Technical Disclosure Commons

Defensive Publications Series

January 2021

Secure Execution of JSONP Scripts within a Sandbox

Sebastian Lekies

Follow this and additional works at: https://www.tdcommons.org/dpubs_series

Recommended Citation
Lekies, Sebastian, "Secure Execution of JSONP Scripts within a Sandbox", Technical Disclosure Commons, (January 11, 2021)
https://www.tdcommons.org/dpubs_series/3952

This work is licensed under a Creative Commons Attribution 4.0 License.
This Article is brought to you for free and open access by Technical Disclosure Commons. It has been accepted for inclusion in Defensive Publications Series by an authorized administrator of Technical Disclosure Commons.

Secure Execution of JSONP Scripts within a Sandbox

ABSTRACT

The same-origin policy isolates web applications from each other based on application origin, but can prevent legitimate data exchange between two web applications with different origins. JavaScript Object Notation with Padding (JSONP) is a workaround for the same-origin policy in which data is inserted into a script file included within another application. However, this approach is vulnerable to malicious or compromised external sources. This disclosure proposes a solution that performs execution of third-party JSONP scripts within a sandbox in order to isolate them and avoid causing harm in case the code contained in the script is malicious. The sandbox is created by the use of sandboxed iframes, message channels, and the srcdoc attribute of iframes. A secure version of a library function is used to initialize a sandbox used by the parent page to execute the JSONP script and send the data back to the parent page over a bidirectional communication channel that permits message exchange.

KEYWORDS

- Same-origin policy
- Cross-site scripting (XSS)
- Sandbox
- JavaScript Object Notation with Padding (JSONP)
- Sandboxed iframe
- srcdoc attribute
- Secure code execution
- Data exchange

BACKGROUND

The same-origin policy is one of the most fundamental security mechanisms within web browsers used to guard against cross-site scripting (XSS) attacks. The same-origin policy isolates web applications from each other based on the application origin, defined as the protocol, host, and port combination used to retrieve the content of a given web application or site. While the isolation enforced by the same-origin policy is critical for security, it can prevent legitimate data exchange between two web applications with different origins.

Developers can employ various workarounds in order to support legitimate data exchange needs across applications despite the restrictions of the same-origin policy. One such workaround is known as JavaScript Object Notation with Padding (JSONP). JSONP involves inserting data into a script file that is then included within another application. Since the same-origin policy permits the inclusion of scripts across domain boundaries, inserting data within a script file allows cross-domain applications to exchange data without violating the same-origin policy.

A web application that uses JSONP typically includes a script from another source, using a library to create the callback tag for this script as: `someLibrary.jsonp(source_URL, 'callbackName')`. Since JSONP relies on the inclusion of a script file from an external source, the approach is vulnerable to malicious or compromised external sources that insert harmful code along with, or instead of, the expected data exchange.

For many years, JSONP was the only approach for cross-site data exchange needed to achieve certain desired functionality. JSONP alternatives, such as cross-origin research sharing, have emerged in recent times. However, JSONP still remains popular as the default choice to

implement cross-site data exchange. Therefore, despite its security vulnerability, JSONP is dominant on the web.

DESCRIPTION

This disclosure describes techniques to execute third-party JSONP scripts within a sandbox in order to isolate them and avoid causing harm in case the code contained in the script is malicious. The sandboxed environment is created by combining three existing browser mechanisms: sandboxed iframes, message channels, and the srcdoc attribute of iframes. To that end, the typical library function is replaced with a secure version with the same interface, i.e., `someLibrary.jsonps(source_URL, 'callbackName')`. The secure version initializes a sandbox used by the parent page to execute the JSONP script and retrieve the data. The data is passed back to the parent page over a bidirectional communication channel that permits message exchange with the sandbox.

