ACS - Active Content Signatures

By Eduardo Vela Nava
ACS - Active Content Signatures

How to solve XSS and mix user's HTML/JavaScript code with your content with just one script

Eduardo Vela Nava (sirdarckcat@gmail.com)

Contents:

== Introduction ==

One of the main challenges in secure web application development is how to mix user supplied
content and the server content. This, in its most basic form has created one of the most common
vulnerabilities now a days that affect most if not all websites in the web, the so called
Cross Site Scripting (XSS).

A good solution would be a technology capable of limiting the scope of XSS attacks, by a way
to tell the browser what trusted code is, and what is not. This idea has been out for ages,
but it has always required some sort of browser native support.

ACS (Active Content Signature) comes to solve this problem, by creating a JavaScript code that
will work in all browsers, without the need to install anything, and that will completely
remove from the DOM any type of dangerous code.

I will start by demonstrating how ACS solves XSS issues, without the need of the server to do
any type of parsing or filtering, and without endangering the user.

ACS will be appended at the beginning of the webpage's HTML code. As a simple external JavaScript
code, something like:

```
<html>
    <head>
        <script type="text/javascript" src="/acs.js">/*signature
here*/<plaintext/></script>
```

When this script is loaded, it will automatically stop the parsing of the rest of the HTML page.
It will, then recreate the DOM itself, taking out dangerous snippets of code (like event handlers,
or scripts), making the browser display it's content's without any danger.

Now, if the user wants to make exceptions so their own scripts are loaded, they can do so, in
a very simple way... adding a signature.

A signature is a simple definition of the active content in the page. It will instruct ACS,
which scripts are safe to run, and that any other scripts should be blocked (and optionally
notified).

This has a big similarity with Mozilla's Content Security Policy (https://wiki.mozilla.org/Security/CSP)
proposal, that will be implemented on a near future version of Firefox and Chrome (possibly
IE), but with a couple of differences.

In the first place, ACS allows you to sandbox untrusted scripts; this gives you the ability to
mix your content with user's javascript code, and still be safe. It's important to note that
this is cross browser, giving you the possibility to provide your users (as of right now) the
ability to add arbitrary HTML and javascript code to your website, and still keep it safe.

Besides this protection against XSS attacks, developers can benefit from ACS's functionality

that is provided via an API. This API, allows developers to parse HTML safely, sandbox javascript,
and provides a safe environment where user's supplied code can interact with the original
content of the page in safe way.

This program's sandbox functionality overlaps with some already existent tools that require
server side interaction, or the user to install add-ons like Silverlight or java. ACS main
advantage is that it works completely on javascript, and requires nothing in the user's client,
allowing developers to automatically support their entire previous user base instantly.

ACS will include in itself a system that will automatically create a signature for a webpage,
to help the developer on the creation of the signature.

== Signatures ==

Originally, ACS was conceived to be a new idea, however later on, it was obvious that there are
a lot of similarities with Mozilla's Content Security Policy rules, syntax and objectives.

Either way, ACS has some different and extra signatures lacking on CSP, specifically, making it
easier to migrate to it.

In general, CSP forbids the webpage to display any type of inline javascript code, forcing all
the code to be loaded as an external resource. As well, it forbids the creation of code via
javascript strings (e.g. the use of javascript: URIs, eval, Function, setTimeout, and setInterval
with string arguments), giving you the option to either change all your code to external resources,
or opt out of it, and so, be vulnerable to XSS.

ACS, besides the fact of allowing a page to select which external resources to load, it also

provides an extra set of signatures that allow web owners to keep those scripts without enabling
XSS, by specifying a hash of the code inside it.

Another difference is that ACS provides the web owner to limit the amount of javascript
code/forms/frames/etc... that are loaded in the page. This is a very important feature, since
it allows web owners that if they put all user content after their active content, without the
need of making any type hashing, the user code will be blocked (or sandboxed), and still keep
the original scripts and styles, frames, and forms allowed.

Yet another difference is the fact that ACS provides sandboxing functionality that can be enabled
from the filter itself. It allows the user to sandbox CSS (removing dangerous properties),
linearise CSS (removing the danger of ui redressing attacks), a frame breaker (allowing the
page to avoid being framed), a XSS filter (that protects the application from XSS attacks being,
reflected [also DOM level]), it allows to sandbox calls to eval() or disable it as a whole,
it allows to disable javascript: URIs (as opt in feature), enforce SSL (avoid the page to load
via HTTP), and enforce SSL cookies (delete cookies if they were leaked, and force them to be
"secure" if they aren't).

