CS297 Report

Bookmarklet Builder for Offline Data Retrieval

Sheetal Naidu

sheetalnaidu@yahoo.com

Advisor: Dr. Chris Pollett

Department of Computer Science

San Jose State University

Fall 2007

Table of Contents

Introduction ……. 3

Deliverable 1 ……... 3

Deliverable 2 ……... 4

Deliverable 3 ……... 6

Deliverable 4 ……... 7

Future work ……. 8

Conclusion …….. 8

References ……... 8

Introduction

A bookmarklet is a small computer application that is stored as a URL of a bookmark in the browser. Bookmarlet builders exist for storing single webpages in hand held devices and these webpages are stored as PDF files. The goal of my project is to develop a tool that can save entire web page applications as bookmarklets. This will enable users to use these applications even when they are not connected to the Internet. The main technology beyond Javascript needed to do this is the data: URI scheme. This enables images, Flash, applets, PDFs, etc. to be directly embedded as base64 encoded text within a web page. This URI scheme is supported by all major browsers other than Internet Explorer. Our program will obfuscate the actual resulting JavaScript so these complete applications could potentially be sold without easily being reverse engineered. The application will be made available online, to users who are typically website owners and would like to allow their users to be able to use the applications offline.

This report describes the background work that was conducted as preparation for CS298 when the actual implementation of the project will take place. This preparation work was split into four main deliverables. The deliverables were designed at gaining relevant information and knowledge with respect to the main project. The first deliverable was to write a simple JavaScript code that allowed to manipulate some data in a HTML file. The second deliverable was about a web crawler. I had to choose a suitable crawler using which a sample website was crawled and data was read from the crawled database to display one of the saved files from the sample website. The third deliverable was to learn about available JavaScript code obfuscation tools. The fourth deliverable involved converting data to the data:URI scheme. In the remainder of this report I will described in detail all of the work done as part of the deliverables. I will conclude with some main ideas about how the project will be implemented in my CS298 master's project course.

Deliverable 1

My first deliverable was to learn JavaScript. As a concrete deliverable, I wrote a program that demonstrated user interaction with an HTML object. In this program, there is a black box on an HTML page and this black box is defined using tags and not <img> tags. There are four buttons below the black box that allow users to move the box around the screen. Each click moves the box by 15pts in the direction specified by the button. e.g Top moves the box in the top direction, Left moves the box in the left direction and so on.

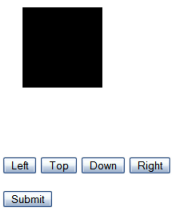

Figure 1: Front-end for Deliverable 1

When you click on Submit, you are taken to the next page which displays the x and y coordinates of the last position of the black box.

Ex:
The x coordinate is : 525px and the y coordinate is: 400.

This was one of the earliest JavaScript codes I wrote and I learned about the use of tags, accessing the query string from a URL and using regular expressions for pattern matching. On the first HTML page I had two hidden form elements which contain the present x and y coordinates of the black box. When we hit on Submit, these two values are sent as part of the query string to the next page. On the second page, I retrieved these values from the query string and displayed them in the browser.

Deliverable 2

An important part of our final project will be, to be able to crawl a specified site in order to retrieve all the pages of that site. As part of the second deliverable, my job was to do some research and choose a web crawler and learn about it so that I could exploit its features and learn about its commands so that I could use them. For this, I chose a crawler called Nutch. Nutch is a web crawler written in Java which is part of an open source search engine based on Lucene for its search and index component. Nutch was previously a part of Apache but now it is a subproject of Lucene. This deliverable was conducted in two parts. In the first part I did some research on how to install Nutch and use it from the command line. It maintains a database of pages and links that it fetched during the crawl. The pages have scores that are assigned by analysis. Every time the crawl is made, it looks for high-scoring out-of-date pages and fetches them to update its database.

The second part of the second deliverable was to write PHP code so that I could run some of the Nutch commands through my script. For this, the user is presented with a front-end where he/she can enter the URL of the site to be crawled. This URL is given as input to Nutch's 'crawl' command and the crawled data is stored in a directory. In the PHP script, I next retrieve some useful data from the database and display it in a human-readable form. For this, I used the readdb command in Nutch. This command reads data such as the URLs of all the pages visited during the crawl.

