A First Approach to Counter "JavaScript Malware"*

Paper for the talks "Exploiting the Intranet with a Webpage"
and "On XSRF and Why You Should Care"

Martin Johns
Security in Distributed Systems (SVS)
University of Hamburg, Dept of Informatics
Vogt-Koelln-Str. 30, D-22527 Hamburg
johns@informatik.uni-hamburg.de

1 Introduction

> *"We're entering a time when XSS has become the new Buffer Overflow and JavaScript Malware is the new shellcode."* [3]

The term "JavaScript Maleware" was coined by J. Grossman in 2006. It describes script-code that is embedded in webpages to stealthy use the web browser as vehicle for attacks on the victim's intranet. In this paper we exemplify capabilities of such scripts and propose first defensive approaches.

1.1 Definitions

For the remainder of this paper we will use the following naming conventions:

- **Local IP addresses**: The specifier *local* is used in respect to the boundaries of the intranet that a given web browser is part of. A local IP address is therefore an address that is located inside the intranet. Such addresses are rarely accessible from the outside.
- **Local URL**: If a URL references a resource that is hosted on a local IP address, we refer to a *local URL*.
- **Implicit authentication**: With *implicit authentication* we mean authentication tracking mechanisms that are executed by the web browser and that require no further interaction after the initial authentication, e.g., cookies, client side SSL, or http authentication.

1.2 Cross Site Request Forgery

Cross Site Request Forgery (XSRF / CSRF) a.k.a. *Session Riding* is a client side attack on web applications that exploits implicit authentication mechanisms. The actual attack is executed by causing the victim's web browser to create http requests to restricted resources. This can be achieved e.g., by including hidden iframes in harmless appearing webpages. The iframe itself references a state changing URL of a remote web application, thus creating an http request (see Figure 1). As the browser provides this requests automatically

*This work was supported by the German Ministry of Economics (BMWi) as part of the project "secologic", www.secologic.org.

with authentication information, the target of the request is accessed with the privileges of the person that is currently using the attacked browser. See [14] or [1] for further details.

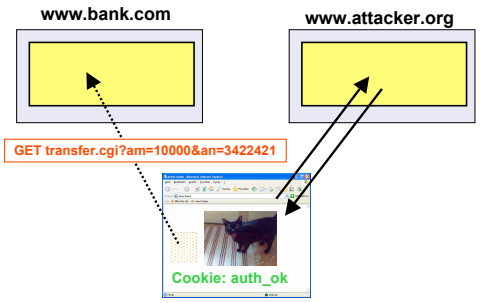


Figure 1: A XSRF attack on an online banking site

1.3 The firewall as a means of authentication

A company's firewall is often used as a means of implicit authentication (see figure 2): The intranet server are positioned behind the company's firewall and only the company's staff has access to computers inside the intranet. As the firewall blocks all outside traffic to the server, it is believed that only members of the staff can access these servers. For this reason intranet server and especially intranet web server are often not protected by specific access control mechanisms. For the same reason intranet applications often remain unpatched even though well known security problems may exist and home-grown applications are often not audited for security problems thoroughly.

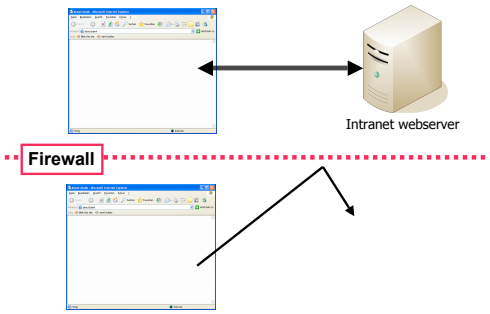


Figure 2: Protecting intranet server with a firewall

2 Attacking the intranet with JavaScript

2.1 Using a webpage to get behind the firewall

Web browsers are installed on virtually every contemporary desktop computer and only few companies refuse their employees to access the web via http. Furthermore, the evolution of active client-side technologies like JavaScript, Java or Flash has slowly but steadily transformed the web browser into a rich

application platform. Additionally all these active technologies possess certain, limited networking capabilities. By constructing a malicious webpage and succeeding to lure an unsuspecting employee of the target company to visit this page, attackers can create malicious script code that is executed within the intranet's boundaries. The following sections summarize recent findings in respect to the potential capabilities such scripts possess.

