RSAConference2019
San Francisco | March 4–8 | Moscone Center

SESSION ID: CSV-W12

Red Team View: Gaps in the Serverless Attack Surface.

Mike Cotton

SVP Research & Development
Digital Defense Inc.

Overview

- Shift in Technology -> Shift in Tactics
  - Serverless Another Major Phase of the Web
  - From a Security Perspective Maybe Most Significant Shift
  - We've Already Seen The Beginnings of This
    - Rampant Leaking of Data Cloud Storage Services
    - Didn't Make the Technology 'Customer Proof'
  - This Talk Aims to Help Close the Gap
    - Red Team: (Assessment / PT) -> New Tactics You Can Use
    - Blue Team: (Dev / DevOps) Ways to Better Lock Down Your Systems

Shift in Technology => Shift in Tactics

- Not a theoretical talk
  - Going through Real World Examples
  - Technical Details & Demos
  - New Tactics We're Using
- General Components
  - API Lambda / Cloud-Function Layer
  - Looking for Peer Infrastructure
  - Cloud Storage Layer

RSAConference2019

Problems

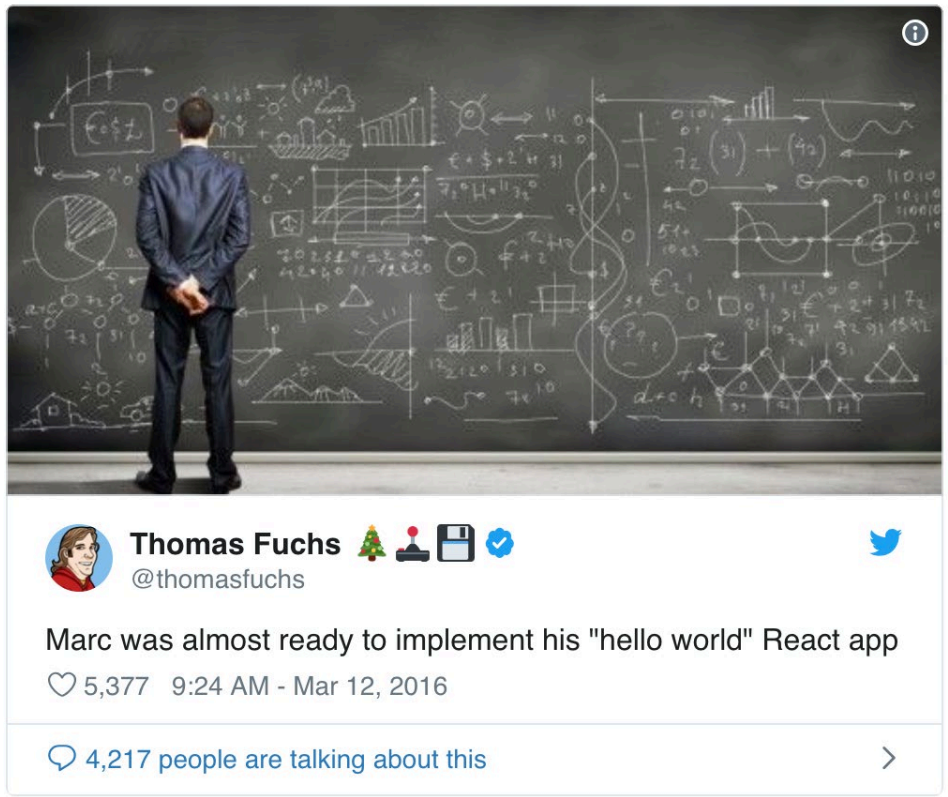
API Security / Modern MVVM Frameworks

- The LAMP Era Had Some Advantages
  - Hard to Believe but this is True!
  - Server-Side Rending Masked API and App Details
  - Administrative Functions Had to be Inferred from Frameworks
- Modern Javascript Client-Side UI -> API Apps
  - This is the Typical Stack Deployment for Serverless
    - Similar to MEAN / MERN, But with Dynamo and Node Runs in Lambda
    - Android and IOS Mobile Apps: AWS Amplify SDK to API in Cloud
  - API is the remote interface so it's not a black box

Modern Cloud / App - Complexity

Much Steeper Learning Curve

- Create-React-App HelloWorld
  - Plus Amplify SDK init APIS
  - 33,000+ Files in HelloWorld Dir
  - 85 Files Excluding node_modules
  - Many Things Being Done For You



API Security / Modern MVVM Frameworks

- Sometimes Partial Security Controls Can Be Problematic
  - Good Enough to Pass Muster With Dev / Test / DevOps
  - Lulls you into sense of security
  - Keep out random pranksters / sweeps
  - Determined attackers find exposed systems
  - Bottom Line – It's important to understand the details

