A Deep Dive into macOS MDM

Jesse Endahl, Fleetsmith
Max Belanger, Dropbox

August 9th, 2018

Introductions

Jesse Endahl — CPO & CSO, *Fleetsmith*

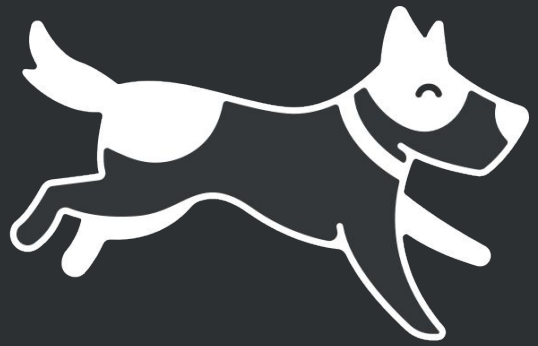
Max Bélanger — Staff Engineer, *Dropbox*

Why we did this research

Fleetsmith automates device setup, OS and app updates, security, and compliance for Apple devices.

We do a lot to ensure our product is **secure by design**.

Our goal was to increase security of DEP & MDM, and raise the bar for MDM vendors.



FLEETSMITH

Agenda

- Basics
- Overview
- Deep Dive
- Vulnerability Details
- Exploit Demo
- Fix Details
- Conclusion and Takeaways

Basics

What is MDM (Mobile Device Management)?

- A way to achieve centralized device management
- Requires an MDM server which implements support for the MDM protocol
- MDM server can send MDM commands, such as remote wipe or “install this config”

Basics

What is DEP (Device Enrollment Program)?

- Allows a device to automatically enroll in pre-configured MDM server the first time it’s powered on
- Most useful when the device is brand new
- Can also be useful for reprovisioning workflows (wiped with fresh install of the OS)

Basics

What is SCEP (Simple Certificate Enrollment Protocol)?

- An relatively old protocol, created before TLS and HTTPS were widespread.
- Gives clients a standardized way of sending a Certificate Signing Request (CSR) for the purpose of being granted a certificate.

Basics

What are Configuration Profiles (aka `mobileconfigs`)?

- Apple’s official way of setting/enforcing system configuration.
- File format that can contain multiple *payloads*.
- Based on property lists (the XML kind).
- “can be signed and encrypted to validate their origin, ensure their integrity, and protect their contents.”

— Page 70, iOS Security Guide, January 2018.

Overview — Entities

- Apple
- Reseller (includes Apple Retail)
- MDM vendor
- Customer
- Device

Overview — Protocols

MDM

- Combination of APNs (Apple servers) + RESTful API (MDM vendor servers)
- Communication occurs between a device and a server associated with a device management product
- Commands delivered in `plist`-encoded dictionaries
- All over HTTPS. MDM servers can be pinned.

Overview — Protocols

MDM Authentication

- Push notification device token (APNs)
- Client certificate (MDM server)

Overview — Protocols

DEP

- 3 APIs: 1 for resellers, 1 for MDM vendors, 1 for device identity (undocumented)
  - More modern and JSON based (vs. plist)
- Today we'll focus on the MDM ↔ Apple *aka* the DEP "cloud service" API

Overview — Protocols

DEP “cloud service” API

- RESTful
- Two main uses:
  - sync device records *from* Apple to the MDM server
  - sync “DEP profiles” *to* Apple from the MDM server (delivered by Apple to the device later on)

Overview — Protocols

DEP “cloud service” API

- A DEP “profile” contains:
  - MDM vendor server URL
  - Additional trusted certificates for server URL (optional pinning)
  - Extra settings (e.g. which screens to skip in Setup Assistant)
- Authentication: OAuth 1.0a token

Overview — Protocols

SCEP

- RESTful
- In lieu of TLS/HTTPS, **PKCS#7** signed data is relied upon to ensure message integrity
- Supports concept of a “Challenge Password” for authenticating the enrollment request (SCEP CSRs)

Overview — Establishment of trust

- **Three legal entities involved**: Apple, MDM vendor, customer
- Trust must be established for *both* DEP and MDM between these parties
- Once trust is established, MDM vendor is granted right to:
  - send MDM push notifications to devices
  - manage device records and “DEP profiles”

Overview — Establishment of trust

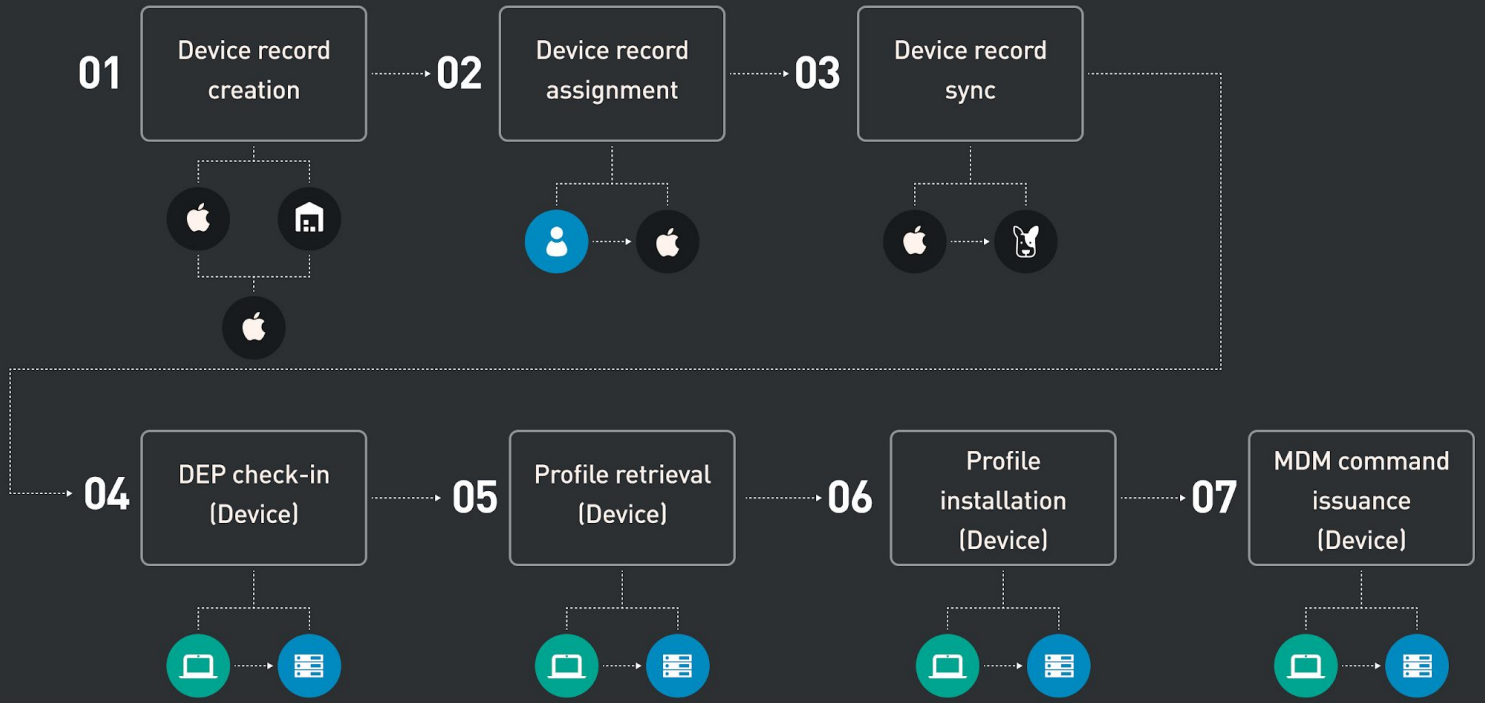
Some differences:

- With MDM, what’s being granted to the MDM vendor by Apple is an APNs certificate.
- With DEP, what’s being granted to the MDM vendor by Apple is an OAuth token.

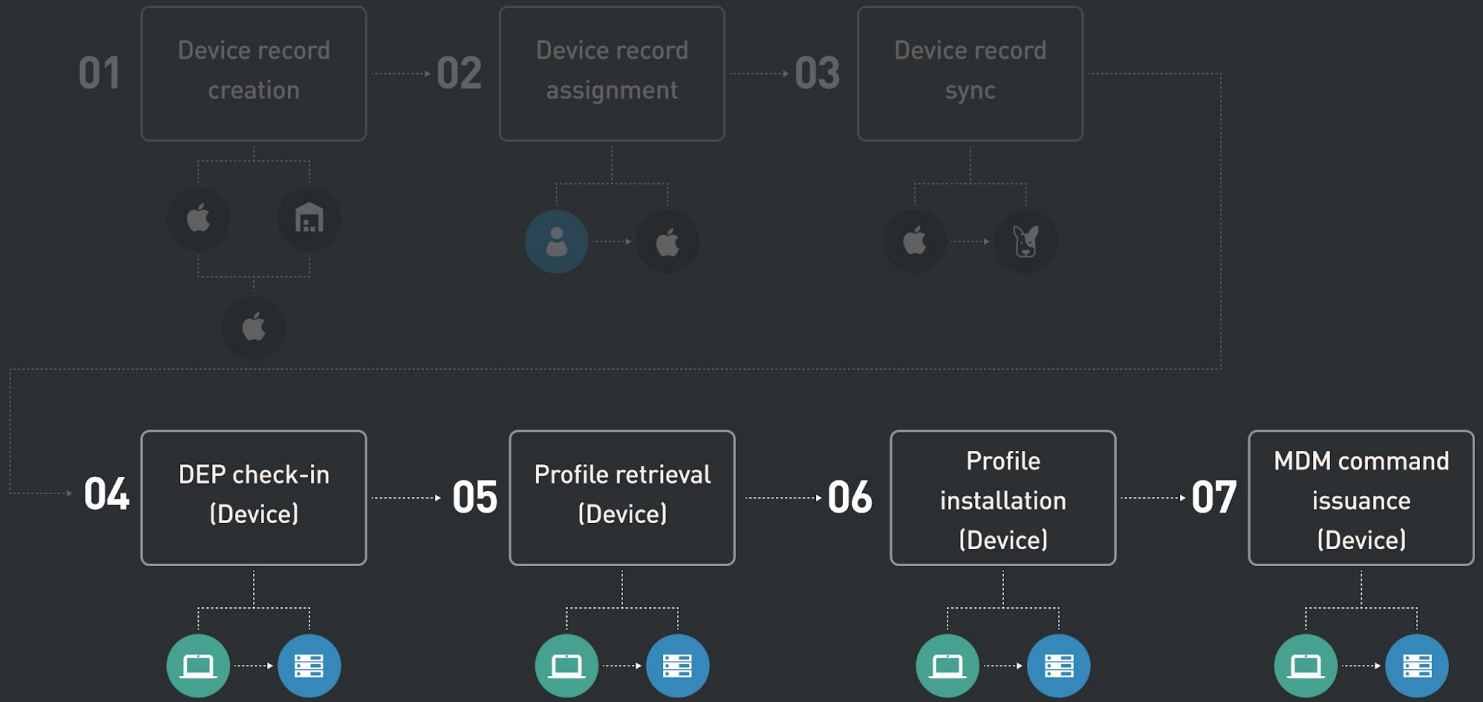
Overview — Putting it all together

1. Device record creation (Reseller, Apple)
2. Device record assignment (Customer)
3. Device record sync (MDM vendor)
4. DEP check-in (Device)
5. Profile retrieval (Device)
6. Profile installation (Device)
    a. incl. MDM, SCEP and root CA payloads
7. MDM command issuance (Device)

Device bootstrap overview



Device bootstrap overview



Deep Dive

- The latter parts of this process (steps 4–7) involve the device itself
- macOS interacts directly with Apple and MDM vendor servers
- **Scenario**: **A user unboxes a brand new MacBook previously configured by their employer to set itself up automatically via DEP + MDM**

Deep Dive - Architecture

- DEP and MDM enrollment involves many agents, daemons
- `ConfigurationProfiles.framework` provides abstraction for the enrollment process
- Exports functions that map well to high-level “steps” of enrollment process

Welcome

In just a few steps, you can register and set up your Mac.

