International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

Attacks in HTML5

Kailash L. Methawani[1], Prof. Avinash P. Wadhe[2]

[1]Student of Master of Engineering in (CSE), G.H. Raisoni College of Engineering and Management, Amravati, India
[2]Assistant Professor Department of (CSE), G.H. Raisoni College of Engineering and Management, Amravati, India

Abstract: ***Now days, a web user can enjoy chatting, playing games and Internet banking from simple structured text. The browsers have changed from being a simple structured display to supporting complex multimedia application. All these applications have something in common, they can be run on different platforms and in some cases they will run offline. This is possible due to new features in latest specification HTML, specifically, drawing featured canvas. In this paper, after providing some background to HTML5, we are going to discuss attack surface and possible threats like CSRF and leveraging CORS to bypass SOP, Attacking WebSQL and client side SQL injection, ClickJacking & Phishing by mixing layers and iframe, Stealing information from Storage and Global variables, DOM injections and Hijacking with HTML 5.***

Keywords: HTML, HTML5, WebSQL, CSS3

1. Introduction

HTML5 is the latest version of HTML for websites from the World Wide Web Consortium (W3C). It was first launched in 2008, but was not actually in used till 2011. In 2011, HTML5 was released, user started using it, but the support was limited for different browsers. Now a day's almost all browsers (Firefox, Opera, Chrome, Safari) support HTML5, therefore the user can enjoy newest HTML technology at its best.

HTML5 uses CSS3 as styling sheets and is still in development. Since its release, HTML5 is continuously being development with new features; therefore it is difficult to say that that HTML5's development will end.

Now days, a web user can enjoy chatting, playing games and Internet banking from simple structured text. The browsers have changed from being a simple structured display to supporting complex multimedia application. The new markup language was developed based on pre-set standards:

- The need for external plugins (like Flash) needs to be reduced.
- New features should be based on HTML, CSS, JavaScript, Flash, etc.
- Scripting has to be replaced by more markup.
- Error handling should be easier than in previous versions.
- HTML5 should be device-independent.

2. Literature Survey

HTML known as Hypertext Mark-up Language, Created in the early 1990s, began as an application of SGML. HTML was used as standard way to describe the hypertext documents structure. The term "Hypertext" simply refers to that the text "contains links to other texts".

HTML: The Early Years

"HTML 1" was defined as a simple, tag-based syntax for explaining document structure. The earliest version didn't even consist of table elements or img. Version1.2 came with image support. Version2.0 came with slightly change in HTML grammar. Due to this user could use end tags for different elements such as p and li, these end tags were optional. Font element was added in version 3.2. User could use Java applets and the applet element in version 3.2. The most important feature of version 3.2 was that it started to support style sheets.

HTML4 came with new features. HTML 4 offered three options:

- Strict, which only allowed HTML 4 elements
- Transitional allowed for a mix of deprecated HTML 3.2 elements and HTML 4
- Frameset, which allowed multiple documents to be embedded in one using the frame element

In 1998, W3C stopped working on HTML 4 and replaced it with XHTML.

XHTML

XHTML 1.0 known as eXtensible Markup Language was created as "a reformulation of HTML 4 as an XML 1.0 application." XML.

XHTML required end tags for all non-empty elements such as p and li. Empty elements such as br and img had to be closed with a />. XHTML, for example, required lower case tags while HTML allowed upper case tags, lower case

tags, or a mix of the two. All pages had to be served as application/xml+xhtml MIME type.

HTML5

HTML5 is a simply set of new features made available for developing web applications, adding to the existing capabilities we find in HTML4. It is particularly designed to improve the language with much better support for multimedia and server communication, making a web developer's job much easier.

HTML5 is not a new version of HTML4 in comparison to when new software versions are released. It comprises an entire set of small additions to the existing web standards.

