Bachelor thesis
Computer Science

Radboud University

DigiD vs. JavaScript: the risk of using third party JavaScript on government websites

Author:
Koen Buitenhuis
s4069471

First supervisor/assessor:
Jaap-Henk Hoepman
jhh@cs.ru.nl

Second assessor:
Erik Poll
erikpoll@cs.ru.nl

August 20, 2013

Abstract

Use of DigiD on goverment websites has grown to be more and more essential, and there has been an increasing presence of third-party JavaScript on these websites as well. This paper attempts to answer the question: Can third-party JavaScript applications compromise the DigiD authentication process on a website that uses both? This is tested by formulating several approaches by which JavaScript can attack DigiD and trying to attack from there. Results show, however, that none of the considered approaches had any real effect on DigiD. However, one approach did show risks for websites themselves.

Contents

Chapter 1

Introduction

Anyone who has used Dutch government institutions, and especially their online services, for the last decade or so has become quite familiar with the DigiD system. DigiD is an authentication system used by the Dutch goverment to let people sign in to government online services in a secure manner over the web, so that they can interact with the goverment online to file tax forms, apply for support and filing criminal complaints, among other services. DigiD as a system has become essential to the Dutch people. For example, the only way for someone to apply for a university is to go through Studielink, a website which requires you to authenticate through DigiD.

As such an essential system, naturally there have been questions about the security of DigiD. It is a system making us depend on the web, and the web is far from being secure, even (and perhaps especially) for goverment work, as can be evidenced from the 2011 incident with Diginotar[9], for example. There are more general concerns than just those singular incidents, however. In particular, concerns can be raised over the security of the websites that use DigiD themselves. These websites are usually made by the government itself, which mostly ensures that no abuse can take place. However, it is prohibitively expensive for the goverment to have to make their own version for everything, and so there are examples of several DigiD-supported websites that make use of third party JavaScript applications.

Third party JavaScript applications are essentialy applications made by a third party, unrelated to the goverment, in JavaScript, a common programming language in web development. This poses a problem. JavaScript is a relatively simple and open language. Many things can be done with it. Furthermore, third party code is often subject to proprietary constraints, or simply not scrutinized enough, being taken at face value as trusted. There is thus an increased possibility that malicious code is present in a third party application. This poses the following problem for the DigiD authentication system: it was not designed to take into account security faults in the website

that uses it. It is uncharted territory. The question this paper will attempt to answer is then: Can third-party JavaScript applications compromise the DigiD authentication process on a website that uses both?

In order to answer this question we need to know several things. First, how does DigiD work? It is important to know the system itself, and what base assumptions are made for it to function. What security guarantees does DigiD offer? These things are discussed in the first half chapter 3.

Second, JavaScript needs to be examined. In the second half of chapter 3, JavaScript's capabilities are covered, along with several other web development concepts that are relevant, such as HTML iframes and cookies.

Knowing these things we can conceive a general idea of how and at what points JavaScript can affect DigiD. In order to test these assumptions a test environment simulating the basic process of DigiD login was constructed. This environment consists of several servers imitating the agents involved in logging in a user. In chapter 4, we describe the testing environment in further detail.

Using this testing environment we can then test possible avenues of attack. We have performed several attacks with JavaScript: cookie-stealing, changing the redirect link, loading the DigiD page into an iframe and trying to set a server in the middle. All but changing the redirect link proved impossible for JavaScript to perform. In chapter 5, we fully detail these attacks.

We ultimately conclude that while the DigiD process itself is secure, there is still a risk in using third-party JavaScript on goverment websites, as the only attack that worked did not involve DigiD at all.

Chapter 2

Research Plan

In order to answer our primary question, "Can third-party JavaScript applications compromise the DigiD authentication process on a website that uses both?", we need to split it into several subquestions to solve:

How does DigiD work?

Answering this question will let us know exactly how DigiD works, which will let us know where and how we can possibly affect it with JavaScript. We answer this question by studying the DigiD documentation, and by observing the authentication process itself via tools that capture network request messages.

What security guarantees does DigiD offer?

Once we know how DigiD works, we will also need to know in what ways it is secure. In what ways can it guarantee security, and what does this mean for JavaScript? Answering this question narrows down the possibilities for JavaScript attacks to a manageble degree. As with the previous question, we answer this question by studying documentation and observing the process in action.

What can JavaScript do?

Knowing the capabilities and limits of JavaScript is essential to figuring out what harm it can cause on a goverment website. Answering this question is difficult, as there is a lot JavaScript can do. Therefore we have chosen to research the capabilities of JavaScript specifically with regards to the possible avenues of attack: we think up a way to affect a part of the DigiD process or a security guarantee, and then use documentation, examples and tests to figure out if JavaScript can accomplish this.

What DigiD security guarantees can JavaScript compromise?

Knowing what security guarantees JavaScript can compromise will let us know how JavaScript can affect the DigiD authentication process and if this is indeed a problem. We will answer this question by using the answers of the previous questions to formulate attacks on DigiD, and then testing these attacks in a simulated DigiD environment to see their effectiveness.

Once all these questions are answered, we can conclude wether JavaScript can compromise DigiD authentication.

Chapter 3

Preliminaries

3.1 DigiD

DigiD is a digital authentication system used by government websites and web services [10]. The service was launched in limited fashion in 2003, under the name Nieuwe Authenticatie Voorziening (NAV). It was renamed DigiD (Digitale Identiteit) in 2004, and made available for use by all Dutch citizens on 1 January 2005 [4]. The service is currently being managed by Logius, a department of the Dutch Ministry of the Interior and Kingdom Relations concerned with ICT-management. Logius provides ICT products used by most branches of the goverment.