United States
Afghanistan
Åland Islands
Albania
Algeria
American Samoa
Andorra
Angola

Back

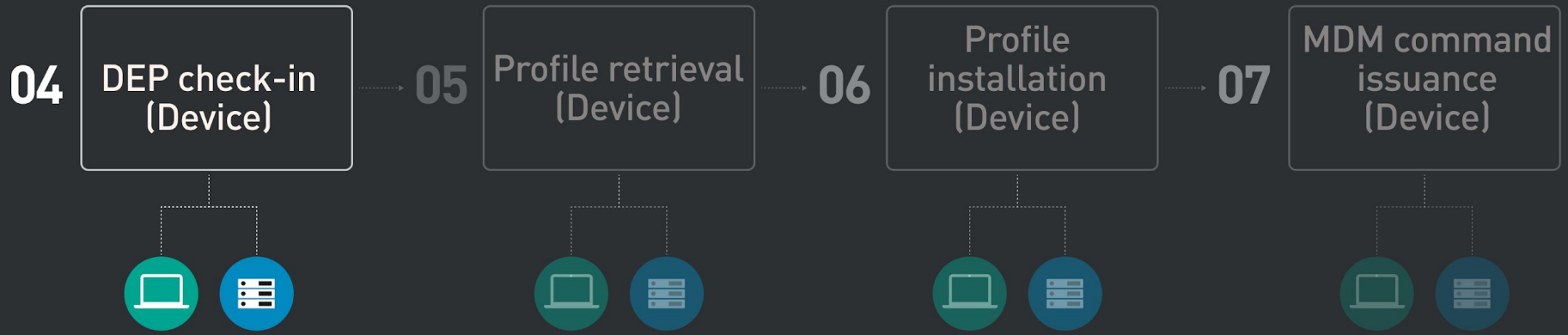
Continue

Setup Assistant.app

Links

ConfigurationProfiles.framework

Device bootstrap overview



Step 4: Getting the Activation Record

- Purpose: determine whether device is DEP enabled
- Activation Record is the internal name for DEP “profile”
- Begins as soon as the device is connected to WAN
- Driven by **`CPFetchActivationRecord`**
- Implemented by **`cloudconfigurationd`** via XPC
  - LaunchDaemon (always runs as `root`)

Setup Assistant.app

Links

ConfigurationProfiles.framework

XPC

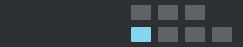
cloudconfigurationd

Step 4: Getting the Activation Record - Absinthe

`MCTeslaConfigurationFetcher` manages process

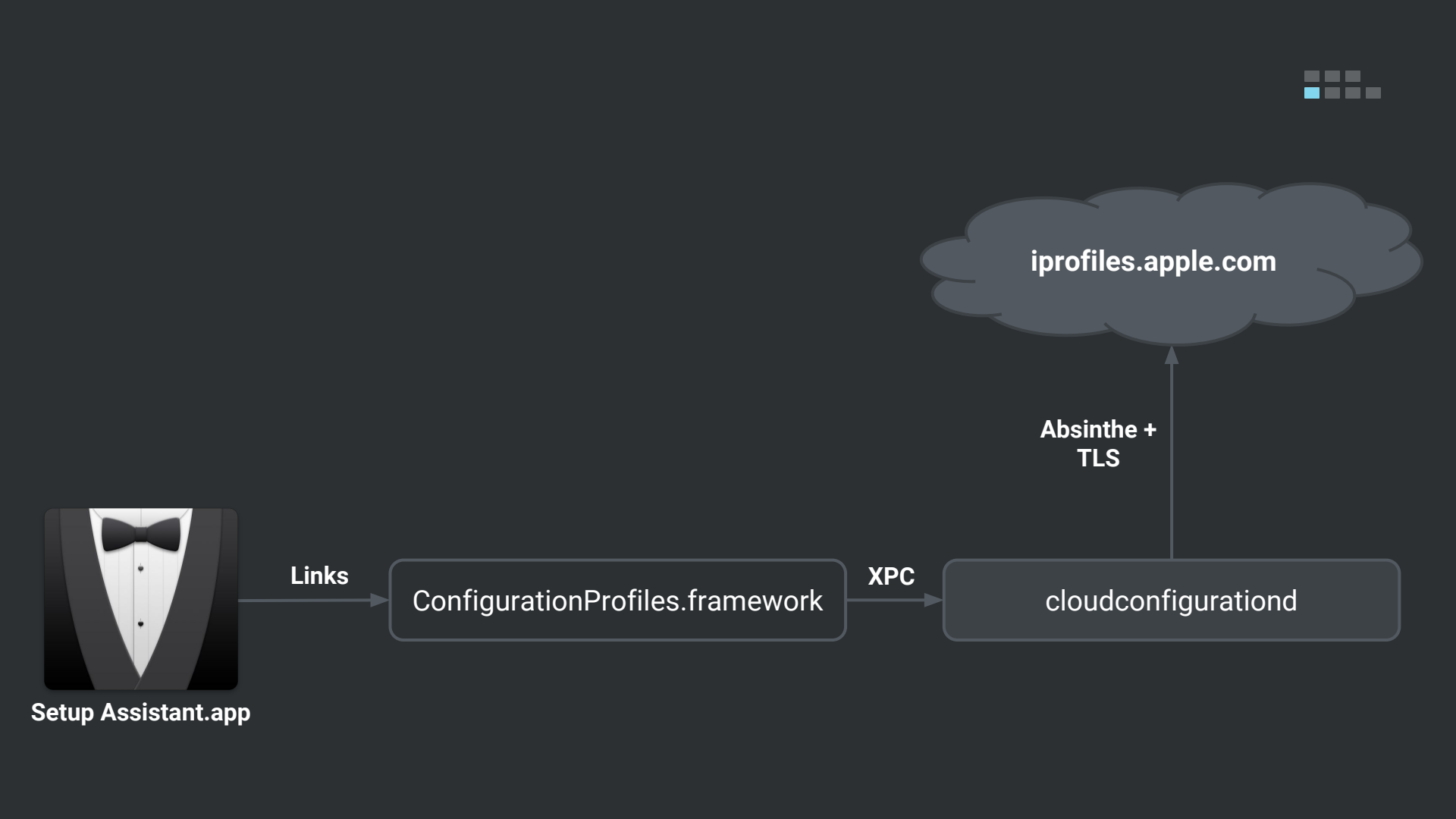
1. Retrieve certificate
   - GET https://iprofiles.apple.com/resource/certificate.cer
2. Initialize state from certificate (`NACInit`)
   - Uses various device-specific data (i.e. Serial Number via IOKit)
3. Retrieve session key
   - POST https://iprofiles.apple.com/session
4. Establish the session (`NACKeyEstablishment`)