3. Proposed Work

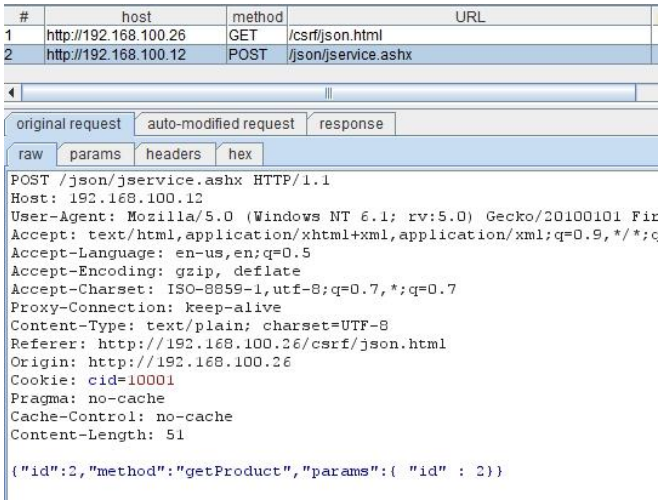
CSRF and leveraging CORS to bypass SOP

Same Origin Policy (SOP) dictates cross domain calls and allows establishment of cross domain connections. SOP bypasses allow CSRF attack vector, an attacker can inject a payload on cross domain page that initiate a request without consent or knowledge of the target user. HTML 5 is having one more policy in place called CORS (Cross Origin Resource Sharing). CORS is a "response blind" technique and controlled by extra added HTTP header "origin" and their variants but it allows request to hit the target in one way direction. Hence, it is possible to do one-way CSRF. It is possible to initiate CSRF vector using XHR-Level 2 on HTML 5 pages and can prove really lethal attack vector. XHR establishes a stealth connection and remains much hidden, XHR connection can be set using "with Credentials" as true along with POST method. It allows cookie to replay and helps in crafting successful CSRF scenario or session riding. Interestingly HTML 5 along with CORS allows performing file upload CSRF as well. It is possible to craft a JavaScript using XHR and inject JSON payload as cross domain. If server side code on JSON library is not validating the "Content-Type" then it will process the request and allows successful CSRF.

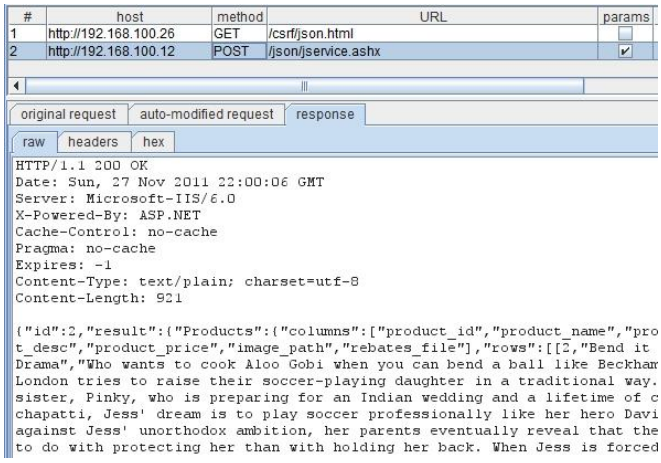
Here is a script which will do CSRF on cross domain. Here, we have "Content-Type" as "text-plain" and no new extra header added so CORS will not initiate OPTIONS to check rules on the server side and directly make POST request. At the same time we have kept credential to "true" so cookie will replay.

```
<script language="javascript"  type="text/javascript">

function getMe()
{
    var http;
    http = new XMLHttpRequest();

    http.open("POST", "http://192.168.100.12/json/jservice.ashx", true);
    http.setRequestHeader('Content-Type', 'text/plain');
    http.withCredentials= "true";
    http.onreadystatechange = function()
    {
        if (http.readyState == 4) {
                var response = http.responseText;
                document.getElementById('result').innerHTML = response;
    }

}
http.send('{\"id\":2,\"method\":\"getProduct\",\"params\":{ \"id\" : 2}}');
}

getMe();
</script>
```

Above request will cause CSRF and send following on the wire.



As we can see cookie is replayed and JSON POST has been initiated. We get following response back from application.