2.2 A closer look on JavaScript

For security reasons, the networking functions of client-side browser technologies are subject to major restrictions. We describe these restrictions only in respect to JavaScript, but similar concepts apply to e.g., Flash or Java applets.

Network capabilities

Foremost JavaScript is limited to http communication only. Furthermore, a script is not allowed to communicate with arbitrary http hosts. This is enforced by the *Same Origin Policy (SOP)*: The Same Origin Policy was introduced by Netscape Navigator 2.0 [13]. It defines and limits various rights of JavaScript. The origin of an element is defined by the protocol, the domain and the port that were used to access this element. The SOP is satisfied when the origin for two elements matches. All explicit network functionality of JavaScript is restricted to communication with targets that satisfy the SOP. This effectively limits a script to direct communication with its origin host.

There is only one possibility for JavaScript to create http requests to targets that do not satisfy the SOP: The script can dynamically include elements like images from foreign hosts into the document's DOM tree.

Access rights

Additionally, the SOP defines the access rights of a given script. A JavaScript is only allowed access to elements that are part of a document which has been obtained from the same origin as the JavaScript itself. In this respect, the SOP applies on a *document level*. Thus, if a JavaScript and a document share a common origin, the SOP allows the script to access all elements that are embedded in the document. Such elements could be e.g., images, stylesheets, or other scripts. These granted access rights hold even if the elements themselves where obtained from a different origin.

Example: The script `http://exa.org/s.js` is included in the document `http://exa.org/i.html`. Furthermore `i.html` contains various images from `http://picspicspics.com`. As the script and the document come from the same origin, the script has access to the properties of the images, even though their origin differs from the script's.

A loophole in the SOP

As explained above, the cross-domain networking capabilities of JavaScript are restricted by the SOP. However, this policy allows including elements from cross domain http hosts into the DOM tree of the document that contains the JavaScript. This exception in the networking policy and the fact that the SOP applies on a document level creates a loophole in SOP. In the next sections we explain how this loophole can be exploited for malicious purposes.

2.3 Portscanning the intranet

It was shown by various parties [10, 11, 4] how malicious web pages can use its capability to port-scan the local intranet. While the specific techniques vary the general approach is always the same:

1. The script constructs a local URL that contains the IP address and the port that shall be scanned.

2. Then the script includes an element in the webpage that is addressed by this URL. Such elements can be e.g., images, iframes or remote scripts.

3. Using JavaScript's time-out functions and eventhandlers like `onload` and `onerror` the script can decide whether the host exists and the given port is open: If a time-out occurs, the port is probably closed. If an `onload`- or `onerror`-event happens, the host answered with some data, indicating that the host is up and is listening on the targeted port.

To launch such an discovery attack, the malicious script needs to know the IP range of the local intranet. In case this IP range is unknown to the attacker, he can use a Java-Applet [9] to obtain the IP address of the computer that currently executes the web browser which is vehicle of the attack. Using this address the attacker's script can approximate the intranet's IP range.

Limitation: Some browsers like FireFox enforce a blacklist of forbidden ports [12] that are not allowed in URLs. In this case JavaScript's port scanning abilities are limited to ports that are not on this list. Other browsers like IE6 allow access to all ports.

2.4 Fingerprinting of intranet hosts

After determining available IP hosts and their open ports, a malicious script can try to use fingerprinting techniques to get more information about the offered services. Again the script has to work around the limitations that are posed by the *same origin policy*. For this reason the fingerprinting method resembles closely the port-scanning method that was described above [10, 4].

