Bai7 GmbH

Sven Schleier B.Eng.

August 2025

The Mobile Playbook: A guide to iOS & Android App Security - Day 2

Android Training

Preparation

Github

For the training today you need to use a free Github account to follow the labs. If you don’t have one, please create one now.

https://github.com/

Sven Sch

© 2025 by Bai7 GmbH | All rights reserved

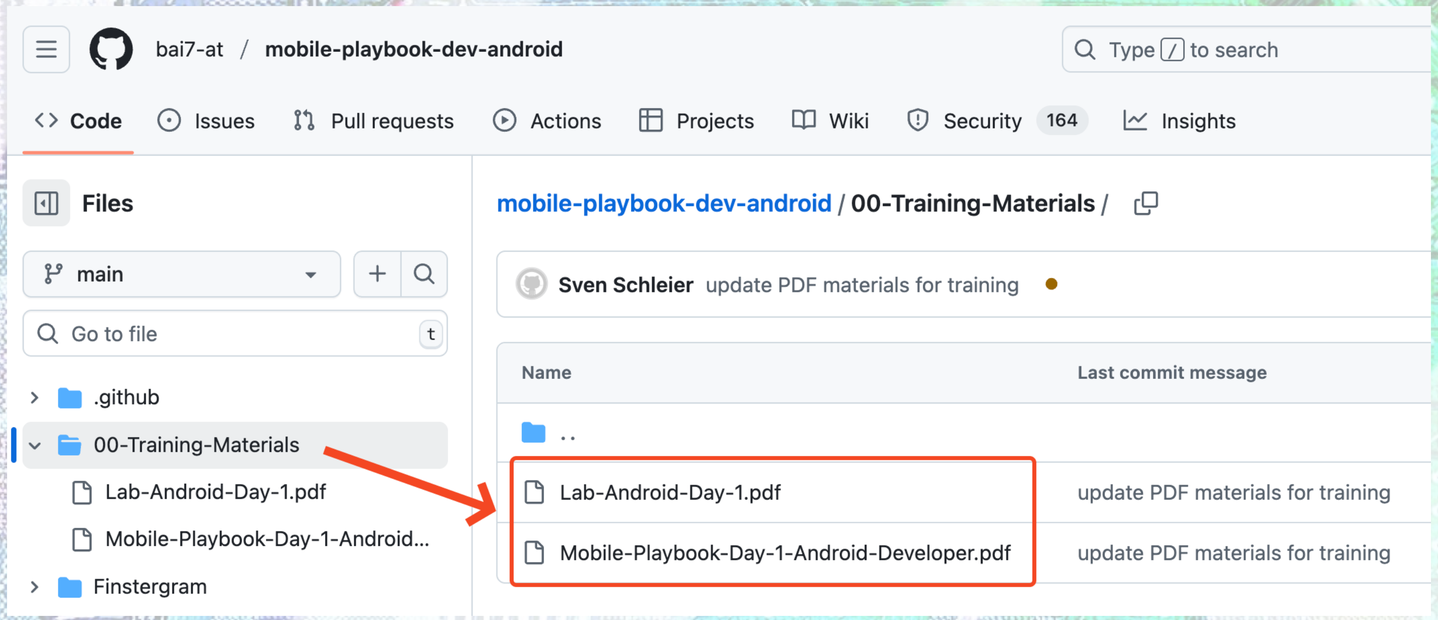
Github

Clone the following repo to your machine:

https://github.com/bai7-at/mobile-playbook-dev-ios

In the folder 00-Training-Materials there are 2 PDFs:

“Lab-iOS-Day-2.pdf” => These are the lab instructions you will go through today.
“Mobile-Playbook-Day-2-iOS-Developer.pdf” => This is the slide deck for today.



© 2025 by Bai7 GmbH | All rights reserved

Preparation

Please download both PDFs, which is the material we use for today.
The link can also be found in Slack.

https://bit.ly/4jgonol

We DONT use the chat in ZOOM!

© 2024 by Bai7 GmbH | All rights reserved

iOS Training

Introduction

/usr/bin/whoami

My name is Sven and I will be your instructor!

- Doing consulting for 15+ years in various roles in Cyber Security (focus on AppSec, Pentesting and Cloud Security)
- Worked and lived in Germany, China, Singapore and Austria.
- Principal Security Consultant at Crayon, leading the cyber security consulting team
- Since 2016 one of the project leaders for:
  - OWASP Mobile AppSec Testing Guide (MASTG) and
  - OWASP Mobile AppSec Verification Standard (MASVS)

Instructor

Sven Schleier

Contact:
- Linkedin
- sven@bai7.at

© 2024 by Bai7 GmbH | All rights reserved

Agenda - iOS Modules Day 2

9:00 – 10:30

- OWASP MAS and the new MASWE
- Mobile Threat Modeling
- Software Composition Analysis (SCA) for iOS
- Testing for sensitive data in an iOS app using Xcode, iOS Simulator and Finder.

10:30 – 11:00 break

11:00 – 12:30

- Manual review of an iOS app
- Static scanning of an iOS app (developed in Swift)
- Using Corellium to set up a jailbroken iOS device
- Jailbreaks and testing without a jailbroken device

12:30 – 13:30 lunch

13:30 – 15:00

- Intercepting network traffic from iOS apps (machine-in-the-middle attacks) and App Transport Security (ATS)
- Dynamic Instrumentation 101 - Exploring Frida, a tool used by attackers to dynamically reverse engineer iOS apps.

15:00 – 15:30 break

15:30 - 17:00

- Reverse engineering controls and their effectiveness explained in detail for iOS, such as Jailbreak or Frida detection.
- CTF and win a prize!

© 2024 by Bai7 GmbH | All rights reserved

Rules / Disclaimer

There are no stupid questions!

You are highly encouraged:

- To ask during the training
- Reach out to me during the breaks
- Or contact me via Linkedin/Slack/Mail/$foo

The intention of this course is to be interactive, and this depends on your participation!

The content for today is based on the OWASP Mobile App Sec Testing Guide and my experience during mobile pentests.

Feel free to use it after the training, but please don't share the training content publicly!

All views and opinions in this training slides and that are mentioned during the training are my own and not the view of our employer.

© 2024 by Bai7 GmbH | All rights reserved

Rules / Disclaimer

- This training is for informational and educational purpose only.
- The aim of this course is to give you an understanding of what a Penetration Tester or Reverse Engineer is doing when assessing a mobile app.
- Do not attempt to violate the law with anything contained here.
- Without the permission of the owner of the software / API / Mobile App / $foo you are committing a crime as these are considered illegal activities.
- There are bug bounty platforms if you are interested to “hack” mobile apps, where you can assess real mobile apps, like:
  - HackerOne
  - Bugcrowd
  - Intigriti
- Intentionally vulnerable apps for your personal training can be found here.

© 2024 by Bai7 GmbH | All rights reserved

iOS Training

Setup

Github

For the training today you need to use a free Github account to follow some of the labs. If you don't have one, please create one now:

https://github.com/

Sven Sch

© 2024 by Bai7 GmbH | All rights reserved

Android Training

OWASP MAS (MASVS | MASTG | MASWE)

Mobile Application Security Verification Standard (MASVS)

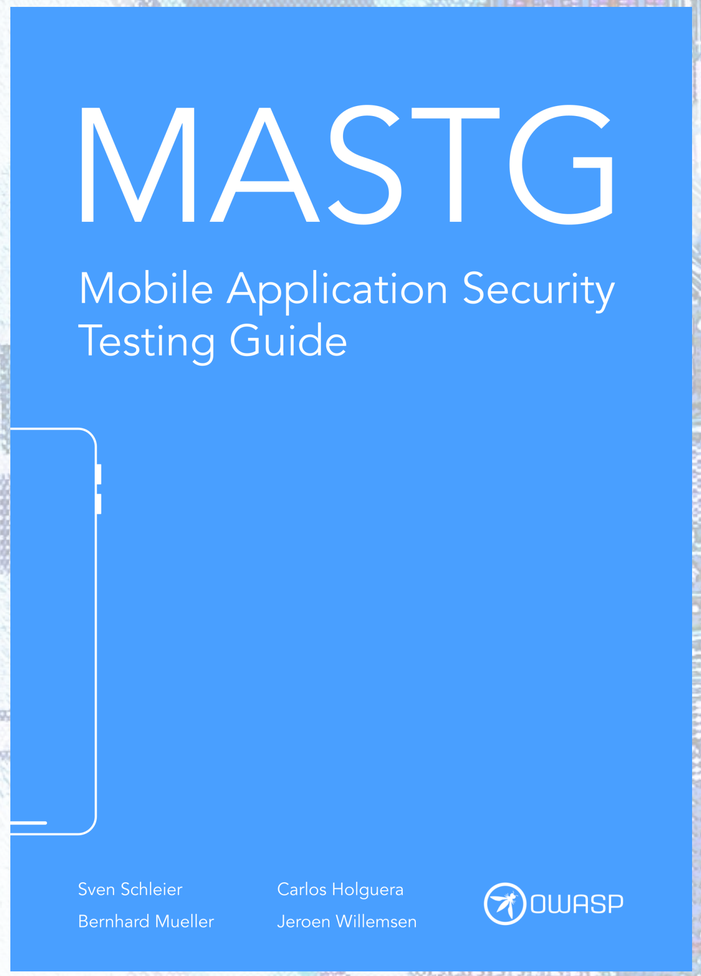
https://mas.owasp.org/MASVS/

The standard is divided into various groups of controls, labeled MASVS-XXXXX, that represent the most critical areas of the mobile attack surface:

- MASVS-STORAGE: Storage
- MASVS-CRYPTO: Cryptography
- MASVS-AUTH: Authentication and Authorisation
- MASVS-NETWORK: Network Communication
- MASVS-PLATFORM: Platform Interaction
- MASVS-CODE: Code Quality
- MASVS-RESILIENCE: Resilience against Reverse Engineering and Tampering
- MASVS-PRIVAY: Privacy

© 2024 by Bai7 GmbH | All rights reserved

Mobile Application Security Testing Guide (MASTG)



The OWASP Mobile Application Security Testing Guide (MASTG) is a comprehensive manual for mobile app security testing and reverse engineering. It describes technical processes for verifying the controls listed in the OWASP MASVS.

Structure of MASTG:

- Tests
- Techniques
- Tools
- Apps

https://mas.owasp.org/MASTG

© 2024 by Bai7 GmbH | All rights reserved

Mobile Application Security Weakness Enumeration (MASWE)

The Mobile Application Security Weakness Enumeration (MASWE) is a list of common security and privacy weaknesses in mobile applications. It is intended to be used as a reference for developers, security researchers, and security professionals. It acts as the bridge between the MASVS and the MASTG.

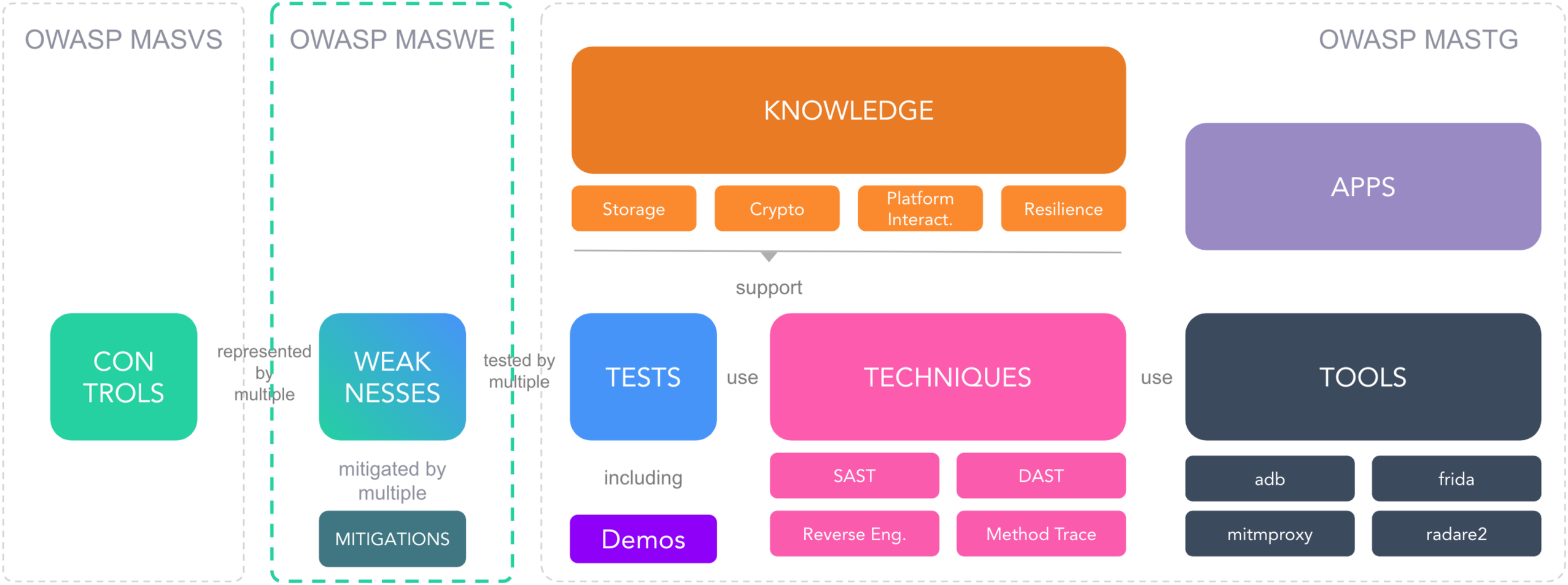
For its definition we draw inspiration from the Common Weakness Enumeration (CWE), which is a community-developed list of common software security weaknesses. The MASWE is intended to be a **complementary list to the CWE**, focusing specifically on security weaknesses in mobile applications.

A **weakness** is a security or privacy issue that can be introduced into a mobile application. Weaknesses are categorized by the MASVS categories and controls. For example, a weakness related to the use of insecure random number generators is categorized under the `MASVS-CRYPTO-1` control.

https://mas.owasp.org/MASWE

© 2024 by Bai7 GmbH | All rights reserved

OWASP MAS Project



https://mas.owasp.org/news/2024/07/30/new-maswe/

© 2024 by Bai7 GmbH | All rights reserved

Mobile Application Security Weakness Enumeration (MASWE)

MASVS-CRYPTO-1

The app employs current strong cryptography and uses it according to industry best practices.

L1 L2

Weakness: Cryptographically Weak Pseudo-Random Number Generator (PRNG)

Test 1: Insecure Random API Usage

Demo 1: Common Uses of Insecure Random APIs

- Sample Code
- Test Script
- SAST Rule *
- Output

Test 2: Non-random Sources Usage

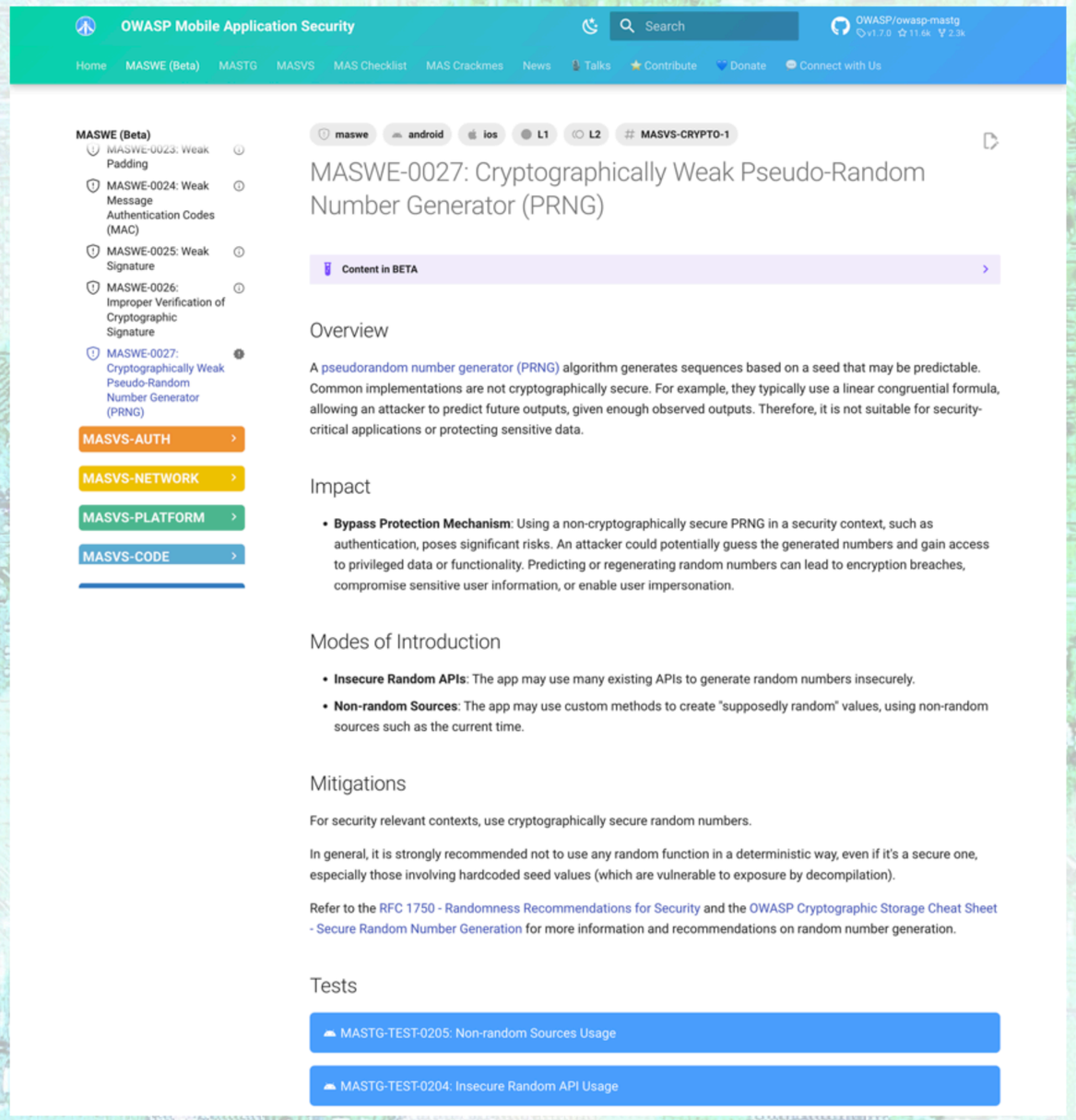
https://mas.owasp.org/MASWE/MASVS-CRYPTO/MASWE-0027/

© 2024 by Bai7 GmbH | All rights reserved

Mobile Application Security Weakness Enumeration (MASWE)

Each weakness contains the following information:

- Overview: A brief description of the weakness.
- Impact: The potential impact of the weakness on the security or privacy of the application.
- Modes of Introduction: The ways in which the weakness can be introduced into an application.
- Mitigations: Recommendations for mitigating the weakness.
- Example: MASWE-0027



OWASP Mobile Application Security

Search

OWASP/owasp-mastg

Home | MASWE (Beta) | MASTG | MASVS | MAS Checklist | MAS Crackmes | News | Talks | Contribute | Donate | Connect with Us

MASWE (Beta)

- MASWE-0023: Weak Padding
- MASWE-0024: Weak Message Authentication Codes (MAC)
- MASWE-0025: Weak Signature
- MASWE-0026: Improper Verification of Cryptographic Signature
- MASWE-0027: Cryptographically Weak Pseudo-Random Number Generator (PRNG)

MASVS-AUTH

MASVS-NETWORK

MASVS-PLATFORM

MASVS-CODE

maswe | android | ios | L1 | L2 | MASVS-CRYPTO-1

MASWE-0027: Cryptographically Weak Pseudo-Random Number Generator (PRNG)

Content in BETA

Overview

A pseudorandom number generator (PRNG) algorithm generates sequences based on a seed that may be predictable. Common implementations are not cryptographically secure. For example, they typically use a linear congruential formula, allowing an attacker to predict future outputs, given enough observed outputs. Therefore, it is not suitable for security-critical applications or protecting sensitive data.

Impact

- **Bypass Protection Mechanism**: Using a non-cryptographically secure PRNG in a security context, such as authentication, poses significant risks. An attacker could potentially guess the generated numbers and gain access to privileged data or functionality. Predicting or regenerating random numbers can lead to encryption breaches, compromise sensitive user information, or enable user impersonation.

Modes of Introduction

- **Insecure Random APIs**: The app may use many existing APIs to generate random numbers insecurely.
- **Non-random Sources**: The app may use custom methods to create "supposedly random" values, using non-random sources such as the current time.

Mitigations

For security relevant contexts, use cryptographically secure random numbers.

In general, it is strongly recommended not to use any random function in a deterministic way, even if it's a secure one, especially those involving hardcoded seed values (which are vulnerable to exposure by decompilation).

Refer to the RFC 1750 - Randomness Recommendations for Security and the OWASP Cryptographic Storage Cheat Sheet - Secure Random Number Generation for more information and recommendations on random number generation.

Tests

MASTG-TEST-0205: Non-random Sources Usage

MASTG-TEST-0204: Insecure Random API Usage

© 2024 by Bai7 GmbH | All rights reserved

Mobile Application Security Weakness Enumeration (MASWE)

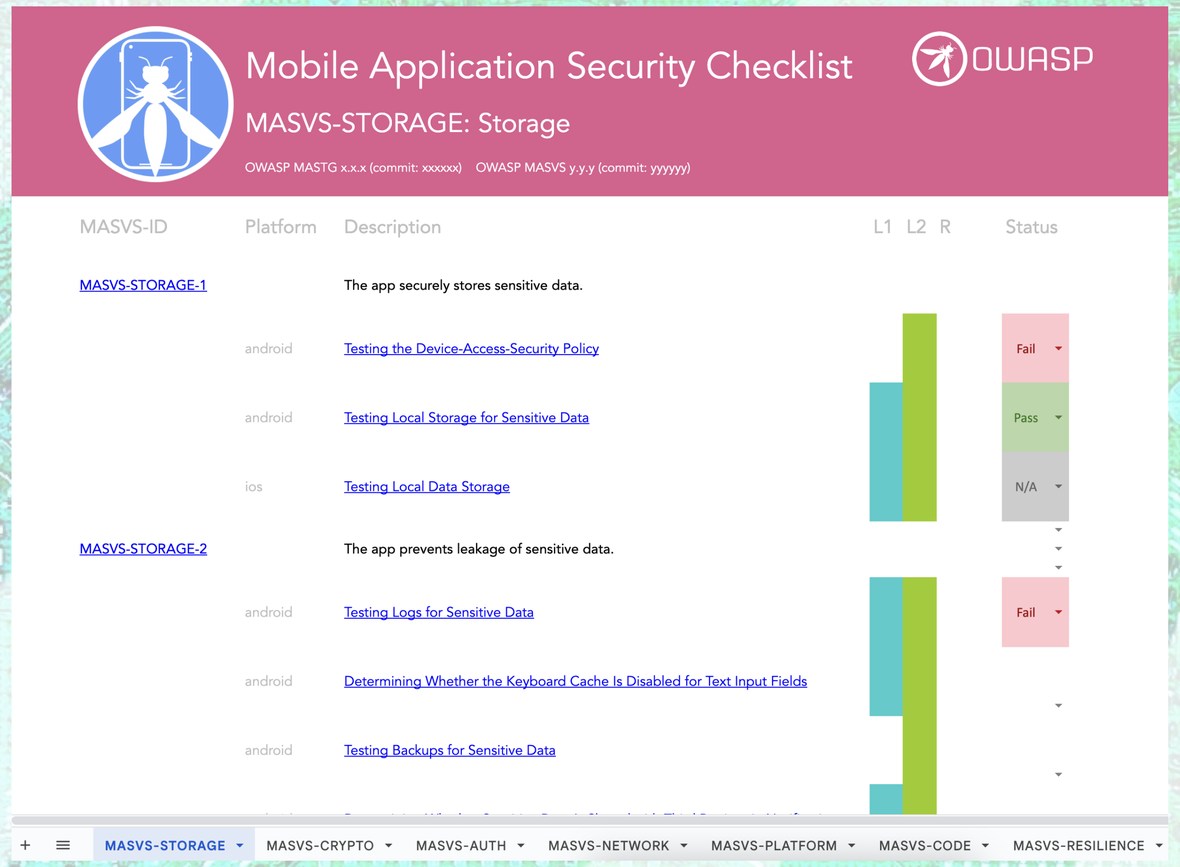
Available at mas.owasp.org/MASWE

© 2024 by Bai7 GmbH | All rights reserved

Mobile Application Security Checklist

The OWASP Mobile Application Security Checklist contains links to the MASTG test cases for each MASVS control.

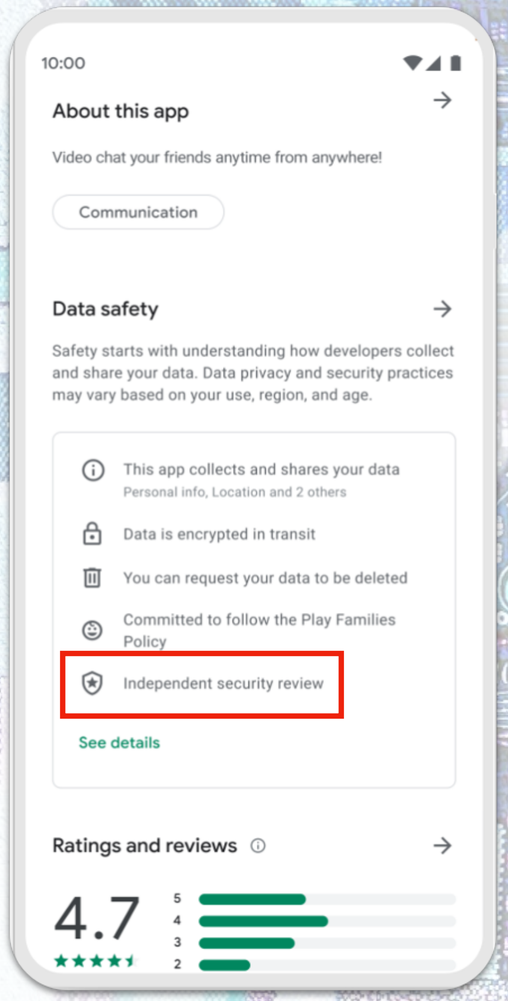
- **Security Assessments / Pentests:** ensure you're at least covering the standard attack surface and start exploring.
- **Standard Compliance:** includes MASVS and MASTG versions and commit IDs.
- **Learn & practice** your mobile security skills.
- **Bug Bounties:** go step by step covering the mobile attack surface.



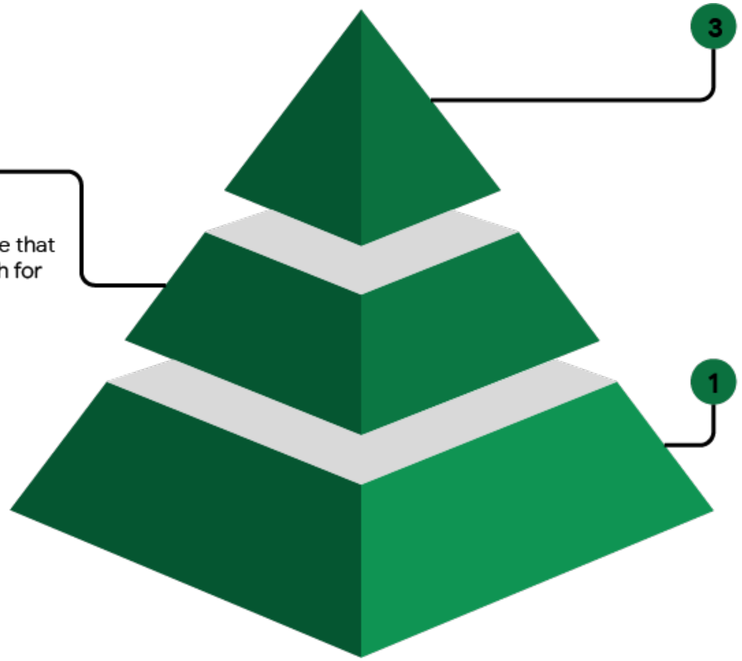
https://mas.owasp.org/checklists/

© 2024 by Bai7 GmbH | All rights reserved

Mobile Application Security (MAS) Project - Adoption in the Industry



Google Play + ADA



3 Authorized labs

Google provides a list of accredited labs for developers

OWASP MASTG 2

Corresponding testing guidance that provides a single source of truth for how to objectively test for compliance with MASVS

1 OWASP MASVS

Industry recognized set of security criteria for mobile applications

https://appdefensealliance.dev

© 2024 by Bai7 GmbH | All rights reserved

iOS Training

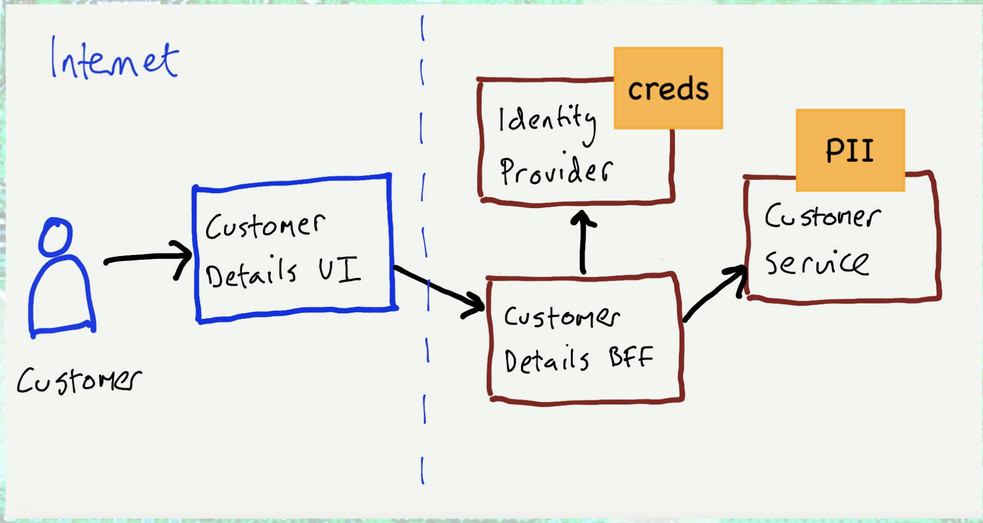
Threat Modeling

What is Threat Modeling?

"It is a structured process used to identify, assess, and mitigate potential security threats within an application. It involves analyzing the application's architecture, data flows, and potential attack vectors to proactively address vulnerabilities before they can be exploited."

Threat modeling consists of:

- Documenting and analyzing all captured information of the identified systems (or applications) and its security risks.
- Create a model (with identified threats).
- List of security recommendations to address the threats.



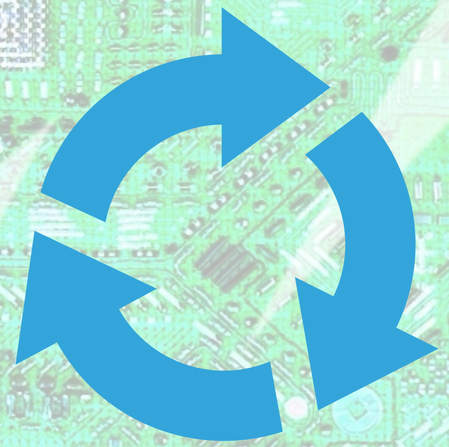
https://www.threatmodelingmanifesto.org/

© 2024 by Bai7 GmbH | All rights reserved

Threat Modeling Approach

At the highest levels, when we threat model, we ask four key questions “Shostack's Four Question Frame for Threat Modeling”:

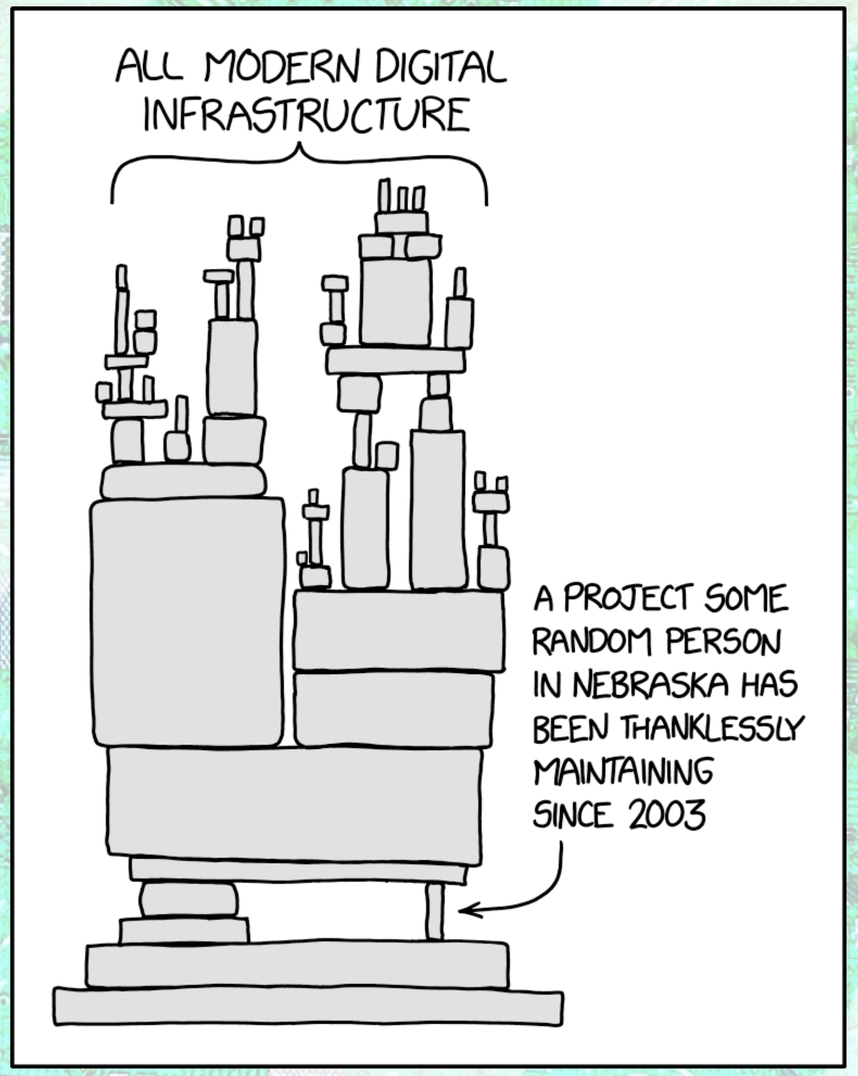
1. What are we working on?
2. What can go wrong?
3. What are we going to do about it?
4. Did we do a good job?



© 2024 by Bai7 GmbH | All rights reserved

Why should I use Threat Modeling?

- It's an essential process to secure (mobile) application development and identify design flaws (prevention) and recognize what can go wrong in a system.
- Identify risks early in the SDLC and address them.
- Justify costs! It allows organizations to determine security controls and to produce effective countermeasures within budget. For example, there is little point in spending $100,000 for fraud control for an app that has negligible fraud risk.



© 2024 by Bai7 GmbH | All rights reserved

STRIDE

- STRIDE is a Threat Classification model to identify security threats
- STRIDE was developed by Microsoft and is used to think malicious (put your hacker hat on!)
  - Spoofing, e.g. identity, is it possible to steal credentials and impersonate users?
  - Tampering with Data (malicious modification of data)
  - Repudiation associated with users who deny performing an action without other parties having any way to prove otherwise.
  - Information Disclosure threats involve the exposure of information to individuals who are not supposed to have access to it
  - Denial of Service attacks deny service to valid users
  - Elevation of Privilege, an unprivileged user gains privileged access and thereby has sufficient access to compromise or destroy the entire system.

© 2024 by Bai7 GmbH | All rights reserved

Threat Modelling - What are we working on?

Data considered sensitive in the context of the app is clearly identified:

- Ideally a classification policy or scheme is available to be able to separate between sensitive and non-sensitive data.
- If not available, it needs to be defined during the design phase of the project.
- All data in rest and in transit that is processed by the app need to be identified.
- Data need to be classified to be able to apply the correct security controls to it

© 2024 by Bai7 GmbH | All rights reserved

Threat Modelling - What are we working on?

Classification	Secret	Confidential	Public
Description	Extremely sensitive information of highest value to the company.	Sensitive information of considerable value to the company.	Non-sensitive information that is available for external release to the general public
Impact	Unauthorized disclosure could adversely impact the company and cause **severe** financial, legal or reputation damage.	Unauthorized disclosure could adversely impact the company and cause financial, legal or reputation damage.	No potential adverse impact
Examples	Usernames, passwords, certificates, keys and authentication or session tokens Fingerprint, Face scan, Credit/Debit Card/ATM Info: Number, Expiry date, CVV	First and last name Email Birthplace & Birth Date Mobile Number Driver License National identification number Health records of any kind	Marketing information

This is only an example, your classification would need to be aligned to the applicable laws and industry regulations.

© 2024 by Bai7 GmbH | All rights reserved

Threat Modelling - What are we working on?

