Analysis of Payment Service Provider SDKs in Android

Samin Yaseer Mahmud, K. Virgil English, Seaver Thorn, William Enck, Adam Oest, Muhammad Saad

Presented at Annual Computer Security Applications Conference (ACSAC), 2022

NC STATE UNIVERSITY PayPal

Digital Payment has Revolutionized Commerce

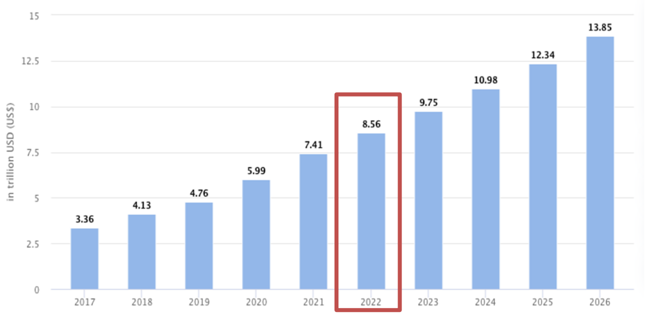
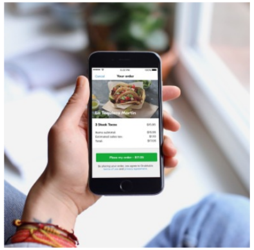
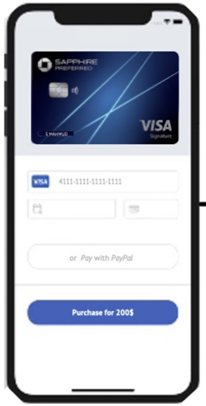


Figure: Worldwide transaction using Digital payment



Mobile applications are becoming a predominant form of payment.

Payment Processing SDK

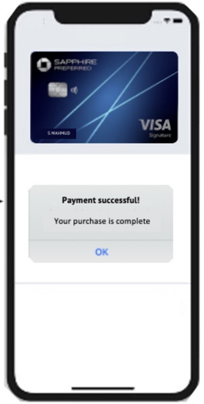


Step 1: SDK collects payment information from users through the app and securely forwards them to payment service provider's (PSP) server.

Step 2: PSP server facilitates the payment processing with the bank maintaining appropriate specification and protocols.

Step 3: Bank informs the PSP whether the payment request is approved or not, completing transaction accordingly.

Step 4: PSP forwards the response back to the SDK; SDK notifies the user showing appropriate message in the app.



Mobile Application Security Verification Standard

- **MASVS (**v1.2) is an industry security standard for mobile applications, proposed by **OWASP**.
- Captures common android security weaknesses
- We designed **28** security checks of 4 broader categories from the standard that are applicable to Payment and SDKs
- Classifies requirements into:
  - MASVS L1 (Ordinary Use Cases)
  - MASVS L2 (Sensitive Use Cases)
- MASVS captures many requirements of PCI DSS

OWASP Mobile Security Project

MASVS Categories

Architecture
Data Storage
Cryptography
Authentication
Network Communication
Platform Interaction
Code Quality
Resiliency against RE

Data Storage Requirements

- Goal is to protect **sensitive user data** in device
- Many requires dataflow analysis for tracking data
  - Mostly **PCI DSS** related
- Others involve looking at syntax in **code**, **layout**, **config** files
- Excluded checks:
  - App removes data from system memory
  - App educates user about processed data

MASVS Check	Approach
Android Keystore not utilized	Syntax (Code)
Accessed data from External Storage	Syntax (Code, Config)
Logged sensitive data	Data Flow
Data shared with third parties	Data Flow
Keyboard cache enabled	Syntax (Layout)
Data leaked through IPC	Data Flow
Data leaked through UI	Syntax (Layout)
Auto backup turned on	Syntax(Config)
Screenshot disabling not detected	Syntax(Code)
Screen lock presence not detected	Syntax(Code)
Sensitive data stored	Data Flow
Sensitive data stored unencrypted	Data Flow

☐ MASVS-L1 ☐ MASVS-L2

Goal and Key Challenges

Goal: To build a novel tool that would **enable dataflow analysis** of an **SDK** without the need of a **host app**

Challenges:

- **Entry points** to SDK analysis are not easily inferred
- Existing tools consider **whole application context**

Goal: To perform a security evaluation of payment SDKs with industry security standards

Challenges:

- Security standards are written in **natural languages**
- Security standards are written with **application** in mind

