Multiple browsers Cross-Origin-Resource-Sharing design flaw

Description:

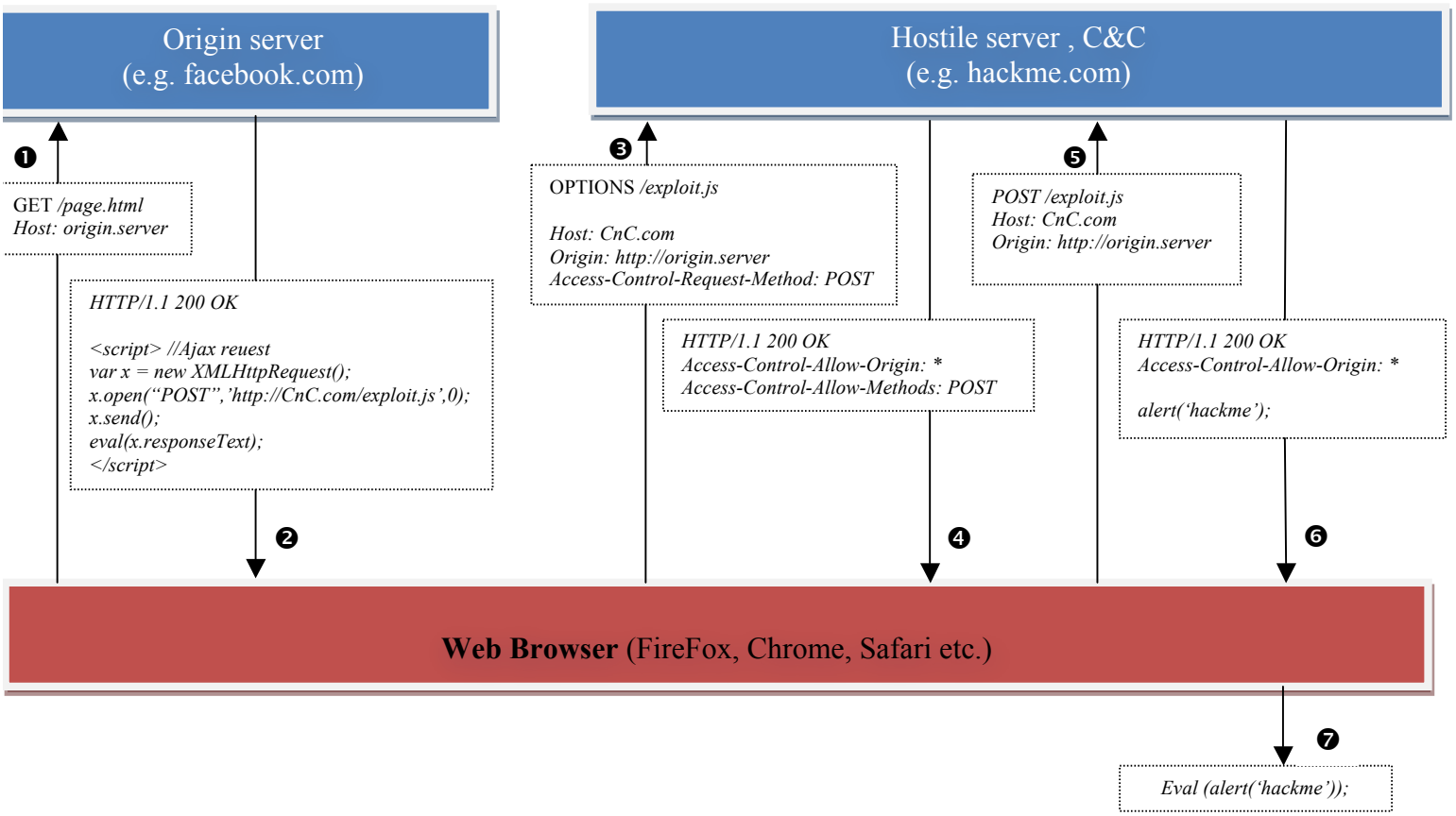
Historically, web browsers enforced a security policy called “same origin policy” on AJAX requests. One of the restrictions imposed by this policy was that AJAX request could only be send to the domain on which the webpage is hosted (origin).