Define the scope and understand the app architecture:

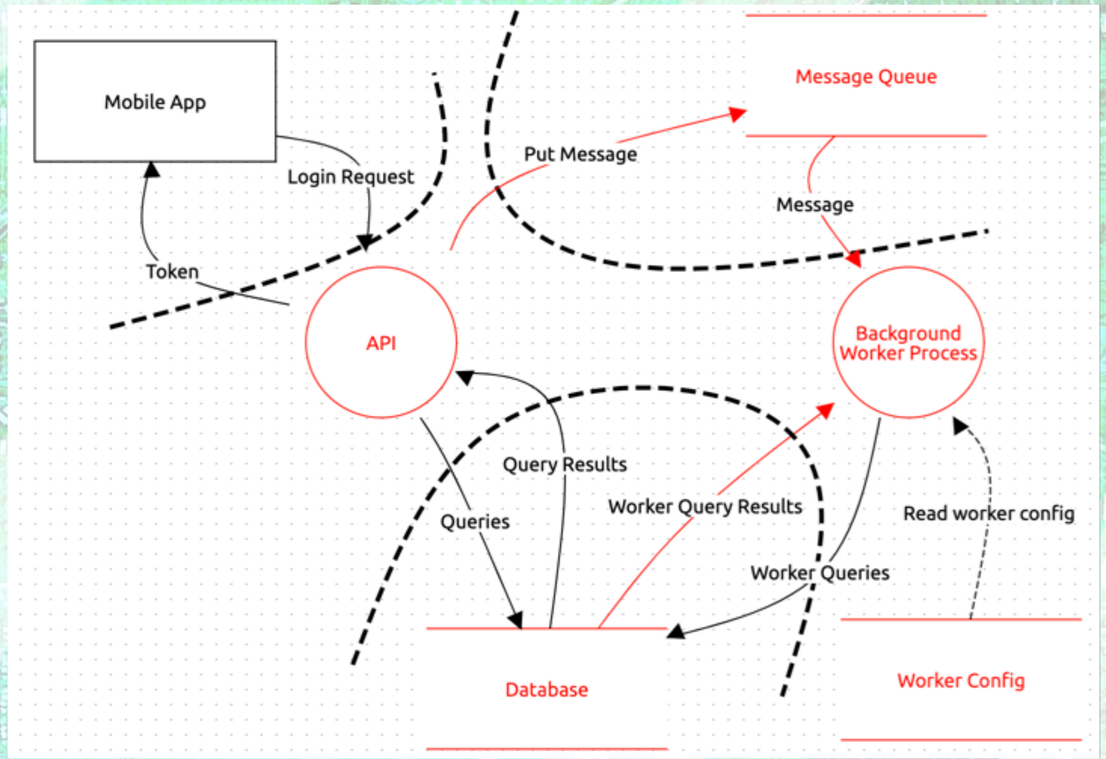
Analyze the application design on a high level including all connections made

Identify Data Flows, Components and Trust Boundaries (dotted black lines)

Free tools:

- OWASP Threat Dragon
- Microsoft Threat Modeler (Windows only)

Alternative to get started: Use your user stories and UI screens!



© 2024 by Bai7 GmbH | All rights reserved

Threat Modelling - What can go wrong?

Identify Threats by following STRIDE

1. User may have a weak password for their mobile app account
2. Data validation may allow SQL injection (on the mobile device or on the server-side)
3. Communication might be unencrypted
4. Authorization may fail, allowing unauthorized access

© 2024 by Bai7 GmbH | All rights reserved

Threat Modelling - What can go wrong?

Define the Risk:

- Your Internal Risk Rating (if available)
- OWASP risk rating - https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

Simple Risk Rating we are using:

Likelihood			
High	3	6	9
Med	2	4	6
Low	1	2	3
	Low	Med	High

Severity

Likelihood x Severity = Risk

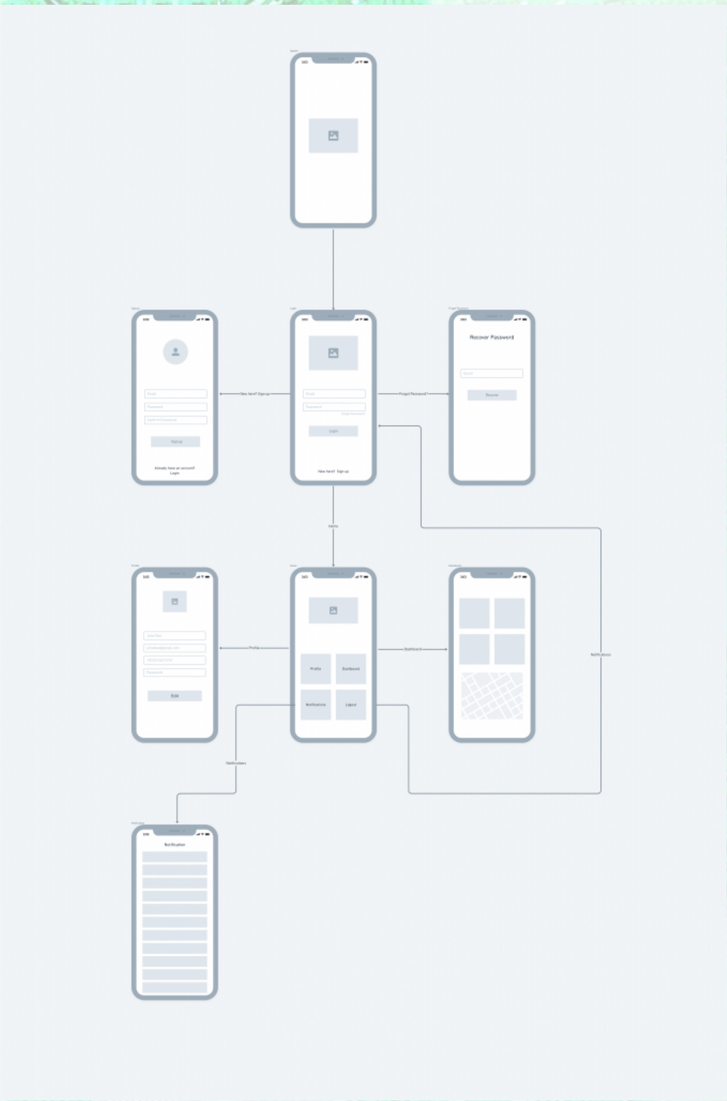
2 x 2 = 4 -> Medium risk

© 2024 by Bai7 GmbH | All rights reserved

User Flow Threat Modeling

User flow threat modeling is a security assessment method focused on identifying potential vulnerabilities within a system's user interactions. It involves analyzing how users navigate and interact with a system, mapping out their paths, and then evaluating potential security risks at each step.

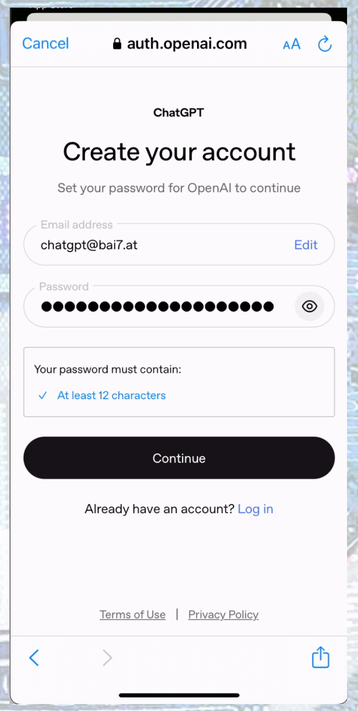
- Understanding User Journeys (“What are we working on?”)
- Identifying Potential Threats and Analyzing Risks (“What can go wrong?”)
- Developing Mitigation Strategies (“What are we going to do about it?”)



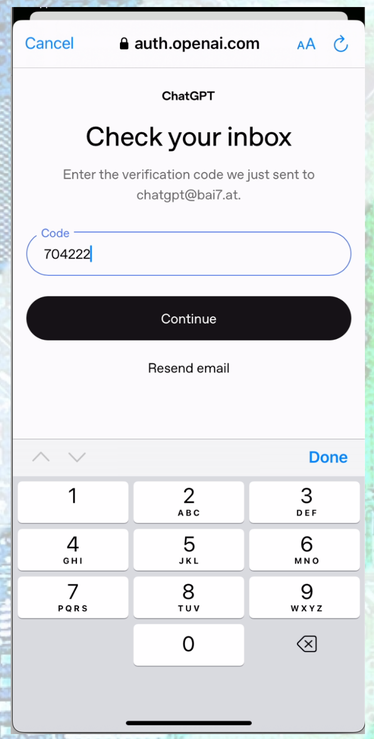
© 2024 by Bai7 GmbH | All rights reserved

User Flow Threat Modeling - Example - Registering an Account

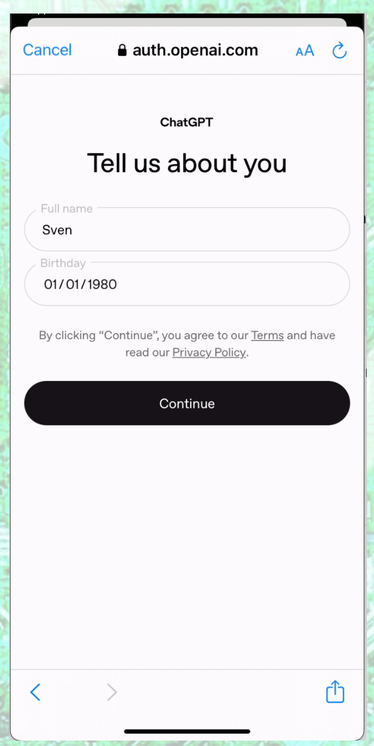
Start ChatGPT Mobile App



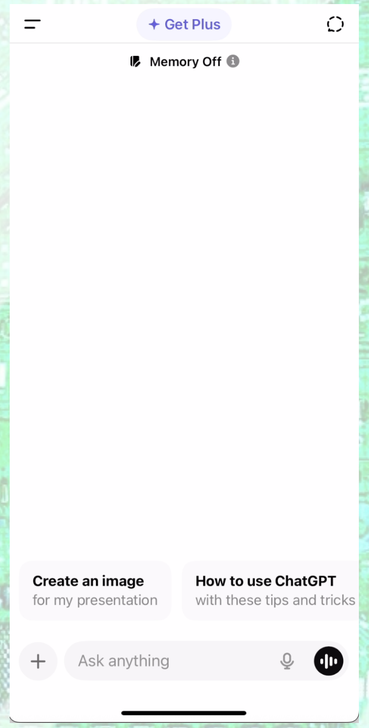
Create Account



Verify E-Mail Address



Provide further PII



Start using the service

What can go wrong and what is the attack surface that can be exploited by a malicious user or app?

-> Define threats, like "As a malicious user, I can try to inject data to verify a SQL Injection vulnerability."

© 2024 by Bai7 GmbH | All rights reserved

What can go wrong in the code verification flow and what is the attack surface that can be exploited by a malicious user?

Nobody has responded yet.

Hang tight! Responses are coming in.

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

EXAMPLE – CREATING ACCOUNT - THREATS

Threat / Abuse Case	Likelihood	Severity	Risk
As a malicious user, I can probe which mail addresses are registered	3	1	Medium
As a malicious user, I can try to onboard another mail address	1	2	Low
As a malicious user, I can try to guess (brute-force) the verification code	2	3	High
As a malicious user, I can try to bypass the verification code	1	3	Medium
As a malicious user, I can try to spam addresses with verification code mails in the name of ChatGPT	2	2	Medium
The library used for the verification code mechanism is outdated and contains a known vulnerability.	1	3	Medium

THESE ARE JUST EXAMPLES, YOU NEED TO RE-EVALUATE EVERY RISK SPECIFIC TO YOUR APP!

What can we do to mitigate guessing (Brute-forcing) of the verification code?

Nobody has responded yet.

Hang tight! Responses are coming in.

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Example – Onboarding ChatGPT App - What are we going to do about it?

Address the identified threats of the abuse case “As a malicious user, I can try to guess the verification code”:

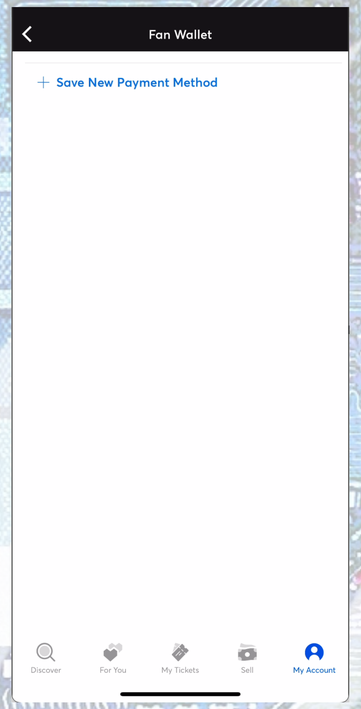
- Change/redesign the function or architecture, e.g.
  - Verification code is only valid for 2 minutes
  - Verification code can only be used once
  - Verification code is only valid for the mail address it was requested from
  - Verification code is invalided after 3 failed attempts
- Usage of a standard library or product (don’t reinvent the wheel!)
- Accept the risk of a vulnerability (might be a viable option it there is low business impact or investment to fix would exceed the costs of exploitation of the vulnerability)

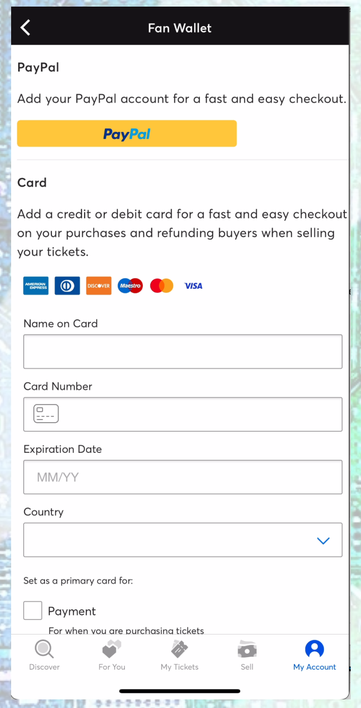
Check the Authentication Cheat Sheet from OWASP for best practices:

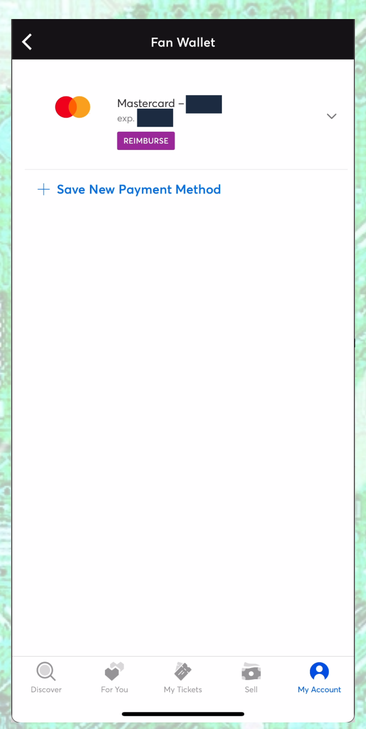
https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html

© 2024 by Bai7 GmbH | All rights reserved

User Flow Threat Modeling - Example - Storing Credit Card in Ticketmaster App







Save new payment method → Add credit card details → Stored as primary payment form

What can go wrong and what is the attack surface that can be exploited by a malicious user or app?

© 2024 by Bai7 GmbH | All rights reserved

What can go wrong when adding payment information and what is the attack surface that can be exploited by a malicious user or app?

Nobody has responded yet.

Hang tight! Responses are coming in.

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

EXAMPLE – ADDING CREDIT CARD - THREATS

Threat / Abuse Case	Likelihood	Severity	Risk
	3	1	Medium
	1	2	Low
	2	3	High
	1	3	Medium
	2	2	Medium
	1	3	Medium

THESE ARE JUST EXAMPLES, YOU NEED TO RE-EVALUATE EVERY RISK SPECIFIC TO YOUR APP!

What can we do to mitigate threats through insecure adding payment information?

Nobody has responded yet.

Hang tight! Responses are coming in.

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Example – Onboarding ChatGPT App - What are we going to do about it?

Address the identified threats of the abuse case “The app is storing the credit card data locally unencrypted”:

- Change/redesign the function or architecture, e.g.
  - Credit Card data should never be stored locally
  - Credit Card data should only be the server side with the payment provider
- Usage of a standard library or product (don’t reinvent the wheel!)
- Follow the best practices to implement the payment flow by calling APIs or a through a SDK

Check the Authentication Cheat Sheet from OWASP for best practices:

https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html

© 2024 by Bai7 GmbH | All rights reserved

Threat Modelling - What can go wrong?

The following sources can be used as "mobile attack libraries” to be aware of mobile app threats:

- OWASP MASWE - https://mas.owasp.org/MASWE/
- MITRE ATT&CK - https://attack.mitre.org/matrices/mobile

The following sources can be used as “server side attack libraries” to be aware of backend threats:

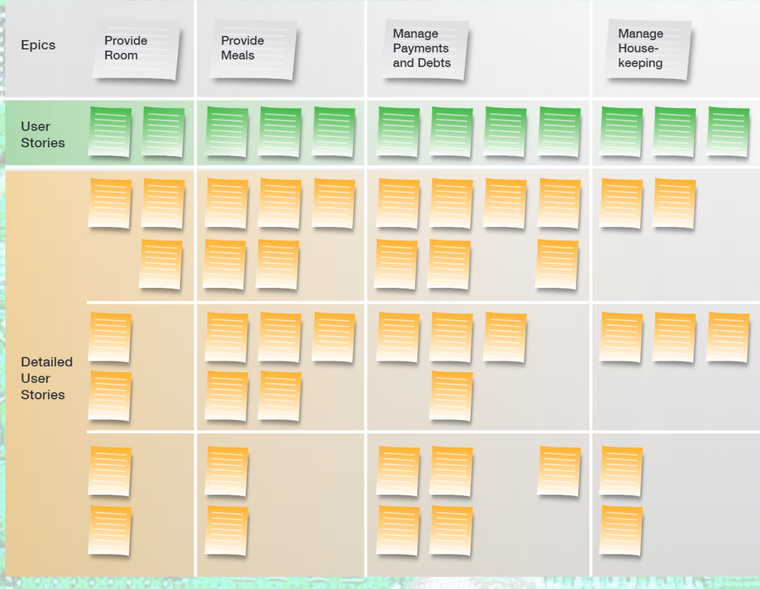
- OWASP Web Testing Guide - https://owasp.org/www-project-web-security-testing-guide/
- OWASP ASVS - https://owasp.org/www-project-application-security-verification-standard/
- CAPEC - https://capec.mitre.org

© 2024 by Bai7 GmbH | All rights reserved

Threat Modeling Approach

Outcome of the Threat Model can be part of your user stories. Examples:

- As a user, I want that the verification code is implemented in a secure way.
- As an authenticated user, I want to access my profile.
- As a user, I want to receive an email when someone logs in from a new device so that I can detect unauthorized access.
- As an admin, I want to enforce two-factor authentication so that user accounts are more secure.
- As a developer, I want to validate input fields so that the app is protected from injection attacks.



© 2024 by Bai7 GmbH | All rights reserved

Threat Modelling - How to get started?

- Use the models and input that you already have (diagrams, UI Screens, User Journeys) and work out a flow that works for your development process to identify and document threats
- A Threat Model tool would only make sense if the whole company is using the same approach for defining models, otherwise you would need to redo the work again in the tool
- Start small, make it part of your Definition of Done (DoD) and “Threat Model Every Story”
- Focus on specific areas first that usually lead to critical issues and that are hard to detect during SAST, like Authentication and Authorization.
- Example for continuous threat modeling for developers - https://github.com/Autodesk/continuous-threat-modeling
- Different approaches for execution of Threat Modeling - https://shostack.org/files/papers/Fast-Cheap-and-Good.pdf

© 2024 by Bai7 GmbH | All rights reserved

iOS Training

Software Composition Analysis (SCA)

iOS - Software Composition Analysis (SCA)

OWASP Dependency Check is an open source tool to scan libraries (dependencies) in software projects to detect publicly disclosed vulnerabilities in them.

OWASP dependency-check-cli is a command line tool and can be installed on all main operating systems.

The tool will generate a report listing the dependency and the associated Common Vulnerability and Exposure (CVE) and CVSS entries.

© 2024 by Bai7 GmbH | All rights reserved

iOS - Software Composition Analysis (SCA)

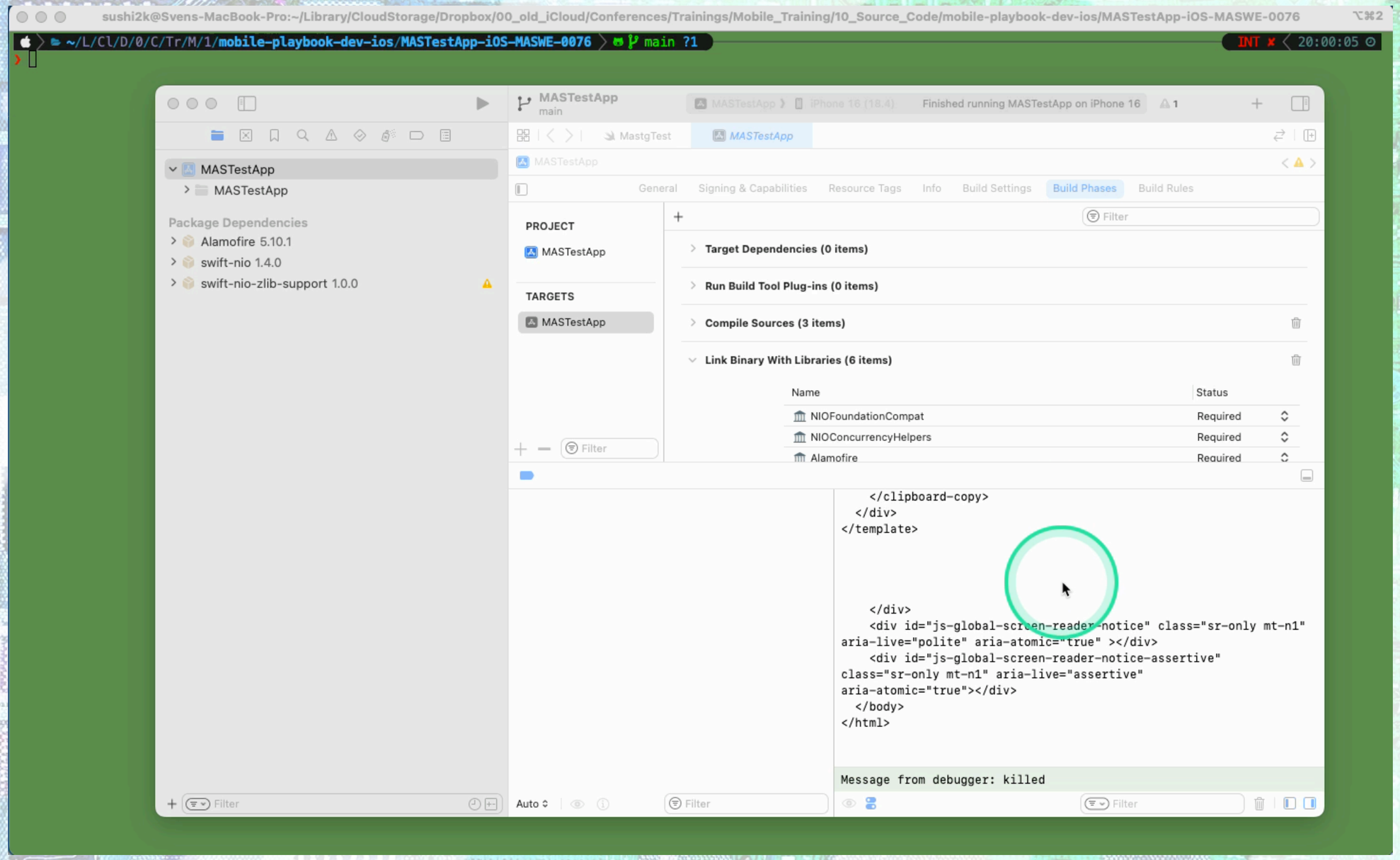
3 widely used package managers for iOS:

- Swift Package Manager (experimental support by dependency check)
- Carthage (experimental support by dependency check)
- CocoaPods (experimental support by dependency check)

© 2024 by Bai7 GmbH | All rights reserved

iOS - Software Composition Analysis (SCA)

Executing Dependencyp-Track for SwiftPM dependencies.



© 2024 by Bai7 GmbH | All rights reserved

iOS - Software Composition Analysis (SCA)

Dependency-check report and results:

DEPENDENCY-CHECK

Dependency-Check is an open source tool performing a best effort analysis of 3rd party dependencies; false positives and false negatives may exist in the analysis performed by the tool. Use of the tool and the reporting provided constitutes acceptance for use in an AS IS con its use. Any use of the tool and the reporting provided is at the user's risk. In no event shall the copyright holder or OWASP be held liable for any damages whatsoever arising out of or in connection with the use of this tool, the analysis performed, or the resulting report.

How to read the report | **Suppressing false positives** | **Getting Help: github issues**

Sponsor

Project:

Scan Information (show all):

- *dependency-check version*: 12.1.3
- *Report Generated On*: Fri, 15 Aug 2025 20:01:03 +0200
- *Dependencies Scanned*: 3 (3 unique)
- *Vulnerable Dependencies*: 2
- *Vulnerabilities Found*: 9
- *Vulnerabilities Suppressed*: 0
- ...

Summary

Summary of All Dependencies(click to show less)

Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count
alamofire:5.10.1		pkg:swift/alamofire@5.10.1		0		5
swift-nio-zlib-support:1.0.0	cpe:2.3:a:zlib:zlib:1.0.0:*:*:*:*:*:*	pkg:swift/swift-nio-zlib-support@1.0.0	CRITICAL	4	Highest	5
swift-nio:1.4.0	cpe:2.3:a:apple:swift:1.4.0:*:*:*:*:*:* cpe:2.3:a:apple:swift_foundation:1.4.0:*:*:*:*:*:*	pkg:swift/swift-nio@1.4.0	HIGH	5	High	5

© 2024 by Bai7 GmbH | All rights reserved

iOS - Software Composition Analysis (SCA)

Local Scan:

- The first scan will take a while as the tool is downloading all public known vulnerabilities (CVE) from NVD and is building up its local database.
- Subsequent scans will only download the diff of new CVEs and will be faster.

There is no build scan available for Xcode, only local scan via CLI.

© 2024 by Bai7 GmbH | All rights reserved

iOS - Software Composition Analysis (SCA)

How to fix identified vulnerabilities?

Several options:

- Upgrade the library to the “nearest” version that is free of vulnerabilities, e.g. vulnerable version is 2.4.5 and fixed versions are 2.4.6 and 3.0.1. Major version version upgrades might have many changes in the usage of the API and might need refactoring of the code, so upgrading to 2.4.6 might be the easiest way to upgrade.
- Prioritize fixing of vulnerabilities with the highest impact to your apps, as not all vulnerabilities can be fixed on the same time, by
  - Checking if there are public exploits; can be done through EPSS https://www.first.org/epss/
  - Is the vulnerable function in the library called in your code (some enterprise tools support this)

© 2024 by Bai7 GmbH | All rights reserved

iOS - Software Composition Analysis (SCA)

How to fix identified vulnerabilities?

How do you know which library is free of vulnerabilities? => Manual research or enterprise tools.

Commercial products (sorted alphabetically) can detect outdated libraries and offer an overview of library versions that are free of known vulnerabilities:

- Blackduck | Github Dependabot | Snyk | Sonatype Nexus IQ | $vendor

In case there is no fix available for the library:

- Remove the library (if not needed)
- Change to another library
- Accept the risk and security team will track it in their risk register

© 2024 by Bai7 GmbH | All rights reserved

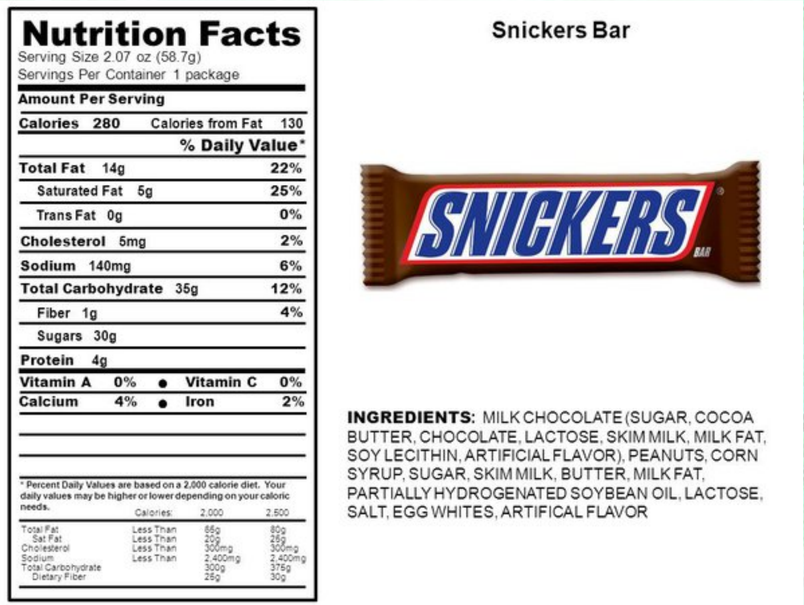
What is a Software Bill of Material (SBOM)?

© 2024 by Bai7 GmbH | All rights reserved

iOS - Software Composition Analysis (SCA) by using SBOM

Scanning for known vulnerabilities - Software Bill of Material (SBOM)

- List of components used to build software
- SBOM describes the components in detail
- Single Source of Truth and your inventory for libraries (when used with tools like https://dependencytrack.org)
- Identify affected software that uses vulnerable libraries



Nutrition Facts
Serving Size 2.07 oz (58.7g)
Servings Per Container 1 package

Amount Per Serving			
Calories 280		Calories from Fat	130
			% Daily Value*
Total Fat 14g			22%
Saturated Fat 5g			25%
Trans Fat 0g			0%
Cholesterol 5mg			2%
Sodium 140mg			6%
Total Carbohydrate 35g			12%
Fiber 1g			4%
Sugars 30g			
Protein 4g			
Vitamin A 0%	•	Vitamin C	0%
Calcium 4%	•	Iron	2%

* Percent Daily Values are based on a 2,000 calorie diet. Your daily values may be higher or lower depending on your caloric needs.

Calories:		2,000	2,500
Total Fat	Less Than	65g	80g
Sat Fat	Less Than	20g	25g
Cholesterol	Less Than	300mg	300mg
Sodium	Less Than	2,400mg	2,400mg
Total Carbohydrate		300g	375g
Dietary Fiber		25g	30g

Snickers Bar

INGREDIENTS: MILK CHOCOLATE (SUGAR, COCOA BUTTER, CHOCOLATE, LACTOSE, SKIM MILK, MILK FAT, SOY LECITHIN, ARTIFICIAL FLAVOR), PEANUTS, CORN SYRUP, SUGAR, SKIM MILK, BUTTER, MILK FAT, PARTIALLY HYDROGENATED SOYBEAN OIL, LACTOSE, SALT, EGG WHITES, ARTIFICAL FLAVOR

© 2024 by Bai7 GmbH | All rights reserved

iOS - Software Composition Analysis (SCA) by using SBOM

Software Bill of Material (SBOM) - Formats

- There are different open-source formats, where CycloneDX and SPDX are popular ones.
- Comparing CycloneDX and SPDX, see also here.

CycloneDX	SPDX
Example of CycloneDX	Example of SPDX
Focus on vulnerability analysis and specifically built for SBOMs. "What is it, and is it safe to use right now?"	Focus more on licensing (e.g., which open-source license is used), not vulnerability analysis. “What is it, and under what legal terms can I use it?”
Backed by OWASP	Backed by Linux Foundation project

© 2024 by Bai7 GmbH | All rights reserved

iOS - Software Composition Analysis (SCA) by using SBOM

CycloneDX and cdxgen

- cdxgen - Creates CycloneDX Bill of Materials (BOM) for your projects from source and container images.
- Supports iOS package managers:
  - SwiftPM
  - Cocoapods

cyclonedx.github.io/cdxgen/#/PROJECT_TYPES

CDXGEN

Ruby (Specific version)	`ruby*` . Example: ruby2.5.4, ruby3.4.0	`Gemfile.lock` , `gemspec`	Only for
cocoapods	`cocoa` , `ios` , `objective-c`	Podfile, Podfile.lock	Yes
PHP	`php` , `composer`	Composer.lock	Yes

cyclonedx.github.io/cdxgen/#/PROJECT_TYPES

Supported languages and package formats

CDXGEN

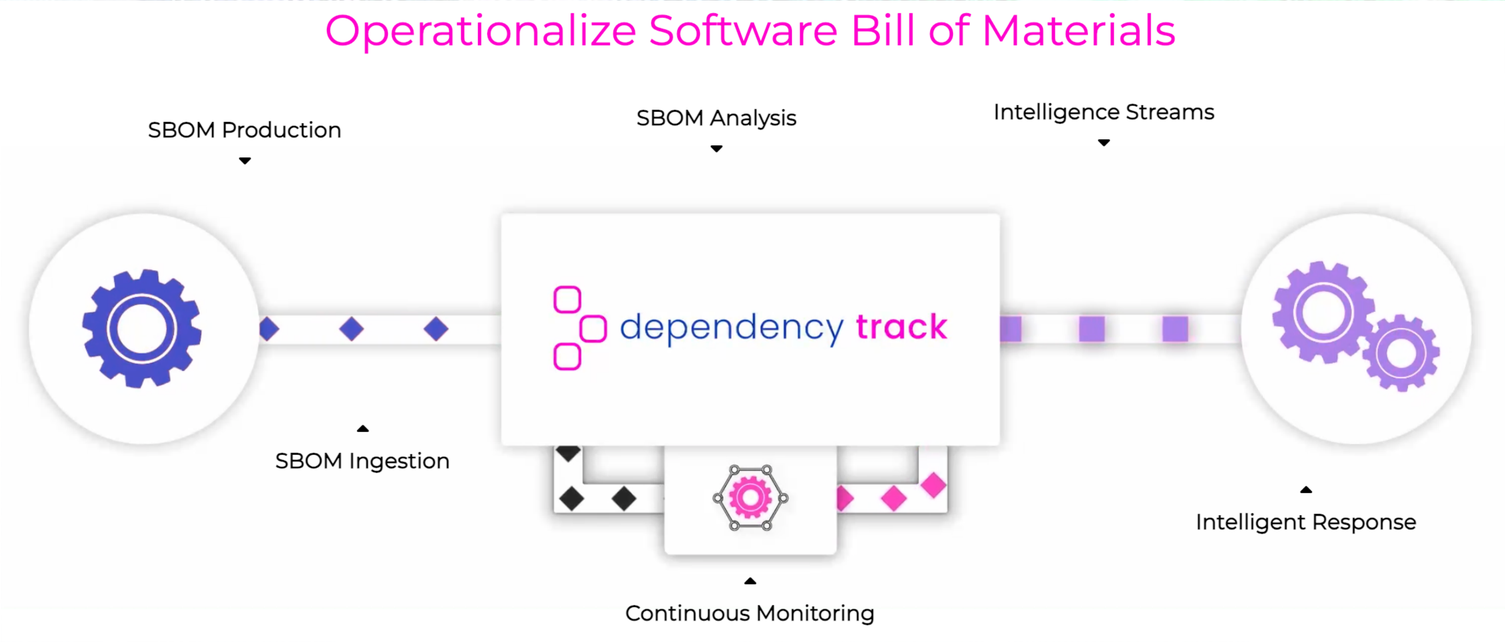
	`yaml-manifest`		
Google Cloud Build	`cloudbuild`	`cloudbuild.yaml`	n/a
Swift (iOS)	`swift`	`Package.resolved` , `Package.swift` (swiftpm)	Yes
Binary	`binary` , `blint`		

https://cyclonedx.github.io/cdxgen/#/PROJECT_TYPES

© 2024 by Bai7 GmbH | All rights reserved

iOS - Software Composition Analysis (SCA) by using SBOM

- Dependency Track is a tool from OWASP and is a “Component Analysis platform”
- It can ingest the SBOM directly for continuous monitoring via API in your CI/CD pipeline
- Monitors component usage across all versions of every application in its portfolio in order to proactively identify risk across an organization



© 2024 by Bai7 GmbH | All rights reserved

iOS - Software Composition Analysis (SCA) by using SBOM

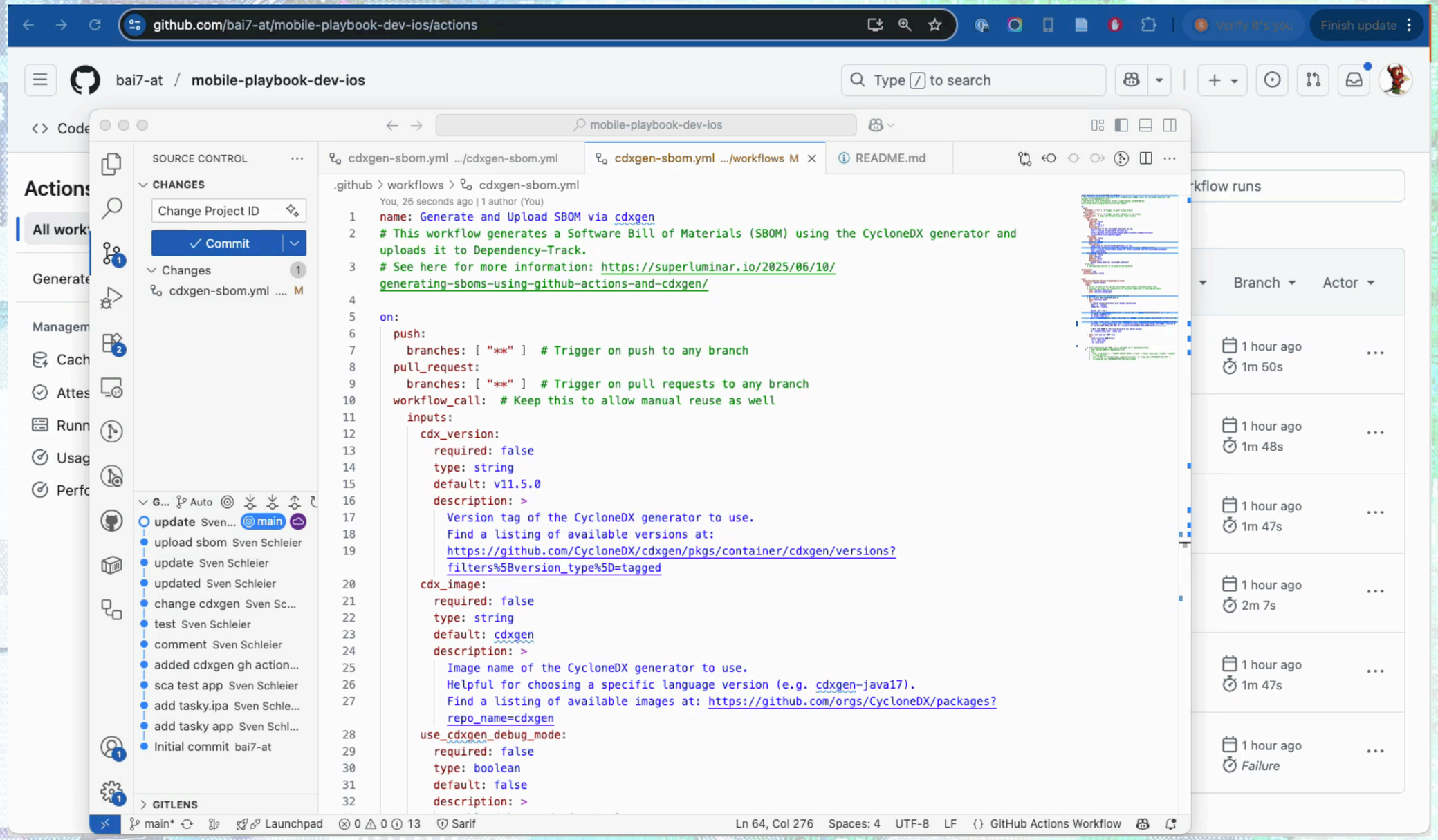
```
~/Doc/mastg-a/MASTestApp-iOS-MASWE-0076  main !2 ?4  INT ✘  venv-frida-ios-dump  20:02:02
❯
```

Generating SBOM via cdxgen locally and uploading to the dependency-track server via API.