The goal of DigiD is to authenticate a user's identity to a government web service. The DigiD service is used for a variety of online government services, among which are filing tax forms, applying for support and filing criminal complaints. In order to authenticate a user the service is tied to the BSN (Burgerservicenummer) of the user, a unique 9-digit number assigned to each Dutch citizen. A user's DigiD identity then consists of their BSN, a username, a password and a mobile phone number. The latter three are used when logging in to identify a user as being the person associated with that particular BSN.

In the more general case, a DigiD identity is not always tied to the BSN, but to their sector number. There are several sectors within DigiD (currently four) that each have a sector code and an associated unique kind of sector number to identify someone:

- **Sector Code S00000000**: this sector uses the BSN as sector number.
- **Sector Code S00000001**: this sector uses the SOFI number as sector number.
- **Sector Code S00000002**: this sector uses the A-number number as sector number.

- **Sector Code S00000100**: this sector uses the OEB as sector number.

This sector number is given to the websites to which the user signed in using DigiD.

There are three security levels: basic, middle and high. At the basic level, a user logs in with their username and password. At the middle level, a user logs in with their username, password, and an SMS code that is sent to their registered mobile phone number. The high level is not currently in use, but in the future, the user will have to authenticate at this level with an electronic identity card.

In addition to this, DigiD offers the option for Eenmalig Inloggen (EI), also know as Single Sign On (SSO). EI allows a user to make use of several different websites without having to reauthenticate for each. If a website supports EI, DigiD remembers that the user has logged in. If they then try to log in on another website that supports EI, they won't have to prove their identity for that website again. DigiD keeps several federations of websites to which a user can authenticate simultaneously using EI.

A typical DigiD login session (from the user's perspective) is as follows: A user wants to access restricted content on a government website. The user clicks on the DigiD login button on the website, and is redirected to a DigiD login page. There, depending on the requested security level, the user enters their username and password and optionally an SMS code sent to their mobile phone. If these are entered correctly, the user is redirected back to the government website. He is now logged in.

The DigiD system currently offers support for the following four protocols:

- A-Select with CGI.
- A-Select with SOAP.
- A-select with WSDL.
- SAML.

In this paper we only describe DigiD using SAML, because this version is intended to serve as a replacement for the others.

3.1.1 SAML

Security Assertion Markup Language (SAML) is an XML-based framework used to exchange security information, such as authentication data. SAML specifies three kinds of actors: Service Providers (SP), Principals and Identity Providers (IdP). A Service Provider is a service that wants to authenicate a Principal via an Identity Provider. The Principal authenticates himself to the Identity Provider, who is trusted by the Service Provider. The

Identity Provider then securely sends the identity information of the Principal to the Service Provider. In DigiD these roles are filled as follows: the Service Providers are government web services that support the use of DigiD, the Idenity Providers are the DigiD servers and the Principals are Dutch citizens.

SAML itself is described by means of assertions, protocols, bindings and profiles. An SAML assertion [6] is a secure information packet that contains statements about a Principal that an asserting party considers true, and is transferred to a relying party. The asserting party in this case is the Identity Provider, with the relying parties being the Service Providers. An SAML protocol [6] describes the way SAML elements are packaged into an SAML request or response. An SAML binding [7] is a description of how to transform an SAML message into a format that can be used by standard communication protocols, such as a SOAP envelope or an HTTP Redirect. An SAML profile [8] describes a combination of assertions, protocols and bindings that can be used to convey security information to solve specific problems. The SAML profiles used by DigiD are the standard Web Browser SSO Profile and the Single Logout profile.

Web Browser SSO Profile

The Web Browser SSO Profile [8] is used to allow a user to make use of several distinct webservices using the same Identity Provider without having to reauthenticate at each webservice. DigiD uses this profile for both its EI login and its normal login. The EI login is functionally identical to a normal login, only after DigiD has authenticated the Principal, it checks if the Principal has indicated that they want to use EI. If so, DigiD notes the user in a table. A cookie is also stored on the Principal's side, to identify him as being part of an EI session to DigiD. After this, if a user attempts to login to a different web service using DigiD, he sees a screen explaining why he does not need to login again.

The protocol used by the Web Broswer SSO Profile as used by DigiD can be summarized as follows [10]:

1. The user requests access to a government service using DigiD.
2. The user is redirected to the DigiD website via an HTTP GET or POST message. This message contains a SAML AuthnRequest.
3. The user arrives at the DigiD login page.
4. The user logs in to the DigiD login page.
5. The user is redirected back to the government service, again via an HTTP GET or POST message. DigiD gives the user a SAML artifact, which the user passes on to the government service.

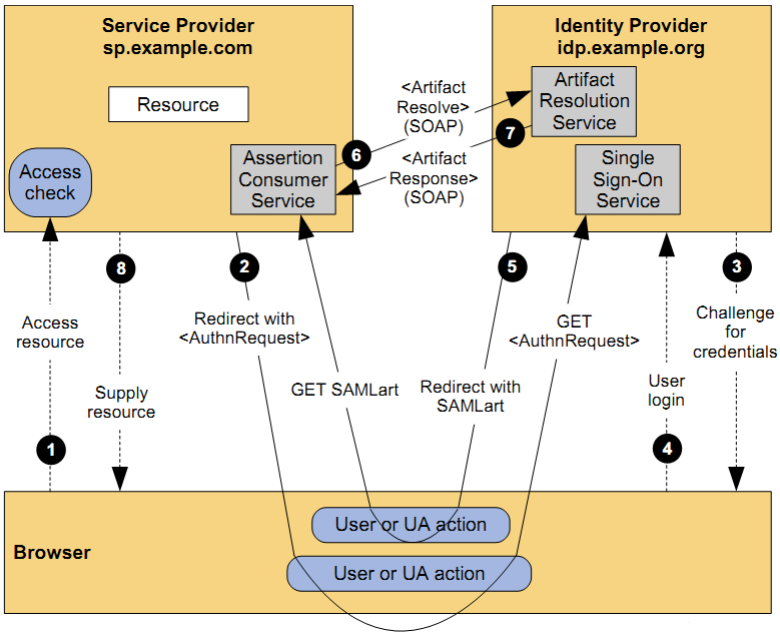

Figure 3.1: DigiD Single Sign On

6. The government service passes the received artifact directly back to DigiD, without first passing it through to the user.

