SYNOPSYS®

Build hybrid mobile apps like a security pro!

Vineeta Sangaraju
Sr. Research Engineer
July 14, 2023

About me

- Consultant turned Researcher at Synopsys
- Master's degree in Computer Science
- Violin, dance, hiking
- Current interest - vegetable gardening

Agenda

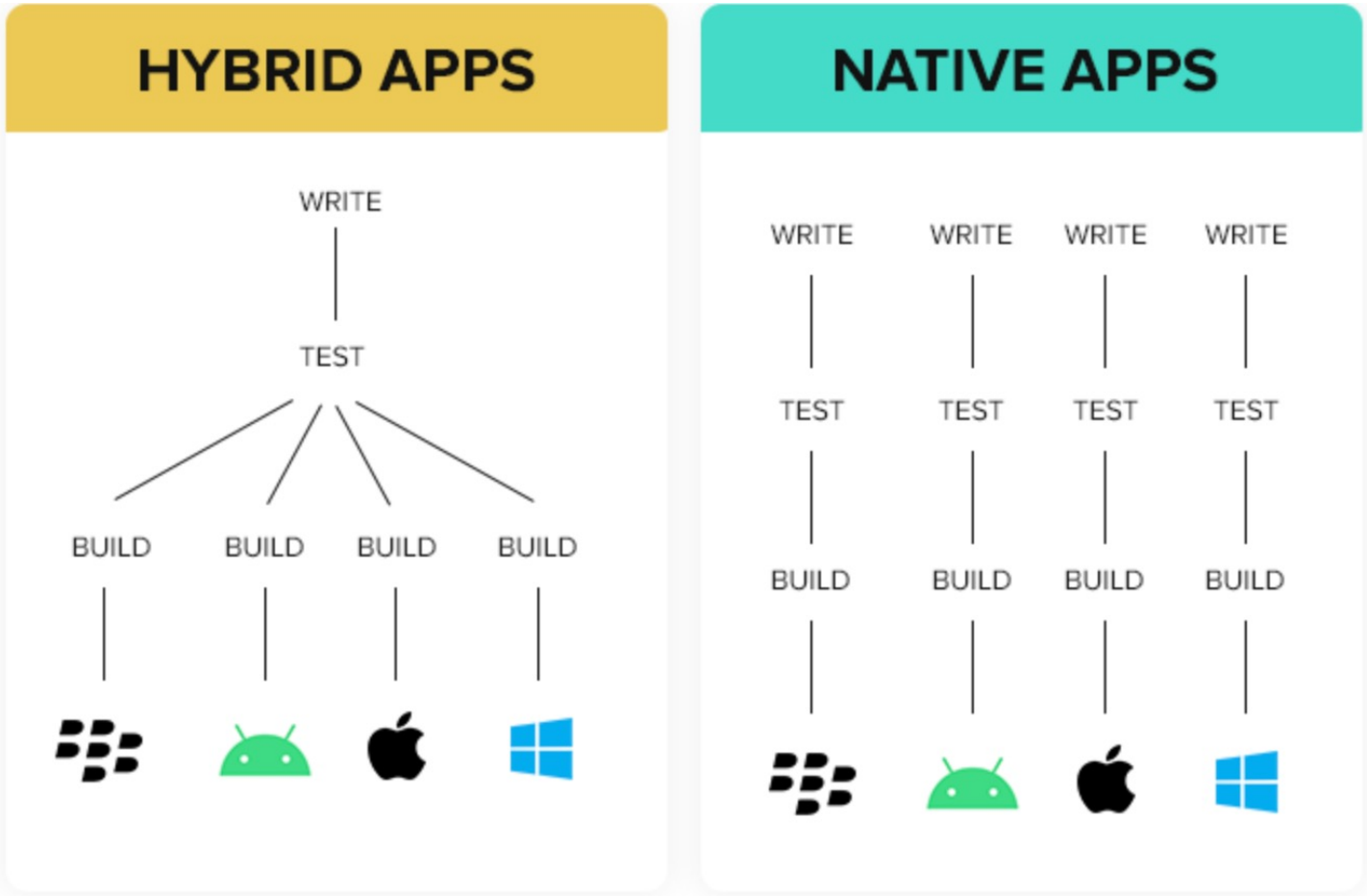
- Mobile applications 101
- Characteristics of a mobile app
- Threats and threat agents
- Hybrid mobile applications 101
- Following principles of application security
- Exercise - Spot the bad apple
- Summary
- QA

