black hat
ASIA 2021
May 6-7, 2021
BRIEFINGS

Mining and Exploiting (Mobile) Payment Credential Leaks in the Wild

Shangcheng Shi, Xianbo Wang, Wing Cheong Lau

The Chinese University of Hong Kong

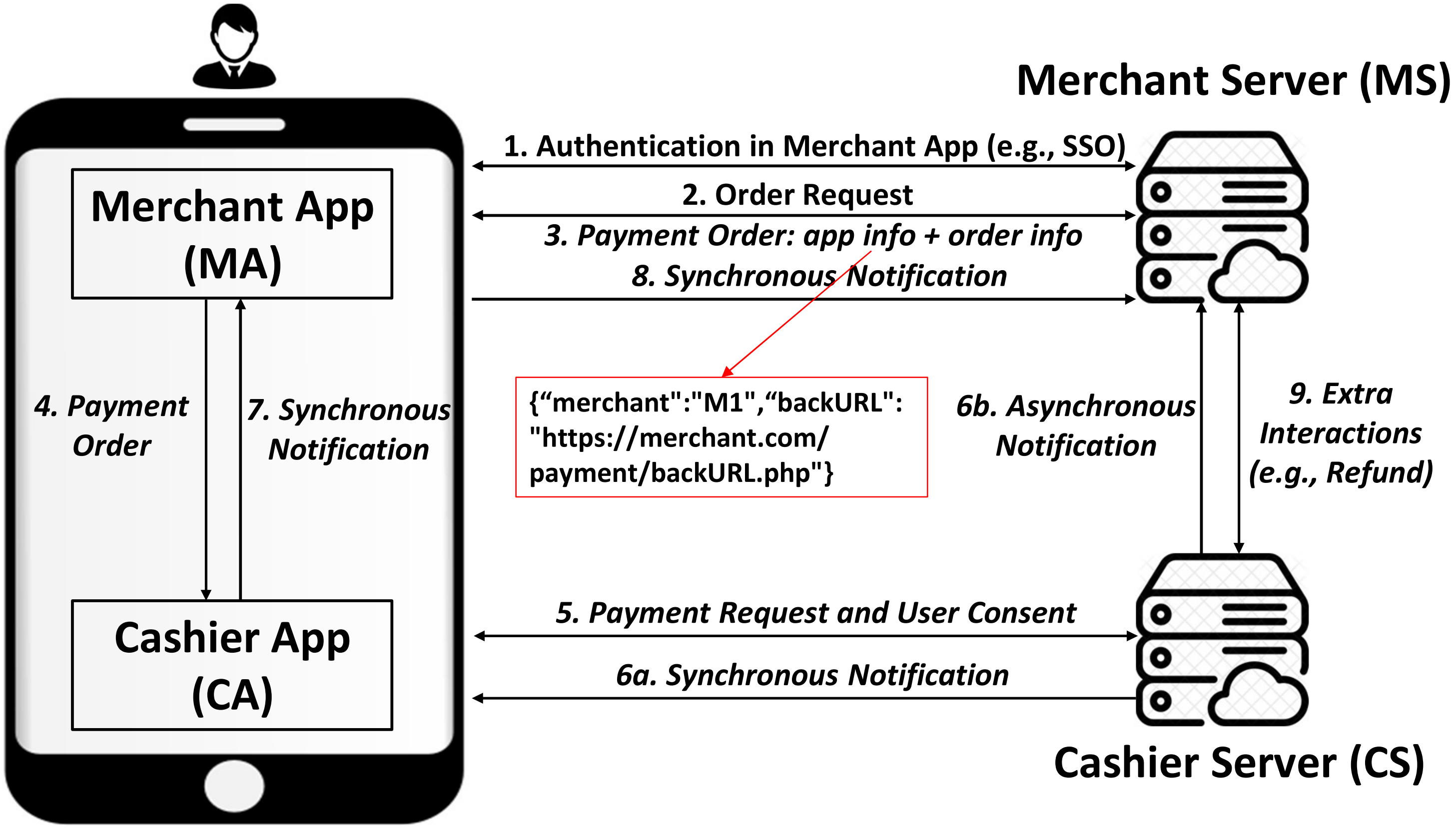
Outline

- Introduction to Third-Party Payment Service for Mobile Apps
- Overview on Payment Credentials
- Leaking Sources of Payment Credentials
- Exploiting the Leaked Payment Credentials
- Automatic Mining for Payment Credentials Leaked in the Wild
- Resolving the Leaking App Identity
- Suggested Fixes

Outline

- Introduction to Third-Party Payment Service for Mobile Apps
- Overview on Payment Credentials
- Leaking Sources of Payment Credentials
- Exploiting the Leaked Payment Credentials
- Automatic Mining for Payment Credentials Leaked in the Wild
- Resolving the Leaking App Identity
- Suggested Fixes

Third-Party Payment Service for Mobile Apps



Merchant Server (MS)

Merchant App (MA)

1. Authentication in Merchant App (e.g., SSO)

2. Order Request

3. Payment Order: app info + order info

8. Synchronous Notification

4. Payment Order

7. Synchronous Notification

```
{“merchant":"M1",“backURL":
"https://merchant.com/
payment/backURL.php"}
```

6b. Asynchronous Notification

9. Extra Interactions (e.g., Refund)

Cashier App (CA)