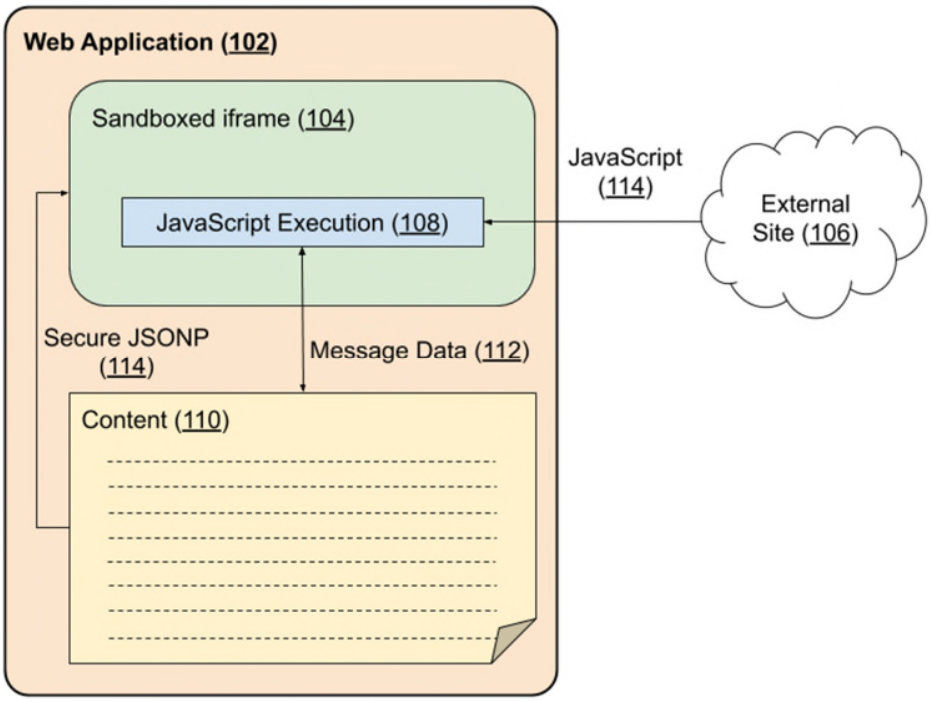

Fig. 1: Executing JSONP scripts within a secure sandbox prior to relaying the data

Fig. 1 shows an operational implementation of the techniques described in this disclosure. A web application (102) employs a secure JSONP call (114) to create a sandboxed iframe (104) to retrieve JavaScript (114) from an external site (106). The JavaScript execution (108) takes place within the sandbox, thus isolating the parent web application from potential harm in case the script contains malicious code. The results of the script execution are passed to the parent web application as data in the form of non-executable messages (112). The data can then be incorporated within the application content (110) or application backend.

Pseudocode demonstrating an operational implementation of the techniques described in this disclosure is provided at the end of this document. Invoking the secure version of the library function triggers the following operations to achieve sandboxed execution of the JSONP script:

1. **Initialize the sandbox:** A sandbox is initialized in the form of a sandboxed iframe (`var sandboxFrame = document.createElement("iframe")`). The sandboxed iframe is then explicitly set to permit script execution, which is forbidden by default (`sandboxFrame.sandbox = "allow-scripts"`). Since sandboxed iframes are usually executed in the null origin, content and scripts inside the initiated sandboxed iframe do not have access to the parent page, i.e., the web application executing the JSONP script. Therefore, the srcdoc property of the sandboxed iframe is used to include a small snippet of JavaScript to handle communication with the parent page to achieve the data exchange intended by the JSONP script (`sandboxFrame.srcdoc = "JavaScript_snippet_to_excecute";`).
2. **Indicate sandbox ready status to the parent page:** Once the initialization process in the previous step is complete, the sandbox is ready for operation. The ready status of the sandbox

is indicated to the parent page by the load event of the sandboxed iframe (`sandboxFrame.onload = notifyHostingPage`).

