Program Analysis For Database Injections

By: Chelsea Ramsingh

Advisor: Paolina Centonze

May 2017, Iona College

ACKNOWLEGMENTS

I would like to thank my professor and advisor, Dr. Centonze for her continuous support and encouragement during my Master year and all the faculty and students at Iona College who helped me achieve my goal. I am very grateful to my family, who has stood by me and motivated me through this and I would like to thank them for everything they have done for me. I would like to thank my parents for guiding me, loving me and being there for me, no matter what happens in my life. They taught me how to be the strong and kind person I am today and to always be positive in life. I am blessed to have an amazing sister, Ashley, who supports me and helps me through every difficult day and night. I would like to thank her because I wouldn't have gotten through this without her. Last I would like to thank my brothers/cousins, Akash and Avikash. Although you both live far away, not a day goes by that you didn't show me how much you believe in me and support me. I'm truly blessed to have these people in my life.

Table of Contents

Program Analysis For Database Injections

BY: Chelsea Ramsingh

ABSTRACT

Today businesses all around the world use databases in many different ways to store sensitive data. It is important that the data stored stay safe and does not get into the wrong hands. To perform data management in a database, the language SQL (Structured Query Language) can be used. It is extremely crucial to prevent these databases from being attacked to ensure the security of the users' sensitive and private data. This paper will focus on the most common way hackers exploit data from databases through SQL injection, and it presents dynamic and static code testing to find and prevent these SQL cyber attacks by comparing two testing tools. It will also present a comparative analysis and static/dynamic code testing of two SQL injection detection tools. Burp Suite and Vega will be used to identify possible flaws in test cases dealing with users' sensitive and private information. Currently, there are no comparisons of these two open-source tools to quantify the number of flaws these two tools are able to detect. Also, there are no detailed papers found fully testing the open-source Burp Suite and Vega for SQL Injection. These two open-source tools are commonly used but have not been tested enough. A static analyzer detecting SQL Injection will be used to test and compare the results of the dynamic analyzer. In addition, this paper will suggest techniques and methods to ensure the security of sensitive data from SQL injection. The prevention of SQL injection is imperative and it is crucial to secure the sensitive data from potential hackers who want to exploit it.

KEYWORDS:

SQL, SQL Injection, SQL Injection Vulnerability, Dynamic Analysis, Static Analysis, Detection Tools, Open- source tools, SQL Injection Detection, SQL Injection Types

1. INTRODUCTION

SQL injection has become one the top vulnerabilities that hackers take advantage of in web applications. Since it is number one in the OWASP Top Ten List of Web Application vulnerabilities, it is important to learn about the types of SQL injections and how to detect and prevent them. [15][6] This paper briefly explains the types and the prevention methods that all developers must use in order to ensure a secure web application. By allowing developers to become aware of this vulnerability, it will prevent future websites from being created poorly and unsecure. This paper also will give information about how to detect these SQL Injections statically and dynamically. The VEGA tool was used as the static analyzer to compare to findings from the Burp Suite tool, which was the dynamic analyzer. It is essential for developers to go through their code from the start of the process to the end just to make sure that the website does not give sensitive information. Also for the best results, using both the static and dynamic analyzer is significant.

2. BACKGROUND

2.1 What is SQL?

SQL is a certain programming language used specifically for communicating with a database. When using SQL, the developer creates statements, also called queries, to provide different functions with the database. These queries can create, select, insert, or delete from a database. There are also other functions that can be used by a developer to adjust the database.

Example 1:

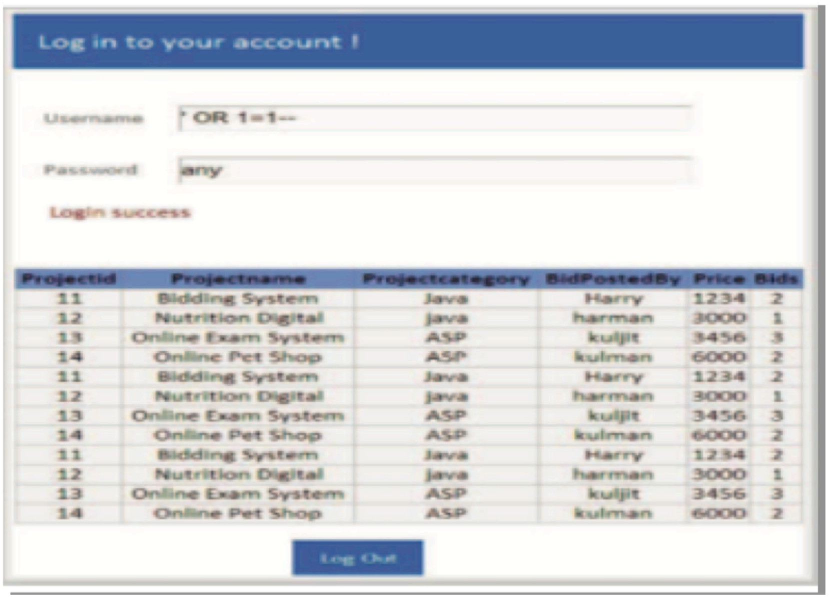
“Select * FROM Things WHERE color = ‘red’”