© 2024 by Bai7 GmbH | All rights reserved

iOS - Software Composition Analysis (SCA) by using SBOM

Generating SBOM via cdxgen via Github Action and uploading to the dependency-track server via API.



© 2024 by Bai7 GmbH | All rights reserved

iOS - Software Composition Analysis (SCA) by using SBOM

cdxgen creates a SBOM and uploads it to the depdency-track server in one go.

mobile-playbook-dev-ios / .github / workflows / cdxgen-sbom.yml

Code | Blame | 83 lines (73 loc) · 3.29 KB | Raw

```
51          # Retrive secret for dependency-Track API key
52          # ${{ secrets.DEP_TRACK_API_KEY }}
53          - name: Generate SBOM
54            run: |
55              # Create output directory with proper permissions
56              mkdir -p ./output
57              chmod 777 ./output
58
59              docker run --rm \
60              ${{ (inputs.use_cdxgen_debug_mode || false) && '-e CDXGEN_DEBUG_MODE=debug' || '' }} \
61              -e FETCH_LICENSE=true \
62              -v $(pwd):/app:rw \
63              ghcr.io/cyclonedx/${{ inputs.cdx_image || 'cdxgen' }}:${{ inputs.cdx_version || 'v11.5.0' }} \
64              -r /app --type swift ./MASTestApp.xcodeproj/project.xcworkspace/xcshareddata/swiftpm/Package.resolved \
65              -o /app/output/sbom.json --server-url http://dep-track.s7ven.info:8081 \
66              --api-key ${{ secrets.DEP_TRACK_API_KEY }} --project-id 782364f2-fdec-430f-a2c5-112118ba5d12
67
```

https://cyclonedx.github.io/cdxgen/#/
https://github.com/bai7-at/mobile-playbook-dev-ios/blob/main/.github/workflows/cdxgen-sbom.yml#L63

© 2024 by Bai7 GmbH | All rights reserved

iOS - Software Composition Analysis (SCA) by using SBOM

Explore Dependency-Track:

URL: http://dep-track.s7ven.info:8080/

User: bloomberg

Password: murdered-fax-meaning

© 2024 by Bai7 GmbH | All rights reserved

iOS - Software Composition Analysis (SCA) by using SBOM

MASWE-0076: Dependencies with Known Vulnerabilities

MASWE that has several test cases for iOS and Android. These test cases are more developer focused or when you have access to the source code.

In case you don't have access to the source code, ask the developer to share a SBOM in CycloneDX format (if the package manager is supported by cdxgen). This still allows you to test for known vulnerabilities in the packages with dependency-track.

Tests ¶

- MASTG-TEST-0275: Dependencies with Known Vulnerabilities in the App's SBOM
- MASTG-TEST-0273: Identify Dependencies with Known Vulnerabilities by Scanning Dependency Managers Artifacts
- MASTG-TEST-0272: Identify Dependencies with Known Vulnerabilities in the Android Project
- MASTG-TEST-0274: Dependencies with Known Vulnerabilities in the App's SBOM

© 2024 by Bai7 GmbH | All rights reserved

iOS - Software Composition Analysis (SCA) - Summary

- SCA (Software Composition Analysis) => security scanning of 3rd party libraries
  - On OS level (Debian or Red Hat packages)
  - On application level (package managers such as Maven, Node.JS or PyPI).
- Very effective way to identify vulnerabilities
- Vulnerable libraries need to be fixed or removed
- SBOM is an essential part Application Security and need to be continuously created and monitored
- For Pentesters: Ask the development team which tool they are using and if you can have access to the report

© 2024 by Bai7 GmbH | All rights reserved

iOS - Software Composition Analysis (SCA) - Summary

Evaluate tool support for SCA:

- Identify which package managers you are using (for server-side and app-side)
- All SAST tools also cover SCA, find a tool that covers both and that can give you full coverage of your tech stack
- Find a tool that can show you if the vulnerable library is called and therefore exploitable to prioritise the right libraries that should be fixed.
- Integrate it into your build pipeline and scan frequently (e.g., daily)

© 2024 by Bai7 GmbH | All rights reserved

iOS Training

Analyzing data in iOS Apps

iOS - Analyzing data in iOS Apps - Analyse App Sandbox

Conventional wisdom suggests that as little sensitive data as possible should be stored on permanent local storage. In most practical scenarios, however, some type of sensitive user data must be stored, e.g.:

- Session Information (Cookies)
- Access / Refresh Tokens (JWT)
- Encryption keys
- Sensitive data for offline usage

Disclosing sensitive information has several consequences, like:

- Social engineering (if PII has been disclosed) or
- Account hijacking (if session information or an authentication token has been disclosed).

© 2024 by Bai7 GmbH | All rights reserved

iOS - Analyzing data in iOS Apps - Analyse App Sandbox

Storing Data is essential for mobile apps.

The following list of storage techniques are widely used on the iOS platform:

- Plist
- File System
- CoreData
- Databases

© 2024 by Bai7 GmbH | All rights reserved

iOS - Analyzing data in iOS Apps - Analyse App Sandbox

On iOS, system applications can be found in the /Applications directory while user-installed apps are available under /var/mobile/Containers/Data/Application/.

Every app on every device gets a unique and random 128-bit UUID (Universal Unique Identifier) assigned during installation for its directory names.

The Bundle Directory (where the app is stored) is in another directory.

```
info.s7ven.ios.shack on (iPhone: 12.4) [usb] # env

Name               Path
-----------------  ------------------------------------------------------------------------------------------
BundlePath         /var/containers/Bundle/Application/4BB068B7-7CAE-4C10-83C1-AB00CA08A61D/iOS-Shack.app
CachesDirectory    /var/mobile/Containers/Data/Application/5590D458-8F73-4224-BD0D-DADCCD1D0678/Library/Caches
DocumentDirectory  /var/mobile/Containers/Data/Application/5590D458-8F73-4224-BD0D-DADCCD1D0678/Documents
LibraryDirectory   /var/mobile/Containers/Data/Application/5590D458-8F73-4224-BD0D-DADCCD1D0678/Library
```

© 2024 by Bai7 GmbH | All rights reserved

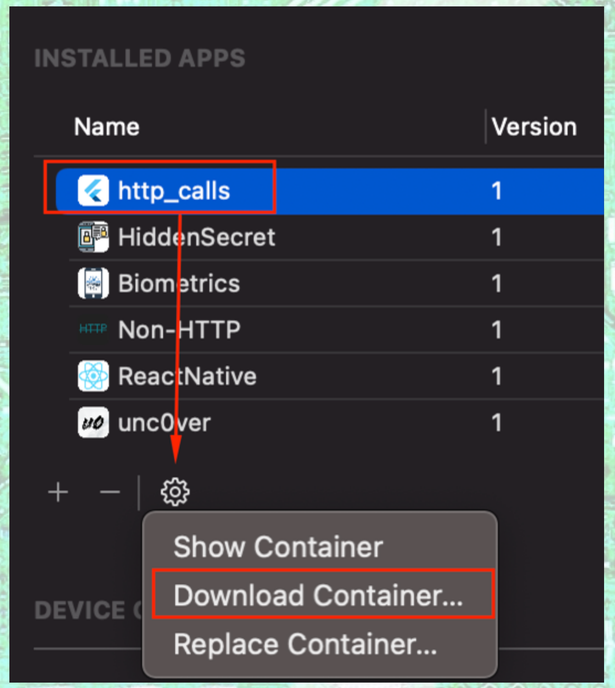
iOS - Analyzing data in iOS Apps - Analyse App Sandbox

- Xcode
- Analysing sandbox of iOS Simulator in the local file system
- Tools powered by Frida (jailbroken and non-jailbroken)

© 2024 by Bai7 GmbH | All rights reserved

iOS - Analyzing data in iOS Apps - Download App Container

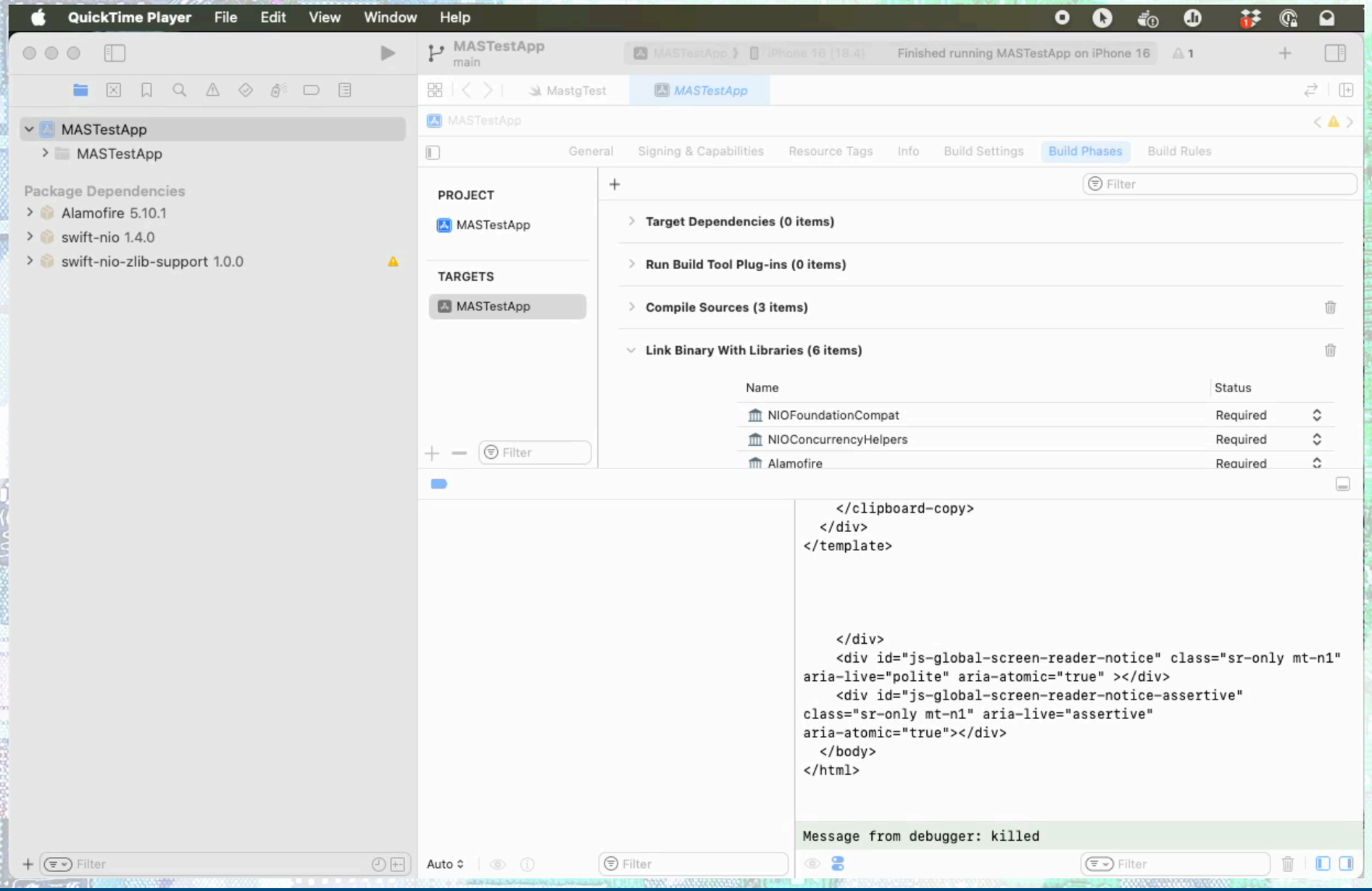
- Open Xcode and go to “Window / Devices and Simulator”
- Select the app you want to analyse and “Download Container”
- The file stored has the extension “xcappdata”. You can either:
- Change the file extension to “.zip” and unzip it, or
- In Finder make a right click and select “Show Package Contents”.
- The content of the archive will contain the data directory of the app.
- This only works with apps that were signed by you!



© 2024 by Bai7 GmbH | All rights reserved

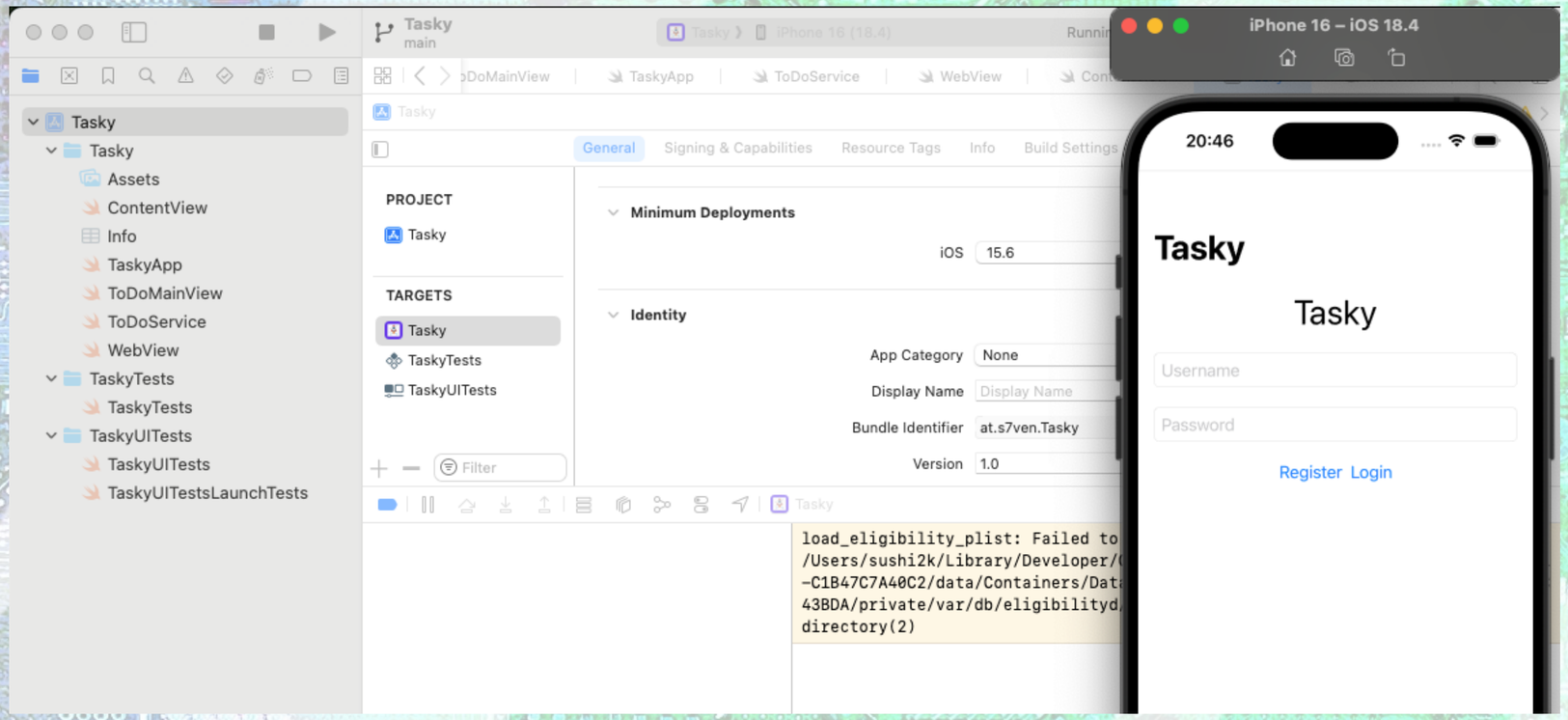
iOS - Analyzing data in iOS Apps - Download App Container

Demo - Xcode
Download App Container



© 2024 by Bai7 GmbH | All rights reserved

iOS - Analyzing data in iOS Apps - Analyse data sandbox of iOS Simulator



As developer use the tools you already have:

- Xcode and iOS Simulator
- You already have full access to the file system of the iOS Simulator
- You can query the path with “xcrun”

```
❯ xcrun simctl get_app_container booted at.s7ven.Tasky data
/Users/sushi2k/Library/Developer/CoreSimulator/Devices/CBA05B0C-7C59-4F3F-B5E1-C1B47C7A40C2
/data/Containers/Data/Application/BE139DD7-588B-4E3D-8E14-7867F0C43BDA
```

© 2024 by Bai7 GmbH | All rights reserved

iOS - Analyzing data in iOS Apps - Analyse data sandbox of iOS Simulator

Every app and simulator gets a random 128-bit UUID (Universal Unique Identifier) assigned during installation for its directory names. When using the iOS Simulator the path is:

~/Library/Developer/CoreSimulator/Devices/<Device-UUID>/data/Containers/Data/Application/<App-UUID>

A very handy way to open the data directory of our app running in the current simulator in Finder is the following:

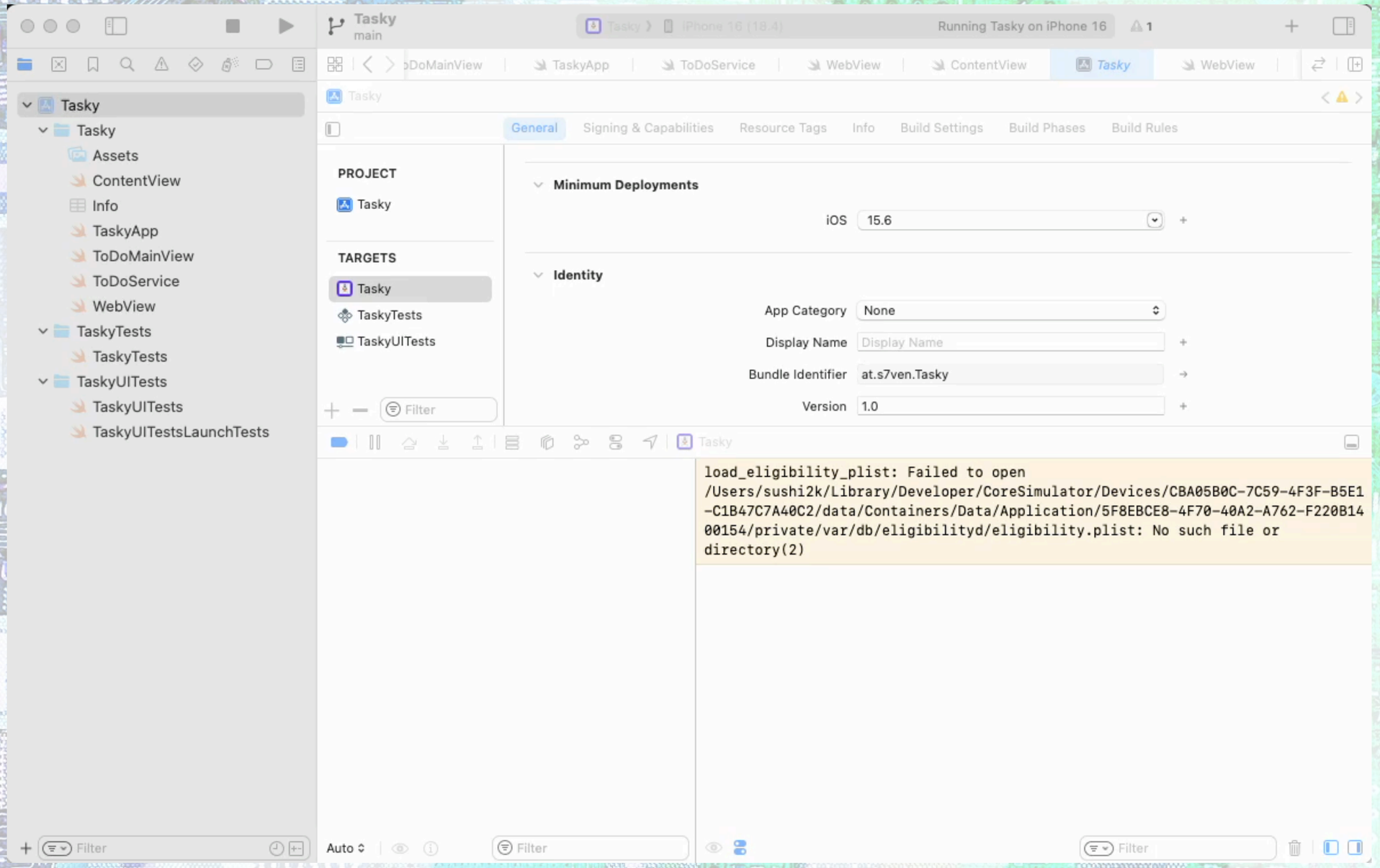
```
$ open "$(xcrun simctl get_app_container booted at.s7ven.Tasky data)”
```

The bundle name would need to be explicitly specified, which is at.s7ven.Tasky in this case.

© 2024 by Bai7 GmbH | All rights reserved

iOS - Analyzing data in iOS Apps - Analyse data sandbox of iOS Simulator

Demo - Xcode
Download App Container



© 2024 by Bai7 GmbH | All rights reserved

iOS - Analyzing data in iOS Apps - Attacker View

https://github.com/ChiChou/grapefruit

Node.js based

Can be installed and upgraded by using npm

Setup

Grapefruit is now on npm as igf! Interprets as iGrapefruit or maybe iOS Graphical Frida

`npm install -g igf`

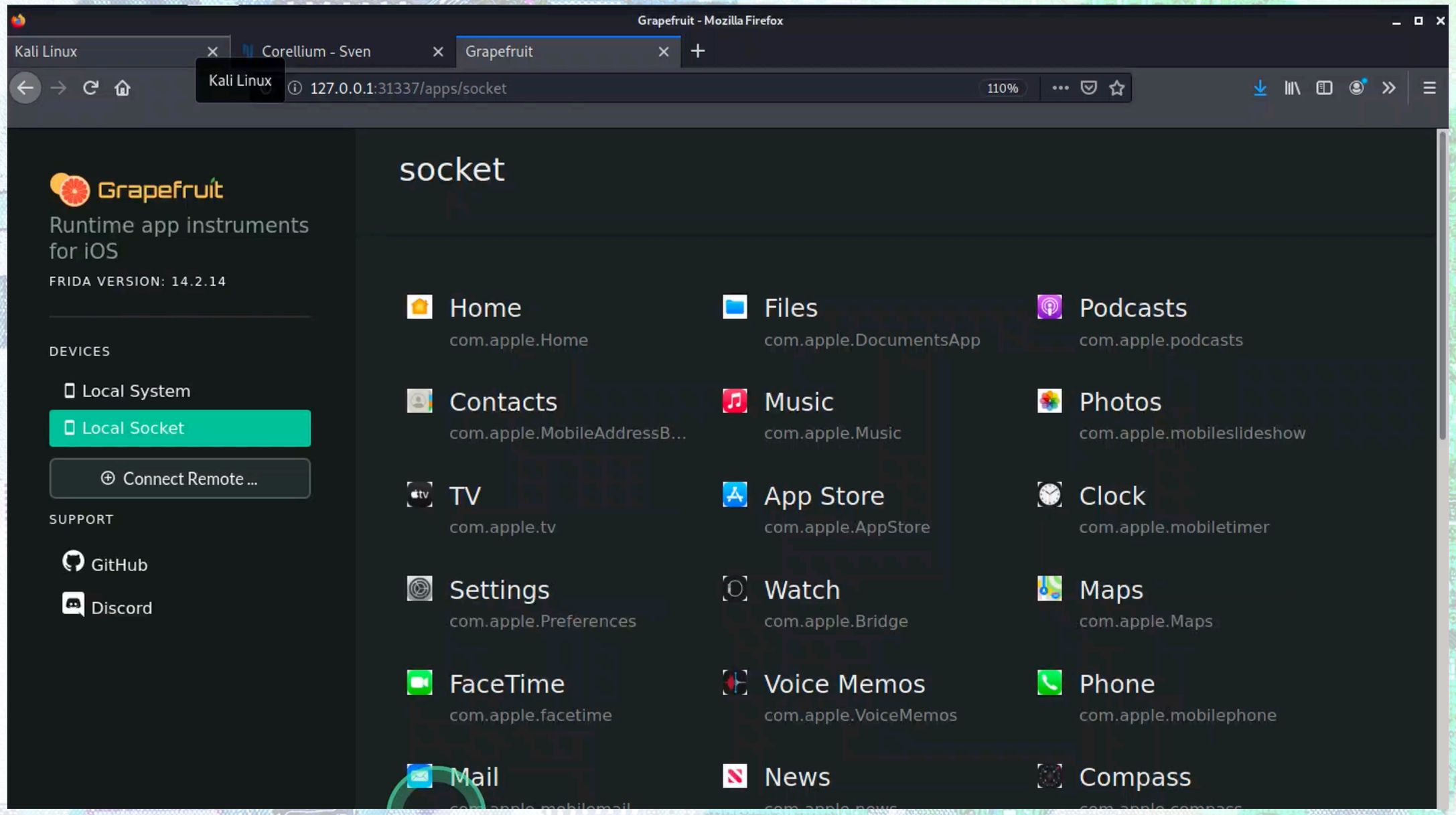
Now you have the grapefruit cli as `igf`:

```
~ igf --help

Usage: igf [options]

Options:
  -h, --host <string>  hostname (default: "127.0.0.1")
  -p, --port <number>  port of the server side (default: 31337)
  --help               display help for command
```

Default URL for the web UI is `http://localhost:31337`

© 2024 by Bai7 GmbH | All rights reserved

iOS - Analyzing data in iOS Apps - Attacker View



© 2024 by Bai7 GmbH | All rights reserved

iOS - Analyzing data in iOS Apps - Analysis Techniques Summary

ID	Technique	Jailbroken device needed?	Are there build-in modules for data analysis?	Can data be analysed in real-time?
1	Xcode	No	No	No, it's a Snapshot
2	Local File System (iOS Simulator)	No	Depending on installed software on macOS	Yes
3	Based on Frida (e.g. Grapefruit)	supports jailbroken and non-jailbroken	Yes	Yes

ID 2 is best for developers, ID 3 is best for penetration testers and security researchers.

© 2024 by Bai7 GmbH | All rights reserved

iOS - Analyzing data in iOS Apps

Should sensitive data always be encrypted? It depends….

The same that we discussed for Android also depends for iOS. Only encrypt the data in the app sandbox, if you identify a specific threat you want to protect against.

Tests

Android

MASVS-STORAGE

MASTG-TEST-0001: Testing Local Storage for Sensitive Data

MASTG-TEST-0003: Testing Logs for Sensitive Data

- This includes `SharedPreferences`, databases, Internal Storage, External Storage, etc.

NOTE: For MASVS L1 compliance, it is sufficient to store data unencrypted in the application's internal storage directory (sandbox). For L2 compliance, additional encryption is required using cryptographic keys securely managed in the Android KeyStore. This includes using envelope encryption (DEK+KEK) or equivalent methods, or using the Android Security Library's `EncryptedFile` ↗/ `EncryptedSharedPreferences` ↗.

© 2024 by Bai7 GmbH | All rights reserved

iOS - Analyzing data in iOS Apps

What is “L1/L2 compliance” aka Testing Profiles?

MAS-L1 - Essential Security

MAS-L1 provides a baseline for the most fundamental security requirements and best practices that every mobile app should meet to protect against common threats.

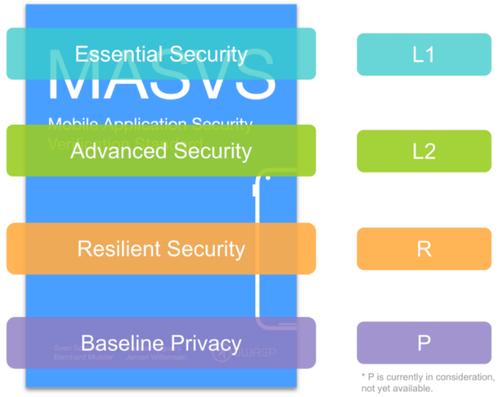
MAS-L2 - Advanced Security

MAS-L2 extends MAS-L1 introducing additional security measures and best practices for mobile apps to address advanced threats requiring more rigorous threat modeling and testing strategies.

MAS Testing Profiles

As part of the MASVS refactoring, we've replaced the three traditional verification levels (L1, L2, and R) with security testing profiles in the MASTG. These new profiles are designed to enhance our ability to capture various security nuances associated with mobile apps, allowing us to evaluate different situations for the same MASVS control. For instance, in MASVS-STORAGE-1, it's acceptable to store data unencrypted in app internal storage for MAS-L1, but MAS-L2 requires data encryption.

The new MAS Testing Profiles include revamped versions of the traditional levels and one new addition:



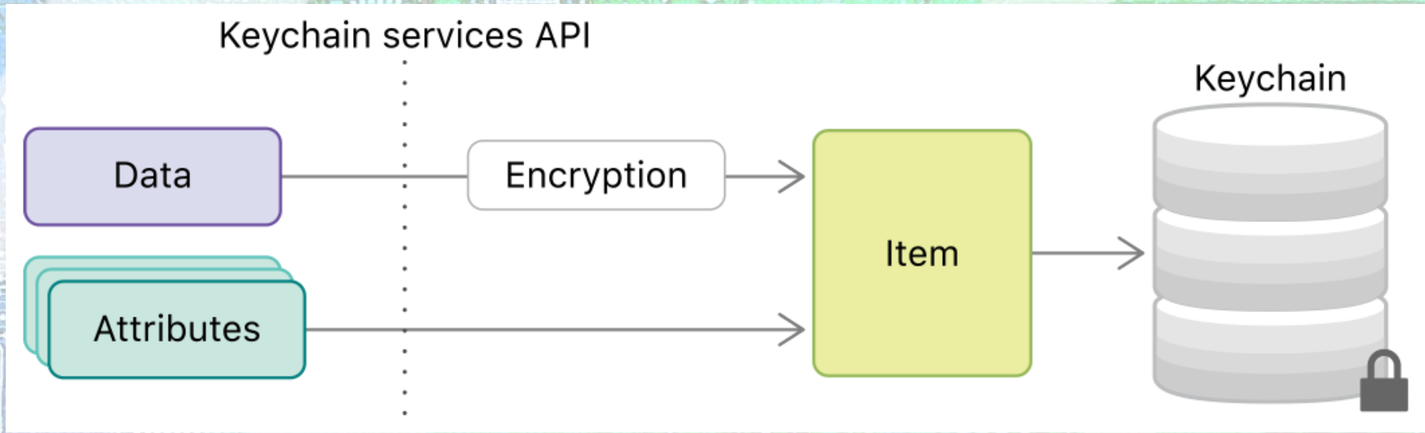
https://mas.owasp.org/news/2023/07/28/mas-testing-profiles-and-mastg-atomic-tests/

© 2024 by Bai7 GmbH | All rights reserved

iOS - Analyzing data in iOS Apps

Keychain

The iOS Keychain can be used to securely store short, sensitive bits of data, such as encryption keys and session tokens. It is implemented as an SQLite database that can be accessed through the Keychain APIs only.



https://developer.apple.com/documentation/security/keychain-items

© 2024 by Bai7 GmbH | All rights reserved

iOS - Analyzing data in iOS Apps - Summary

- Using the keychain is best practice to store sensitive data.
- As developer, use the tools available in Xcode and through the iOS Simulator to analyse the apps' sandbox.
- For penetration testers, they use tools such as grapefruit or objection (based on Frida) to analyste the local storage of an app during runtime.
- Store only what you need on permanent local storage!
- Testing of local storage should be done after triggering functions with sensitive data involved and at the very end of testing

© 2024 by Bai7 GmbH | All rights reserved

iOS Training

Manual Static Analysis of iOS Apps

iOS - Static Application Security Testing (SAST) - Tooling

IDEs (Integrated Development Environments) often provide basic code review functions and can be extended with various plugins.

Performing code review using automated SAST tools in the CI/CD pipeline.

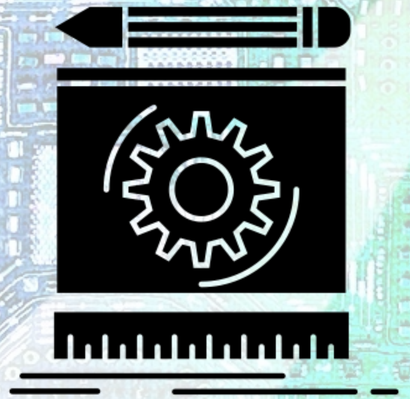
A penetration tester / security researcher can decompile the app and performs manual code review on it (”grepping through code”)

In most cases, a hybrid of automatic and manual approach is used. Automatic scans catch the low-hanging fruits, and the penetration tester can explore the code base with specific attack paths in mind.

© 2025 by Bai7 GmbH | All rights reserved

iOS - Static Application Security Testing (SAST) - Decompiling

Lets explore an insecure Deeplink Implementation via manual static analysis.



© 2025 by Bai7 GmbH | All rights reserved

iOS - Deep Link

But first: What is a deep link?

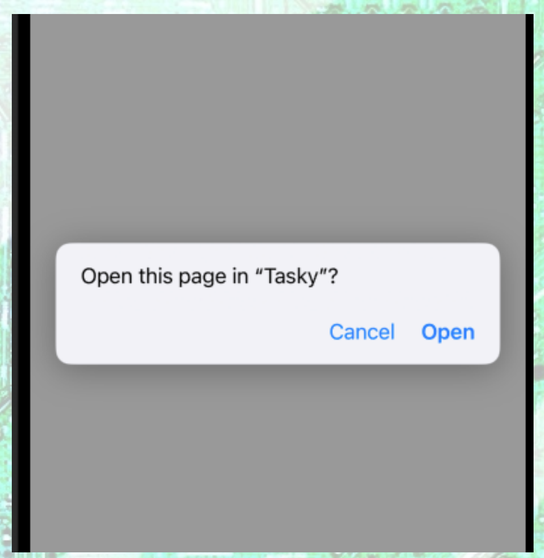
And what is a Universal Link?

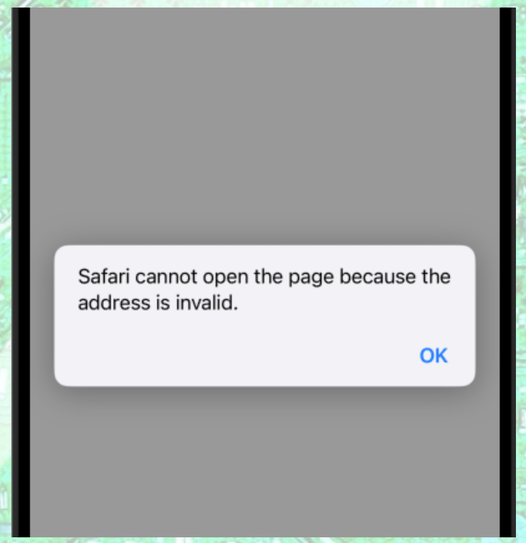
© 2025 by Bai7 GmbH | All rights reserved

iOS - Deep Link

A deep link is a link that takes you directly to a specific destination within an app.

When a clicked link or programmatic request invokes the deeplink, the iOS system tries to find the link with the prefix and opens the App associated with the prefix, for example tasky://.





Deep links are custom URI schemes that, when opened, cause iOS to launch the registered app and pass the URL to it. This can or cannot have the “://“ after the scheme, e.g spotify has a URN-Style URI like spotify:track:3n3Ppam7vgaVa1iaRUc9Lp

<scheme>://<host>[<path>|<pathPrefix>|<pathPattern>]

© 2025 by Bai7 GmbH | All rights reserved

iOS - Universal Links

Universal links are Apple's way of linking to a page within an app since iOS 9 or allow communication between apps. When a clicked link or programmatic request invokes a web URL, the iOS system tries each of the following actions, in sequential order, until the request succeeds:

- If the app is installed, it will launch and the user be sent to the relevant content.
- If the app is not installed, the page is opened within a web browser like a traditional URL.

Universal Links are standard web URLs (e.g., https://www.facebook.com/pages/wikipedia/103767749661773) that can open directly inside a specific app instead of the browser — if the app has registered and verified ownership of that domain.

https://developer.apple.com/documentation/xcode/allowing-apps-and-websites-to-link-to-your-content/

© 2025 by Bai7 GmbH | All rights reserved

iOS - Deep Links vs Universal Links

Regardless if (Deep | Universal) ***links*** all are defined in the Info.plist.