Application processed the request and sent JSON back. It is clear case of CSRF.

Attacking WebSQL and client side SQL injection

HTML 5 uses WebSQL as offline databases. If the application is vulnerable to XSS then an attacker can steal information from WebSQL and transfer it across domains. We have seen SQL injections on the server side but this mechanism can open up client side SQL injections. Consider the example of shopping portal storing the last 20 transactions on WebSQL being the target of an XSS attack.

HTML5 has two important data points WebSQL and Storage. They are controlled by well defined RFCs and specifications. These can be accessed by using JavaScript. Assume, we get an entry into DOM then also we are unaware with WebSQL table names and storage keys. It can be enumerated that data during pen-testing and assessments.

We need following information to extract target content for Blind SQL enumeration.

1. Table structure created on SQLite
2. Database object
3. User table on which we need to run select query

The script is given below.

```
var db_tbl;
var db_obj;
var db_tbl1;
for(a in window){
obj = window[a];
try{
if(obj.constructor.name=="Database"){
db_obj = obj;
obj.transaction(function(tx){
tx.executeSql('SELECT name FROM sqlite_master WHERE
type=\'table\'',[],function(tx,results){
table=results;
},null);
});
}
}catch(ex){}
}
if(table.rows.length>1)
db_tbl1=table.rows.item(1).name;
```

a) User will run through all objects and get object where constructor is "Database"
b) He will make Select query directly to sqlite_master database
c) He will grab 1st table leaving webkit table on 0th entry.

In this way, he will get the actual table name residing on WebSQL for this application, next he can run SQL query and loop through results.

He will get the name of the table and now we can use same database object to run the query through script.

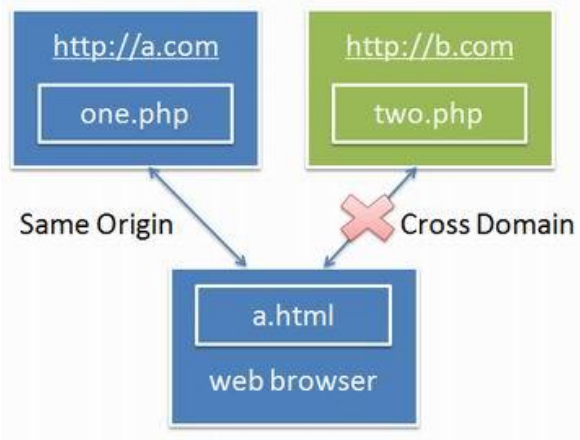
ClickJacking & Phishing by mixing layers and iframe

Click-jacking refers to the process of "stealing" clicks on your site, redirecting it to other places, either by using an XSS Vector to replace the targets of links (or whole sections of the page) or by putting your page in an iframe and placing the attacker's content over yours.

In daily life, we may visit a website, where click on event is iframed like "click here to win prizes", "Click here to kill viruses". This I nothing but the process of redirecting to other places.

Stealing information from Storage and Global variables

HTML5 uses LocalStorage, wherein a developer can create LocalStorage for the application and can store some information. This storage can be accessed from anywhere in the application. This feature of HTML 5 offers great flexibility on the client side. LocalStorage can be accessed through JavaScript.



DOM injections and Hijacking with HTML 5

Browser specifications are changed in three dimensions – HTML 5, DOM-Level 3 and XHR-Level2; each tightly integrated with the other. It is not possible to separate them while coding an application. HTML 5 applications use DOM extensively and dynamically change content via XHR calls. DOM manipulation is done by several different DOM-based calls and poor implementation allows DOM-based injections. These injections can lead to a set of possible attacks and exploits like DOM-based XSS, content extraction from DOM, variable manipulation, logical bypasses, information enumeration, etc. At the same time DOM loads different objects like Flash and Silverlight, making for interesting attack points. It is possible to hijack the entire DOM along with these objects and craft several different attack vectors as part of cross domain mechanism. DOM injections can allow add-on hacking and other browser-related hacks.