Webviews – Unsafe URI Loading

Archived: 2026-04-29 02:13:15 UTC

Webviews – Unsafe URI Loading Stay organized with collections Save and categorize content based on your preferences.

OWASP category: MASVS-CODE: Code Quality

Overview

An unsafe URI Loading occurs when an Android application fails to correctly evaluate the validity of a URI before loading it into a WebView.

The underlying reason behind this type of vulnerability is that a URI consists of multiple parts, of which, at a minimum, the scheme and the host (of the authority part) must be verified (e.g. allowlisted) before the URI gets loaded to a WebView or is used internally by the application.

The most common mistakes include:

- Checking the host but not the scheme, allowing an attacker to use schemes like `http://`, `content://` or `javascript://` with an authenticated host.
- Failing to parse the URI correctly, especially in cases where the URI is received as a string.
- Validating the scheme but not the host (insufficient host validation).

Regarding the last case, this usually occurs when the application needs to allow arbitrary subdomains of a primary domain. So, even if the hostname has been extracted correctly, the app uses methods such as `startsWith`, `endsWith,` or `contains` of the `java.lang.String` class to validate the presence of a primary domain in the extracted string section. Used incorrectly, these methods may lead to faulty results and force the application to improperly trust a potentially malicious host.

Impact

Depending on the context in which the host is used, the impact can vary. In cases where loading a malicious URI (i.e., one that bypassed filtering/allowlist) in a WebView could potentially lead to account takeover (e.g. using

phishing), code execution (e.g., loading malicious JavaScript), or device compromise (exploit code delivered using hyperlink).

Mitigations

When handling string URIs, it is important to parse the string as a URI and validate both the scheme and the host:

```
fun isUriTrusted(incomingUri: String, trustedHostName: String): Boolean {
   try {
       val uri = Uri.parse(incomingUri)
       return uri.scheme == "https" && uri.host == trustedHostName
   } catch (e: NullPointerException) {
       throw NullPointerException("incomingUri is null or not well-formed")
   }
}
```

```
public static boolean isUriTrusted(String incomingUri, String trustedHostName)
   throws NullPointerException {
       try {
           Uri uri = Uri.parse(incomingUri);
           return uri.getScheme().equals("https") &&
           uri.getHost().equals(trustedHostName);
       } catch (NullPointerException e) {
           throw new NullPointerException(
               "incomingUri is null or not well-formed");
       }
   }
```

For host validation, after isolating the corresponding URI part, it is important to validate it entirely (rather than partially) to accurately identify whether the host is trusted or not. When using methods like `startsWith` or `endsWith` can't be avoided, it is important to use the correct syntax and not overlook necessary characters or symbols (for example, `endsWith` requires the "`.`" dot character before the domain name for an accurate match). Neglecting these characters may lead to inaccurate matches and compromise security. Since subdomains can be infinitely nested, regular expression matching is not a recommended strategy for validating hostnames.

Contributors: Dimitrios Valsamaras and Michael Peck of Microsoft Threat Intelligence

Resources

- getHost() documentation
- getScheme() documentation

Last updated 2024-09-24 UTC.