The basic idea of this technique is to request URLs that are characteristic for a specific device, server, or application. If such a URL exists, i.e. the request for this URL succeeds, the script has a strong indication about the technology that is hosted on the fingerprinted host. For example, the default installation of the Apache web server creates an directory called "icons" in the document root of the web server. This directory contains image files that are used by the server's directory listening functionality. If a script is able to successfully access such an image for a given IP address, it can conclude that the scanned host runs an Apache web server. The same method can be used to identify web applications, web interfaces of network devices or installed scripting languages (e.g., by accessing PHP easteregg s).

2.5 Attacking intranet servers

After discovering and fingerprinting potential victims in the intranet, the actual attack can take place. A malicious JavaScript has for example the following options:

- **Exploiting unpatched vulnerabilities**: Intranet hosts are frequently not as rigorously patched as their publicly accessible counterparts as they are believed to be protected by the firewall. Thus, there is a certain probability that comparatively old exploits may still succeed if used against an intranet host. A prerequisite for this attack is that these exploits can be executed by the means of a web browser [4].

- **Opening home networks**: The following attack scenario mostly applies to home users. Numerous end-users devices like wifi routers, firewall appliances or DSL modems employ web interfaces for configuration purposes. Not all of these web interfaces require authentication per default and even if they do, the standard passwords frequently remain unchanged as the device is only accessible from within the "trusted" home network.

  If a malicious script was able to successfully fingerprint such a device, there is a certain probability that it also might be able to send state changing requests to the device. In this case the script could

e.g., turn of the firewall that is provided by the device or configure the forwarding of certain ports to a host in the network, e.g., with the result that the old unmaintained Windows 98 box in the cellar is suddenly reachable from the internet. Thus using this method the attacker can create conditions for further attacks that are not limited to the web browser anymore [4].

- **Leaking intranet content**: The same origin policy should prevent cross domain access to content hosted on intranet web servers. In 1996 [15] showed how short lived DNS entries can be used to weaken this policy.

  Example: Attacking an intranet host located at 10.10.10.10 would roughly work like this:

  1. The victim downloads a malicious script from www.attacker.org
  2. After the script has been downloaded, the attacker modifies the DNS answer for www.attacker.org to 10.10.10.10
  3. The malicious script requests a web page from www.attacker.org (e.g via loading it into an iframe)
  4. The web browser again does a DNS lookup request for www.attacker.org, now resolving to the intranet host at 10.10.10.10
  5. The web browser assumes that the domain values of the malicious script and the intranet server match, and therefore grants the script unlimited access to the intranet server.

  To counter this attack modern browsers employ "DNS pinning": The mapping between a URL and an IP address is kept by the web browser for the entire lifetime of the browser process even if the DNS answer has already expired. While in general this is an effective countermeasure against such an attack, unfortunately there are scenarios that still allow the attack to work: Mohammad A. Haque has shown in [5] how in a multi session attack a script that was retrieved from the browser's cache still can execute this attack. Furthermore, we have recently shown [7] that current browsers are vulnerable to breaking DNS pinning by selectively refusing connections.

  Using this attack, the script can access the server's content. With this ability the script can do refined fingerprinting, leaking the content to the outside or locally analyze the content in order to find further security problems.

