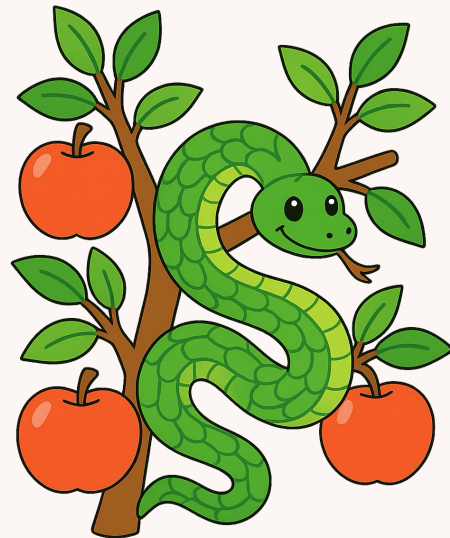
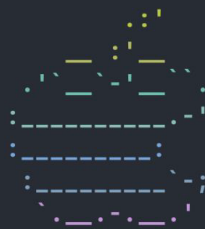


A Snake in the Apple Tree

Bridging Python and Objective-C for
Red Team Tradecraft





Hello, I am **Luke (@rookuu)**!

PROFESSIONAL

- Sr. Red Team Engineer @ [GitHub](#), specializing in macOS malware and TTPs.
- Co-Founder of [Phorion](#), a cybersecurity startup focused on macOS endpoint security.
- Previously held various roles in consultancy and internal security teams.

CONTACT

GitHub: @rookuu
Twitter: @rookuu_

INTERESTS

- ◆ All things offensive security in the macOS space
- ◆ Any sport that involves 2 feet on a board and some mild risk
- ◆ Ice hockey... playing, watching - go pens!

Red Team 101*

- We hack our employer or clients (with their permission), in order to identify and demonstrate security issues in a **real-world** setting.
- Operations have **tangible goals** for the attacking force to achieve, for example, steal \$\$\$ from a bank.
- The defending force is the **Blue Team**, who protect the organisation from attackers (just like us!)
- This work is done **covertly**, and the Blue Team is often not given any information, or even that it is happening in the first place.

**The industry does not agree on a common definition of red team work, and this topic is often in debate. This is my opinion only, other folks will probably have different ones!*

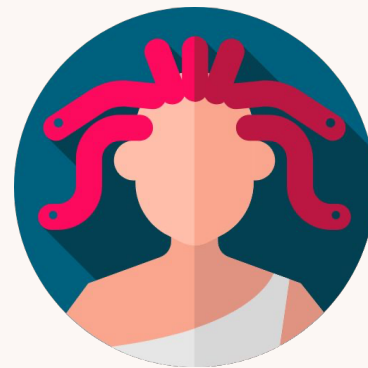
1. Make someones computer do bad things

1. Make someones computer do bad things
2. Don't get caught doing it

1. Make someones computer do bad things
- 2. Don't get caught doing it**



- **Mythic**
- Open-source C2 framework written by Cody Thomas (@its_a_feature)
- Provides the framework for deploying implants and giving them tasks.



- **Medusa**
- Open-source implant for Mythic written by Alfie Champion (@ajpc500).
- Written in Python. The actual code that runs on the target's device.

Now what?

- We could look around the file system for
 - SSH keys
 - Cookies
 - Password files
 - Keychain
- Install persistent access via a launchdaemon
- Keylog the user
- Take a screenshot to see what the user is doing
- Sniff the clipboard for interesting content

Now what?

- We could look around the file system for
 - SSH keys
 - Cookies
 - Password files
 - Keychain
- Install persistent access via a launchdaemon
- Keylog the user
- Take a screenshot to see what the user is doing
- **Sniff the clipboard for interesting content**

macOS Commands

Command	Syntax	Description
clipboard	<code>clipboard</code>	Output contents of clipboard (uses Objective C API, as outlined by Cedric Owens, macOS only, Python 2.7 only).

```
python3
→ ~ python3
Python 3.13.3 (main, Apr  8 2025, 13:54:08) [Clang 17.0.0 (clang-1700.0.13.3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import Cocoa
Traceback (most recent call last):
  File "<python-input-0>", line 1, in <module>
    import Cocoa
ModuleNotFoundError: No module named 'Cocoa'
>>> □
```

