XUnprotect: Reverse Engineering macOS XProtect Remediator

Koh M. Nakagawa (@tsunek0h)

FFRI Security, Inc.

NSUserFullName()

- Koh M. Nakagawa (@tsunek0h)
- Security researcher at FFRI Security, Inc.
- Mainly focusing on Apple product security
- Gave talks at Black Hat and CODE BLUE

FFRI

black hat®

CODE BLUE
2023@TOKYO

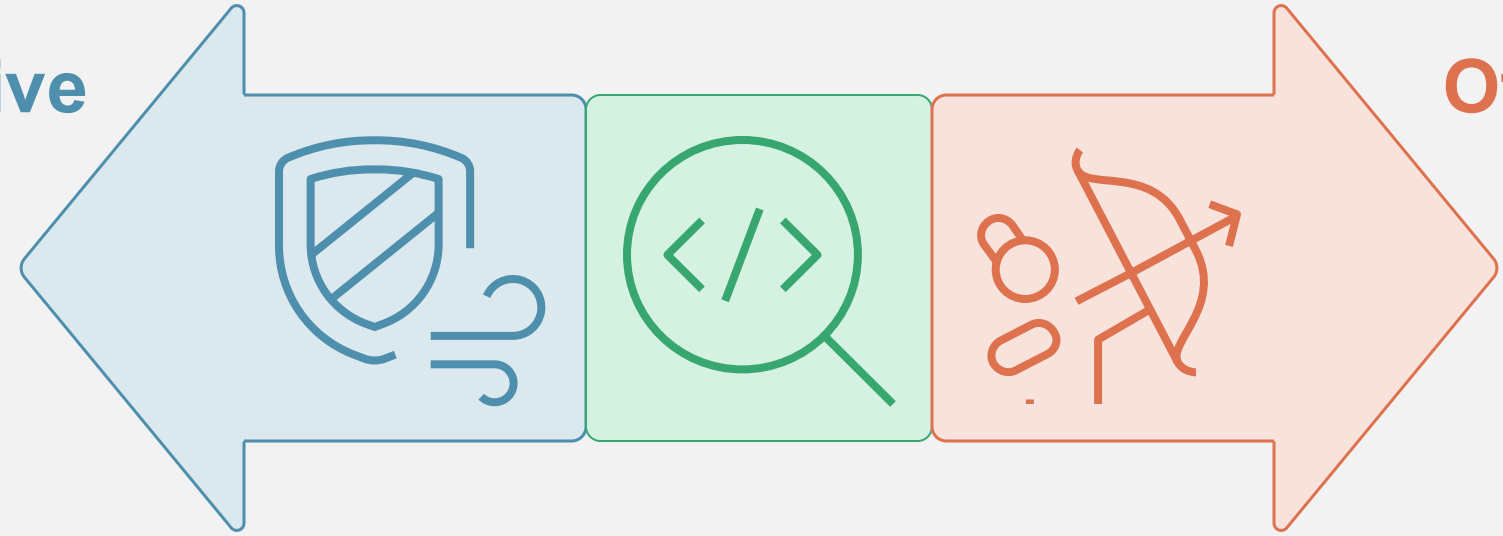
About This Presentation

- **This presentation covers:**
  - Technical deep dive into XProtect Remediator (XPR)
    - How XPR's detection logic works
    - Malware removed (or 'remediated') by each scanner
    - Provenance Sandbox (which XPR utilizes for identifying the source of files being remediated)
- **This presentation does not cover:**
  - Evaluation of XPR
    - Such as effectiveness as a macOS security product
  - Traditional XProtect
    - For this topic, see Stuart Ashenbrenner's excellent talk at MDOYVR23
    - https://youtu.be/43BIK-e7FBE

What You'll Gain from This Talk?

Deep understanding of XPR

Defensive



Offensive

For Blue Teamers:

Learn XPR's detection/remediation capabilities & Apple-exclusive threat intel

For Red Teamers:

Learn TCC & Provenance Sandbox bypass

Outline

1. **Introduction**
2. Tooling
3. RE results
4. Vulnerability Research
5. 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]

…

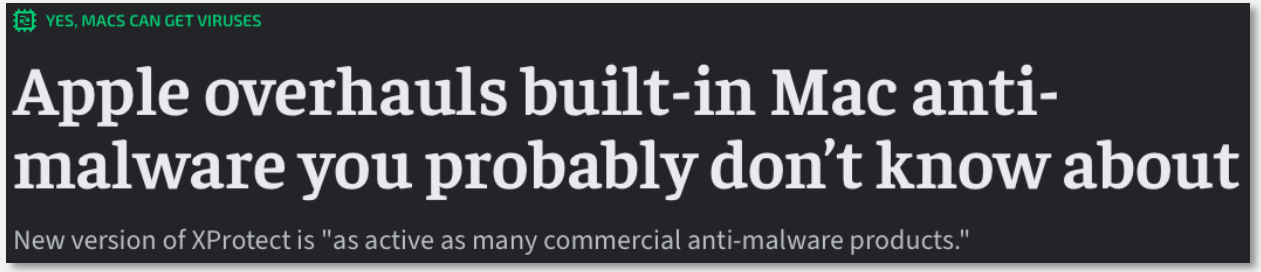
XProtect[Remediator] acts to remediate malware that has managed to successfully execute.

- “Apple Platform Security” by Apple

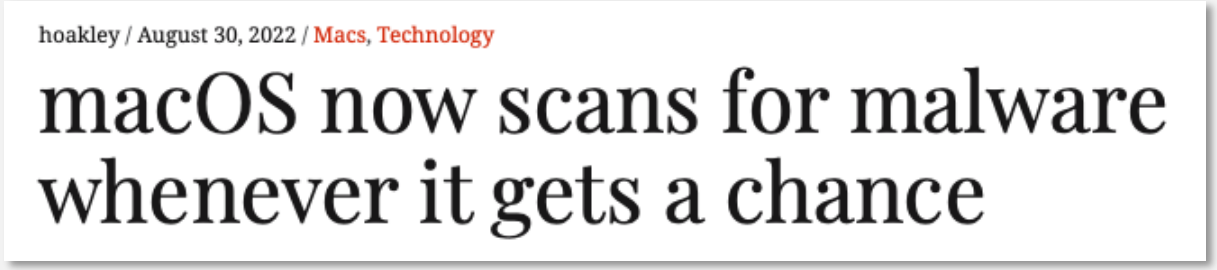
https://help.apple.com/pdf/security/en_US/apple-platform-security-guide.pdf

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



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



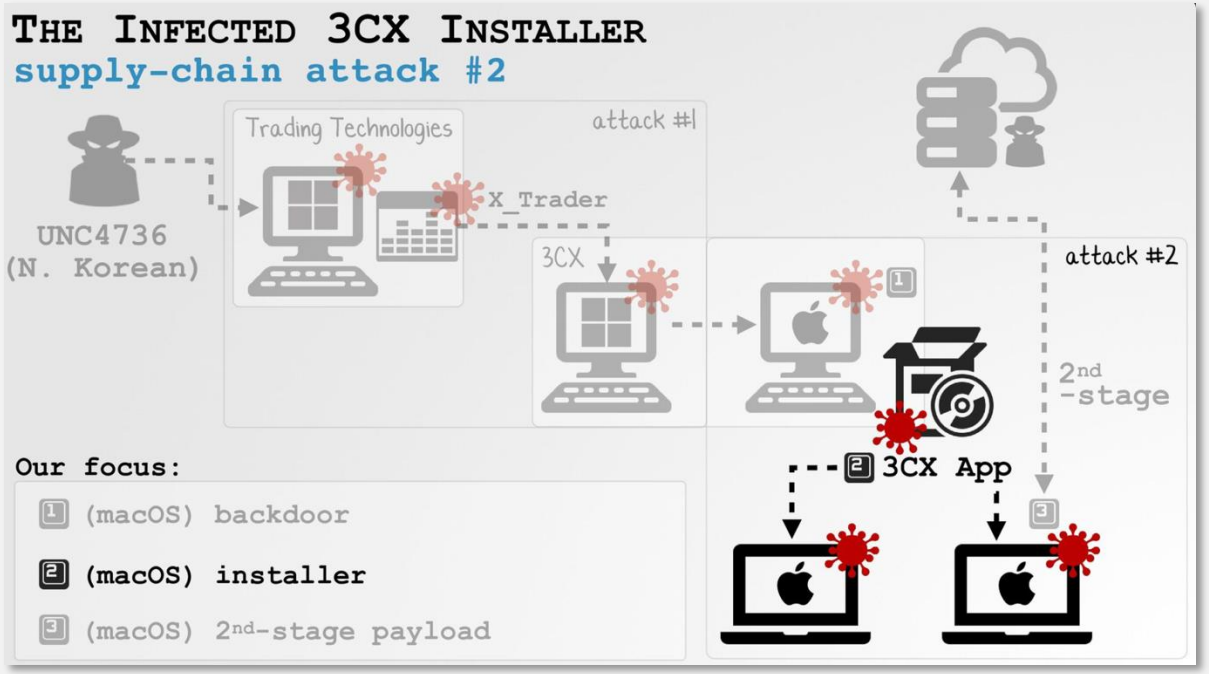
https://eclecticlight.co/2022/08/30/macos-now-scans-for-malware-whenever-it-gets-a-chance/

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

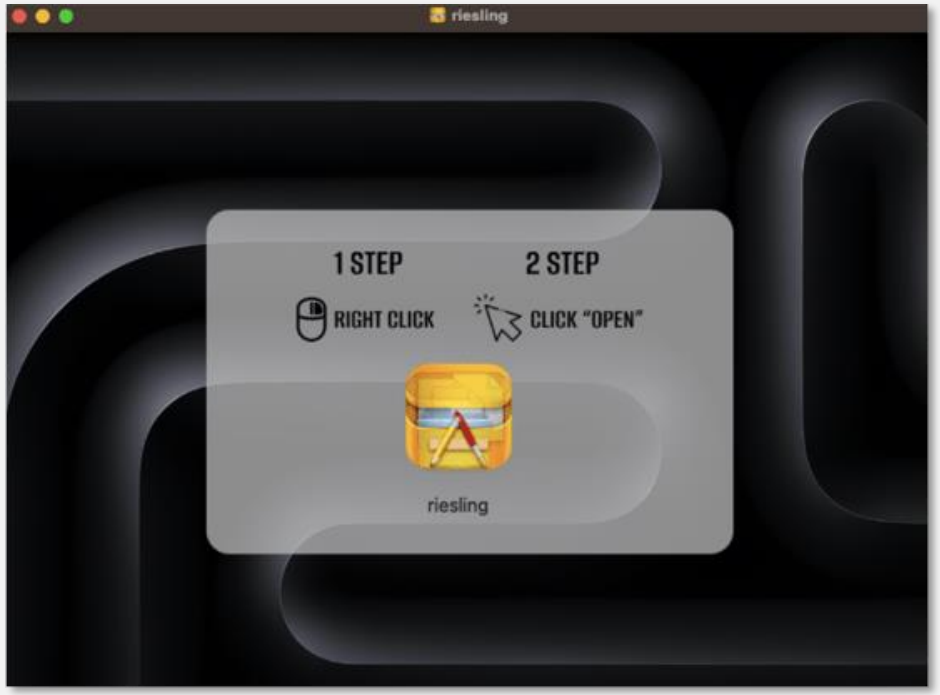
Each scanner targets a specific malware family (e.g., XProtectRemediatorAdload is a scanner for well-known Adload adware)

Why Is Remediation Needed?

- Some malware samples bypass the first and second layers of defense:
  - Through supply chain attacks (such as the 3CX supply chain attack)
  - By tricking users into disabling Gatekeeper through social engineering
- Apple needs a way to remove malware that slips through these defenses



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 offensive security perspective
  - XPR scanners are attractive exploitation targets due to their powerful entitlements
  - TCC bypass:
    - Some scanners have FDA entitlement (kTCCServiceSystemPolicyAllFiles)
    - Gergely Kalman's CVE-2024-40842 (TCC info leak)
  - User-to-root privilege escalation:
    - XPR scanners run with both root and user privileges

```
[Key] com.apple.private.tcc.allow
[Value]
        [Array]
                [String] kTCCServiceSystemPolicyAllFiles
                [String] kTCCServiceSystemPolicyAppBundles
```

Research Motivation

- From defensive security perspective
  - Several malware families targeted by XPR remain unknown
    - Howard Oakley, Alden Schmidt, and Phil Stokes have identified several targets
    - However, several remain unknown due to limited reverse engineering efforts
  - XPR's remediation logic is unclear
    - Is XPR's remediation simply scanning files with YARA and deleting any that match?