3 Defense strategies

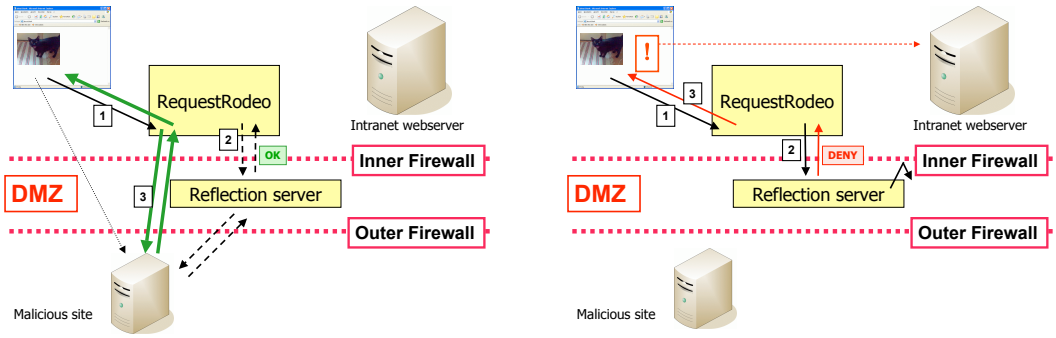
In this section we discuss possible strategies to mitigate the threats described in section 2.

3.1 Turning of active client-side technologies

The most effective solution to counter the described attacks is to turn of active client-side technologies in the web browser. To achieve the intended protection at least JavaScript, Flash and Java Applets should be disabled. As turning off JavaScript completely breaks the functionality of many modern websites, the usage of browser-tools that allow per-site control of JavaScript like the NoScript extension [6] is advisable.

3.2 Using a reflection server

On an abstract level the attacks shown in Section 2 are actually XSRF attacks which exploit the fact that the firewall is used as a means of implicit authentication. In [8] we proposed *RequestRodeo* a client side countermeasure for protection against XSRF attacks. Part of this countermeasure is defence against such



a. legal request b. prohibited request

Figure 3: Usage of a reflection server

attacks. In the following paragraphs we show how RequestRodeo can be employed to counter "JavaScript Maleware".

In [8] we introduced a classification that separated http requests into *entitled* and *unentitled*. In this context *entitled* denotes requests that originated because of the interaction with a web application. Such interactions are e.g., clicking on a hyperlink or submitting an HTML form. *Entitled* requests are therefore requests that have both their origin and their target within the same web application. Such requests cannot be the result of an attack as it was described in Section 2 and should therefore be allowed.

Accordingly, all *unentitled* requests are "cross domain requests" and therefore suspicious to be part of a XSRF attack. For this reason they should be treated with more caution. In [8] we proposed to remove all authentication information from these requests to counter potential attacks. However, in the given case the requests do not carry any authentication information. They are implicitly authenticated as their origin is inside the boundaries that are defined by the firewall. For this reason other measures have to be taken to protect local servers.

The method that is used to do the actual classification is out of scope of this paper. In [8] we introduced a client side proxy mechanism for this purpose, though ultimately we believe such a classification should be done within the web browser.

Our proposed solution introduces a *reflection server* that is positioned on the outer side of the firewall. All *unentitled* requests are first routed through this server. If such a request succeeds, we can be sure that the target of the request is reachable from outside. Such a target is therefore not specifically protected by the firewall and the request is therefore permissible.

Example: As depict in figure 3a. a web browser requests a webpage from a server that is positioned outside the local intranet. In our scenario the request is *unentitled*. It is therefore routed through the reflection server. As the reflection server can access the server unhindered, the browser is allowed to pose the request and receives the webpage's data. The delivered webpage contains a malicious script that tries to request a resource from an intranet web server (see figure 3b.). As this is a cross domain request, it also is *unentitled* and therefore routed through the reflection server as well. The reflection server is not able to successfully request the resource, as the target of the request lies inside the intranet. The reflection server therefore returns a warning message which is displayed by the web browser.

Position of the server: It is generally undesirable to route internal web traffic unprotected through an outside entity. Therefore the reflection server should be positioned between the outer and an inner firewall. This way the reflection server is treated as it is not part of the intranet while still being protected by the outer firewall. Such configurations are usually used for DMZ hosts.

3.3 Further possible protection approaches

Besides using a reflection server there are other potential approaches to protect the intranet against the attacks specified in Section 2:

- **Element-level SOP:** As shown above, the loophole which allows the attacks is that the SOP is defined on a document level. If the policy would be extended to take the origin of single elements into account, the loophole would be closed.
- **Teaching the browser the intranet's boundaries:** If the web browser would be able to differentiate between local and non-local URLs, it could prevent the attacks by implementing a simple policy: Scripts with an non-local origin are not allowed to create http requests to local resources.