How could you implement a clipboard stealer?

```
man pbpaste
PBCOPY(1)                                General Commands Manual                                PBCOPY(1)

NAME
  pbcopy, pbpaste - provide copying and pasting to the pasteboard (the Clipboard) from
  command line

SYNOPSIS
  pbcopy [-help] [-pboard {general | ruler | find | font}]

  pbpaste [-help] [-pboard {general | ruler | find | font}] [-Prefer {txt | rtf | ps}]

DESCRIPTION
  pbcopy takes the standard input and places it in the specified pasteboard. If no
  pasteboard is specified, the general pasteboard will be used by default. The input is
  placed in the pasteboard as plain text data unless it begins with the Encapsulated
  PostScript (EPS) file header or the Rich Text Format (RTF) file header, in which case it
  is placed in the pasteboard as one of those data types.

  pbpaste removes the data from the pasteboard and writes it to the standard output. It
  normally looks first for plain text data in the pasteboard and writes that to the standard
  output; if no plain text data is in the pasteboard it looks for Encapsulated PostScript;
  if no EPS is present it looks for Rich Text. If none of those types is present in the
  pasteboard, pbpaste produces no output.

  * Encoding:

  pbcopy and pbpaste use locale environment variables to determine the encoding to be used
  for input and output. For example, absent other locale settings, setting the environment
  variable LANG=en_US.UTF-8 will cause pbcopy and pbpaste to use UTF-8 for input and output.
  If an encoding cannot be determined from the locale, the standard C encoding will be used.
  Use of UTF-8 is recommended. Note that by default the Terminal application uses the UTF-8
  encoding and automatically sets the appropriate locale environment variable.
```

```
~
~ pbpaste
this is an example of Luke's clipboard buffer.
~
```

[/usr/bin/pbcopy](#)

Evasion & Understanding your tools

- Executing any action has an impact on the system.
 - It might cause a popup to the user
 - It might leave behind logs
 - It might be outright blocked by security tooling
- **Blue Teams** use logs to write rules that detect malicious activity, which in this case is our malware :(

The platform binary `pbpaste` was executed.

With a parent process of `Python`

With `Terminal` as it's responsible app.

```
1 {
2   "hostname": "lukes-macbook-pro.local",
3   "pid": 58987,
4   "event_type": "ProcessExec",
5   // What was executed?
6   "binary_path": "/usr/bin/pbpaste",
7   "signing_id": "com.apple.pbpaste",
8   "platform_binary": true,
9   // Parent process
10  "parent_binary_path":
11    "/opt/homebrew/Cellar/python@3.13/.../Contents/MacOS/Python",
12  // Responsible process
13  "responsible_binary_path":
14    "/System/Applications/.../Contents/MacOS/Terminal",
15 }
```

Objective-C APIs are a lot stealthier



```
1 #import <Foundation/Foundation.h>
2 #import <AppKit/AppKit.h>
3
4 int main(int argc, const char * argv[]) {
5     @autoreleasepool {
6         NSPasteboard* pasteboard = [NSPasteboard generalPasteboard];
7
8         NSString* pasteboardContents = [pasteboard stringForType:@"public.utf8-plain-
text"];
9
10        NSLog(@"%@", pasteboardContents);
11
12    }
13    return 0;
14 }
15
```



```
1 #import <Foundation/Foundation.h>
2 #import <AppKit/AppKit.h>
3
4 int main(int argc, const char * argv[] {
5     @autoreleasepool {
6         NSPasteboard* pasteboard = [NSPasteboard generalPasteboard];
7
8         NSString* pasteboardContents = [pasteboard stringForType:@"public.utf8-plain-
text"];
9
10        NSLog(@"%@", pasteboardContents);
11    }
12    return 0;
13 }
14 }
15 }
```