7. DigiD sends a SAML authentication message to the government service.

8. The user gains access to the government service.

It is important to note that if an HTTP GET message is used in step 2, it is *required* that an HTTP GET message is used in step 5 as well. This is also true for the HTTP POST messages in those steps.

There are two channels of communication present between the Service Provider and the Identity Provider: a direct backchannel between the two, and a frontchannel where communication is passed through the Principal. Backchannel SAML messages are passed along in signed SOAP envelopes [5] sent via HTTP. As the backchannel is unreachable for JavaScript, it is not relevant to this paper and will not be discussed in more detail.

Frontchannel messages are SAML messages sent directly as HTTP messages. This is done via either the SAML HTTP Redirect binding or the SAML HTTP Post binding.

In the HTTP Redirect binding [7], an SAML message is transmitted in URL parameters of an HTTP GET request using the Principal as an intermediary. First, any signature on the message itself is removed (this

excludes signatures in the content of the message). Second, the message is compressed according to the DEFLATE compression mechanism [17]. The compressed data is base64-encoded according the the IETF RFC 2045 format [18]. The resultant data is then URL-encoded, and added to the URL as a query string parameter named SAMLRequest. If the original SAML message was signed, a new signature is computed from the SAMLRequest parameter and a new SigAlg paramter, which indicates which algorithm was used to sign the message.

In the HTTP POST binding [7], an SAML message is transmitted in the content of an HTML form control in an HTTP POST request using the Principal as an intermediary. A SAML message is encoded into the form by first encoding the XML representation of the message in base64 [18] and then placing the result in a form as per HTML 4.01 specification [15] section 17. If the message is a SAML request, the form control is named SAMLRequest. If it is a response, the form is named SAMLResponse. The action attribute of the form is the HTTP endpoint to which the message is to be delivered. The method attribute is POST.

In this profile, there are two messages passed through the frontchannel: an AuthnRequest and a SAML artifact.

The AuthnRequest [6] contains the following standard attributes:

- **ID**: A unique identifier for the request.
- **Version**: The version of the request. The SAML version DigiD uses is 2.0.
- **IssueInstant**: The time instant the request was issued. This time is in Coordinated Universal Time (UTC).

In addition, there are several DigiD specific attributes [10]:

- **Issuer**: The name of the web service. This is used to check the signature.
- **RequestedAuthnContext**: Contains an attribute "Comparison = minimum" and a AuthnContextClassRef which contains the minimum required security level of the webservice.
- **ForceAuthn**: A boolean field which defaults to "false". A web service can use this field to force a user to authenticate.
- **ProviderName**: An optional field. This contains the name of the web service shown to the user during DigiD authentication.
- **AssertionConsumerServiceIndex** or **AssertionConsumerServiceURL**: The metadata index or URL the user is sent to when authentication ends.

- **Digital Signature**: The signature of the web service over the entire message. This field is only included when the HTTP Post binding is used.

The SAML artifact [7] is a small message passed along from the Identiy Provider via the Principal to the Service Provider. The Service Provider uses this message to directly request an authentication response from the Identity Provider via the backchannel. The artifact itself is a short string. SAML V2.0 specifies one artifact type, of the following format:

1. **TypeCode**: A two-byte code identifying the artifact type. In this case the type code has value 0x0004.
2. **EndpointIndex**: A two-byte index identifying a specific endpoint that the Service Provider must contact to receive an authentication response.
3. **SourceID**: The SHA-1 hash of the identification URL of the Identity Provider.
4. **MessageHandle**: A pseudorandom number sequence padding the total length of the string to 20 bytes.

This string is then base64-encoded [18]. An SAML artifact is passed on unsigned. This is not a security risk because the information in the artifact is only a reference to an authentication message the Identity Provider has. Without access to the Identity Provider this information is essentially useless.

Single Logout Profile

The Single Logout Profile [8] is used to simultaneously end all of a user's sessions with all webservices he is currently authenticated to. DigiD only uses this profile to let a user logout from an EI session.

The Single Logout process as used by DigiD can be summarized as follows [10]:

1. The user indicates he wants to globally log out to the webservice.
2. The user is redirected to the DigiD website via an HTTP GET or POST message. This message contains a LogoutRequest.
3. DigiD uses the HTTP-SOAP binding to send a LogoutRequest to all other webservices the user is logged into.
4. All other webservices remove the user's local session and send a LogoutResponse to DigiD.