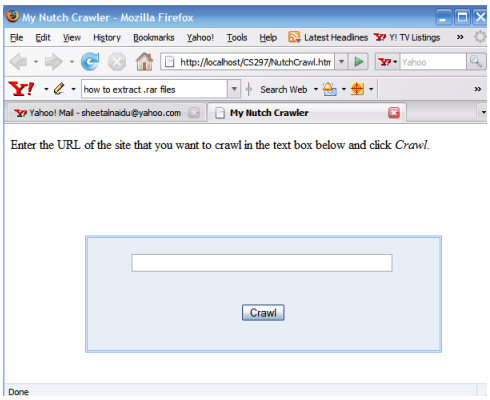

Figure 2: Front-end of Nutch execution

The front end provided to the user to enter the crawl site is shown in the screenshot above. When the user enters the URL of the site in the text box and clicks on Crawl, a PHP script is invoked which crawls the site with the URL provided and stores the data in a directory. After the crawl is completed, the user sees the following message.

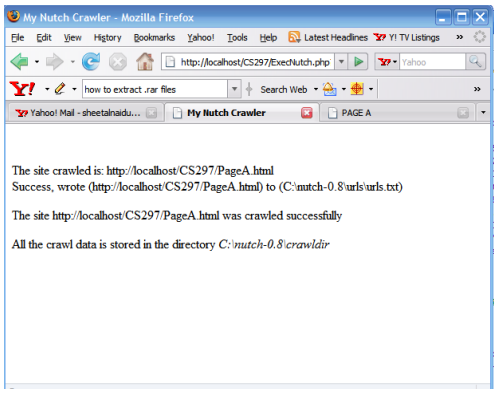

Figure 3: After crawl message

Then, when we use the readdb command, it displays the following information. Different options supplied to the readdb command provide different kinds of output.

http://localhost/CS297/PageA.html Version: 4
Status: 2 (DB_fetched)
Fetch time: Fri Dec 07 16:28:34 PST 2007
Modified time: Wed Dec 31 16:00:00 PST 1969
Retries since fetch: 0
Retry interval: 30.0 days
Score: 1.6666667
Signature: e48ea88ce7aaa83d3115c598205ea05e
Metadata: null

http://localhost/CS297/PageB.html Version: 4
Status: 2 (DB_fetched)
Fetch time: Fri Dec 07 16:28:44 PST 2007
Modified time: Wed Dec 31 16:00:00 PST 1969
Retries since fetch: 0
Retry interval: 30.0 days
Score: 2.0
Signature: a1c8ea6e230e235c5ff1acb2be4bd202
Metadata: null

http://localhost/CS297/PageC.html Version: 4
Status: 1 (DB_unfetched)
Fetch time: Wed Nov 07 16:28:46 PST 2007
Modified time: Wed Dec 31 16:00:00 PST 1969
Retries since fetch: 0
Retry interval: 30.0 days
Score: 1.6666667
Signature: null
Metadata: null

http://localhost/CS297/PageD.html Version: 4
Status: 1 (DB_unfetched)
Fetch time: Wed Nov 07 16:28:46 PST 2007
Modified time: Wed Dec 31 16:00:00 PST 1969
Retries since fetch: 0
Retry interval: 30.0 days
Score: 1.6666667
Signature: null
Metadata: null

Deliverable 3

Our program will need to use some form of obfuscation on the actual resulting JavaScript so the complete applications that we will make into a bookmarklet can potentially be sold without easily being reverse engineered. This is what formed the basis for my third deliverable. For the first part of it I found out information on the available techniques for code protection. These techniques include watermarking – which is a defense against software piracy, tamper-proofing – which enables the detection of tampered code so that the code becomes unusable, and obfuscation - which is a tool for obfuscating code so that it becomes difficult to reverse engineer. Code is obfuscated by mainly applying certain transformations to the original code. I did further study on how code can be obfuscated in different ways. There are several ways that code can be obfuscated in. These ways depend on what area of the code we aim at obfuscating. These include:

1. Layout transformation – modify variable names
2. Data transformation – modify data structures
3. Control transformation – change program flow while preserving semantics and
4. Preventive transformation – use anti-debugging and anti-disassembly techniques

Most commercially available JavaScript obfuscators obfuscate the code only by aiming at transforming the lexical structure of the code or by changing the layout of the code. For our project, we picked a freeware called Stunnix. As an example, let's examine how a piece of code is obfuscated using Stunnix.

