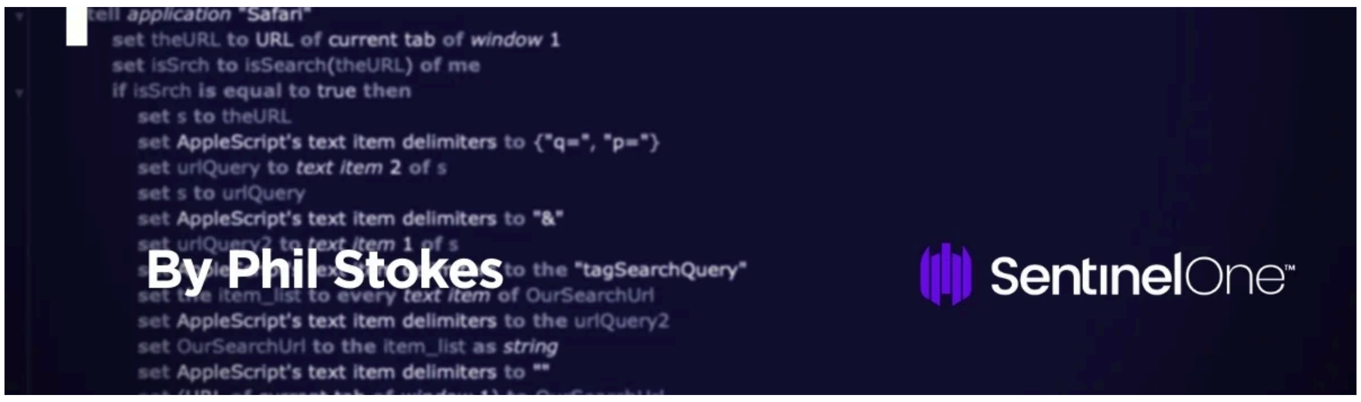
How AppleScript Is Used For Attacking macOS

March 16, 2020
by Phil Stokes

PDF

When we think about security on macOS and the tools used by offensive actors, whether those are real in the wild attacks or red team exercises, we tend to think of things like python scripts, shell scripts, malicious documents, shady extensions and of course, the fake, doctored or trojan application bundle. There is much less attention in the security field on AppleScript – a built-in macOS technology – despite the fact it's been around for as long as Python and predates macOS 10 itself by 8 or 9 years.

As I'll show in this post, AppleScript is widely used by offensive actors. This includes its use in adware, its use for tasks such as persistence, anti-analysis, browser hijacking, spoofing and more. Worryingly, given the lack of attention paid to AppleScript in the research community, that is all without even leveraging some of AppleScript's most powerful or unique features, some of



Why Have the Good Guys Ignored AppleScript?

Unlike Bash and other shell languages, and unlike Python, a cross-platform, beginner-friendly scripting language that has achieved widespread adoption and praise, AppleScript is a language peculiar to macOS; not only can you NOT use it on other Desktop operating systems like Windows and Linux, you can't even use it on Apple's other operating systems like iOS and iPadOS.

As a language, AppleScript has a reputation for being quirky, slow and difficult to develop even simple scripts with. It's quirky because it attempts to use "natural language" but in a grammar that is entirely artificial, often inconsistent and frustratingly unintuitive. It's also incredibly verbose. Compare the code for the simple task of counting the number of items in `/usr/bin` directory. As ever, a shell script will always be the most concise:

```
ls -l /usr/bin | wc -l
```

Python is a little more verbose, but still fairly clean and familiar:

```
#!/usr/bin/python
import os
path, dirs, files = next(os.walk("/usr/bin"))
print len(files)
```

The AppleScript version, however, is something of an entirely different nature.

```
tell application "System Events"
	set theFiles to name of every file of folder "bin" of folder "usr" of startup d
	count of theFiles
end tell
```