Example of a Deep Link with scheme tasky://:

Tasky 〉 Tasky 〉 Info 〉 No Selection

Key	Type	Value
Information Property List	Dictionary	(1 item)
URL types	Array	(1 item)
Item 0 (at.s7ven.Tasky)	Dictionary	(3 items)
Document Role	String	Editor
URL identifier	String	at.s7ven.Tasky
URL Schemes	Array	(1 item)

© 2025 by Bai7 GmbH | All rights reserved

iOS - Deep Link

Are there any risks to deep linking?

© 2025 by Bai7 GmbH | All rights reserved

iOS - Deep Link

Are there any risks to deep linking?

- Sensitive functionality or data could be accessible and therefore ***exposed*** to other apps.
- Any input to a deep links that is not validated could be the starting point of an attack.
- Deep linking will open up a new attack surface for the app, hence we can creatively chain deep linking with other bugs.

© 2025 by Bai7 GmbH | All rights reserved

iOS - Static Application Security Testing (SAST) - MobSF

- Allows to scan the APK or IPA
- Available as docker container or cli

Mobile Security Framework (MobSF)

Version: v4.0

Mobile Security Framework (MobSF) is a security research platform for mobile applications in Android, iOS and Windows Mobile. MobSF can be used for a variety of use cases such as mobile application security, penetration testing, malware analysis, and privacy analysis. The Static Analyzer supports popular mobile app binaries like APK, IPA, APPX and source code. Meanwhile, the Dynamic Analyzer supports both Android and iOS applications and offers a platform for interactive instrumented testing, runtime data and network traffic analysis. MobSF seamlessly integrates with your DevSecOps or CI/CD pipeline, facilitated by REST APIs and CLI tools, enhancing your security workflow with ease.

Made with ❤ in India

python 3.10+ | pypi package 3.9.7 | platform osx/linux/windows | license GPL-3.0-only | Docker pulls 3.7M

MobSF tests passing | quality gate passed | issues 1.5k closed | openssf best practices in progress 97%

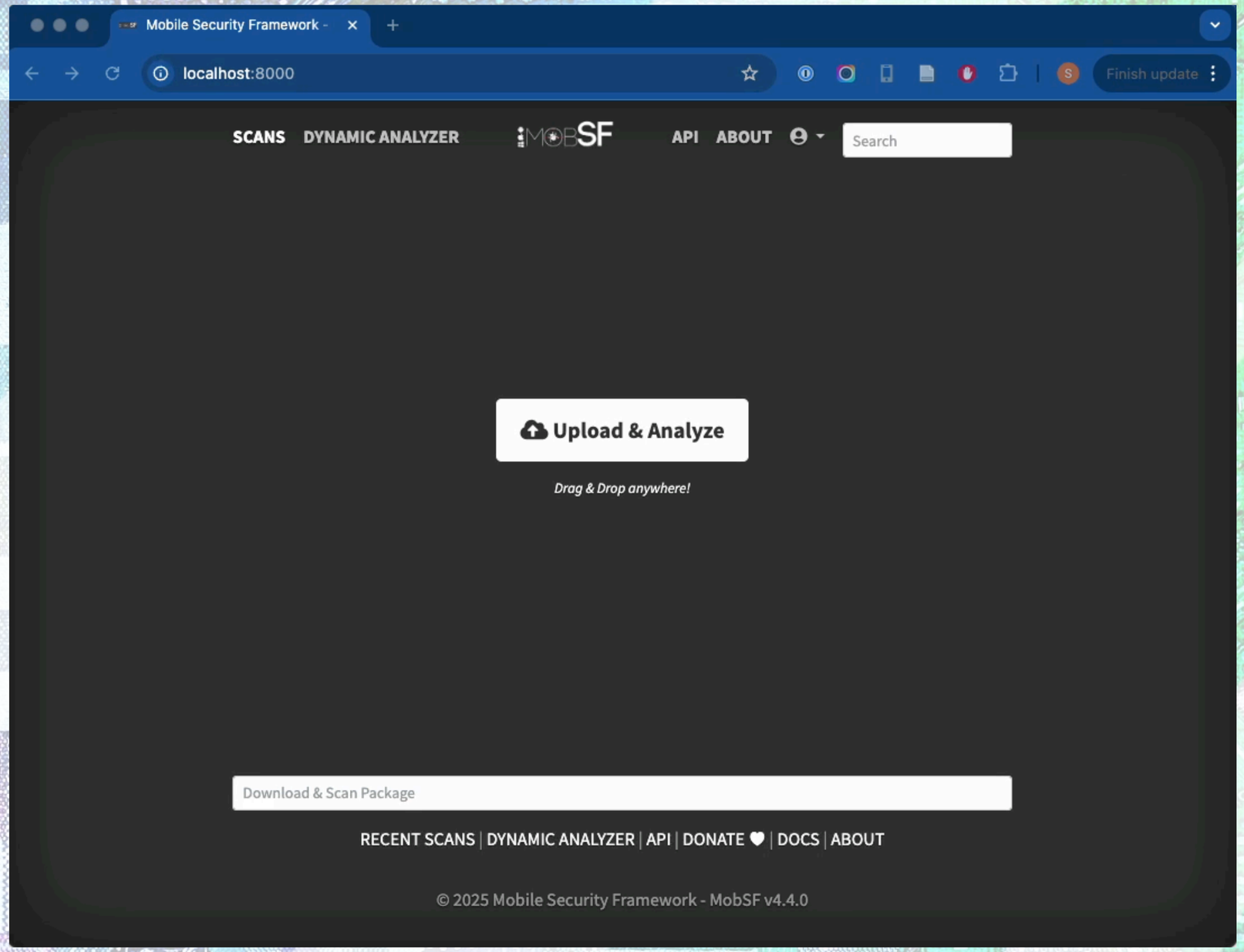
ToolsWatch Rank 5 | Year 2016 | ToolsWatch Rank 9 | Year 2017 | Black Hat Arsenal Asia 2015 | Black Hat Arsenal Asia 2018 | Black Hat Arsenal Europe 2023

MobSF is also bundled with Android Tamer, BlackArch and Pentoo.

https://github.com/MobSF/Mobile-Security-Framework-MobSF
https://mobsf.github.io/docs/

© 2025 by Bai7 GmbH | All rights reserved

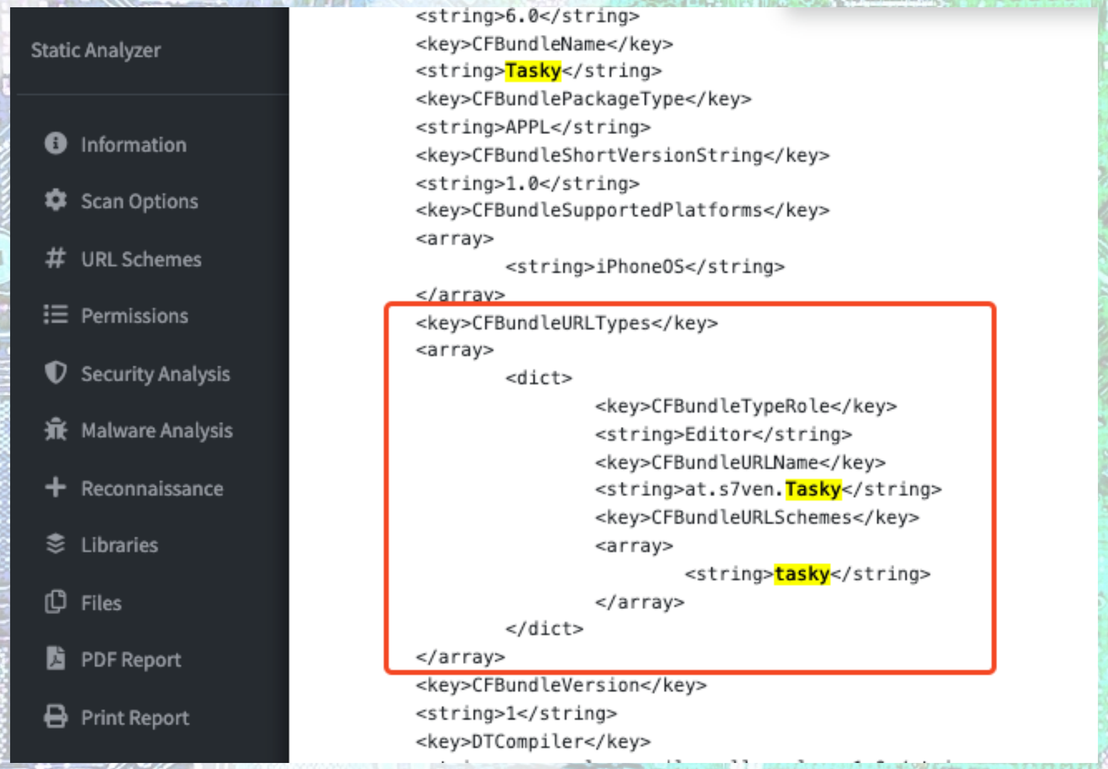
iOS - Deep Link



© 2025 by Bai7 GmbH | All rights reserved

iOS - Deep Link

To identify a deep link, you first need to inspect the Info.plist. The below example shows the scheme tasky.



© 2025 by Bai7 GmbH | All rights reserved

iOS - Deep Link

Exposing a deeplink is not a vulnerability by default!

Most deeplink functions are views, such as "fb://profile" that will bring a user to the Facebook profile page which is not a security risk at all.

With the information so far we know how to build the deeplink url in the app, but to find a vulnerable deeplink it is crucial to understand the triggered function by the deeplink.

Let’s dig deeper and have a look at the source code.

© 2025 by Bai7 GmbH | All rights reserved

iOS - Deep Link

What does this code do?

Tasky 〉 Tasky 〉 ToDoService 〉 ToDoService

```
11  class ToDoService: ObservableObject {
81      func handleDeeplink(_ incoming: URL) {
82
83          guard incoming.scheme?.lowercased() == "tasky" else { return }
84          guard let comps = URLComponents(url: incoming, resolvingAgainstBaseURL: false),
85                let urlValue = comps.queryItems?.first(where: { $0.name == "url" })?.value,
86                let candidate = URL(string: urlValue),
87                ["http","https"].contains(candidate.scheme?.lowercased() ?? "") else { return }
88
89          print(accessToken);
90
91          var req = URLRequest(url: candidate)
92          if let token = accessToken {
93              req.setValue("Bearer \(token)", forHTTPHeaderField: "Authorization")
94          } else {
95              print("No access token available for Authorization header")
96          }
97
98          DispatchQueue.main.async { self.deepLinkRequest = req }
99      }
```

Tasky 〉 Tasky 〉 TaskyApp 〉 body

```
8   import SwiftUI
9
10  @main
11  struct TaskyApp: App {
12
13      @StateObject var service = ToDoService()
14
15      var body: some Scene {
16          WindowGroup {
17              ContentView()
18                  .environmentObject(service)
19                  .onOpenURL { url in
20                      service.handleDeeplink(url)
21                  }
22          }
23      }
24  }
25
```

Tasky 〉 Tasky 〉 WebView 〉 makeUIView(context:)

```
11  struct WebView: UIViewRepresentable {
12      let request: URLRequest
13      func makeUIView(context: Context) -> WKWebView {
14          let wv = WKWebView()
15          wv.load(request)
16          return wv
17      }
18      func updateUIView(_ webView: WKWebView, context: Context) { }
19  }
```

Tasky 〉 Tasky 〉 ContentView 〉 ContentView

```
12  struct ContentView: View {
13      @EnvironmentObject var service: ToDoService
14
15      var body: some View {
16          VStack {
17              Divider()
18              ToDoMainView()
19          }
20
21          .sheet(item: Binding(
22              get: { service.deepLinkRequest.map { IdentifiedRequest(request: $0) } },
23              set: { _ in service.clearDeepLink() }
24          )) { item in
25              WebView(request: item.request)
26                  .ignoresSafeArea()
27          }
```

© 2025 by Bai7 GmbH | All rights reserved

iOS - Deep Link

What does this code do?

Tasky 〉 Tasky 〉 ToDoService 〉 ToDoService

```
11  class ToDoService: ObservableObject {
81      func handleDeeplink(_ incoming: URL) {
82
83          guard incoming.scheme?.lowercased() == "tasky" else { return }
84          guard let comps = URLComponents(url: incoming, resolvingAgainstBaseURL: false),
85                let urlValue = comps.queryItems?.first(where: { $0.name == "url" })?.value,
86                let candidate = URL(string: urlValue),
87                ["http","https"].contains(candidate.scheme?.lowercased() ?? "") else { return }
88
89          print(accessToken);
90
91          var req = URLRequest(url: candidate)
92          if let token = accessToken {
93              req.setValue("Bearer \(token)", forHTTPHeaderField: "Authorization")
94          } else {
95              print("No access token available for Author
96          }
97
98          DispatchQueue.main.async { self.deepLinkRequest = req }
99      }
```

url parameter is URL path for WebView

acessToken (JWT) is read

Tasky 〉 Tasky 〉 TaskyApp 〉 body

```
8   import SwiftUI
9
10  @main
11  struct TaskyApp: App {
12
13      @StateObject var service = ToDoService()
14
15      var body: some Scene {
16          WindowGroup {
17              ContentView()
18                  .environmentObject(service)
19                  .onOpenURL { url in
20                      service.handleDeeplink(url)
21                  }
22          }
23      }
24  }
25
```

url parameter is sent to handleDeeplink

Tasky 〉 Tasky 〉 WebView 〉 makeUIView(context:)

```
11  struct WebView: UIViewRepresentable {
12      let request: URLRequest
13      func makeUIView(context: Context) -> WKWebView {
14          let wv = WKWebView()
15          wv.load(request)
16          return wv
17      }
18      func updateUIView(_ webView: WKWebView, context: Context) { }
19  }
```

Tasky 〉 Tasky 〉 ContentView 〉 ContentView

```
12  struct ContentView: View {
13      @EnvironmentObject var service: ToDoService
14
15      var body: some View {
16          VStack {
17              Divider()
18              ToDoMainView()
19          }
20
21          .sheet(item: Binding(
22              get: { service.deepLinkRequest.map { IdentifiedRequest(request: $0) } },
23              set: { _ in service.clearDeepLink() }
24          )) { item in
25              WebView(request: item.request)
26                  .ignoresSafeArea()
27          }
```

© 2025 by Bai7 GmbH | All rights reserved

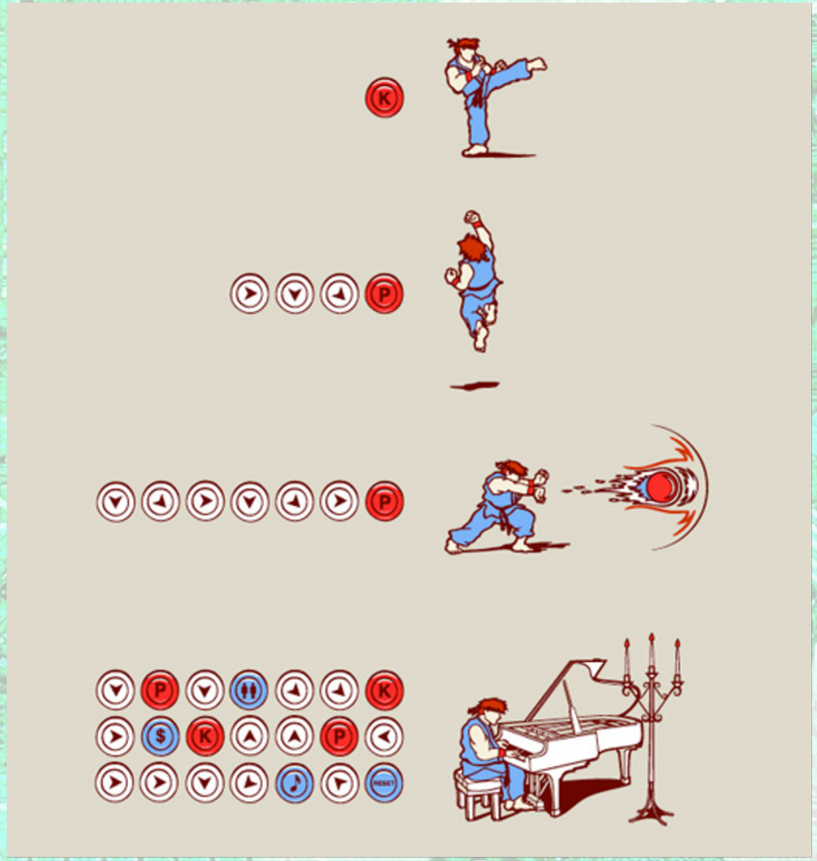
iOS - Deep Link

To summarise what we identified:

- Usage of a Deep Link
- An Open Redirect (the URL can be controlled through the parameter url)
- Usage of a WebView where the URL is rendered
- Usage of an authentication token that is added to the “Authorization” header

If you haven‘t noticed, these bugs can be chained together to achieve a “token leakage” (better bug name is welcomed).

An attacker can simply trick a user to click on the malicious deep link on the browser and the WebView will redirect user to an attacker's control server to receive the token.

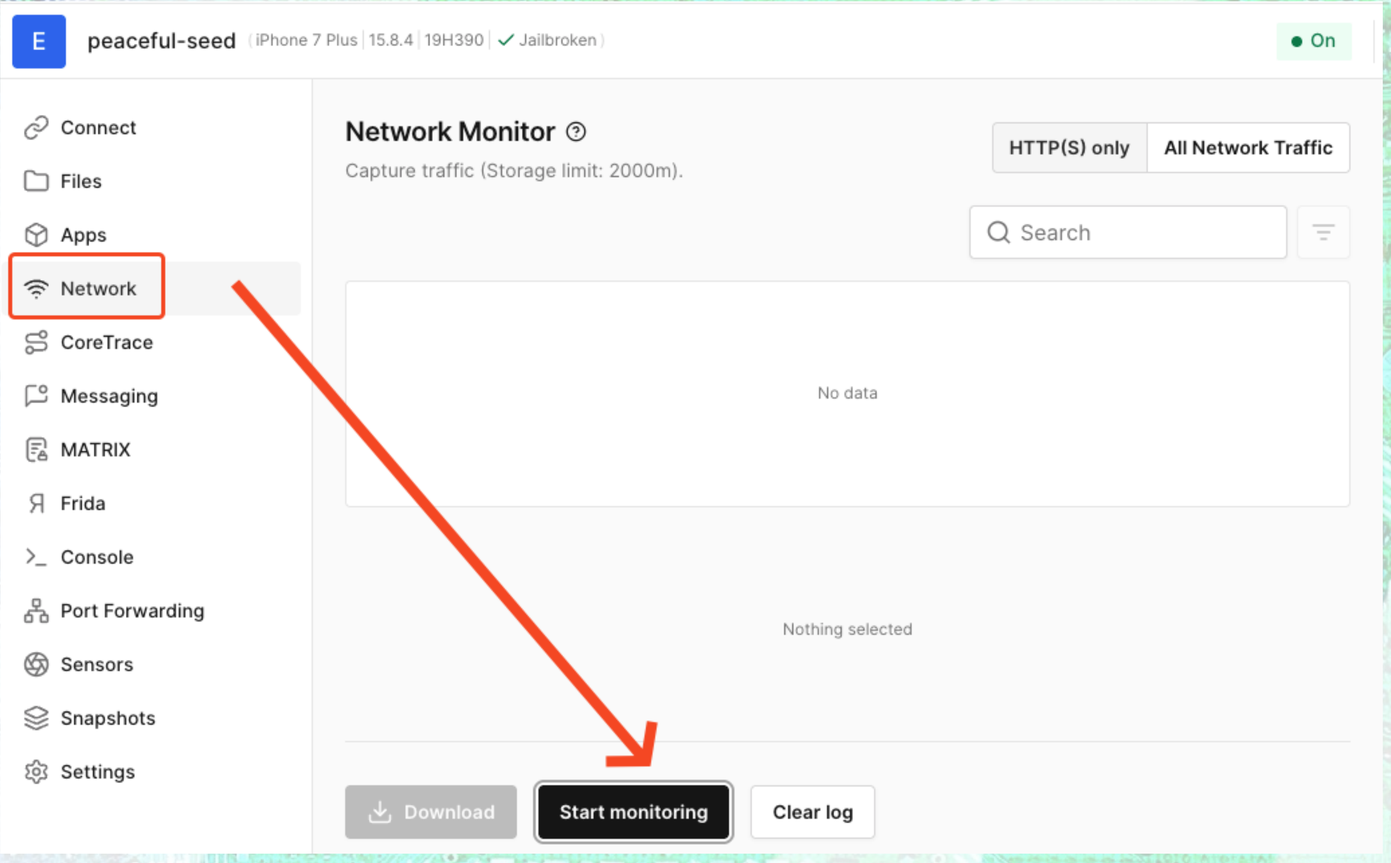


© 2025 by Bai7 GmbH | All rights reserved

iOS - Deep Link

Open “Network” and click on “Start Monitoring”.

This will capture all HTTP(S) requests.



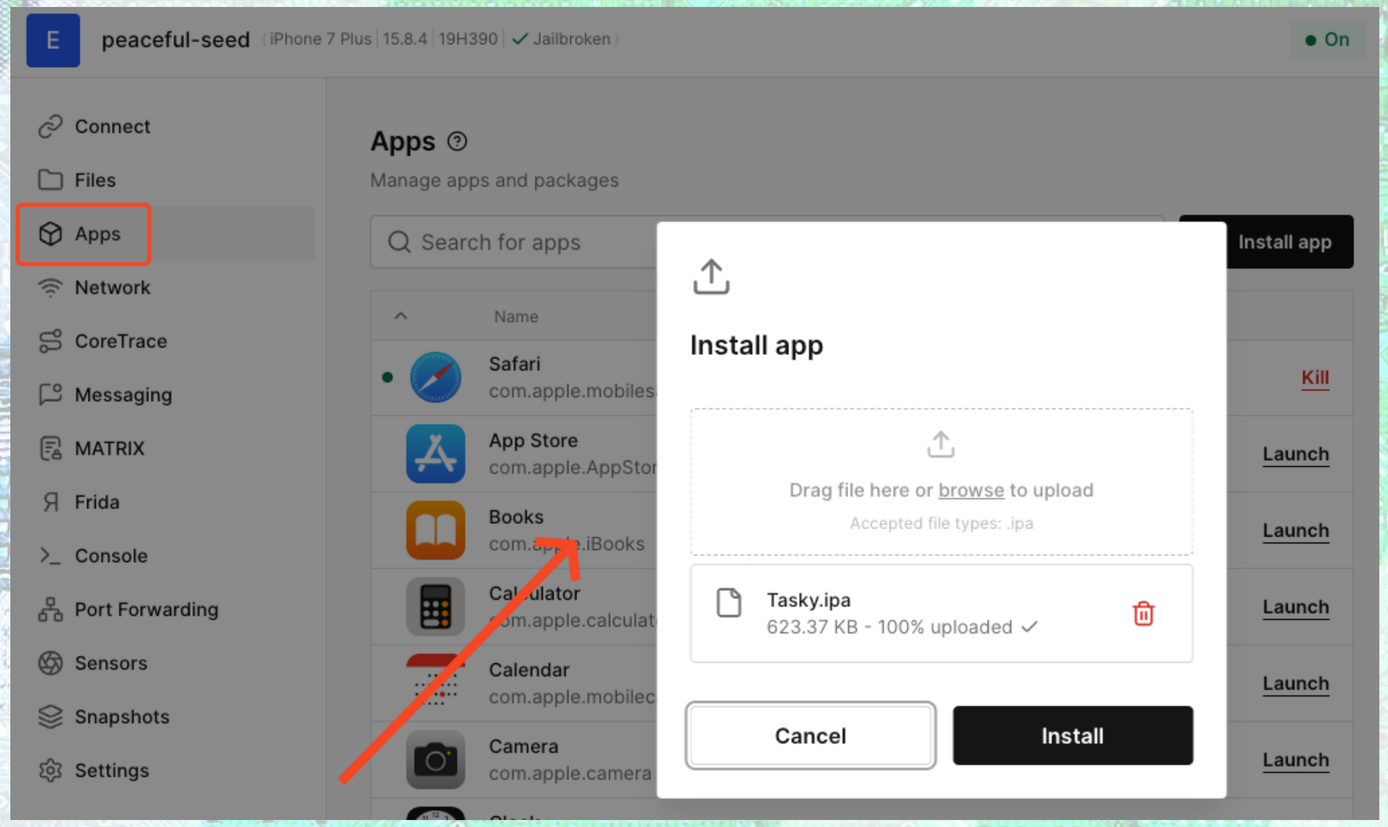
© 2025 by Bai7 GmbH | All rights reserved

iOS - Deep Link

Let’s exploit a deep link vulnerability together!

Select “Apps” in Correllium.

Then install Tasky by dropping the Tasky.ipa into Corellium apps section.
Or click the “Install App” button and select the IPA.



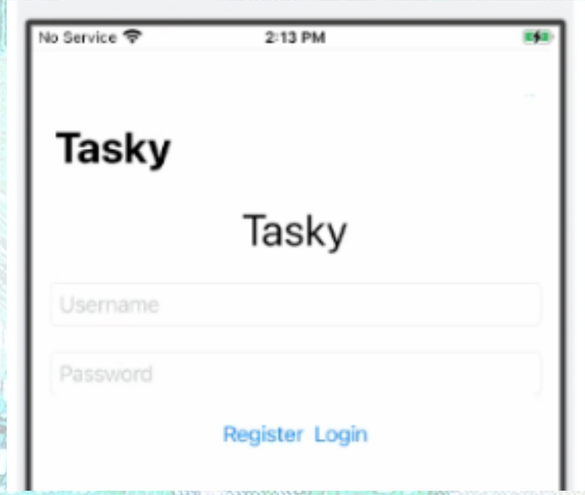
© 2025 by Bai7 GmbH | All rights reserved

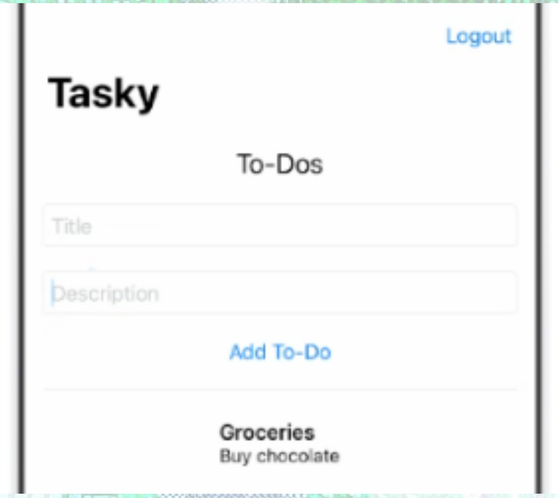
iOS - Deep Link

Once installed the start the app.

Choose a username and password, click on “Register”, wait for the message and if successful click on “Login”.

Once logged in you can create a To-Do.

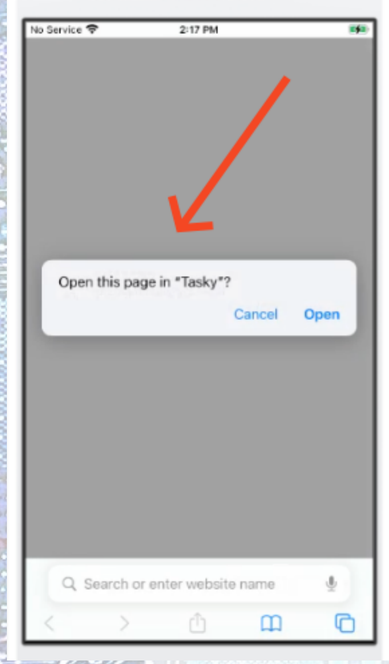




© 2025 by Bai7 GmbH | All rights reserved

iOS - Deep Link

Key in the following in Safari: tasky://open?url=https://httpbin.org/headers



© 2025 by Bai7 GmbH | All rights reserved

Steal JWT

Resources | Devices | Find device | BS

E peaceful-seed (iPhone 7 Plus | 15.8.4 | 19H390 | Jailbroken) ● On

Connect

Connect to your device

Quick Connect

Connect to your virtual device and use features like SSH without a VPN.

>_ SSH

Connect to SSH on your virtual device via an SSH tunnel without VPN.

```
ssh -J bf7d8c2c-0b89-49de-bd76-e721150cc3cd@proxy.enterprise.corellium.com root@10.11.1.1
```

The default password is alpine

Manage SSH keys here.

OR

Connect via VPN

To use gdb, ssh, or serial console you must first connect to VPN.

© 2025 by Bai7 GmbH | All rights reserved

iOS - Deep Link

What is the real risk of such a vulnerability?

An attacker can create a HTML page to lure a user to click this link or share the link to users via messaging services to steal their JWT access token. This allows to impersonate the users.

© 2025 by Bai7 GmbH | All rights reserved

iOS - Deep Links

How can we fix it?

© 2025 by Bai7 GmbH | All rights reserved

iOS - Deep Links

How can we fix it?

- Do not allow the URL as user input for the destination!
- If needed, have the user provide a short name, ID or token which is mapped to a full target URL, like:
  - 1 => https://customer.enterprise.com
  - 2 => https://vendor.enterprise.com

In general, if you need to process data coming from external:

- Do Input Validation
- Always use an Allow-list over a Deny-list

© 2025 by Bai7 GmbH | All rights reserved

iOS - Universal Links - Digital Asset Links

Universal Links use HTTP URLs that link to a website domain you own, so no other app can use your links. One of the requirements for Universal Links is that you verify ownership of your domain through one of our website association methods.

Declare the association between your website and your app by hosting a association file (JSON) at the following location:

https://<domain>/.well-known/apple-app-site-association

You must host the file using https:// with a valid certificate and with no redirects.

Example association file:

```
{
  "applinks": {
      "details": [
           {
             "appIDs": [ "ABCDE12345.com.example.app", "ABCDE12345.com.example.app2" ],
             "components": [
               {
                  "#": "no_universal_links",
                  "exclude": true,
                  "comment": "Matches any URL with a fragment that equals no_universal_links and instr
               },
               {
                  "/": "/buy/*",
                  "comment": "Matches any URL with a path that starts with /buy/."
               },
               {
                  "/": "/help/website/*",
                  "exclude": true,
                  "comment": "Matches any URL with a path that starts with /help/website/ and instruct
               },
               {
                  "/": "/help/*",
                  "?": { "articleNumber": "????" },
                  "comment": "Matches any URL with a path that starts with /help/ and that has a query
               }
             ]
           }
       ]
   },
   "webcredentials": {
      "apps": [ "ABCDE12345.com.example.app" ]
   },

    "appclips": {
        "apps": ["ABCDE12345.com.example.MyApp.Clip"]
    }
}
```

https://developer.apple.com/documentation/Xcode/supporting-associated-domains

© 2025 by Bai7 GmbH | All rights reserved

iOS - Deep Links - Detailed description of real-world exploitation

Research Threat intelligence Vulnerabilities and exploits · 11 min read

Vulnerability in TikTok Android app could lead to one-click account hijacking

By Microsoft Threat Intelligence

August 31, 2022

Vulnerabilities and exploits

Android

Microsoft discovered a high-severity vulnerability in the TikTok Android application, which could have allowed attackers to compromise users' accounts with a single click. The vulnerability, which would have required several issues to be chained together to exploit, has been fixed and we did not locate any evidence of in-the-wild exploitation. Attackers could have leveraged the vulnerability to hijack an account without users' awareness if a targeted user simply clicked a specially crafted link. Attackers could have then accessed and modified users' TikTok profiles and sensitive information, such as by publicizing private videos, sending messages, and uploading videos on behalf of users.

The vulnerability allowed the app's deeplink verification to be bypassed. Attackers could force the app to load an arbitrary URL to the app's WebView, allowing the URL to then access the WebView's attached JavaScript bridges and grant functionality to attackers. We've previously researched JavaScript bridges for their potential wide-reaching implications. Emphasizing the importance of exercising caution when clicking unknown links, this research also displays how collaboration within the security community is necessary to improve defenses for the overall digital ecosystem.

https://www.microsoft.com/en-us/security/blog/2022/08/31/vulnerability-in-tiktok-android-app-could-lead-to-one-click-account-hijacking/

© 2025 by Bai7 GmbH | All rights reserved

iOS - Deep Links - Detailed description of real-world exploitation

526

#401793

[Grab Android/iOS] Insecure deeplink leads to sensitive information disclosure

Share:

SUMMARY BY GRAB

A deeplink feature was found missing validation that led to sensitive information disclosure. Once triggered, the deeplink would direct users to load any attacker-controlled URL within a webview. The impact was further escalated as the webview contain sensitive information. A temporary patch was distributed shortly after the submission was verified and a permanent patch was released and completely rolled out soon after.

Grab appreciate @bagipro's contribution to our bug bounty program, @bagipro displayed strong mobile offensive security skills and detailed report which allowed us to quickly reproduce and validate the submission. As a mobile-first company, mobile security is our utmost focus, Grab look forward to seeing more of his creative bug reports to our program.

SUMMARY BY BAGIPRO

I've found a set of possible deeplinks, one of them (`HELPCENTER`) could lead that an arbtrary URL was opened in a built-in browser in activity `com.grab.pax.support.ZendeskSupportActivity` using that code (should be used in an external browser/messenger)

Code 235 Bytes — Unwrap lines Copy Download

```
<!DOCTYPE html>
<html>
<head><title>Page 1</title></head>
```

Reported August 28, 2018, 6:36pm UTC

bagipro

Participants

Reported to	Grab Managed
Report Id	#401793 Resolved
Disclosed	March 15, 2019, 11:41pm UTC
Severity	High (7.1)
Weakness	Cross-site Scripting (XSS) - Generic
Bounty	*None*
CVE ID	*None*
Account de...	*None*

https://hackerone.com/reports/401793
https://hackerone.com/bagipro

© 2025 by Bai7 GmbH | All rights reserved

iOS - Deep Links

Best Practices:

- Use Universal Links that allows a verification of the domain and iOS App to prevent link hijacking.
- Implement Strong Input Validation for any input coming from Deep links
- Ensure that deep links are processed only after authentication, to avoid data leakage.
- See also the deep link test case in the MASTG.

© 2025 by Bai7 GmbH | All rights reserved

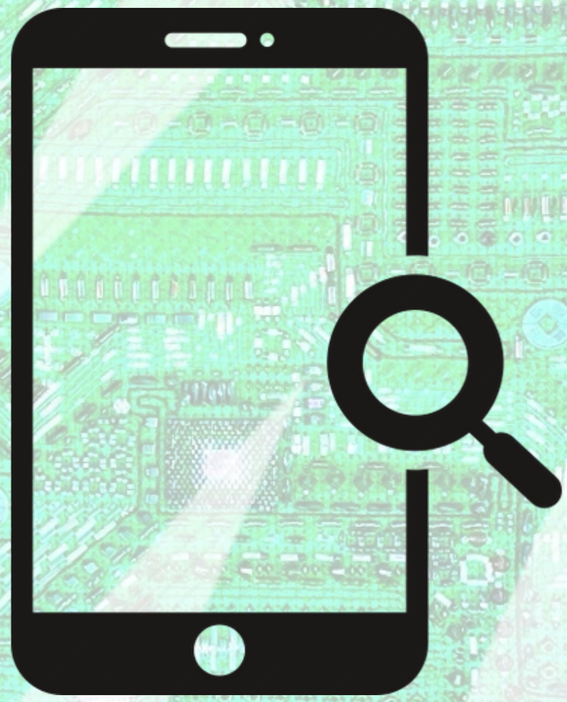
iOS Training

Static Scanning of a Swift project

iOS - Static Application Security Scanning (SAST)

mobsfscan

mobsfscan is a static analysis tool that can find insecure code patterns in your Android and iOS source code. Supports Java, Kotlin, Swift, and Objective C Code. mobsfscan uses MobSF static analysis rules and is powered by semgrep and libsast pattern matcher.



- Installation: $ pip install mobsfscan
- Can be used on cli, but also in Github Actions: https://github.com/MobSF/mobsfscan
- Export as SARIF possible, so can be opened in VS Code or integrated into Github Code Scanning

© 2024 by Bai7 GmbH | All rights reserved

iOS - Static Application Security Scanning (SAST) - SARIF

What is SARIF?