5. Payment Request and User Consent

6a. Synchronous Notification

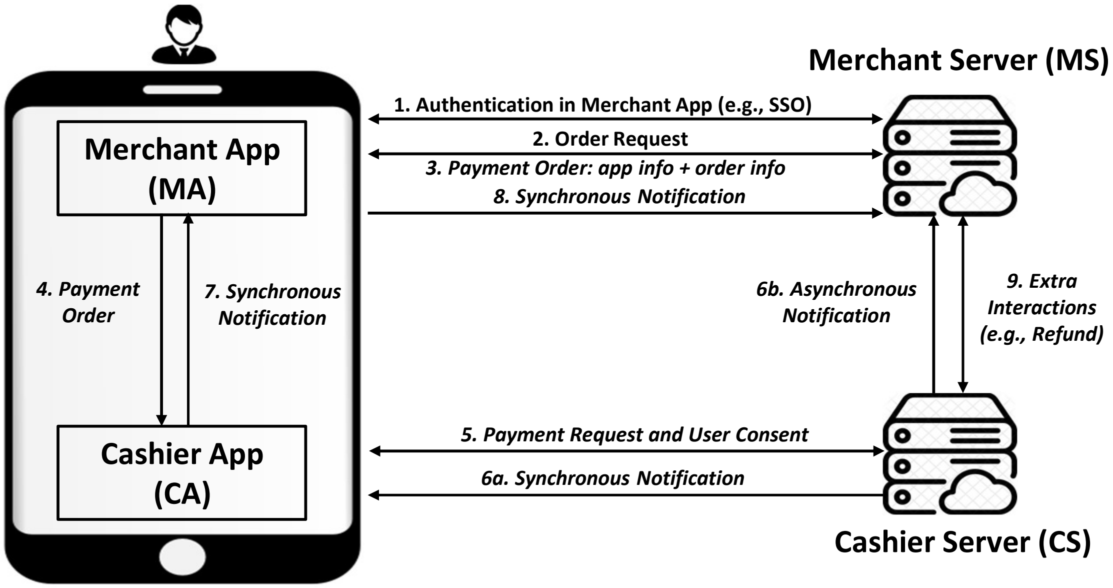
Cashier Server (CS)

Outline

- Introduction to Third-Party Payment Service for Mobile Apps
- Overview on Payment Credentials
- Leaking Sources of Payment Credentials
- Exploiting the Leaked Payment Credentials
- Automatic Mining for Payment Credentials Leaked in the Wild
- Resolving the Leaking App Identity
- Suggested Fixes

Overview on Payment Credentials

- Most payment-related messages, in italic, are protected cryptographically.
- The related payment credentials are defined by the Cashiers without a standard.
- We study four first-tier Cashiers in this work and anonymize them here due to their request.


Merchant Server (MS)
Merchant App (MA)
Cashier App (CA)
Cashier Server (CS)
1. Authentication in Merchant App (e.g., SSO)
2. Order Request
3. Payment Order: app info + order info
8. Synchronous Notification
4. Payment Order
7. Synchronous Notification
6b. Asynchronous Notification
9. Extra Interactions (e.g., Refund)
5. Payment Request and User Consent
6a. Synchronous Notification

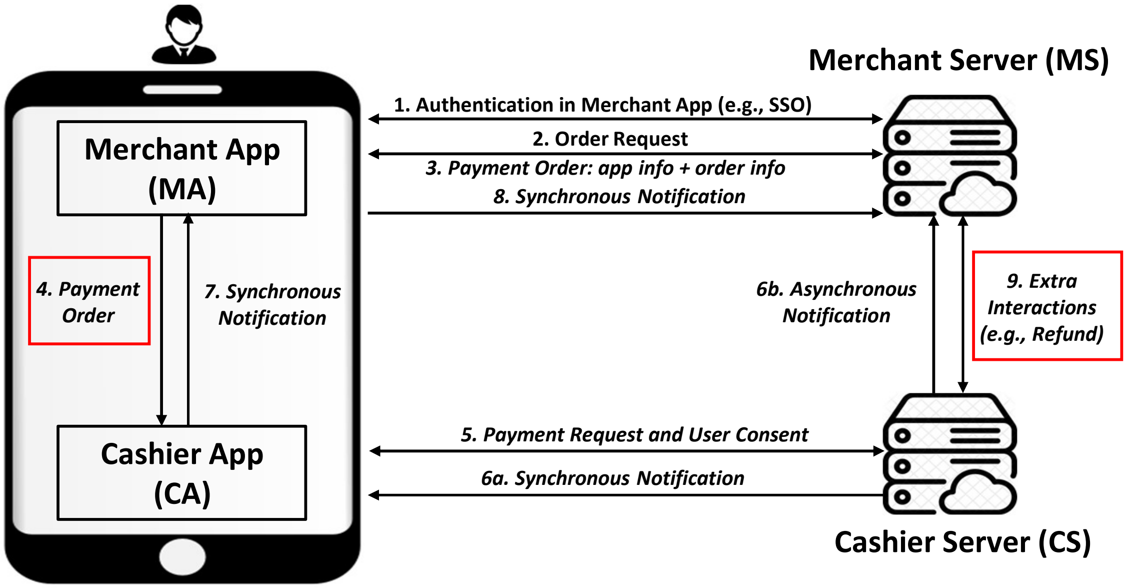
Payment Credentials: Payment Key

- The Merchants can choose either digital signature or HMAC to secure the messages.
- The security setting of the payment keys differs among the Cashiers.