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/

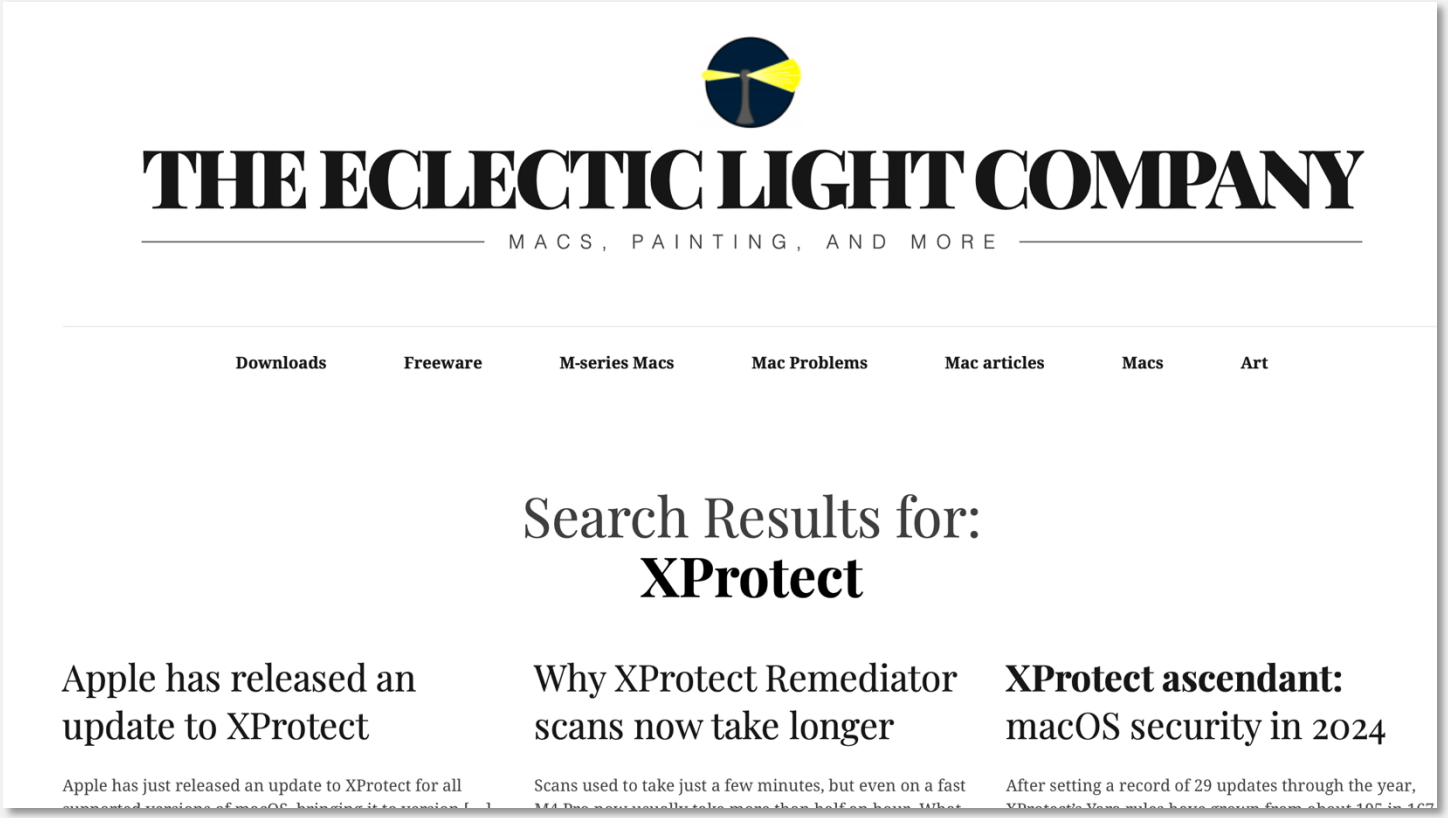
Research Target

- /Library/Apple/System/Library/CoreServices/XProtect.app
  - Contents/MacOS/XProtectRemediator*
  - Contents/MacOS/XProtect
  - Contents/XPCServices/XProtectPluginService.xpc
- These 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
```

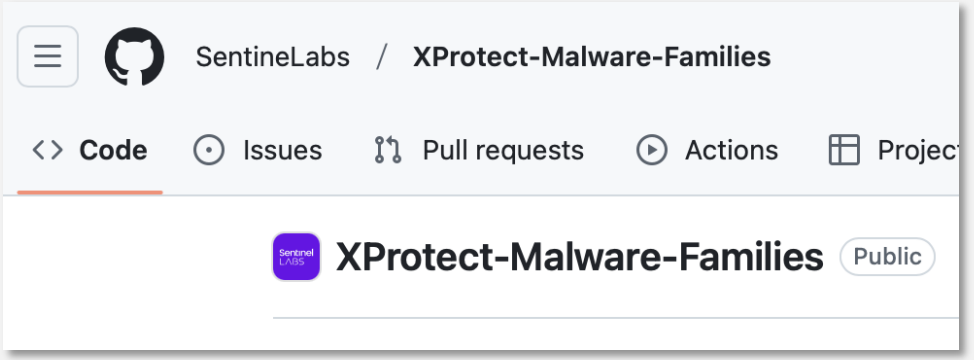
Swift-specific sections

Related Work



https://eclecticlight.co

https://alden.io/posts/secrets-of-xprotect/

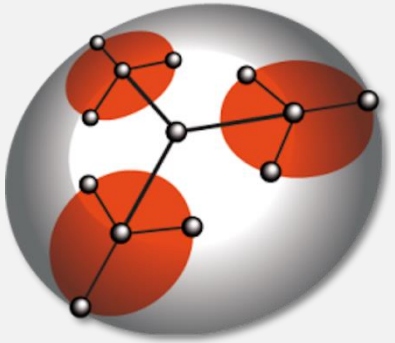


https://github.com/SentineLabs/XProtect-Malware-Families

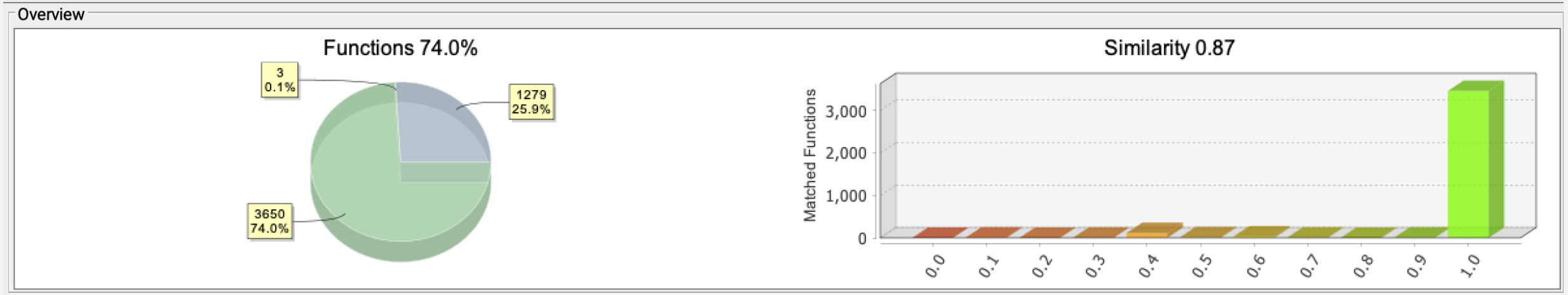
Outline

1. Introduction
2. **Tooling**
3. RE results
4. Vulnerability Research
5. Conclusion

Static Analysis



- Binary Ninja
- Stripped Swift Mach-O binaries
- Symbols are stripped, but some symbols can be recovered
  - BinDiff reveals many shared functions between XPR scanners and libXProtectPayloads.dylib
  - We can import symbols exported by libXProtectPayloads.dylib into XPR scanners



Challenges in RE of Stripped Swift Binaries

- Some key missing information 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 internal metadata for reflection
- This metadata includes type metadata accessor, type metadata, PWT
  - __TEXT.__swift5_protos, __TEXT.__swift5_types, __TEXT.__swift5_fieldmd, and more
  - “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 accessor and type metadata parsing
  - PWT analysis for structs and classes
  - Class method identification
  - Swift string analysis (immortal and large strings)
  - Visual representation of protocol conformance and class inheritance

Type Metadata Accessor Identification

```
100077ea0    int64_t sub_100077ea0() __pure

100077ea9    return &data_1000f09a0
```

```
100077ea0    int64_t type metadata accessor for YaraRuleVariable.VariableType() __pure

100077ea9    return &type metadata for YaraRuleVariable.VariableType
```

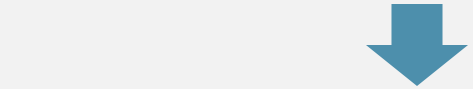
Symbols | type metadata for XPPluginAPI

Name

```
type metadata accessor for XPPluginAPI.YaraMatcher
type metadata accessor for XPPluginAPI.YaraMeta
type metadata accessor for XPPluginAPI.YaraError
type metadata accessor for XPPluginAPI.YaraScanResult
type metadata accessor for XPPluginAPI.YaraMetaType
type metadata accessor for XPPluginAPI.YaraRule
type metadata accessor for XPPluginAPI.YaraRuleVariable
type metadata for XPPluginAPI.YaraError
type metadata for XPPluginAPI.YaraMetaType
```

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

- LLDB Python Scripting Bridge
  - Branch tracing script (https://github.com/kohnakagawa/LLDB)
    - Swift binaries contain many indirect branches, such as function calls via VTable and PWT
    - Manually identifying branch targets in LLDB is time-consuming
    - This script automatically captures target addresses
    - Trace data is exported as JSON for import via my binja-missinglink plugin
    - http://github.com/FFRI/binja-missing-link

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
100099e83    void* r14 = *(ar
100099e87    int64_t r15 = *(arg1 +
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<swift
100099f19    // swift::TargetMetadata<swift::InProcess> const*)
100099f19    rax_15(rsp, rax)
100099f22    sub_10009bb90(&s)
100099f31    // BML_dst:
100099f31    // <libswiftC                                    es<swift
100099f31    // swift::Opa                                    ocess>
100099f31    // const*)
100099f31    (*(rax_1 + 0x
```

PWT information is also added for function calls via PWT

Resolved symbol information is also added

Dynamic Analysis – Custom LLDB Commands

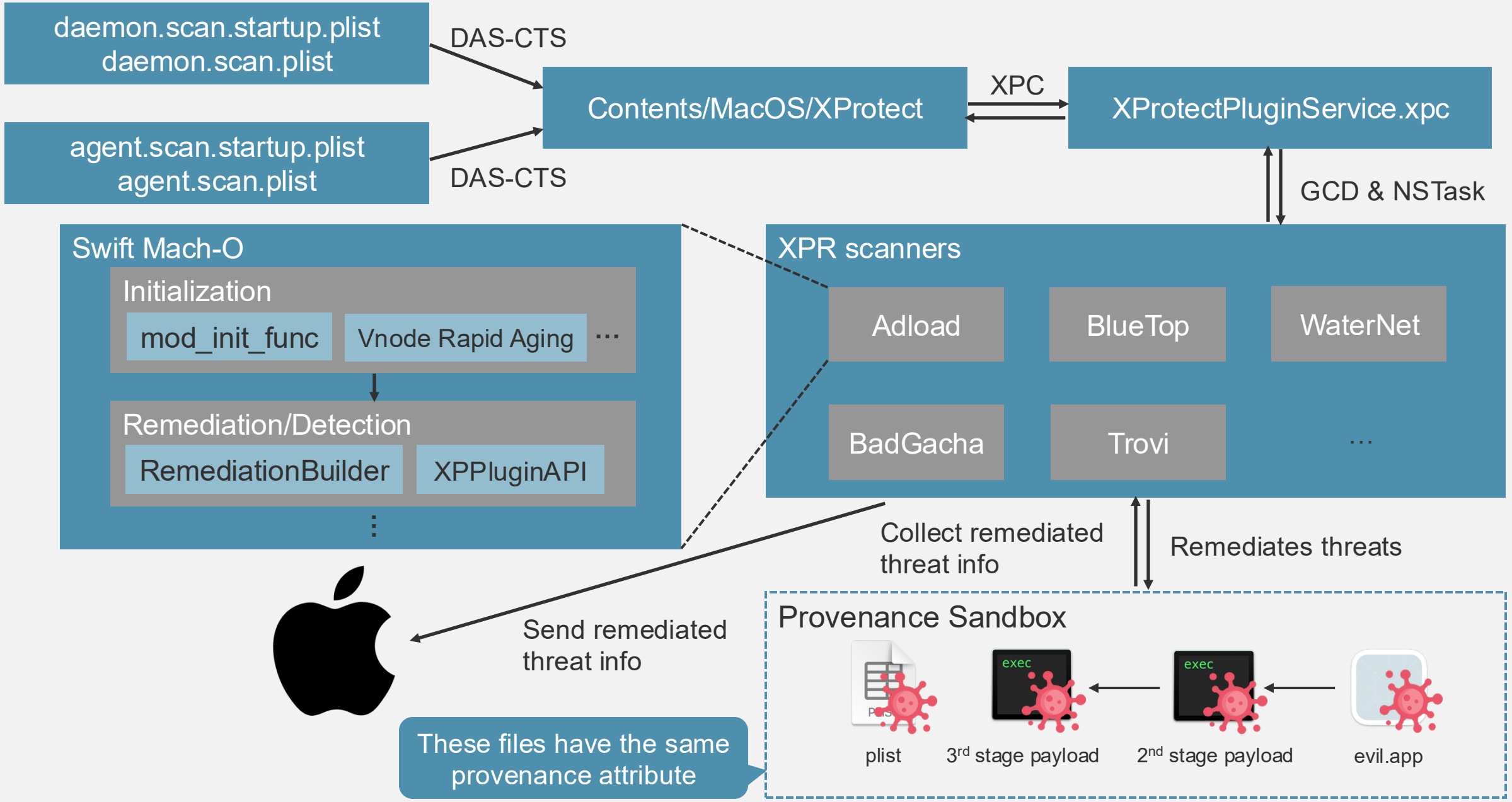
- Custom commands for dumping Swift Objects
  - Standard expr -O -l Swift -- <address> command does not work for complex Swift objects like existential containers and Swift arrays…
  - Created enhanced commands for dumping Swift objects utilizing Swift reflection

```
command regex p_boxed_array 's/(.+)/expr -D 1000 -l Swift -- protocol Empty {}; unsafeBitCast(%1, to: [any
Empty].self)/'
command regex po_boxed_array 's/(.+)/expr -O -l Swift -- protocol Empty {}; unsafeBitCast(%1, to: [any Empty].self)/'
command regex dump_boxed_array 's/(.+)/expr -l Swift -- protocol Empty {}; let $tmp = dump(unsafeBitCast(%1, to: [any
Empty].self))/'
command regex sdump 's/(.+)/dwim-print -l Swift -- dump(unsafeBitCast(%1, to: AnyObject.self))/'
```

Outline

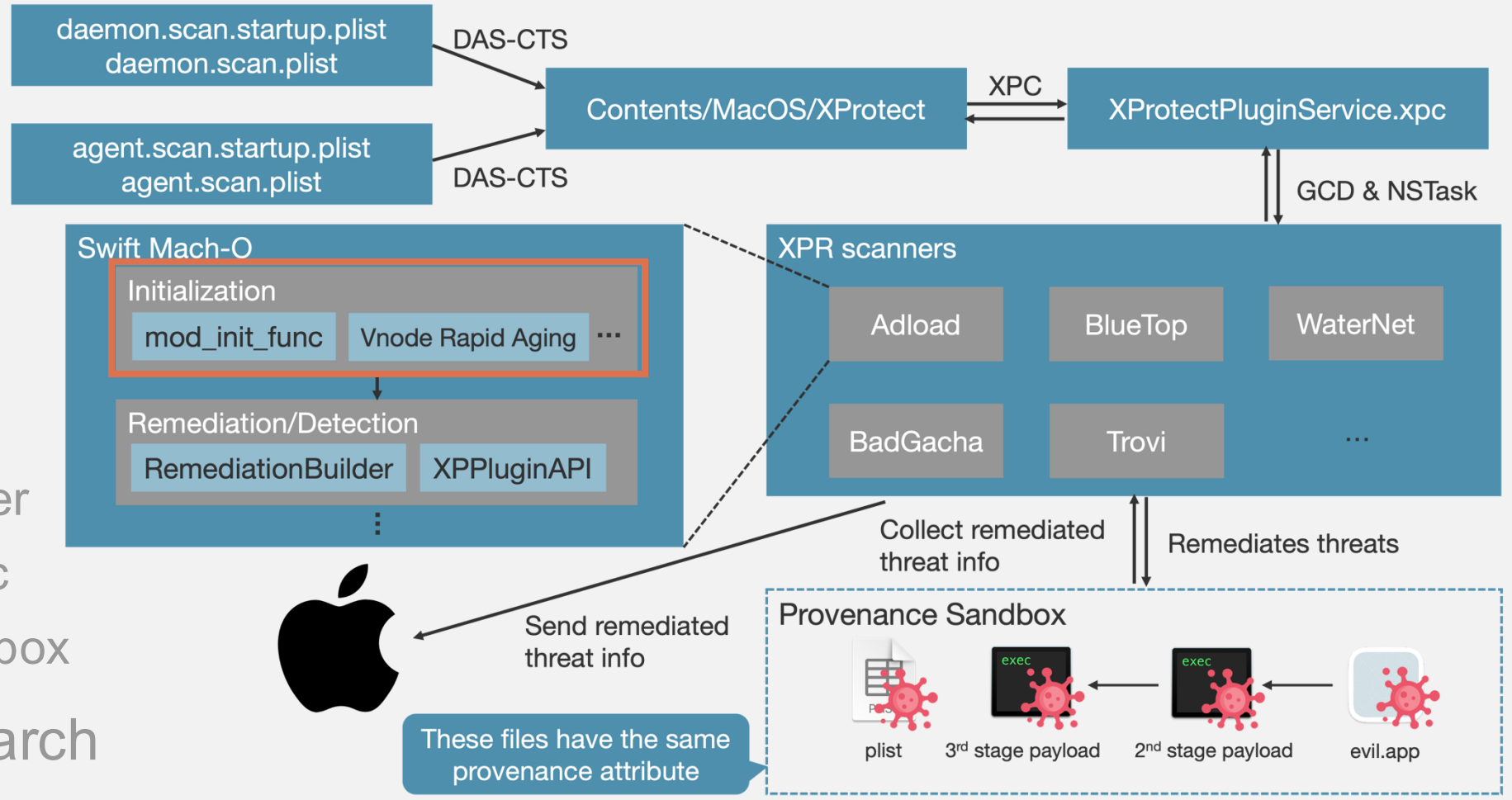
1. Introduction
2. Tooling
3. **RE results**
    1. **Overview**
    2. Initialization
    3. RemediationBuilder
    4. Remediation Logic
    5. Provenance Sandbox
4. Vulnerability Research
5. Conclusion