Step 4: Getting the Activation Record - Request



- 5. Make the request
  - POST https://iprofiles.apple.com/macProfile
- JSON payload encrypted using Absinthe (**NACSign**)
- All requests over HTTPs, built-in root certificates are used
- Example:

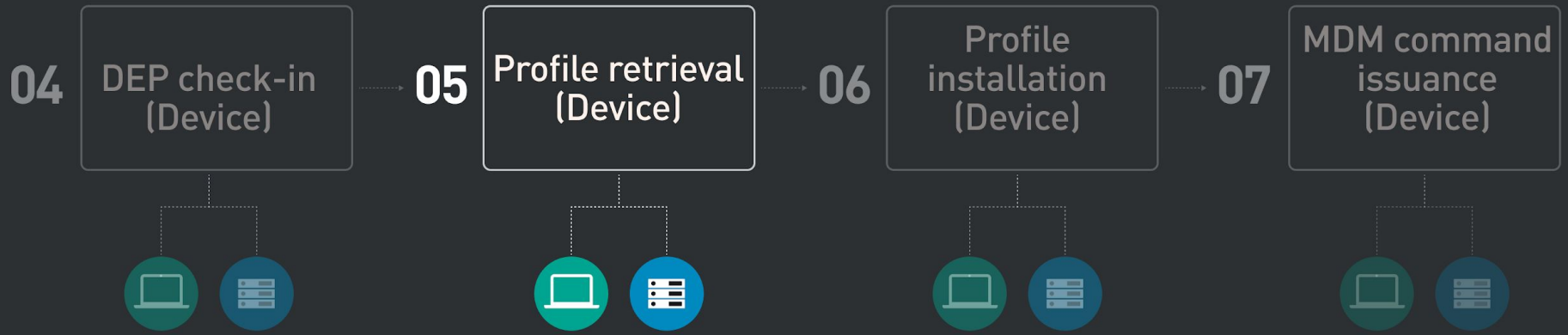
```
{
  “action”: “RequestProfileConfiguration”,
  “sn”: “<device serial number>”
}
```


iprofiles.apple.com
Absinthe +
TLS
Links
ConfigurationProfiles.framework
XPC
cloudconfigurationd
Setup Assistant.app

Step 4: Getting the Activation Record - Response

- In response, the server provides JSON dictionary
- Similar to the DEP profile
  - *Example*: Customization of Setup Assistant
- Two fields matter for the next step:
  - `url`: URL of the MDM vendor host for the activation profile.
  - `anchor-certs`: Array of DER-encoded certificates used as trusted anchors.

Device bootstrap overview



Remote Management

Remote management enables the administrator of “Fleetsmith, Inc.” to set up email and network accounts, install and configure apps, and manage this computer’s settings.

“Fleetsmith, Inc.” can automatically configure your computer.