Cashier	Payment Key	Usage	Assigned by the Cashier?	Shared Cashier's Public Key
Cashier1	Secret Key	HMAC	✓	n/a
	RSA Key	Digital Signature	✗	✓
	RSA' Key	Digital Signature	✗	✗
Cashier2	Secret Key	HMAC	✗	n/a
Cashier3	Secret Key	HMAC	✓	n/a
	PFX Certificate	Digital Signature	✓	✓
Cashier4	Secret Key	HMAC	✓	n/a

Payment Credentials: Other Credentials

- Android Signing Key (in *Cashier2* and *Cashier4*)
- Client Certificate (in *Cashier2*)

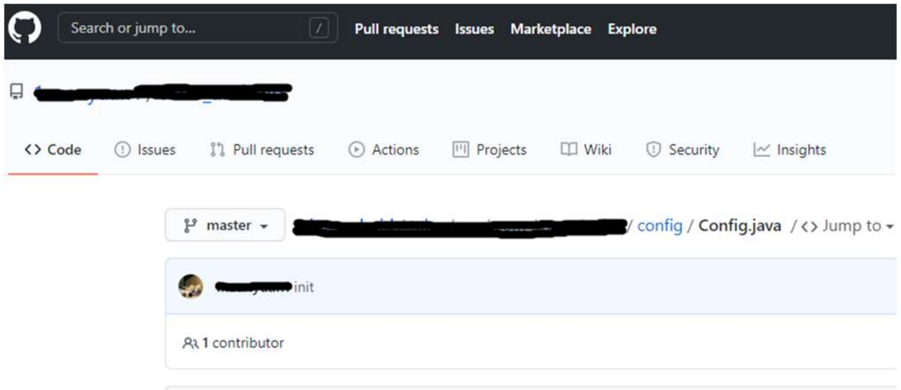

Merchant Server (MS)
Merchant App (MA)
Cashier App (CA)
1. Authentication in Merchant App (e.g., SSO)
2. Order Request
3. Payment Order: app info + order info
8. Synchronous Notification
4. Payment Order
7. Synchronous Notification
6b. Asynchronous Notification
9. Extra Interactions (e.g., Refund)
5. Payment Request and User Consent
6a. Synchronous Notification
Cashier Server (CS)

Outline

- Introduction to Third-Party Payment Service for Mobile Apps
- Overview on Payment Credentials
- Leaking Sources of Payment Credentials
- Exploiting the Leaked Payment Credentials
- Automatic Mining for Payment Credentials Leaked in the Wild
- Resolving the Leaking App Identity
- Suggested Fixes

Leaking Sources of Payment Credentials

- Public Git Repositories: (1) GitHub (2) GitLab

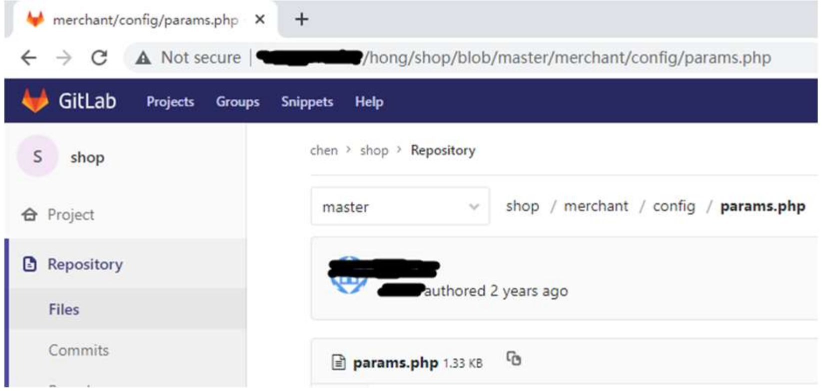


[redacted]

[redacted] / config / Config.java

[redacted] init

1 contributor



Not secure | [redacted]/hong/shop/blob/master/merchant/config/params.php

chen › shop › Repository

master

shop / merchant / config / params.php

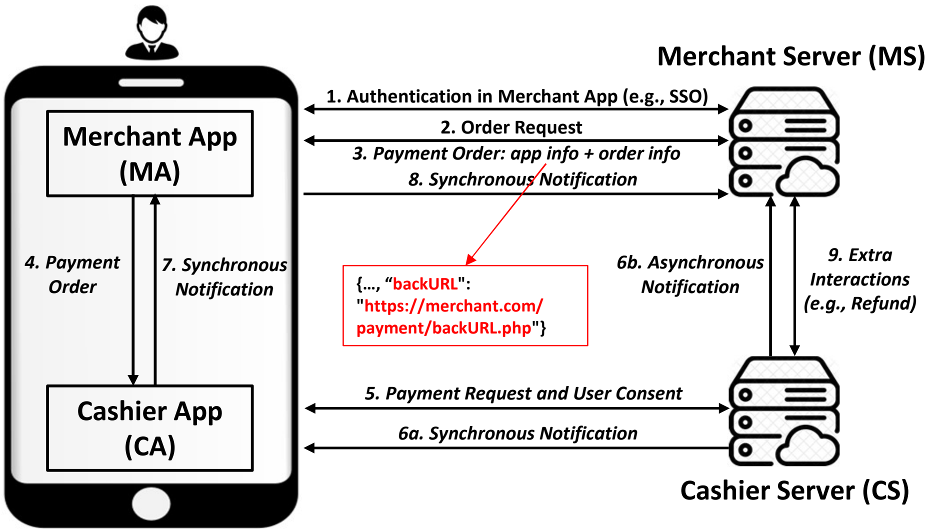
[redacted] authored 2 years ago

params.php 1.33 KB

- Mobile Apps (e.g., Android APKs): Leaked credentials may only exist in old app versions.

