Audit Based Privacy Preservation for the OpenID Authentication Protocol

Philip J. Riesch, Xiaojiang Du
Department of Computer and Information Sciences
Temple University
Philadelphia, Pennsylvania
{philip.riesch, xjdu}@temple.edu

Abstract*—This paper studies a privacy vulnerability within OpenID, a distributed single sign on protocol. An OpenID system consists of three components: User Agent (UA); Relying Party – A web application that a UA would like to authenticate with using their unique identifier; and Identity Provider – A web server that provides a globally unique identifier for the UA and validates the identity of UAs on behalf of Relying Parties. The privacy vulnerability has been identified in existing literatures. However, no effective solution has been proposed to date. In this paper, we present an effective scheme to mitigate this vulnerability. In order for OpenID to gain wider acceptance, this vulnerability must be addressed with a solution that is convenient to the users of single sign on. We propose a method for mitigating this vulnerability by creating vertical levels of trust between constituents of an OpenID network through expanding the role of OpenID Identity Providers to include auditing OpenID Relying Parties for privacy vulnerabilities. In addition, Identity Providers may keep records of audits that identify Relying Parties that do not protect the privacy of OpenID users. The primary issue with this privacy vulnerability is that it is completely transparent – it occurs without the user ever being aware that it is happening. We cannot force Relying Parties to guarantee the privacy of OpenID users, nor would we like to burden individual users with browser level solutions that are often overly technical and difficult to understand. We have designed an audit solution at the level of the Identity Provider, which can accurately inform users when Relying Parties may be sharing information with third parties, therefore giving OpenID users the ability to make a conscious choice to share that information. We have performed real network experiments to validate our scheme, and the experimental results show that our scheme is effective.

Keywords—authentication; distributed systems; OpenID; privacy; security

I. INTRODUCTION

As more computer applications and services are placed online, Single Sign On (SSO) systems are quickly becoming an emerging technology that is designed to combat the insecure and inconvenient practice of requiring users to remember dozens of permutations of user names and passwords. SSO identifies the user with a globally unique identifier that can be used to verify the identity of a user to multiple computer applications across one or many domains. This allows a user to only be required to remember one user name and password set that can validate the user across multiple computer applications, greatly reducing the inconvenience and inherent security risk of having to memorize very large sets of authentication credentials.

The most popular distributed SSO protocol in use today is OpenID. OpenID is the most well supported SSO, and its governing foundation has support from industry constituents including Google, Microsoft, Symantec, and Verizon [1]. However, OpenID has contained a known privacy vulnerability that had first been identified in 2010, and remains unresolved to this day [2]. This vulnerability can expose information that uniquely identifies an OpenID user to third parties such as computer systems used for the purposes of advertising and consumer analytics. Once a third party has obtained this information, an OpenID user may then be tracked, monitored, and followed across all computer applications that have this vulnerability exploited. This vulnerability has been difficult to resolve because its cause is neither a bug within OpenID nor is it an issue of improper implementation. Rather, this vulnerability is the unfortunate result of a design flaw of OpenID itself. Mitigation of this vulnerability requires a solution that cannot be circumvented by Relying Parties, who in some cases may be knowingly and willingly exposing this information for monetary gain. In addition, a solution should not require cumbersome and complicated tasks to be completed by the individual OpenID user.

The structure of this paper is as follows: Section II briefly outlines how OpenID works, and outlines the known privacy vulnerability that is addressed in this paper. Section III provides an analysis of the vulnerability, and utilizes data of in-the-wild examples of this exploit to identify the exact vectors that enable this vulnerability to occur. Section IV introduces an audit solution that overcomes the unique challenges of working with a distributed computer system such as OpenID. The paper is then briefly concluded in Section V.

II. OPENID AND A KNOWN PRIVACY VULNERABILITY

OpenID utilizes several key pieces of terminology to define the constituents of an OpenID network. As first defined by Uruenya [2], there are three types:

- User Agent (UA): A human user utilizing a web browser that would like to authenticate with a computer application by providing an *OpenID Identifier*, a globally unique value that is obtained from an *Identity Provider* of the user's preference.

NOTICE: All domain names have been intentionally changed to ficticious values and are intended for illustrative purposes only.