Sample code:

```
function foo( arg1)
{
  var myVar1 =  "some string";  //first comment
  var intVar =  24 *  3600; //second comment
  /* here is
a long
multi-line comment blah */
  document.write( "vars are:" +myVar1+  " " +intVar +" " +arg1);
};
```

The obfuscated code looks as follows:

```
function z001c775808( z3833986e2c) { var z0d8bd8ba25=
"\x73\x6f\x6d\x65\x20\x73\x74\x72\x69\x6e\x67"; var z0ed9bcbcc2= (0x90b+785-0xc04)*
(0x1136+6437-0x1c4b); document. write( "\x76\x61\x72\x73\x20\x61\x72\x65\x3a"+ z0d8bd8ba25+
"\x20"+ z0ed9bcbcc2+ "\x20"+ z3833986e2c);};
```

We can make the following observations about how Stunnix works:

1. The Stunnix obfuscator targets at obfuscating only the layout or the lexical structure of the JavaScript code
2. As the obfuscator parses the code, it removes spaces, comments and new line feeds
3. While doing so, as it encounters user defined names, it replaces them with randomly generated strings
4. It replaces print strings with their hexadecimal values
5. It replaces integer values with complex equations

In our sample code that was obfuscated, we can observe that the user defined variable foo was replaced with the random value z001c775808; the variable arg1 was replaced with z3833986e2c; the variable myvar1 was replaced with z0d8bd8ba25 and the variable intvar was replaced with z0ed9bcbcc2. The

integer value 20 was replaced with (0x90b+785-0xc04) and 3600 replaced with (0x1136+6437-0x1c4b). The print strings “vars are” was replaced with its hexadecimal value

\x76\x61\x72\x73\x20\x61\x72\x65\x3a

and all spaces were replaced with the hexadecimal value of \x20. We also saw that all comments and new line feeds were removed from the resulting code. While this kind of code obfuscation makes the resulting code more compact it also has an overhead of having to decode the equations wherever we need to use simple numerical values. But this only adds a constant extra time.

Deliverable 4

Data:URI scheme - The data:URI scheme is a way of representing data that can be included inline in any file. In order to convert an entire page into a bookmarklet, there should be a way to include objects like images and video files inline with all the other text on the page. For this reason, the images and other objects should be base64 encoded into the data:URI scheme. Accomplishing this was set to be my fourth deliverable. First, I had to convert images into their base64 encoded form. PHP has a very convenient base64_encode() function, which takes in string values and base64 encodes them.
Next, I wrote a complete PHP script that takes the URL of a page as input from the user and fetches the page using cURL() function. It then parses it to see if there are any image files embedded in the file. It then fetches each of the image files from their address, again using cURL() and then base64 encodes these image files individually and displays the images back in the browser in the data:URI scheme. The format for using data in the data:URI form is

data:[<MIME-type>][;base64],<data>

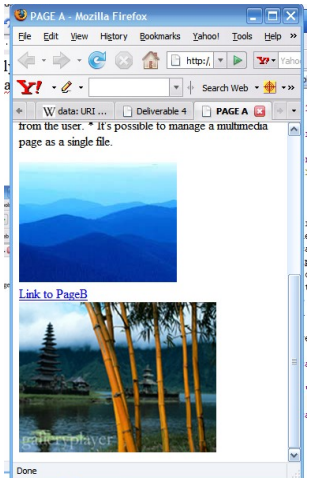

Figure 4: Original page with images

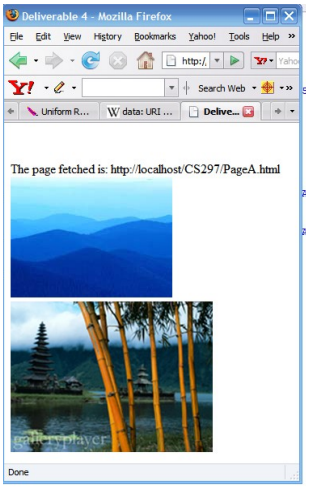

Figure 5: New page with base64 encoded images

The two figures above show the HTML pages before and after the base64 encoding has been applied to the image files. Although they appear the same in the browsers, their sources look significantly different.