Recent Posts

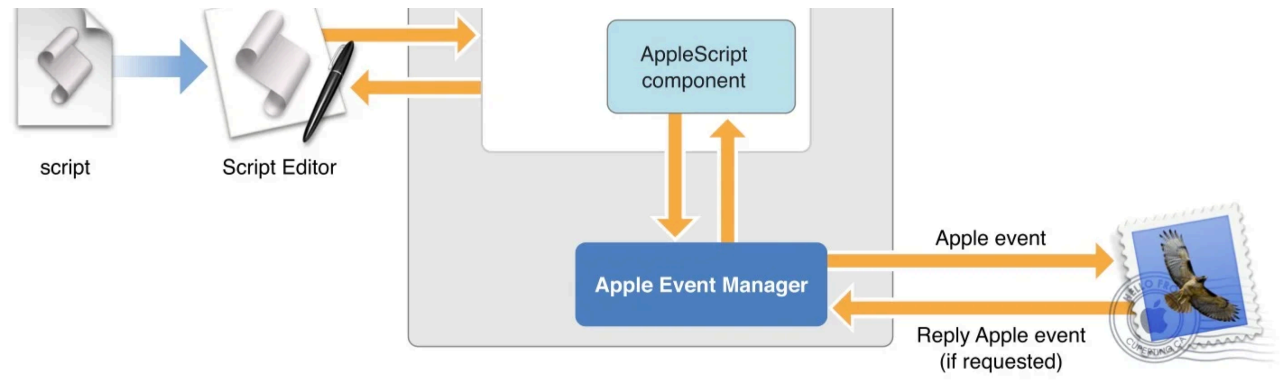
Safely Expanding the Frontiers of AI & LLMs | S Ventures' Investment in Galileo
October 25, 2024