Learn more about remote management

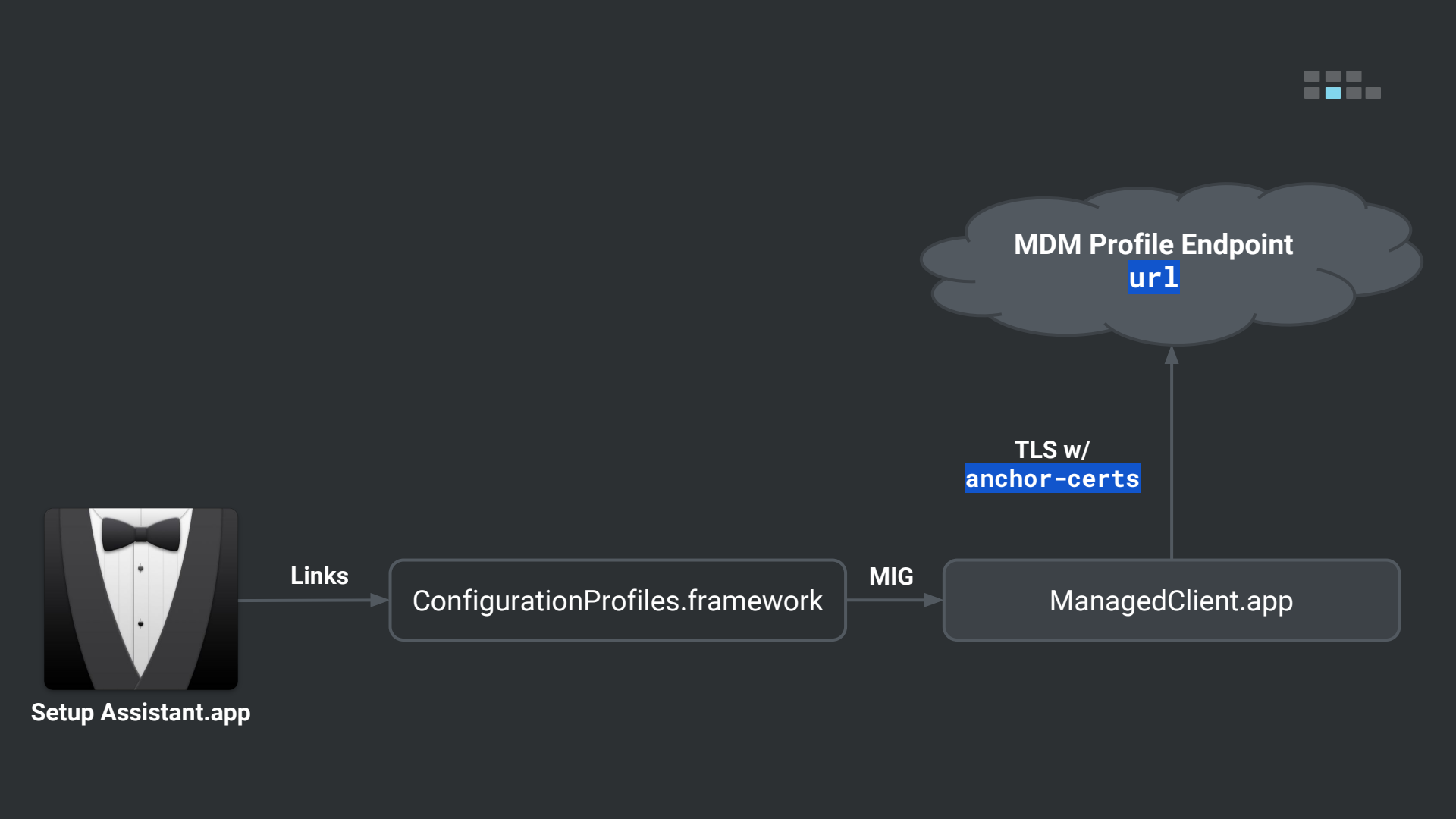
Back Continue

Step 5: Getting the Activation Profile

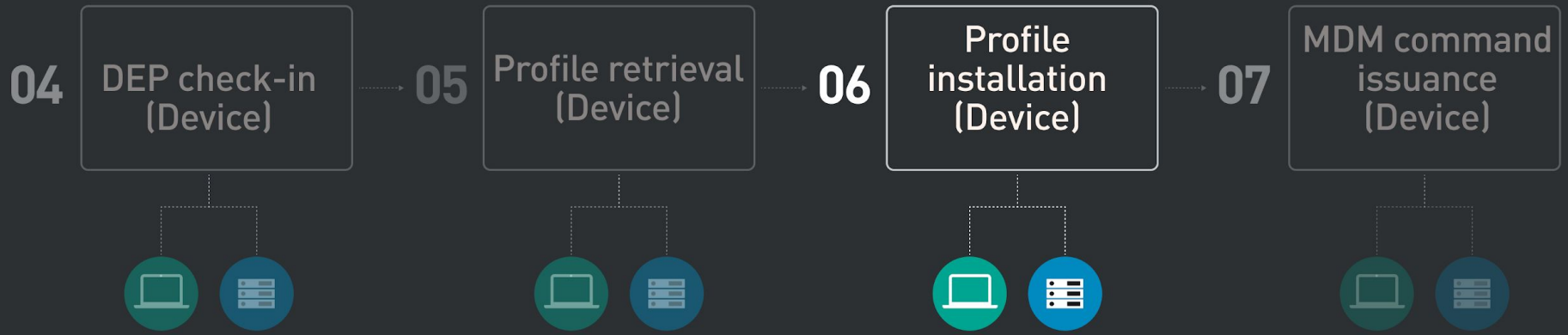
- Activation Profile = DEP-delivered configuration profile
- Same as a regular profile: includes multiple payloads
- Begins when user clicks “next”
- Driven by `CPGetActivationProfile`
- Implemented by `ManagedClient.app` over MIG
  - LaunchDaemon (as root) with auxiliary per-user LaunchAgent (as user)
  - Implementation is named `mcxSvr_cloudconfiguration`

Step 5: Getting the Activation Profile

- Request sent to `url` provided in DEP profile.
- Anchor certificates are used to evaluate trust *if provided*.
  - *Reminder*: the `anchor_certs` property of the DEP profile
- Request is a simple `.plist` with device identification
  - *Examples*: UDID, OS version.
- CMS-signed, DER-encoded
- Signed using the device identity certificate (from APNS)
- Certificate chain includes expired **Apple iPhone Device CA**


MDM Profile Endpoint
url
TLS w/
anchor-certs
Links
ConfigurationProfiles.framework
MIG
ManagedClient.app
Setup Assistant.app

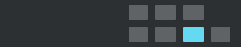
Device bootstrap overview



Step 6: Installing the Activation Profile

- Once retrieved, profile is stored on the system
- This step begins automatically (if in setup assistant)
- Driven by `CPInstallActivationProfile`
- Implemented by `mdmclient` over XPC
  - LaunchDaemon (as root) or LaunchAgent (as user), depending on context

Step 6: Installing the Activation Profile



- Configuration profiles have multiple payloads
- *Installation*: loop to install each payload in the profile
- Framework has a **plugin-based architecture** for installing profiles
- Each payload type is associated with a plugin
  - Can be XPC (in framework) or classic Cocoa (in `ManagedClient.app`)
- Example:
  - Certificate Payloads use `CertificateService.xpc`