2.2 What is SQL Injection?

When a hacker decides to take pieces of SQL statements and place them into input fields, with an intention of getting back sensitive data that is SQL Injection. In other words, “SQL Injection is a code injection technique that obtains unauthorized access to databases in which maliciously crafted SQL statements are injected into an entry field (like textbox) for execution”. [6]

Example 2: [6]

Figure 1: SQL Injection where a hacker logs in and retrieves all the sensitive data



Log in to your account !

Username ' OR 1=1--

Password any

Login success

Projectid	Projectname	Projectcategory	BidPostedBy	Price	Bids
11	Bidding System	Java	Harry	1234	2
12	Nutrition Digital	java	harman	3000	1
13	Online Exam System	ASP	kuljit	3456	3
14	Online Pet Shop	ASP	kulman	6000	2
11	Bidding System	Java	Harry	1234	2
12	Nutrition Digital	java	harman	3000	1
13	Online Exam System	ASP	kuljit	3456	3
14	Online Pet Shop	ASP	kulman	6000	2
11	Bidding System	Java	Harry	1234	2
12	Nutrition Digital	java	harman	3000	1
13	Online Exam System	ASP	kuljit	3456	3
14	Online Pet Shop	ASP	kulman	6000	2

2.3 Why is it important to prevent SQL Injection?

SQL Injection needs to be prevented so hackers cannot get sensitive information from databases. Currently, OWASP (The Open Web Application Security Project) says that it is the number one web application vulnerability. [15][6] Considering it is at the top of the list, SQL Injection has become a vulnerability that many are trying to detect and prevent. Web Application Developers must make sure that these defects are not in their applications.

2.4 Different SQL Injection Types

2.4.1 Inference

2.4.1.1 Blind Injections

With blind injections, a developer will hide the error messages coming from inserting the wrong code by directing the user to another webpage that contains no visible code to the users' eye. Although the code cannot be seen there are ways to get these error messages for instance by using true or false SQL statements. [5]

Example 3: [5]

```
SELECT accounts FROM users WHERE login= 'doe' and 1 =0 -- AND pass = AND pin=O
SELECT accounts FROM users WHERE login= 'doe' and 1 = 1 -- AND pass = AND pin=O
```

2.4.1.2 Timing Attacks

For these attacks, the hacker will use SQL queries to figure out, from the database, the response time. Once a hacker finds out the delay reaction he/she can conclude other information. [5]

Example 4: [5]

```
declare @ varchar(8000) select @ = db_nameO if (ascii(substring(@, 1, 1)) & ( power(2, 0)))
> 0 waitfor delay '0:0:5'
```

2.4.2 Tautologies

This type of SQL injection happens when a SQL query is made to always be true by using these tautology statements. Tautology commands are inserted into the SQL statements in order for a hacker to gain sensitive information. [5]

Example 5: [4]

```
(SELECT * FROM login WHERE username=admin or 1=1--- AND password=nothing)
```

2.4.3 Illegal/ Incorrect Queries

This injection occurs when an attacker attempts to insert SQL statements and an error messages comes back with sensitive information. These error messages can give table names, what database is being used, where the error might be or other data if the developer did not secure the database and website.[5]

Example 6: [4]

```
http://localhost/?EmpId=10‘
```

2.4.4 Union Query

In this injection, a hacker attempts to attach malicious SQL query statements to the original SQL statements. The hacker would use a UNION keyword to do this and access information from the database that was not meant for them. [5]

Example 7: [5]

```
SELECT Name, Phone FROM Users WHERE Id=1 UNION ALL SELECT
creditCardNumber,1 FROM CreditCardTable
```

2.4.5 Piggy- Backed Queries

For this injection, the hacker would add “;” to the end of the original SQL query and then add another malicious query to end of it to acquire sensitive data from the database. [5]

Example 8: [5]

```
SELECT info FROM users WHERE login='doe' ANDpin=0; drop table users
```

2.4.6 Stored Procedures

This attack involves stored procedures, which are SQL statements in a program saved in part of the database and can be hacked at any time. [4] [5]

2.5 How can SQL Injection be prevented?

Security must be used from the beginning of the implementation processes of a web application. From design to the end, security has to be the main component of a web application. By doing this, we can begin to prevent any vulnerabilities, including SQL Injection.