Leaking Sources of Payment Credentials

- Merchant Server: Caused by (1) flawed backend SDKs by the Cashier
  (2) insecure access control setting by the Merchant
- The attacker can infer the URL endpoint of the payment credentials from backURL, e.g., https://merchant.com/payment/backURL.php => https://merchant.com/payment/cred/privateKey.pem


Merchant Server (MS)
Merchant App (MA)
Cashier App (CA)
Cashier Server (CS)
1. Authentication in Merchant App (e.g., SSO)
2. Order Request
3. Payment Order: app info + order info
8. Synchronous Notification
4. Payment Order
7. Synchronous Notification
{..., "backURL": "https://merchant.com/payment/backURL.php"}
6b. Asynchronous Notification
9. Extra Interactions (e.g., Refund)
5. Payment Request and User Consent
6a. Synchronous Notification

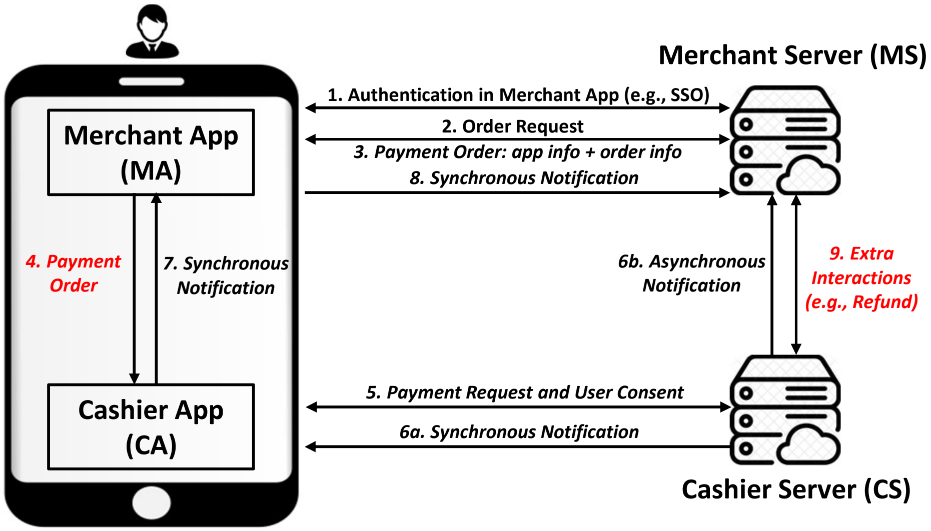
```
backend_php_sdk
├ ...
├cred
│ ├privateKey.pem
│ └cashierKey.pem
├paymentConfig.php
├backURL.php
└ ...
```

Outline

- Introduction to Third-Party Payment Service for Mobile Apps
- Overview on Payment Credentials
- Leaking Sources of Payment Credentials
- Exploiting the Leaked Payment Credentials
- Automatic Mining for Payment Credentials Leaked in the Wild
- Resolving the Leaking App Identity
- Suggested Fixes

Exploits with Leaked Payment Credentials

- Merchant Impersonation: (1) transaction record (2) refund (3) money transfer*


Merchant Server (MS)
Merchant App (MA)
Cashier App (CA)
Cashier Server (CS)
1. Authentication in Merchant App (e.g., SSO)
2. Order Request
3. Payment Order: app info + order info
8. Synchronous Notification
4. Payment Order
7. Synchronous Notification
6b. Asynchronous Notification
9. Extra Interactions (e.g., Refund)
5. Payment Request and User Consent
6a. Synchronous Notification

- Android Package Signature Forgery: Overall, 400+ valid signing keys are detected.