PenTest Experience : Undocumented API's

- The Basis for Most Custom Discoveries
  - Target the Infrastructure / Framework
  - Target the API's Directly
- Security Disclosures
  - Our Zero-Day Advisories (Over 20 in Last 2 Years)
    - VMWare, Sonicwall, Veritas, ManageEngine, EMC,NUUO, Avaya, etc. Mostly Appliance API abuse.
    - Bottom Line: Sophisticated Developers Still Have Challenges w/ This.
  - Similarities to What We See in the Cloud/Serverless: API Abuses
    - Theme is: Enumerate -> Tamper -> Obtain Admin Access
    - Complexity of SaaS Disclosures

Types of Protection (AWS-Lambda / AWS-Mobile)

- ○ **Private**
  This API is accesible only to those users who have signed in.
- ◉ **Protected**
  This API is accessible to all your app's users.
- ○ **Public**
  This API can be called by anyone on the web. WARNING: Carefully consider your security requirements before making this choice.

Paths

By default all paths will create a new AWS Lambda function that will echo request parameters in the response. If you want two paths to be handled by the same function, set the Lambda Function Name to be the same. Once this API is created, the edit link in the list of APIs allows you to edit the code of functions backing your paths. We have provided "items" as an example resource path in your API. You may modify "items" and/or create additional resource paths.

In the Details

- Public – You Can Access it with Curl
  - You'll know format just from Javascript Debugger
- Protected
  - Adds in HMAC SHA256 Signing via client token
  - Client Side Key: AccessKeyID, SecretKey obtained via Cognito
- Private
  - Similar Signing Protections to Protected but
  - Couples that with Authentication, SessionKey

Working w/ Protected APIs [DEMO]

```
content-type;host;x-amz-date
f09210c9fd71839289cf5d9f031bd9f56300049d500ca3b6f4585fe0ce82d219
{ '[DEBUG] 48:06.930 Signer': { service: 'execute-api', region: 'us-east-1' } }
{ '[DEBUG] 48:06.931 RestClient - Signed Request: ':
   { method: 'POST',
     url:
      'https://eji0i60n52.execute-api.us-east-1.amazonaws.com/Development/notes',
     host: 'eji0i60n52.execute-api.us-east-1.amazonaws.com',
     path: '/Development/notes',
     headers:
      { 'content-type': 'application/json; charset=UTF-8',
        host: 'eji0i60n52.execute-api.us-east-1.amazonaws.com',
        'x-amz-date': '20181213T204806Z',
        Authorization:
         'AWS4-HMAC-SHA256 Credential=[redacted]1213/us-east-1/execute-api/aws4_request, SignedHeade
rs=content-type;host;x-amz-date, Signa[redacted]8a72a041fb7546beff8820e9b7e0fe50afa061430a40' },
     data: '{"text":"demo","nid":100}',
     body: '{"text":"demo","nid":100}' } }
{ success: 'post call succeed!', url: '/notes', data: {} }

skywarp:demo mcotton$
```

Note On: Key Patterns / AWS-Labs / Git-Secrets

--scan-history

Scans repository including all revisions. When a file contains a secret, the matched text from the file being scanned will be written to stdout and the script will exit with a non-zero status. Each matched line will be written with the name of the file that matched, a colon, the line number that matched, a colon, and then the line of text that matched.

--list

Lists the `git-secrets` configuration for the current repo or in the global git config.

--add

Adds a prohibited or allowed pattern.

--add-provider

Registers a secret provider. Secret providers are executables that when invoked output prohibited patterns that `git-secrets` should treat as prohibited.