The Good, the Bad and the Ugly in Cybersecurity – Week 43
October 25, 2024

Climbing The Ladder | Kubernetes Privilege Escalation (Part 1)
October 23, 2024

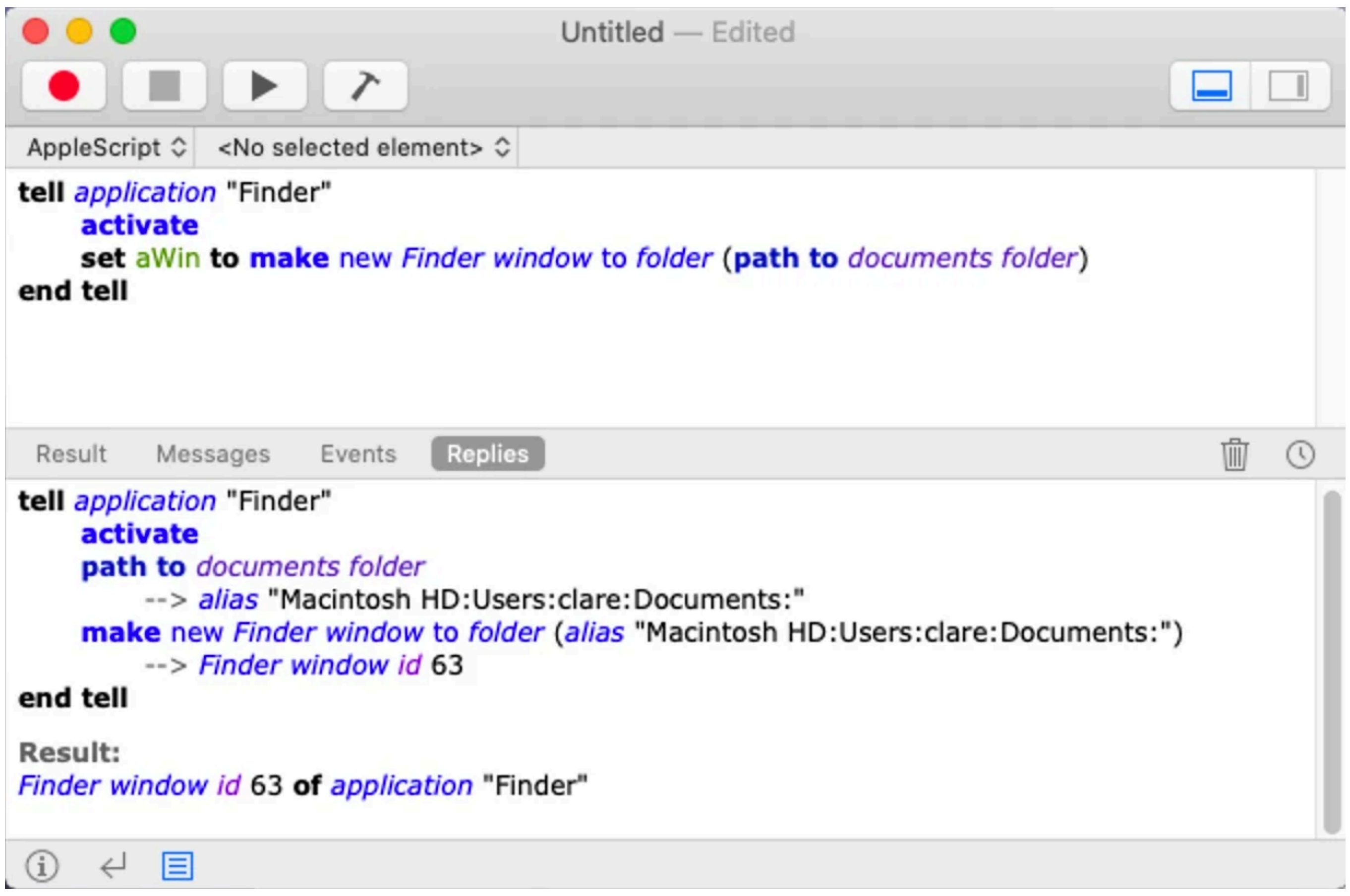
Blog Categories

Cloud
Company
Data Platform
Feature Spotlight
For CISO/CIO
From the Front Lines
Identity
Integrations & Partners
macOS
PinnacleOne
The Good, the Bad and the Ugly



Source: AppleScript Overview, © 2002, 2007 Apple Inc.

And it's historically been difficult to develop scripts with AppleScript because most people who come to it will attempt to use the free, built-in but notoriously spartan (Apple)Script Editor.app, which lacks almost every and any feature developers normally expect and need. There's no debugger, there's no variable introspection, there's no code snippets or effective code completion, to name just a few missing features.



```
tell application "Finder"
	activate
	set aWin to make new Finder window to folder (path to documents folder)
end tell
```

```
tell application "Finder"
	activate
	path to documents folder
		--> alias "Macintosh HD:Users:clare:Documents:"
	make new Finder window to folder (alias "Macintosh HD:Users:clare:Documents:")
		--> Finder window id 63
end tell

Result:
Finder window id 63 of application "Finder"
```

Until recently, the only 3rd party alternative to Script Editor was priced at $200 and had a time-limited, 20-day demo period.

In short, the investment in time, effort and money required to produce something that still, after all that, can only be used on the macOS Desktop, effectively puts AppleScript at the bottom of the list for most people when it the goal being to allow ordinary users to chain together repetitive tasks and execute them without further user interaction. For example, you can have Mail.app automatically trigger a script when it receives an email from a certain sender, or with a particular keyword in the subject line or content, extract whatever details you want from the email, and then populate a database in Excel or Numbers with the desired information, formatted and sorted on-the-fly as the data comes in. There's no need for the user to be involved in any of this once the script is set up.

And as it turns out, automating interapplication communication and sidestepping user interaction is a godsend for malware authors. What could be more useful than bending popular applications like email clients, web browsers and the Microsoft Office suite to your will without needing to involve the user (*aka* in this scenario, **the victim**)?

And so, despite its general lack of appeal in almost all possible audiences for a scripting or programming language, there is one audience that does use AppleScript widely, if not particularly artfully or cleverly, and that is threat actors. Let's look at some examples.

Recent Examples of AppleScript in macOS malware

A recent browser hijacker targeting Safari installs a hidden LaunchAgent that, via a shell script, loads, compiles and executes AppleScript.

It starts with a shell installer script packaged inside a DMG that's supposed to

`55529224e9f70f5cab007e2ca98f6aec5cf31eb923fdfc09f60c01cc45c80666`

Which eventually produces the hidden launch agent that executes another shell script containing the following AppleScript code, launched via `osascript`.

`cdaa2121d79031cf39159198dfe64d3695a9c99ff7c3478a0b8953ade9052ecc`

The purpose of the AppleScript is to replace the user's search query on popular search engines google, bing and yahoo, with one provided by the attacker's shell script. It's a quick and easy way for bad actors to make money out of clicks which negatively impacts the victim's productivity.

While this particular sample comes packed in a separate file, many others write their AppleScript directly into a MachO binary, either in plain text strings or in obfuscated base64 or similar encoding.

The following strings extracted from a Bundlore installer show that the code tries to force enable JavaScript execution in Google Chrome, then uses AppleScript to execute it in the active tab of the browser's front window.

Strings from

The next sample is a variant of a Pirrit malware.

`21331ccee215801ca682f1764f3e37ff806e7510ded5576c0fb4d514b4cf2b7c`

The authors use both plain text AppleScript and base64 encoded AppleScript, targeting Safari, Chrome and Firefox browsers.

Here's some of the decoded base64 targeting Firefox and attempting to perform automated keystrokes to copy the current URL to the user's clipboard before replacing the URL to that of the attacker's choice:

Using AppleScript Without Apple Events

Speaking of base64, the next example illustrates something that many ordinary users and developers have overlooked about AppleScript, but which offensive actors have not: you can use AppleScript to execute any other kind of script, including python scripts like this one which drops the Empire exploit kit.