The above restriction was changed having http://www.w3.org/TR/cors/

The Cross-Origin Resource Sharing standard works by adding new HTTP headers that allow servers to describe the set of origins that are permitted to read that information using a web browser.[…] For HTTP request methods that can cause side-effects on user data (in particular, for HTTP methods other than GET, or for POST usage with certain MIME types), the specification mandates that browsers "preflight" the request, soliciting supported methods from the server with an HTTP OPTIONS request header, and then, upon "approval" from the server, sending the actual request with the actual HTTP request method. (Quoted from https://developer.mozilla.org/en/HTTP_access_control)

Our research concluded that FireFox, Chrome, Safari and probably other browsers utilizing the WebKit or Gecko components with the CORS implementation of XMLHttpRequest objects are having a flaw that enables misuse of CORS for malicious purposes.

Below is a simplified diagram demonstrating an AJAX POST request to a hostile domain:



Step-by-step process description:

1. Client sends a Request message to a legit server
2. The Reply received by the client results in CORS request sent to a foreign (malicious) website. This may happen by a flaw in the legit website or inserted script by the hacker.
3. The browser sends OPTIONS request to the foreign website for checking that POST request from the origin websites are allowed (as defined on the w3c CORS spec)
4. The foreign server responds with "Access-Control-Allow-Origin: *" and "Access-Control-Allow-Methods: POST" which gives the web browser permission to carry on with the original POST request
5. The client sends the post request to the foreign (malicious) website
6. The foreign (malicious) website responds with a code to be executed by the client
7. The client executes the instructions received from the malicious server e.g. alert('hackme') having the script provided in step #2

As the diagram describes, the Origin server is not being questioned if to allow the requests to the Hostile server. It is the Hostile server to reply back to AJAX and enable the transmission.

In WebKit and Gecko browsers (e.g. Firefox 3.5 and above, Safari 4 and above, and in Chrome 4 and above) this mechanism was added to the existing XMLHttpRequest object. Microsoft Internet Explorer 8 introduced a new object, “XDomainReques” that implements this mechanism as well.

As Websites like Blogpost.com and others enables users to add their custom scripts into their custom pages, the attack scenario is very wide. Our research identified several high traffic website allowing such script to be added.

In addition, man-in-the-browser attacks can take advantage of this functionality to connect with their remote C&C servers without interruption the security policy of the browser or utilize an independent HTTP client.

Reference:

1. https://developer.mozilla.org/En/HTTP_access_control
2. http://www.w3.org/TR/cors/

Affected Vendors:

1. Mozilla (Firefox 3.5 and above) – all supported OS
2. Google (Chrome 4 and above) – all supported OS
3. Apple (Safari 4 and above) – iPad, iPhone, all supported Mac platforms

Proof of concept

We prepared a PoC demonstrating the above in action. Our PoC utilizes high traffic and legit websites allowing users to add scripts to their web pages or use AJAX for requesting remote content.

The PoC code is in alpha quality.

Our PoC can also run having the functionality available on blogspot.com where AJAX request can be created in a custom JS script.

How does it work?

1. We created a custom page on Blogpost.com (the legit site)
2. As Blogpost.com allows you to host a custom JS script, we added a script that sends AJAX requests to a 3rd domain using the RFC draft.
3. The injected custom script on Blogpost.com mimics a compromised webpage. Note that we did not inject any malicious code in the script, its just a simple AJAX
4. The injected script is using AJAX and constantly sends Requests to the C&C server asking for new commands
5. C&C replies to the injected script on Blogpost.com with a JS code abd executes it using eval()
6. The result: you can execute code from the C&C in the context of Blogpost.com – I can get your credentials or anything else I want.
7. Additional legit sites that allows users to use AJAX are vulnerable by default as well

Recommendation:

It is our recommendation that AJAX will 'consult' with the Origin server if a request to an external server (Hostile server) is permitted prior of sending the OPTIONS request.