Step 6: Installing the Activation Profile

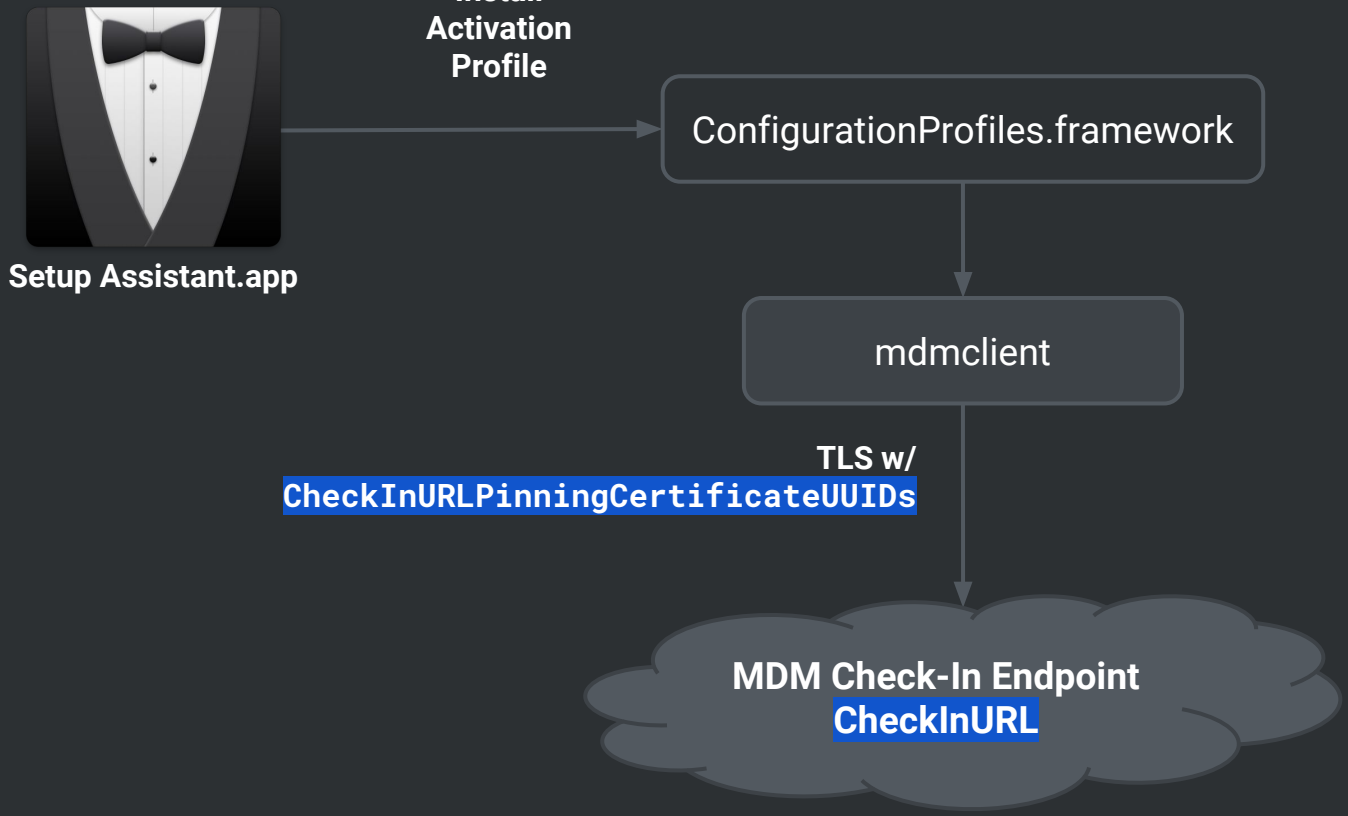
- Typically, activation profile provided by an MDM vendor will include the following payloads:
  - `com.apple.mdm`: to enroll the device in MDM
  - `com.apple.security.scep`: to securely provide a client certificate to the device.
  - `com.apple.security.pem`: to install trusted CA certificates to the device's System Keychain.

Step 6: Installing the Activation Profile

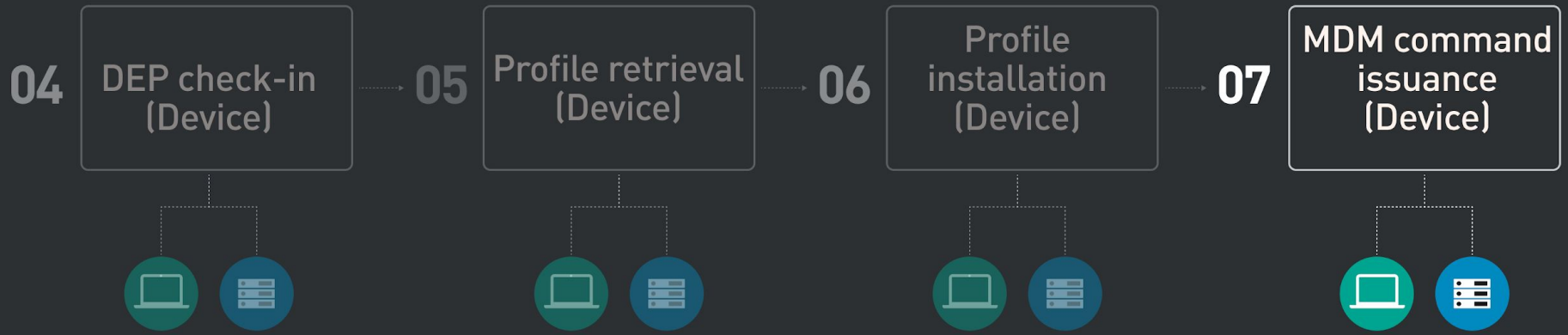
- Installing the MDM payload equivalent to **MDM check-in**
- Configures the device for MDM
- Payload contains key properties:
  - MDM Check-In URL (`CheckInURL`)
  - MDM Command Polling URL (`ServerURL`) + APNs topic to trigger it
- To install MDM payload, request is sent to `CheckInURL`
- Implemented in `mdmclient`

Step 6: Installing the Activation Profile

- MDM payload can depend on other payloads
- Allows requests to be pinned to specific certificates:
  - *Property*: `CheckInURLPinningCertificateUUIDs`
  - *Property*: `ServerURLPinningCertificateUUIDs`
  - Delivered via PEM payload
- Allows device to be attributed with an identity certificate:
  - *Property*: `IdentityCertificateUUID`
  - Delivered via SCEP payload