Mobile applications 101

Back to basics

Mobile applications 101

- Apps to use ”on the go”
- Not just for productivity
- Android and iOS – 99% of the market
- 800 iOS, 2k Android apps released per week
- SDKs for development
- Native and hybrid apps



Mobile apps - characteristics

- Apps are “contained” for isolation
- Apps use inter process communication
- Code is located on the device
- Data is stored on the device – persistently
- Apps still communicate with server
- Authentication could be client-side
- iOS Apps are vetted before release
- Jailbreaking and rooting

Assets and threats: Mobile apps

- Typical assets
  - user data – client side, long term storage
  - application data – code and assets live client side
  - Functionality – compiled code lives on client side
  - Credentials – handled client-side

- Typical threat agents
  - Malicious apps
  - Non-owner of device – stolen phone
  - MitM – network attacks
  - Malicious user – hacker

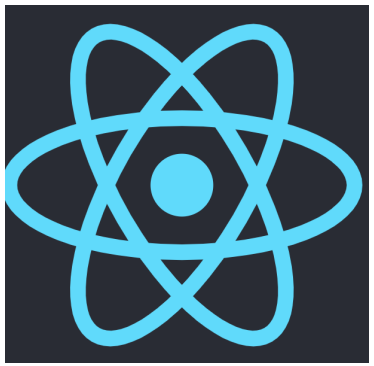
- Overall, low likelihood

Hybrid mobile applications

What are they made of?

Building hybrid mobile apps

- Select framework – Flutter, React Native etc.
- Implement authentication - Store auth tokens, incorporate biometrics
- Handle API calls, user inputs, data storage –
  - UI Components or WebViews for user inputs
  - Make GET/POST requests to app server using library APIs, ex. fetch()
- Google play, BMW, eBay, Skype, Tesla, Walmart



What are these libraries?

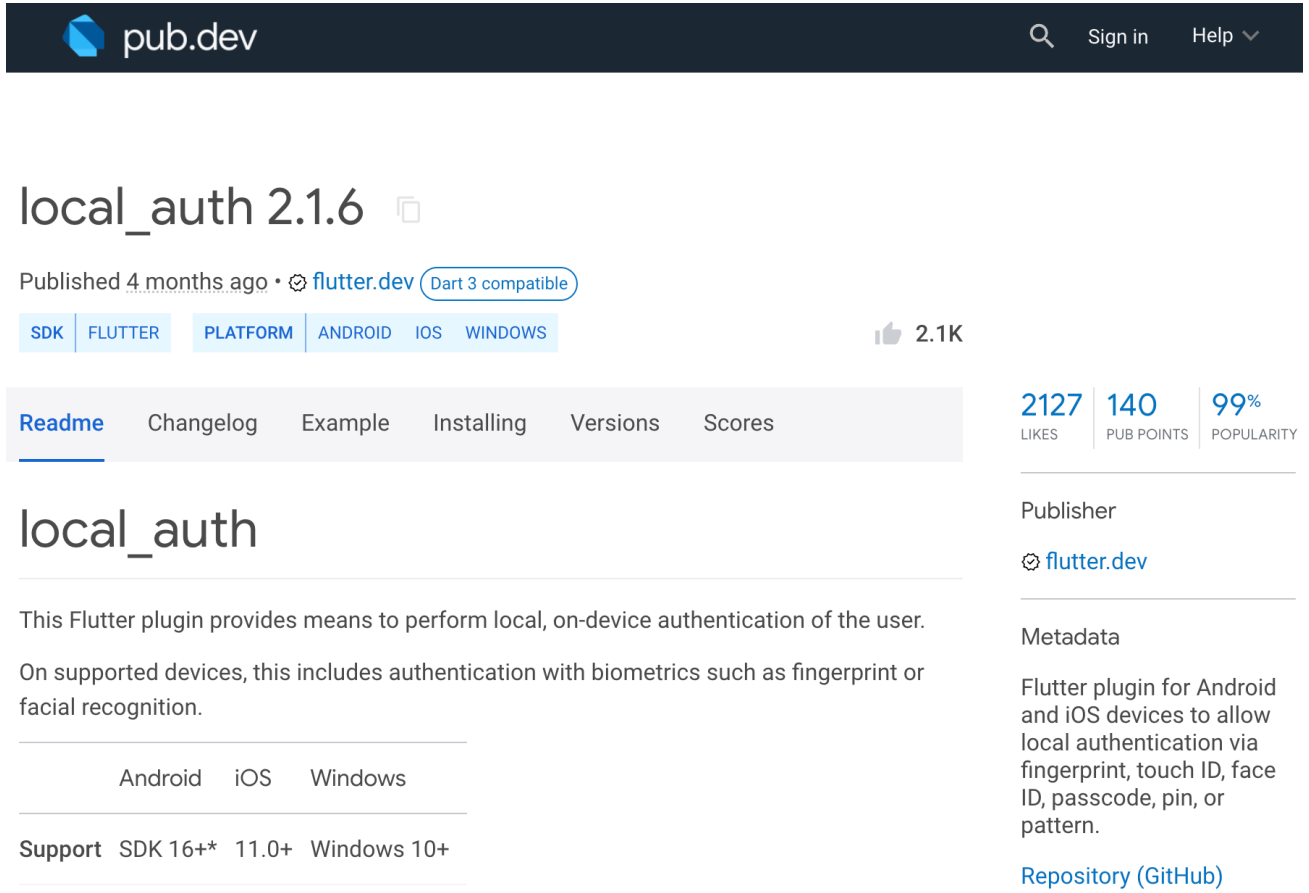
- Reusable packages or modules
- Maintained by community, 3rd party or open source
- Pre-built components, tools and utilities
- Easy integration into an app -> fast development
- Framework bridges gap, library defines the set of desired native capabilities
- Flutter 36k, React Native 46k