--register-aws

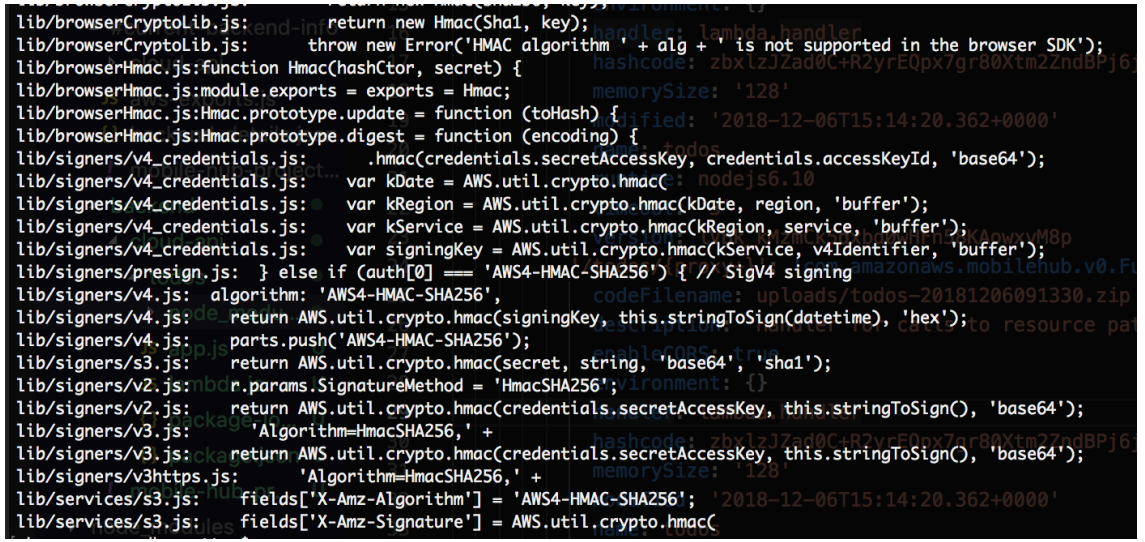
Adds common AWS patterns to the git config and ensures that keys present in `~/.aws/credentials` are not found in any commit. The following checks are added:

- AWS Access Key IDs via `(A3T[A-Z0-9]|AKIA|AGPA|AIDA|AROA|AIPA|ANPA|ANVA|ASIA)[A-Z0-9]{16}`
- AWS Secret Access Key assignments via ":" or "=" surrounded by optional quotes

API SDK Client-Server Signing

HMAC Code Details …

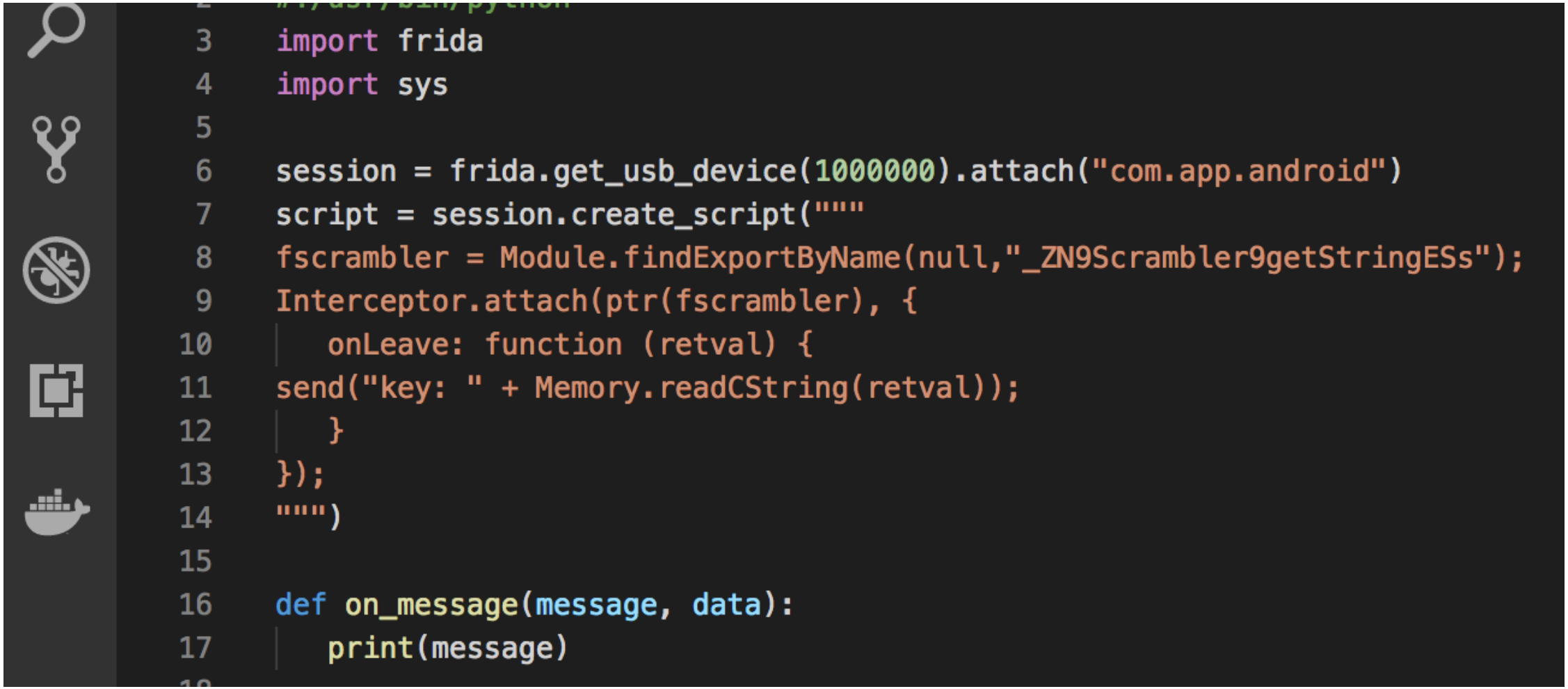
- Various Versions and Configurations of this
  - Wraps both Pre-Auth and Post-Auth on Non Public Functions.
    - AWS This Comes Through Amplify
  - Key Attributes Are Typically
    - Signs Headers / Timestamp / Body / Instance Referenced
    - HMAC SHA256 of the Payload
  - Can Extract Identity Pool / Keys from Client Debuggers