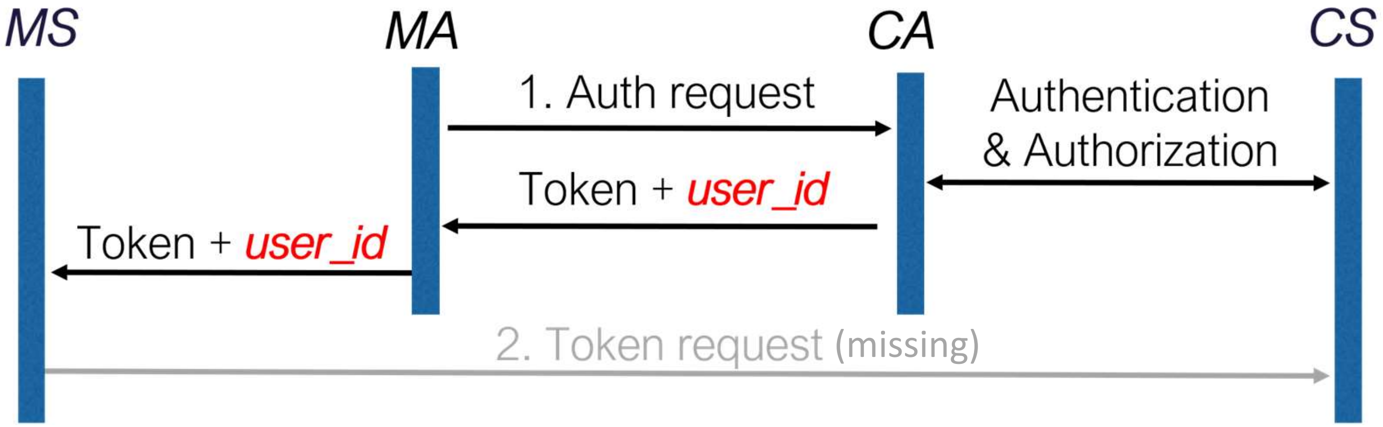
Exploits with Leaked Payment Credentials

- Backward SSO Attack

a) Some Cashiers provide third-party SSO service but fail to isolate their services, e.g., shared user ids.

b) With Profile Exploit [1], the attacker can then hijack the victim account in the Merchant Apps.

c) Re-usage of payment key as the SSO secret



Ref: [1] Ronghai Yang, Wing Cheong Lau, Tianyu Liu, “Signing into One Billion Mobile App Accounts Effortlessly with OAuth2.0,” Black Hat Europe 16

Exploits with Leaked Payment Credentials

- Cross-App Notification Forgery

a) In the case of digital signature, the public key of the Cashier are usually shared across Merchants.

b) Some Merchant Servers do not verify payment notifications properly and overlook the app identifiers.

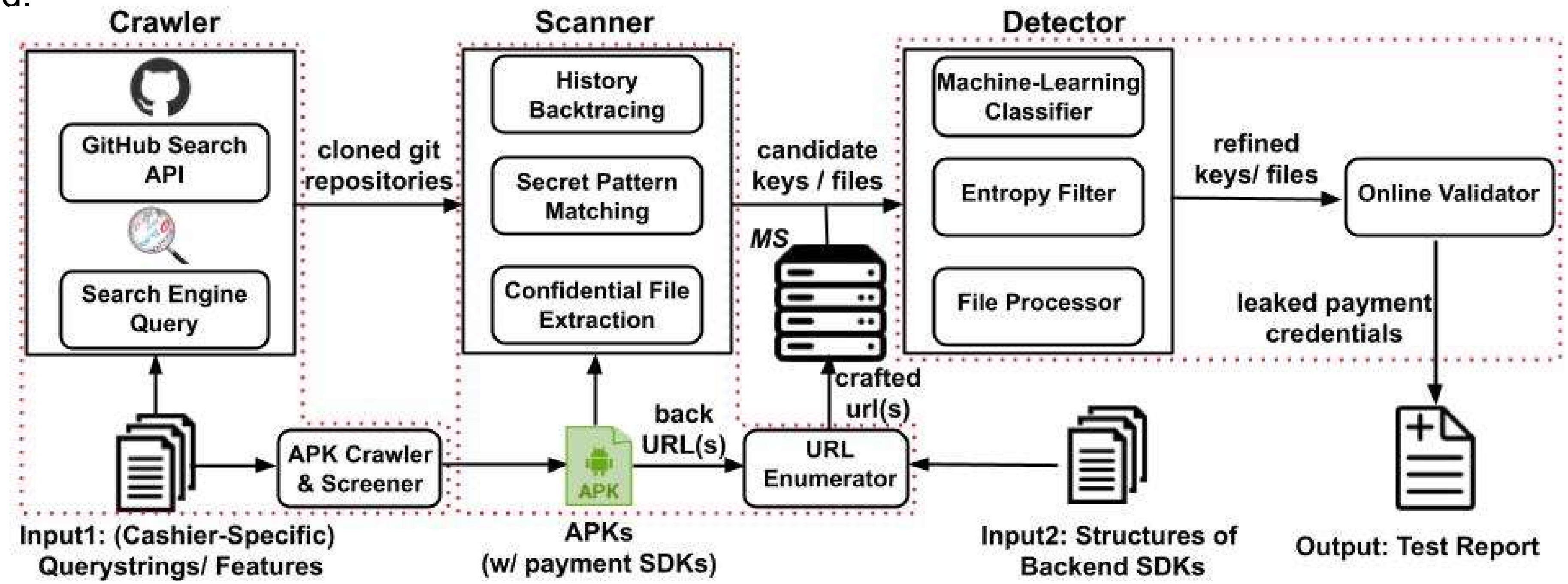
c) The attacker may use leaked payment keys to craft valid notifications to cheat another Merchant App.

Outline

- Introduction to Third-Party Payment Service for Mobile Apps
- Overview on Payment Credentials
- Leaking Sources of Payment Credentials
- Exploiting the Leaked Payment Credentials
- Automatic Mining for Payment Credentials Leaked in the Wild
- Resolving the Leaking App Identity
- Suggested Fixes

Automatic Mining Tool

- We develop an automatic tool to enable large-scale mining for payment credentials leaked in the wild.