Example

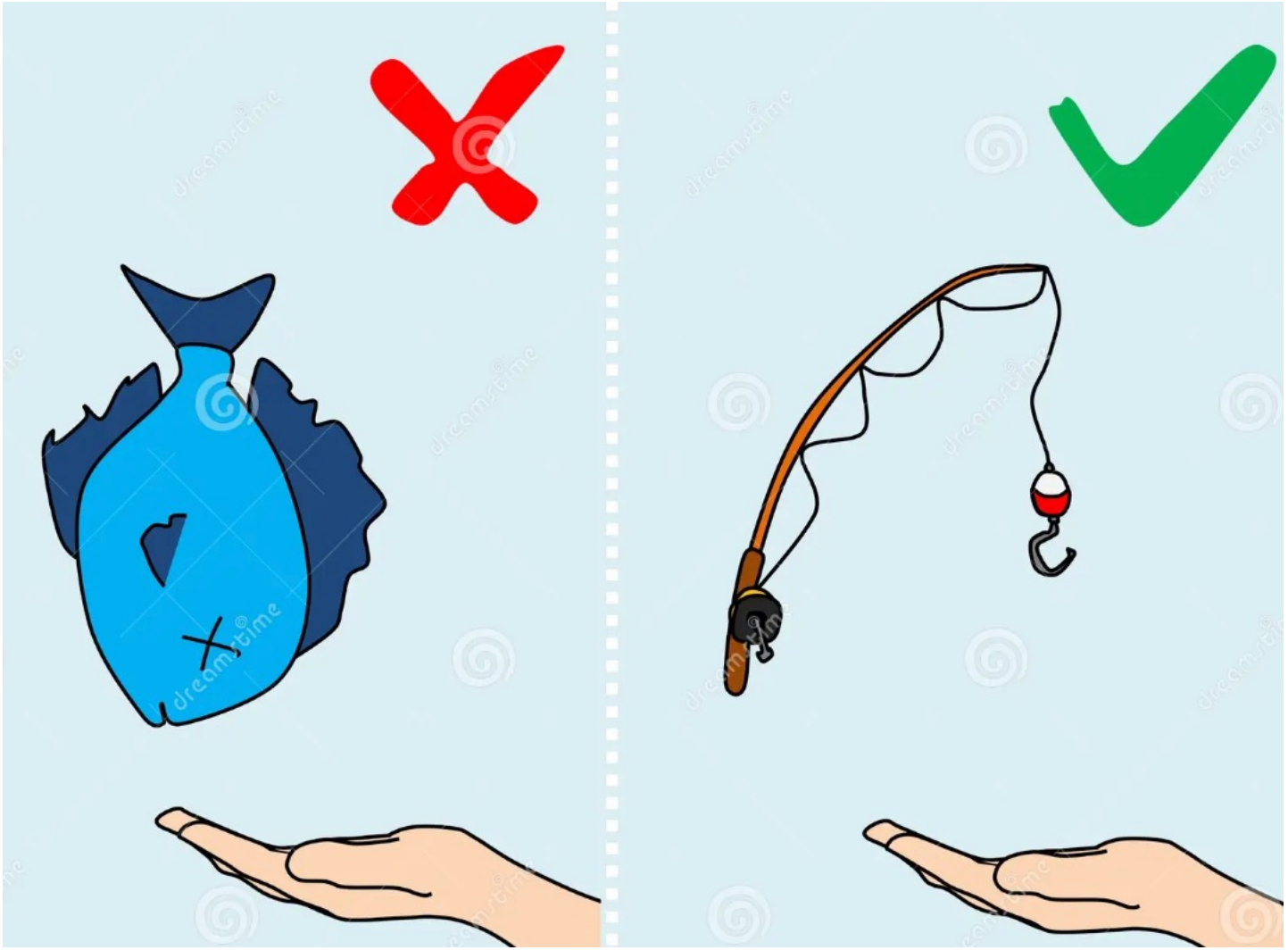
Flutter library

```
import 'package:local_auth/local_auth.dart';
final LocalAuthentication auth = LocalAuthentication();

final bool didAuthenticate = await auth.authenticate(
localizedReason: 'Please authenticate to show account balance');
```