- Static Analysis Results Interchange Format (SARIF)
- Is an output file format and defined as JSON schema
- Is trying to harmonize security tool output and is becoming more popular in various security scanning tools.
- Supported by Github Security / Code Scanning alerts

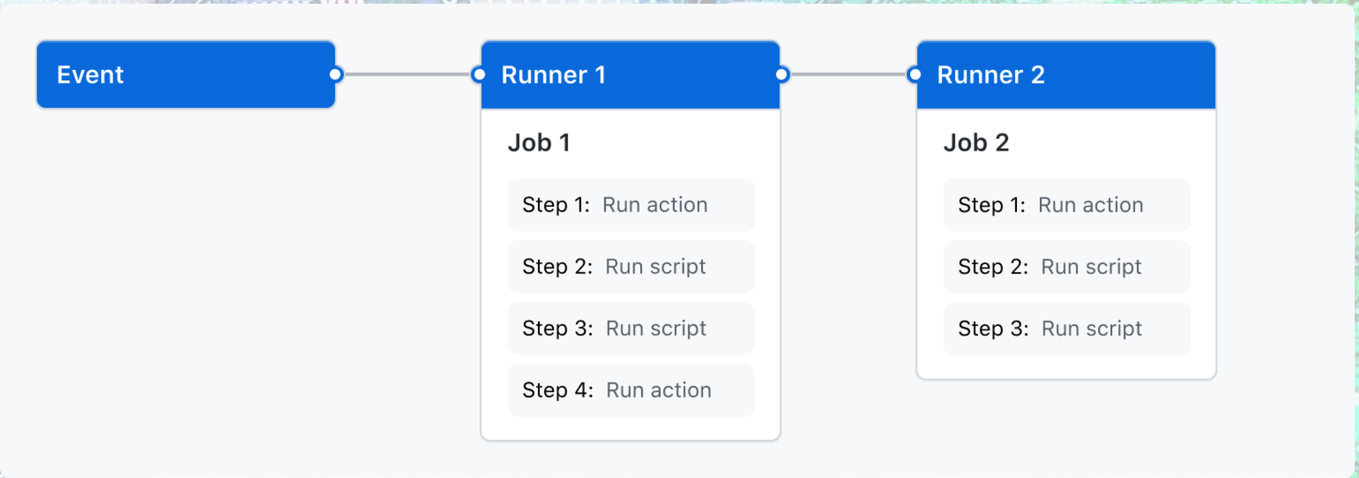

```
207        "results": [
319          {
320            "message": {
321              "text": "Insecure Implementation of SSL. Trusting all the certificates or
                 accepting self signed certificates is a critical Security Hole. This
                 application is vulnerable to MITM attacks."
322            },
323            "level": "error",
324            "locations": [
325              {
326                "physicalLocation": {
327                  "region": {
328                    "snippet": {
329                      "text": "        final SSLContext context = SSLContext.getInstance
                         (\"TLS\");\n        context.init(null, tmf.getTrustManagers(),
                         null);"
330                    },
331                    "endColumn": 58,
332                    "endLine": 95,
333                    "startColumn": 9,
334                    "startLine": 94
335                  },
336                  "artifactLocation": {
337                    "uri": "app/src/main/java/sg/vp/owasp_mobile/OMTG_Android/
                       OMTG_NETW_004_SSL_Pinning_Certificate.java"
338                  }
339                }
340              },
```


https://github.com/microsoft/sarif-tutorials

© 2024 by Bai7 GmbH | All rights reserved

iOS - Static Application Security Scanning (SAST) - Github Actions

- GitHub Actions is a continuous integration and continuous delivery (CI/CD) platform that allows you to automate your build, test, and deployment pipeline.
- You can create workflows that build and test every pull request to your repository, or deploy merged pull requests to production.



https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions

© 2024 by Bai7 GmbH | All rights reserved

iOS - Lab Time

Open the file Lab-Mobile-Security-Day-2.pdf

Section:
Lab – mobsfscan

Note: Some PDF readers do not copy properly and miss whitespaces etc.

© 2024 by Bai7 GmbH | All rights reserved

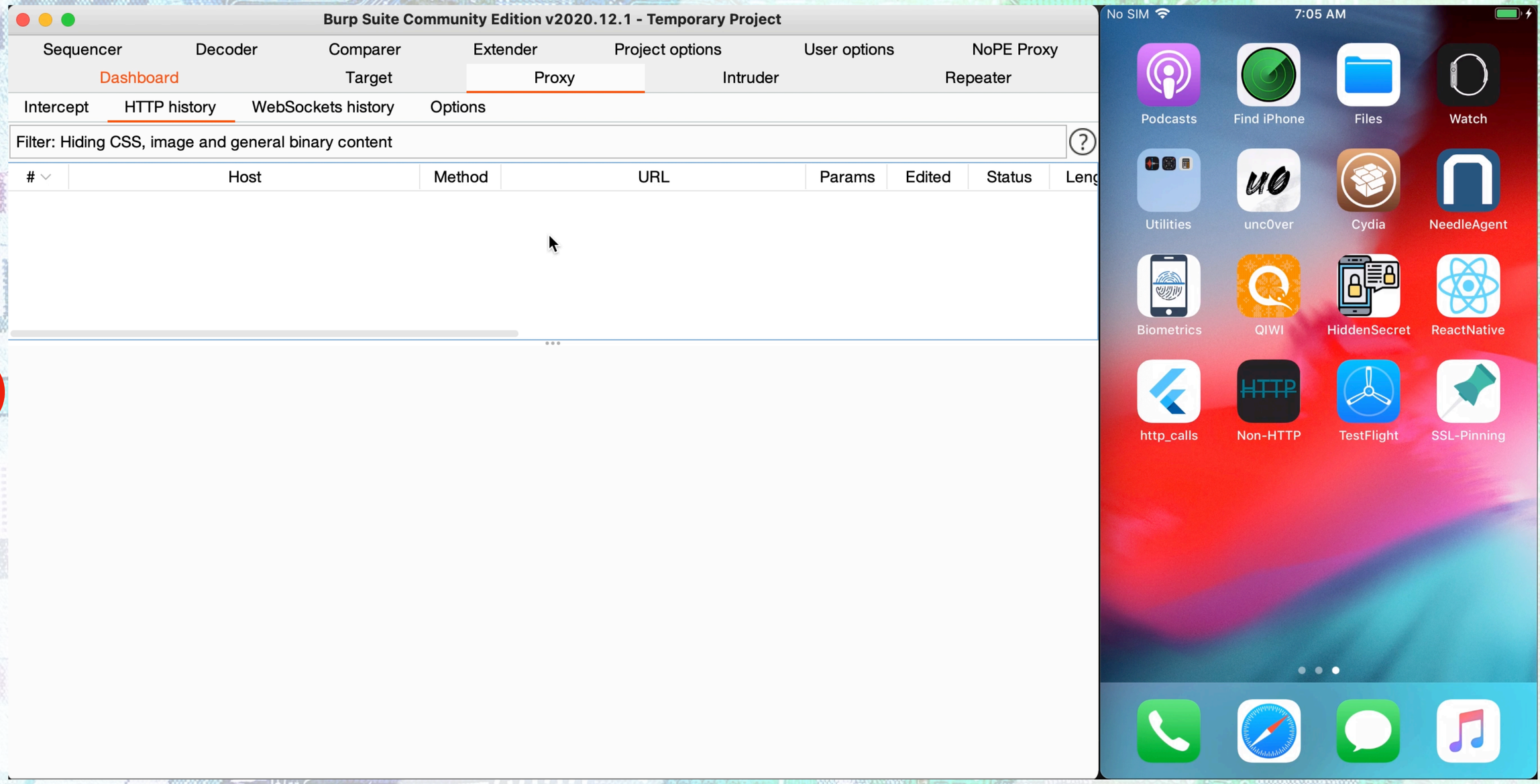
FIND VULNERABILITIES IN MOBILE APPS

Finding	False Positive	Vulnerability	Won't Fix	Comment
App allows self-signed or invalid SSL Certificates	✅			Code in Networking library AF Networking and is false positive. Should be checked if latest library is used.
Weak Hash algorithm used. The hash algorithm is known to have hash collisions.	✅			Crypto Library Imported but not used in security context.
SHA1 is a weak hash known to have hash collisions.	✅			Not used in a security context.
A key stored in the Keychain is using a weak accessibility value. Use stronger ACLs	✅			Used in Google Utilities and another Pod, but all false positives.
Files may contain hardcoded sensitive information like usernames, passwords, keys etc.		✅		An AWS Key and clear text password can be found. Rest is false positives.
Biometric auth should be based on Keychain, not based on bool.		✅		Not a secure implementation for biometrics
The App uses an insecure Random Number Generator	✅			Code in libraries used and PRNG not used in a security context.
This app does not have Jailbreak detection capabilities.			✅	Depends on the outcome of a threat model if needed
This app does not have Certificate Pinning implemented in code.			✅	
This app does not have Reverse engineering detection capabilities.			✅	

FIND VULNERABILITIES IN MOBILE APPS

Finding	False Positive	Vulnerability	Won't Fix	Comment
This app does not have custom keyboards disabled.			✅	Not applicable for secure text fields https://apple.co/3FmOcQ9
This app does not disable Keyboard cache. It must be disabled for all sensitive data inputs.		✅		See https://mas.owasp.org/MASTG/tests/ios/MASVS-STORAGE/MASTG-TEST-0055/
This app uses UIWebView	✅			UIWebView is used in the AFNetworking library. Library should be updated.
The App logs information. Sensitive information should never be logged. (ios_log)		✅		There is sensitive data being logged, such as a secret key in the Keychain. Rest are false positives.
The App logs information. Sensitive information should never be logged. (ios_app_logging)		✅		Credentials are logged. Rest are false positives.

iOS - SAST - Identity (Certificate) Pinning



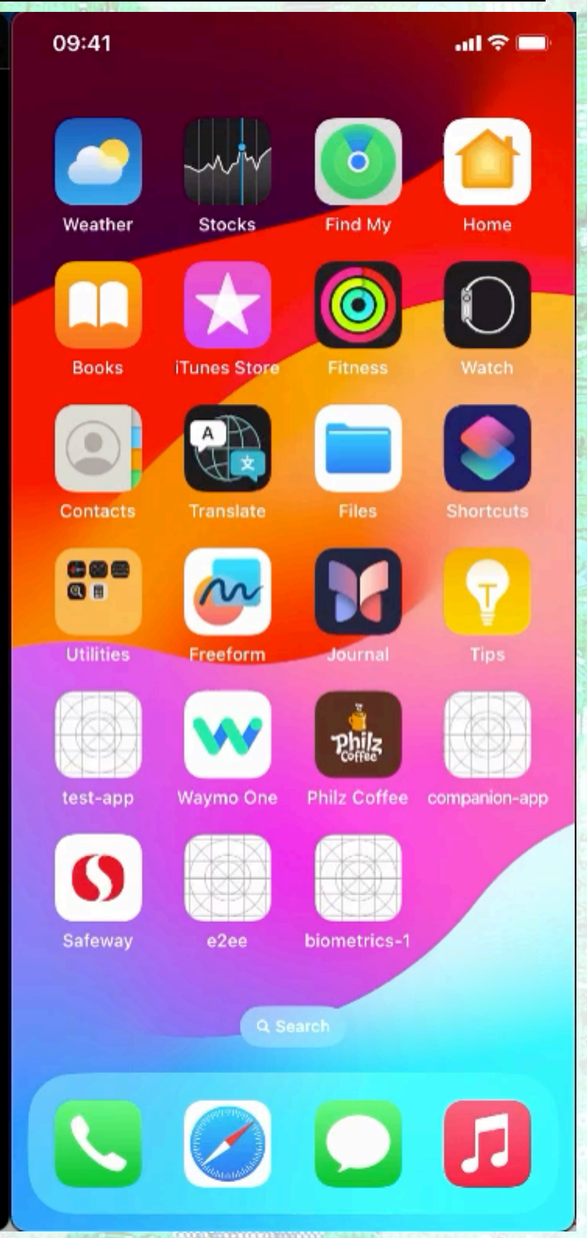
TBD - Pinning

© 2024 by Bai7 GmbH | All rights reserved

iOS - SAST - Biometric Authentication

Bypassing Biometric Authentication in an iOS app.

```
(lldb)
error: Process 0 is currently being debugged, kill the process before connecting
.
(lldb)
error: Process 0 is currently being debugged, kill the process before connecting
.
(lldb)
error: Process 0 is currently being debugged, kill the process before connecting
.
(lldb) process launch
error: Cannot launch '/private/var/containers/Bundle/Application/13809F99-FAA0-4
8C9-BF7A-47E772114A50/biometrics-1.app': The operation couldn't be completed. Un
able to launch * because the device was not, or could not be, unlocked.
(lldb) process launch
error: the platform is not currently connected
(lldb) platform select remote-ios
  Platform: remote-ios
 Connected: no
  SDK Path: "/Users/sven/Library/Developer/Xcode/iOS DeviceSupport/iPhone12,1 17
.6.1 (21G93)"
 SDK Roots: [ 0] "/Users/sven/Library/Developer/Xcode/iOS DeviceSupport/iPhone12
,1 17.6.1 (21G93)"
(lldb) target create '/Users/sven/Downloads/biometrics/Payload/biometrics-1.app'

Current executable set to '/Users/sven/Downloads/biometrics/Payload/biometrics-1
.app' (arm64).
(lldb) script lldb.target.module[0].SetPlatformFileSpec(lldb.SBFileSpec('/privat
e/var/containers/Bundle/Application/13809F99-FAA0-48C9-BF7A-47E772114A50/biometr
ics-1.app'))
True
(lldb) process connect connect://[fdea:d9e1:c9cb::1]:49447
(lldb) process connect connect://[fdea:d9e1:c9cb::1]:49447
error: Process 0 is currently being debugged, kill the process before connecting
.
(lldb) process launch
```



© 2024 by Bai7 GmbH | All rights reserved

iOS - SAST - Biometric Authentication

Why is it possible to bypass Biometric Authentication?

the objection bypass

Within `objection`, when you run the `ios ui biometrics_bypass` command, a hook is executed that listens for invocations of the `-[LAContext evaluatePolicy:localizedReason:reply:]` selector. If the `evaluatePolicy` method is called, the hook will replace the `success` boolean to a `True` in the code block that is executed when a `reply` is received.

what it does not and can not do

Once the `reply` has had its success boolean set to `true`, and original, intended code block is executed, just like it would have under normal conditions. If this code block contains further validations, then those need to be inspected separately and hooked/manipulated appropriately.

What should be clear now is that this 'bypass' is purely a local bypass for when a target application calls `evaluatePolicy` and a failed response is received. This 'bypass' will not work in cases where keychain items are protected with access control flags such as `kSecAccessControlTouchIDAny` or `kSecAccessControlTouchIDCurrentSet`.

The verification of a successful Face ID / Touch ID Authentication when implemented via LAContext is event-based and done within the App and can therefore be manipulated!

https://github.com/sensepost/objection/wiki/Understanding-the-iOS-Biometrics-Bypass

© 2024 by Bai7 GmbH | All rights reserved

iOS - SAST - Biometric Authentication

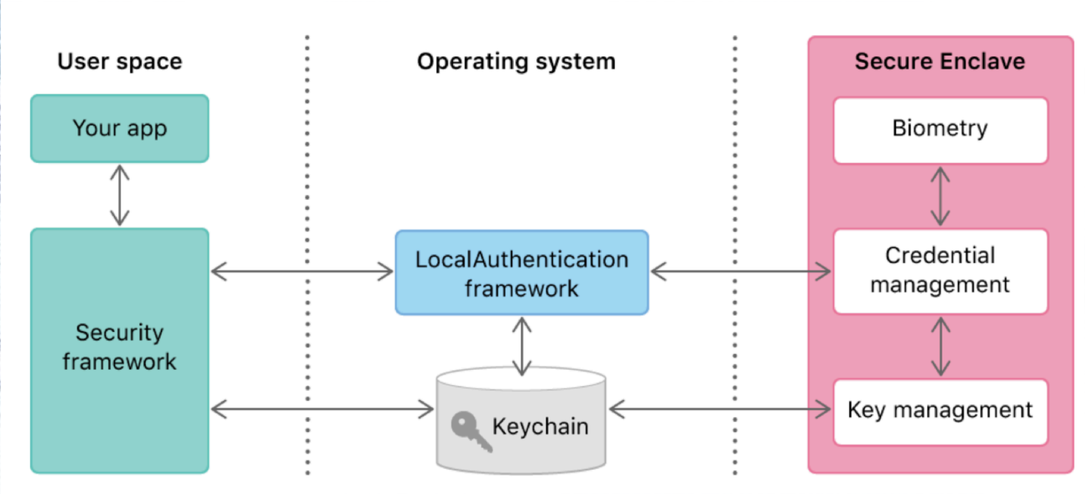
Vulnerable implementation of biometric authentication

```
37          var authError: NSError?
38          let reasonString = "To access the secure data"
39
40          if localAuthenticationContext.canEvaluatePolicy(.deviceOwnerAuthenticationWithBiometrics, error: &authError) {
41
42              localAuthenticationContext.evaluatePolicy(.deviceOwnerAuthenticationWithBiometrics, localizedReason:
                    reasonString) { success, evaluateError in
43
44                  DispatchQueue.main.async{
45
46                  if success {
47
48                      //TODO: User authenticated successfully, take appropriate action
```

© 2024 by Bai7 GmbH | All rights reserved

iOS - SAST - Biometric Authentication

Vulnerable implementation of biometric authentication



2 different implementations are available:

- ~~Local Authentication Framework only (LAContext)~~
- LAContext together with KeyChain Services

- App stores either a secret authentication token or another piece of secret data identifying the user in the Keychain.
- A valid set of biometrics must be presented before the key is released from the Secure Enclave to decrypt the keychain entry itself.
- This solution cannot be bypassed (even on jailbroken devices), as the verification is done within the Secure Enclave (SE).

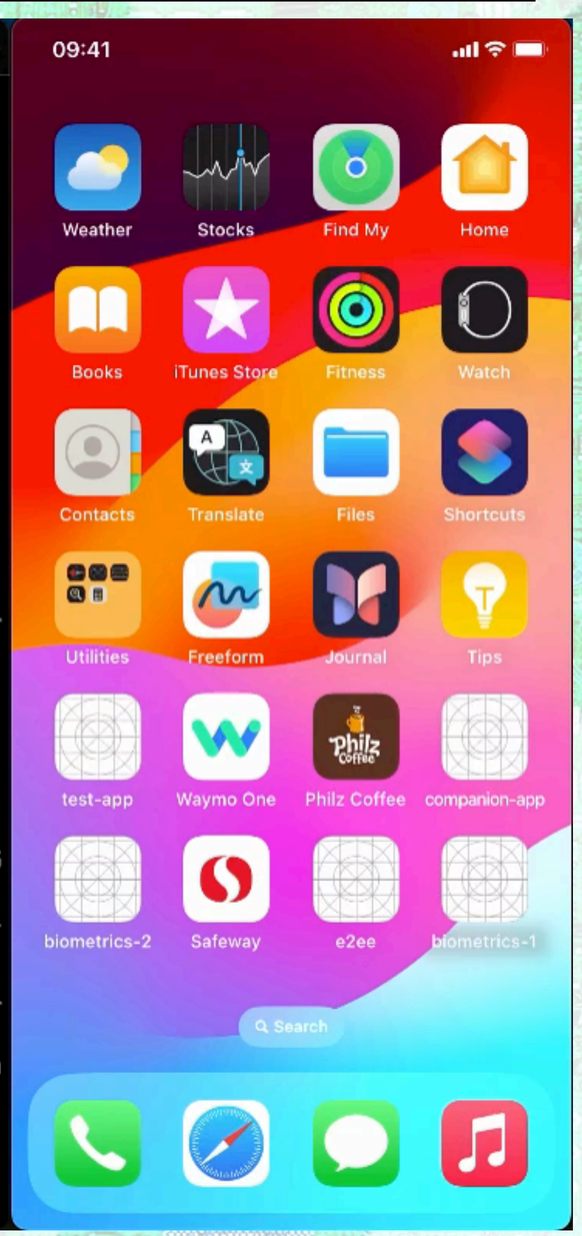
https://apple.co/2KUscTr

© 2024 by Bai7 GmbH | All rights reserved

iOS - SAST - Biometric Authentication

Bypass Biometric Authentication… NOT

```
(lldb)
True
(lldb)
True
(lldb)
True
(lldb)
True
(lldb)
True
(lldb)
True
(lldb)
True
(lldb)
True
(lldb) process connect connect://[fd9a:e98d:fc0f::1]:55476
(lldb) process launch
Process 5529 launched: '/Users/sven/Downloads/biometrics/Payload/biometrics-2.app/biometr
ics-2' (arm64)
Process 5529 exited with status = 0 (0x00000000)
(lldb) process launch
error: the platform is not currently connected
(lldb) platform select remote-ios
  Platform: remote-ios
 Connected: no
  SDK Path: "/Users/sven/Library/Developer/Xcode/iOS DeviceSupport/iPhone12,1 17.6.1 (21G
93)"
 SDK Roots: [ 0] "/Users/sven/Library/Developer/Xcode/iOS DeviceSupport/iPhone12,1 17.6.1
 (21G93)"
(lldb) target create '/Users/sven/Downloads/biometrics/Payload/biometrics-2.app'
Current executable set to '/Users/sven/Downloads/biometrics/Payload/biometrics-2.app' (ar
m64).
(lldb) script lldb.target.module[0].SetPlatformFileSpec(lldb.SBFileSpec('/private/var/con
tainers/Bundle/Application/70C9B3B8-91DE-4727-90C4-8BCA2779199B/biometrics-2.app'))
True
(lldb) process connect connect://[fd9a:e98d:fc0f::1]:55476
(lldb) process launch
```



© 2024 by Bai7 GmbH | All rights reserved

iOS - SAST - Summary

- Request for access to the iOS source code
- Tools like mobsfscan can be used to scan the source code
- By using SARIF and VS Code it is manageable to go through the findings and have the direct link back to the source code and root cause of the issue.

© 2024 by Bai7 GmbH | All rights reserved

iOS Training

App Transport Security (ATS)

App Transport Security (ATS)

APP TRANSPORT SECURITY (ATS)

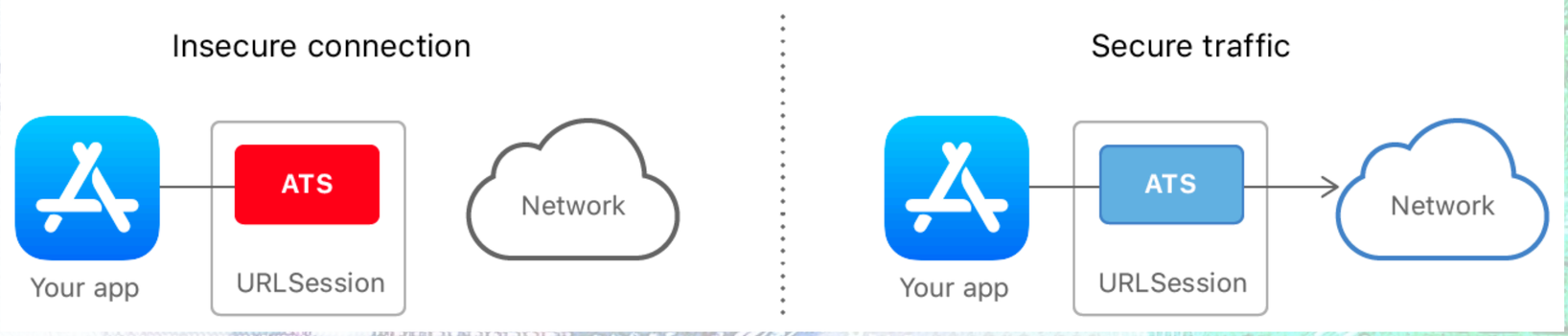
Search:

ISSUE	STATUS	DESCRIPTION
App Transport Security AllowsArbitraryLoads is allowed	insecure	App Transport Security restrictions are disabled for all network connections. Disabling ATS means that unsecured HTTP connections are allowed. HTTPS connections are also allowed, and are still subject to default server trust evaluation. However, extended security checks like requiring a minimum Transport Layer Security (TLS) protocol version—are disabled. This setting is not applicable to domains listed in NSExceptionDomains.

© 2024 by Bai7 GmbH | All rights reserved

App Transport Security (ATS)

- Improves privacy and data integrity for all apps and app extensions by enforcing that all HTTP connections use HTTPS
- Is enforcing a Minimum Security Specification on the URL Loading System (class URLSession)



© 2024 by Bai7 GmbH | All rights reserved

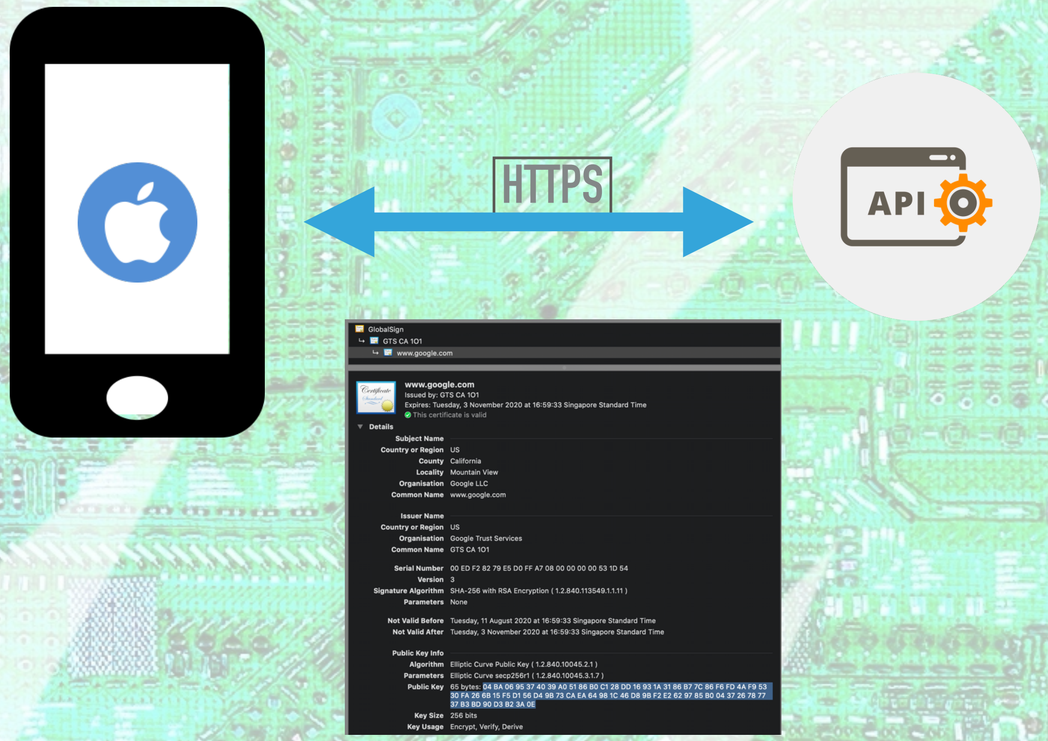
App Transport Security (ATS)

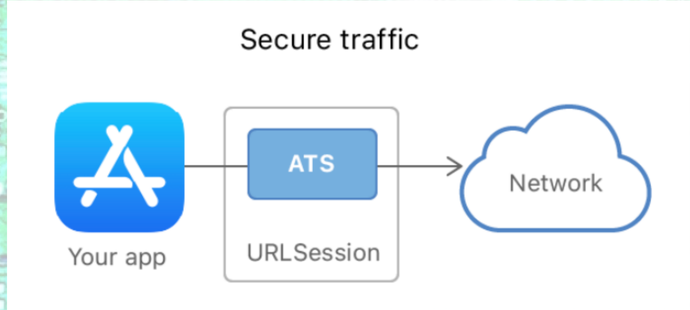
Verify server certificate during TLS Handshake, like:

- checking that the certificate isn't expired,
- that the name matches the server's DNS name,
- that the signature of the certificate hasn't been tampered with and
- that it's signed by a trusted CA.

ATS enforces additionally in the app (examples):

- Connection must use Transport Layer Security (TLS) protocol version 1.2 or later
- Certificate must use the SHA-256 or greater





© 2024 by Bai7 GmbH | All rights reserved

App Transport Security (ATS)

ATS exceptions are configured in the Info.plist file. Examples of common exceptions:

```
5    <key>NSAppTransportSecurity</key>
6    <dict>
7        <key>NSAllowsArbitraryLoads</key>
8        <true/>
```

A Boolean value indicating whether App Transport Security restrictions are disabled for all network connections.

```
5    <key>NSAppTransportSecurity</key>
6    <dict>
7        <key>NSExceptionDomains</key>
8        <dict>
9            <key>example.com</key>
10           <dict>
11               <key>NSExceptionAllowsInsecureHTTPLoads</key>
12               <true/>
13               <key>NSIncludesSubdomains</key>
14               <true/>
```

Set the value for the key NSExceptionAllowsInsecureHTTPLoads to YES to allow insecure HTTP loads for the given domain (example.com).

Set the value for the key NSIncludesSubdomains to YES to apply the ATS exceptions for the given domain to all subdomains of the domain.

Listing of all ATS exceptions: https://developer.apple.com/documentation/bundleresources/information_property_list/nsapptransportsecurity#topics

© 2024 by Bai7 GmbH | All rights reserved

App Transport Security (ATS)

During a penetration test exceptions might be found for ATS!

- Testing usually not the production build, but a UAT build.
- In test environments there might be:
  - self signed certificates or
  - weak TLS server settings.
- Therefore ATS exceptions are widely used in UAT but the ATS configuration in production might be completely different.
- Share the Info.plist from the production build, so they can verify the prod settings.
- **Removing the exceptions from Info.plist might break the app!**

© 2024 by Bai7 GmbH | All rights reserved

App Transport security (ATS) - nsurl

Not sure if you break your App when using ATS? Use the following command, the endpoint might be more mature than you think (only working on macOS):

```
➜  ~ /usr/bin/nscurl --ats-diagnostics --verbose https://example.com
Starting ATS Diagnostics

Configuring ATS Info.plist keys and displaying the result of HTTPS loads to https://example.com.
A test will "PASS" if URLSession:task:didCompleteWithError: returns a nil error.
================================================================================

Default ATS Secure Connection
---
ATS Default Connection
ATS Dictionary:
{
}
Result : PASS
---
```

https://developer.apple.com/documentation/security/preventing_insecure_network_connections/

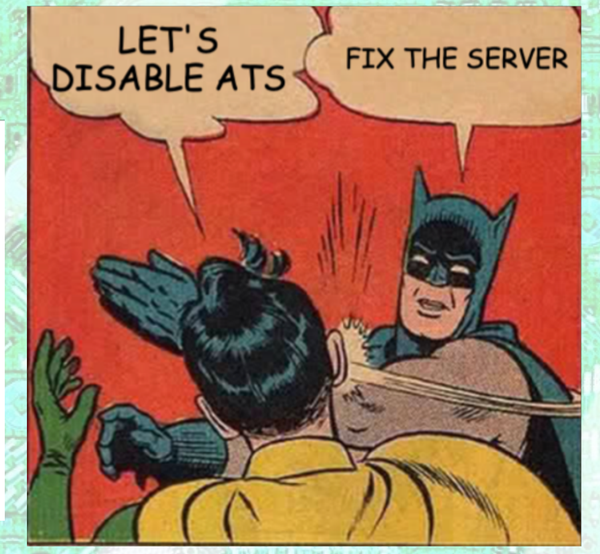
© 2024 by Bai7 GmbH | All rights reserved

App Transport security (ATS) - Exceptions

If an endpoint is not capable of supporting ATS, the server configuration should be updated!

Important

It's always better to fix the server when faced with an ATS failure. Exceptions reduce the security of your app. Some also require justification when submitting an app to the App Store, as described in the next section.



TLS Best Practices: https://github.com/ssllabs/research/wiki i
SSL Configuration Generator: https://ssl-config.mozilla.org/

© 2024 by Bai7 GmbH | All rights reserved

App Transport security (ATS) - Summary

Reference:

https://developer.apple.com/documentation/security/preventing_insecure_network_connections

https://developer.apple.com/documentation/bundleresources/information_property_list/nsapptransportsecurity

https://www.nowsecure.com/blog/2017/08/31/security-analysts-guide-nsapptransportsecurity-nsallowsarbitraryloads-app-transport-security-ats-exceptions/

https://github.com/ssllabs/research/wiki

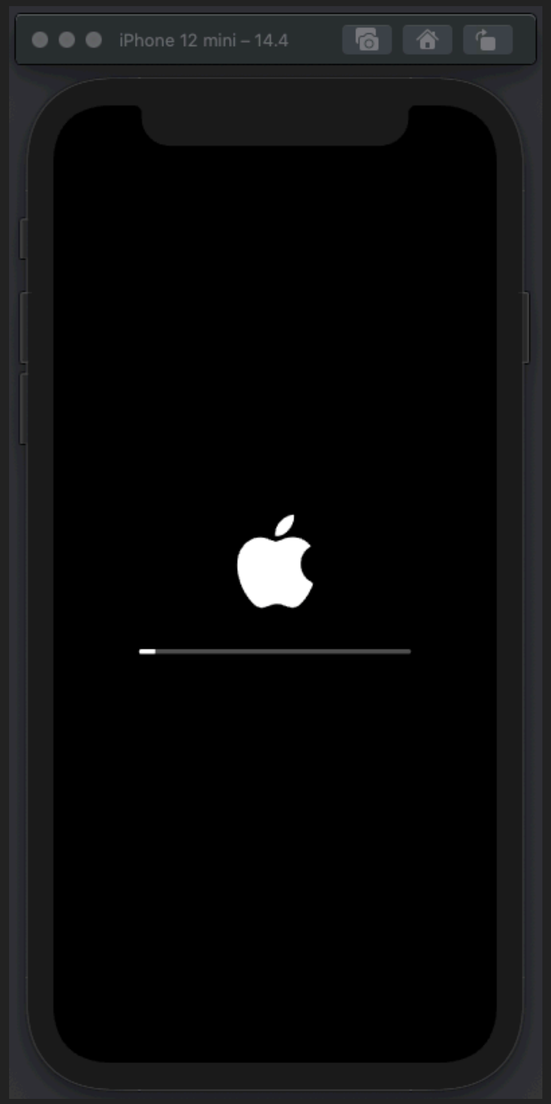
© 2024 by Bai7 GmbH | All rights reserved

MASTG HANDS-ON - IOS EDITION

IOS SECURITY TESTING BASICS

IOS SIMULATOR

- Part of Xcode tools
- Prototyping and test builds of your app during the development process.
- It is not possible to install an IPA from the App Store, this can only be done on a real device. ".app" files can be created in Xcode and installed on the iOS Simulator (see here)
- Limitations of the Simulator includes missing support for hardware features like Camera, Motion Support, Barometer and more.



IOS EMULATOR

- An emulator is capable to run iOS apps natively.
- Apple doesn't provide an iOS emulator currently, and only a small number of vendors offers such a service (such as Corellium).
- iOS emulation is a “Grey area”. Even though emulation is technically possible, iOS can only run on Apple Hardware according to the Apple license agreement.
- **For this training you will be using Corellium!**

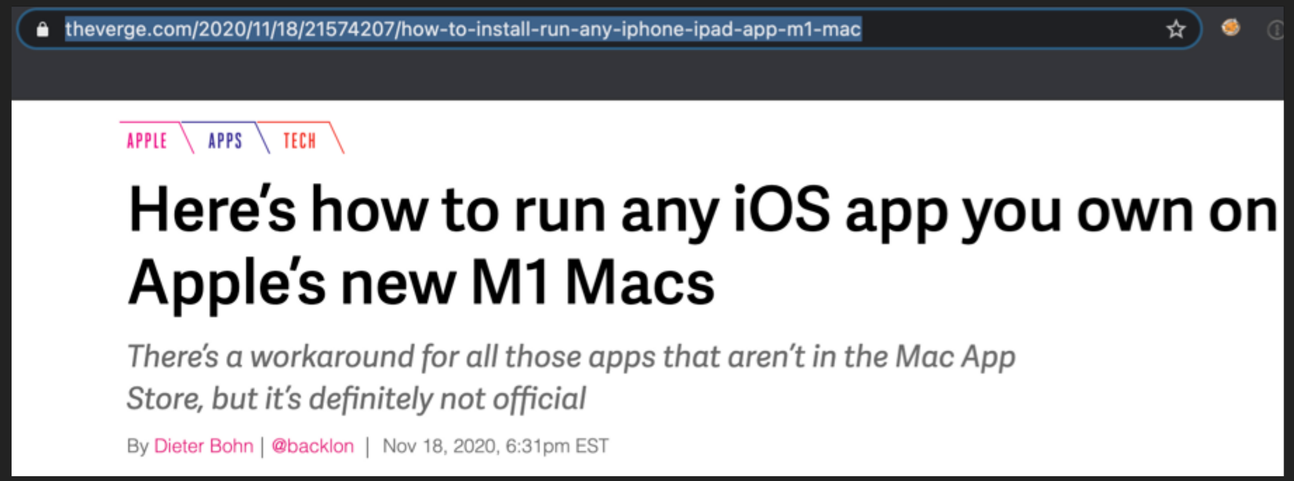
SECURITY TESTING WITH A PHYSICAL DEVICE

- Physical device is the preferred option to learn iOS application security.
- All hardware features are available
- Better performance than virtual devices
- **For testing with Jailbreak**: Bare minimum should be iPhone 6s or iPhone SE and newer as they still supports iOS 15 (but not all might support such an outdated iOS version anymore)
- **For testing without Jailbreak**: Bare minimum should be iPhone XS or iPhone SE (3rd Gen.) and newer as they still support iOS 17 and iOS 18

IOS APPS RUN NATIVELY ON APPLE SILICON!

iOS Apps are build for ARM and the new Silicon Macs are based on the new M1/M2/M3 System on a Chip which is also ARM based.