Flow of “Remediation”



Outline

1. Introduction
2. Tooling
3. **RE results**
   1. Overview
   2. **Initialization**
   3. RemediationBuilder
   4. Remediation Logic
   5. Provenance Sandbox
4. Vulnerability Research
5. Conclusion



mod_init_func_0

- mod_init_func_0 (function with constructor attr, executed before _start)
  - Sensitive strings (YARA, file paths, etc.) for remediation are encrypted with XOR cipher
  - These strings are decrypted before _start
  - Pointers to decrypted strings are stored in __DATA.__common

```
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

```
04e23817983f1c0e9290ce7f90e6c9e75bf45190
99c31f166d1f1654a1b7dd1a6bec3b935022a020
```

```
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/
```

```
.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
```

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

```
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
```

Program Entry Point

- A plugin class is instantiated
  - Each XPR scanner typically defines one plugin class (such as AdloadPlugin)
- XPAPIHelpers is instantiated and passed to the plugin main function
  - The plugin entry point is XProtectPluginProtocol.main(api: XPPluginAPI.XPAPIHelpersProtocol)

```
let adloadPlugin = AdloadPlugin("ADLOAD", 6, XPPluginStatusCollator())
adloadPlugin.main(api: XPAPIHelpers.shared)
```

Plugin class is instantiated

XPAPIHelpers is instantiated and passed to the plugin main

XPAPIHelpers

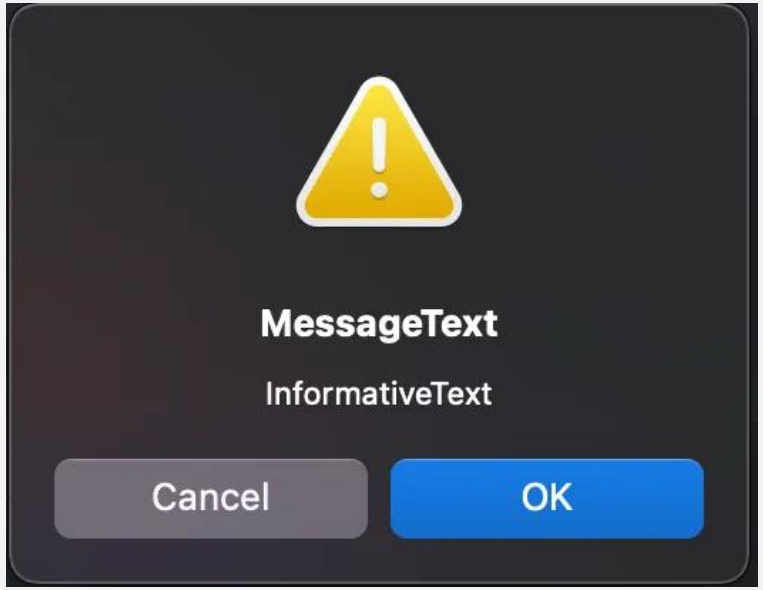
```
class XPAPIHelpers {
    let logger: XPLogger
    var pluginService: XProtectPluginDispatchProtocol
    let codeSignature: XProtectPluginCodeSignatureAPIProtocol
    let file: XProtectPluginAPIPath
    var launchd: XProtectPluginLaunchdAPIProtocol
    var launchServices: XPLaunchServicesProtocol
    var yara: XProtectPluginAPIYaraProtocol
    let process: XProtectPluginProcessAPIProtocol
    var event: XProtectPluginAPIEventsProtocol
    let networkSettings: XProtectPluginAPINetworkSettingsProtocol
    var keychain: XProtectPluginKeychainAPIProtocol
    var plugin: XProtectPluginProtocol ?
    var pipeline: _OBJC_CLASS_$_CPProfile ?
    var connection: VerifiableXPCConnectionProtocol
    var configProfiles: XProtectConfigProfilesAPIProtocol
    var lazy alertGUI: XPAlertGUIProtocol ?
    var memory: XPProcessMemoryAPI
    var lazy behavioralEvents: XPEventDatabaseAPIProtocol ??
}
```

XPAPIHelpers: Interesting Property

- var lazy alertGUI: XPAlertGUIProtocol
  - Contains methods that display an alert dialog to users using NSAlert
  - Current XPR silently remediates threats without notifying users
  - I have not seen any XPR scanners using this property during my research
  - XPR may introduce user notifications for remediation events in the future?

```
protocol conformance XPAlertWindow : XPAlertGUIProtocol

class XPAlertWindow {
    let alert: NSAlert
    var logger: XPLogger
}
```



XPR Plugin Main

- XProtectPluginProtocol.main(api: XPPluginAPI.XPAPIHelpersProtocol) -> XProtecPluginCompletionStatus
  - Instantiating XPLogger class
  - Recording performance data using os_signpost
  - Unsetting the MAGIC environment variable (fix for CVE-2024-40842)
  - Verifying XProtectPluginService by checking its com.apple.private.xprotect.trustedpluginservice entitlement
  - Enabling Vnode Rapid Aging
- After enabling Vnode Rapid Aging, the remediation begins

Vnode Rapid Aging

- Vnode Rapid Aging is a feature that suppresses atime updates
  - Updates are suppressed on a per-process basis
  - Can be enabled via sysctl (no entitlement required)
  - Appears to be intended for performance improvement and preservation for forensic investigation
  - Disabled after remediation

```c
void enable_vnode_rapid_aging(int enabled) {
    int mib[] = {CTL_KERN, KERN_RAGEVNODE};
    if (sysctl(mib, 2, NULL, NULL, &enabled, sizeof(enabled)) != 0) {
        perror("Failed to call sysctl");
    } else {
        puts("Success: Vnode Rapid Aging is enabled");
    }
}
```

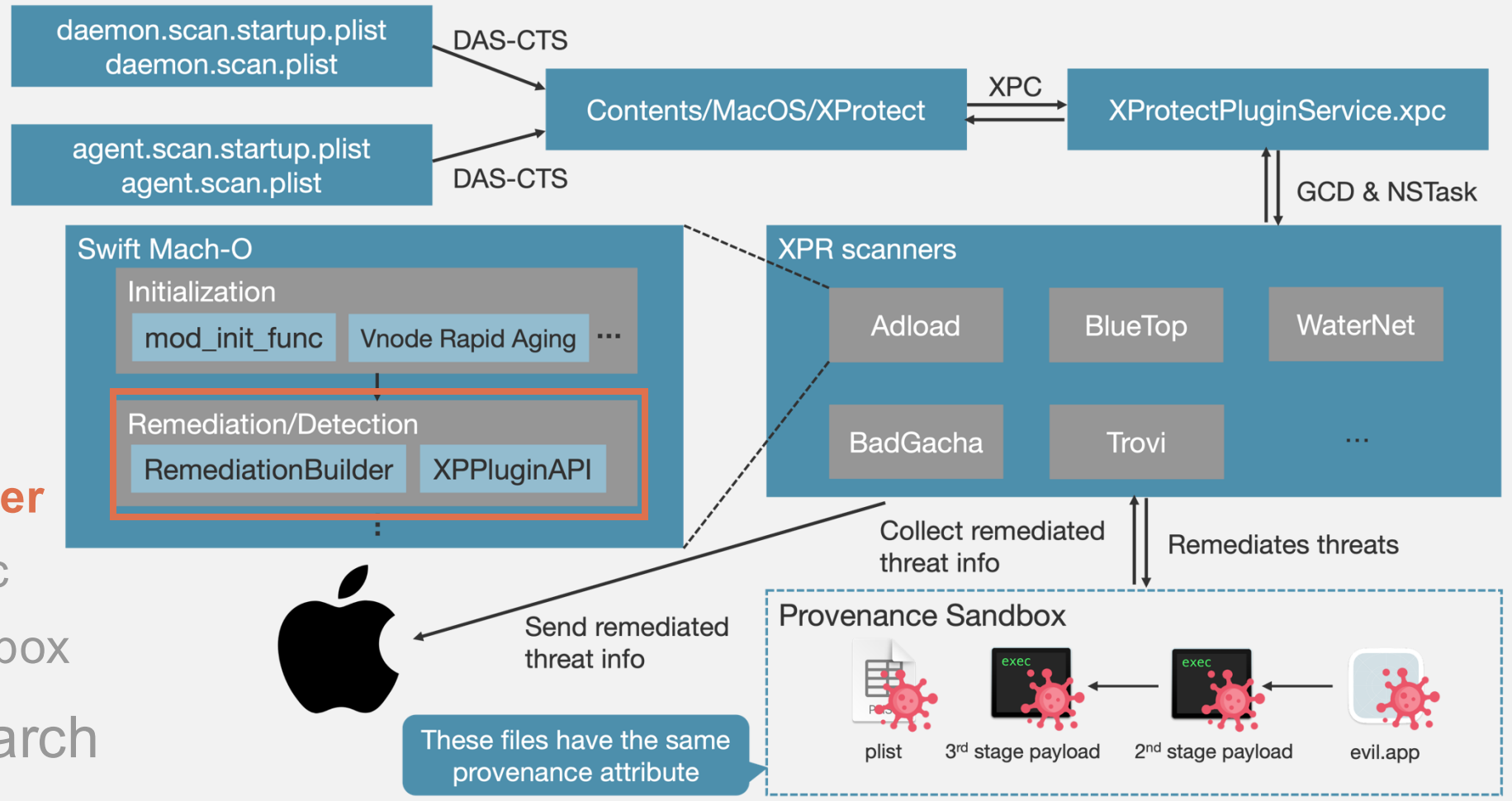
According to the Kernel sources, there's something called "rapid aging" that might be relevant. Documentation is sparse so I don't know its intended use, but it looks like something you can set per-process that will prevent access times from being set.

- "WrMeta" by darwin-dev@googlegroups.com

https://groups.google.com/g/darwin-dev/c/7F6uth1rhKw/m/SJQ3zWxeIgEJ

Outline

1. Introduction
2. Tooling
3. **RE results**
    1. Overview
    2. Initialization
    3. **RemediationBuilder**
    4. Remediation Logic
    5. Provenance Sandbox
4. Vulnerability Research
5. 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 {
    for file in enumerateFiles("/Library/Application Support", 5) {
        if file.size <= 2 * 1024 * 1024 {
            if file.isMacho() {
                if !file.isNotarized() {
                    if yaraMatcherRoot.match(file) {
                        remediate(file)
                    }
                }
            }
        }
    }
}
```

For each file under ~/Library/Application Support

File size is 2 MiB or less

File format is Mach-O

Not notarized

Matches YARA rule

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 {
    for file in enumerateFiles("/Library/Application Support", 5) {
        if file.size <= 2 * 1024 * 1024 {
            if file.isMacho() {
                if !file.isNotarized() {
                    if yaraMatcherRoot.match(file) {
                        remediate(file)
                    }
                }
            }
        }
    }
}
```

For each file under ~/Library/Application Support

File size is 2 MiB or less

File format is Mach-O

Not notarized

Matches YARA rule

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 {
    for file in enumerateFiles("/Library/Application Support", 5) {
        if file.size <= 2 * 1024 * 1024 {
            if file.isMacho() {
                if !file.isNotarized() {
                    if yaraMatcherRoot.match(file) {
                        remediate(file)
                    }
                }
            }
        }
    }
}
```

For each file under ~/Library/Application Support

File size is 2 MiB or less

File format is Mach-O

Not notarized

Matches YARA rule

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 {
    for file in enumerateFiles("/Library/Application Support", 5) {
        if file.size <= 2 * 1024 * 1024 {
            if file.isMacho() {
                if !file.isNotarized() {
                    if yaraMatcherRoot.match(file) {
                        remediate(file)
                    }
                }
            }
        }
    }
}
```

For each file under ~/Library/Application Support

File size is 2 MiB or less

File format is Mach-O

Not notarized

Matches YARA rule

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 {
    for file in enumerateFiles("/Library/Application Support", 5) {
        if file.size <= 2 * 1024 * 1024 {
            if file.isMacho() {
                if !file.isNotarized() {
                    if yaraMatcherRoot.match(file) {
                        remediate(file)
                    }
                }
            }
        }
    }
}
```

For each file under ~/Library/Application Support

File size is 2 MiB or less

File format is Mach-O

Not notarized

Matches YARA rule

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 {
    for file in enumerateFiles("/Library/Application Support", 5) {
        if file.size <= 2 * 1024 * 1024 {
            if file.isMacho() {
                if !file.isNotarized() {
                    if yaraMatcherRoot.match(file) {
                        remediate(file)
                    }
                }
            }
        }
    }
}
```

For each file under ~/Library/Application Support

File size is 2 MiB or less

File format is Mach-O

Not notarized

Matches YARA rule

Implementation for root

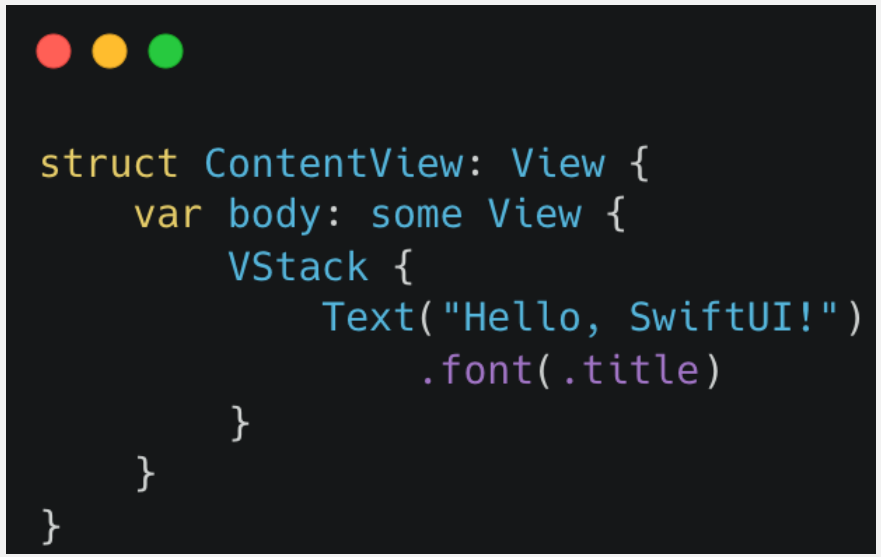
Issues When Implementing Remediation Logic

- Remediation logic is understandable, but…
  - Readability and maintainability decrease as conditions increase
    - If you want to add additional conditions, you need to append more if clauses…
  - 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
  - Allows us to create Domain Specific Languages (DSLs) within Swift code
  - Used in SwiftUI to describe user interfaces 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



```
struct ContentView: View {
    var body: some View {
        VStack {
            Text("Hello, SwiftUI!")
                .font(.title)
        }
    }
}
```

A result builder type is a type that can be used as a result builder, which is to say, as an embedded DSL for collecting partial results from the expression-statements of a function and combining them into a return value.

- "Swift Evolution: Result builders"

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

Example: Generating HTML

Without Swift result builders

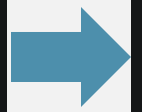
Redundant variables

```
let chapter = spellOutChapter ? "Chapter " : ""

let d1header = useChapterTitles ? [header1(chapter + "1. Loomings.")] : []
let d1p1 = paragraph(["Call me Ishmael. Some years ago"])
let d1p2 = paragraph(["There is now your insular city"])
let d1 = division(d1header + [d1p1, d1p2])

let d2header = useChapterTitles ? [header1(chapter + "2. The Carpet-Bag.")] : []
let d2p1 = paragraph(["I stuffed a shirt or two"])
let d2 = division(d2header + [d2p1])

return body([d1, d2])
```



This element is added when useChapterTitles is set to True

```
<body>
  <div>
    <h1>Chapter 1. Loomings.</h1>
    <p>Call me Ishmael. Some years ago</p>
    <p>There is now your insular city</p>
  </div>
  <div>
    <h1>Chapter 2. The Carpet-Bag.</h1>
    <p>I stuffed a shirt or two</p>
  </div>
</body>
```

[spellOutChapter: True, useChapterTitles: True]

It’s not clear what the final HTML structure will look like

Example: Generating HTML

```
return body {
  let chapter = spellOutChapter ? "Chapter " : ""
  division {
    if useChapterTitles {
      header1(chapter + "1. Loomings.")
    }
    paragraph {
      "Call me Ishmael. Some years ago"
    }
    paragraph {
      "There is now your insular city"
    }
  }
  division {
    if useChapterTitles {
      header1(chapter + "2. The Carpet-Bag.")
    }
    paragraph {
      "I stuffed a shirt or two"
    }
  }
}
```

```
<body>
  <div>
    <h1>Chapter 1. Loomings.</h1>
    <p>Call me Ishmael. Some years ago</p>
    <p>There is now your insular city</p>
  </div>
  <div>
    <h1>Chapter 2. The Carpet-Bag.</h1>
    <p>I stuffed a shirt or two</p>
  </div>
</body>
```

[spellOutChapter: True, useChapterTitles: True]

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 {
    for file in enumerateFiles("/Library/Application Support", 5) {
        if file.size <= 2 * 1024 * 1024 {
            if file.isMacho() {
                if !file.isNotarized() {
                    if yaraMatcherRoot.match(file) {
                        remediate(file)
                    }
                }
            }
        }
    }
}
```

Power of Result Builders

```
let isRoot = getuid() == 0

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

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

Power of Result Builders

```
let isRoot = getuid() == 0

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

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

For each file under ~/Library/Application Support

⊕

File size is 2 MiB or less

⊕

File format is Mach-O

⊕

Not notarized

⊕

Matches YARA rule

Power of Result Builders

```
let isRoot = getuid() == 0

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

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

Enabled when running as root

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 {}
```

Which Scanner Uses RemediationBuilder?