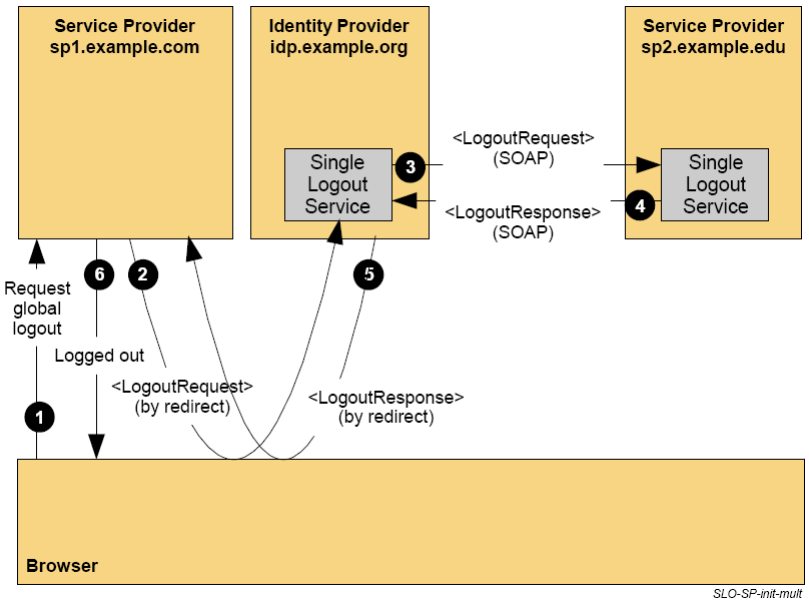

Figure 3.2: DigiD Single Logout

5. DigiD removes the user's SSO-session from its servers and sends the user back to the webservice it logged out from via an HTTP GET or POST message. This message contains a LogoutResponse for the webservice.

6. De webservice informs the user he is logged out, and sends him to a publically available page.

It is again important to note that if an HTTP GET message is used in step 2, it is *required* that an HTTP GET message is used in step 5 as well. This is also true for the HTTP POST messages in those steps.

This profile, like the Web Service SSO profile, contains a backchannel and a frontchannel. Backchannel SAML messages are passed along in signed SOAP envelopes [5] sent via HTTP. Frontchannel communication is achieved via either the HTTP Redirect binding or the HTTP POST binding [7].

There are two kinds of messages passed along the frontchannel: a LogoutRequest and a LogoutResponse.

The LogoutRequest and LogoutResponse contain the same attributes as specified by DigiD [10]:

- **IssueInstant**: The time instant the request was issued. This time is in Coordinated Universal Time (UTC).

- **Issuer**: The name of the web service. This is used to check the signature.

- **Subject**: The sectorcode and sectornumber of the user.
- **Digital Signature**: The signature of the web service over the entire message. This field is only included when the HTTP Post binding is used.

Security Guarantees

The Digid with SAML protocol provides the following security guarantees [10].

Firstly, all HTTP messages are protected because they are transported using either TLS 1.0 or SSL 3.0 with PKIoverheid certificates. This holds for messages from DigiD to the Service Provider as well as messages from DigiD to the Principal. This means that for messages in transit to and from DigiD, confidentiality and data integrity are assured.

Secondly, the SOAP messages on the backchannel between DigiD and the Service Provider are always signed, as are Authentication Request messages. This is a further measure to ensure data integrity.

Thirdly, the protocol itself offers a guarantee of safety from weaknesses on the Principal's device by the use of the SAML artifact to identify the user to the Service Provider. Because the artifact is only a reference to an actual Authentication Message, no actual sensitive information is passed via the insecure channel of the Principal.

In addition to this, DigiD offers some special guarantees for its EI service. DigiD keeps track of the IP adress of a user currently in an EI session. If the IP adress changes during the course of this session, the EI session is terminated immediately. This is done to prevent someone remotely hijacking the EI session.

There is also a time limit on an EI session. The time limit is present to ensure the user is active during the EI session and to minimize the possibility of a malicious third party with physical access to the user's machine being able to misuse his DigiD. In order to enforce this time limit, DigiD keeps track of three timers: a session timeout (15 minutes), a grace period timeout (15 minutes) and an absolute timeout (2 hours). After the session timeout runs out, the user's EI session will be marked inactive; if the user tries to access a different website with EI, he will have to reauthenticate. Upon reauthentication the session timeout will be reset to 15 minutes. After the session timeout runs out, the grace period timeout starts running. If the user accesses resources on the same website within the grace period, their EI session will be marked as active again. If the absolute timeout, which runs from the start of the EI session, runs out, the EI session is permanently deleted.

3.2 Web Services

In this section we will describe some key JavaScript concepts, as well as related web development items.

3.2.1 HTML

HyperText Markup Language [15] (HTML) is a publishing language used by the World Wide Web. HTML is used to create webpages to be displayed in a browser. An HTML webpage consists of an HTML document, which is retrieved from a server and interpreted by a browser to display the desired information/webpage. A document is made up of several elements. These elements represent different parts of a webpage, such as links, tables, and paragraphs of text. An HTML element generally consists of two tags, a start tag (representend as $< element >$) and an end tag ($< /element >$), and the content in between these tags. An element can also be assigned specific attributes, such as a name or class. These attributes are codified into the start tag as follows: $< elementattribute1 = "value" attribute2 = "value" >$.

3.2.2 Iframes

A specific type of HTML element is the frame. Frames allow an HTML webpage to be seperated into several different segements. Each of these segments can contain a seperate HTML document. This allows several pieces of seperate content to be viewed on the same webpage. A particular frame of interest is the iframe, which is short for inline frame. An iframe is a frame present inside another HTML document, instead of several HTML documents existing alongside each other as with normal frames.

As iframes load other pages directly into the current webpage, it is possible for them to be used to inject malicious code into the webpage [16]. Iframes can also be used in, for example, malicious advertising, embedding hidden, malicious advertisements into a webpage [21].