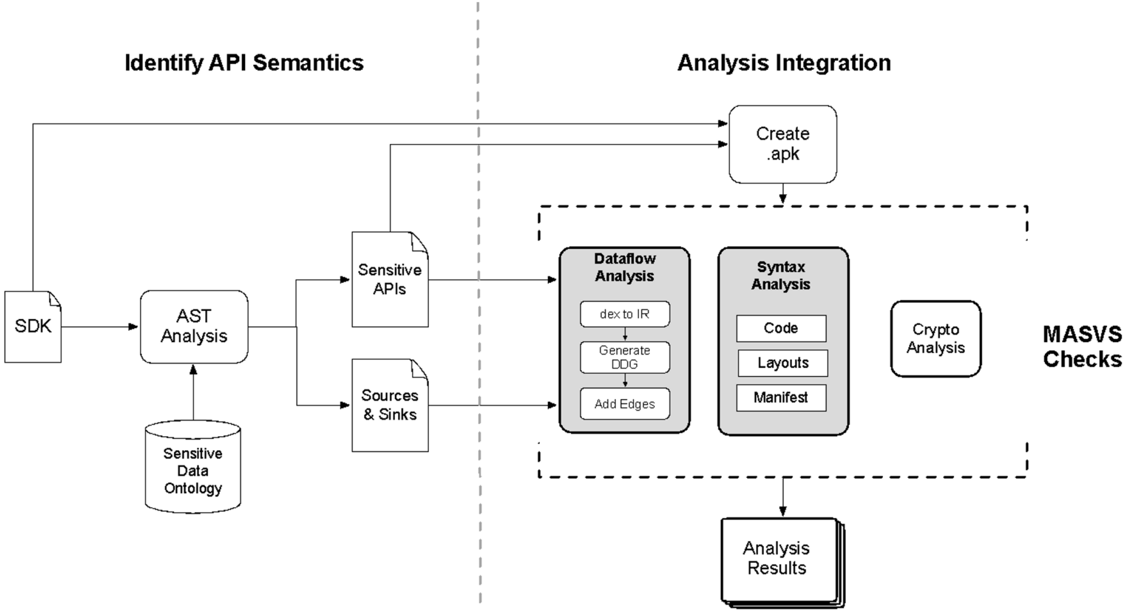
AARDroid Architecture

Preprocessing Steps:

1: Connecting to the SDK:
- ➔ Create empty android project
- ➔ Include SDK

2: Identifying relevant API:
- ➔ Consider all public API
- ➔ Consider API related with Strings

3: Resolving API Semantics:
- ➔ Incorporate Text analytics
- ➔ Incorporate Data ontology
- ➔ Connect dummy edges from app -> SDK



Resolving API Semantics

Step 1: Generate the AST of the SDK to extract API Semantics

Step 2: Determine if an API is ***sensitive***

- Inspect API **method** name
- Inspect API **parameters**
- `E.g., public void processTransaction (String creditCardNumber)`

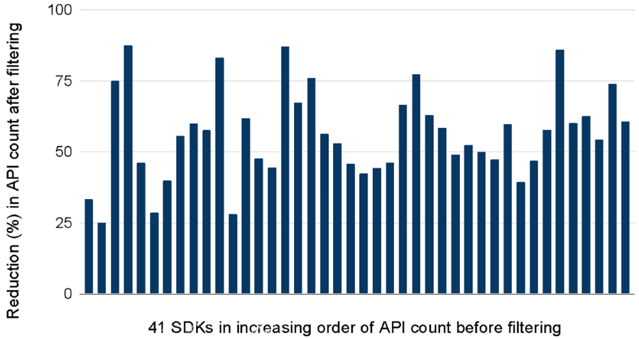
Step 3: Identify sensitive ***parameter***

- Data Ontology

Step 4: Assign sensitivity as High, Medium, Low

Table 1: Accuracy of sensitive API identification

SDK	Total API	FN	API Count After Filtering	API Reduction	TP	FP
redsys	15	0	9	40%	7	2
simplify	18	3	8	55%	8	0
tranzzo	26	2	9	43%	7	2
square	30	6	5	84%	4	1
payjp	47	3	6	88%	0	0



Payment SDK Analysis

SDK Data Set

- No specific market for hosting SDK
- 50 android Payment SDK
- SDKs collected from Google Pay and Apple Pay's website

AARDroid Analysis

- We ran our 28 MASVS checks on 50 Payment SDKs
- 9 SDK code were obfuscated, 6 dataflow checks were skipped on these
- 24 SDK did not have a UI, 4 UI checks were skipped on these
- Average analysis time ~8 minutes

Manual Validation

- We manually validated 50 SDKs over a two-week period with JD-GUI, Fernflower

Highlighted Findings

- **3 SDKs** save unencrypted sensitive information such as **credit card number** and **CVC**. A fourth SDK stores a CVC in encrypted form.
  - *PCI-DSS standards prohibit such storage.*
- **11 SDKs** rely on outdated cryptographic primitives for sensitive functionality.
  - e.g., DES, Blowfish, SHA1, MD5
- **10 SDKs** do not follow industry standards on UI for taking credit card data from users.
  - 4 SDK display CVC in plain text
  - 3 SDK displays unmasked Credit card number (not reported)
  - 6 does both

Highlighted Findings

- **26 SDKs** use WebViews, all of which have at least one improper configuration, that introduces unnecessary attack surface.
  - 20 enables JS , 8 allows JS Bridge, 23 allow local file access, 21 do not clear WebView cache
- **37 SDKs** do not meet at least one of the basic (MASVS-L1) security requirements, **12 SDKs** do not meet at least one of the more advanced (MASVS-L2) security requirements.