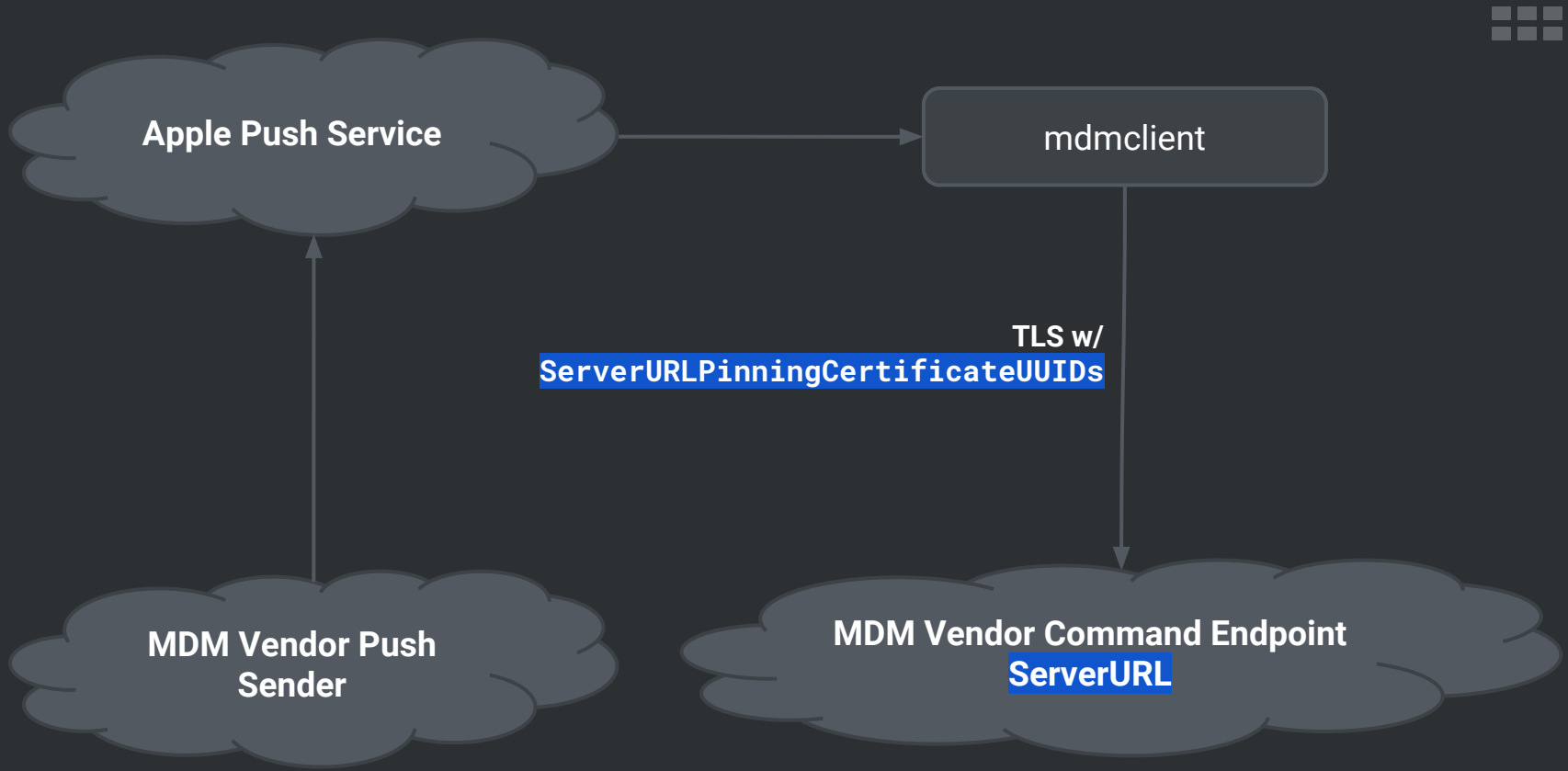

Install
Activation
Profile
Setup Assistant.app
ConfigurationProfiles.framework
mdmclient
TLS w/
CheckInURLPinningCertificateUUIDs
MDM Check-In Endpoint
CheckInURL

Device bootstrap overview



Step 7: Listening for MDM commands

- After MDM check-in is complete, vendor can issue push notifications using APNs
- Upon receipt, handled by **`mdmclient`**
- To poll for MDM commands, request is sent to **`ServerURL`**
- Makes use of previously installed MDM payload:
  - **`ServerURLPinningCertificateUUIDs`** for pinning request
  - **`IdentityCertificateUUID`** for TLS client certificate


Apple Push Service
mdmclient
TLS w/
ServerURLPinningCertificateUUIDs
MDM Vendor Push Sender
MDM Vendor Command Endpoint
ServerURL

Vulnerability: InstallApplication

- Wide range of commands available
- Very common: `InstallApplication`
- Allows remote, silent installation of an application on the device
- Implemented using App Store infrastructure