This allows to run iOS apps natively on a Mac (iPhone is still preferred for testing).



https://www.theverge.com/2020/11/18/21574207/how-to-install-run-any-iphone-ipad-app-m1-mac

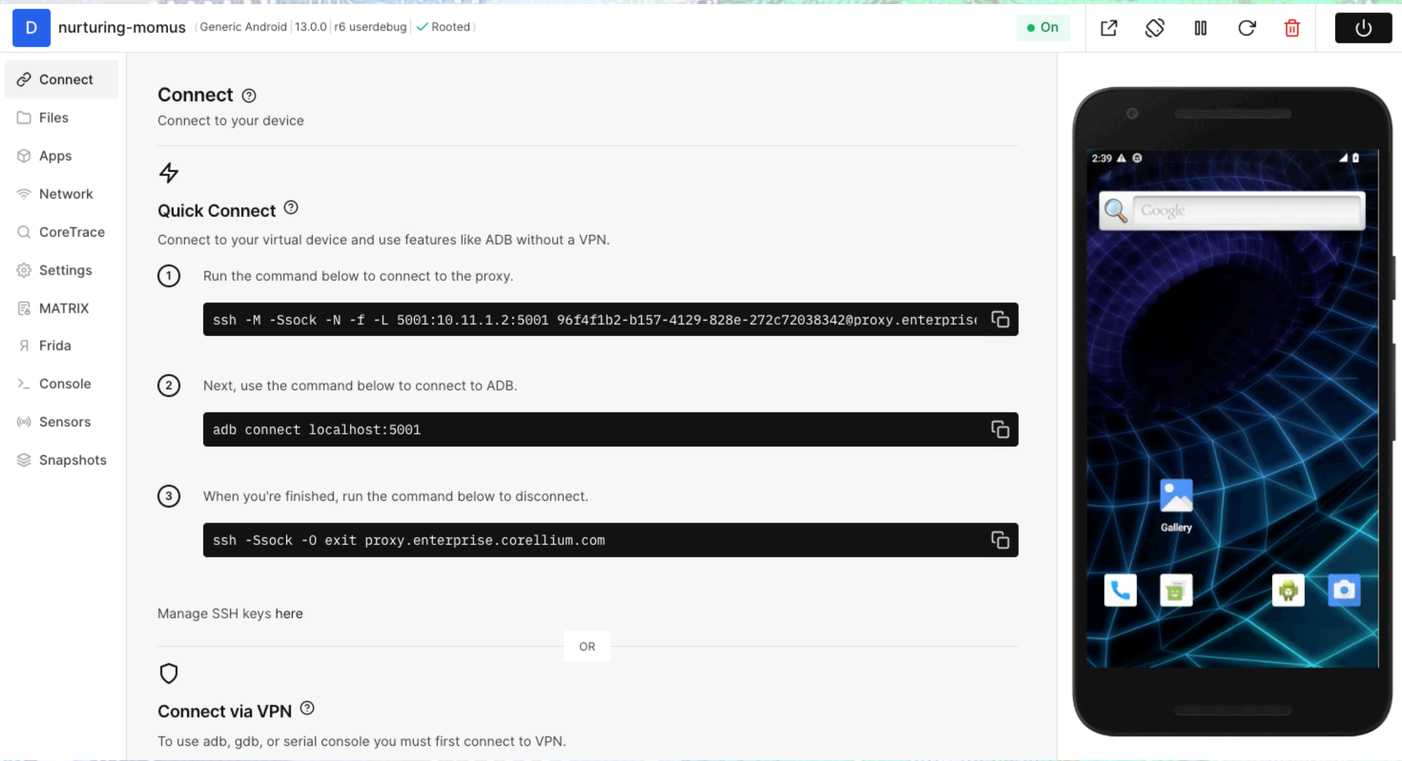
https://twitter.com/SparkZheng/status/1334808131981910019

Android Training

Corellium Setup

Corellium

- Corellium is SaaS and allows to emulate iOS in the cloud and also Android
- Please follow the next slides to start your Android device



© 2024 by Bai7 GmbH | All rights reserved

Corellium - Setup

Everybody is getting one rooted Android device, please stick to it!

Password: Cyber_2023

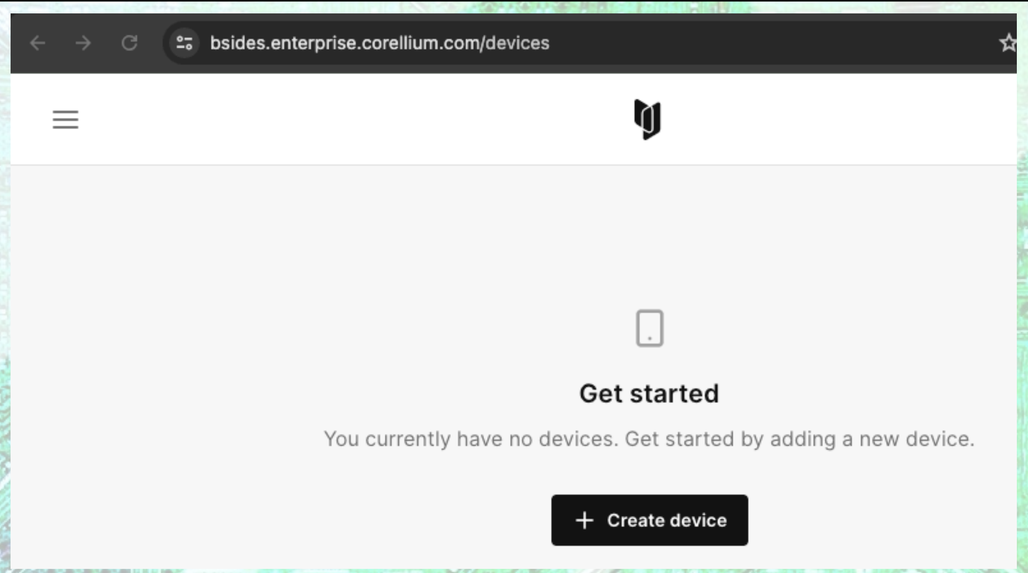
Username	Student (last name)	Username	Student (last name)
01@s7ven.info		16@s7ven.info	
02@s7ven.info		17@s7ven.info	
03@s7ven.info		18@s7ven.info	
04@s7ven.info		19@s7ven.info	
05@s7ven.info		20@s7ven.info	
06@s7ven.info		21@s7ven.info	
07@s7ven.info		22@s7ven.info	
08@s7ven.info		23@s7ven.info	
09@s7ven.info		24@s7ven.info	
10@s7ven.info		25@s7ven.info	
11@s7ven.info		26@s7ven.info	
12@s7ven.info		27@s7ven.info	
13@s7ven.info		28@s7ven.info	
14@s7ven.info		29@s7ven.info	
15@s7ven.info		30@s7ven.info	

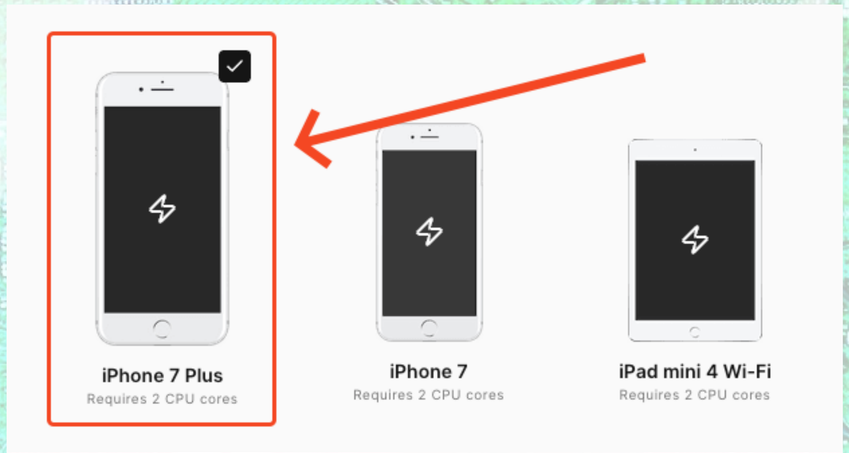
© 2024 by Bai7 GmbH | All rights reserved

Corellium - Setup

Disable your VPN first !

- Go to https://bsides.enterprise.corellium.com
- Login with your credentials
- Click on “Create Device”
- Select your project (there should only be one)
- Filter for iOS and select “iPhone 7 Plus”

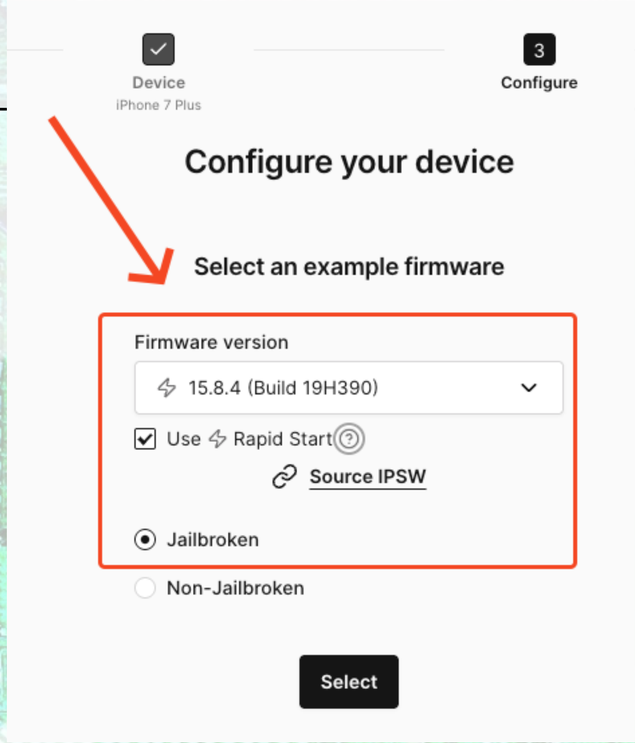


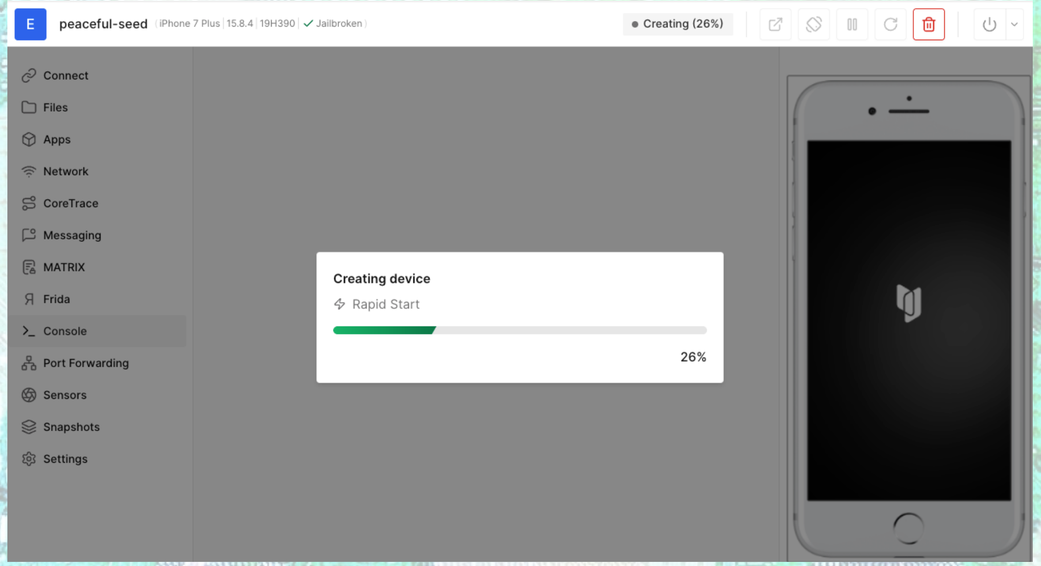


© 2024 by Bai7 GmbH | All rights reserved

Corellium - Setup

- Select iOS 15.8.4
- Tick“Use Rapid Start”
- Select “Jailbroken"
- Click on “Select” and then “Create device”





© 2024 by Bai7 GmbH | All rights reserved

iOS Training

Jailbreaks

iOS Security Architecture and Jailbreaks

iOS Security Architecture… is 0wned

- A jailbreak will disable some of these iOS security features, namely Apple's code signing mechanism so that arbitrary unsigned code can be run.
- Jailbreaking is usually the result of a series of vulnerabilities being chained together to bypass numerous iOS security mechanisms and restrictions.
- Not all jailbreaks are the same; it depends on the availability and exploitability of the bugs in the iOS version.

Jailbreaking your device might damage it and void its warranty. We hold no responsibility for your actions.

© 2024 by Bai7 GmbH | All rights reserved

iOS Security Architecture and Jailbreaks

The iOS security architecture consists of several core features:

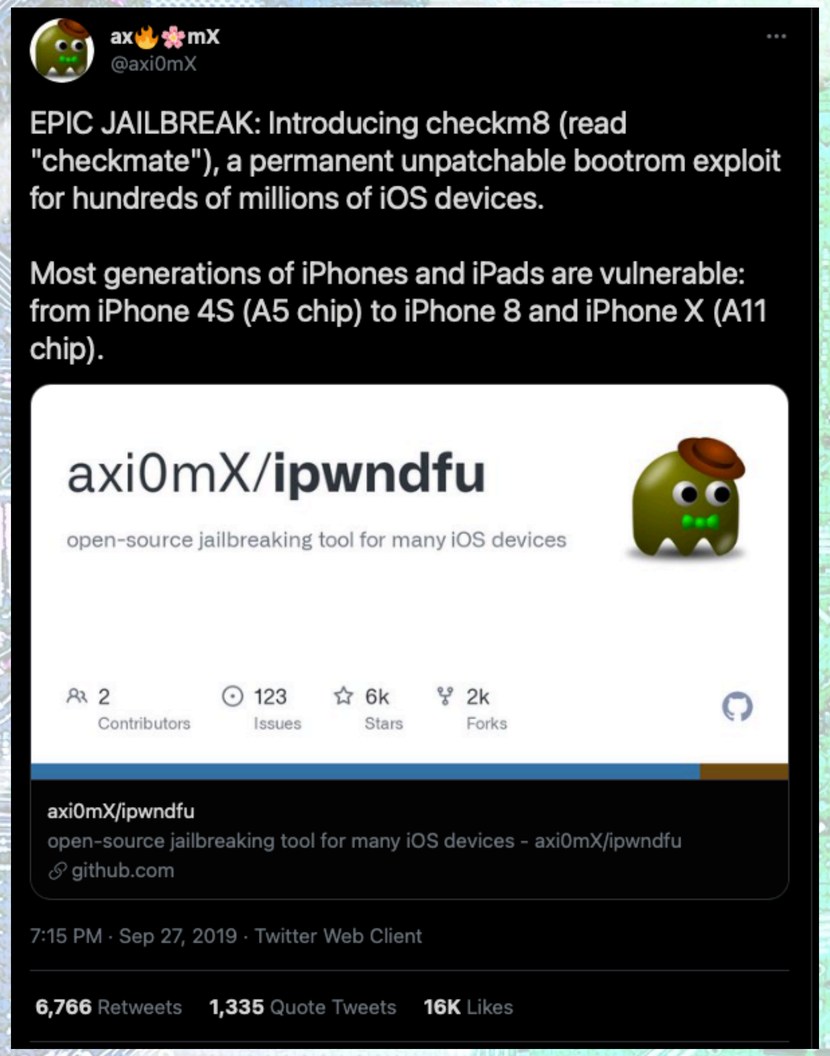
- Hardware Security (Secure Enclave) => Still considered secure, even on a jailbroken device
- ~~Code Signing~~
- ~~Sandboxing~~

“Apple Platform Security” (which also includes iOS):

https://support.apple.com/en-gb/guide/security/welcome/web

© 2024 by Bai7 GmbH | All rights reserved

iOS Security Architecture and Jailbreaks



https://twitter.com/axi0mx/status/1177542201670168576

© 2024 by Bai7 GmbH | All rights reserved

iOS Security Architecture and Jailbreaks

CheckM8 - bootrom exploit, Apple cannot patch it!

- Exploit (not a Jailbreak) working for any iPhone device since 2010 (iPhone 4S to iPhone X)
- Downgrade to any iOS version
- USB Access needed (no remote exploit!)

This exploit is used in palerai1n, still one of the main jailbreaks that penetration testers use to assess iOS Apps.

https://github.com/axi0mX/ipwndfu
https://theapplewiki.com/wiki/Palera1n

© 2024 by Bai7 GmbH | All rights reserved

iOS Security Architecture and Jailbreaks

- Since iOS 15 a “sealed Root File System” is implemented
- Jailbreaks were usually modifying the root FS and mounting it r/w
- Modifications to it in iOS 15 would cause a bootloop
- Public jailbreaks are available for iOS 15, but are very limited for iOS 16 and iOS 17 and are not existent for iOS 18.

iOS

iOS	Jailbreak Tool	Tool Version	iPhone															
			XR	XS	XS Max	11	11 Pro	11 Pro Max	SE (2nd generation)	12 mini	12	12 Pro	12 Pro Max	13 mini	13	13 Pro	13 Pro Max	SE (3rd generation)
18.0	No Tool Available																No	
18.0.1	No Tool Available																No	
18.1	No Tool Available																No	
18.1.1	No Tool Available																No	
18.2	No Tool Available																No	
18.2.1	No Tool Available																No	
18.3	No Tool Available																No	
18.3.1	No Tool Available																No	
18.3.2	No Tool Available																No	
18.4	No Tool Available																No	
18.4.1	No Tool Available																No	
18.5	No Tool Available																No	
18.6	No Tool Available																No	

https://theapplewiki.com/wiki/Jailbreak/18.x

© 2024 by Bai7 GmbH | All rights reserved

Package Manager

- A Package Manager is essential on a jailbroken device and Cydia or Silo are one of the most popular choices. One can think of it as the “App Store’s” for the jailbreak community.
- It ensures ease of managing, installing and discovering tweaks, themes, cli tools and apps that can be installed on a jailbroken device.

© 2024 by Bai7 GmbH | All rights reserved

How to get IPAs?

- Use an old jailbroken device with iOS15/16/17, install the app via the App Store and dump the app via frida-ios-dump.
- Use sites like https://decrypt.day/
- For a penetration test: Ask the developer to provide an IPA file to you (not a Test flight link).

Decrypt IPA Store

github.com/AloneMonkey/frida-ios-dump

README MIT license

frida-ios-dump

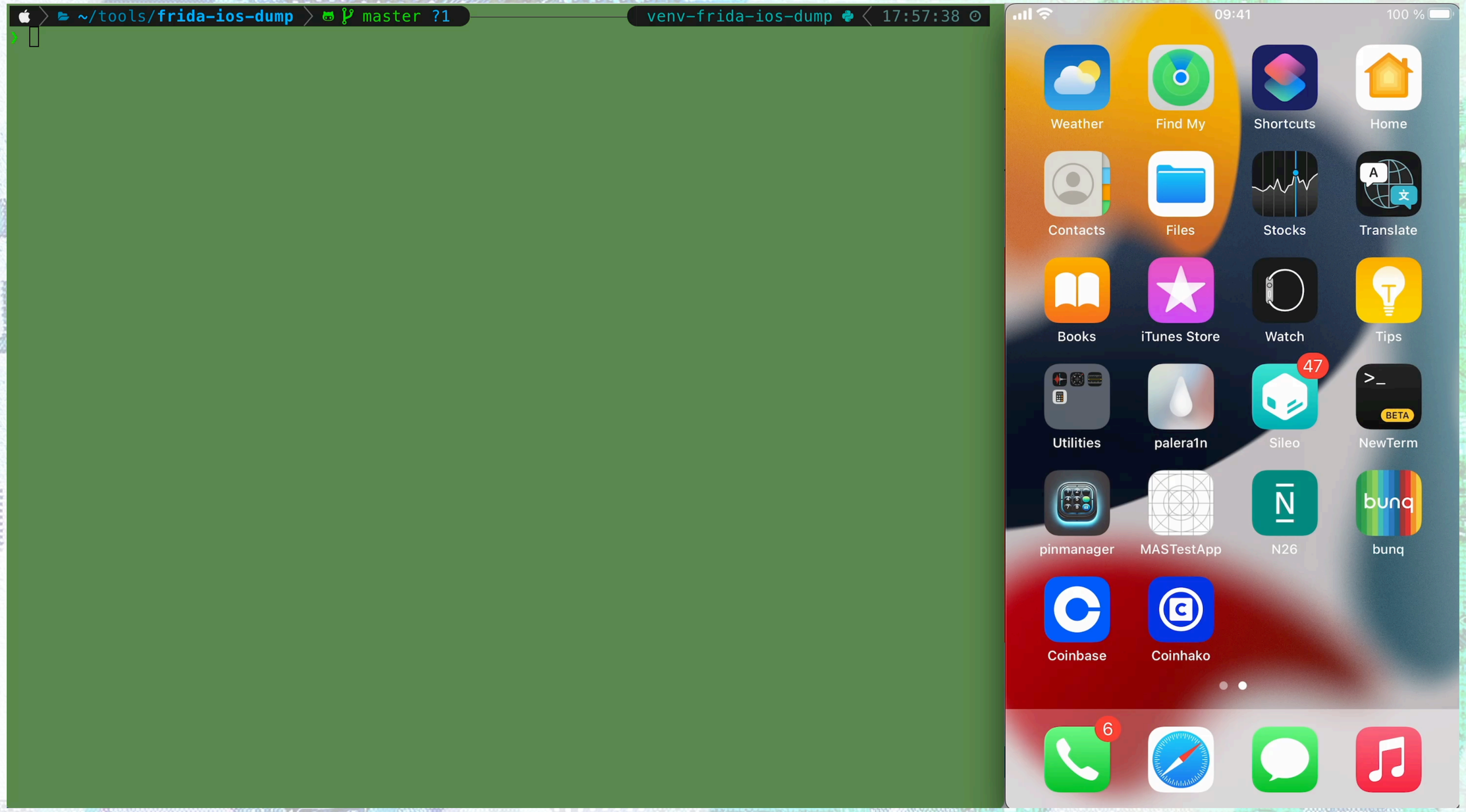
Pull a decrypted IPA from a jailbroken device

Usage

1. Install frida on device
2. `sudo pip install -r requirements.txt --upgrade`
3. Run usbmuxd/iproxy SSH forwarding over USB (Default 2222 -> 22). e.g. `iproxy 2222 22`
4. Run ./dump.py `Display name` or `Bundle identifier`

© 2024 by Bai7 GmbH | All rights reserved

iOS Platform



© 2024 by Bai7 GmbH | All rights reserved

iOS Platform

Apps can be dumped while they run on jailbroken device to get the unencrypted IPA.

Can be done with Frida-iOS-Dump, but jailbroken device is needed!

```
~/tools/frida-ios-dump  master ?1                                  40s  venv-frida-ios-dump
❯ ls -alh
total 165448
drwxr-xr-x@ 13 sushi2k  staff   416B May 25 17:55 .
drwxr-xr-x@ 16 sushi2k  staff   512B May 24 20:49 ..
-rw-r--r--@  1 sushi2k  staff   6.0K Feb 26 04:57 .DS_Store
drwxr-xr-x@ 12 sushi2k  staff   384B May 25 17:52 .git
-rw-r--r--@  1 sushi2k  staff    30B Oct  5  2024 .gitignore
-rw-r--r--@  1 sushi2k  staff    69M May 25 17:55 Coinbase.ipa
-rw-r--r--@  1 sushi2k  staff    11K Oct  5  2024 dump.js
-rwxr-xr-x@  1 sushi2k  staff    11K Oct  5  2024 dump.py
-rw-r--r--@  1 sushi2k  staff   1.0K Oct  5  2024 LICENSE
-rwxr-xr-x@  1 sushi2k  staff   2.0K Oct  5  2024 process.sh
-rw-r--r--@  1 sushi2k  staff   3.2K Oct  5  2024 README.md
-rw-r--r--@  1 sushi2k  staff   157B Oct  5  2024 requirements.txt
drwxr-xr-x@  6 sushi2k  staff   192B Oct  5  2024 venv-frida-ios-dump
```

© 2024 by Bai7 GmbH | All rights reserved

iOS Security Architecture and Jailbreaks

Benefits of using a jailbroken device:

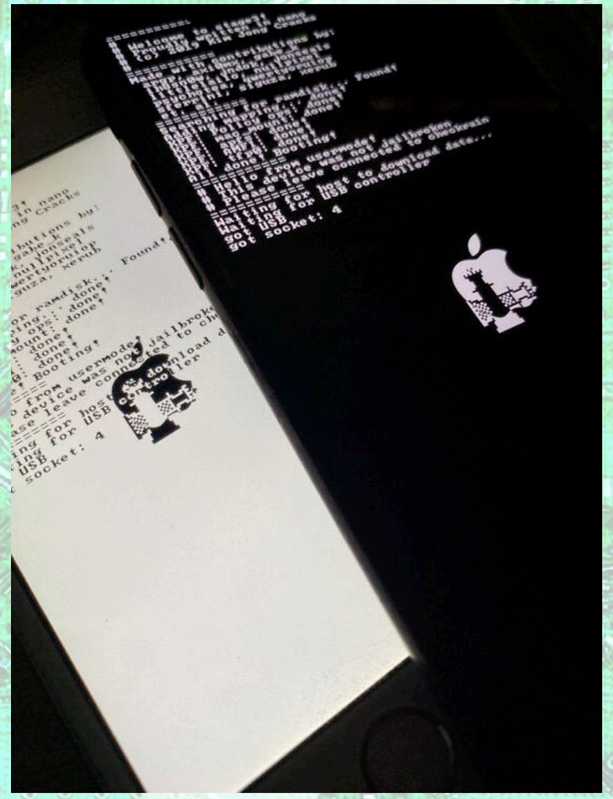
- Gain full root access to the device including file system
- Possibility of executing applications that haven't been signed by Apple (which includes many security testing tools)
- Unrestricted debugging and dynamic analysis

© 2024 by Bai7 GmbH | All rights reserved

iOS Security Architecture and Jailbreaks

Summary:

- The iOS platform (software + hardware) is very homogenous including the App Store (compared to Android)
- The Secure Enclave, App Signing and Sandboxing are some of the main security features, which are all described in the "Apple Platform Security"
- By jailbreaking an iOS device you can get full root access and bypassing the majority of the security features.
- Maintaining a jailbroken device is becoming more and more problematic, due to security restrictions in iOS and the availability of public jailbreaks.
- Developers should share the IPA for full transparency and efficient use of the time during a penetration test.

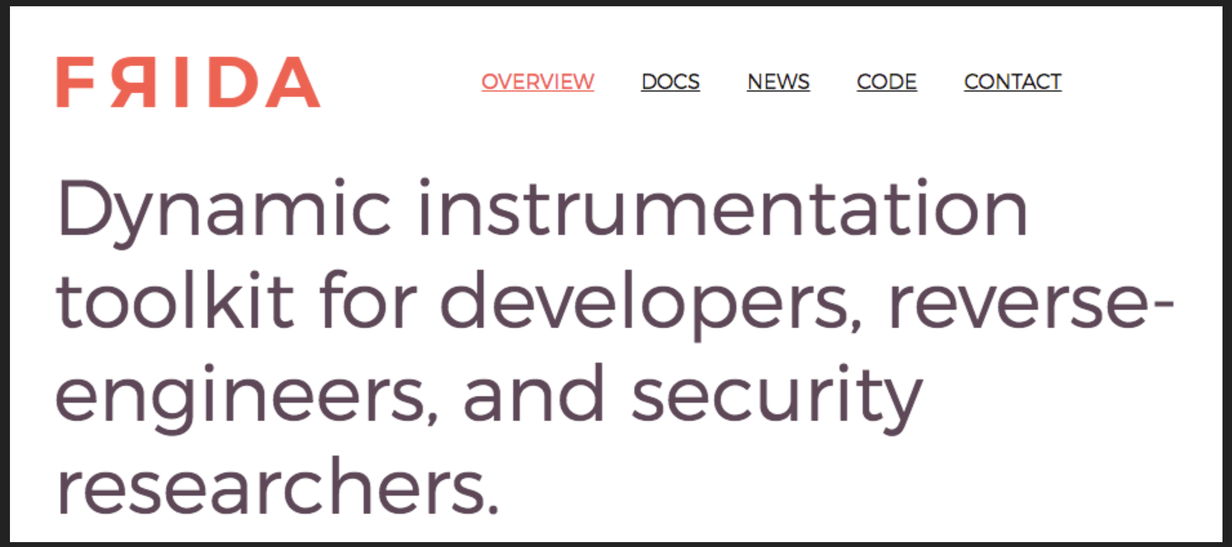


© 2024 by Bai7 GmbH | All rights reserved

MASTG HANDS-ON - IOS EDITION

FRIDA 101

FRIDA



Dynamic instrumentation is a technical process of analysing and modifying the behaviour of a binary at runtime through the injection of instrumentation code.

Created by Ole Andrew V. Ravås (@oleavr)

https://www.frida.re

https://github.com/frida/frida/

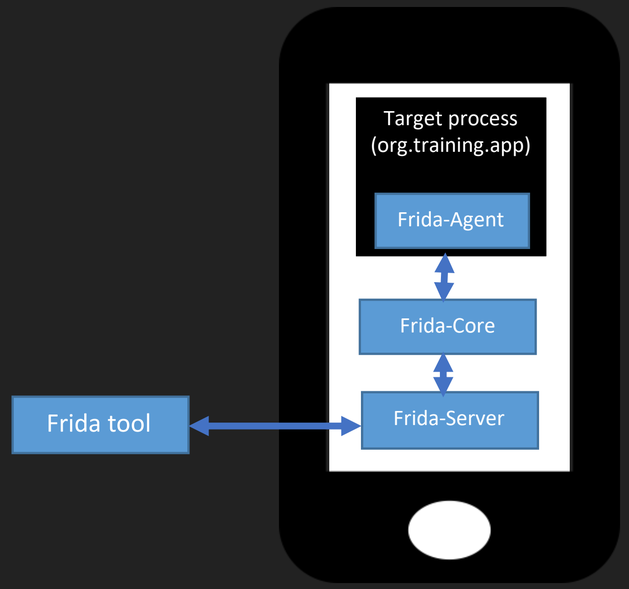
WHAT IS FRIDA EXACTLY?

- Frida is a dynamic instrumentation toolkit and it's free and open source.
- Frida injects a complete JavaScript runtime (~~Duktape or Google V8~~, since Frida 14 it is QuickJS) into the target process, along with a powerful API .
- The APIs allows you to modify and analyse a process or binary during runtime by injecting JavaScript code directly into the process memory.
- Frida can be used via CLI and it can be scripted.
- With Frida you can instrument processes on Windows, Linux, macOS and also Android and iOS apps.

FRIDA – MODES OF OPERATION

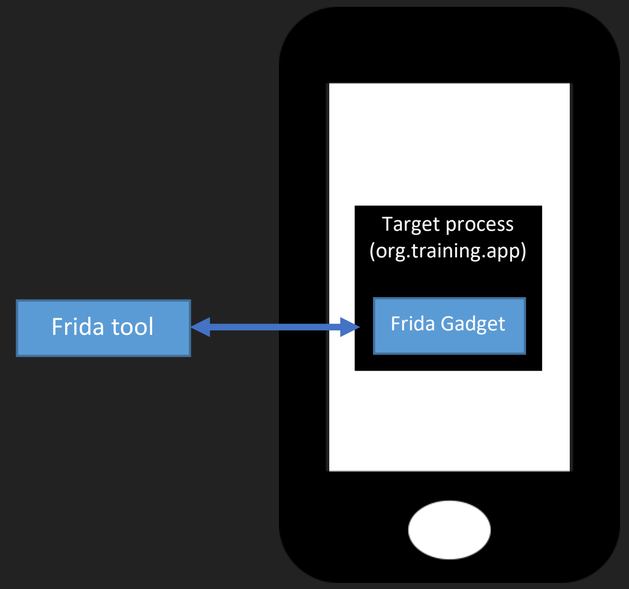
Injected into a process by running the Frida server on the device

- Only possible on jailbroken devices
- Frida handles the injection



Embedded as shared library (frida-gadget.so) into the mobile app

- Working on non-jailbroken devices
- Repackaging and resigning required



FRIDA SERVER VS GADGET

- Frida Server can only be installed on a jailbroken device.
- Frida Server will allow you to connect to any app and you can debug all processes compared to Frida-Gadget.

- When using Frida Gadget in an app you can bypass Jailbreak detection, as you can install the app on a non-jailbroken device and can still use Frida.
- Frida Gadget will allow you to instrument and debug only the app the Gadget is embedded in.
- When using Frida Gadget you need to repackage the app first, before you can test.
- Some customers might only accept the risk if you can prove that you could test on a non-jailbroken device

FRIDA – MODES OF OPERATION

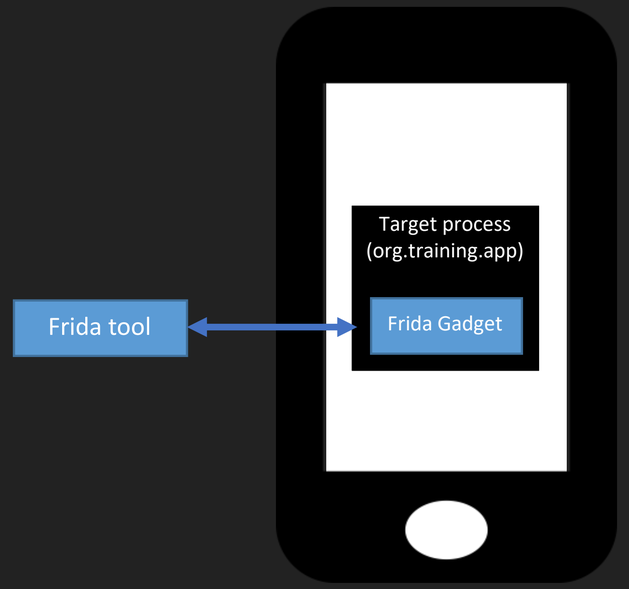
This is what we will be doing with the Corellium instance!

Injected into a process by running the Frida server on the device

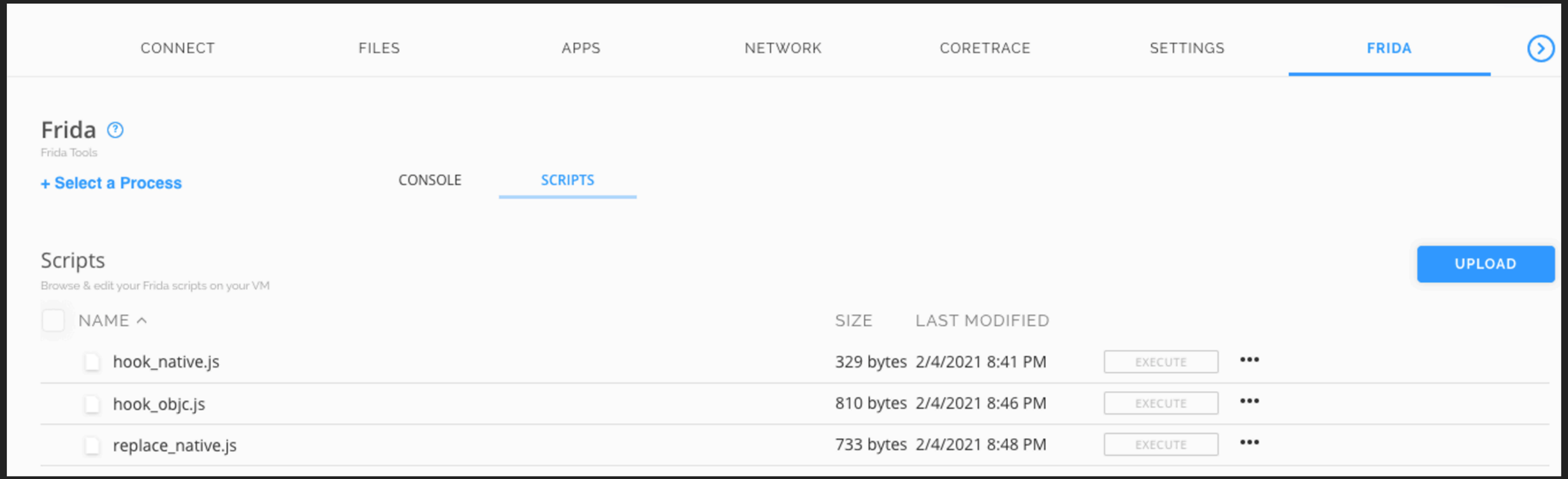
- Only possible on jailbroken devices
- Frida handles the injection

Embedded as shared library (frida-gadget.so) into the mobile app

- Working on non-jailbroken devices
- Repackaging and resigning required



FRIDA IS ALREADY INTEGRATED IN CORELLIUM UI AND THE IOS INSTANCE!



INSTALLING FRIDA SERVER ON A PHYSICAL IOS DEVICE

Install the Frida server via Cydia

https://frida.re/docs/ios/#with-jailbreak

With Jailbreak

This is the most powerful setup, as it lets you instrument system services and apps with very little effort.

In this tutorial we will show you how to do function tracing on your iOS device.

Setting up your iOS device

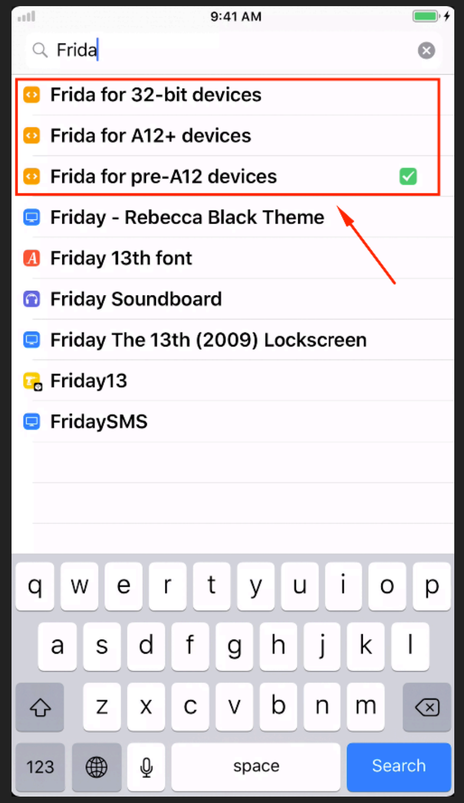
Start `Cydia` and add Frida's repository by going to `Manage` -> `Sources` -> `Edit` -> `Add` and enter `https://build.frida.re`. You should now be able to find and install the `Frida` package which lets Frida inject JavaScript into apps running on your iOS device. This happens over USB, so you will need to have your USB cable handy, though there's no need to plug it in just yet.

When installing Frida via Cydia, you have 3 options:

Frida for 32-bit devices	All devices < iPhone 5s
Frida for A12+ devices	All devices >= iPhone XR
Frida for pre-A12 devices	All devices <= iPhone X

A12 is referring to the System on a Chip (Soc) in an iOS device. Detailed device listing:

https://en.wikipedia.org/wiki/List_of_iOS_and_iPadOS_devices



FRIDA BASICS

Hook

- Similar to setting up a Software breakpoint on a function to inspect, instrument or manipulate the code.

Manipulate

- Changing the flow of the function or completely overwriting it

Inspect

- Inspecting arguments being passed into the functions.

Instrument

- Injecting additional instructions that helps to debug and reverse engineer the application.