List of secure libraries

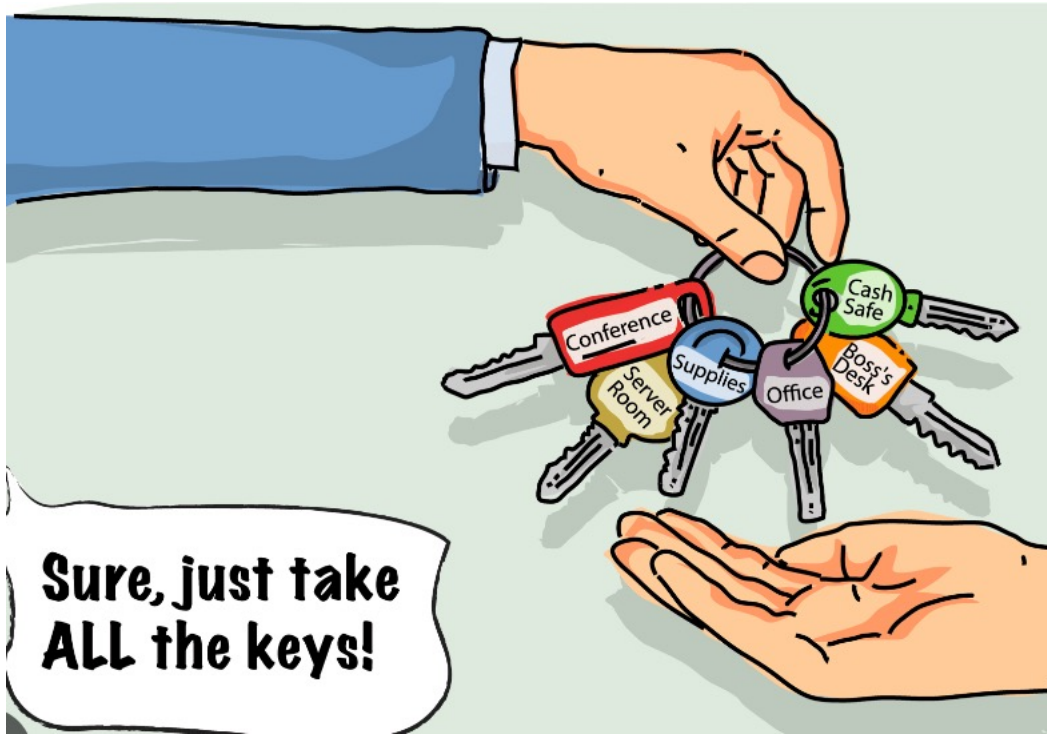


Principles of application security

And how to apply them in hybrid mobile apps

Principles of application security: Least Privilege

- Principle: Less is more
- Permissions in a mobile app such as camera, location, mic
- Minimum set of permissions
- Set permissions in configuration files
- Caution - Runtime permission requests
- Evaluate the library
  - Does not request unnecessary permissions
  - Allows configuration
  - No strict defaults
  - Supports authz controls for ex. data storage



Principles of application security: Security by obscurity

- Principle: Secrecy for security
- Debunked, but useful for mobile apps
- Secure all parts:
  - Code – use obfuscation, code splitting
  - data – use encryption
  - network – use HTTPS
- Evaluate the library
  - encryption and key generation algorithms are secure, check defaults
  - Code obfuscation for all code, not just JS
  - Sensitive UI components can be masked

Principles of application security: Minimize attack surface

- Principle: limit the number of entry points into your app
- Mobile OS sandboxing is not enough
- Secure data stored on device and in transit
- Clipboard, Keyboard, Logs, WebViews
- Evaluate the library
  - Forms should support validation of inputs
  - Avoids excessive logging
  - No debug code
  - JavaScript in WebViews
- Deep links – Apply the ‘KISS’ principle

```
handleWebViewLoad = () => {
   const { userInput } = this.state;
   const webView = this.webview.ref;

webView.injectJavaScript(`document.getElement
      ById('userInput').innerHTML = '${userInput}';`);
}
```

Principles of application security: Client trust

- Principle: Selectively trust
- Trusted on busted