- RemediationBuilder is used in the following XPR scanners:
  - Adload, BadGacha, CardboardCutout, ColdSnap, Eicar, KeySteal, Pirrit, RankStank, RedPine, RoachFlight, SheepSwap, SnowDrift, WaterNet, Dolittle, Bundlore
- The remaining scanners rely on XPPluginAPI for their implementation
  - Some XPR scanners describe remediation logic both declaratively and imperatively

```
struct AdloadPlugin.AdloadRemediator {
    var statusReports: XPPluginAPI.XPPluginStatusCollator
    var remediations: RemediationBuilder.Remediations
}

struct CardboardCutoutPlugin.CardboardCutoutRemediator {
    var statusReports: XPPluginAPI.XPPluginStatusCollator
    var remediations: RemediationBuilder.Remediations
}
```

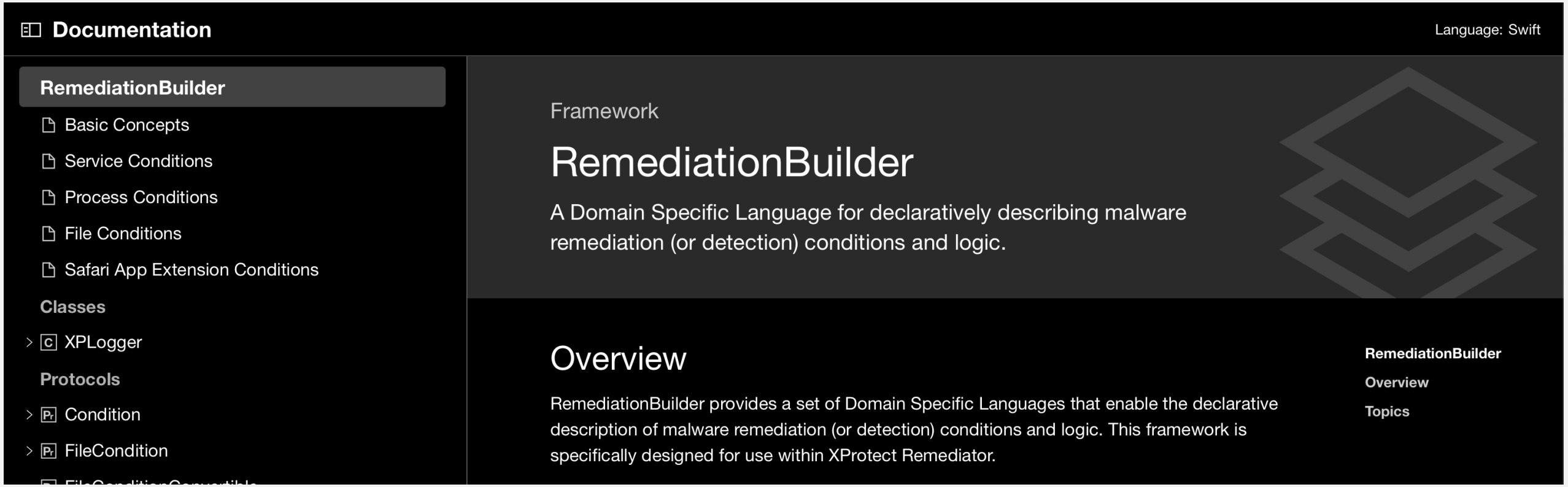
```
struct ColdSnapPlugin.ColdSnapPlugin.ColdSnapRemediator {
    var statusReports: XPPluginAPI.XPPluginStatusCollator
    var remediations: RemediationBuilder.Remediations
}

struct EicarPlugin.EicarRemediator {
    var statusReports: XPPluginAPI.XPPluginStatusCollator
    var remediations: RemediationBuilder.Remediations
}
```

Specification of RemediationBuilder DSL

https://github.com/FFRI/RemediationBuilderDSLSpec

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



FileRemediationBuilder Example

```
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

ProcessRemediationBuilder Example

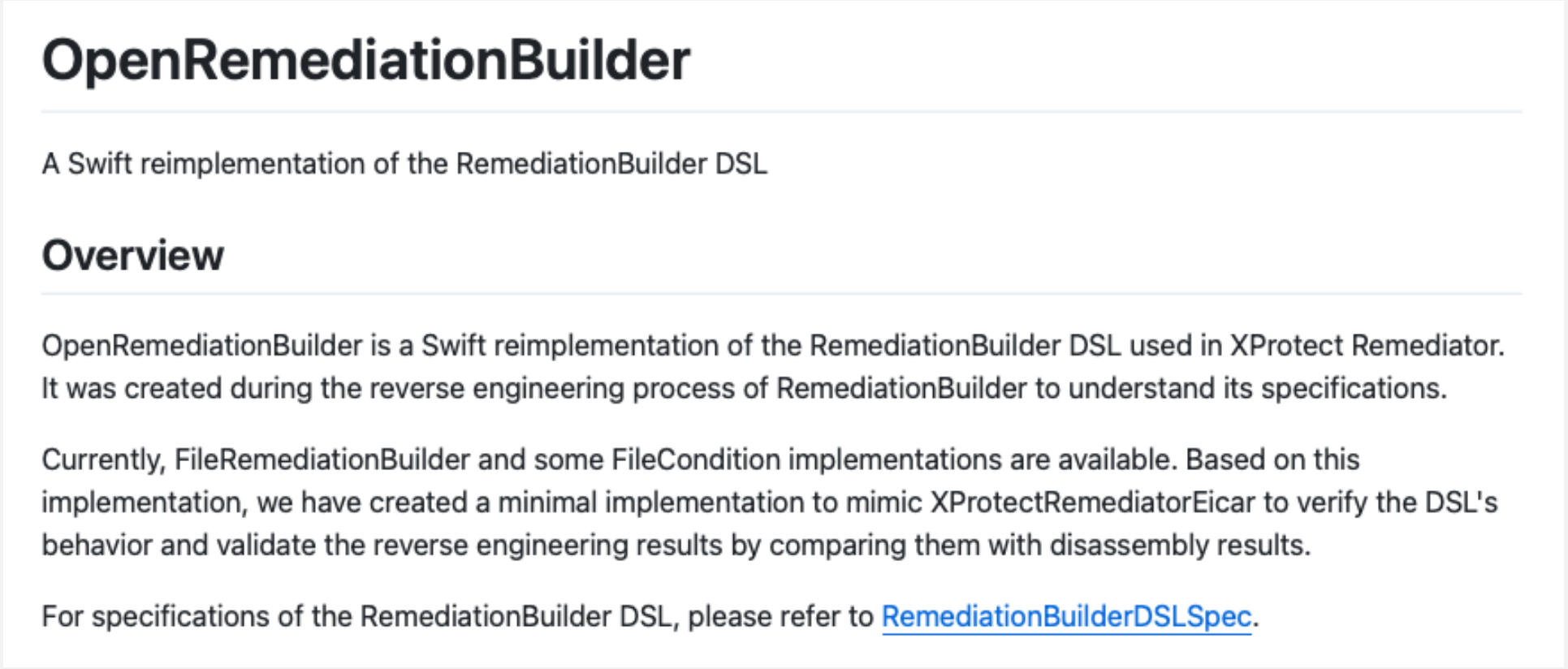
```
let pathPatterns = ["/Library/Application Support/",
"/Library/ApplicationSupport/", ".mitmproxy", "/tmp/", "Install.command"]

AdloadRemediator {
    for pathPattern in pathPatterns {
        Process { // ProcessRemediationBuilder DSL block
            // Process conditions go here
            ProcessIsNotarised(false)
            ProcessMainExecutable { // FileRemediationBuilder DSL block
                // File conditions go here
                FilePath(.StringContains(pathPattern))
                FileYara(YaraMatcher(adloadYara))
            }
        }
    }
```

Process is NOT notarized + Backing file path is /tmp/, .mitmproxy, … + Backing file matches Adload YARA rule

OpenRemediationBuilder

- Open-source reimplementation of RemediationBuilder
- A minimal implementation that reproduces XPR Eicar's functionality
- https://github.com/FFRI/OpenRemediationBuilder



OpenRemediationBuilder

A Swift reimplementation of the RemediationBuilder DSL

Overview

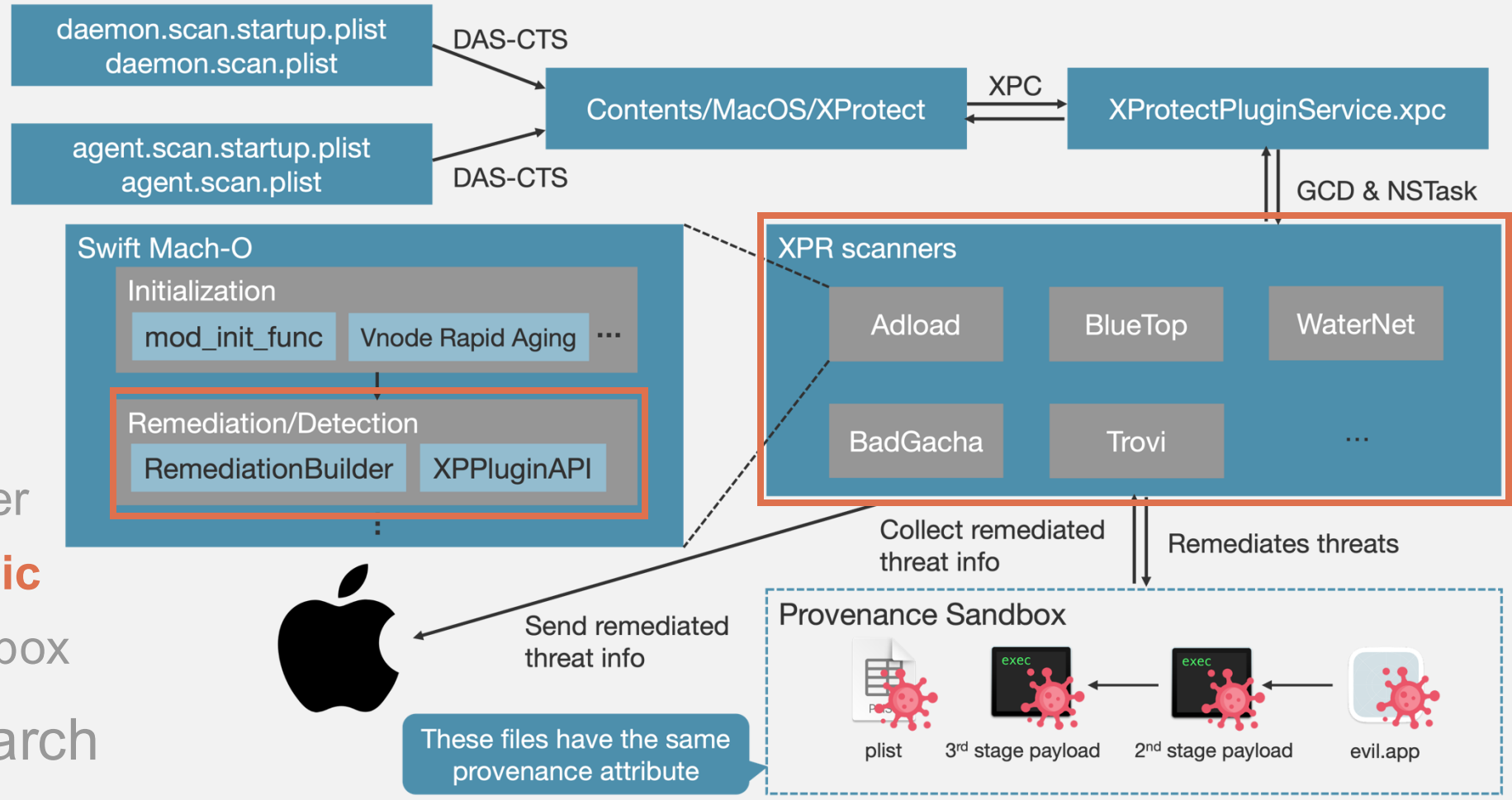
OpenRemediationBuilder is a Swift reimplementation of the RemediationBuilder DSL used in XProtect Remediator. It was created during the reverse engineering process of RemediationBuilder to understand its specifications.

Currently, FileRemediationBuilder and some FileCondition implementations are available. Based on this implementation, we have created a minimal implementation to mimic XProtectRemediatorEicar to verify the DSL's behavior and validate the reverse engineering results by comparing them with disassembly results.

For specifications of the RemediationBuilder DSL, please refer to RemediationBuilderDSLSpec.

Outline

1. Introduction
2. Tooling
3. **RE results**
    1. Overview
    2. Initialization
    3. RemediationBuilder
    4. **Remediation Logic**
    5. Provenance Sandbox
4. Vulnerability Research
5. Conclusion



XPR RoachFlight

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

Remediation Logic of XPR RoachFlight

```
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
  - The CDHash of the 2nd stage payload used in the 3CX supply chain attack
  - This sample is commonly referred to as UpdateAgent
  - The sample was analyzed by Patrick Wardle and presented at BHUSA 2023