Mobile App -> Protected / Private APIS

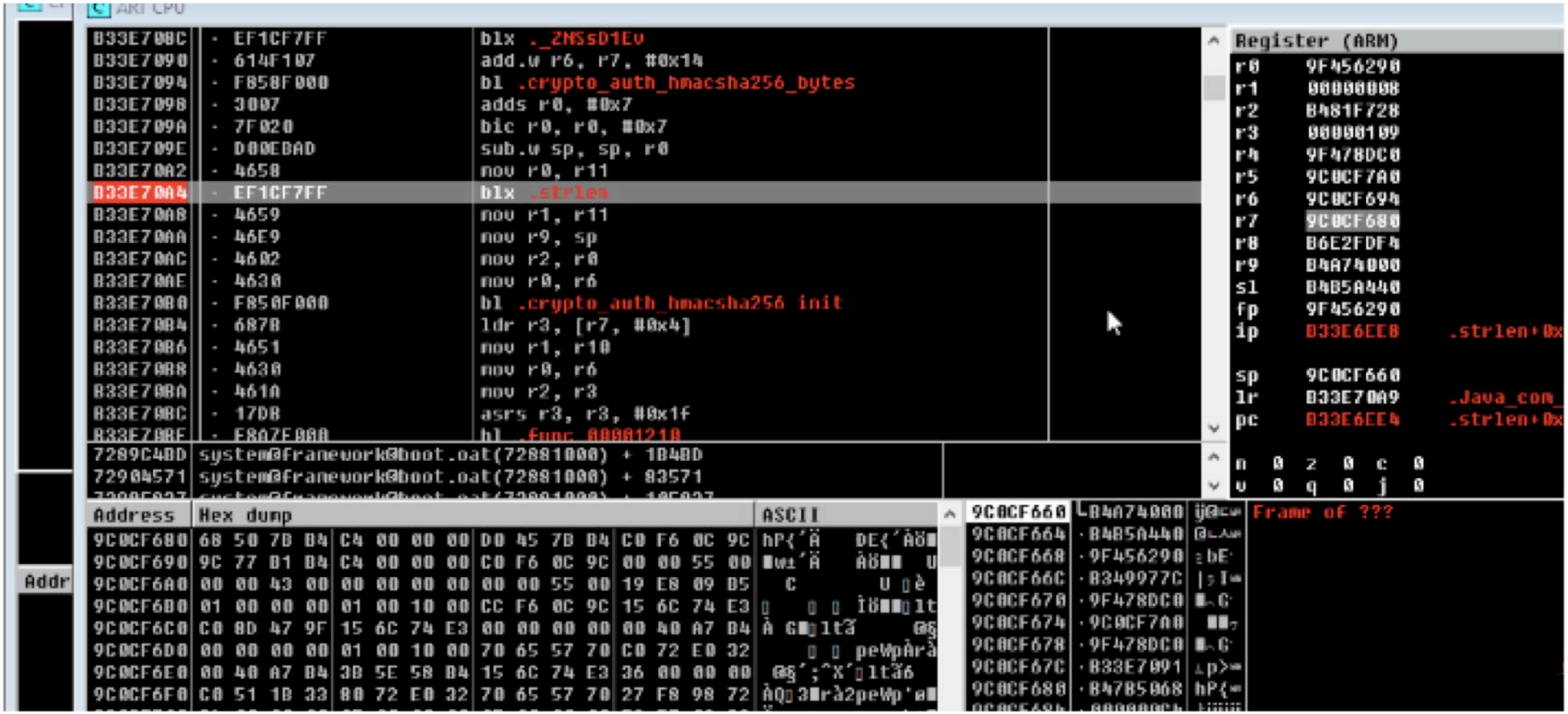
- Mobile Apps: API Enum / Key Retrieval can be Challenging
  - Particularly if SSL Certificate Pinning Enabled
  - Prevents MITM even with an SSL Proxy + Root Cert Install
- Some Tools to Help Here:
  - Frida-Server - Android Debugger Instrumentation Framework
  - ADB: Android Debug Bridge (Standard Component)
  - Genymotion Cloud Android Emulator
    - Useful for Emulating Various OS Versions
    - Can Also Use a Local Device Instead of This