ronaldoussoren / pyobjc

Code Issues 105 Pull requests 1 Actions Projects Security Insights

pyobjc Public Watch 12 Fork 60 Star 687

master 8 Branches 76 Tags

ronaldoussoren Drop debug print ca857e0 · 3 weeks ago 5,678 Commits

- .github/ISSUE_TEMPLATE
- development-support
- docs
- pyobjc-core
- pyobjc-framework-AVFoundation
- pyobjc-framework-AVKit
- pyobjc-framework-AVRouting
- pyobjc-framework-Accessibility
- pyobjc-framework-Accounts
- pyobjc-framework-AdServices
- pyobjc-framework-AdSupport
- pyobjc-framework-AddressBook
- pyobjc-framework-AppTrackingTransparency
- pyobjc-framework-AppleScriptKit
- pyobjc-framework-AppleScriptObjC
- pyobjc-framework-ApplicationServices
- pyobjc-framework-AudioVideoBridging
- pyobjc-framework-AuthenticationServices
- pyobjc-framework-AutomaticAssessmentCo...

About

The Python <-> Objective-C Bridge with bindings for macOS frameworks

[pyobjc.readthedocs.io](#)

mrJean1 / PyCocoa

Code Issues 4 Pull requests Actions Projects Wiki Security Insights

PyCocoa Public Watch 2 Fork 2 Star 20

master 1 Branch 0 Tags

mrJean1 macOS 15.4 Sequoia 2da2ea1 · 3 months ago 82 Commits

- dist macOS 15.4 Sequoia 3 months ago
- docs macOS 15.4 Sequoia 3 months ago
- pycocoa macOS 15.4 Sequoia 3 months ago
- test macOS 15.4 Sequoia 3 months ago
- MANIFEST.in Renamed macOS.py to cocoa.py and moved to the py... 7 years ago
- README.rst macOS 15.4 Sequoia 3 months ago
- cocoaavc-18.6.2-Python-2.7.14-adjust.jpg Revamped NSMain globals, new functions isAlias, isLink. 7 years ago
- cocoaavc-18.6.2-Python-3.6.5.jpg Revamped NSMain globals, new functions isAlias, isLink. 7 years ago
- cocoaavc-snapshotA-TableWindow.pdf More menu enhancements. 7 years ago
- cocoaavc-snapshotB-TableWindow.pdf More menu enhancements. 7 years ago
- cocoaavc-snapshotC-TableWindow.pdf More menu enhancements. 7 years ago
- cocoaavc-snapshotD-MediaWindow.png More menu enhancements. 7 years ago
- docs.html Revamped NSMain globals, new functions isAlias, isLink. 7 years ago
- setup.py macOS 15.4 Sequoia 3 months ago

README

PyCocoa

A basic, ctypes-based Python binding to the [macOS](#) Objective-C Cocoa runtime and several other [macOS](#)

About

Basic, partial Python binding to Objective-C Cocoa

- Readme
- Activity
- 20 stars
- 2 watching
- 2 forks
- Report repository

Releases

No releases published

Packages

No packages published

Deployments 82

github-pages 3 months ago

+ 81 deployments

Languages

- Python 99.9%
- HTML 0.1%

Objective-C

Peeking behind the curtain

Objective-C Runtime

- What makes Objective-C different from C? *The Objective-C Runtime*.
- It's all of the internals that power the language, implemented as a library, rather than being fully implemented at the compiler level (like traditional C/C++).
- It allows you to dynamically, at runtime, change things that would otherwise be set by the compiler, like adding methods to classes.
- The core runtime functions are exposed to us at runtime.