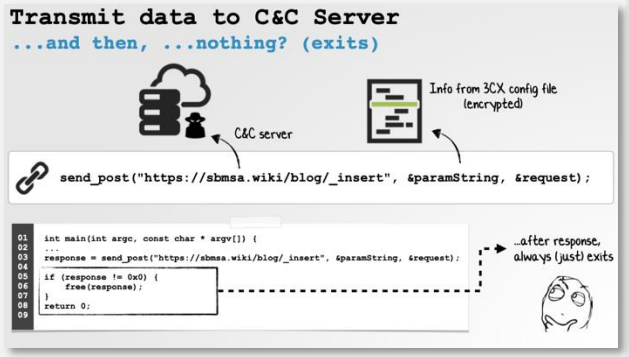
https://x.com/patrickwardle/status/1641690082854989827

```
% codesign -dvvv UpdateAgent
Executable=/Users/kato/Downloads/SmoothOperator/UpdateAgent
Identifier=payload2-55554944839216049d683075bc3f5a8628778bb8
Format=Mach-O thin (x86_64)
CodeDirectory v=20100 size=450 flags=0x2(adhoc) hashes=6+5 location=embedded
Hash type=sha256 size=32
CandidateCDHash sha256=04e23817983f1c0e9290ce7f90e6c9e75bf45190
CandidateCDHashFull sha256=04e23817983f1c0e9290ce7f90e6c9e75bf4519020aa8aacb16e174566c380c6
Hash choices=sha256
CMSDigest=04e23817983f1c0e9290ce7f90e6c9e75bf4519020aa8aacb16e174566c380c6
CMSDigestType=2
CDHash=04e23817983f1c0e9290ce7f90e6c9e75bf45190
Signature=adhoc
Info.plist=not bound
TeamIdentifier=not set
Sealed Resources=non
Internal requirements count=0 size=12
```

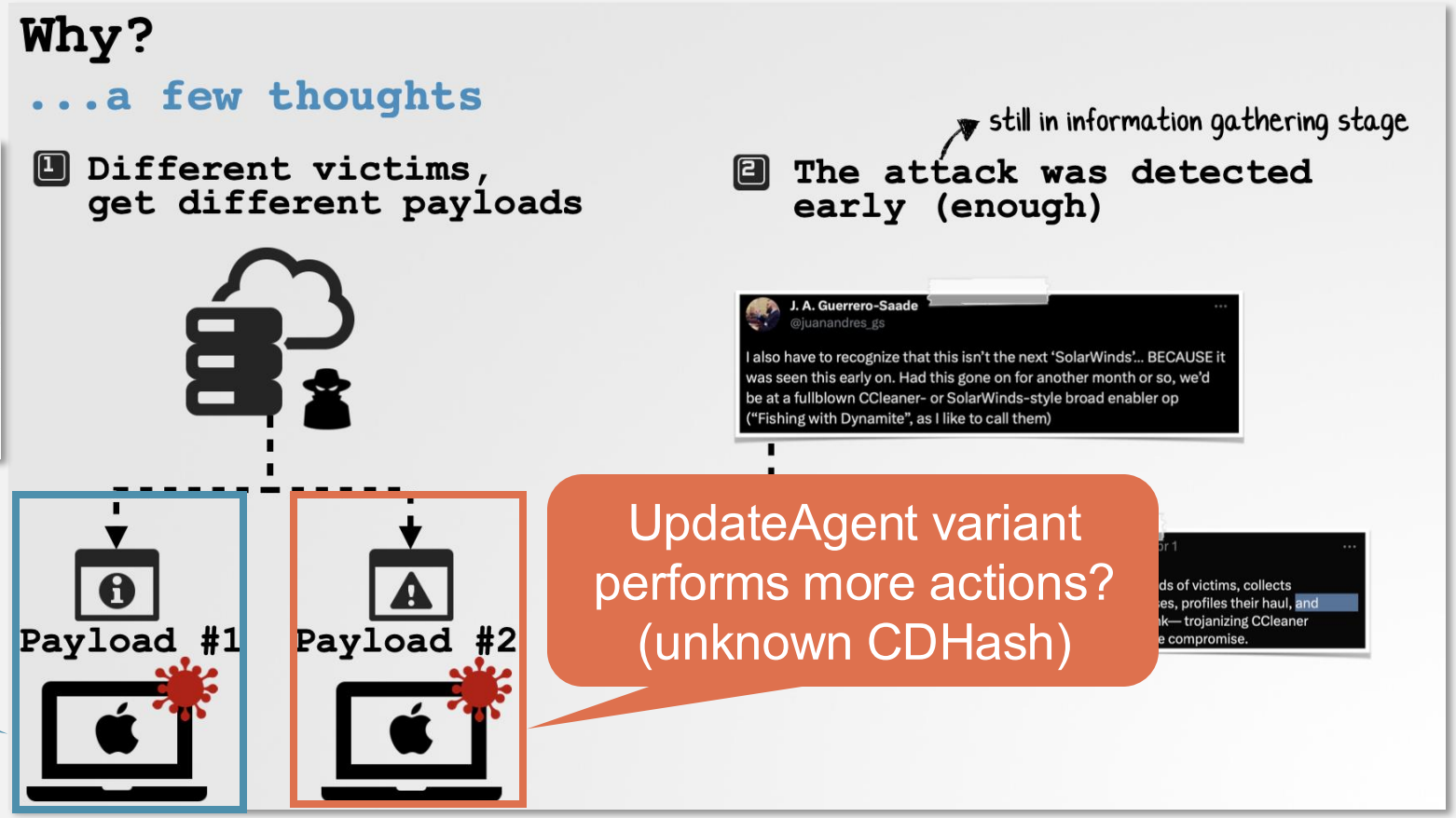
What Are These Two CDHashes?

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

Sample analyzed by Patrick Wardle



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

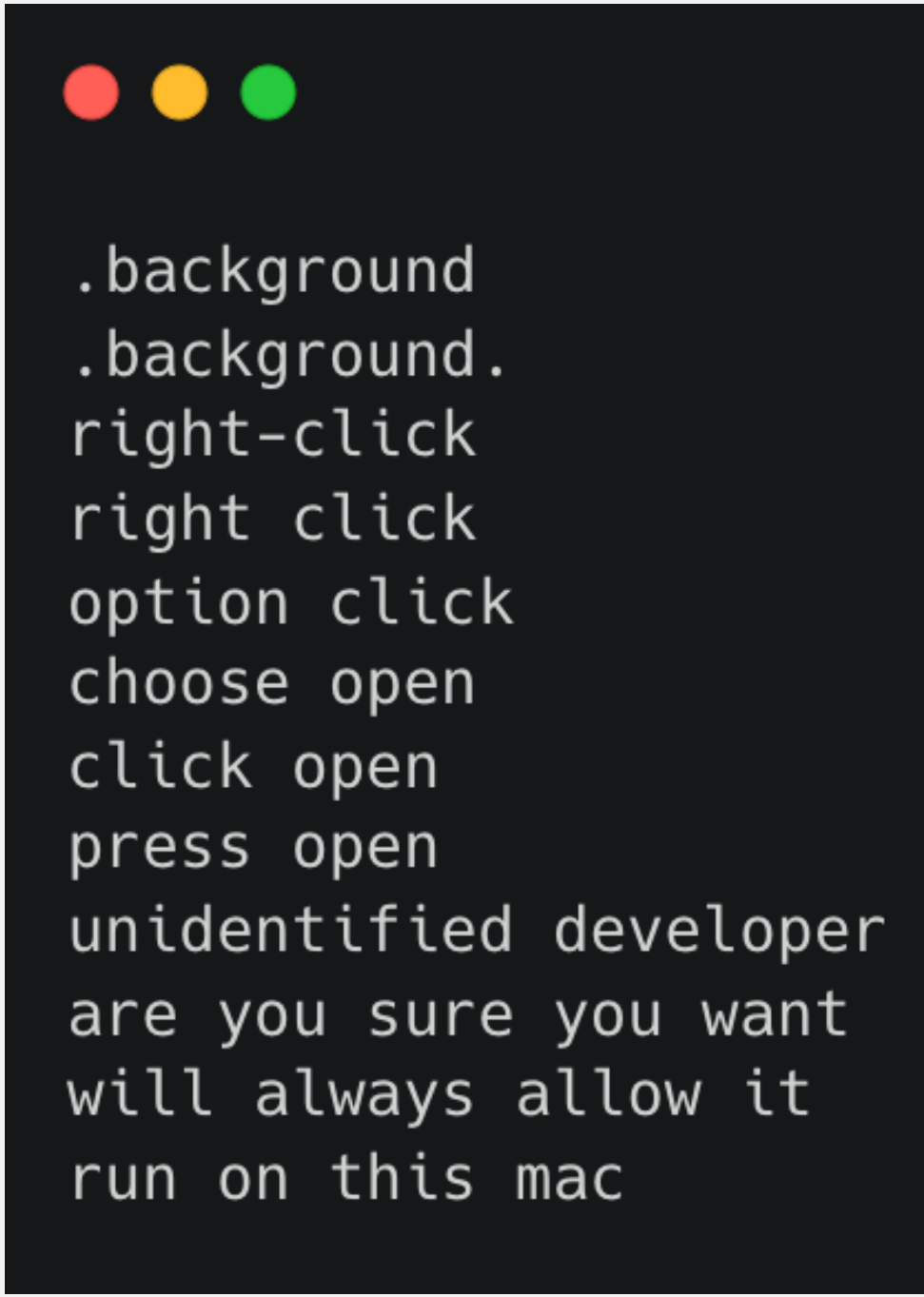


UpdateAgent variant performs more actions? (unknown CDHash)

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

XPR BadGacha

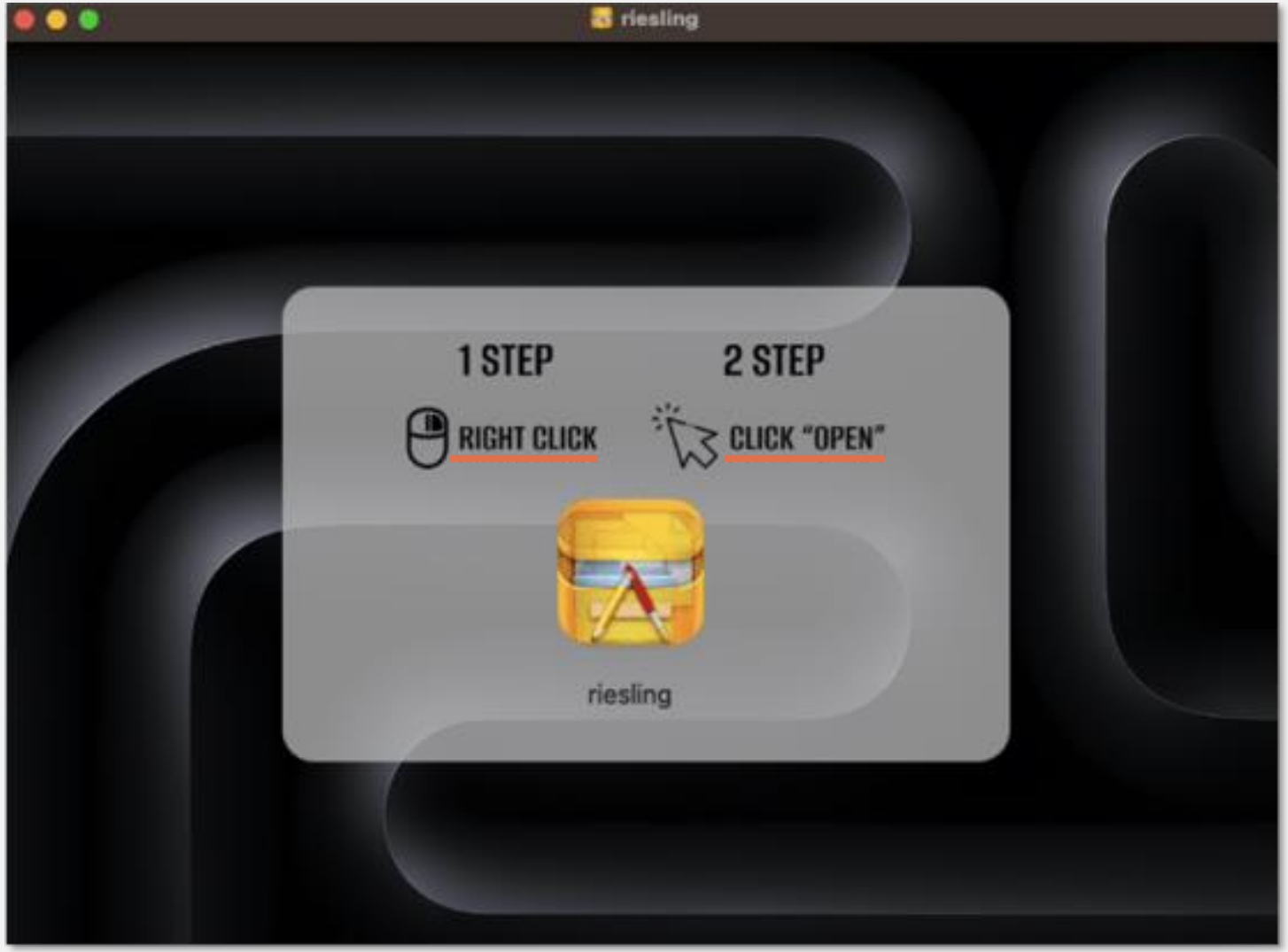
- Added in XPR version 91 on 2 March 2023
- The decrypted strings appear unrelated to any remediation functionalities
- What are these texts used for?



XPR BadGacha: 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

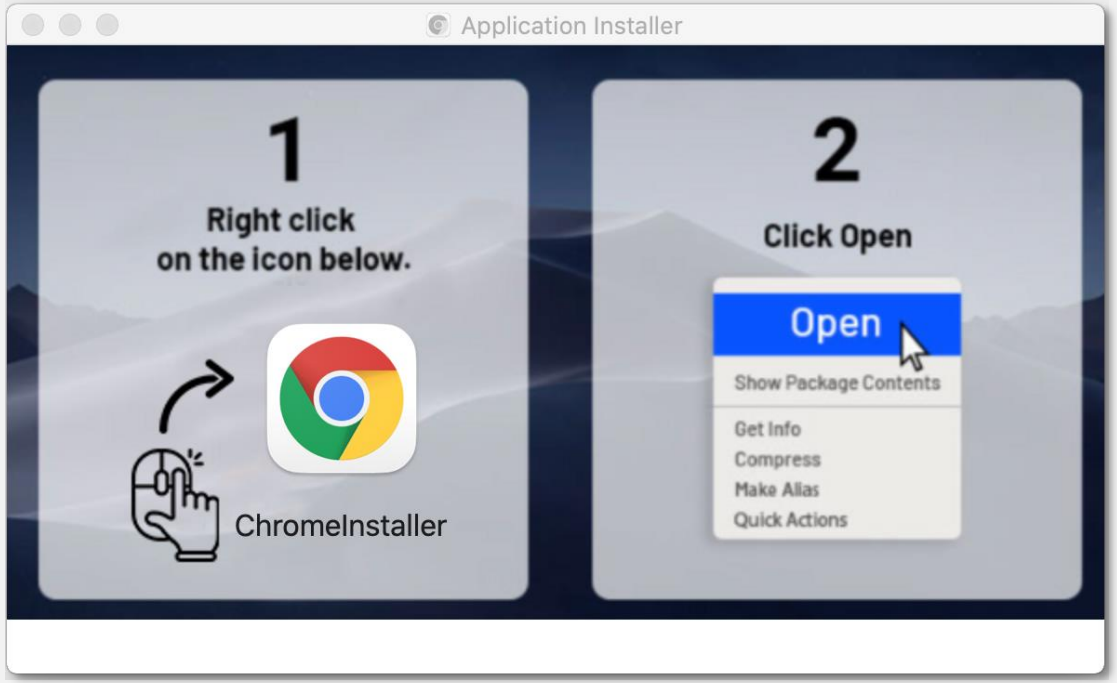
OCR-based Gatekeeper Bypass Detection

- XPR BadGacha contains detection logic for Gatekeeper bypass
  - Enumerates mounted DMG files using FileManager.mountedVolumeURLs
  - Retrieves text strings in background images of mounted volumes using OCR
  - Searches for Gatekeeper bypass-related strings
- If it find strings, it reports the threat including the DMG file information
  - Only reporting is performed, without deleting or unmounting the DMG

Which Malware Family Does XPR BadGacha Detect?