It also provides more actions to happen when an error occurs, being to warn the user, to stop
the execution of the page, to sandbox the whole page, to disable the offending element, and/or
to notify the web owner of the error. You can specify more than one action, and select which
error will trigger which action(s).

Now follows a description of the rules and events for ACS:

Occurrence Counter:

* script-count - Would specify the number of acceptable scripts inside the page
* embed-count - Would specify the number of acceptable embeds inside the page
* event-count - Would specify the number of acceptable html attribute events inside the page
* jsuri-count - Would specify the number of acceptable JS URIs (or vbscript) in href/src/etc..
arguments inside the page.
* datauri-count - Would specify the number of acceptable data URIs in href/src/etc..
arguments inside the page.
* frame-count - Would specify the number of acceptable frames inside the page
* form-count - Would specify the number of acceptable forms inside the page
* css-count - Would specify the number of acceptable CSS styles (<link> or <style>) inside
the page
* style-count - Would specify the number of acceptable styles (as attributes) inside the page
* breaker-count - Would specify the number of acceptable HTML breakers inside the document
(such as other <plaintext>, <xmp>, <xml>, open arguments, etc.. with more HTML content inside,
this is to avoid the corruption of the other counters, this defaults to 0 if not explicitly set).
* default-count – Would specify the default value for counters (except breaker count).
Can be either 0 or *, the declaration of this element is mandatory.
* dyn-occurrence – Replace occurrence with any of the previous defined occurrence definitions,
including dyn-breaker-count and dyn-default-count, if no value is specified for dyn-breaker-count
it's going to default to the remain of breaker-count , and dyn-default-count will default to
default-count value.
* all-count - Would specify the total amount of active content elements in the DOM to be allowed.

Occurrence Counter Values:

* A number.

* An *, specifying any amount of values are accepted.

Occurrences:

NOTE: The hashes should be put in the format algorithm:hash, for example: sha1:f71bc019a37e4651f471f1c13d74c
means the hash is using SHA-1.

* script-hash - A hash or list of hashes allowed as source code.
* jsuri-hash - A hash or list of hashes allowed as JS URI (or vbscript).
* datauri-hash - A hash or list of hashes allowed as data URI.
* css-hash – A hash or list of hashes allowed as css.
* style-hash - A hash or list of hashes allowed as a style.
* event-hash - A hash or list of hashes allowed as inline events in the form:
hash(tagName::event::sourcecode)
* script-src - A source to match (as a global expression, regex or exact match)[15].
* embed-src - A source to match (as a global expression, regex or exact match).[16]
* frame-src - A source to match (as a global expression, regex or exact match).[17]
* css-src - A source to match (as a global expression, regex or exact match).
* form-src - A source to match (as a global expression, regex or exact match).
* default-src – A default source to match for all elements.

Filter:

* sandbox-css: Disables all active CSS properties (expression,binding).
* linearise -css: Disables absolute and relative styles (as well as opacity and filters).
* frame-breaker: Sets whether to be allowed to be framed (accepts an optional argument:
same-origin, specifying that only can be framed if parent is same origin).
* xss-filter: Enables the XSS filter for DOM based threads.
* sandbox-eval: Sandboxes calls to eval/setTimeout/setInterval/Function with a string
argument (sandboxed code has no access to the DOM).
* disable-eval: Disables normal calls to eval/setTimeout/setInterval/Function with a
string argument.

* disable-jslocation: Disables the page to set the location to javascript/vbscript: URIs
via javascript.
* enforce-ssl: Forces the use of SSL.
* enforce-ssl-cookies: Changes all current cookies to cookies with “secure” flag.[18]

Events:

* on-error - Specifies what to do in case of any error by default, the declaration of this
element is mandatory.
* on-error[#,##,-###...] - Specifies what to do in case of an error on the #'th, or ##'th,
(or ###'th starting from the last).

Event Actions:

* warn - Just warn the user
* stop – Don’t load the page (at all)
* sandbox - Load the whole page sandboxed (no scripting)
* disable - Disable (or delete, depending) the matched element/s (on style/events/jsuri/datauri
its disable and on everything else delete).
* notify – The event is going to be notified to the web owners.