Automatic Mining Tool: Crawler

- We use GitHub Search API and Search Engines (i.e., Google, Bing, Yahoo, and Baidu) to collect public GitHub and GitLab repositories.
- We summarize and construct the query strings based on the invariants in the integration of payment service.

Type	Cashier	Sample	Illustration
Data	Cashier3	CUYx***	Part of Cashier's public key
Code	Cashier2	**PayConfig	Classes defined in backend SDK
File	Cashier1	**PayPlugin.a	SDK files for iOS apps

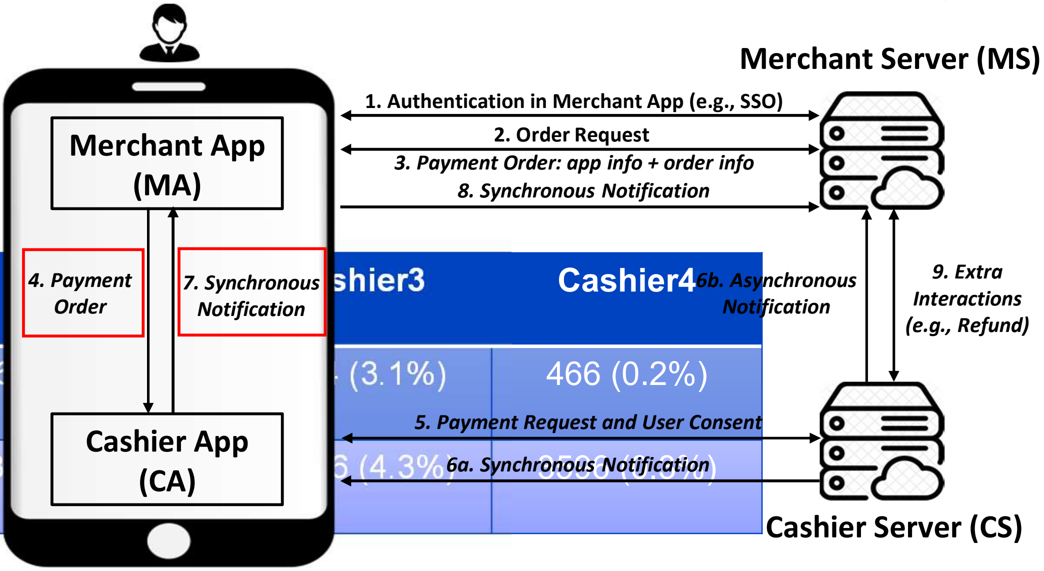
Automatic Mining Tool: Crawler

- We crawl Android APKs, including older versions, from third-party app markets to set up a full-scale database.
- The tool preprocesses the collected APKs to identify the payment-related ones with the following two heuristics.

a) Frontend SDK from the Cashiers

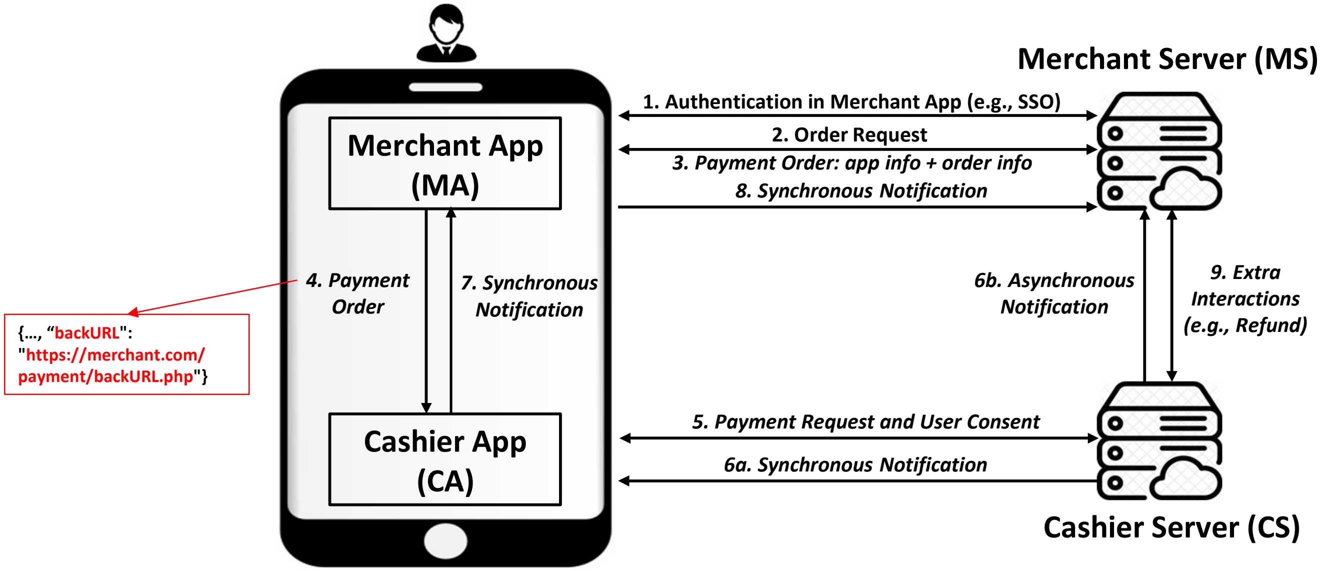
b) Activities Registered in AndroidManifest.xml