- Appears to be a generic detection module?
  - In fact, the detection logic has triggered on several different malware families
    - E.g., Empire Transfer and ChromeLoader
  - Apple may have designed XPR BadGacha as a threat hunting scanner

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



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

Other BadGacha Detection

- A mechanism to detect processes without their backing files was previously implemented (removed in XPR version 135)
  - The detection was likely removed due to frequent false positive detections
  - This logic also appears not be designed to target a specific malware family

```
BadGachaRemediator {
    Process {
        ProcessHasBackingFile(false)
    }.reportOnly()
}
```

After installing the latest stable version of Chromium, I have been getting the following warnings when running an XProtect Remediator scan. I'm not sure if this is a bad issue, but I think it is something Apple should look at. Thanks.

- "Apple Developer Forums"

https://developer.apple.com/forums/thread/742828

False positive alert reported by a user

XPR RedPine

- Added in version 114 on October 12, 2023, and retired in 2024
- Decrypted strings are a YARA rule and four file paths
  - The YARA rule detects the TriangleDB iOS implant
- Kaspersky researchers noted the possibility of TriangleDB macOS implant
  - RedPine appears to be TriangleDB macOS implant
  - No details about TriangleDB macOS implant have been made public

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/

XPR RedPine: Two Scans

- XPR RedPine has the com.apple.system-task-ports.read entitlement
  - Allows to obtain task ports and read memory of other processes
- When XPR RedPine is executed as root, it performs two scans
  - Scans the main executable file in memory
  - Scans loaded libraries (called LoadedLibrary Scanner)

```
% codesign -dv --entitlements -
/Library/Apple/System/Library/CoreServices/XProtect.app/Contents/MacOS/XProtectRemediatorRedPine
Executable=/Library/Apple/System/Library/CoreServices/XProtect.app/Contents/MacOS/XProtectRemediatorRedPine
...
    [Key] com.apple.system-task-ports.read
    [Value]
        [Bool] true
```

Scanning the Main Executable in Memory

- XPProcessMemoryAPI is used for in-memory scanning
  - Only __TEXT segment is scanned and matches it against the YARA rule
  - 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)
```

```
do
    char rax_6

    if ((arg1.b & 1) != 0)
        rax_6 = _is_platform_binary(zx.q(i))

    if ((arg1.b & 1) == 0 || rax_6 == 0)
        if (_swift_isUniquelyReferenced_nonNull_native(result) == 0)
            result = sub_10000a2d0(0, *(result + 0x10) + 1, 1, result)
```

Why Does XPR RedPine 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: YARA scan described with ProcessRemediationBuilder is performed 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 really dylib paths?

Peculiar Logic

- Except for /usr/lib/libsqlite3.dylib, no actual file paths are specified!
  - CoreLocation and AVFoundation are symlinks
    - When these are loaded as libraries, their symlinks are resolved
  - FMCore.framework is a directory
    - Of course, it's impossible to load a directory as a dylib…

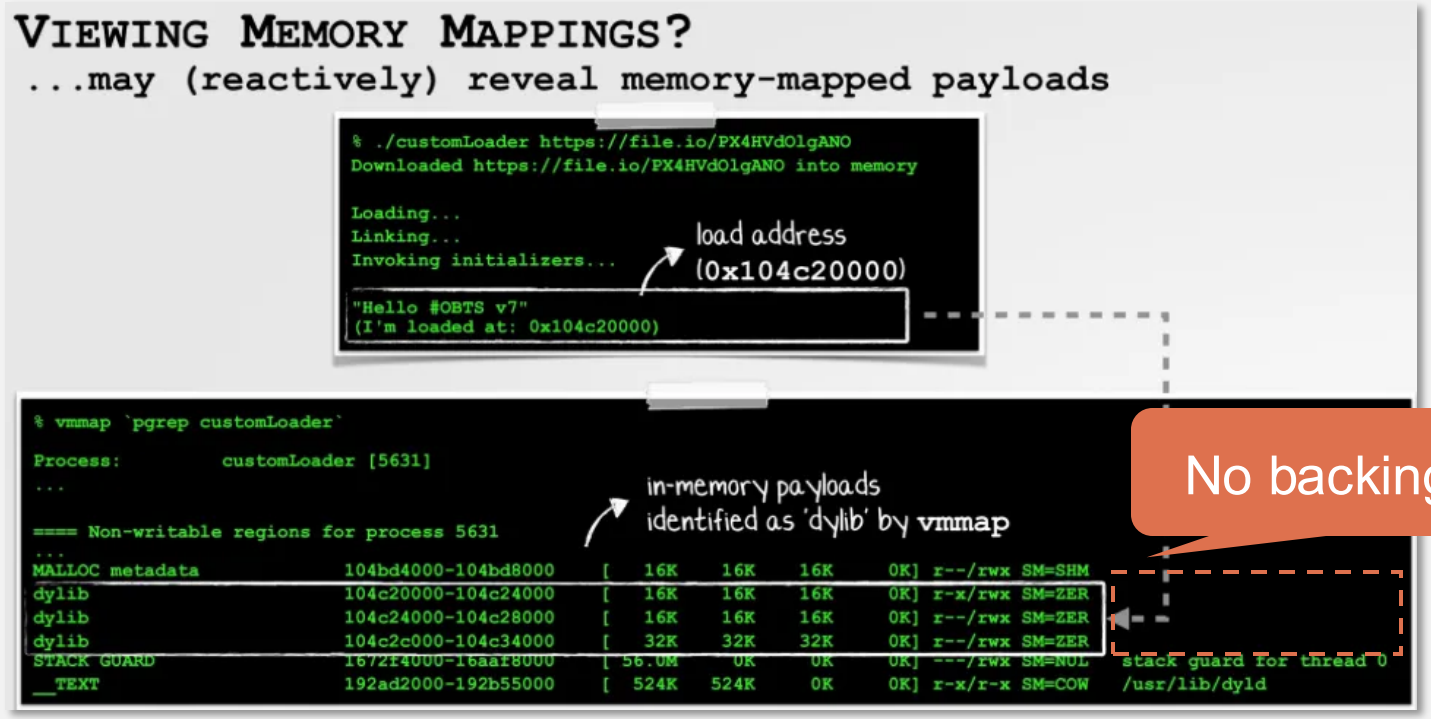
```
% file /System/Library/PrivateFrameworks/FMCore.framework
/System/Library/PrivateFrameworks/FMCore.framework: directory
% file /System/Library/Frameworks/CoreLocation.framework/CoreLocation
/System/Library/Frameworks/CoreLocation.framework/CoreLocation: broken symbolic link to Versions/Current/CoreLocation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



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

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

Stealthier Reflective Loader

- I developed a new reflective loader that can specify a backing file
  - Achieved by modifying dyld's all_imges_info
- 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
```

Directory path is specified as the backing file

Remaining Mysteries

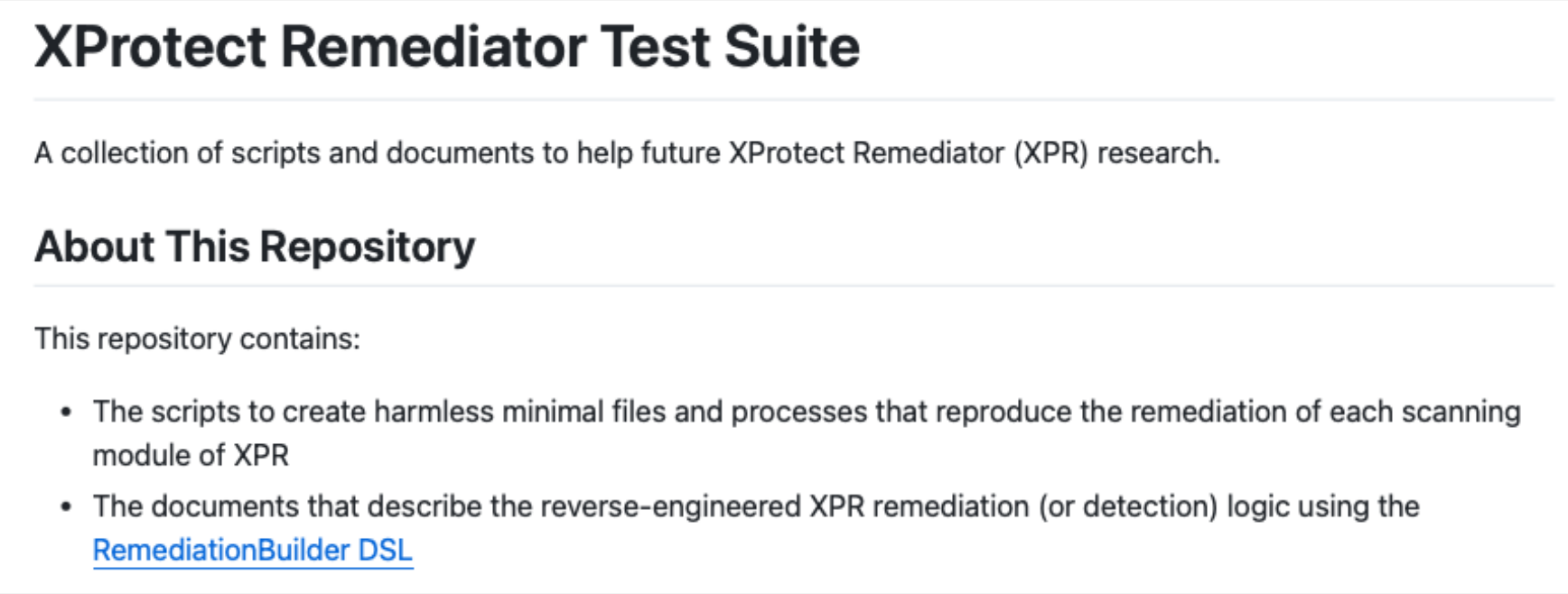
- It’s more natural to specify an unused system library path as a backing file
  - Why specify a directory or symlink?
- Why doesn’t XPR RedPine remediate threat?
  - Because reportOnly property is set to True
  - If remediation wasn't the goal, what was the purpose of deploying it?

```
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.

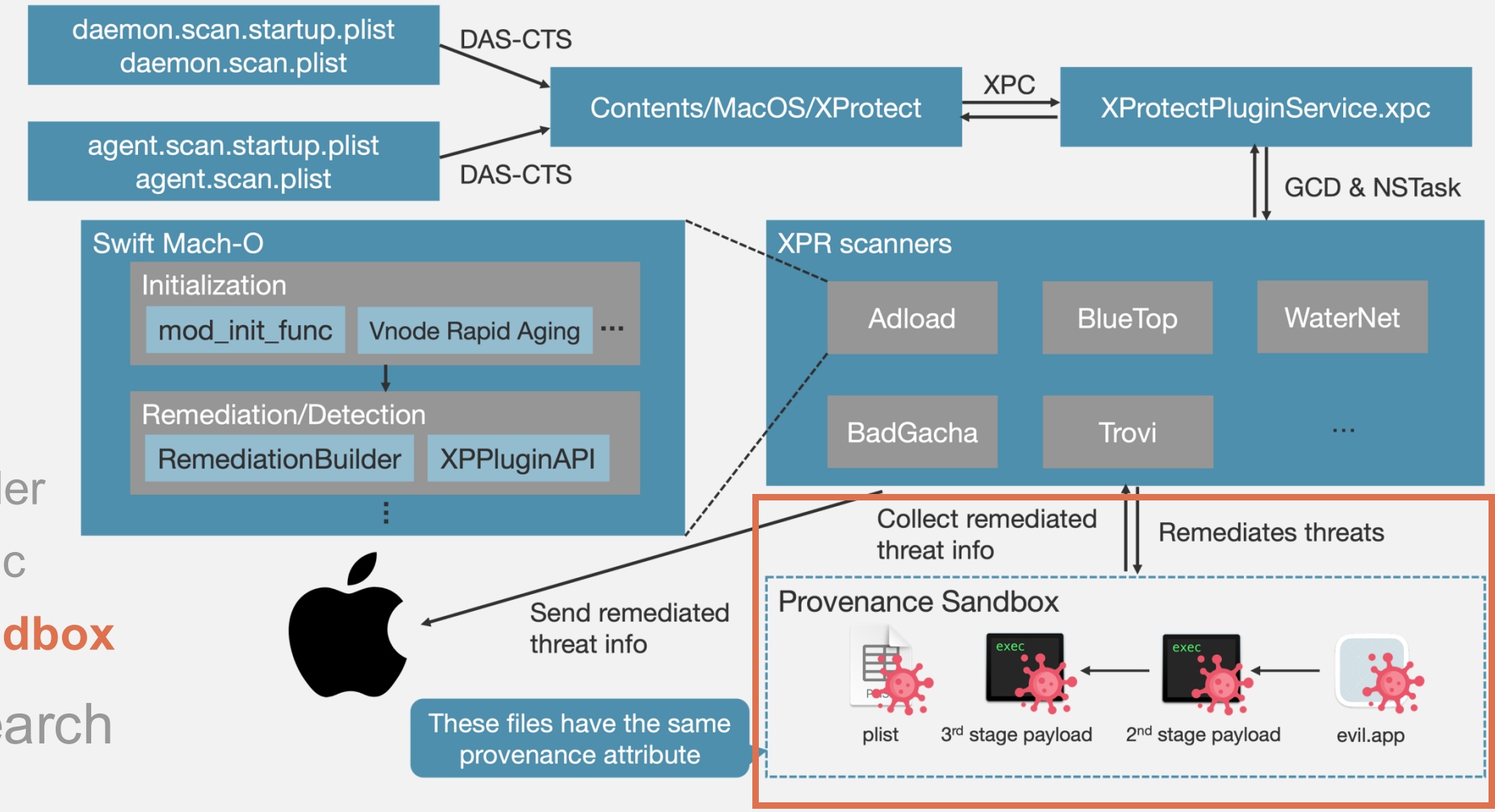
About This Repository

This repository contains:

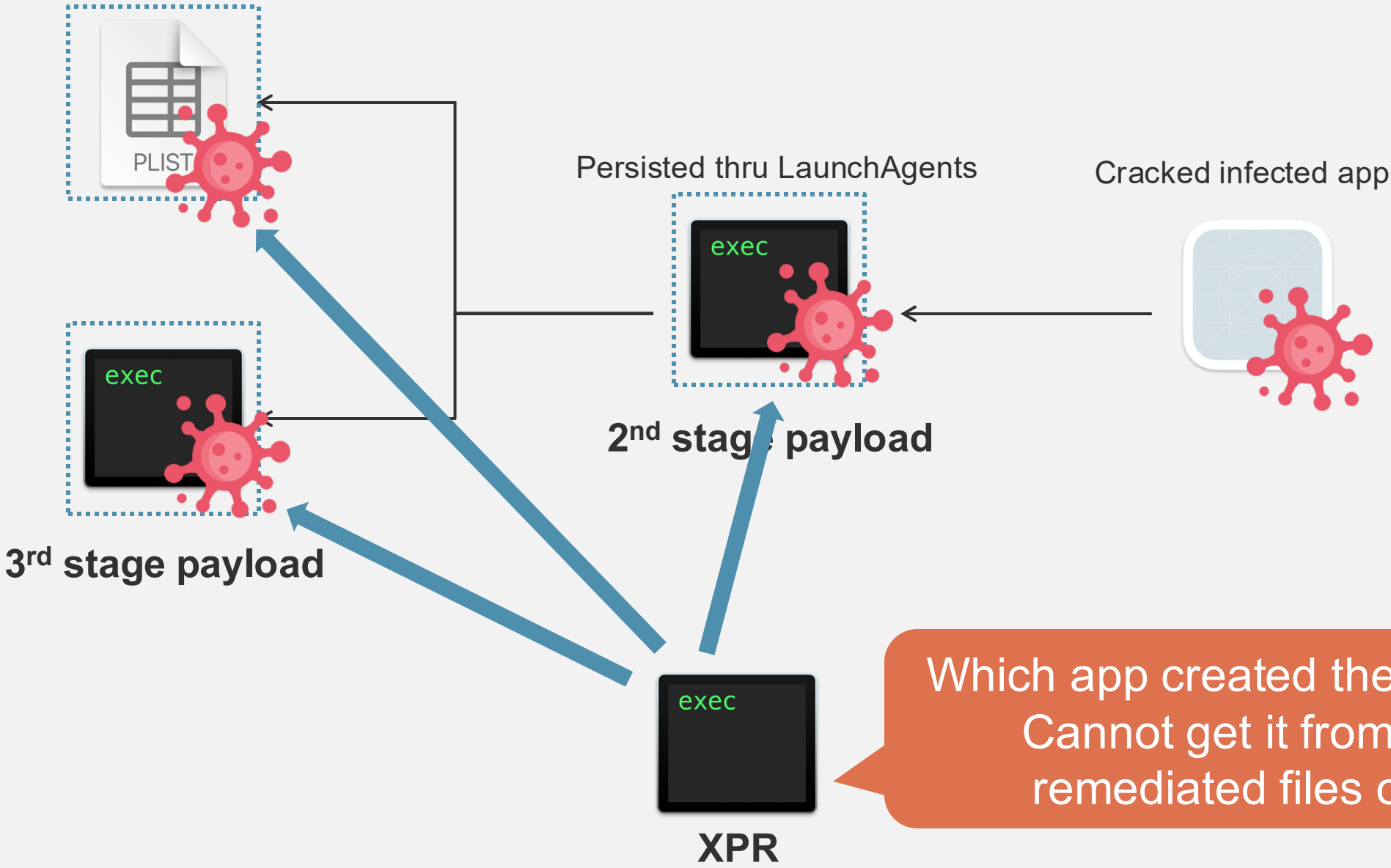
- The scripts to create harmless minimal files and processes that reproduce the remediation of each scanning module of XPR
- The documents that describe the reverse-engineered XPR remediation (or detection) logic using the RemediationBuilder DSL

Outline

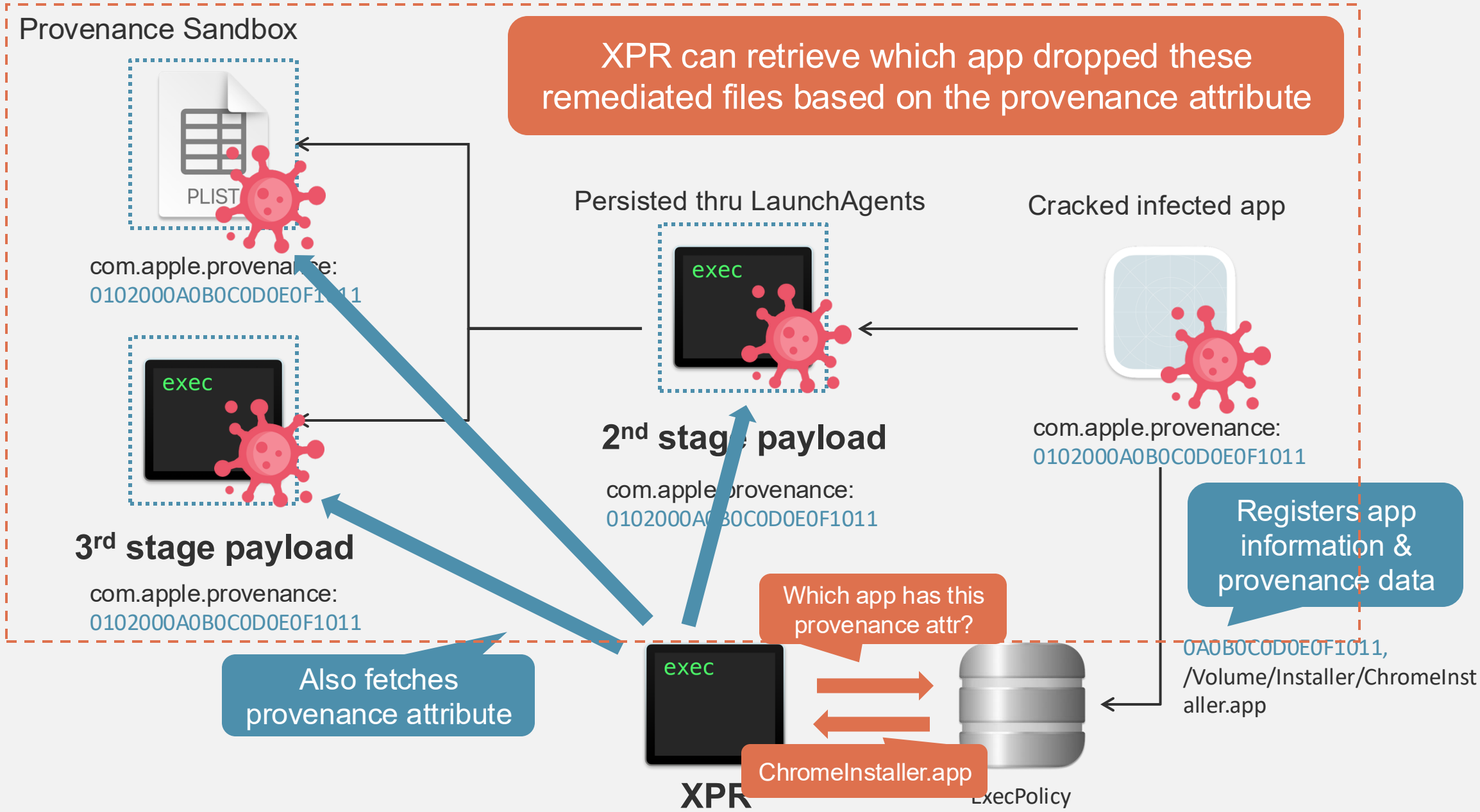
1. Introduction
2. Tooling
3. **RE results**
   1. Overview
   2. Initialization
   3. RemediationBuilder
   4. Remediation Logic
   5. **Provenance Sandbox**
4. Vulnerability Research
5. Conclusion



Which App Created Remediated Files?



Solution: Provenance Sandbox



Provenance Sandbox

- Enables identification of processes that create and modify files
  - For App Sandbox, files that are dropped have a quarantine attribute attached
  - You can think of Provenance Sandbox as being replaced by the provenance attribute
  - Like App Sandbox, it also applies to child processes

- When a process is running in Provenance Sandbox, a provenance attribute is attached to files during the following operations:
  - create, rename, setacl, setattrlist, setextattr, setflags, setmode, setowner, setutimes, truncate, deleteextattr, swap, open (called with O_RDWR or O_TRUNC flags), link

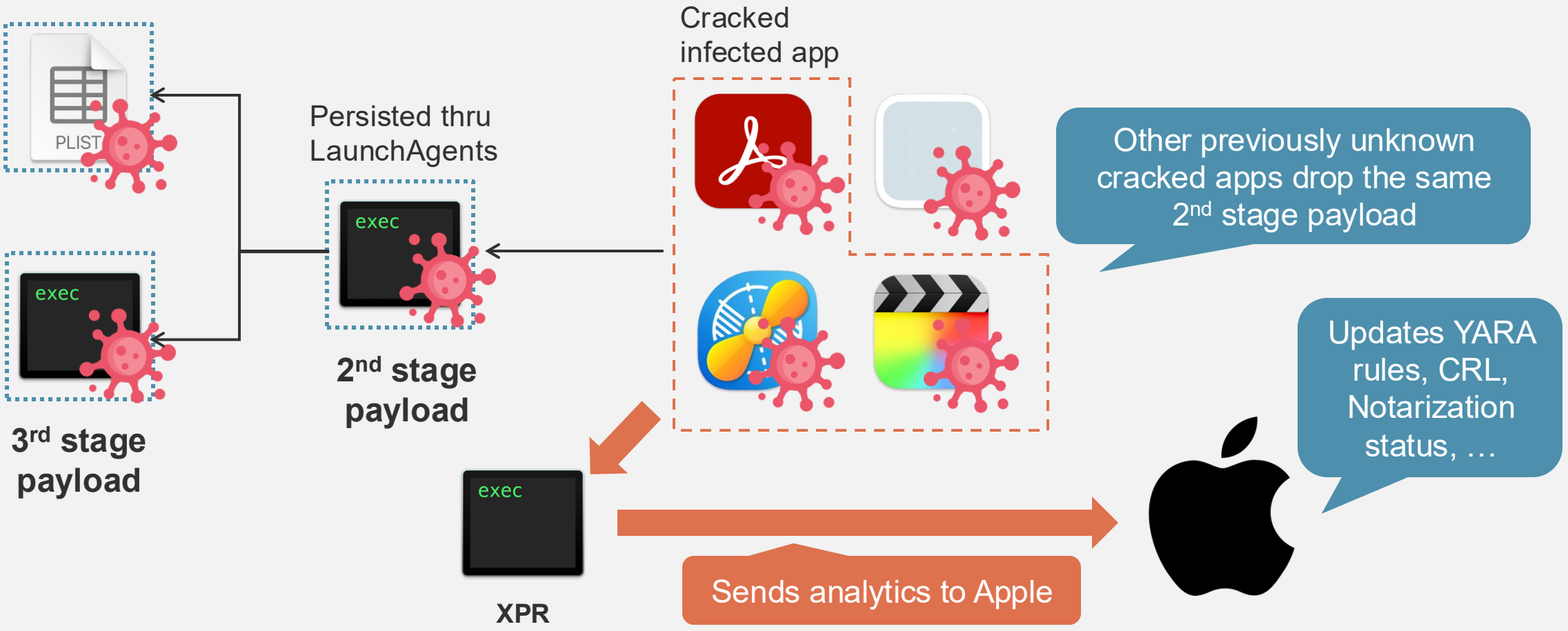
com.apple.provenance

- An 11-byte integer value
  - 01 02 00 E9 AC 02 3A 98 15 DF 25
    - The use of the first 3 bytes is unknown
    - The following 8 bytes are random numbers (generated by arc4random)

```
Default     0x0                 196    0    syspolicyd: [com.apple.syspolicy.exec:default] GK evaluateScanResult: 0, PST: (path:
2a7545b632d3156f), (team: UBF8T346G9), (id: com.microsoft.VSCode), (bundle_id: com.microsoft.VSCode), 1, 0, 1, 0, 4, 4, 0
...
Default     0x0                 196    0    syspolicyd: [com.apple.syspolicy.exec:default] Created provenance data for target:
TA(25df15983a02ace9, 2), PST: (path: 2a7545b632d3156f), (team: UBF8T346G9), (id: com.microsoft.VSCode), (bundle_id:
com.microsoft.VSCode)
Default     0x0                 196    0    syspolicyd: [com.apple.syspolicy.exec:default] Handling provenance root:
TA(25df15983a02ace9, 2)
Default     0x0                 196    0    syspolicyd: [com.apple.syspolicy.exec:default] Wrote provenance data on target:
TA(25df15983a02ace9, 2), PST: (path: 2a7545b632d3156f), (team: UBF8T346G9), (id: com.microsoft.VSCode), (bundle_id:
com.microsoft.VSCode)
Default     0x0                 196    0    syspolicyd: [com.apple.syspolicy.exec:default] Putting executable into provenance with
metadata: TA(25df15983a02ace9, 2)
Default     0x0                 196    0    syspolicyd: [com.apple.syspolicy.exec:default] Putting process into provenance tracking
with metadata: 732, TA(25df15983a02ace9, 2)
Default     0x0                 196    0    syspolicyd: [com.apple.syspolicy.exec:default] Tracking process with attributes: 732,
TA(25df15983a02ace9, 2)
```

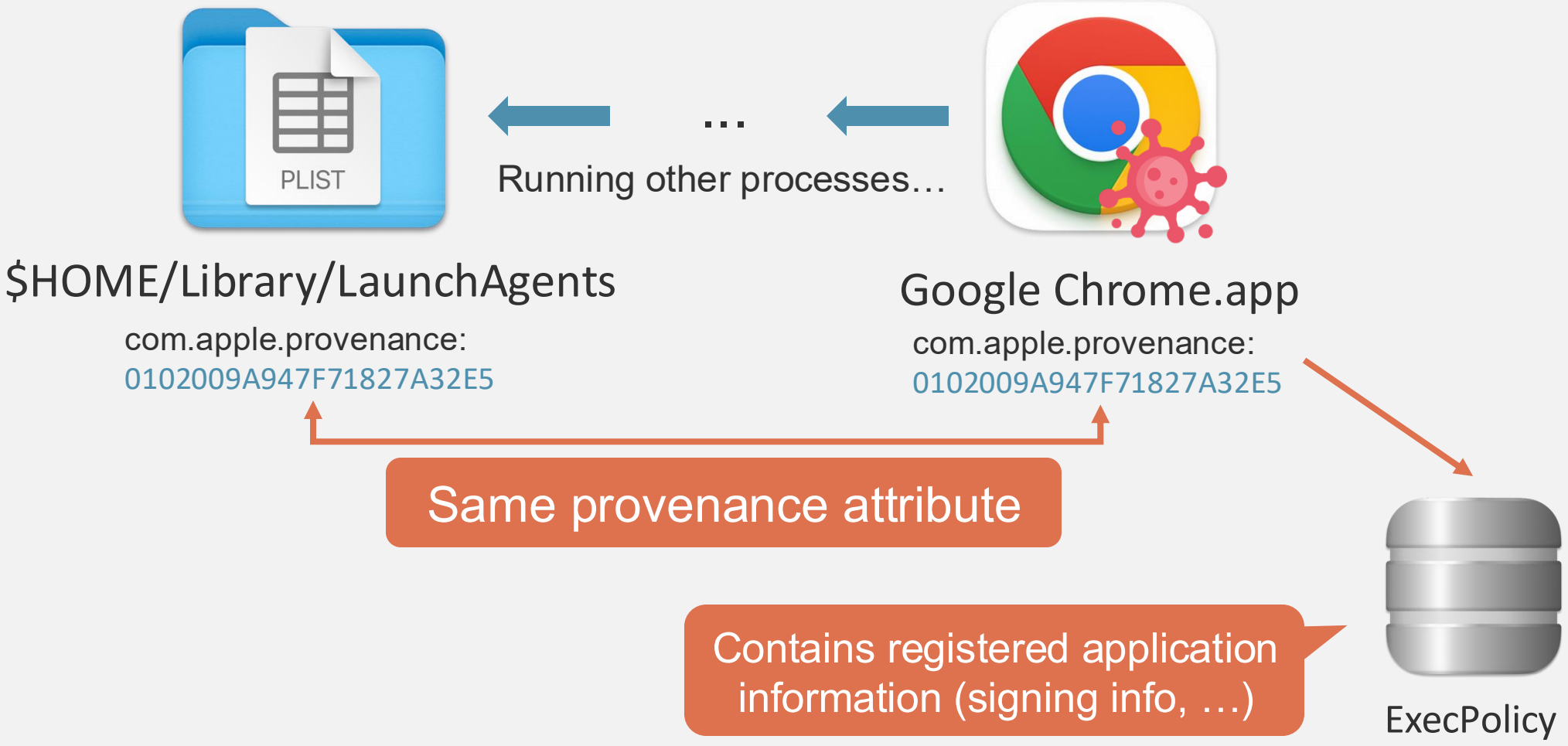
Why XPR Collects Provenance Attribute?