And what's true of AppleScript and Python, is true of AppleScript and Perl, AppleScript and Bash and indeed AppleScript and absolutely any command line tool at all: you can call them all and bring their functionality into your AppleScript and combine that with other utilities.

wise, more or less on a par with Objective C executed in a MachO binary.

And interestingly, although we haven't seen threat actors making use of these powerful capabilities so far, there's at least two reasons why we may well do so in the future: first to avoid detection, and second, because of the easy availability of a good development environment.

Using AppleScript to Avoid Detection

Avoiding detection on execution is a primary objective for all malware (even ransomware, which doesn't want to get noticed until after execution). AppleScript offers offensive actors a plethora of ways to execute. In addition to simply executing a `.scrpt` file, you can run AppleScripts from Mail rules, from a shell script, in memory, from the command line, from within a MachO, in a plain text, uncompiled file, from an Automator workflow, from a Folder Action, a Finder Service or from a Calendar event.

Because of AppleScript's ability to execute Objective C code without needing a compiled binary, this opens up a number of interesting attack possibilities. It also potentially opens up the ability to bypass detection tools based on Apple's new kextless security framework introduced in macOS Catalina 10.15.

In an excellent post by Cedric Owens called Taking the macOS Endpoint Security Framework For A Quick Spin, Owens sets out to test what can and cannot be detected using three recently developed 3rd-party security tools that leverage the new Apple Endpoint Security framework.

One of the interesting things that Owens found was that if you tried capturing the user's clipboard via `osascript` and vanilla AppleScript, this activity would be easily picked up by all the tools he was testing.

```
osascript -e 'the clipboard'
```

However, when using the native Cocoa API, NSPasteboard, none of the Endpoint Security framework-powered tools Owens tested appeared to capture that activity. But now, of course, we can execute NSPasteboard natively from AppleScript, too!

a thing of the past. The 3rd party alternative mentioned earlier in this post now has an unlimited free trial version and retails at half of its old price; more importantly, it also allows you to drag and drop a great deal of boilerplate code like that used in the script above straight into your scripts. And it provides developer-friendly functionality like code completion and API lookups that really take the pain out of developing AppleScript code.

Let's look at another example. A lot of offensive operations want to avoid targets that are running particular software. Little Snitch is a prime example, various VM software is another. We can easily get a list of running apps by name and test for those, again directly by calling into Cocoa APIs, this time via NSWorkspace.

If we just want a true/false test for the existence of specific apps, we can just put the app names in a list and return true on the first hit.

In other words, by leveraging AppleScript's hook into Cocoa frameworks, we can execute native code without the overhead of building MachO binaries or MachO apps (although you can do both of those with AppleScript, too!). We can do this filelessly so that we don't get caught by new 'kextless' tools such as those tested by Owens, and we can execute this code in far more ways than any other kind of code available on macOS, whether that's shell scripts, Python scripts or native macOS bundles.

Conclusion

The upshot here is that the main reasons why the good guys have typically eschewed AppleScript are in fact no longer relevant or true. Since we've already seen threat actors taking advantage of AppleScript despite those obstacles in the past, it's only reasonable to assume that they may delve deeper into what this unique language has to offer in the future. Thanks to the native hook into Objective C and the powerful Cocoa frameworks, the variety of execution methods and now the availability of an excellent, free-to-use IDE, AppleScript has become a tool that is powerful, versatile and easy-to-develop with.

Attackers will always look to exploit the things defenders ignore, and to say that AppleScript has been ignored by the security community thus far is an understatement. I have elsewhere described AppleScript as "the PowerShell of macOS". Certainly, it's time we stopped thinking of AppleScript as the Most Unlovable Programming Language Ever and recognize that it may actually be the One macOS Programming Language to Rule Them All.

- Backdoor Activator Malware Running Rife Through Torrents of macOS Apps
- Protecting macOS | 7 Strategies for Enterprise Security in 2024
- macOS Adload | Prolific Adware Pivots Just Days After Apple's XProtect Clampdown

Read More

Get a demo

Defeat every attack, at every stage of the threat lifecycle with SentinelOne

Book a demo and see the world's most advanced cybersecurity platform in action.

SentinelLabs

SentinelLabs: Threat Intel & Malware Analysis

We are hunters, reversers, exploit developers, & tinkerers shedding light on the vast world of malware, exploits, APTs, & cybercrime across all platforms.

Wizard Spider and Sandworm

MITRE Engenuity ATT&CK Evaluation Results

SentinelOne leads in the latest Evaluation with 100% prevention. Leading analytic coverage. Leading visibility. Zero detection delays.