- Identity Provider (IP): A web server that provides a globally unique identifier for the UA, and validates the identity of UAs on behalf of *Relying Parties*.
- Relying Party (RP): A web application that a UA would like to authenticate with using their unique identifier. The UA is validated by using this identifer to discover and refer the UA to their preferred IP.

OpenID utilizes the Hypertext Transfer Protocol (HTTP) as its preferred transport protocol. Information is transmitted through encoding OpenID information into the parameters of the HTTP request [3], therefore allowing OpenID to require no special software in order to be properly used by UAs. While there are many individual steps involved in an OpenID transaction, including the generation and exchange of security associations through Diffie-Hellman based key exchanges [5], an OpenID transaction can be described in four very high level steps (fig. 1):

1) A UA would like to utilize a computer application that has RP software installed and configured to allow authentication through OpenID. The UA will reveal an identifier to the RP, typically with an HTML form that is provided by the RP.
2) An identifier is in the form of a valid Uniform Resource Locator (URL) [6] for an HTTP server. This URL points to a web page that provides information about the UA and its IP. On receipt of a UA's identifier, the RP will utilize the identifier to retrieve this page and discover the preferred IP of the UA.
3) Through the use of redirection capabilities built into HTTP [3], the RP will redirect the UA to their preferred IP, as discovered from the UA's identifier. The RP encodes information about the OpenID transaction directly into the redirection URL. This information can include state information, security associations, and cryptographic message signatures [4].
4) The UA authenticates with the IP. Again using HTTP redirection features, the user is redirected back to the RP with information again encoded into the URL that validates who the user is and whether they were properly authenticated.

The vulnerability in OpenID originates from the choice of how transaction information is transmitted. The choice to utilize URL parameters for the transmission of OpenID transaction data can unsafely expose unique and identifying information about the UA to third parties via a utility feature of HTTP, OpenID's underlying transport protocol [2]. The vulnerability is exploitable during the last step of the OpenID authentication process, when the UA has successfully authenticated with their IP and is redirected back to the requesting RP. If this page contains any external resources, such as scripts, stylesheets, and image files, the UA becomes exposed to the vulnerability.

HTTP contains a utility feature originally intended for web site maintenance known as a `Referer` value [3]. Web pages may contain various external resources in order to render as intended, and all modern browsers will retrieve these resources automatically. In addition, modern browsers will attach a `Referer` value to HTTP requests for these resources. This

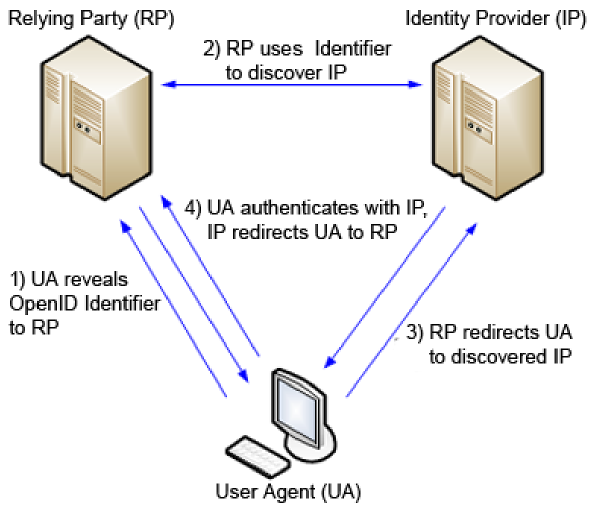

Relying Party (RP)
Identity Provider (IP)
2) RP uses Identifier to discover IP
4) UA authenticates with IP, IP redirects UA to RP
1) UA reveals OpenID Identifier to RP
3) RP redirects UA to discovered IP
User Agent (UA)

Figure 1: High level illustration of a typical OpenID transaction.

`Referer` value contains the original URL that requires the external resource, including all URL parameters and other information that was encoded into the original URL. OpenID transaction data is cryptographically signed using an HMAC based hasing algorithm [7] in order to prevent tampering, however this data is not encrypted. In consequence, unencrypted OpenID information encoded into the URL becomes exposed to third parties if the external resource exists outside of the two domains participating in the OpenID transaction.

III. VERIFYING THE VULNERABILITY

Testing for in-the-wild examples of this vulnerability is fairly simple, and indicates how prolific this vulnerability has become. An analysis can be conducted by capturing all HTTP traffic transmitted between a UA's web browser and the various HTTP servers that it connects to during the course of an OpenID transaction. UAs generate a trail of requests as they proceed through the OpenID transaction, including requests for all external resources that may need to be retrieved. By analyzing the HTTP control data within this trail of requests, we are able to determine the presence or absence of OpenID information that can uniquely identify a UA.