There are also other ways that SQL Injection can be prevented while writing the code:

2.5.1 Prepared Statements

When using prepared statements, developers actually write parameterized queries. Parameterized queries pass in the parameters after the SQL code is fully written. This allows the parameters to be checked for any illegal coding that might disrupt the database and bring back sensitive data. It also would not allow the malicious SQL statements to be read as part of the SQL query code but as just a string and this can protect the database. [14]

2.5.2 Stored Procedures

Similarly to prepared statements, stored procedures use parameterized queries to make sure no malicious SQL queries are being injected. The only difference is these statements are stored in the database. When the application needs this statement it would be called from the database. [14]

2.5.3 White List Input Validation

This SQL injection prevention is recommended as a secondary guard and it can work by checking user inputs. It will check for important keywords such as table names or column names. If these were checked before getting the data from the database then it would not allow the user to get to the database by inputting any of those words. [14]

2.5.4 Escaping All User Supplied Input

Prevention using this is not fully recommended. Before entering an input into a certain query, the input is escaped. That means any symbols or characters that may be a part of malicious statements are removed before being placed into the original query. [14]

2.5.5 Least Privilege

Least Privilege describes the privileges that are given to each user using the database and web application. The administrator would be allowed more access to the database than say a regular user. This would allow less access to the database and fewer chances of an attack on the database. [14]

2.6 Burp Suite

The first tool that is used is called Burp Suite (https://portswigger.net/burp/). There are two versions of this tool, the open- source version and the professional version. The open-source version is used for this research purpose. The difference between the two versions is that a payment is needed for the professional version. Also the professional version does have more features compared to the open-source version. The open-source version consists of the intercept browser, which uses the man-in-the-middle proxy and repeater. The professional version includes a scanner, intruder and also the features in the open-source version. [7]

Figure 2: Burp Suite's Logo

2.7 VEGA

The second tool that is used is called VEGA (https://subgraph.com/vega/). This tool has only one version and it is all open-source. It contains an automated scanner, intercepting proxy and also a proxy scanner. The tool was created by SubGraph in 2014 so its fairly old and there is only one version. [8]

Figure 3: VEGA's Logo

3. CURRENT LIMITATIONS

Burp Suite is better known than VEGA by most people. But yet both still have not been tested to determine how well they work. For this thesis, testing the open-source version of Burp Suite would be the main idea. This could be helpful to other students when they need to find a tool to do testing for SQL Injection later on. The Burp Suite tool was tested previously but the professional version was used for testing purposes. Many times students cannot afford to buy the professional version so to find out how well the free version detects these vulnerabilities can be very helpful. Some of the testing that used Burp Suite before is very informative but the tests were used to find vulnerabilities other than SQL Injection. Considering SQL Injection is one of the major vulnerabilities today for web applications, getting tools to detect them so a developer can prevent them is one of the important ways to get rid of this vulnerability. Burp Suite and VEGA have never been tested and compared statically or dynamically. There has not been much research involving the tool VEGA and not much information providing details on if it should be used to do testing for any vulnerability. Research papers also provided very minimum details about this tool and if it is worth using. By testing these two tools, more information for later users can be provided.

In 2014, Kinnaird McQuade researched different types of detection tools for his paper, *Open Source Web Vulnerability Scanners: The Cost Effective Choice?* This paper showed tests for vulnerabilities in the top open- source/ low-cost detection tools, IRONWASP, ZED Attack Proxy (ZAP), Arachni, and Burp Suite Professional and compared it to high- cost tools such as Acunetix, IBM Appscan, WebInspect, and Netsparker. The author also tested the open-source/ low-cost detection tools with a web application by Oracle called Duke's Forest. The conclusion and results show that for SQL injection, Burp Suite was the only tool to detect this vulnerability. Also no scanners could detect any Cross-Site Scripting, Burp Suite and IRONWASP detected all Cleartext Credentials, Burp Suite found the most Insecure cookies, Burp Suite and IRONWASP found all session tokens in URL, Burp

Suite, ZAP, Acunetix and Arachni found password auto-enabled vulnerabilities and Acunetix, ZAP and Arachni found the most Missing anti-Cross-Site Request Forgery token vulnerability. The findings show Burp Suite can pick up on many of the vulnerabilities. [2]