- Provenance attribute helps to discover malware variants
  - In case that there are other samples that drop the same 2nd stage payload



How to Utilize Provenance Attribute

- Identifying applications that achieved persistence




```
INSERT INTO provenance_tracking VALUES(-19313465895199977798,'/Applications/Google
Chrome.app','com.google.Chrome','b130ce35ffa637db6c63cba3f7ddc93f64caa40f','EQHXZ8M8AV','com.google.Chrome',
```

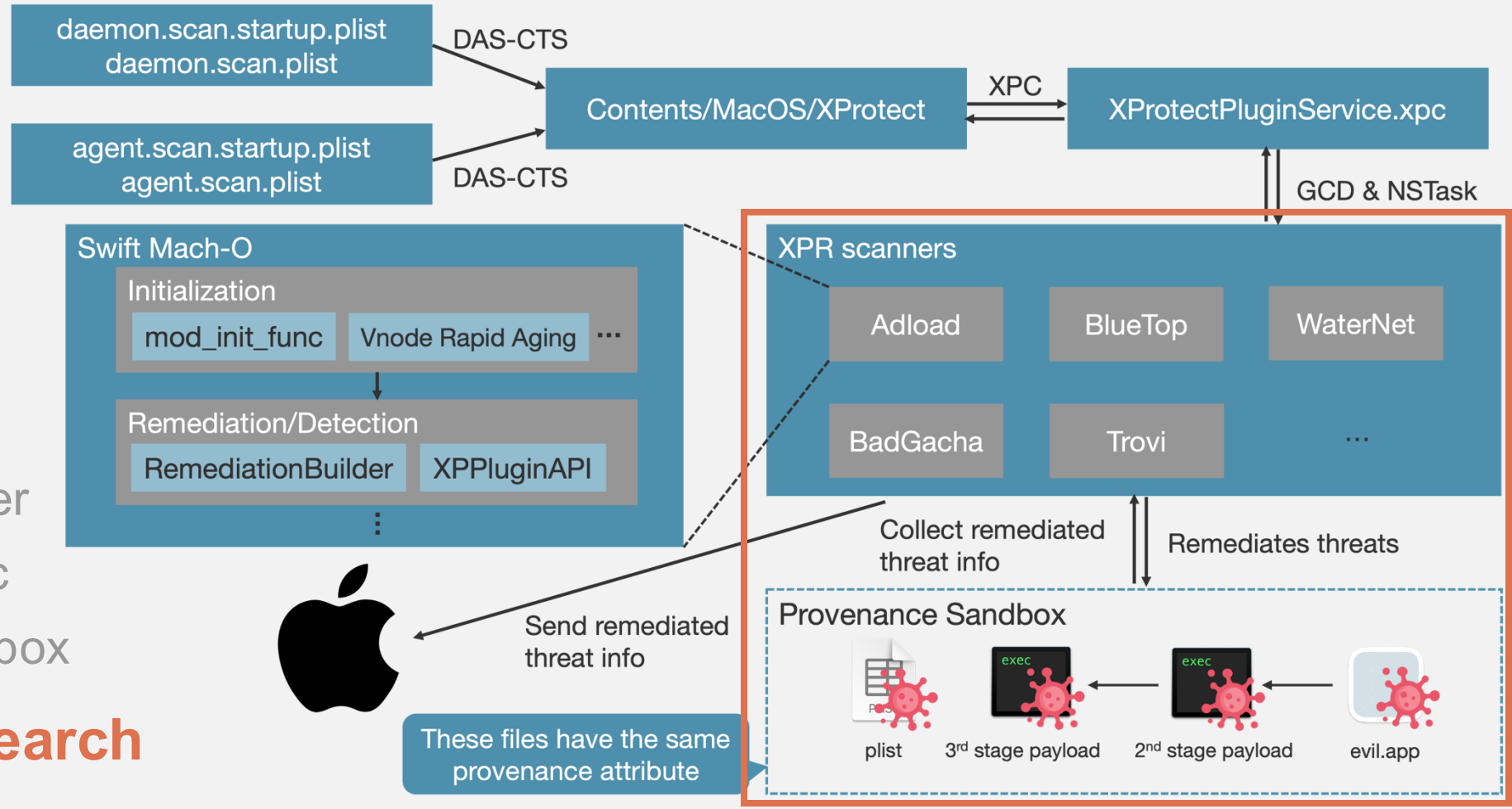
Tools to Utilize Provenance Attribute

- ShowProvenanceInfo
  - This app retrieves provenance attribute, then enumerates which apps created and modified files
  - https://github.com/FFRI/ShowProvenanceInfo
- Aftermath plugin collecting provenance attribute is also implemented
  - Planning to submit a Pull Request after this talk

https://github.com/jamf/aftermath

Outline

1. Introduction
2. Tooling
3. RE results
    1. Overview
    2. Initialization
    3. RemediationBuilder
    4. Remediation Logic
    5. Provenance Sandbox
4. **Vulnerability Research**
5. Conclusion



Arbitrary File Deletion (TCC Bypass)

- Arbitrary file deletion vulnerability
  - Inspired by “Aikido Wiper” by Or Yair
  - Vulnerabilities allow to delete arbitrary files by exploiting TOCTOU in EDR and AV
  - His research is focused on Windows platform
  - On macOS, achieving arbitrary file deletion requires TCC bypass

https://www.safebreach.com/blog/safebreach-labs-researcher-discovers-multiple-zero-day-vulnerabilities/

Classic TOCTOU: CVE-2024-40843

- YARA rule matching → Remediating file
- Replace the target file using a symlink
  - After matching YARA rule before remediating file
  - The timing of YARA rule match can be monitored through log command

```
user=$(stat -f %Su /dev/console)
if [ -n "$(log stream --level debug --process XProtectRemediatorBlueTop | grep -m 1 "YARARuleMatchV4")" ]; then
    mv /private/tmp/hoge /private/tmp/fuga
    ln -s /Users/$user/Desktop /private/tmp/hoge
fi
```

```
Default     0x0                 3670   0    XProtectRemediatorBlueTop: [com.apple.XProtectFramework.PluginAPI:YARARuleMatchV4] Rule
YaraRule[macos_bluego]: Hit
```

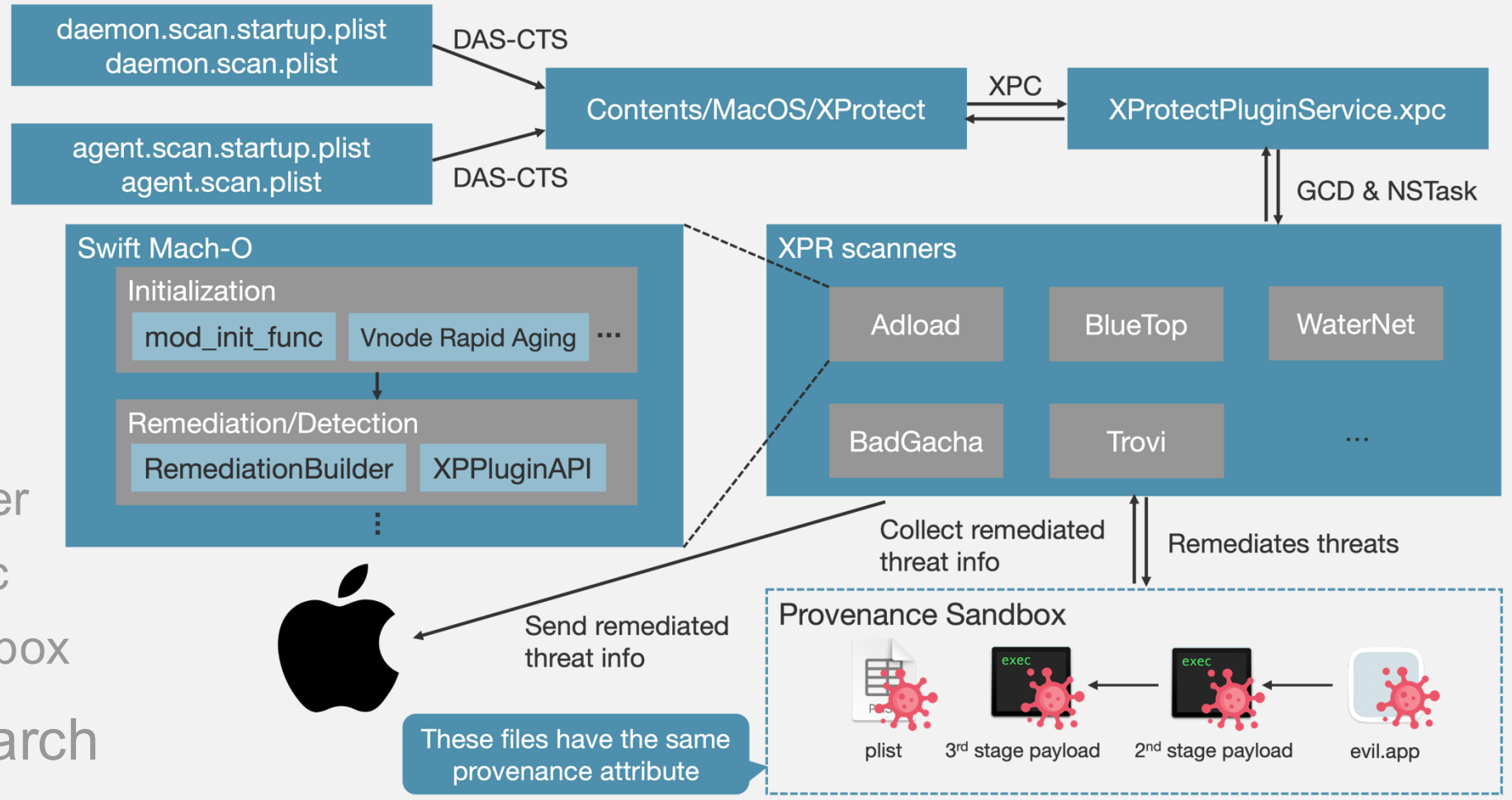
Provenance Sandbox Bypass

- I reported several bypass methods
- Example 1: Process execution via LaunchServices
  - Drop a .terminal script and execute .terminal using open
    - While executed by Terminal.app, Terminal does not run within the Provenance Sandbox
- Example 2: Bypass through XPC
  - Execute workflow files via automator (fixed in Sequoia 15)

- **Previous App Sandbox bypass techniques are likely applicable**

Outline

1. Introduction
2. Tooling
3. RE results
    1. Overview
    2. Initialization
    3. RemediationBuilder
    4. Remediation Logic
    5. Provenance Sandbox
4. Vulnerability Research
5. **Conclusion**



Conclusion

- **Covered:**
  - Tooling and how to analyze XPR
  - XPR internals (initialization, XPAPIHelpers, RemediationBuilder, remediation logic)
  - Provenance Sandbox (brief overview, how to utilize provenance attribute)
  - A bit of vulnerability research
- **Not covered:**
  - Provenance Sandbox internals and other use cases of provenance attribute
  - Other XPR scanners internals (such as XPR CardboardCutout)
  - Several bugs of XPR scanners

Future Work

- XProtect Behavior Service (XBS)
  - XBS internals and how can XBS detection be bypassed?
  - Stay tuned!
- Tracking Gatekeeper
  - I found this while analyzing syspolicyd
  - It also appears to use a provenance attribute

```
void -[AppleMetricsProvider sendTrackingGatekeeperViolationForTool:withActor:withProvenanceData:withToolType:isNotarizedOrBetter:](
  struct AppleMetricsProvider* self, SEL sel, id sendTrackingGatekeeperViolationForTool, id withActor, id withProvenanceData,
  NSUInteger withToolType, char isNotarizedOrBetter)
```

```
void -[ExecManagerService handleTrackingGatekeeperViolationForPath:withParentPath:withProvenanceID:withToolType:](
  struct ExecManagerService* self, SEL sel, id handleTrackingGatekeeperViolationForPath, id withParentPath, id withProvenanceID,
  NSUInteger withToolType)
```

Black Hat Sound Bytes

- **XPR is a treasure trove of Apple's threat intelligence**
  - Security researchers should actively engage in analyzing scanners in future updates
  - My custom tools for XPR analysis will be published on GitHub, so please use them
- **Provenance attribute serves as a valuable forensic artifact**
  - Blue teams make the most of it
  - Red teams may need to bypass Provenance Sandbox to achieve stealth operations
- **Vulnerabilities in XPR and Provenance Sandbox are quite basic**
  - Similar bugs found in AVs on other platforms may still exist in XPR
  - Previous App Sandbox escape bugs may apply to Provenance Sandbox bypass

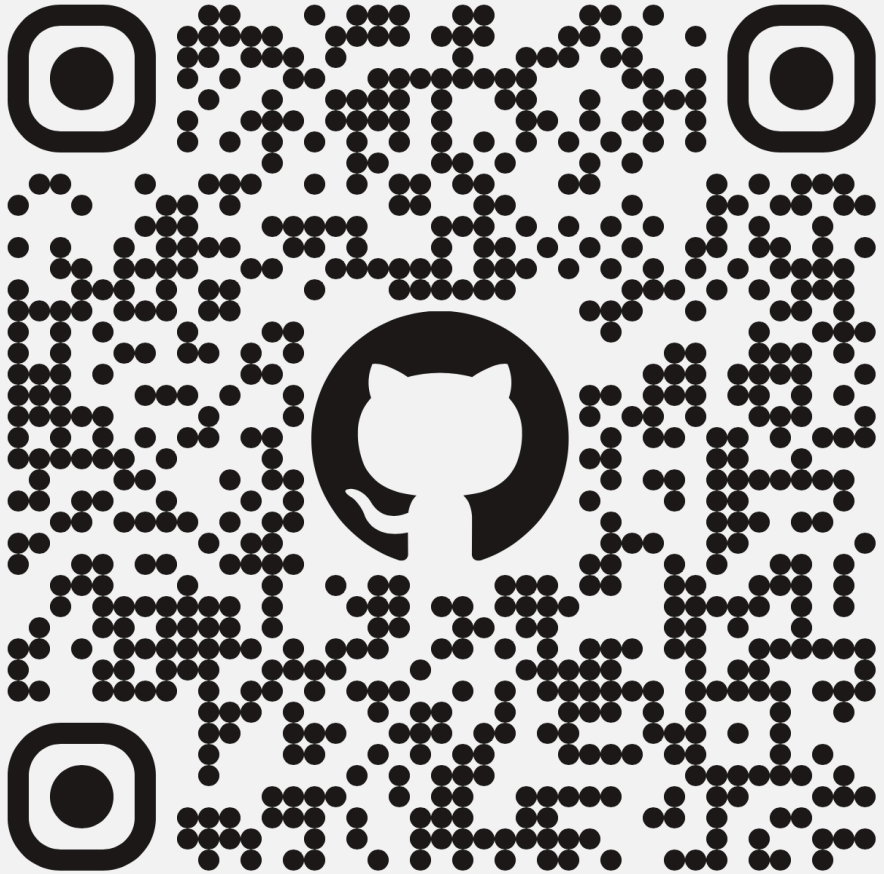
Acknowledgements

- @howardnoakley
- @Morpheus_____
- @birchb0y
- @philofishal
- @patrickwardle
- @gergely_kalman
- @blacktop__
- @oryair1999

Published Tools

- All published tools are available from the following link
- https://github.com/FFRI/PoC-public/tree/main/bhusa2025/xunprotect

FFRI/XPRSecretConfigs
FFRI/RemediationBuilderDSLSpec
FFRI/XPRTestSuite
FFRI/OpenRemediationBuilder
FFRI/ShowProvenanceInfo
FFRI/ProvenanceChecker
FFRI/binja-xpr-analyzer
FFRI/binja-missinglink
FFRI/binja-swift-analyzer



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!

Feedback? Ideas?

@tsunek0h (X)

@tsunekoh@infosec.exchange (Mastodon)

research-feedback@ffri.jp

White paper (in progress)

Icon

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