Both approaches are yet to be implemented and evaluated.

4 Conclusion

We have shown that carefully crafted script code embedded in webpages is capable to bypass the same origin policy and thus can access intranet resources. For this reason simply relying on the firewall to protect intranet http server against unauthorized access is not sufficient.

Furthermore, we introduced approaches to counter website driven attacks against intranet server. Right now only one of these countermeasures is currently in development [8] and not yet ready for production use. Therefore, until usable countermeasures are available, all we can do is to conclude with some general protection advice:

- **Do not use the firewall for authentication:** All http services in the intranet should employ authentication mechanisms on their own.
- **Change all default passwords on home appliances:** Authentication is useless if the password is known.
- **Disable JavaScript:** Enable JavaScript only for trusted pages that really require JavaScript to function. This does not provide protection for the case that one of this pages was victim of an XSS [2] attack, but it reduces the attack surface significantly.

References

[1] Jesse Burns. Cross site reference forgery - an introduction to a common web application weakness. Whitepaper, `https://www.isecpartners.com/documents/XSRF_Paper.pdf`, 2005.

[2] David Endler. The evolution of cross-site scripting attacks. Whitepaper, iDefense Inc., `http://www.cgisecurity.com/lib/XSS.pdf`, May 2002.

[3] Jeremiah Grossman. Javascript malware, port scanning, and beyond. Posting to the websecurity mailing list, `http://www.webappsec.org/lists/websecurity/archive/2006-07/msg00097.html`, July 2006.

[4] Jeremiah Grossman and TC Niedzialkowski. Hacking intranet websites from the outside. Talk at Black Hat USA 2006, `http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Grossman.pdf`, August 2006.

[5] Mohammad A. Haque. Dns: Spoofing and pinning. Webpage, http://viper.haque.net/~timeless/blog/11/, (14/11/06), September 2003.

[6] InformAction. Noscript firefox extension. Software, http://www.noscript.net/whats, 2006.

[7] Martin Johns. (somewhat) breaking the same-origin policy by undermining dns-pinning. Posting to the Bug Traq Mailinglist, http://www.securityfocus.com/archive/107/443429/30/180/threaded, August 2006.

[8] Martin Johns and Justus Winter. Requestrodeo: Client side protection against session riding. In Frank Piessens, editor, *Proceedings of the OWASP Europe 2006 Conference, refereed papers track, Report CW448*, pages 5 – 17. Departement Computerwetenschappen, Katholieke Universiteit Leuven, May 2006.

[9] Lars Kindermann. My address java applet. Webpage, http://reglos.de/myaddress/MyAddress.html (11/08/06), 2003.

[10] SPI Labs. Detecting, analyzing, and exploiting intranet applications using javascript. Whitepaper, http://www.spidynamics.com/assets/documents/JSportscan.pdf, July 2006.

[11] Petko Petkov. Javascript port scanner. Website, http://www.gnucitizen.org/projects/javascript-port-scanner/, (11/08/06), August 2006.

[12] Mozilla Project. Mozilla port blocking. Webpage, http://www.mozilla.org/projects/netlib/PortBanning.html (11/13/06), 2001.

[13] Jesse Ruderman. The same origin policy. Webpage, http://www.mozilla.org/projects/security/components/same-origin.html (01/10/06), August 2001.

[14] Thomas Schreiber. Session riding - a widespread vulnerability in today's web applications. Whitepaper, SecureNet GmbH, http://www.securenet.de/papers/Session_Riding.pdf, December 2004.

[15] Princeton University Secure Internet Programming Group. Dns attack scenario. Webpage, http://www.cs.princeton.edu/sip/news/dns-scenario.html, February 1996.