FRIDA SCRIPTS

Frida CodeShare Log In

Project: ios-app-info

Try this code out now by running `$ frida --codeshare dki/ios-app-info -f YOUR_BINARY`

```
/*
 *  Convenience functions to access app info.
 *  Dump key app paths and metadata:
 *      appInfo()
 *
 *  Print contents of Info.plist:
 *      infoDictionary()
 *
 *  Query Info.plist by key:
 *      infoLookup("NSAppTransportSecurity")
 *
 */

function dictFromNSDictionary(nsDict) {
    var jsDict = {};
    var keys = nsDict.allKeys();
    var count = keys.count();
    for (var i = 0; i < count; i++) {
        var key = keys.objectAtIndex_(i);
        var value = nsDict.objectForKey_(key);
        jsDict[key.toString()] = value.toString();
    }

    return jsDict;
```

Frida CodeShare Log In

Unleash the power of Frida.

"If I have seen further, it is by standing on the shoulders of giants."
-Sir Issac Newton

The Frida CodeShare project is comprised of developers from around the world working together with one goal - push Frida to its limits in new and innovative ways.

Frida has amazing potential, but needed a better forum to share ideas, so we've put together CodeShare to help stoke innovation and push Frida to its highest potential!

BROWSE CODE

- Many problems are already solved! So leverage on existing resources such as CodeShare that offer Frida Scripts: https://codeshare.frida.re/
- You can load any script from codeshare with the flag “--codeshare”
- You can load scripts in Frida by using the flag “-l”

OBJC API

- Interface between Frida scripts and Objective-C runtime and essential to iOS dynamic instrumentation.
- Java API is the equivalent in Android.
- Some common APIs are
  - $ ObjC.Classes
  - $ ObjC.Object

INTERCEPTOR

- Main API we will be using to instrument a process
- Injects a trampoline and redirect code flow to our Frida script and return to original code (In-line hooking).
- Some Common APIs are:
  - Interceptor.attach(target, callbacks[, data])
    - Intercept calls to function at target. ("onEnter" and "onLeave" callback functions)
    - See also https://frida.re/docs/javascript-api/#interceptor-attach
  - Interceptor.replace(target, replacement[, data])
    - Replace function at target with implementation at replacement.
    - See also https://frida.re/docs/javascript-api/#interceptor-replace

LAB TIME!

Open the file Lab-iOS.pdf

Section: Lab - Frida 101

Please type the commands and do not copy & paste!

1.You will remember them easier

2.Some PDF readers do not copy properly and miss whitespaces etc.

iOS Training

iOS testing without a jailbroken device

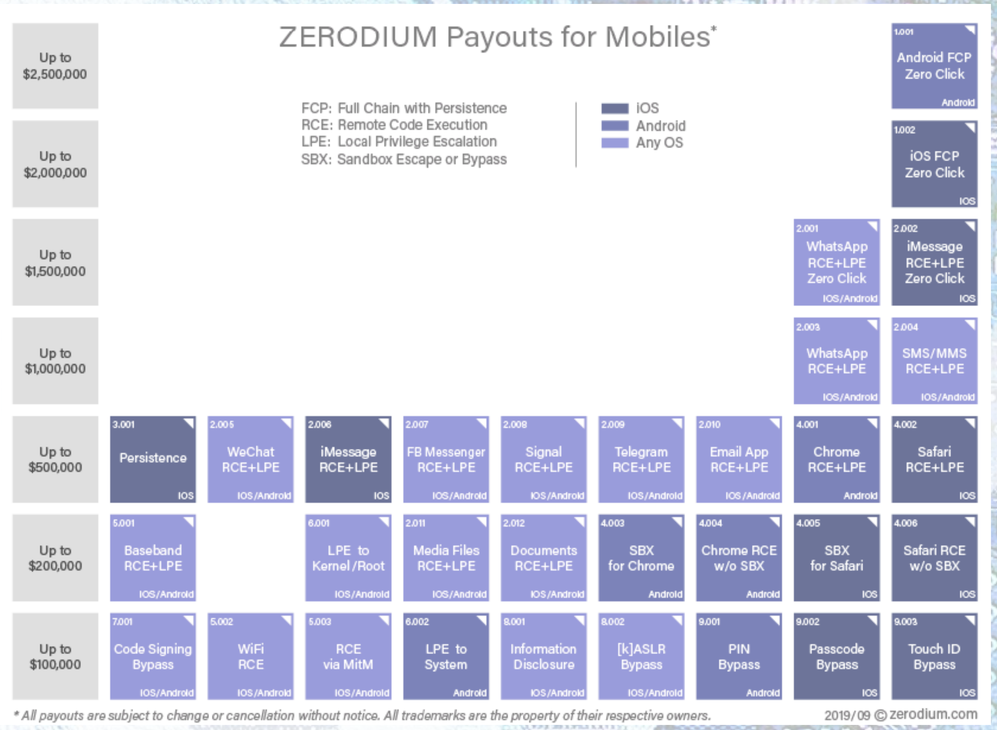
Jailbreaks

Zero-click kernel code execution with persistence and kernel PAC bypass — **$100,000 – $1,000,000**

Examples

$1,000,000: Zero-click remote chain with full kernel execution and persistence, including kernel PAC bypass, on latest shipping hardware. As an example, you demonstrated a zero-click remote chain with full kernel execution and a PAC/PPL bypass with persistence on the latest iOS device.

Apple Security Bug Bounty



Zerodium

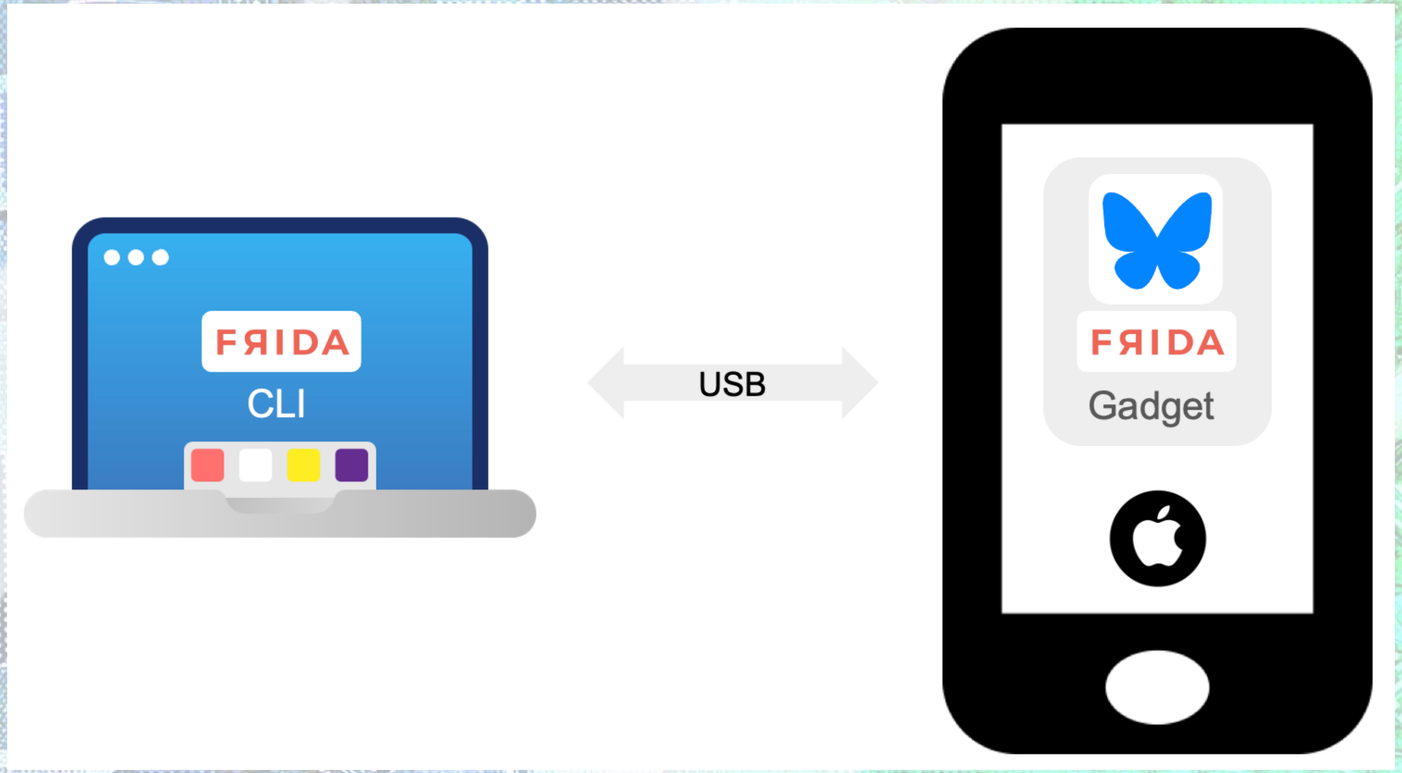
iPadOS

iPadOS	Jailbreak Tool	Tool Version	iPad												
			(6th gen.)	(7th gen.)	(8th gen.)	(9th gen.)	(10th gen.)	mini (5th gen.)	mini (6th gen.)	Air (3rd gen.)	Air (4th gen.)	Air (5th gen.)	Air 11-inch (M2)	Air 13-inch (M2)	Pro (12.9-inch) (2nd gen.)	Pro (10.5-inch)
17.0	palera1n	2.1 beta 2	Semi-Tethered	No									—		Semi-Tethered	
17.0.1	palera1n	2.1 beta 2	Semi-Tethered	No									—		Semi-Tethered	
17.0.2	palera1n	2.1 beta 2	Semi-Tethered	No									—		Semi-Tethered	
17.0.3	palera1n	2.1 beta 2	Semi-Tethered	No									—		Semi-Tethered	

Jailbreaks iOS 18 / Jailbreaks iOS 17

© 2024 by Bai7 GmbH | All rights reserved

Testing without Jailbreak



MASTG-TOOL-0031: Frida

© 2024 by Bai7 GmbH | All rights reserved

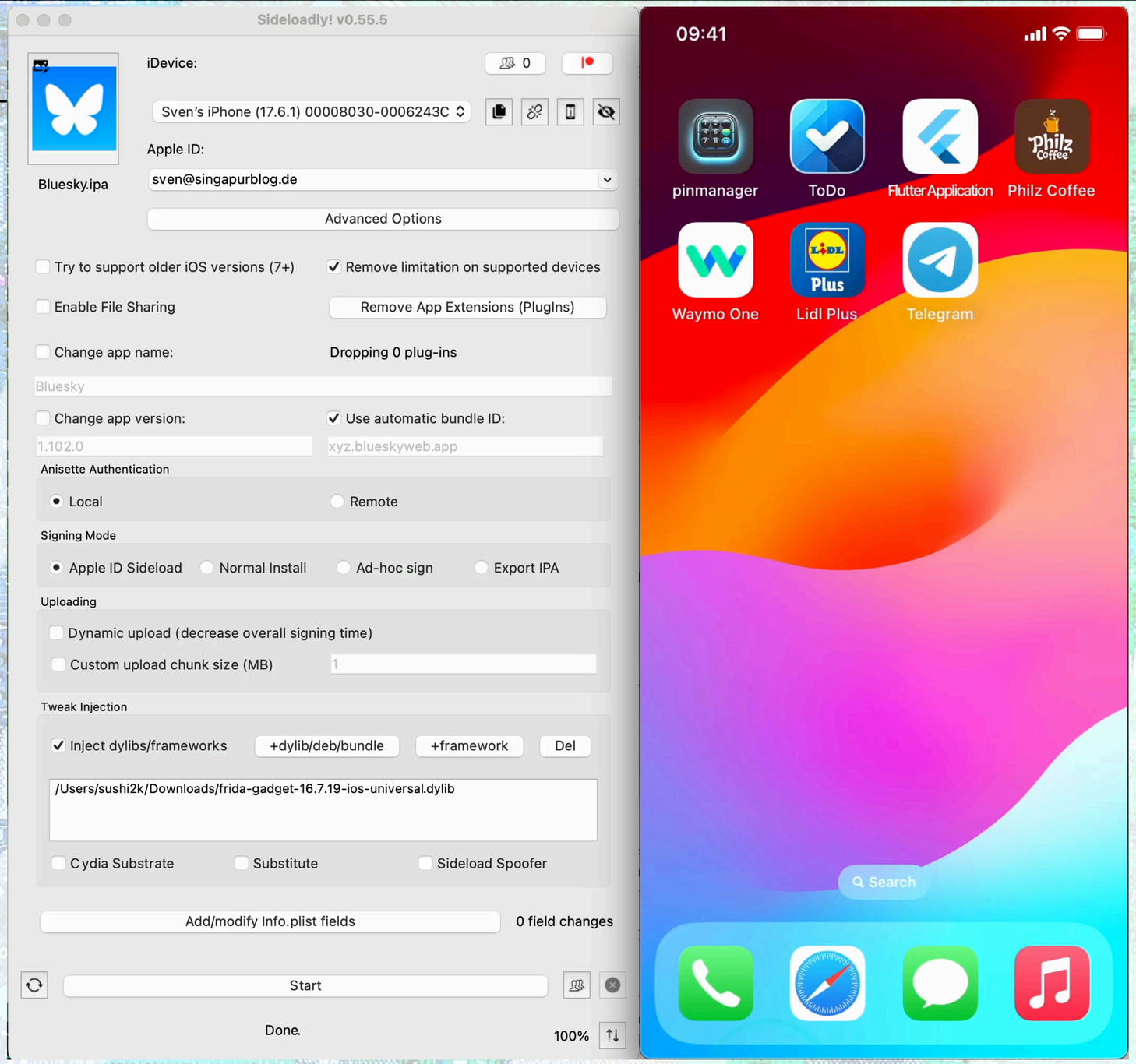
Testing without Jailbreak

An iOS App (IPA) can be installed in various ways on a non-jailbroken device.

In this demo we use Sideloadly.

Sideloadly will re-package the app with the Frida Gadget and install it on the non-jailbroken device.

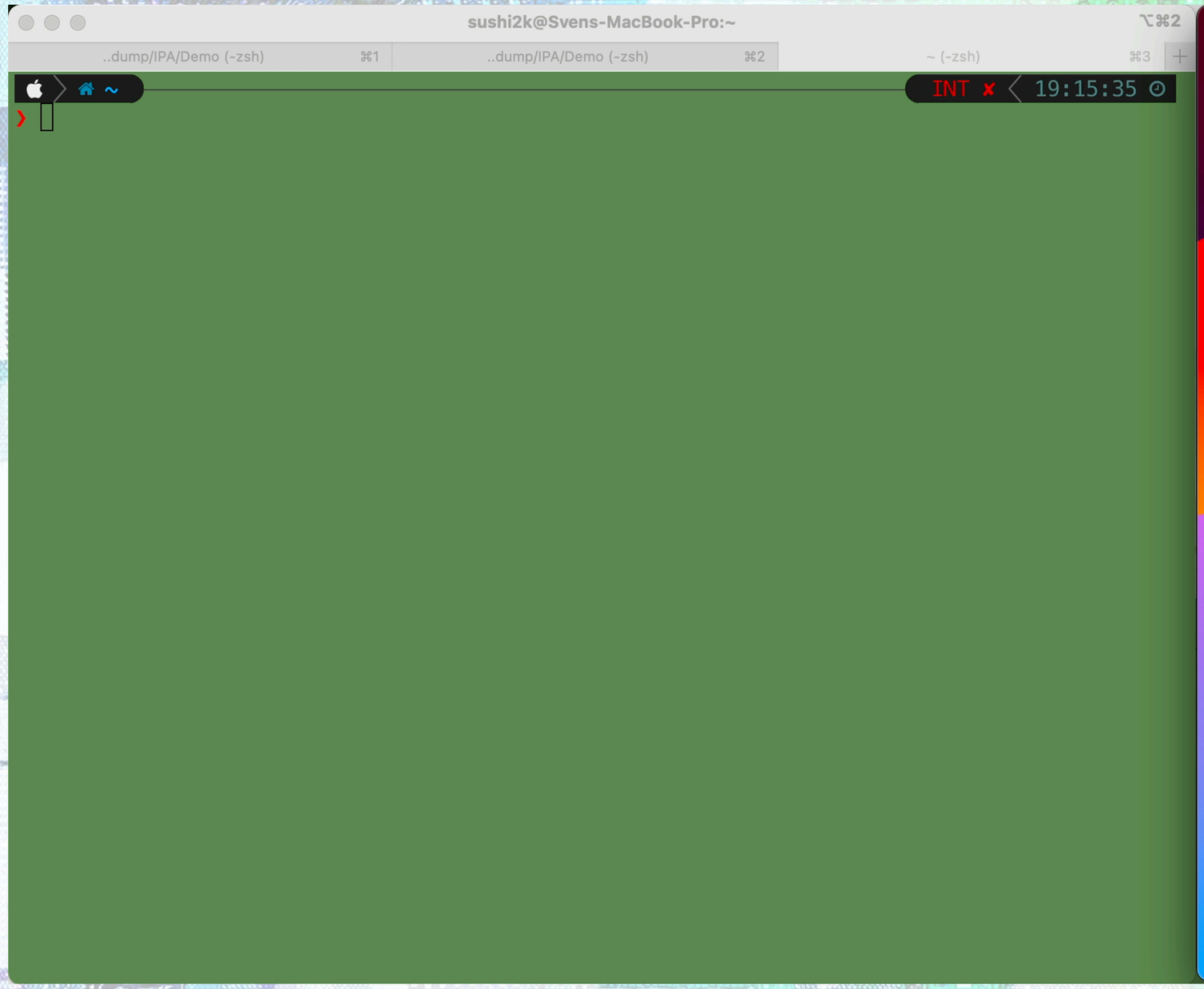
MASTG-TECH-0056: Installing Apps

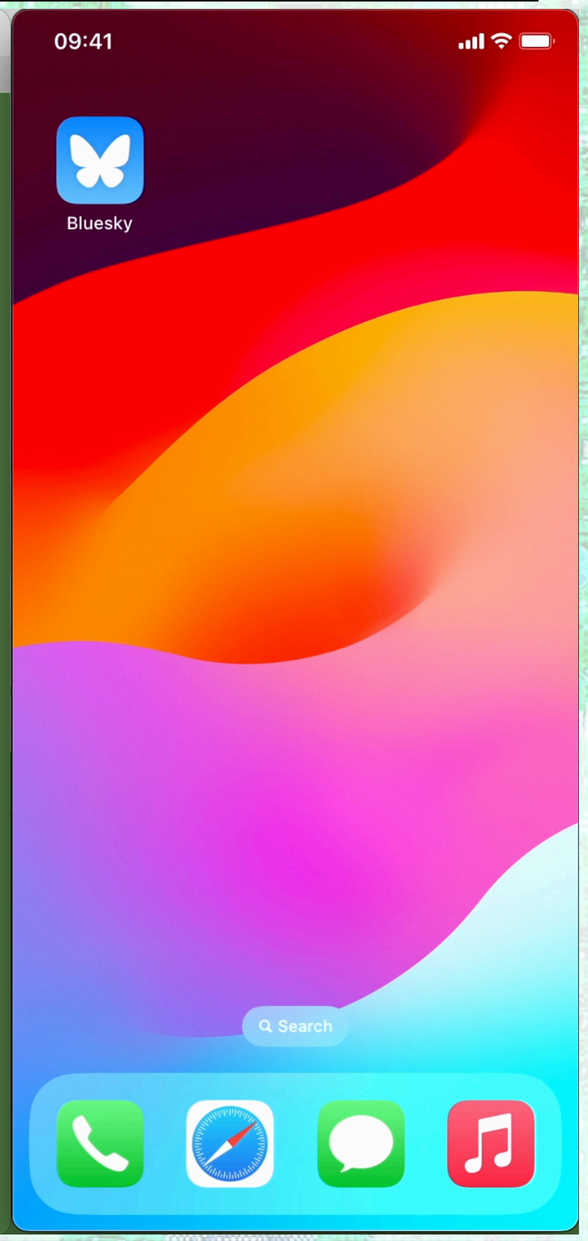


© 2024 by Bai7 GmbH | All rights reserved

Testing without Jailbreak

MASTG-TECH-0055: Launching a Repackaged App in Debug Mode



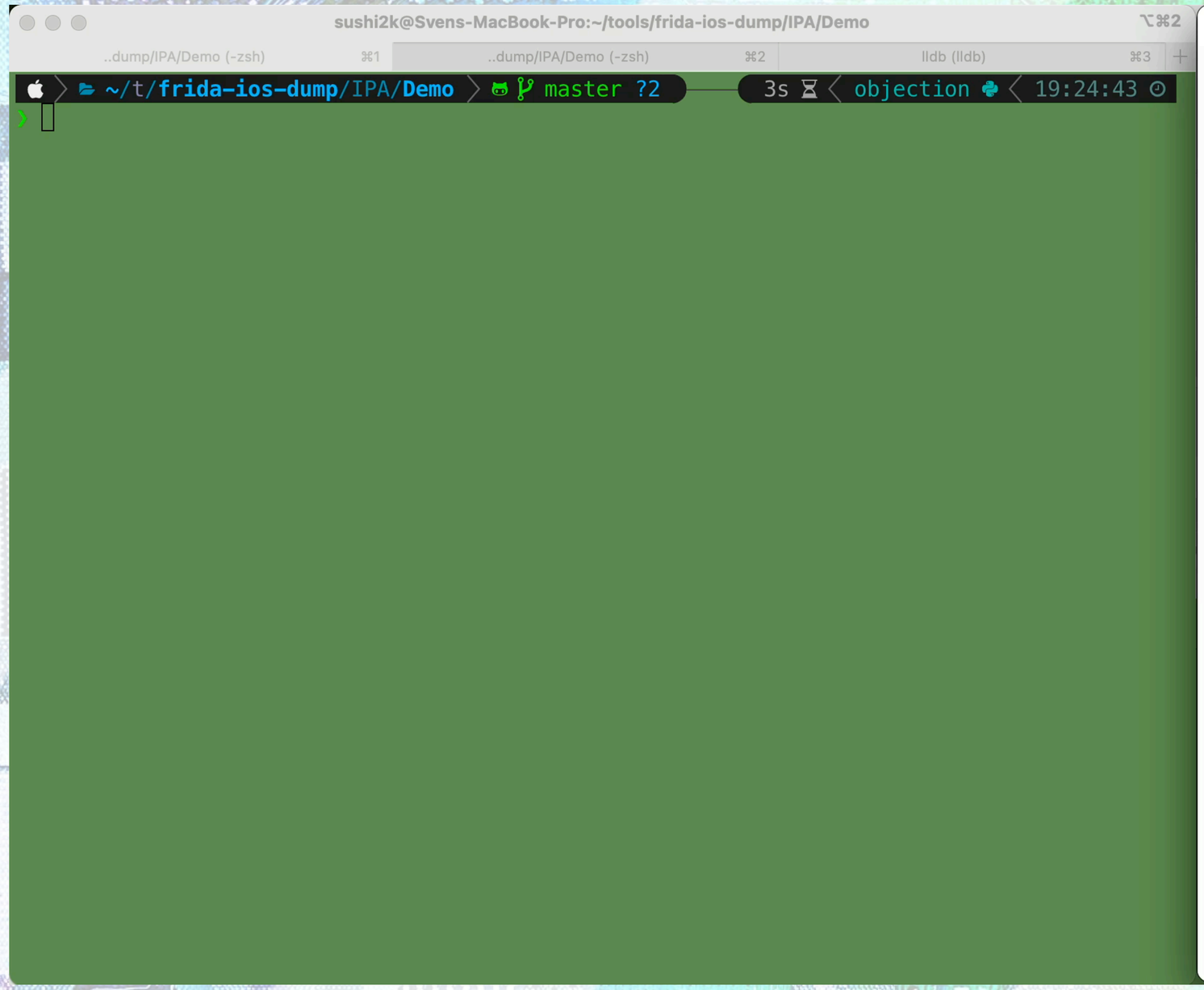


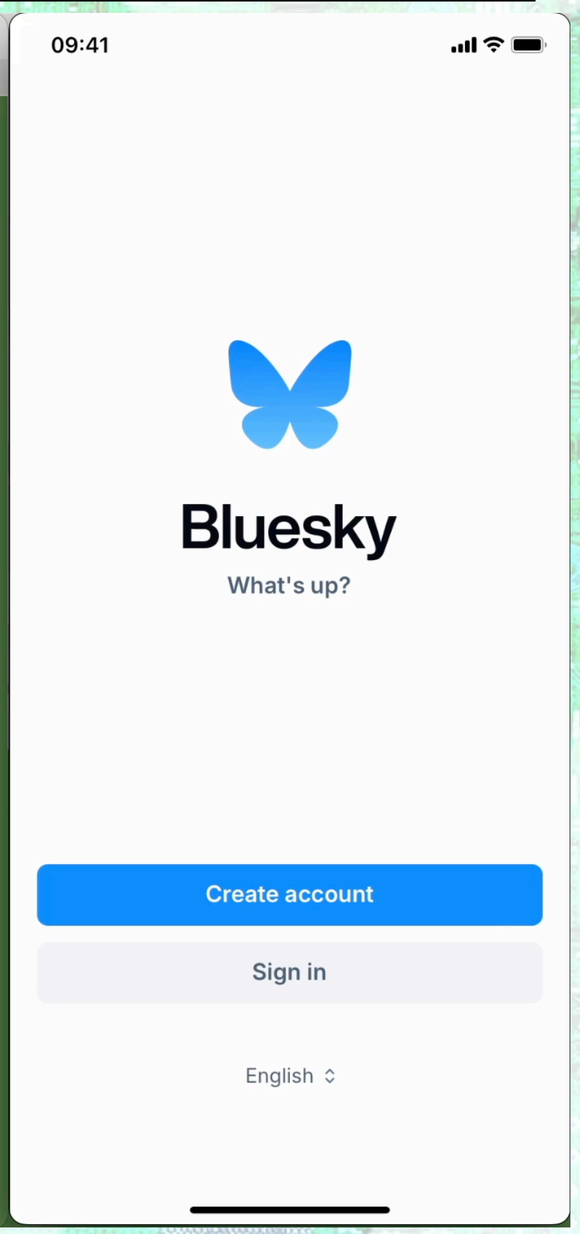
© 2024 by Bai7 GmbH | All rights reserved

Testing without Jailbreak

MASTG-TOOL-0038: Objection

Through the Frida Gadget we can execute a full penetration test against the app, including using tools like objection!



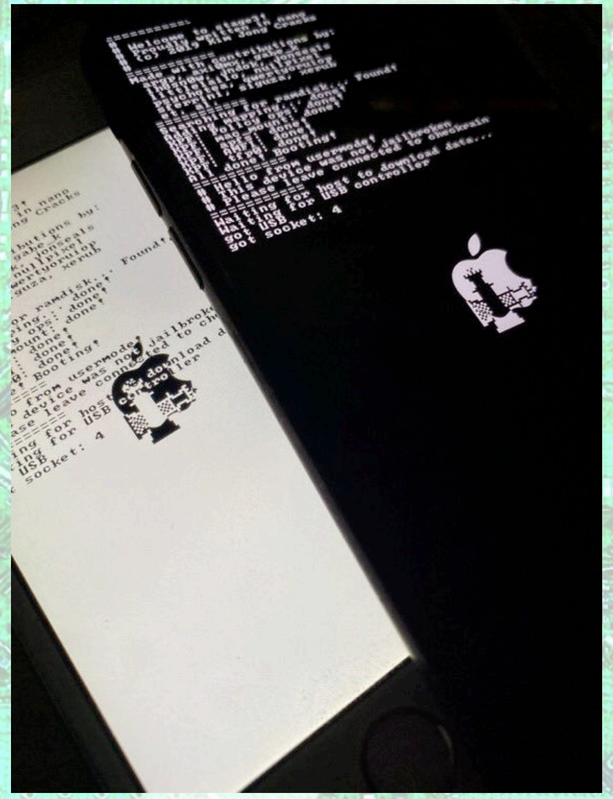


© 2024 by Bai7 GmbH | All rights reserved

iOS Security Architecture and Jailbreaks

Summary:

- Testing on a non-jailbroken (or jailed) device allows pentesters also to use Frida and you can execute a full penetration test against the app
- It might even make testing easier (e.g. no need to bypass jailbreak detection)
- It can make more impact to the client to proof that a vulnerability is a risk!
- You either need a debuggable version of the app (to inject Frida Gadget via lockdown) or you need to repackage the app



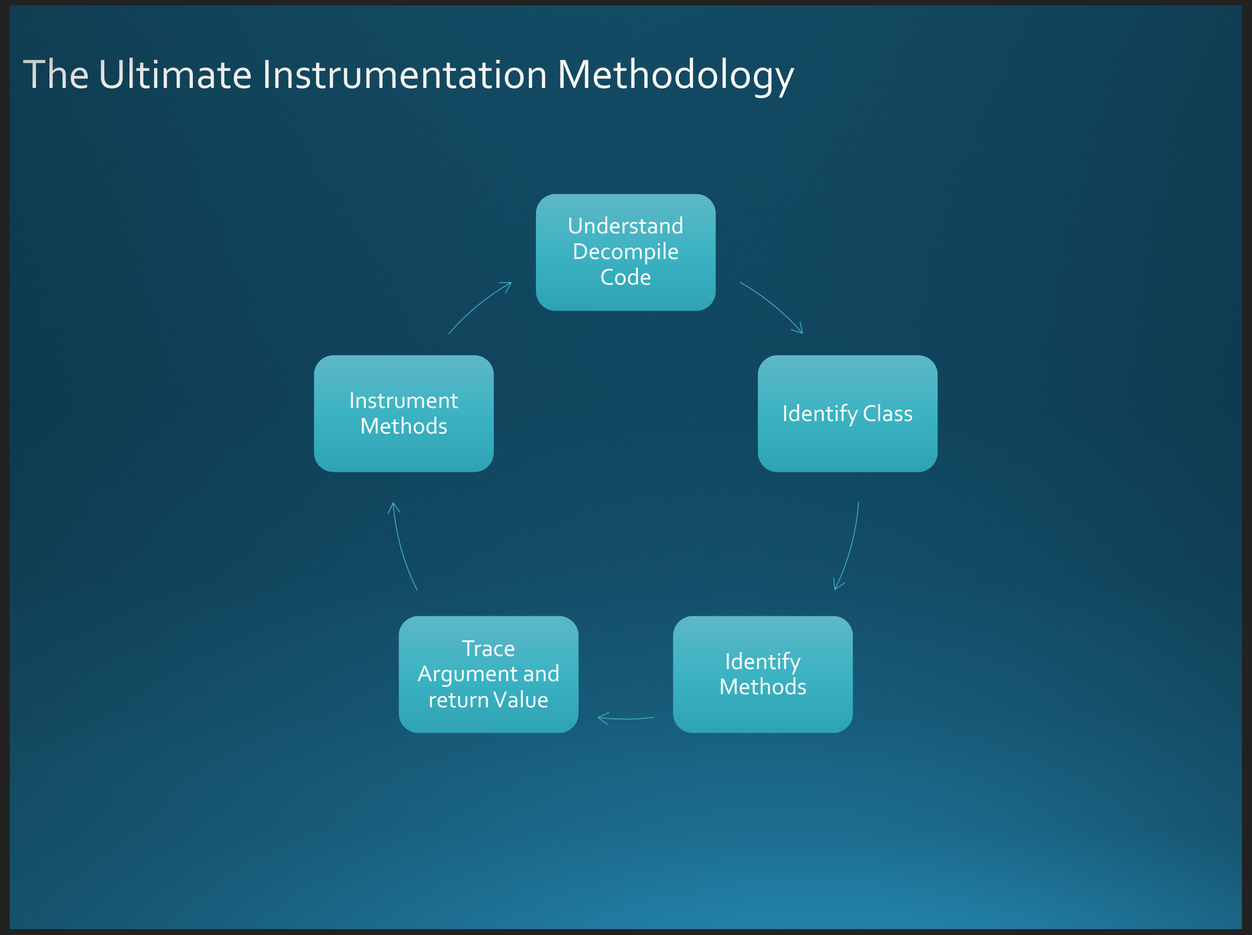
© 2024 by Bai7 GmbH | All rights reserved

MASTG HANDS-ON - IOS EDITION

FRIDA FLOW

THE ULTIMATE INSTRUMENTATION METHODOLOGY

- Understand disassembled code
- Enumerate and Identify Class
- Enumerate and Identify Method
- Trace argument and return value
- Instrument methods
- Repeat

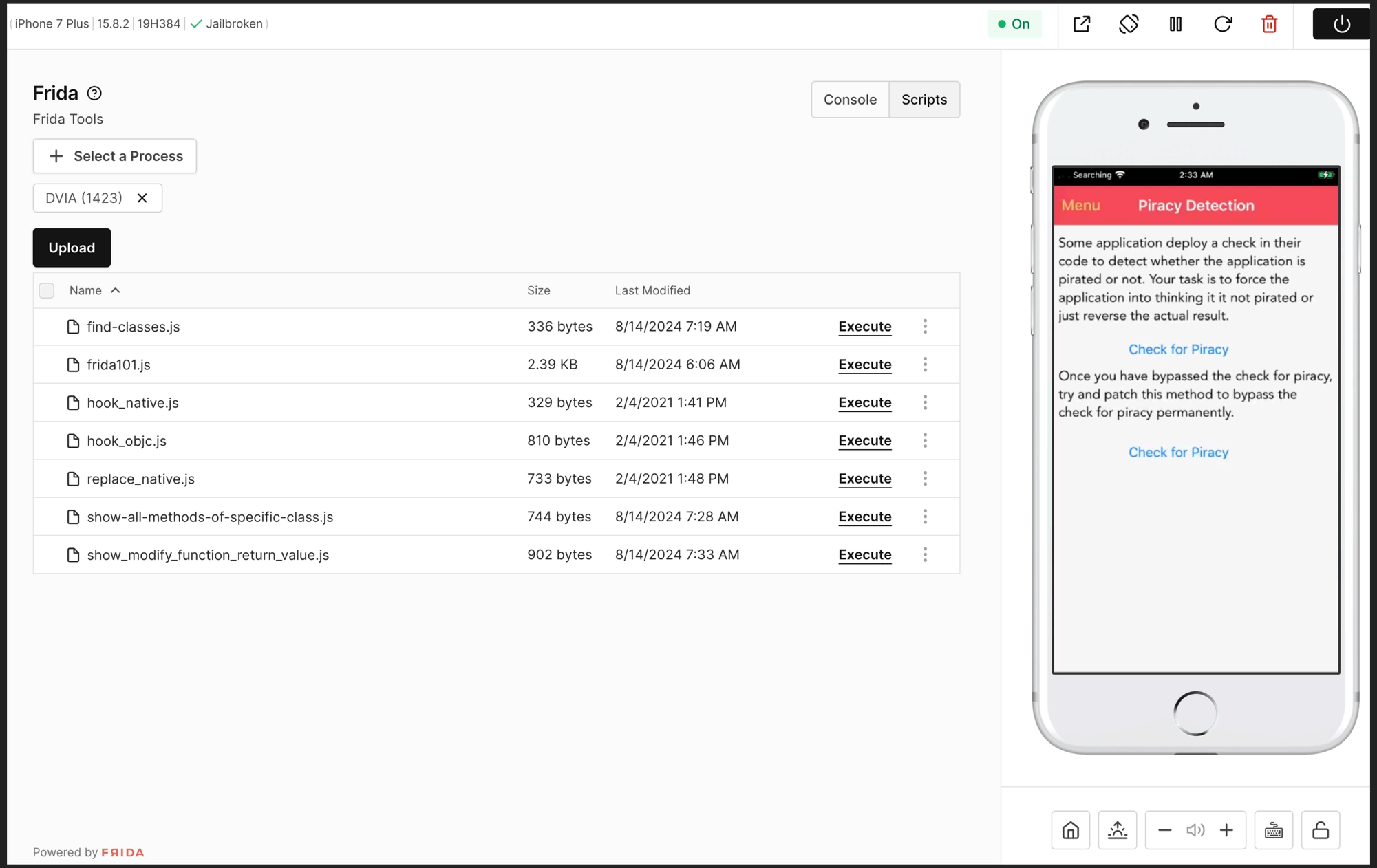


UNDERSTAND DISASSEMBLED CODE

- Not mandatory but aid instrumentation process
- Hopper, IDA, Ghidra and R2
- Essential in black box environment

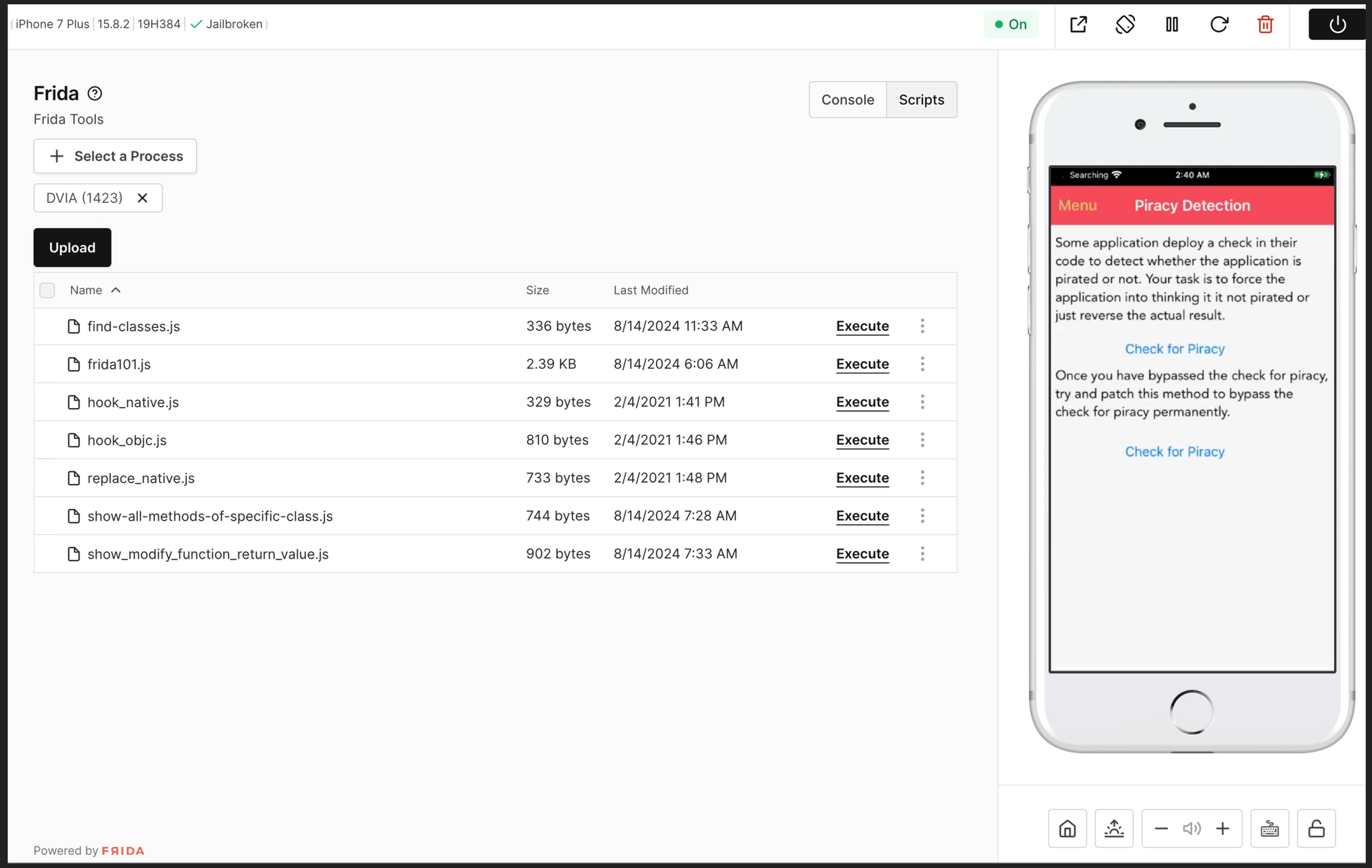
ENUMERATE AND IDENTIFY CLASS

- Class dump
- Custom frida script: find-classes.js
- Frida API - [ObjC.enumerateLoadedClasses]



ENUMERATE AND IDENTIFY METHOD

- Class-dump
- Custom frida script: show-all-methods-of-specific-class.js
- Frida API - ObjC
  - $ownMethod
  - argumentType
  - ReturnType



TRACE ARGUMENTS AND RETURN VALUE

- Frida Interceptor API
- Frida ObjC API

```
Interceptor.attach(ObjC.classes.$CLASS['$METHOD'].implementation,
{

  onEnter: function (args) {
    var arg = ObjC.Object(args[2]).toString();
       send('First argument is ' + arg);
  },


  onLeave: function (retval) {
        var ret = ObjC.Object(retval).toString()
        send('return value is ' + ret);

  }
});
```

LAB TIME!