3. **Establish communication channel with the sandbox:** Once the parent page confirms via the onload method that the sandbox is ready, data exchange between the parent page and the sandbox takes place by establishing a bidirectional communication channel (MessageChannel) that permits two iframes to communicate across origin boundaries.
4. **Execute JSONP script within the sandbox:** Once the channel is established, the sandbox is instructed to include and retrieve the JSONP script within the sandbox. Once retrieved, the JSONP is executed within the sandbox. Execution of legitimate JSONP code results in secure retrieval of the data desired by the parent page from the external source. Since the only functionality contained within the sandbox is code for communicating messages with the parent page, any JavaScript code within the sandbox can achieve nothing more than send messages to the parent page via the bidirectional communication channel (MessageChannel) established in the previous step. However, messages cannot cause harm within the parent page because they are not treated as executable code. Therefore, any potential harm from malicious code within the JSONP script is restricted to the sandbox.
5. **Retrieve data from the sandbox:** Once the included JSONP script is executed within the sandbox, the retrieved data is passed back to the parent page via the established communication channel (MessageChannel).
6. **Destroy the sandbox:** Once the data exchange is successfully completed, the sandbox is destroyed to ensure that each call executes within a separate sandbox.

In addition to protection from potentially malicious acts from untrusted third parties, the described approach hardens web applications against compromises suffered by third parties

when the parties are known and trusted. The techniques described in this disclosure leverage currently existing mechanisms, thus requiring no changes to developer practices. Apart from use within individual websites, the techniques can be integrated within libraries that use JSONP functionality, hardening the security of these libraries. The operations described above can operate at scale and makes it unnecessary to review the security of every single case of JSONP use.

Implementation of the sandboxing techniques described in this disclosure in JavaScript libraries with JSONP functionality can improve the security of a heavily used web mechanism, thus contributing to enhancing online security at scale across the web. Any entity that operates or develops web applications that utilize JSONP can make use of the described techniques.

CONCLUSION

The same-origin policy isolates web applications from each other based on application origin, but can prevent legitimate data exchange between two web applications with different origins. JSONP (JavaScript Object Notation with Padding) is a workaround for the same-origin policy in which data is inserted into a script file included within another application. However, this approach is vulnerable to malicious or compromised external sources. This disclosure proposes a solution that performs execution of third-party JSONP scripts within a sandbox in order to isolate them and avoid causing harm in case the code contained in the script is malicious. The sandbox is created by the use of sandboxed iframes, message channels, and the srcdoc attribute of iframes. A secure version of a library function is used to initialize a sandbox used by the parent page to execute the JSONP script and send the data back to the parent page over a bidirectional communication channel that permits message exchange.

REFERENCES

1. Channel Messaging API. https://developer.mozilla.org/en-US/docs/Web/API/Channel_Messaging_API
2. Play safely in sandboxed IFrames. https://www.html5rocks.com/en/tutorials/security/sandboxed-iframes/

Pseudocode showing JSONP execution within a secure sandbox iframe

```
(function() {
  "use strict";
  function sandboxedScript() {
    function handler(messageEvent) {
      if (messageEvent.ports.length > 0) {
        jsonp(messageEvent.data.url, messageEvent.data.callbackName,
messageEvent.ports[0]);
      }
    }
    function setupMessaging() {
      window.addEventListener('message', handler);
    }
    function jsonp(url, callbackName, messagePort) {
      var script = document.createElement("script");
      script.src = url;

## STEP 4: Execute JSONP script within the sandbox ##
      window[callbackName] = function(data) {
        messagePort.postMessage(data);
      };
      document.body.appendChild(script);
      document.body.removeChild(script);
    }
    setupMessaging();
  }

## STEP 1: Initialize the sandbox ##
  window.jsonps = function(url, callbackName) {
    var sandboxFrame = document.createElement("iframe");
    sandboxFrame.sandbox = "allow-scripts";
    sandboxFrame.srcdoc = "<script>(" + sandboxedScript.toString() +
")();</script>";

## STEP 2: Indicate sandbox ready status to the parent page ##
    sandboxFrame.onload = requestJSONPData;

    function requestJSONPData() {
      var messageHandler = function(message) {
        window[callbackName](message.data);

## STEP 6: Destroy the sandbox ##
        document.body.removeChild(sandboxFrame);
      };

## STEP 3: Establish communication channel with the sandbox ##
      var mc = new MessageChannel();
      mc.port1.onmessage = messageHandler;

## STEP 5: Retrieve data from the sandbox ##
      sandboxFrame.contentWindow.postMessage({"url": url, "callbackName":
callbackName}, "*", [mc.port2])
    };
    document.body.appendChild(sandboxFrame);
  };
})();
```