The screenshot shows the Apple Developer website's documentation page for the Objective-C Runtime. The top navigation bar includes 'Developer', 'News', 'Discover', 'Design', 'Develop', 'Distribute', 'Support', and 'Account'. The left sidebar is titled 'Documentation' and lists various categories: 'All Technologies', 'Objective-C Runtime', 'Classes', 'NSObject', 'Protocols', 'Reference', and 'Objective-C Runtime'. Under 'Objective-C Runtime', there is a list of methods such as `class_getName`, `class_getSuperclass`, `class_setSuperclass` (marked as deprecated), `class_isMetaClass`, `class_getInstanceSize`, `class_getInstanceVariable`, `class_getClassVariable`, `class_addIvar`, `class_copyIvarList`, `class_getIvarLayout`, `class_setIvarLayout`, `class_getWeakIvarLayout`, `class_setWeakIvarLayout`, `class_getProperty`, `class_copyPropertyList`, `class_addMethod`, `class_getInstanceMethod`, `class_getClassMethod`, `class_copyMethodList`, `class_replaceMethod`, `class_getMethodImplementation`, `class_getMethodImplementation_stret`, `class_respondToSelector`, `class_addProtocol`, `class_addProperty`, and `class_replaceProperty`. The main content area has a purple header with the title 'Objective-C Runtime' and a subtitle 'API Collection'. Below this is an 'Overview' section with text explaining the runtime library's role and its implementation on macOS. A callout box notes that 'All char * in the runtime API should be considered to have UTF-8 encoding.' and a note below it states 'Deprecated' below means 'deprecated in OS X version 10.5 for 32-bit code, and disallowed for 64-bit code.' The page also includes sections for 'Who Should Read This Document' and 'Topics', with 'Working with Classes' listed as a topic.

<https://developer.apple.com/documentation/objectivec/objective-c-runtime?language=objc>

```

_main:
  sub    sp, sp, #0x50
  stp   fp, lr, [sp, #0x40]
  add   fp, sp, #0x40
  mov   w8, #0x0
  stur  w8, [fp, var_1C]
  stur  wzr, [fp, var_4]
  stur  w0, [fp, var_8]
  stur  x1, [fp, var_10]
  bl    imp__stubs__objc_autoreleasePoolPush ; objc_autoreleasePoolPush
  str   x0, [sp, #0x40 + var_28]
  adrp  x8, #0x100008000
  add   x0, x8, #0x108 ; argument "class" for method imp__stubs__objc_alloc, _OBJC_CLASS_$_Toaster
  bl    imp__stubs__objc_alloc ; objc_alloc
  ldr   x1, [sp, #0x40 + var_38]
  adrp  x2, #0x100004000 ; 0x100004070@PAGE
  add   x2, x2, #0x70 ; 0x100004070@PAGEOFF, @"red"
  bl    _objc_msgSend$initWithColour: ; _objc_msgSend$initWithColour:
  ldr   x1, [sp, #0x40 + var_38]
  sub   x8, fp, #0x18
  str   x8, [sp, #0x40 + var_30]
  stur  x0, [fp, var_18]
  ldur  x0, [fp, var_18]
  adrp  x2, #0x100004000 ; 0x100004090@PAGE
  add   x2, x2, #0x90 ; 0x100004090@PAGEOFF, @"Bagel"
  adrp  x3, #0x100004000 ; 0x1000040c8@PAGE
  add   x3, x3, #0xc8 ; 0x1000040c8@PAGEOFF, 0x1000040c8
  adrp  x4, #0x100004000 ; 0x1000040e0@PAGE
  add   x4, x4, #0xe0 ; 0x1000040e0@PAGEOFF, 0x1000040e0
  bl    _objc_msgSend$makeToastWithBread:forNumberOfSeconds:onPowerSetting: ; _objc_msgSend$makeToastWithBread:forNumberOfSeconds:onPowerSetting:
  ldr   x0, [sp, #0x40 + var_30] ; argument "addr" for method imp__stubs__objc_storeStrong
  mov   x1, #0x0 ; argument "value" for method imp__stubs__objc_storeStrong
  bl    imp__stubs__objc_storeStrong ; objc_storeStrong
  ldr   x0, [sp, #0x40 + var_28] ; argument "pool" for method imp__stubs__objc_autoreleasePoolPop
  bl    imp__stubs__objc_autoreleasePoolPop ; objc_autoreleasePoolPop
  ldur  w0, [fp, var_1C]
  ldp   fp, lr, [sp, #0x40]
  add   sp, sp, #0x50
  ret

; endp

```

Message Sending

- When you call a method on an object `[mystring length]`. The runtime works out exactly what function code should be called.
- The compiler can't know for certain where the function code will live for a String's `length` method, because it may have changed.
- Instead we have the concept of message sending.

```
○ ○ ○
1 #import <Foundation/Foundation.h>
2 #import <AppKit/AppKit.h>
3
4 int main(int argc, const char * argv[]) {
5     @autoreleasepool {
6         NSPasteboard* pasteboard = [NSPasteboard generalPasteboard];
7
8         NSString* pasteboardContents = [pasteboard stringForType:@"public.utf8-plain-
text"];
9
10        NSLog(@"%@", pasteboardContents);
11    }
```

Message Sending

```
@selector(stringForType:)
```

```
Arg1: "public.utf8-plain-text"
```

(1) What are we sending?