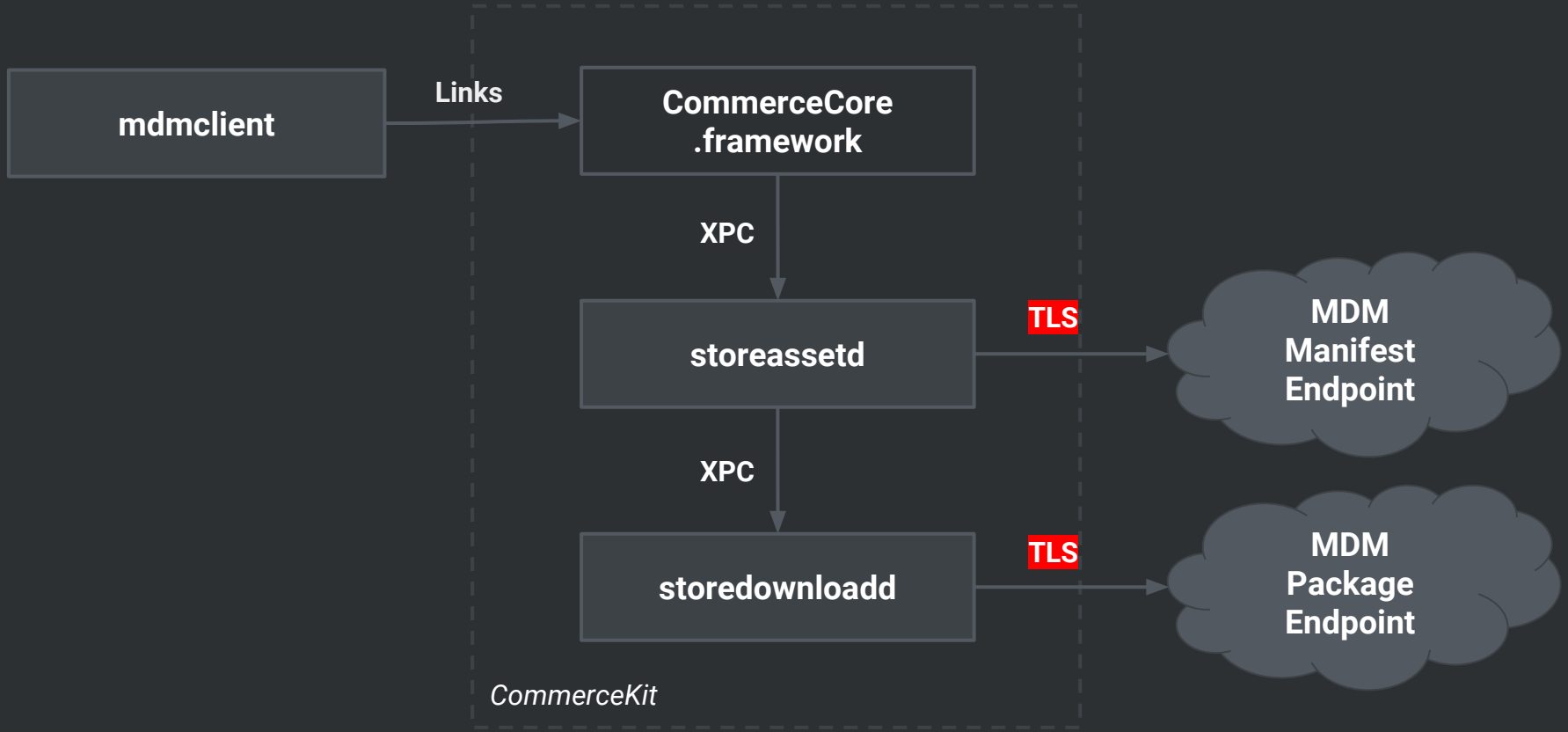
Vulnerability: InstallApplication

- The command request contains the URL to a **manifest**:
  - The manifest describes the application package
  - The manifest is encoded as a property list (XML)
- The manifest includes the URL to the package to install
- On macOS, this points to a signed distribution package (`.pkg`)

Vulnerability: InstallApplication

- `mdmclient` uses CommerceCore.framework, calling **`CKMDMProcessManifestAtURL`**
- *CommerceKit*: contains various “services”, each backed by a LaunchAgent/LaunchDaemon
  - **`storeassetd`**: downloads and processes the manifest, queuing downloads for each specific package
  - **`storedownloadd`**: downloads and installs the package

Vulnerability: InstallApplication



Vulnerability: InstallApplication

- MDM and Store are two separate components
  - Different threat models
- **`mdmclient`** is not evaluating the trust of the manifest URL
  - In fact, **`StoreFoundation`**'s core networking classes used in this scenario do not appear to evaluate trust at all
  - Allows MITM attack
- *Example*: State could target a specific organization by MITMing commands from the MDM vendor. Not limited to DEP.

Demo

- Simulate malicious ISP or state actor via Internet Sharing (on macOS)
  - Proxy all traffic from new device using `mitmproxy`
- Simulate compromised CA by using valid cert from Fleetsmith's CA
- Intercept request for manifest during device's first boot
- Serve malicious manifest, causing device to download and install a different `.pkg`
  - Personal Developer ID certificate to sign "malicious" package

Demo

Fix: InstallEnterpriseApplication

- New command: `InstallEnterpriseApplication`
- Includes new properties to control trust evaluation
- Available in macOS 10.13.6 (thus also 10.14)
  - Thus, **time delay for new hardware to support fix**
- *Important*: Requires MDM vendor to adopt new command

Fix: InstallEnterpriseApplication

- Can pin manifest request to specific anchor certificates:
  - *Property*: `ManifestURLPinningCerts`
- Can ensure certificate revocation checks are performed:
  - *Property*: `PinningRevocationCheckRequired`

Fix: InstallEnterpriseApplication

- Under the hood: **`CKMDMProcessManifestAtURL`** now takes in a dictionary of options
  - `_CKMDMManifestOptionPinCertificates`
  - `_CKMDMManifestOptionPinningRevocationCheckRequired`
- **`StoreFoundation`** networking objects now evaluate trust
  - Uses standard **`NSURLConnection`** delegate method for authentication challenges

Takeaways

- Complex system with many moving parts
- Some have different threat models → bugs can appear between components
- Important to perform holistic “systems level” security reviews
- Security is often dependent on the MDM vendor → vendors can do more

Recommendations for Apple

- Fully document the security model for DEP & MDM, including the role of the Apple iPhone Device CA
- Require pinning at each step (currently optional)
- Require any Configuration Profile containing sensitive data (e.g. Wi-Fi password) to be both signed and encrypted (currently optional)
- Make factory installed OS version and build number available via DEP APIs

MDM Vendor Security Checklist

- **Pin at every step of the process**
  - Step 4: Pin MDM **anchor_certs** in DEP profile
  - Step 5: Pin MDM URLs in payload (`/checkin`, `/commands`)
  - Step 7: Pin using **`InstallEnterpriseApplication`**
  - All networking calls from agent binary (if one exists)
- **Use SCEP**
  - Avoid generating private keys server-side
  - Use SCEP “Challenge Password” (HMAC works well)

MDM Vendor Checklist

- **Configuration Profiles**
  - All should be signed
  - All that contain sensitive data should be signed and **encrypted**, using the device’s public key
- **Encrypt sensitive customer data at rest**
  - e.g. WiFi passwords

Disclosure timeline

- First disclosed to Apple on **April 28th**, acknowledged May 2nd
- `InstallEnterpriseApplication` announced in “What's New in Managing Apple Devices” session at WWDC on **June 7th**
- MDM Protocol Reference documentation updated to include `InstallEnterpriseApplication` on July 5th
- Fix is introduced with macOS 10.13.6 on **July 9th**

Conclusion

Thank you!

- Shout out to Fleetsmith cofounder **Stevie Hryciw** for first discovering the vulnerability + his assistance on early research
- Shout out to **@groob @bruienne @mikeymikey @jessecpeterson** for their research, open source work, and contributions to the Mac security community
- Shout out to **Apple** for their quick reaction and courteous response as well as the great work their security engineering team is doing to continually improve platform security for both iOS & macOS

Resources

- **Twitter**:
  - @jesseendahl
  - @maxbelanger
- **Fleetsmith**: fleetsmith.com