3.2.3 JavaScript

JavaScript is a programming language for web development. JavaScript is generally used to make client-side scripts, which are computer programs executed in a user's web browser. These programs can alter the HTML document, allowing for a measure of interactivity on webpages. JavaScript is a very open programming language, and can be used in many ways. For example, there is the concept of bookmarklets [19], which are JavaScript applications contained in a bookmark. Such applications can then be executed on any page the user wishes, allowing one to alter practically any website locally. Another example is Greasemonkey [2], a Mozilla Firefox extension. Greasemonkey allows users to install scripts which can alter the

webpage content in their browser before or, like bookmarklets, after the page is loaded. For Greasemonkey, changes made to a page are repeated every time the page is loaded, which effectively makes them permanent alterations as far as the user is concerned.

JavaScript is a well known and popular language, and is used on virtually every website for many things, allowing for dynamic content without refreshing the page, for example. As it is so omnipresent, it is also used in a lot of hacks and for circumventing browser security. A popular method of using JavaScript is for Cross-Site Scripting attacks (XSS) [22]. These attacks enable attackers to inject client-side scripts into websites visited by other people. This allows them to alter the site to, for example, gather and pass on data to a third party.

3.2.4 Same Origin Policy

JavaScript subscribes to the security concept of the Same Origin Policy (SOP) [13]. The SOP is a policy that prevents malicious scripts from one site accessing resources on another site. It accomplishes this by restricting the access a script has. A script running under SOP can only access resources that come from the same origin. An origin, in this case, is a combination of the scheme, host and port from which the script originated. The host is generally represented by the domain name of the site, the scheme by the application layer protocol and the port by the port number of the HTML document to which the script belongs. This prevents, for example, one site from accessing the cookies of another site, as cookies have a field specifying the domain. Thus, following SOP, a site that has a different domain has a different origin, and cannot access these cookies. All modern commonly used browsers (Firefox, Chrome, Internet Explorer) have built-in support for the SOP.

3.2.5 Cookies

Cookies [12] are small pieces of data that a server stores on a user's hard drive. These small pieces of data contain information about the user relevant to the server, such as a session-id or information that identifies a user as being logged in to that site. Each individual cookie is a name-value pair. Cookie information exchange goes as follows: on an initial visit to a webpage, its server sends a Set-Cookie header, which contains the cookie. The user stores this cookie on their hard drive. On subsequent visits to the page, the user adds the cookie to its initial request. In this way, the server can receive information about the user.

In addtion to the normal name-value pair, cookies also commonly have most of the following attributes:

- **Expires**: This attribute represents the maximum lifetime of the cookie,

represented as the time and date at wich the cookie expires. Once expired, a cookie will no longer be recognized as in use.

- **Max-Age**: This attribute represents the maximum lifetime of the cookie, represented as the number of seconds till its expiration. Once expired, a cookie will no longer be recognized as in use.
- **Domain**: This attribute specifies to which hosts a cookie will be sent. If the Domain value is "example.com" , the cookie will be sent with HTTP requests to example.com and all its subdomains.
- **Path**: This attribute represents with which HTTP requests a cookie will be sent. A cookie is only included if the path of the request matches the path in the Path attribute. This allows different cookies to be associated with different pages of a website.
- **Secure**: This attribute ensures that a cookie is only included in an HTTP request if that request is sent over a secure channel.
- **HttpOnly**: This attribute ensures that the user only sends back this cookie with HTTP requests. This prevents "non-HTTP" requests, such as a JavaScript script, from accessing the cookie values, when they can otherwise do so.

These attributes serve to grant information about the cookie itself: when it stops being valid and when it needs to be sent along with a request.

A common attack on a webpage is to steal cookies from a user. Specifically, stealing a *user ID* cookie would allow an attacker to impersonate the original owner of the cookie. We will explore this method of attack on DigiD in the next section.

Chapter 4

Test Environment

In order to test our hypothetical attacks we have constructed a simple test environment to simulate the basic interaction of DigiD authentication. For this test environment, only the login process will be considered, as there is not enough difference between the structure of the messages being passed along. The only difference between messages in the login and logout procedures is the specific information fields that are passed on.

We have three seperate servers: a Service Provider, an Identity Provider and an Evil server. These servers are Apache 2.4 servers[1]. The Service Provider emulates the SAML Service Provider. It shows the user a basic page, based on the goverment website uwv.nl[3]. This page loads malicious JavaScript from the Evil server. If the user tries to login to DigiD, they are directed to the Identity Provider page. This page allows a user to authenticate (in this case by pressing a button) and sends one back to the Service Provider once authenticated. The Evil server serves as a container for all malicious JavaScript. In addition to this, if a user is directed to the Evil server from the Service Provider, they are redirected to the Identity Provider, with the Evil server acting as an intermediary. The user uses a Browser to access these pages/servers.

We can illustrate the test login process as follows:

The process starts immediately upon opening the Service Provider page:

1. As the page loads, the malevolent JavaScript scripts are loaded from the Evil Server alongside the rest of the page.
2. The user tries to login.
3. The user is redirected towards the Identity Provider.
4. The identity provider asks the user to login.
5. The user logs in.
6. The user is directed back towards the Service Provider.