Dumping Keys / API Calls from Apps …



```
3   import frida
4   import sys
5
6   session = frida.get_usb_device(1000000).attach("com.app.android")
7   script = session.create_script("""
8   fscrambler = Module.findExportByName(null,"_ZN9Scrambler9getStringESs");
9   Interceptor.attach(ptr(fscrambler), {
10      onLeave: function (retval) {
11  send("key: " + Memory.readCString(retval));
12      }
13  });
14  """)
15
16  def on_message(message, data):
17      print(message)
```

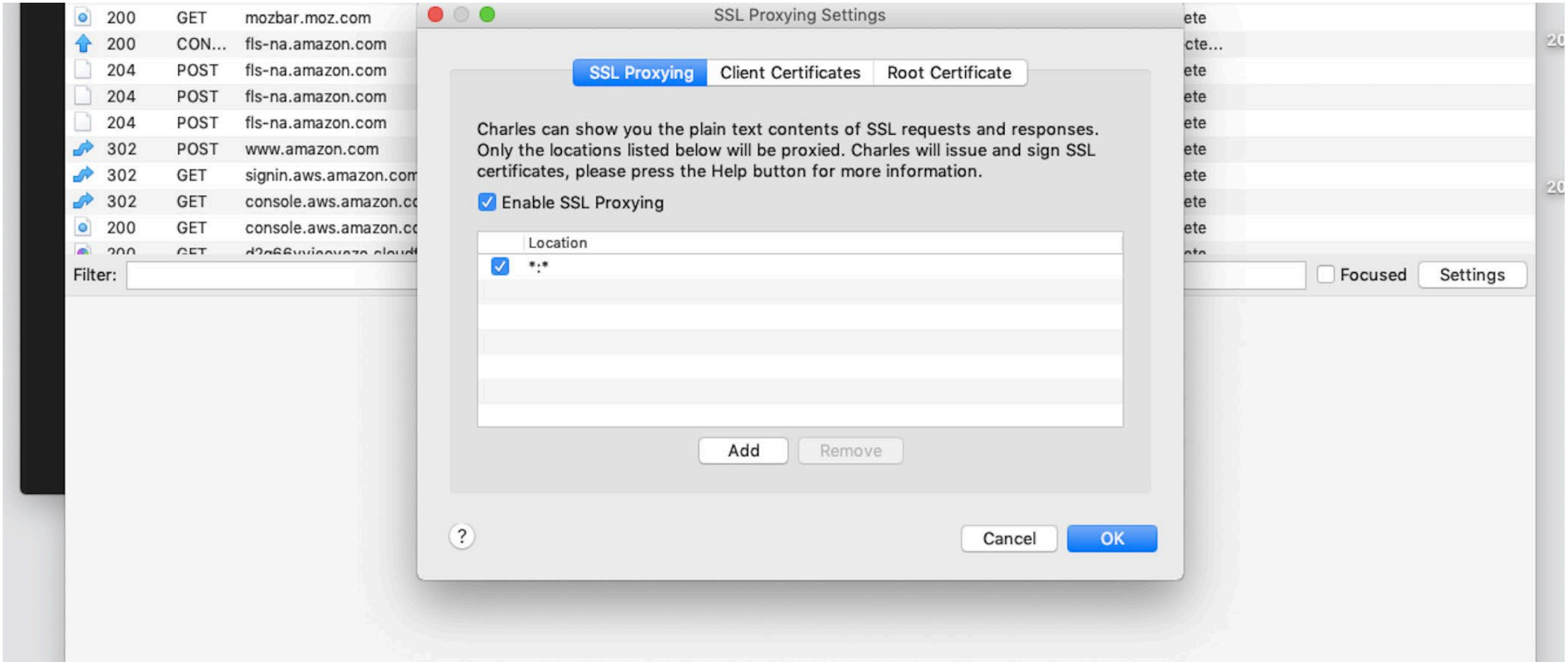
Extracting API Endpoints / Keys (Android) [DEMO]



Recent Experiences

- With Enumeration & Signing Keys
  - Start Tampering with API Calls
- Most Common Patterns
  - Protected Calls which Should Be Private / Limited
    - No Validation Check Other Than Signing Layer
    - Data Leakage / Denial of Wallet Scenarios
    - Use of Privileged Keys in Apps
  - Post-Auth Services With Privilege Abuses
    - Takes Client ID from Request Body Rather then Session
    - Allowing for Queries and Posts on other Users
    - Allows Non-Privileged Users to Execute Admin Calls
      - CreateUserAccount
      - ModifyPrivileges