For Figure3 the source code looks like:

```
<img src="http://localhost/CS297/Images/Image1.jpg" />
<br />
<a href ="PageB.html">Link to PageB</a>
<img src="http://localhost/CS297/Images/Image2.jpg" />
```

Where as for the new page in Figure4 the source code looks like:

```
<img
src="
AAAABAAEAAAEaAAUAAAABAAAAYgEbAAUAAAABAAAAagEoAAMAAAABAAIAAAExAAIAAAAUAAAAcgEyAAIA
```

AAAUAAAAhodpAAQAAAABAAAAnAAAAMgAAAEsAAAAAQAAASwAAAABQWRvYmUgUGhvdG9zaG9wIDcuMA
AyMDA3OjExOjI3IDEyOjAzOjM4AAAAAOgAQADAAAAAf//AACgAgAEAAAAAQAAAMigAwAEAAAAAQAAAJYA
AAAAAAAABgEDAAMAAAABAAYAAAEaAAUAAAABAAABFgEbAAUAAAABAAABHgEoAAMAAAABAAIAAAIBAAQ
AAAABAAABJgICAAQAAAABAAAGJwAAAAAAAABIAAAAAQAAAEgAAAAB/9j/4AAQSkZJRgABAgEASABIAAD/7
QAMQWRvYmVfQ00AAv/uAA5BZG9iZQBkgAAAAAH/2wCEAAwICAgJCAwJCQwRCwoLERUPDAwPFRgTExUTE
″ />
The above example is modified so it fits in this report. The actual base64 encoded representation of a typical image file is much bigger than the string that is shown above.

Future work

In my CS298 course, we will go about implementing the entire project by combining all of the four deliverables that I worked on in this semester. A high level design idea is to take the URL of a particular site, crawl the site to fetch every page from the site (the number of pages we crawl can be limited to a certain depth of the site), and then convert the necessary images and video files to the data:URI format and then go on to present the entire page in the data:URI format. To fetch the pages and the other files, we will use PHP which will form sort of the back-end. The result of this will be a single JavaScript file which can be obfuscated using the obfuscator we picked. Thus, the resulting application will visibly look like a single line of code. The challenging part will be to maintain all the links to all the pages that were fetched. While fetching images, we should take care that only valid image files are fetched. There should be a way of determining and ensuring that we fetch only an image file and not a 'File not found' error message file, because the curl() function gets whatever response it gets from the server.

Conclusion

Most of the groundwork required to implement the final project is well laid in the form of the deliverables. Putting together all of the research done in this semester will help move things faster in the next semester. The deliverables were implemented on a small scale mostly on small sites that were served from my localhost. Scaling them to work on real world websites will be a challenging feat. I learned a lot about some of the Web Technologies. There was a significant amount of coding involved, which kept things exciting. Knowing about web crawlers and spiders opened up a whole new interesting area for me.

References

1. JavaScript Bible, Danny Goodman with Michael Morrison. Wiley. 2004.
2. JavaScript : The Definitive Guide. David Flanagan. O'Reilly. 2006.
3. Programming PHP. Rasmus Lerdorf, Kevin Tatroe, and Peter MacIntyre. O'Reilly. 2006.
4. RFC 3986. Uniform Resource Identifier (URI): Generic Syntax. Network Working Group. http://gbiv.com/protocols/uri/rfc/rfc3986.html
5. Official page of Nutch project. http://lucene.apache.org/nutch/
6. Introduction to Nutch http://today.java.net/pub/a/today/2006/01/10/introduction-to-nutch-1.html
7. C. Collberg, "The Obfuscation and Software Watermarking homepage", - http://www.cs.arizona.edu/collberg/Research/Obfuscation/index.html
8. Stunnix JavaScript Obfuscator www.stunnix.com
9. Shane Ng's GPL-licensed obfuscator "http://daven.se/usefulstuff/javascript-obfuscator.html"
10. Free JavaScript Obfuscator http://www.javascriptobfuscator.com/
11. PHP Manual http://www.php.net/manual/en/
12. Free online encyclopedia – www.wikipedia.org