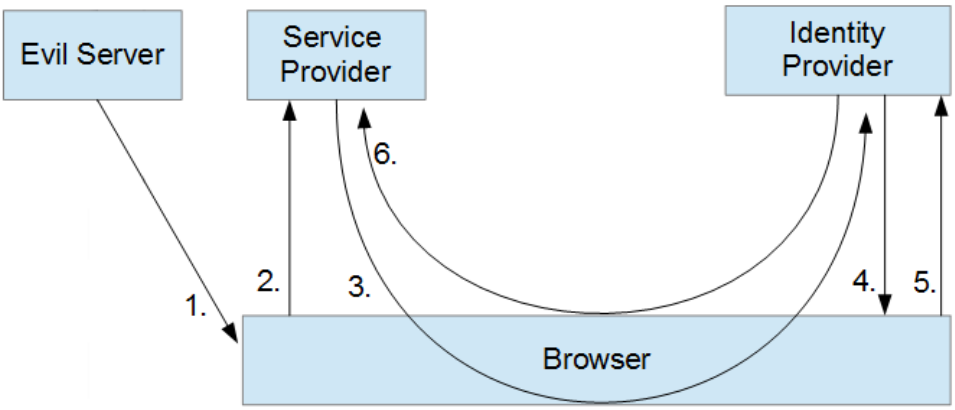

Figure 4.1: The Test Login Process

This represents the basic process, but does not cover the entirety of what is possible. The process is subject to change depending on the specifics of the executed malevolent JavaScript from the Evil Server. For example, a script could change the redirect to the Identity Provider, altering the entire process from 3. onwards.

In this test environment we used Google Chrome as the Browser. Chrome was chosen because it is a widely used browser and because it has powerful developer tools. These developer tools allow us to easily view the HTML document of a webpage, as well as see all the HTTP requests, responses and redirects that are related to that instance of that webpage. Viewing the HTML document makes us able to see the changes a JavaScript script makes to the page.

Chapter 5

Tests

In this section we try to construe an attack on the DigiD authentication process using a third party JavaScript application on a government website. We define no particular restrictions on this application except for those imposed on it by JavaScript itself.

To determine possible avenues of attack, we refer to figure 3.1: essentialy every step that originates from or passes through the user's browser can theoretically be affected by JavaScript. Furthermore, we consider the process before it is initiated and after it is complete.

5.1 Cookies

The first avenue of attack considered is trying to steal an active DigiD session from a user. The most direct way to go about this is simply via trying to acquire the _session_id cookie DigiD stores in a user's browser. There are, however, two major obstacles preventing this from being a viable approach: the Same Origin Policy (SOP) and the httponly attribute of the cookie. The SOP prevents a script from accessing resources not belonging to the same origin. The attacking script has as part of its origin the government website domain, while the _session_id cookie has as its origin the DigiD authentication server. As such, it is generally not possible for the attacking script to reach the cookie. Even assuming the SOP can be bypassed [14], the cookie would still remain inaccessible due to the fact that its httponly attribute is set. This attribute allows only HTTP requests to use this cooke, which means that no script can ever access that particular cookie. Thus, JavaScript cannot access any DigiD cookies to impersonate a user, rendering this avenue of attack impossible.

5.2 Link

The second avenue of attack considered is trying to change the link to the protected resource that initiates the DigiD login process. If possible, this opens up several possibilities, like linking the user to a malicious copy of the DigiD site, or attempting to establish a Man-In-The-Middle setup.

A cursory evaluation of several government sites (in this case, `http://mijn.overheid.nl` and `http://www.uwv.nl/Particulieren/mijnuwv/index.aspx`) reveals that the links to the protected resource are usually present as the href attribute of an $< a >$ tag in the body of the html page. In both cases, the tag also had either a class or id attribute. Taking this knowledge into account, changing the link becomes fairly trivial with the following JavaScript code:

```
1 window.onload = function(){
2 document.getElementsByClassName("digid")[0].href="http://evil.
      com";
3 }
```

Listing 5.1: Changing the link via Class

or

```
1 window.onload = function(){
2 document.getElementById("DigiD").href="http://evil.com";
3 }
```

Listing 5.2: Changing the link via ID

This code waits until the page has finished loading, and then locates and changes the link to the protected resource, making it point anywhere the attacker might want it to. As a result, it becomes possible for the attacker to bypass DigiD entirely. He might make the link point to a duplicate DigiD site to phish for login details, or he might insert himself in between the user and DigiD, potentially allowing him to eavesdrop on sensitive information. It should be noted that this is not an attack on the DigiD process per se, as the protocol is never initiated. It is still a risk, however, as it can allow an attacker to phish for login info.

5.3 Iframe

Another option that was considered was to try and obtain usernames and passwords in a different way. This could theoretically be done by loading the DigiD site into an iframe and then, by using an overlay over the full screen, reading in the user input. This was intially tested using the following script, which creates an iframe when clicking the link to DigiD:

```
1 function makeFrame() {
2     ifrm = document.createElement("IFRAME");
3     ifrm.setAttribute("src", "https://digid.nl");
```

```
4       ifrm.style.width = 100+"%";
5       ifrm.style.position="fixed";
6       ifrm.style.top=0+"px";
7       ifrm.style.left=0+"px";
8       ifrm.style.zindex=5;
9       ifrm.style.height = 100+"%";
10      document.body.appendChild(ifrm);
11  }
12
13  window.onload=function() {
14       document.getElementsByClassName("digid")[0].href="#";
15       document.getElementsByClassName("digid")[0].onclick=
             makeFrame;
16  }
```

Listing 5.3: Iframe

The makeFrame function creates an iframe that opens the DigiD page. On loading the webpage, the script locates the original link to the protected resource, nullifies it, and then changes the link so that it instead opens the DigiD iframe when clicked.

However, when attempting to load the DigiD site, the iframe was unresponsive. This is because of the X-Frame-Options HTTP response header[11]. The X-Frame-Options header is used to indicate whether or not a browser should be allowed to render a page in a frame or an iframe. There are three possible values for this header:

- **DENY**: The page cannot be displayed in a frame.
- **SAMEORIGIN**: The page can only be displayed in a frame of the same origin as the page, following the SOP.
- **ALLOW FROM *uri***: The page can only be displayed in a frame of the specificed origin.