(3) How are we sending it?
objc_msgSend



(2) Where are we sending it?

(3) How are we sending it?

(2) Where are we sending it?

(1) What are we sending?

Objective-C Runtime / objc_msgSend

Function

objc_msgSend

Sends a message with a simple return value to an instance of a class.

iOS 2.0+ | iPadOS 2.0+ | macOS 10.0+ | tvOS 9.0+ | visionOS 1.0+ | watchOS 2.0+

```
void objc_msgSend(void);
```

Parameters

self
A pointer that points to the instance of the class that is to receive the message.

op
The selector of the method that handles the message.

...
A variable argument list containing the arguments to the method.

Return Value

The return value of the method.

Discussion

When it encounters a method call, the compiler generates a call to one of the functions `objc_msgSend`, `objc_msgSend_stret`, `objc_msgSendSuper`, or `objc_msgSendSuper_stret`. Messages sent to an object's superclass (using the `super` keyword) are sent using `objc_msgSendSuper`; other messages are sent using `objc_msgSend`. Methods that have data structures as return values are sent using `objc_msgSendSuper_stret` and `objc_msgSend_stret`.

(2) Where are we sending it?

(3) How are we sending it?

```
1 #import <Foundation/Foundation.h>
2 #import <objc/runtime.h>
3 #import <objc/message.h>
4
5 int main(int argc, const char * argv[]) {
6     @autoreleasepool {
7         SEL selector = @selector(makeToastWithBread:forNumberOfSeconds:onPowerSetting:);
8         id receiver = [Toaster toaster];
9
10        void (*objc_msgSendTyped)(id, SEL, NSString*, NSNumber*, NSNumber*) = (void (*)(
11        id, SEL, NSString*, NSNumber*, NSNumber*))objc_msgSend;
12
13        objc_msgSendTyped(receiver, selector, @"Bagel", @(30), @(5));
14    }
15 }
16
```

(1) What are we sending?

Missing piece of the puzzle



import ctypes

Python foreign function library, used to load libraries (.dll, .so, .dylib) and execute functions.

- Data types (c_char_p, c_void_p, etc...)
- DLL Load (dlopen)
- Function Calling

ctypes is a default python library!

```
○ ○ ○  
1 import ctypes  
2  
3 # Load the C standard library  
4 libc = ctypes.CDLL("libc.dylib")  
5  
6 # Set the argument and return types  
7 libc.strlen.argtypes = [ctypes.c_char_p]  
8 libc.strlen.restype = ctypes.c_size_t  
9  
10 # Call the strlen function  
11 length = libc.strlen(b"Hello, world!")  
12  
13 print(f'The length is {length}.') # 13
```

Missing piece of the puzzle



Putting it all together...

```
○ ○ ○  
1 _libObjC = ctypes.CDLL('/usr/lib/libobjc.dylib')  
2  
3 nsStringClass = _libObjC.objc_getClass(b"NSString")  
4 allocSelector = _libObjC.sel_registerName(b"alloc")  
5  
6 _libObjC.objc_msgSend.restype = getReturnType(nsStringClass, allocSelector)  
7 _libObjC.objc_msgSend.argtypes = getArgTypes(nsStringClass, allocSelector)  
8  
9 # NSString* allocatedString = [NSString alloc];  
10 allocatedString = _libObjC.objc_msgSend(nsStringClass, allocSelector)
```



(3) How are we sending it?

(2) Where are we sending it?

(1) What are we sending?

Making it usable

- That's a lot of code, to do not very much – but after abstracting all the internals away and wrapping in a nice interface...
- `NSObject(name)` function to build a base object.
- Send messages by calling methods on the base object.
 - Python doesn't support `:` in variable names, so we replace `_`.

