University of Tartu
Faculty of Science and Technology
Institute of Computer Science

CVE-2015-6790

Google Chrome security vulnerability

Oliver Vinkel

Supervisor: Meelis Roos

Tartu 2017
VOCABULARY

- **Anchor tag / anchor element** – a common HTML component used to construct a link to an object within an HTML document. Say there’s a webpage with the URL `http://example.com`, and the document contains an object identified as “target”. One way to link to that object is by including it in the address as follows: `http://example.com#target`
- **HTML Entity / Escape characters in HTML** – Codes representing another symbol in the HTML language. Commonly used to display symbols missing from the keyboard layout, to avoid a syntax conflict or to minimize the risk of a symbol being displayed incorrectly due to encoding discrepancies. E.g. the symbol € can be rewritten as `&euro` or `&#8364`;
- **CSRF (Cross Site Request Forgery)** – Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they’re currently authenticated. CSRF attacks specifically target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request. With a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker’s choosing. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application.
- **CSRF token** – a security method employed to combat CSRF attacks. It involves accompanying every page with a token containing a random “unguessable” value at the end of an URL request. This prevents attackers on malicious websites from successfully executing scripts using known request formats that would trigger a query in your browser, such as transfer requests to a bank if you’re logged into one.
- **XSS (Cross Site Scripting)** – A type of security vulnerability found in web applications. XSS enables attackers to inject client-side scripts into web pages viewed by other users. A cross-site scripting vulnerability may be used by attackers to bypass access controls such as the same-origin policy.

THE VULNERABILITY

CVS-2015-6790 was formally registered as a vulnerability after a white-hat report provided to Google by Inti De Ceukelaire in 2015.

The vulnerability could be abused to conduct a CSRF token theft. As demonstrated by Inti De Ceukelaire, an attacker can inject web scripts in the anchor tag of a given HTML document’s URL. Those scripts would be executed in the client’s browser application if the client opens a saved page containing that URL. This XSS vulnerability could’ve been abused in numerous ways. Depending on the web application, implementations could’ve had access to some protected information, as demonstrated in a proof of concept by Ceukelaire, using the following modified URL containing the symbol ‘”’ and the following script where an anchor is expected:

```
http://www.example.com/#&quot;&gt;&lt;script&gt;alert('Thanks,'+document.getElementsByClassName('_2dpb')\[0\].innerText)</script>
```

Alternatively,

```
http://www.example.com/#&lt;script&gt;alert('Thanks,'+document.getElementsByClassName('_2dpb')\[0\].innerText)</script&gt;
```
If the script is present in a saved Facebook page, then upon opening that saved page it will retrieve the name of the user and display it in a separate window.

Ceukelaire’s demonstration attached to the report:

EXPLOIT

Building on the previously given example a malicious attacker could, for example, attempt to manipulate the appearance of a website and try to redirect the user to another website. Furthermore they could potentially capture sensitive information like the CSRF token of the page you saved and use that token to bypass security protocols of that web application if you are still authorized in the background. Because an attacker can use the vulnerability to execute various web scripts unbeknownst to the victim, potential attacks are not limited to the provided examples and can employ various web script functionalities.

THE SOURCE AND CORRECTION

The vulnerability was caused by an oversight within the openTagToString() function in the file WebPageSerializerImpl.cpp. Formerly URL arguments were stored in another variable without correctly scanning and converting/escaping entities beforehand.
The following is a diff of the modifications that resolved the issue:

```diff
- if (element->hasLegalLinkAttribute(attrName)) {
  // For links start with "javascript!", we do not change it.
  if (attrValue.startsWith("javascript!", testCaseInsensitive)) {
-      result.append(attrValue);
+      result.append(m_htmlEntities.convertEntitiesInString(attrValue));
    } else {
      // Get the absolute link
      WebLocalFrameImpl* subFrame = WebLocalFrameImpl::fromFrameOwnerElement(element);
-      result.append(param->directoryName);
-      result.append('/');
-      result.append(m_localLinks.get(completeURL));
+      result.append(m_htmlEntities.convertEntitiesInString(m_localLinks.get(completeURL)));
      } else {
-        result.append(completeURL);
+        result.append(m_htmlEntities.convertEntitiesInString(completeURL));
      }
    } else {
```

Lines associated with the bug (marked in red) have been replaced. Now, instead of directly outputting the URL attribute values into the result variable, they are first processed to have entities properly escaped.

**ADDITIONAL INFORMATION**

_vuldb.com_ estimates that the exploit price (potential purchase price for an exploit) for this vulnerability would’ve been between 5000$ and 25000$.

The vulnerability was eliminated in Chrome version 47.0.2526.80.

The vulnerability has received the following CVE ratings and classifications (CVE-2015-6790):

<table>
<thead>
<tr>
<th>Feature</th>
<th>Classification</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confidentiality Impact</td>
<td>None</td>
<td>There is no impact to the confidentiality of the system.</td>
</tr>
<tr>
<td>Integrity Impact</td>
<td>Partial</td>
<td>Modification of some system files or information is possible, but the attacker does not have control over what can be modified, or the scope of what the attacker can affect is limited.</td>
</tr>
<tr>
<td>Availability Impact</td>
<td>None</td>
<td>There is no impact to the availability of the system.</td>
</tr>
<tr>
<td>Access Complexity</td>
<td>Medium</td>
<td>The access conditions are somewhat specialized. Some preconditions must be satisfied to exploit</td>
</tr>
<tr>
<td>Authentication</td>
<td>Not required</td>
<td>Authentication is not required to exploit the vulnerability.</td>
</tr>
<tr>
<td>Vulnerability type</td>
<td>CWE 20 – improper input validation</td>
<td>The product does not validate or incorrectly validates input that can affect the control flow or data flow of a program. When software fails to validate input properly, an attacker is able to craft the input in a form that is not expected by the rest of the application. This will lead to parts of the system receiving unintended input, which may result in altered control flow, arbitrary control of a resource, or arbitrary code execution.</td>
</tr>
</tbody>
</table>

CVSS Score: 4.3
RESOURCES USED

- https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)
- https://bugs.chromium.org/p/chromium/issues/detail?id=542054
- https://chromium.googlesource.com/chromium/src/+/b770d85e37b2d0e248f04cf20606a2f3871ef039%5E%21/#F0
- https://vuldb.com/?id.79780