The benchmark as to what is considered to be uniquely identifiable OpenID information would be the presence of OpenID's `openid.identity` value. This data value contains the OpenID Identifier of the UA, a globally unique value that can be used to identify a UA across multiple RPs [4]. We can consider this information to be unique enough that a user may be tracked with little effort by any third party that obtains this information. It was observed that an OpenID transaction should ideally occur within only two unique domains: the domain of the RP, and the domain of the IP. We will refer to these as the *transacting domains*. The vulnerability is exposed when external resources are called by the UA's browser that exist outside of the transacting domains, therefore exposing the `openid.identity` value via HTTP `Referer` information subsequently transmitted by the UA's browser.

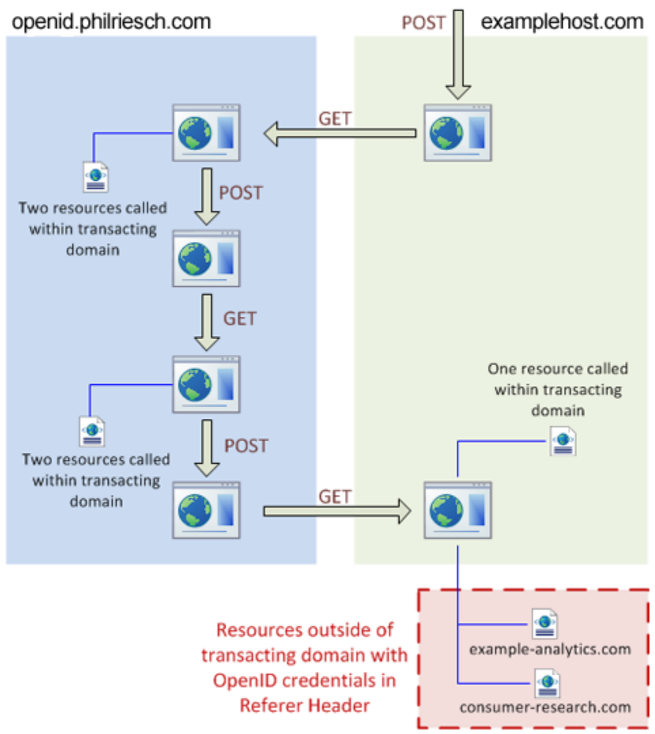

openid.philriesch.com
POST
examplehost.com
GET
POST
Two resources called within transacting domain
GET
Two resources called within transacting domain
POST
GET
One resource called within transacting domain
Resources outside of transacting domain with OpenID credentials in Referer Header
example-analytics.com
consumer-research.com

Figure 2: A visual map of an OpenID transaction generated from HTTP data captured from an in-the-wild RP. This map indicates that this particular RP is susceptible to the vulnerability and is leaking identifiable OpenID information to two separate third parties.

In order to test for in-the-wild instances of this privacy vulnerability, a series of experiments were set up and executed. An IP was established using a lightweight and open source IP server, running on a Linux based web server. A UA was provided by a commercially well known web browser that was modified to contain a module that captured and recorded all HTTP traffic that was transmitted and received by the web browser. Several major web applications were selected that had been purported to support OpenID as of 2010, the time at which this vulnerability was first identified. Access to each would be attempted using our test UA and test IP. The HTTP traffic log that is generated would then be analyzed in order to generate a visual map showing each step taken within the OpenID transaction (fig. 2,3). Any external resources that were called during this transaction would be associated to a specific step on the map. This data would indicate whether any external resources were called that do not exist within the valid space of the transacting domains.