DigiD has set the X-Frame-Options HTTP response header to "SAMEORIGIN". As modern browsers respect the response headers, it thus becomes impossible for the DigiD site to be accessed via iframe for a majority of target users. This makes the attack near impossible, as without the iframe, an overlay created by the initial goverment page cannot possibly reach the DigiD site. The DigiD site is either open in another window/tab, or it replaces the goverment page. Overlays cannot be created for other windows, and if the DigiD site replaces the goverment page the script is gone as well, rendering it impossible to act.

5.4 Man In The Middle

The final option that was looked at was to try to intercept the messages between the Service Provider and the Identity Provider. There are two types

of messages: messages that go directly between the two and messages that travel via the user's browser. Obviously, JavaScript cannot affect the direct messages in any way, so we will look only at messages that pass through the browser.

We were unable to devise a method to capture or observe the messages that passed through the browser using only JavaScript, so we had to look at placing an intermediary agent between the user and the Identity Provider. The chosen agent in this case was the Evil server. This server would, once in place between the user and the Identity Provider, let all traffic throug, recording all the messages.

Unfortunately, we were also unable to create a method to place the Evil server in between the user and Identity Provider. The only way to make it possible would appear to be to change the browser's proxy settings, which a user can do either directly or by receiving a PAC file (Proxy Auto Config file)[20]. However, neither appear to be an option for client-side JavaScript scripts.

Chapter 6

Conclusions

As the importance of the web continues to grow in our lives, so too does DigiD become ever more important. It is therefore important that we continue to question and test the way it is used. In this work, we attempt to attack DigiD through the use of third-party JavaScript applications on goverment websites. This was done by formulating several attacks on different parts of the DigiD login process. Our findings show that for the most part, DigiD is secure from our attempts to compromise it with JavaScript. The only viable attack found was one that assumed that the DigiD authentication has yet to start. This attack is still a risk, however. It can allow an attacker to phish for passwords, by redirecting the user to a fake DigiD site. This shows that while DigiD itself is secure, it is still important to look at how it is used on websites, and extra care should be taken with using third-party scripts on pages that allow one to access DigiD.

Bibliography

[1] Apache HTTP Server Version 2.4 Documentation. from https://httpd.apache.org/docs/2.4/.

[2] Greasespot. from http://www.greasespot.net/.

[3] Uitvoeringsinstituut Werknemersverzekeringen. retrieved from http://www.uwv.nl on the 29th of May, 2013.

[4] DigiD voor alle burgers beschikbaar, 2005. from https://www.digid.nl/nieuws/artikel/artikel/digid-voor-alle-burgers-beschikbaar/.

[5] SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). Technical report, World Wide Web Consortium (W3C), 2007. http://www.w3.org/TR/2007/REC-soap12-part1-20070427/.

[6] Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0 - Errata Composite. Technical report, Organization for the Advancement of Structured Information Standards (OASIS), 2009. https://www.oasis-open.org/committees/download.php/35711/sstc-saml-core-errata-2.0-wd-06-diff.pdf.

[7] Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0 - Errata Composite. Technical report, Organization for the Advancement of Structured Information Standards (OASIS), 2009. https://www.oasis-open.org/committees/download.php/35387/sstc-saml-bindings-errata-2.0-wd-05-diff.pdf.

[8] Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0 - Errata Composite. Technical report, Organization for the Advancement of Structured Information Standards (OASIS), 2009. https://www.oasis-open.org/committees/download.php/35389/sstc-saml-profiles-errata-2.0-wd-06-diff.pdf.

[9] Meest gestelde vragen over Diginotar, 2011. from http://nos.nl/artikel/269652-meest-gestelde-vragen-over-diginotar.html.

[10] Koppelvlakspecificaie SAML Authenticatie. Technical report, Logius, Ministerie van Binnenlandse Zaken en Koningkrijksrelaties, 2012. http://www.logius.nl/fileadmin/logius/product/digid/documenten/Koppelvlakspecificatie_SAML_DigiD4_v2.2.pdf.

[11] The X-Frame-Options response header, 2013. https://developer.mozilla.org/en-US/docs/HTTP/X-Frame-Options.

[12] Barth, Adam. RFC 6265-HTTP State Management Mechanism. *Internet Engineering Task Force (IETF), ISSN*, pages 2070–1721, 2011.

[13] Barth, Adam. The web origin concept. 2011.

[14] Brian Chess, Yekaterina Tsipenyuk O'Neil, and Jacob West. Javascript hijacking. *Online document*, 2007. http://scholar.googleusercontent.com/scholar?q=cache:Rrr2Ephqbj0J:scholar.google.com/+javascript+hijacking&hl=en&as_sdt=0,5.

[15] D. Ragget et al. HTML 4.01 Specification. Technical report, World Wide Web Consortium (W3C), 1999. http://www.w3.org/TR/html4/.

[16] Danchev, D. Mass iframe injectable attacks, March 2008. http://ddanchev.blogspot.nl/2008/03/massive-iframe-seo-poisoning-attack.html.

[17] Deutsch, P. RFC1951, DEFLATE Compressed Data Format Specification, 1996.

[18] Freed, N and Borenstein, N. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. November 1996. *RFC2045*.

[19] Kangas, Steve. About Bookmarklets. http://www.bookmarklets.com/about/.

[20] Pashalidis, Andreas. A cautionary note on automatic proxy configuration. In *IASTED International Conference on Communication, Network, and Information Security, CNIS 2003, New York, USA, December 10-12, 2003, Proceedings*, pages 153–158. ACTA Press, 2003.

[21] Sood, Aditya K and Enbody, Richard J. Malvertising–exploiting web advertising. *Computer Fraud & Security*, 2011(4):11–16, 2011.

[22] Spett, Kevin. Cross-site scripting. *SPI Labs*, 2005.