- Avoid blindly trusting the
  - device, user, library, network
- 3rd party risks are inherent, consider SCA
- Sketchy package managers
- Evaluate the library
  - Verify the publisher and CVEs for libraries
  - Employ controls to secure the network, for ex. ATS
  - Use a safety net for bad user choices, ex. Keyboards, masking

Principles of application security: Defense in depth

- Principle: set of best practices for holistic security
- Ex. JB detection, latest OS version, auth controls on keychain, debuggers, business logic
- Anti-reverse engineering controls
- Evaluate the library –
  - Deprecated methods,
  - Works same as native code
  - insecure defaults
  - Verify ease of a bypass i.e. event based vs result based APIs

Spot the bad apple

Let’s apply our learning

Exercise – evaluate these library APIs

Constructors

HashCryptRaw(HashAlgo type)
Construct with type of algorithm

Sha3_384 → const HashAlgo

Sha3_512 → const HashAlgo

Keccak_224 → const HashAlgo

Keccak_256 → const HashAlgo

Keccak_384 → const HashAlgo

Keccak_512 → const HashAlgo

Sha1 → const HashAlgo

RipeMD_128 → const HashAlgo

RipeMD_160 → const HashAlgo

RipeMD_256 → const HashAlgo

RipeMD_320 → const HashAlgo

Blake2b → const HashAlgo

MD2 → const HashAlgo

MD4 → const HashAlgo

MD5 → const HashAlgo

Tiger → const HashAlgo

Whirlpool → const HashAlgo

Exercise – evaluate these library APIs

```
10    export type Options = {
11        body?: string | {},
12        credentials?: string,
13        headers?: Header,
14        method?: 'DELETE' | 'GET' | 'POST' | 'PUT',
15        sslPinning: {
16            certs: Array<string>
17        },
18        timeoutInterval?: number,
19        disableAllSecurity?: boolean
20    }
21
```