API Abuse - Real-World [DEMO]



SSL Certificate Transparency Infrastructure

- Built Due to Rogue Certificate Creation
  - Industry Consortium
  - Registrars Public Certificate Log
  - i.e. https://transparencyreport.google.com/https/certificates?hl=en
- For Serverless Discovery replacing Older Mechanisms
  - i.e. Nmap Sweeps / Public CIDR Ownership Lookups
  - We already know the ports, and the hosts are fluid
  - SSL Use Has Exploded w/ Automated Certificates (i.e. LetsEncrypt)
  - Browser and Integrations Problems w/o Trusted Connections

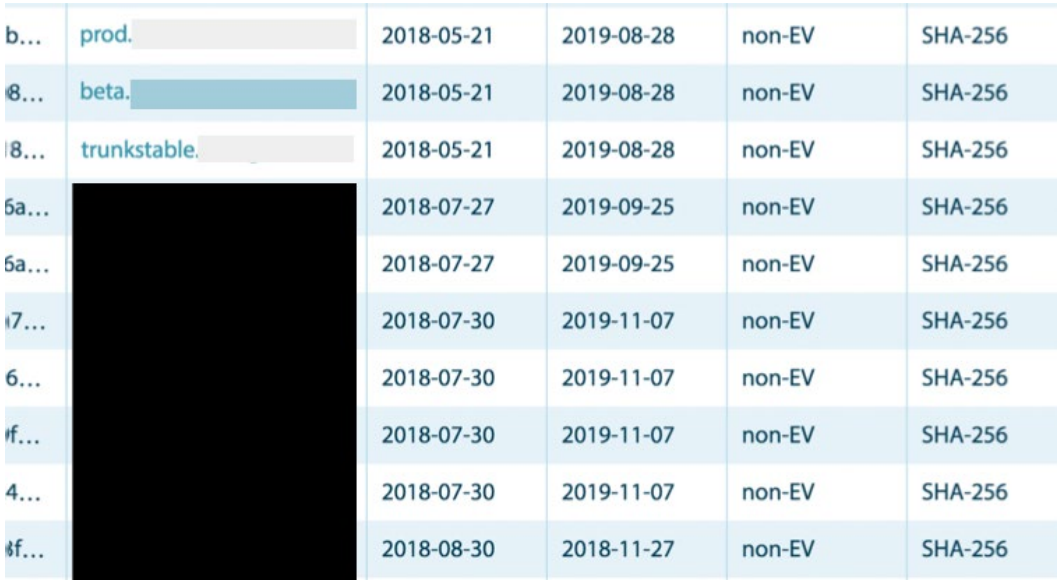
Cloaked Development Infrastructure

- New Trend Seen in Assessment Work
  - Second Layer of Accessible Peer Infrastructure
    - Development / Testing / Staging Purposes
    - Often Times the Weak Point in Primary System Compromise
      - Substantially Weaker / Still w/ Privileged Access Tie-ins.
      - Similar to Kill Chain Mechanics Against Internal Targets
        - Compromise Weaker Nodes -> Gain Creds / Access -> Take Primary
  - Not something traditional architectures have suffered from
    - Traditional Workflow
      - Development / Testing Happened Locally
      - Security Tested / Shipped to Production

Cloaked Infrastructure Discovery

Common Patterns:

- Look for Common Naming Patterns
  - Dev,Prod,Test,Stable,DevStable,etc.
- Dev Instances Often Are:
  - More Open Access / More Open to Trivial User Account Combos
  - Contain Primary Lineup Access Mechanisms
    - Embedded Privileged Accounts
    - Valid API Tokens



b...	prod.	2018-05-21	2019-08-28	non-EV	SHA-256
8...	beta.	2018-05-21	2019-08-28	non-EV	SHA-256
8...	trunkstable.	2018-05-21	2019-08-28	non-EV	SHA-256
6a...	[redacted]	2018-07-27	2019-09-25	non-EV	SHA-256
6a...		2018-07-27	2019-09-25	non-EV	SHA-256
7...		2018-07-30	2019-11-07	non-EV	SHA-256
6...		2018-07-30	2019-11-07	non-EV	SHA-256
f...		2018-07-30	2019-11-07	non-EV	SHA-256
4...		2018-07-30	2019-11-07	non-EV	SHA-256
8f...		2018-08-30	2018-11-27	non-EV	SHA-256