Appendix A

Appendix

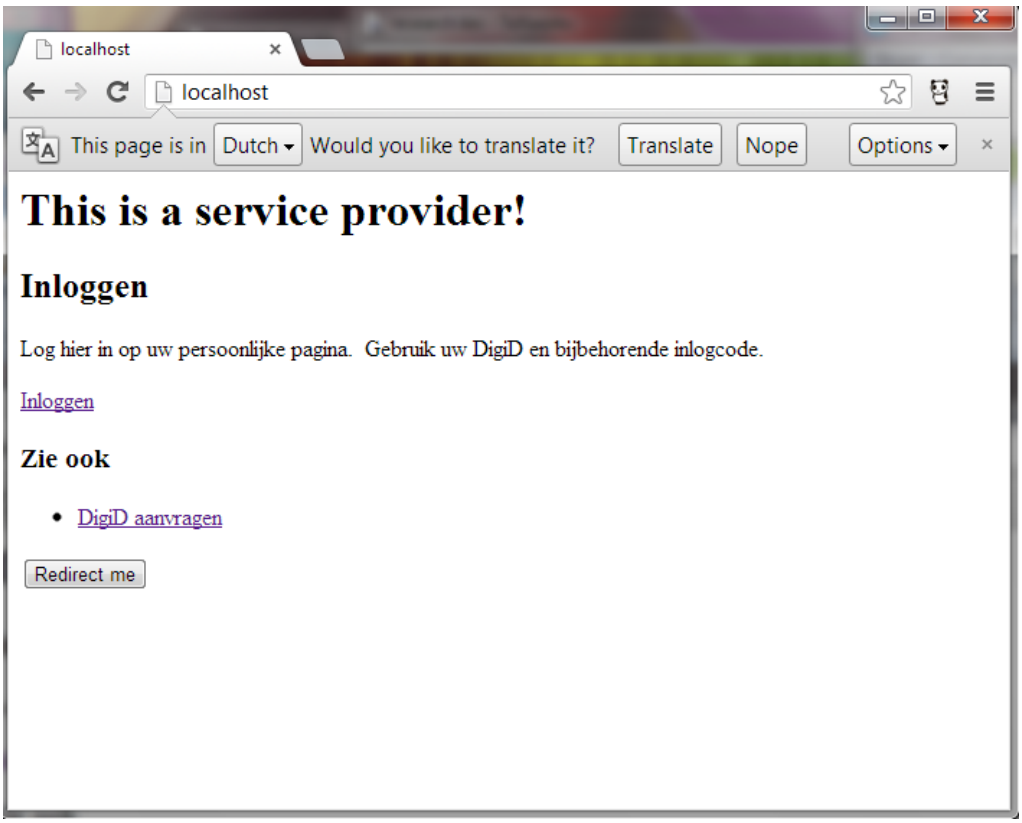

Figure A.1: The Service Provider Test Page

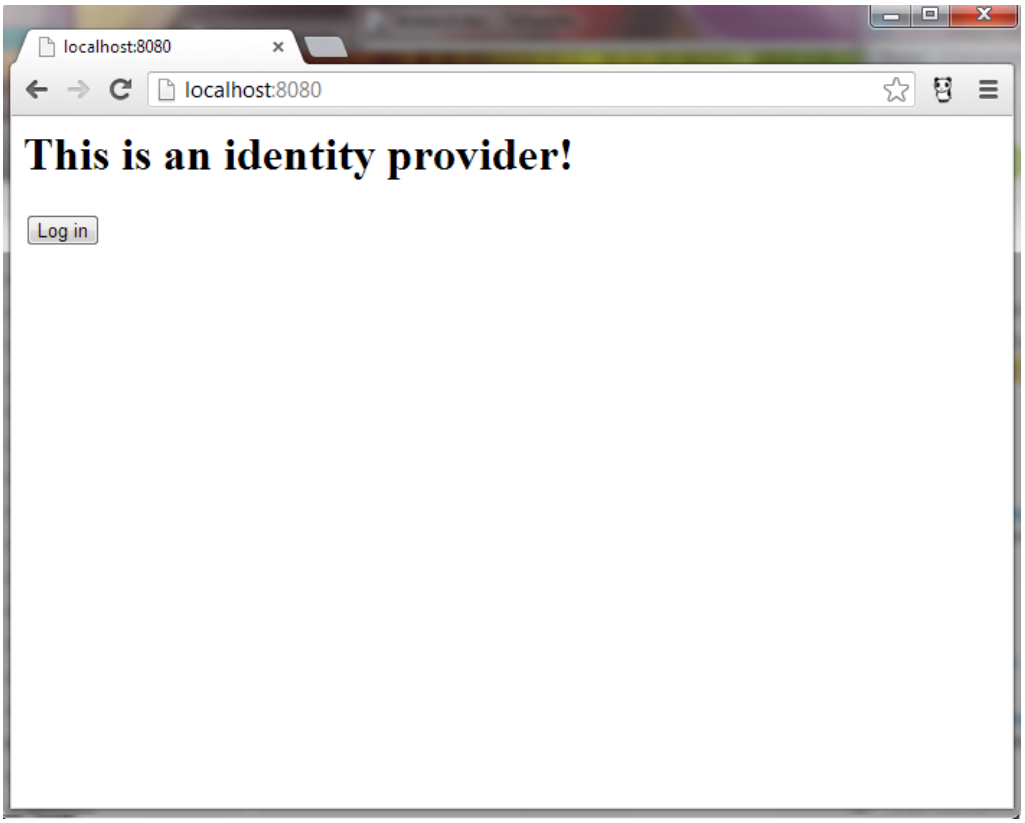

Figure A.2: The Identity Provider Test Page