○ ○ ○

```
1 _load_lib('/System/Library/Frameworks/example.framework/example')
2
3 myString = NSObject("NSString").alloc().initWithUTF8String_(b"helloWorld")
4
5 length = myString.length()
```





Sign in to GitHub - GitHub

github.com/login

Sign in to GitHub

Username or email address

Password [Forgot password?](#)

[Sign in](#)

or

[Continue with Google](#)

New to GitHub? [Create an account](#)

[Sign in with a passkey](#)

Terms Privacy Docs Contact GitHub Support Manage cookies Do not share my personal information

Choose Your Own Story

- 1 /w
- 2 See the l
- 3 /f
- 4

Choose Your Own Story

- README
- License
- App

Passwords - KeePassXC

Title	Username	URL	Notes	Modified
github.com				24/07/2025 10:07

- Copy Username ⌘B
- Copy Password ⌘C
- Copy URL ⌘U
- Copy Attribute
- TOTP
- Tags
- Perform Auto-Type
- Import Passkey
- Edit Entry... ⌘E
- Expire Entry...
- Clone Entry... ⌘K
- Delete Entry... ⌘D
- New Entry... ⌘N
- Open URL ⌘⌘U
- Download Favicon ⌘⌘D

1 Entry

Favourites

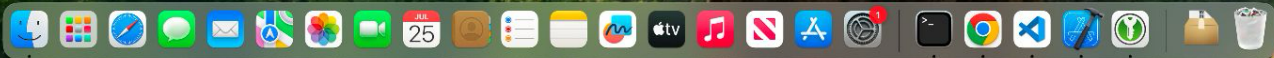
- Recents
- Applications
- Desktop
- Documents
- Downloads

Locations

- iCloud Drive
- My Shared Files
- Parallels Tools

Tags

- Red
- Orange
- Yellow

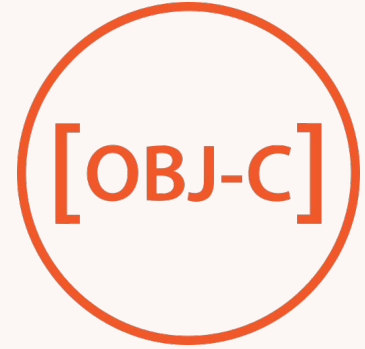


○ ○ ○




```
1 #import <Foundation/Foundation.h>
2 #import <AppKit/AppKit.h>
3
4 int main(int argc, const char * argv[]) {
5     @autoreleasepool {
6         NSPasteboard* pasteboard = [NSPasteboard generalPasteboard];
7
8         NSString* pasteboardContents = [pasteboard stringForType:@"public.utf8-plain-
text"];
9
10        NSLog(@"%@", pasteboardContents);
11    }
12    return 0;
13 }
14 }
15
```

○ ○ ○

```
1 _load_lib('/System/Library/Frameworks/AppKit.framework/AppKit')
2
3 pasteboard = NSObject("NSPasteboard").generalPasteboard()
4
5 pasteboardStringType =
  NSObject("NSString").stringWithUTF8String_(b"public.utf8-plain-text")
6
7 pasteboardContents = pasteboard.stringForType_(pasteboardStringType)
8
9 print(pasteboardContents.UTF8String())
```



Other Applications

- This can be used for anything that requires system or Objective-C APIs.
 - Keylogging? 
 - Persistence? 
 - Bridging to yet another language? 
 - Python -> C -> Objective-C -> JavaScript?

Conclusion

- This isn't a contrived example, this is an insight into what Red Team-flavoured security research can look like.
 - Will be released open-source into Medusa in a couple of weeks. 🚀
- You learn a lot more from writing the code yourself, rather than using off the shelf components.
- On macOS, system API introspection is hard/impossible compared to other platforms.
 - Less visibility == less burnt shells. 👍

Tools / Shoutouts

- Mythic, open-source C2 by Cody Thomas (@its_a_feature)
<https://github.com/its-a-feature/Mythic>
- Medusa, Python implant by Alfie Champion (@ajpc500)
<https://github.com/MythicAgents/Medusa>
- Hopper, decompiler by Cryptic Apps
<https://www.hopperapp.com/index.html>
- https://theevilbit.github.io/posts/getting_started_in_macos_security/