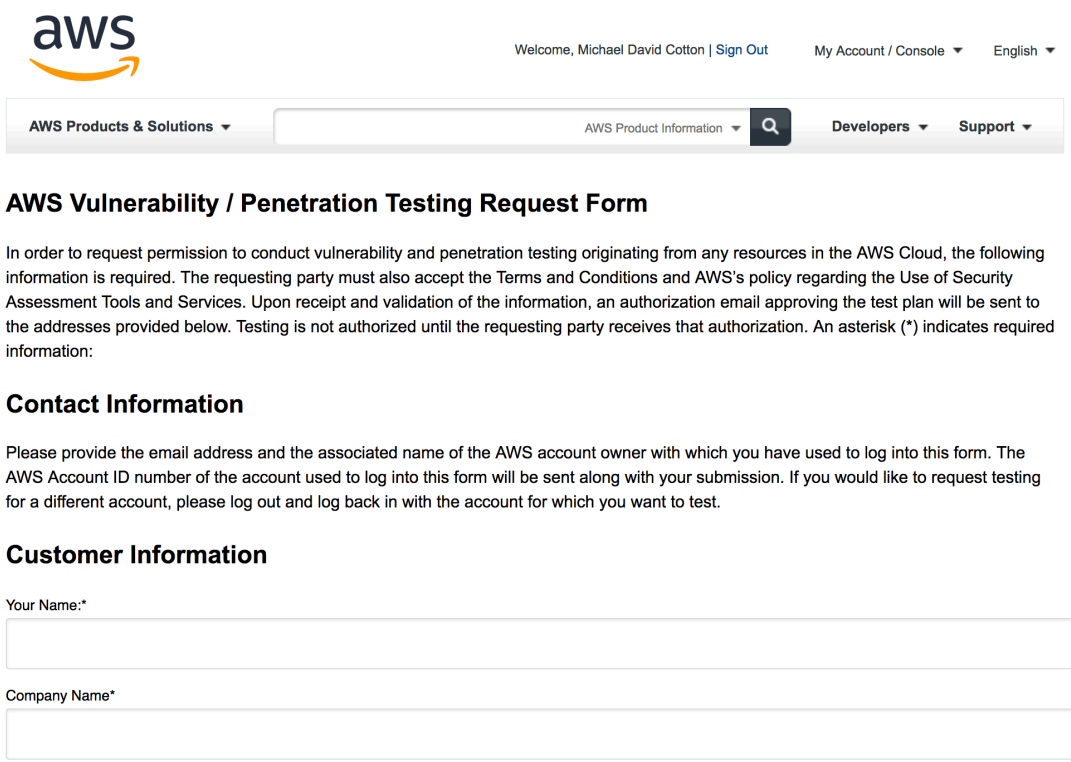
Why This Happens …

- Key Drivers Appear to Be
  - Difficulty of Running the Serverless Service Infrastructure Locally
    - Services AWS Lambda / DynamoDB / Difficult to Simulate Locally for Development Purposes
    - Work Being Done in this Area (i.e. ‘Serverless Framework’)
      - But Production Needs are Running Ahead
  - Desire for a Uniform Working Deployment Model
  - Distributed Workforce Can Complicate Remote Access Controls
  - Not Enough Concern About the Security of Development Test Instances and the Data they Contain.

Proper Scoping …

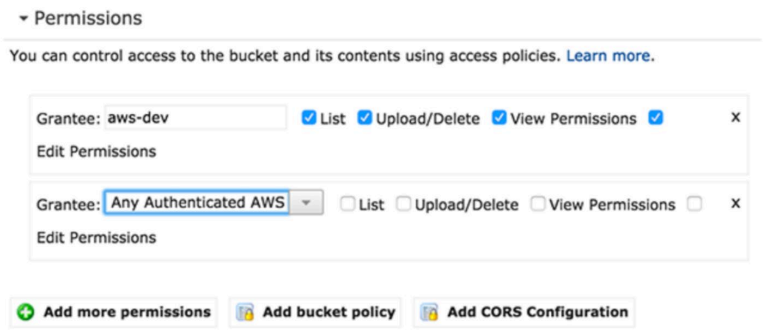
Constant Shifting Targets

- Distributed Nature of Applications Means OSInt Phase need to be beefed up.
  - Need to Scope Application Subcomponents Differently.
  - Consider Changing PT Methodology to Allow for Discovery / Enumeration and Dynamic Scope w/ Client.
  - Have to Avoid Redeployments which can Upend Instances and Permissions



Cloud Storage Layer

- A lot already written on this
  - Layering on personal experiences from assessment work
    - Buckets with permissions issues continue to be a huge problem.
    - Real world: seeing issues > 25% of the time they are in play.
    - Mostly likely subcomponent to be compromised
    - Can totally unwind an otherwise secure stack
  - Since 2016 Changes w/ Certificate Manager
    - Often encounter Bucket -> Cloudfront -> Webroot
    - PKI allows for targeting buckets that were previously obscure
    - Improvements made to permissions in this area, but people still struggling