Notifications:

* notify-url – Will receive one single XHR-POST request (must be same origin) with an array (called acs-notification[]):
    - o acs-notification[][request-uri]=REQUEST_URI
    - o acs-notification[][referrer]=REFERRER
    - o acs-notification[][cookies]=COOKIES_BEFORE_HTML_WAS_LOADED
    - o acs-notification[][signature]=PAGE_SIGNATURE
    - o acs-notification[][errors][]=ARRAY_OF_ERRORS_GENERATED
        + [signature]=SIGNATURE_BROKEN
        + [element]=OFFENDING_ELEMENT_BASE64_ENCODED

== API ==

Now a days, rich content (being not only images and colored text, but also videos, audio, javascript

code and add-ons) have enabled users to enhance their user experience in many websites.

Social Networking Sites have been trying to find a way to balance between user's experience and
safety, so several solutions have been created.

These are OpenSocial (Google Gadgets) that has inherent security problems by allowing any code
at all to be executed, and limit its safety on the idea of putting the code in a frame. The other
one is Google CAJA, which includes an HTML, CSS and JavaScript sandbox, works on every browser,
but requires either java, or server interaction. Another one is Facebook FBJS, requiring also
server side parsing and prone to several vulnerabilities.

ACS wants to enable developers to use the same features ACS uses to protect the webpage, and it's
done extending its own functions to the global scope.

ACS will leak one global object, _acs that will have the following elements:

* _acs.Browser - Returns a 2 bytes string with the browser name (Opera->OP,Chrome->CH,Safari->SF,Firefox->FF,Internet Explorer->IE, KQ->Konqueror, UK->Unknown)
  followed by a dot, and the approximate browser version. This detection is done using native
  JavaScript features that allow us to do fingerprinting disregarding the version reported by window.navigator.
* _acs.isIE,_acs.isIE6,_acs.isIE7,_acs.isIE8,_acs.isFF,_acs.isFF2,_acs.isFF3,_acs.isFF35,

_acs.isSA3,_acs.isSA4,_acs.isCH3,_acs.isCH4,_acs.isOP9,_acs.isOP10,_acs.isKQ,_acs.isWK(webkit),
  _acs.isGK(gecko), with an assertion of the browser.
* _acs.JSSandbox - an instance of the default javascript Sandbox.
* _acs.CSSSandbox - an instance of the default CSS Sandbox.
* _acs.HTMLSandbox - an instance of the default HTML Sandbox.
* _acs.signature - the signature with which this page was evaluated.

== Sandboxes ==

ACS supports by default 3 sandboxes, one HTML sandbox created specifically for ACS, one CSS
sandbox that uses the native browser engine, and JSReg (http://tinyurl.com/jsreg) as a javascript sandbox.

All sandboxes (HTML/CSS/JS) have the same interface, and are used in a similar way.

When you create a new object of a Sandbox, you create a new default sandbox, which can also
be called using: _acs.JSSandbox.default(),_acs.CSSSandbox.default() or _acs.HTMLSandbox.default().

All sandboxes have the following methods:

* Sandbox::parse(code) -> will instruct the sandbox to parse the code given, the return
value is the same sandbox object. In case of an error, the function will throw an error.

* Sandbox::exec(element) -> will instruct the sandbox to execute the code given (in the
context of the optional argument 'element', being, a Node object for the CSS/HTML sandbox
specifying the rules and nodes should be applied under that object, and a Object for the
JS sandbox specifying the global context of the function), and return its value.. This
will return a StyleSheet object for the CSS parser, a Node object for the HTML parser,
and an Object for the JS sandbox.

* Sandbox::toString() -> will return the code safe to eval() or document.write() that was
sent by Sandox::parse().

== Acknowledgements ==

The author (Eduardo {sirdarckcat} Vela Nava) wants to acknowledge Alibaba Group for providing

an oportunity to develop this project, as well as the support and help on the testing of the
prototypes, comments, guidance and suggestions to Giorgio Maone creator of NoScript, and
Gareth Heyes creator of JSReg as well as WHK from elhacker.net, Luoluo, Axis, and Yunshu
from ph4nt0m, Daniel Veditz, and Sid Stamm from Mozilla, David Ross, Billy Rios,
and Eric Lawrence from Microsoft, Adam Barth, Christoph, and Chris Evans from Google,
and kuza55, David Lindsay, and Mario Heiderich from slackers.