Case Study

DotPaySDK

dotpay®

Worldwide secure payment

Participating processors

We've partnered with many Payments Service Providers (PSPs) around the world to make Google Pay integrations simple.

Google Pay

- Stores **unencrypted** credit card number
- Stores Encrypted **CVC**
- Stored **encryption key** on Shared preference **base64 encoded**
- Less secure key size (**RSA 1024**)
- UI leaks **CVC** and **un-masked Credit card** number
- Misconfigured WebView with all excess privileges
- **16** out of **28** MASVS requirements not met

```
public void addCreditStoreSecurityCode(String credit_card_security_code)
  {
    try {
        String secure_code = this.RSAEncrypt(credit_card_security_code);
        L.e("secure_code " + secure_code);
        this.sharedPreferences.edit().putString("
    credit_card_security_code", secure_code).commit();
        this.setOneClickCVVDataAvailable(true);
    } catch (NoSuchAlgorithmException var3) {
        var3.printStackTrace();

    }
}
```

Listing 1: Code snippet of persisting encrypted CVC in Dotpay

```

...
KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
keyPairGenerator.initialize(1024);
KeyPair keyPair = keyPairGenerator.genKeyPair();
PublicKey publicKey = keyPair.getPublic();
PrivateKey privateKey = keyPair.getPrivate();
this.sharedPreferences.edit().putString("publicKey", Base64.
   encodeToString(publicKey.getEncoded(), 2)).commit();
this.sharedPreferences.edit().putString("privateKey", Base64.
   encodeToString(privateKey.getEncoded(), 2)).commit();
...
```

Listing 2: Code snippet of insecurely instantiating and storing RSA key for CVC encryption

Responsible Disclosure

Goal: Get feedback from SDK developers regarding the findings.

- Reached out **46** SDK vendors
- **27** acknowledged receiving it
  - **13** were auto-generated
- **7** vendors are investigating the issues (no further response)
- **1** claimed to **release patches** (BlueSnap)
- **1** claimed them **less severe** (Google Wallet)
- **1** claimed **deprecated** and **informed affected merchants.** (PayPal)
- **4** acknowledged majority of the concerns (**26 out of 37**)

Insights:

- Some issues are **intentional business decision**
- Some issues are **not appropriate** for SDK

Hi,

thank you for your report. I'll verify your findings, however it looks like most of the code that you mention is no longer part of our SDK. The version you analysed is over 4 years old and since then the SDK changed significantly, including changing the ownership of the code between two companies. Is your static analysis tool public? If so, can I get access to it?

> Hi
>
> Thank you for your prompt response. Our data collection step was performed in October 2020 and v 1.2.18 of the SDK was the most updated version till then and therefore our analysis was on that specific version. However we have re-run our analysis on the most recent version (v 2.0.203) of the SDK found in this repository It seems like all the security weaknesses are still present in the current code, after manually validating them. It would be helpful if you can further investigate the issue.

Hi,

thank you for the information. I will analyse the SDK and verify these findings. Quick question:

> Section 2.3 of MASVS (v 1.2.1) suggests no sensitive data should be written to application logs. We find sensitive data such as credit card

Can you provide me with specific instances of this issue?

4 months later . . .

Hi,

sorry for the late answer. We are migrating merchants from this SDK to the newer one however, we plan to improve it anyway. Some of the suggestions you made do not fit into the SDK, but lots of them are valid and we plan to introduce changes. Please find below answers to all 16 reported items.

12 out of 16 issues were acknowledged

Summary

- Static analysis tool for analyzing stand-alone SDKs
- Evaluate 50 Payment SDKs against OWASP's MASVS
- Identified several security weaknesses
- Identified gap between security standards and their implementation in payment SDKs
- Responsibly disclosed to SDK vendors

Questions

- Static analysis tool for analyzing stand-alone SDKs
- Evaluate 50 Payment SDKs against OWASP's MASVS
- Identified several security weaknesses
- Identified gap between security standards and their implementation in payment SDKs
- Responsibly disclosed to SDK vendors

Samin Yaseer Mahmud

Ph.D Candidate, NCSU

Expected Graduation: May, 2023

smahmud@ncsu.edu

https://saminmahmud.com

I am looking for full-time roles in industry!