Cloud Storage Layer

- Typically we're looking for:
  - Application API Keys
    - Generated by developers or users for automated access
    - By design are long-lived and do not require two-factor
    - Typically from privileged accounts
    - Rarely restricted to a handful of calls, typically have full user rights.
  - Passwords in configuration files
  - Infrastructure Keys Which can
    - Don't target the app but the cloud infrastructure itself
    - The sorts of configs your local utility scripts use

RSAConference2019

Solutions

Apply, Locking Things Down …

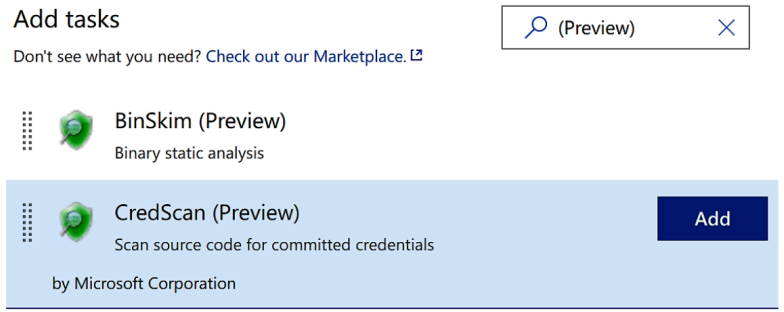
- Next two-weeks you should:
  - Identify the serverless infrastructure used by your organization.
    - (Both production systems and development / test systems)
    - Use Billing Reports to Help Hunt Down any Rogue Lineups
  - Implement a repository-credentials protection system
    - This is important even if you have a private/restricted repo.
    - Major Cloud Vendors Have Projects to Help You with This
      - Azure: CredScan Built Task Utility (https://secdevtools.azurewebsites.net/helpcredscan.html)
      - Amazon: Git-Secrets Github Project (https://github.com/awslabs/git-secrets)

Repository Key Protections

When Implementing Protections:

- Important to:
  - Scan Existing Repositories when First Implementing Protections
  - Ensure that Contrib History Does not Still Retain ‘Deleted’ Credentials
  - Invalidate any keys discovered, even in private repositories.
  - Check all new checkins / reject bad commits in real-time.

1. Open your team project from your Azure DevOps Account.
2. Navigate to the **Build** tab under **Build and Release**
3. Select the Build Definition into which you wish to add the CredScan build task.
   - New - Click **New** and follow the steps detailed to create a new Build Definition.
   - Edit - Select the Build Definition. On the subsequent page, click **Edit** to begin editing the Build Definition.
4. Click **+** to navigate to the **Add Tasks** pane.
5. Find the CredScan build task either from the list or using the search box and then click **Add**.



AWS-Labs / Git Secrets (Discussion)

Prevents you from committing passwords and other sensitive information to a git repository.

Contents

Synopsis

```
git secrets --scan [-r|--recursive] [--cached] [--no-index] [--untracked] [<files>...]
git secrets --scan-history
```

Description

`git-secrets` scans commits, commit messages, and `--no-ff` merges to prevent adding secrets into your git repositories. If a commit, commit message, or any commit in a `--no-ff` merge history matches one of your configured prohibited regular expression patterns, then the commit is rejected.

Installing git-secrets

Apply, Continued …

- Within one Month:
  - Verify there is no publically accessible non-production infrastructure
    - Run Certificate Transparency Queries on Your Systems / Domains
      - https://transparencyreport.google.com/https/certificates?hl=en
      - Look For Common Development Patterns (prod,dev,test,stable)
  - Sit with your development team
    - Gain an understand of your current API security controls
      - Map out all the endpoints for each app, and their access levels
        - Discuss use of public / private / protected APIs
        - Make a plan to restrict endpoints as necessary in new releases.
      - Talk through potential rogue-client & app scenarios

Apply, Continued …

- Within 3 months
  - Move any development or test instances and lineups:
    - To alternate domains not easily associated with your primary
    - Regenerate associated SSL certificates and configurations.
  - Ensure all privileged API keys are accounted for and have appropriate access limits.
    - Use IAM users and roles and to grant access and set least privilege.
      - Many times this is loose during development push, not cleaned up later on
    - Ensure there is a hard line between development key / instance rights and production keys / instances.
    - Invalidate and re-issue all keys / roles if original controls were not sufficient.

RSAConference2019

Questions? / Thank You!

Contact: Mike.Cotton@digitaldefense.com