The results of these experiments was discouraging. Of all web applications tested, about half were found to no longer support OpenID as an RP. Most of these web applications either no longer supported any form of distributed authentication, or forced the user to instead utilize a distributed login system that was proprietary to the site. Of the remaining subjects, all but one RP contained external resources being called that existed outside of the transacting domains, therefore causing our browser to transmit OpenID information within the

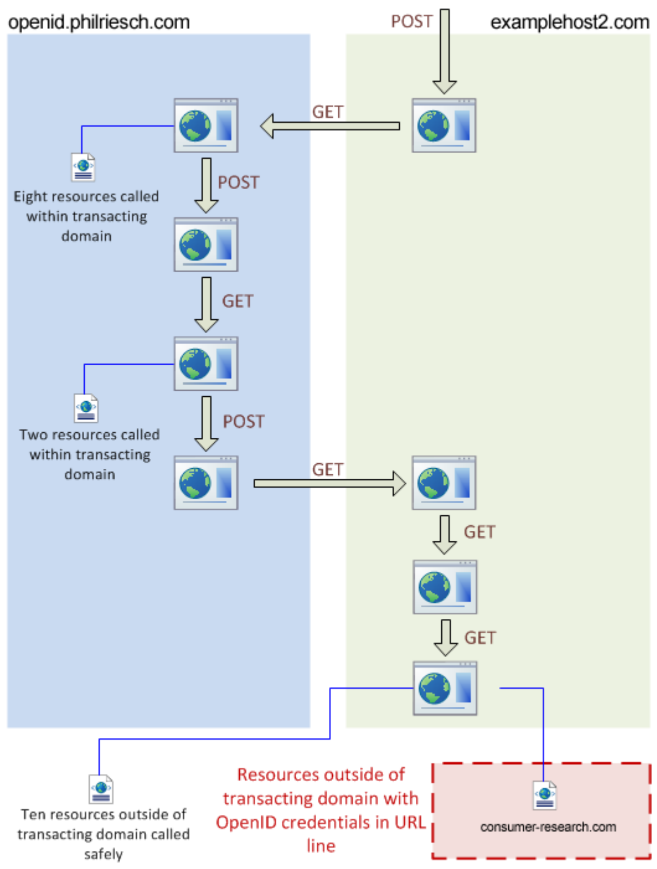

openid.philriesch.com
POST
examplehost2.com
GET
Eight resources called within transacting domain
POST
GET
Two resources called within transacting domain
POST
GET
GET
GET
Ten resources outside of transacting domain called safely
Resources outside of transacting domain with OpenID credentials in URL line
consumer-research.com

Figure 3: Another OpenID transaction map captured from an in-the-wild RP. This RP performed an additional step after receiving the UA from the IP that resulted in the UA being redirected to another page internal to the RP. It was noted that this extra redirect removed OpenID credentials from Referer data. However, this RP still called a resource with OpenID credentials encoded into the external resource request.

```
Referer: https://examplehost.com/login.pl?
[...]&openid.identity=http://openid.philri
esch.com/[...]
```

Figure 4: Raw HTTP Referer data captured from a transaction with an in-the-wild RP. Leaked OpenID credentials underlined and highlighted.

```
GET /__utm.gif?utmwv=5.2.4&utms=3&utmn=124
[...]openid_identity%3Dhttp%253A%252F
%252Fopenid.philriesch.com[...]
```

Figure 5: Raw HTTP GET data captured from a transaction with an in-the-wild RP. Leaked OpenID credentials underlined and highlighted.

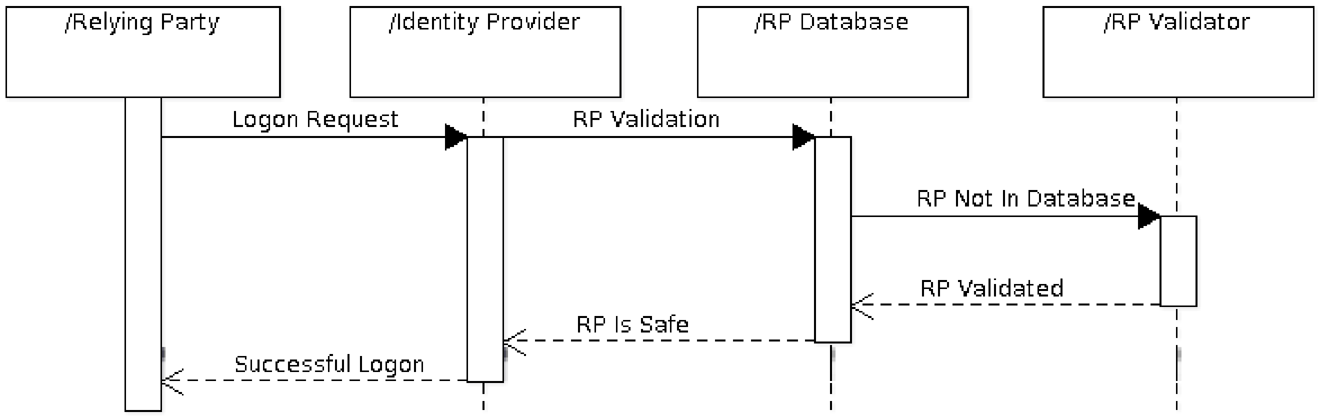


Figure 6: An IP with Relying Party Audit Engine.