Exercise – evaluate these library APIs

launch method

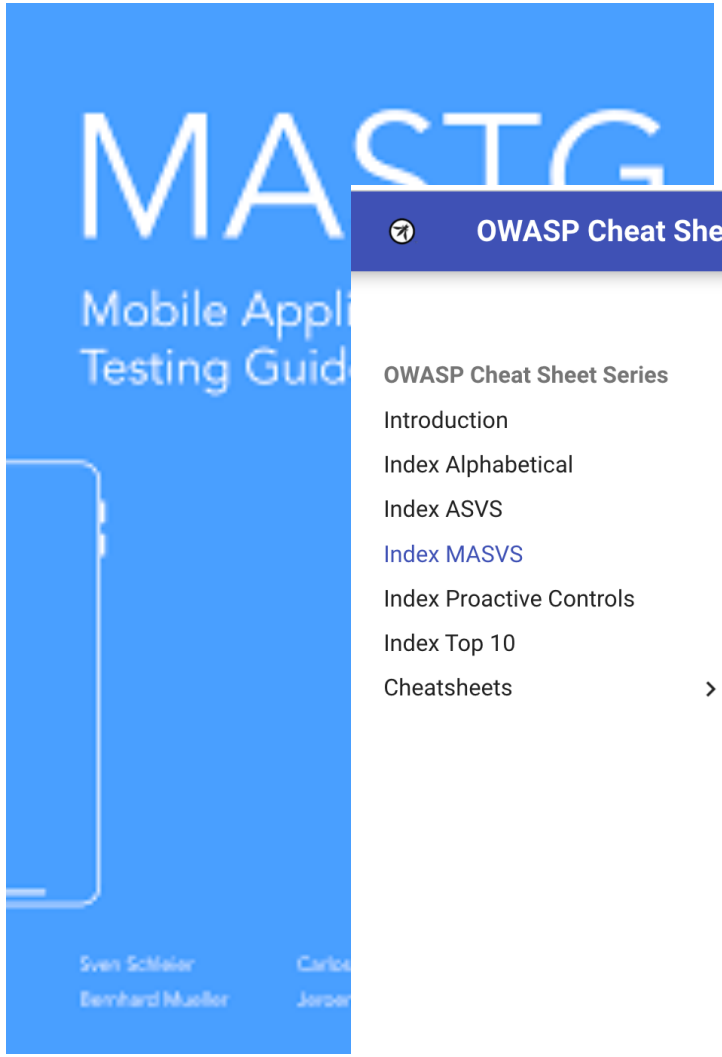
```
Future<void> launch(
    String url,
    {Map<String, String>? headers,
    Set<JavascriptChannel> javascriptChannels = const <JavascriptChannel>{},
    bool withJavascript = true,
    bool clearCache = false,
    bool clearCookies = false,
    bool mediaPlaybackRequiresUserGesture = true,
    bool hidden = false,
    bool enableAppScheme = true,
    Rect? rect,
    String? userAgent,
    bool withZoom = false,
    bool displayZoomControls = false,
    bool withLocalStorage = true,
    bool withLocalUrl = false,
    String? localUrlScope,
    bool withOverviewMode = false,
    bool scrollBar = true,
    bool supportMultipleWindows = false,
    bool appCacheEnabled = false,
    bool allowFileURLs = false,
```

Summary

- Hybrid apps are on the rise
- Dev practices matter more than frameworks & libraries
- Use external libraries, but scrutinize them
  - Verify the library version
  - Disable insecure defaults
  - Verify cross-platform compatibility for the security controls
  - API implementation should be resilient in an untrusted environment
  - Look for CVEs
- Compliment with security controls by the OS

"Skip the blow by blow. Just condense it into a couple of barks."

Resources



OWASP Mobile Top 10

Main | Acknowledgements | Controls

OWASP Cheat Sheet Series
Introduction
Index Alphabetical
Index ASVS
Index MASVS
Index Proactive Controls
Index Top 10
Cheatsheets

MASVS Index

Table of Contents

- Objective
- V1: Architecture, Design and Threat Modeling Requirements
- V2: Data Storage and Privacy Requirements
- V3: Cryptography Requirements
- V4: Authentication and Session Management Requirements
- V5: Network Communication Requirements
- V6: Environmental Interaction Requirements
- V7: Code Quality and Build Setting Requirements
- V8: Resiliency Against Reverse Engineering Requirements

Questions

Thank You