	#Total	Cashier1		shier3	Cashier4
#Android App	233550	29269 (12.5%)	46	(3.1%)	466 (0.2%)
#App Version	1240961	182124 (14.7%)	23	6 (4.3%)	3596 (0.3%)


Merchant Server (MS)
1. Authentication in Merchant App (e.g., SSO)
2. Order Request
3. Payment Order: app info + order info
8. Synchronous Notification
Merchant App (MA)
4. Payment Order
7. Synchronous Notification
Cashier App (CA)
6b. Asynchronous Notification
9. Extra Interactions (e.g., Refund)
5. Payment Request and User Consent
6a. Synchronous Notification
Cashier Server (CS)

Automatic Mining Tool: Scanner

- The module recognizes all potential payment credentials.
- Whitebox Scanning: (1) back tracing the history (2) text pattern matching (3) file format filtering
- Blackbox Scanning: Some Merchant Apps embed the value of their backURLs. Thus, our tool can extract them from the APKs and probe the Merchant Servers behind.


Merchant Server (MS)
Merchant App (MA)
1. Authentication in Merchant App (e.g., SSO)
2. Order Request
3. Payment Order: app info + order info
8. Synchronous Notification
4. Payment Order
7. Synchronous Notification
6b. Asynchronous Notification
9. Extra Interactions (e.g., Refund)
{…, "backURL": "https://merchant.com/payment/backURL.php"}
Cashier App (CA)
5. Payment Request and User Consent
6a. Synchronous Notification
Cashier Server (CS)

Automatic Mining Tool: Detector

- Some output from the Scanner is false positive caused by stored files, e.g., system logs.

a) We deploy a classifier to distinguish configuration files and stored files.

- Some payment keys are generated by the Cashiers and have similar Shannon entropy.

a) We use an entropy filter to remove the false positive.

- The credential files, i.e., PFX certificate, client certificate, and Android signing keys, are password-guarded.

a) Our tool takes different strategies to crack these credential files.

b) The activeness of the given Android signing key can be checked here by comparing the hash value of the related APK, i.e., Android Package Name => Android APK => Hash Comparison.

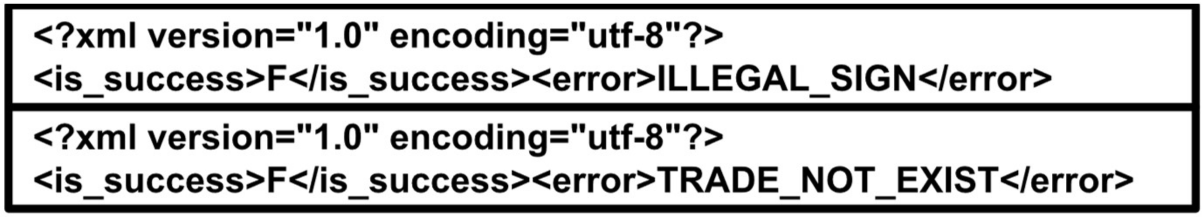
Automatic Mining Tool: Detector

- Further, we use an online method to validate the refined payment credentials.

a) Using the potential payment key, the tool prepares an order query request with invalid parameters.

b) The validity of input credentials can be inferred from the error code from the Cashier.

c) We properly control the intervals between the prepared requests. The average testing time is roughly 300 seconds.



```
<?xml version="1.0" encoding="utf-8"?>
<is_success>F</is_success><error>ILLEGAL_SIGN</error>
```

```
<?xml version="1.0" encoding="utf-8"?>
<is_success>F</is_success><error>TRADE_NOT_EXIST</error>
```

Test Result

- Overall, around 20,000 unique payment credentials are detected by our tool.

Cashier	Cashier1			Cashier2			Cashier3		Cashier4	
Source \ Credential	Secret Key	RSA Key	RSA' Key	Secret Key	Client Cert	Android Key	Secret Key	PFX Cert	Secret Key	Android Key
GitHub Repo	900	1518	1737	6651	3131	491	0	188	25	1
GitLab Repo	9	20	20	57	31	1	0	1	0	0
Android APK	75	1950	354	2567	3	0	2	0	10	0
Merchant Server	0	44	0	0	11	0	0	2	0	0
Overall	975	3332	2085	9093	3170	492	2	189	34	1