`Referer` data contained in the request (fig. 2,4). It was observed that these external resources most commonly existed on domains that belonged to web analytics companies and content distribution networks.

In addition to identifying information being found encoded within `Referer` data, some RPs were found to additionally have this same information encoded directly within the URLs of the external resource request (fig. 3,5). This second vector has also been documented [2], and is also considered to be a vector of the vulnerability. In order to mitigate this vulnerability, an audit system must be devised that is able to inspect the pages returned by RPs in order to determine if a UA can become vulnerable through these two vectors for a given RP.

IV. Creating Trust Through Privacy Auditing

In order to protect OpenID networks from this vulnerability, a system of vulnerability auditing and protocol compliance checking must be developed that can foster layers of trust that extend across the entirety of an OpenID network. This system must be designed to be installed and executed on the IP, the constituent that an OpenID network should theoretically contain the fewest number of. In addition, because the IP is already trusted to safely store the authentication credentials of UAs, we may also safely assume IPs to be the most conscious about the privacy and security of OpenID users. It is not possible to design a solution that will be realistically implemented by RPs, as some RPs could possibly be sharing this information intentionally with third parties for monetary benefit; implied by the findings of our in-the-wild experiments. We cannot design a browser level solution to be implemented by the UA, as this is the largest – and often the most uninformed – group of constituents, making network-wide implementation all but impossible. An IP level solution defines the IP as an authoritative figure, allowing a realistic implementation goal.

Under this solution, an IP would keep records of all RPs that are accessed by all of the UAs managed by the IP. These records would include a history of audits performed on the RP, and whether the audit was successful or unsuccessful. Once a UA has correctly authenticated with their IP, the IP will retrieve the audit history of the RP, and determine if the most recent audit is older than a time-to-live value that is defined

```
procedure Audit
set document to Document Object Model [8] of success page
returned by RP;
set e to an array of all elements in document with tag name
"img";
for i := 0 to length of e do
        if e[i] has attributes do
                set n to value of attribute of e[i] with name
                "src";
                if n is a URL and domain of n is not within
                trust root [4] declared by RP
                        audit has failed;
        end
end
set e to an array of all elements in document with tag name
"link";
for i := 0 to length of e do
        if e[i] has attributes do
                set n to value of attribute of e[i] with name
                "href";
                if n is a URL and domain of n is not within
                trust root declared by RP
                        audit has failed;
        end
end
set e to an array of all elements in document with tag name
"script";
for i := 0 to length of e do
        if e[i] has attributes do
                set n to value of attribute of e[i] with name
                "src";
                if n is a URL and domain of n is not within
                trust root declared by RP
                        audit has failed;
        end
end
```

Algorithm 1: There are three Document Object Model elements in HTML that will invoke a browser to retrieve external resources. The IP must insure that all of these resources occur within the domain of the Trust Root claimed by the RP.

internally by the IP. If the most recent audit is beyond the age of this value, the IP will conduct an audit of the RP (fig. 6).

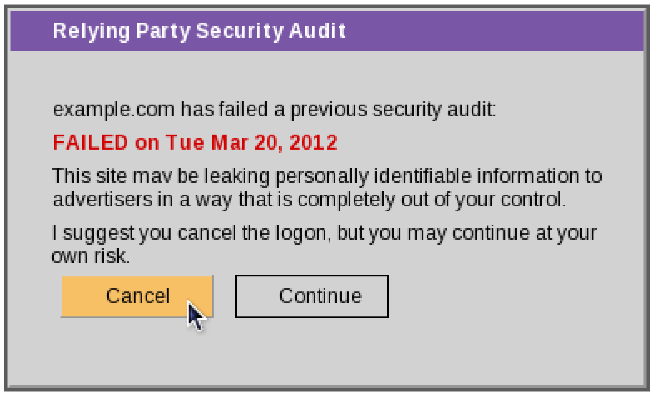
When an audit occurs, the IP will authenticate with the RP as if it were a UA, utilizing its own unique OpenID Identifier. This allows the IP to receive data that would typically be received by a UA. The IP will then process this data and search for any external resources that are called outside of the defined transacting domains: the domain of the IP and the domain of the RP as claimed by the RP's Trust Root [4] (alg. 1). If the IP is to find any resources outside of the transacting domains, it will record this audit to the audit history for this RP. The IP will then warn the UA that originally made the request about the failure. The IP will also warn any other UA that wants to authenticate with this RP (fig. 7). This will allow the IP to actively notify UAs that an RP is unsafe and may be leaking unique, identifiable information to third parties.