Open the file Lab-iOS.pdf

Section: Lab – Bypass "Piracy Detection"

Please type the commands and do not copy & paste!

1.You will remember them easier

2.Some PDF readers do not copy properly and miss whitespaces etc.

Please tick (✅) my message in Slack if you are done with the lab

FRIDA CAN HELP YOU IN MANY WAYS!

- Intercept network traffic, crypto calls, file access etc.
- Pentest and fuzz apps
- Overcome protection mechanisms
- Analyse unknown apps (Malware)
- Solve crackmes/CTFs
- Build your own analysis scripts, frameworks and tools

FRIDA RESSOURCES

FRIDA

- Release Notes:
  - https://frida.re/news/releases/
- Official Documentation:
  - https://www.frida.re/docs/home/
- Detailed slide deck about Frida and how to use in mobile app testing:
  - https://drive.google.com/file/d/1JccmMLi6YTnyRrp_rk6vzKrUX3oXK_Yw/view
- A curated list of awesome projects, libraries, and tools powered by Frida:
  - https://github.com/dweinstein/awesome-frida

LEARN BY EXAMPLES

- Useful Frida scripts for iOS:
  - https://codeshare.frida.re/@ay-kay/ios-dataprotection/
  - https://codeshare.frida.re/@dki/ios-app-info/
  - https://codeshare.frida.re/@oleavr/ios-list-apps/
  - https://codeshare.frida.re/@dki/find-ios-app-by-display-name/
  - https://codeshare.frida.re/@andydavies/ios-tls-keylogger/
  - https://codeshare.frida.re/@ChiChou/objective-c-class-hierarchy-dumper/
  - https://codeshare.frida.re/@maltek/generate-c-header-for-swift-data-types/
- These scripts are great to get an understanding of Frida and to use it as a starting point if you have a specific use case you want to implement.
  - https://github.com/dweinstein/awesome-frida
  - https://github.com/iddoeldor/frida-snippets
  - https://github.com/interference-security/frida-scripts/tree/master/iOS

SUMMARY

- Many Frida Scripts are available, don’t reinvent the wheel!
- ObjC API - API that maps Frida scripts with Objective-C runtime.
- Interceptor - Main API for instrumenting processes.

MASTG HANDS-ON - IOS EDITION

TOOLS POWERED BY FRIDA

OBJECTION

objection - Runtime Mobile Exploration

`objection` is a runtime mobile exploration toolkit, powered by Frida, built to help you assess the security posture of your mobile applications, without needing a jailbreak.

twitter @leonjza | pypi package 1.9.6 | build passing | Black Hat Arsenal Europe 2017 | Black Hat Arsenal USA 2019

- Supports both iOS and Android.
- Inspect and interact with container file systems.
- Bypass SSL pinning.
- Dump keychains.
- Perform memory related tasks, such as dumping & patching.
- Explore and manipulate objects on the heap.
- And much, much more...

OBJECTION
RUNTIME
MOBILE
EXPLORATION
GIT.IO/OBJECTION

https://github.com/sensepost/objection

OBJECTION

- Python based
- Can be installed and upgraded by using pip3

installation

Installation is simply a matter of `pip3 install objection`. This will give you the `objection` command. You can update an existing `objection` installation with `pip3 install --upgrade objection`.

- For more detailed update and installation instructions, please refer to the wiki page:

https://github.com/sensepost/objection/wiki/Installation

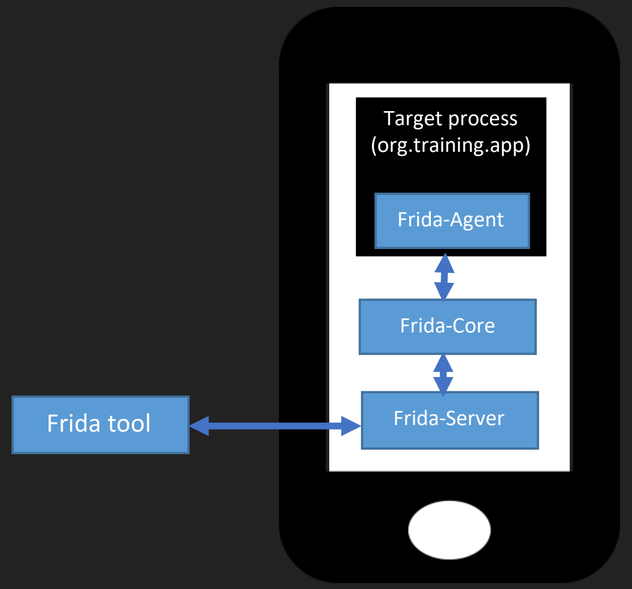
OBJECTION – DOCUMENTATION AND HELP

- Leon Jacobs did a fantastic job not only by implementing the tool, but also for creating a complete documentation!
- Detailed documentation is available in the wiki of the Github repository
- For any problems, raise an issue in Github

FRIDA – MODES OF OPERATION

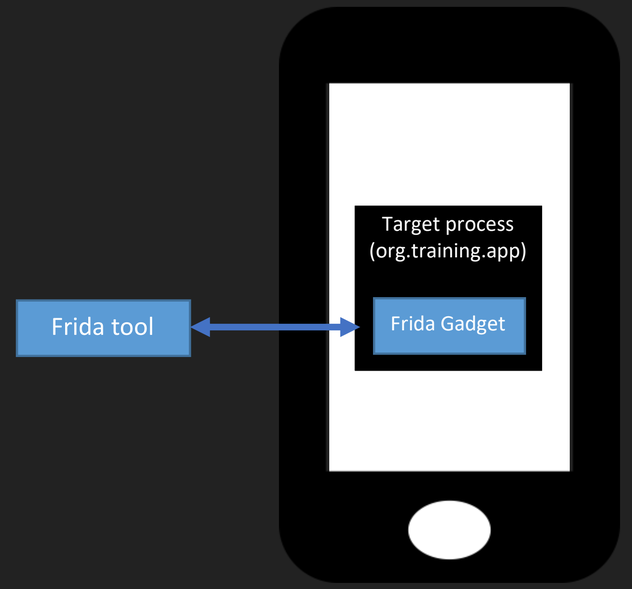
Injected into a process by running the Frida server on the device

- Only possible on jailbroken devices
- Frida handles the injection

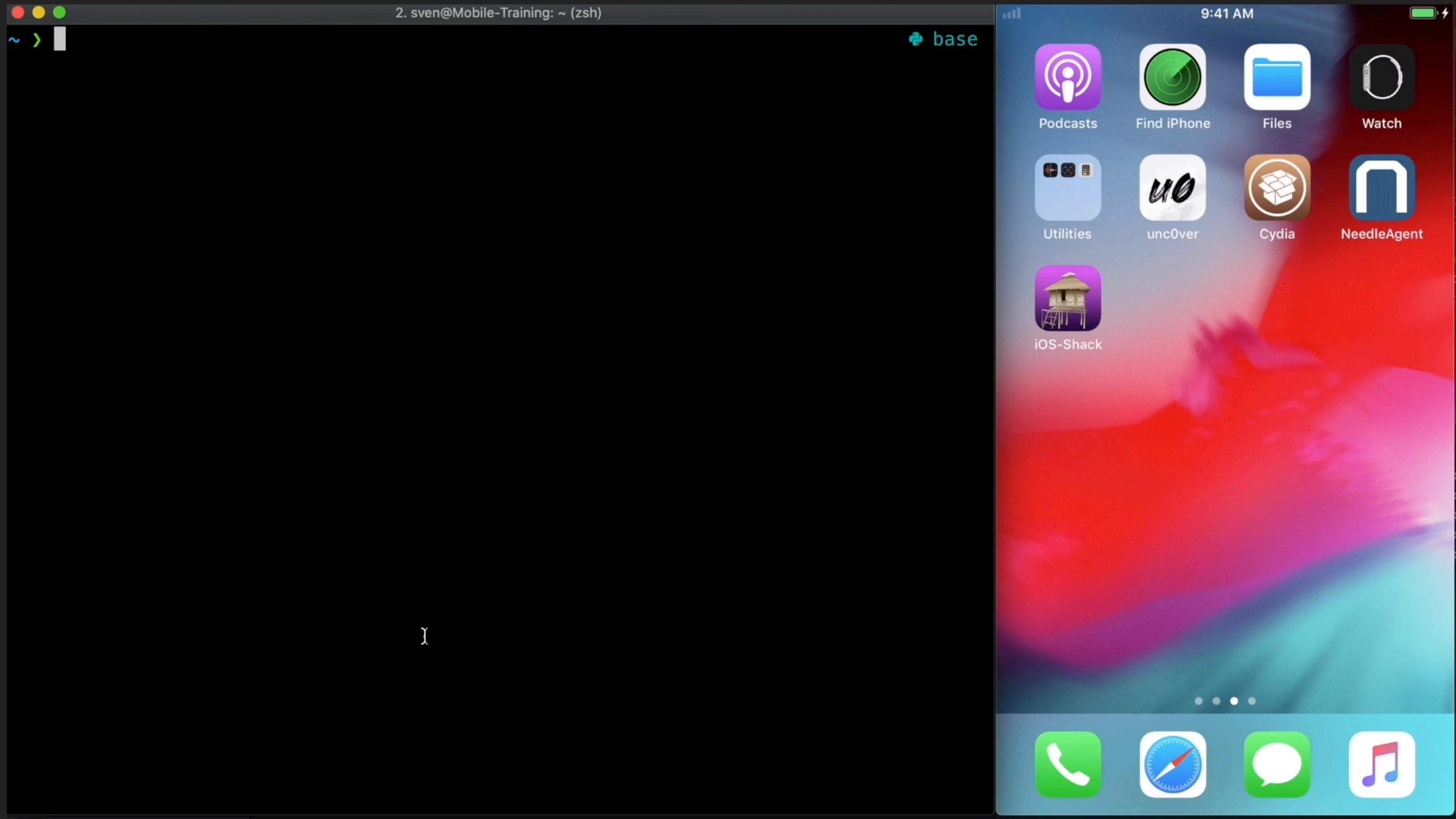


Embedded as shared library (frida-gadget.so) into the mobile app

- Working on non-jailbroken devices
- Repackaging and resigning required



MASTG HANDS-ON - IOS EDITION



TOOLS POWERED BY FRIDA FOR IOS APP ANALYSIS

- Runtime Mobile Security (RMS) - https://github.com/m0bilesecurity/RMS-Runtime-Mobile-Security
  - Python based, can be installed via pip
  - Web UI that allows to manipulate iOS and also Android apps

---

- ~~Passionfruit - https://github.com/chaitin/passionfruit/~~
  - JavaScript based, can be installed via npm
  - Announcement in April 2020 that project is going to be archived: https://github.com/chaitin/passionfruit/issues/74; **It is NOT working with Frida 14.x anymore!**

- Grapefruit - https://github.com/ChiChou/Grapefruit
  - JavaScript based, can be installed via npm
  - Successor of Passionfruit but WIP

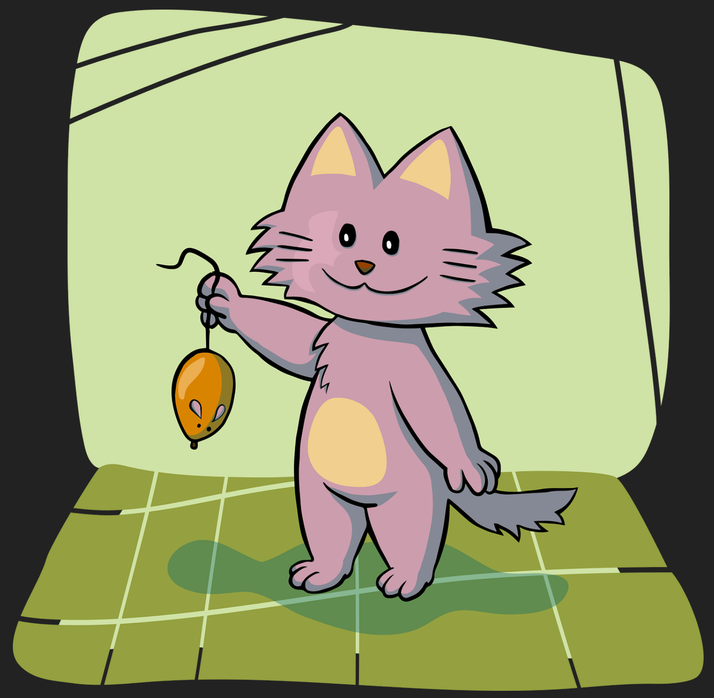
MASTG HANDS-ON - IOS EDITION

SECURITY CONTROLS IN MOBILE APPS

MOBILE APP SECURITY CONTROLS

Controls to make Reverse Engineering of mobile apps **harder**:

- Checking if Reverse Engineering tools are used (Frida)
- Checking if device is jailbroken (multiple checks)
- Obfuscation of source code
- Checking if app was repackaged (Objection/Frida-Gadget)
- Checking if a debugger is used
- etc.



Makes the effort more time consuming and can be used as part of a defence in depth strategy. Reverse Engineering is still possible and will always be a cat and mouse game!

See also: https://mobile-security.gitbook.io/masvs/security-requirements/0x15-v8-resiliency_against_reverse_engineering_requirements

CHECKING IF REVERSE ENGINEERING TOOLS ARE USED (ANTI-FRIDA)

- Checking the integrity of the app
  - Check if the app is being tampered with frida-gadget library
- Check The Environment For Related Artifacts
  - Check if Frida-server is running on the device
- Checking For Open TCP Ports
  - Frida-server bind to TCP Port 27042 by default, check if it's open
- Checking For Ports Responding To D-Bus Auth
  - Frida server uses D-Bus protocol to communicate, send a D-BUS AUTH command to check if the Frida-server response.
- Scanning Process Memory for Known Artifacts
  - Scan the memory for artifacts found in Frida's libraries, e.g. the string "LIBFRIDA" present in all versions of frida-gadget and frida-agent.

See also: https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05j-Testing-Resiliency-Against-Reverse-Engineering.md

CHECKING IF DEVICE IS JAILBROKEN (MULTIPLE CHECKS)

The goal of jailbreak detection is to make running an app on a jailbroken device more difficult, which in turn blocks some of the tools and techniques reverse engineers like to use.

Common Jailbreak Detection Methods:

- File existence checks (e.g. Cydia.app, bash)
- Checking Protocol Handlers (cydia://)
- File permission checks (e.g. writing to /private)

Usually two actions by an app, if it detects a jailbroken device:

- Informs the user about the insecure environment (decision is up to the user to continue or not)
- Termination of the app

See also: https://mobile-security.gitbook.io/mobile-security-testing-guide/ios-testing-guide/0x06j-testing-resiliency-against-reverse-engineering#jailbreak-detection-mstg-resilience-1

JAILBREAK DETECTION BYPASS

- Jailbreak detection mechanisms tries to prevent users from using a Jailbroken device.
- Usually two actions by an app, if it detects a jailbroken device:
  - Informs the user about the insecure environment (decision is up to the user to use it or not)
  - Termination of the app
- Especially popular in gaming and financial mobile apps.

LAB TIME!

Open the file Lab-iOS.pdf

Section: Lab – Bypassing Jailbreak Detection

Please type the commands and do not copy & paste!

1.You will remember them easier

2.Some PDF readers do not copy properly and miss whitespaces etc.

OBFUSCATION OF SOURCE CODE

- Obfuscation is a process of transforming code into a form that is difficult to disassemble and understand and is an integral part of every software protection scheme.
- Swift-Shield
  - SwiftShield can be used to perform name obfuscation. It reads the source code of the Xcode project and replaces all names of classes, methods and fields with random values before the compiler is used.
  - https://github.com/rockbruno/swiftshield

SwiftSecurity 〉 SwiftSecurity 〉 ViewController.swift 〉 Button_JB(_:)

```
9   import UIKit
10
11  class ViewController: UIViewController {
        @IBOutlet weak var Label_Jailbreak: UILabel!
        @IBOutlet weak var Label_Debugger: UILabel!
        @IBOutlet weak var Label_Emulator: UILabel!
        @IBOutlet weak var Label_RE_Tools: UILabel!
16
17
18
        @IBAction func Button_JB(_ sender: UIButton) {
20
21          let jailbreakStatus = IOSSecuritySuite.amIJailbrokenWithFailMessage()
22          if jailbreakStatus.jailbroken {
```

SwiftSecurity 〉 SwiftSecurity 〉 ViewController.swift 〉 ZcGRMAcZDpQiAXVUMuoVQOsEIwGKQAFn(_:)

```
9   import UIKit
10
11  class hTOUoUmUcEZUqhVHRrjrMUnYqbdqWByU: UIViewController {
        @IBOutlet weak var Label_Jailbreak: UILabel!
        @IBOutlet weak var Label_Debugger: UILabel!
        @IBOutlet weak var Label_Emulator: UILabel!
        @IBOutlet weak var Label_RE_Tools: UILabel!
16
17
        @IBAction func ZcGRMAcZDpQiAXVUMuoVQOsEIwGKQAFn(_ sender: UIButton) {
19
20          let jailbreakStatus = uJQzdSwBrpvbyTYcJiDPSTcvZxmwEenu.XqiHjHXmxWISjvBHJhdWEaqcENHgssSL()
21          if jailbreakStatus.jailbroken {
```

See more: https://github.com/OWASP/owasp-mstg/blob/master/Document/0x06j-Testing-Resiliency-Against-Reverse-Engineering.md

CHECKING IF APP WAS REPACKAGED (ANTI-TAMPERING)

- Compare Bundle ID
- Compare mobile provision file
  - Compare the hash value of embedded.mobileprovision with a specified value
- Compare current hash value of executable file
  - Dynamic library
  - Image name
- https://github.com/securing/IOSSecuritySuite/blob/master/IOSSecuritySuite/IntegrityChecker.swift

CHECKING IF A DEBUGGER IS USED (ANTI-DEBUGGING)

- Using ptrace with PT_DENY_ATTACH
  - Ptrace is a system call that is primarily used in tracing and debugging.
  - ptrace(31, 0, 0, 0) and 31 is the value of PT_DENY_ATTACH.
  - informs the operating system that it doesn't want to be traced or debugged.
- Using SYSCTL
  - The sysctl utility retrieves kernel state and allows processes with appropriate privilege to set kernel state.
  - Retrieve the kinfo.kp_proc.p_flag process information flag, and check if it's being debugged.
  - #define P_TIMEOUT 0x00000400 /* Timing out during sleep */
  - #define P_TRACED 0x00000800 /* Debugged process being traced */
  - #define P_DISABLE_ASLR 0x00001000 /* Disable ASLR */
- Apple Documentation on detecting the debugger: https://developer.apple.com/library/archive/qa/qa1361/_index.html

```
11
12  internal class DebuggerChecker {
13
14      // https://developer.apple.com/library/archive/qa/qa1361/_index.html
15      static func amIDebugged() -> Bool {
16
17          var kinfo = kinfo_proc()
18          var mib: [Int32] = [CTL_KERN, KERN_PROC, KERN_PROC_PID, getpid()]
19          var size = MemoryLayout<kinfo_proc>.stride
20          let sysctlRet = sysctl(&mib, UInt32(mib.count), &kinfo, &size, nil, 0)
21
22          if sysctlRet != 0 {
23              print("Error occured when calling sysctl(). The debugger check may not be reliable")
24          }
25
26          return (kinfo.kp_proc.p_flag & P_TRACED) != 0
27      }
28
```

See more: https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05j-Testing-Resiliency-Against-Reverse-Engineering.md

ASK FOR 2 APP BUILDS!

- Security controls, such as anti-debugging and jailbreak detection, may dramatically slow down testing.
- To efficiently use your time, you may want to request two different app build variants from the development team:
  - Release build
  - Debug build (deactivated security controls)
- Testing two different builds is the most efficient way to cover all test cases (might not be possible in all testing scenarios).

MASTG HANDS-ON - IOS EDITION

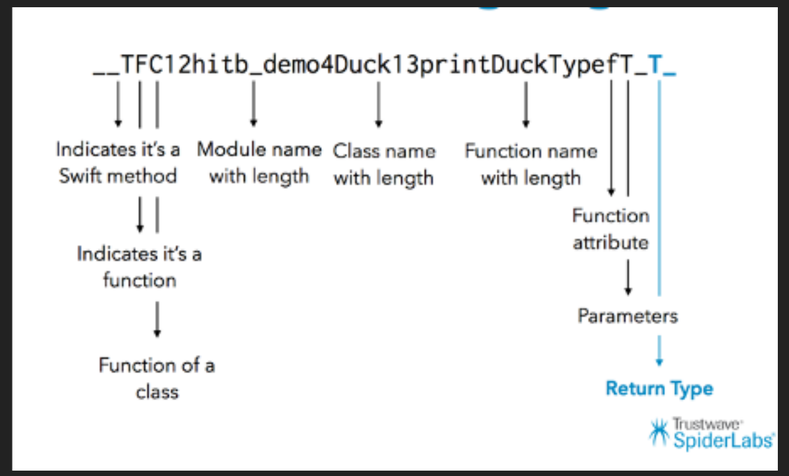
BYPASSING JAILBREAK DETECTION IN SWIFT THROUGH FRIDA

SWIFT MANGLING

Objective-C and Swift are fundamentally different, the programming language in which the app is written affects the possibilities for reverse engineering it. For example, Objective-C allows method invocations to be changed at run time. This makes hooking into other app functions (a technique heavily used by Frida and other reverse engineering tools) easy. This "method swizzling" is not implemented the same way in Swift, and the difference makes the technique harder to execute with Swift than with Objective-C.

```
1  -[ViewController showJailbreakAlert]
2  -[ViewController protectAgainstDebugger]
3  -[ViewController viewDidLoad]
4  -[ViewController handleButtonClick:]
5  -[ViewController didReceiveMemoryWarning]
6  -[ViewController theLabel]
7  -[ViewController setTheLabel:]
```

Objective C



Swift

SWIFT MANGLING

_TtC7DVIA_v232JailbreakDetectionViewController

_T = The prefix for all Swift Symbols

C = Class

7DVIA_v2 = The module name with a prefixed length (DVIA_v2 = 7 characters)

32JailbreakDetectionViewController = The class name the function belongs to, again, with a prefixed length (JailbreakDetectionViewController = 32 characters)

SWIFT MANGLING

Demangling via command line, by using the command “swift”:

```
~ ❯ swift demangle _TtC7DVIA_v232JailbreakDetectionViewController
_TtC7DVIA_v232JailbreakDetectionViewController ---> DVIA_v2.JailbreakDetectionViewController
```

Mangling: https://github.com/apple/swift/blob/master/docs/ABI/Mangling.rst

A very good example for manual reverse engineering of iOS Apps can be found in the MSTG:

https://mobile-security.gitbook.io/mobile-security-testing-guide/ios-testing-guide/0x06c-reverse-engineering-and-tampering#manual-reversed-code-review

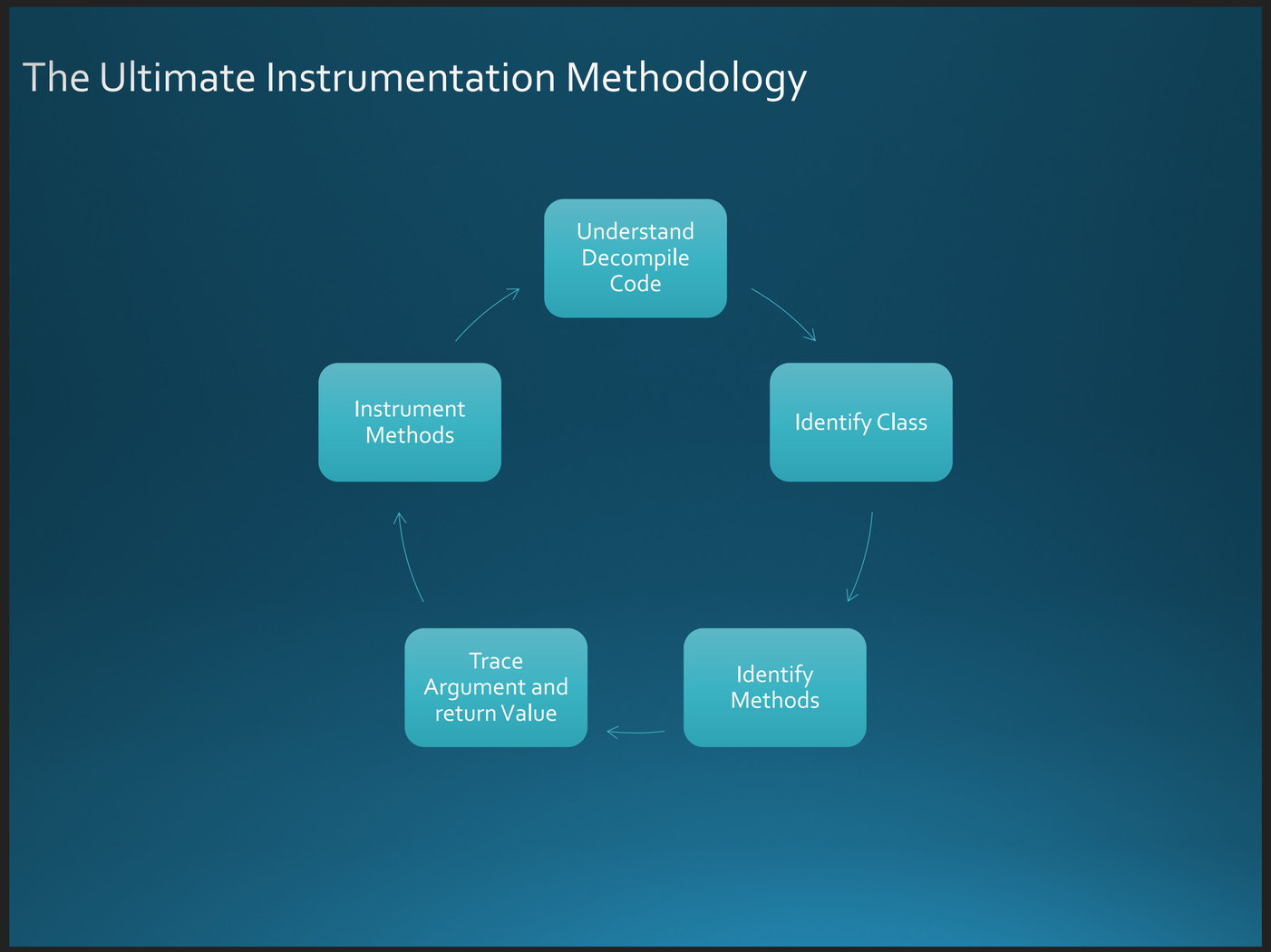
BYPASSING JAILBREAK DETECTION WITH FRIDA

- Swift Security iOS app
  - Utilise iOS Security Suite Swift library
  - https://github.com/securing/IOSSecuritySuite
  - Interesting target
    - Jailbreak detection in a library written in Swift!

Swift Security

TACTICS OVERVIEW

- Frida-trace to identify library class and function
- Trace function with Frida
- Print Argument and return value
- Instrument!!!



LAB TIME!

Open the file Lab-iOS.pdf

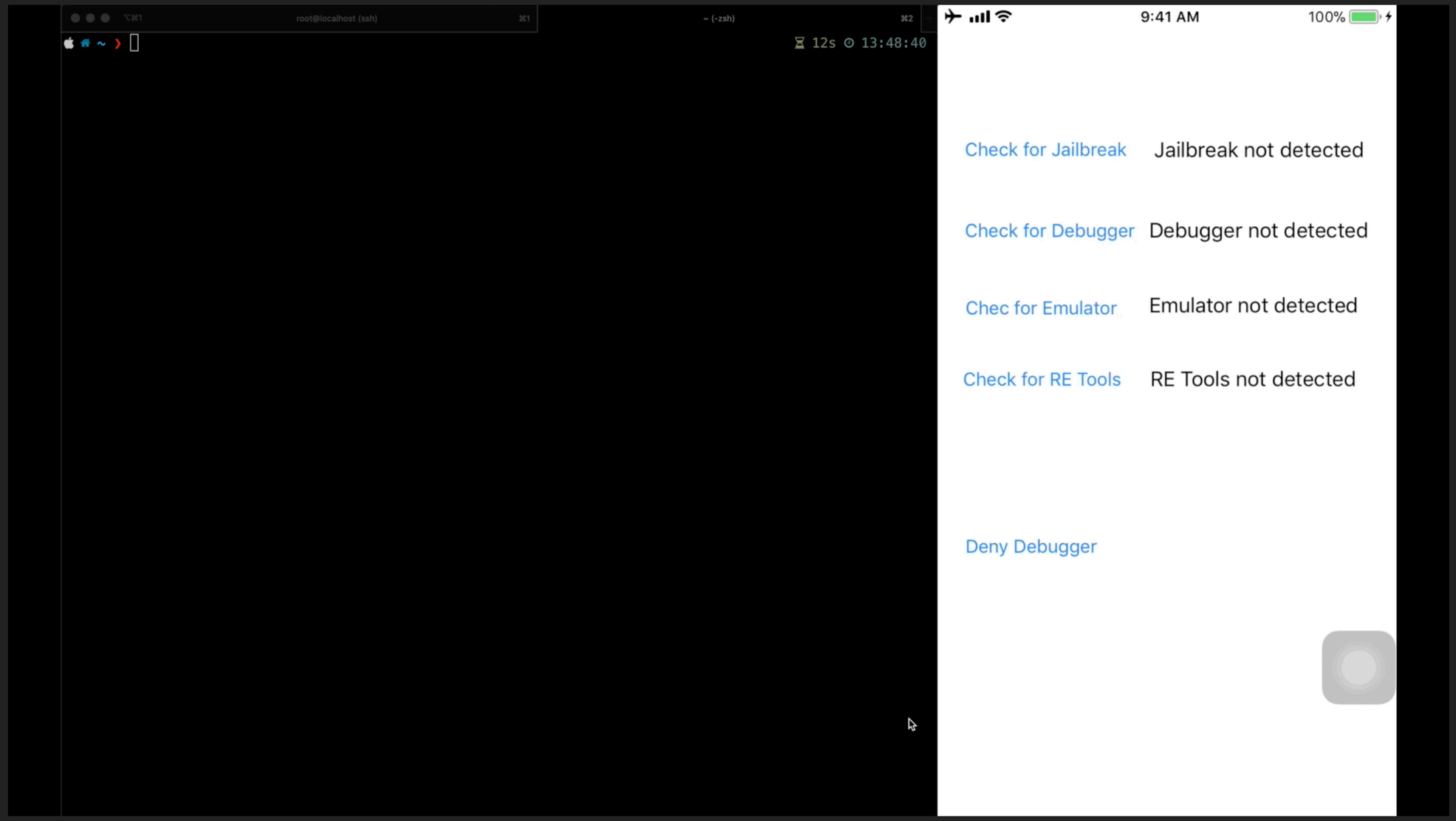
Section: Lab - Frida and Swift

Please type the commands and do not copy & paste!

1.You will remember them easier

2.Some PDF readers do not copy properly and miss whitespaces etc.

FRIDA-TRACE LIBRARY FUNCTION



FRIDA – CHEAT SHEET FOR SWIFT HOOKING

`# frida-trace -U -I`	Frida-trace to identify Class and methods
`> Process.enumerateModules()`	Enumerate library function
`> targetModule.enumerateExports()`	Enumerate imported function from external lib
`> retval.replace(0x00)`	Replace return value of a function
`>targetModule.getExportbyName('$MODULE,$FUNCTION')`	Retrieve memory address of the exported Function

BYPASSING IN SWIFT

- It's always possible to instrument regardless if the function is written in Swift, ObjC, exported, imported etc.
  - https://nik.re/posts/2020-05-15/debugging_and_instrumenting_swift_apps_ios_13
- Manual hooking: Reverse engineering and calculating memory offset
  - https://syrion.me/blog/ios-swift-antijailbreak-bypass-frida/
- Always try to leverage Frida API to retrieve memory address of function when possible.

iOS Training

CTF TIME!

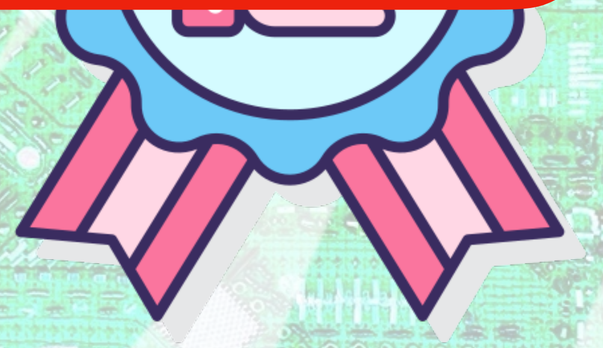
iOS Training

Wrap up Day 2

iOS - Summary of Day 2

- OWAPS MAS project:
  - Use the OWASP MASVS to be aware of the most critical areas of the m surface.
  - Use the OWASP MASTG to get recipes on how to do mobile app securi
  - The OWASP MASVS is recognised by Android and linked into the devel documentation, as part of the Android Application Security Knowledge Project.
- For Deeplinks:
  - When using deeplinking, use Android App Links that allows a verification of the Android App and URL to prevent link hijacking.
  - Implement Strong Input Validation for any input coming from Deep links
  - Ensure that deep links are processed only after authentication, to avoid data leakage.
  - See also the deep link test case in the MASTG.

UPDATE



© 2024 by Bai7 GmbH | All rights reserved

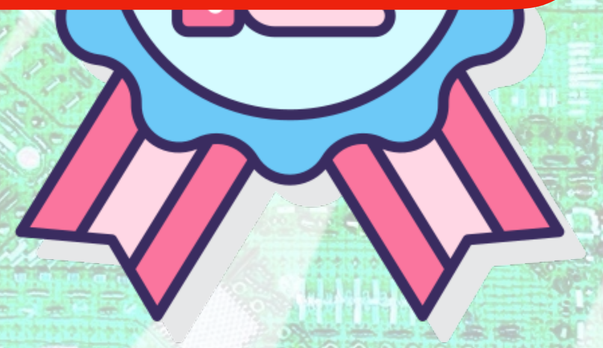
iOS - Summary of Day 2

- SAST:
  - ***First step for SAST:*** Start small with a subset of the available vulnerabilit
    more checks after others have been remediated and are under control!
  - In case you have a variety of different programming languages to cover
    scanning instead of SAST!

- Software Composition Analysis (SCA)
  - Scanning of 3rd party libraries for security vulnerabilities
  - Vulnerable libraries need to be updated or removed
  - SBOM is an essential part Application Security and need to be continuously created and monitored

- Sensitive Data in mobile Apps:
  - Only store data in the internal/external storage that is needed for the mobile app.
  - Only encrypt sensitive data in the App's sandbox if there is a need to. If so, use the Android KeyStore, or the Security Crypto Library.

UPDATE



© 2024 by Bai7 GmbH | All rights reserved

And we're done… Thanks for your time!

Any more questions?

Otherwise contact me:
sven@bai7.at
Linkedin

Thank you for your time today and I hope you can use your new skills soon and spread your new knowledge!

See you tomorrow.

© 2024 by Bai7 GmbH | All rights reserved