Test Result: Public Git Repositories

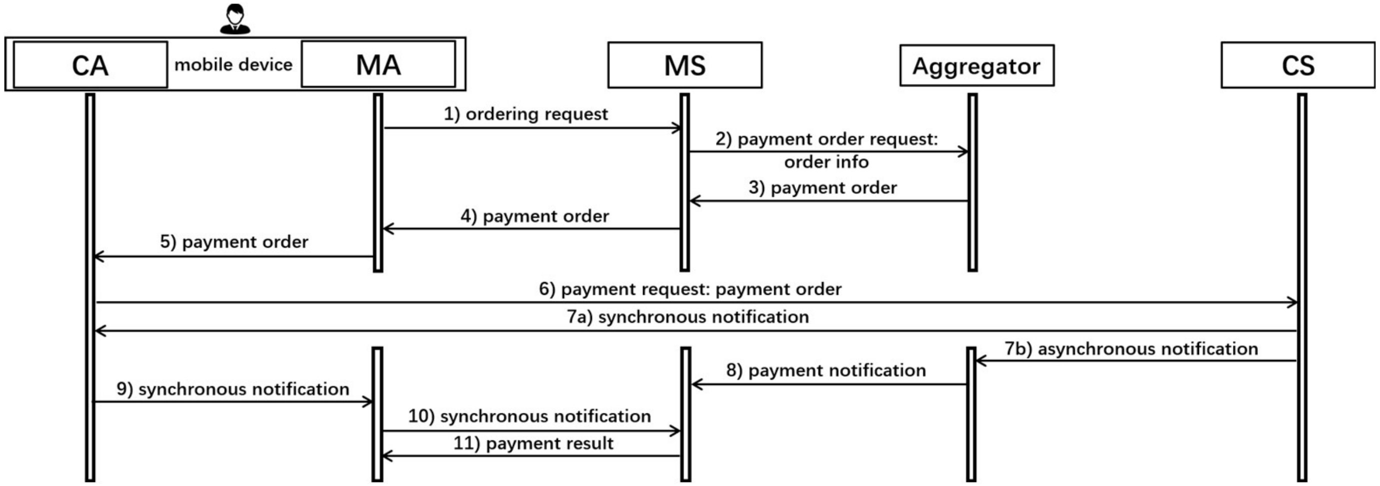
- Overall, we collect and test more than 140,000 public git repositories from various sources.
- 10.3% of these git repositories leak valid payment credentials
- 7.8% of the detected payment credentials only exist in history.
- It takes around 51 days for the developers to take the wrong fix, i.e., pushing new git commits to hide the leaked credentials.
- 712 payment credentials have been detected from the iOS-related GitHub repositories.
- Most of the public GitLab repositories with leaks belong to some outsourcing companies.

Test Result: Android APKs

- 4958 payment keys and 3 client certificates are detected from 7492 Android apps.
- 31.9% of the detected keys only exist in old app versions.
- Other finding:

a) Re-usage of Official Android Demo

b) Credential Leaks by the Payment Aggregator


CA
mobile device
MA
MS
Aggregator
CS
1) ordering request
2) payment order request: order info
3) payment order
4) payment order
5) payment order
6) payment request: payment order
7a) synchronous notification
7b) asynchronous notification
8) payment notification
9) synchronous notification
10) synchronous notification
11) payment result

Test Result: Merchant Servers

- Our tool identifies around 800 Merchant Servers from their frontend apps.
- 57 of these tested servers (7.1%) use flawed SDKs and fail to protect their credentials.
- From the GitHub result, we have manually found that iOS apps can make the same mistake.

Responsible Disclosure & Longitudinal Study

- After the initial testing, we reported over 3,000 payment keys to two of the studied Cashiers.
- We conduct regular monitoring of the related GitHub repositories to study the responses from the Merchants.
- There are 5 types of responses (12 months after our report):

Cashier	Cashier1	Cashier2
#Total	718	3662
#Updating the Key	255 (35.5%)	443 (12.1%)
#Hiding the GitHub Repositories	146 (20.3%)	651 (17.8%)
#Deleting Related Git Commits	65 (9.1%)	198 (5.4%)
#Pushing New Git Commits	3 (0.4%)	24 (0.6%)
#No Actions	249 (34.7%)	2346 (64.1%)

Outline

- Introduction to Third-Party Payment Service for Mobile Apps
- Overview on Payment Credentials
- Leaking Sources of Payment Credentials
- Exploiting the Leaked Payment Credentials
- Automatic Mining for Payment Credentials Leaked in the Wild
- Resolving the Leaking App Identity
- Suggested Fixes

Resolving the Leaking App Identity

- Some credentials are detected from GitHub and we use three approaches to identify the related Merchant Apps.

a) Crafting Payment Request: Some payment credentials also support website payment.

b) Parsing Client Certificate: The Merchant App information is available in the file attributes after unlocking the client certificate.

c) Hooking the Cashier App: The Cashier App extracts the information of the Merchant App, e.g., Android Package Name, from its server in each session.

Outline

- Introduction to Third-Party Payment Service for Mobile Apps
- Overview on Payment Credentials
- Leaking Sources of Payment Credentials
- Exploiting the Leaked Payment Credentials
- Automatic Mining for Payment Credentials Leaked in the Wild
- Resolving the Leaking App Identity
- Suggested Fixes

Suggested Fixes

- We would like to give the following suggestions to mitigate the credential leak issue:

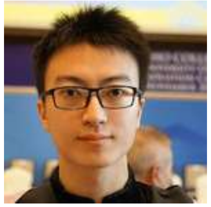
a) The Cashiers should explicitly warn their Merchants about the serious consequences of payment credential leaks.

b) The Cashiers should review their services and fix the insecure implementations, e.g., flawed SDKs and misleading frontend demo projects.

c) The Cashiers should proactively monitor and revoke the leaked payment credentials.

d) The Merchants should periodically change their credentials.

Thanks!
Q&A



Shangcheng Shi
ss016@ie.cuhk.edu.hk

Xianbo Wang
xianbo@ie.cuhk.edu.hk

Wing Cheong Lau
wclau@ie.cuhk.edu.hk