In order to test this solution, a proof of concept web application was developed in PHP and tested on a Linux based web server that implemented an OpenID IP and audit engine that could automatically read OpenID logon forms on RPs and perform an automated OpenID transaction with an RP. When tested on RPs that were known to be vulnerable, this audit engine was able to accurately detect any external resources that had been expected to be vulnerability vectors, as predicted through our validation experiments. In consequence, the IP was able to warn the UA when they were accessing an RP that was unsafe. This indicates that a solution that consists of IPs investigating and reporting on the safety of other participants in an OpenID network is a viable solution that helps to indicate when the vulnerability is occurring, and therefore allows for mitigation by giving UAs the choice to continue with an unsafe OpenID transaction, or abort the transaction altogether.

V. Conclusion

As more computer applications are removed from the confines of the user's personal computer and re-implemented within the Cloud and other Internet enabled environments, a widely accepted distributed authentication protocol will become necessary in order to reduce the amount of authentication credentials that must be memorized by users of these applications. OpenID is continuing to gain traction as the preferred solution that addresses and mitigates this problem in a way that is able to work within a distributed environment. Decisions made regarding the design of OpenID's data transfer protocols has enabled OpenID to function without requiring the user to modify their computers or install any software, however this decision has unfortunately created a vector for a privacy vulnerability that could be exploited for the purpose of tracking OpenID users without their knowledge or consent.

If distributed authentication protocols such as OpenID are to gain wider acceptance, privacy vulnerabilities must be addressed with a short term solution that cannot be circumvented. This goal can be accomplished by expanding the responsibilities of the IP to include guaranteeing the security of the overall network. This is achieved through auditing the network in order to accurately represent the privacy policies of web applications to OpenID users. Such a solution actively works to preserve the privacy of the user by removing the transparent transmission of identifiable information to third parties. The user is granted the knowledge that an RP may leak identifiable information. Therefore, it becomes the conscious choice of the user to continue the transaction and willingly share that information, or cancel the transaction and refuse to share information with third parties.


Relying Party Security Audit

example.com has failed a previous security audit:

FAILED on Tue Mar 20, 2012

This site mav be leaking personally identifiable information to advertisers in a way that is completely out of your control.

I suggest you cancel the logon, but you may continue at your own risk.

Figure 7: The results of our efforts: the IP presents the human user with notification of an unsafe OpenID transaction, enabling the user to make a conscious decision to continue the transaction.

Acknowledgment

This research was supported in part by the US National Science Foundation (NSF) under grants CNS-0963578, CNS-1002974, and CNS-1022552, as well as the US Army Research Office under grant W911NF-08-1-0334.

References

[1] *OpenID Foundation.* `http://openid.net/`

[2] M. Uruenya, C. Busquiel. *Analysis of a Privacy Vulnerability in the OpenID Authentication Protocol*. IEEE Multimedia, Communications, Services, and Security 2010.

[3] T. Berners-Lee, et al. *Hypertext Transfer Protocol – HTTP/1.1*. June 1999.

[4] D. Recordon, B. Fitzpatrick. *OpenID Authentication 1.1*. May 2006.

[5] R. Escola. *Diffie-Hellman Key Agreement Method*. IETF RFC 2631. June 1999.

[6] T. Berners-Lee, L. Masinter. *Uniform Resource Locators*. IETF RFC 1738. December 1994.

[7] H. Krawczyk, M. Bellare, R. Canetti. *HMAC: Keyed-Hashing for Message Authentication*. IETF RFC 2104. February 1997.

[8] W3C DOM IG. *W3C Document Object Model*. `http://www.w3.org/DOM/`. January 2005.