In the paper, *On the Applicability of Combinatorial Testing to Web Application Security Testing: A Case Study*, the authors did similar research as in this research paper. They gathered a few vulnerable websites and tested them using two tools, Burp Suite and ZAP. The research they obtained was however for the vulnerabilities, reflected cross- site scripting and stored cross-site scripting not for SQL Injection. Also the Burp Suite version that was used is the professional one and not the open-source version. For cross-site scripting the results obtained by this research was ZAP was able to identify on average 25% more successfully than Burp Suite but they did not exactly mean Burp Suite did not implement as well as ZAP. They also cannot tell why this happened but explained it could be because each tool when generating a request uses different encoding. [3]

The research paper, *Static Analysis and Penetration Testing from the Perspective of Maintenance Teams*, also tested for vulnerabilities using Burp Suite (penetration testing) and then compared them to HP Fortify Source Code Analyzer (Static Analysis). This research project tested different kinds of vulnerabilities, not only SQL Injection. The experiment got 12 college students with a background in security to test the two tools with two java websites called Pebble and Roller. Each website was assigned two vulnerabilities that the students needed to fix by using these two tools. The conclusion from this experiment was the results from the penetration testing using Burp Suite better helped the students to fix the vulnerabilities in the website than the results coming from the static analyzer using HP Fortify Source Code Analyzer. [1]

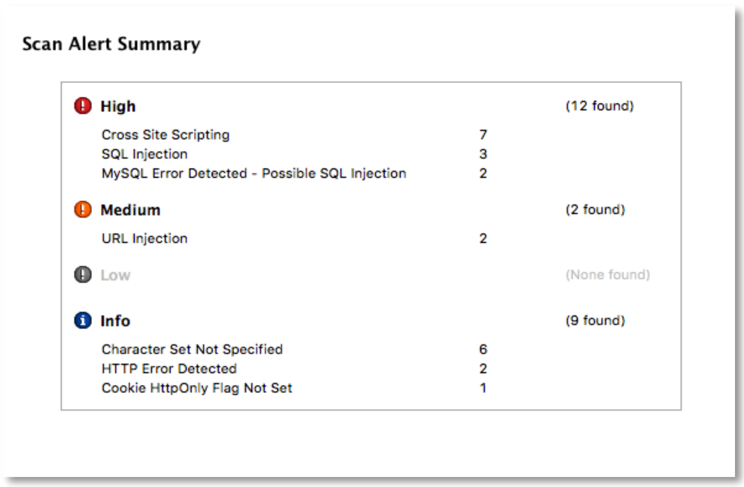
4. METHODOLOGIES

The first step to the process of testing these two tools was to find any current or previous information that these tools had with detecting the vulnerability of SQL Injection. Considering not much was found, the next step was to begin the testing stage. First phase was finding the vulnerable websites. Online there are a set of open websites that have vulnerabilities and allows a hacker (a user practicing hacking or security) to test their skills and learn about why it is important to keep web applications secure. Although hacking these websites was just to test for SQL Injections, there are other vulnerabilities that are purposely unsecure in these websites as well. Some of the vulnerable websites that were found for testing purposes and hacking practice are:

- Vicnum
- WebGoat
- Try2Hack
- Hack This Site
- JuiceShop
- Hack Yourself First
- Hellbound Hackers
- Acunetix(Forum ASP)
- HP/SpiDynamics Free Bank Online
- Altoro Mutual

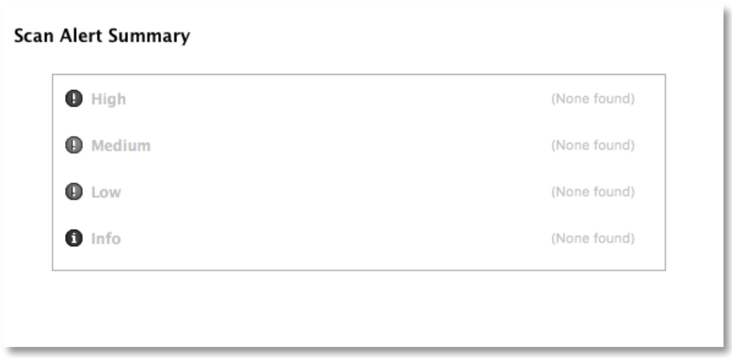
Once the vulnerable websites were found then the testing could begin. When testing these websites the static analysis was the first part of the process. For the static analysis the VEGA scanner was used. Static analysis is when the tool runs through the code checking for vulnerabilities without actually executing the program. [16] Each URL of each website was copied and inserted into VEGA for the scanner to fully go through the code of the websites. With the VEGA scanner once the process was over for each website the screen showed the amount of each of the vulnerabilities contained in the website. This part shows that there are vulnerabilities in the web application but did not give many details.

Figure 4: Vicnum’s Vulnerabilities


Scan Alert Summary

High		(12 found)
Cross Site Scripting	7	
SQL Injection	3	
MySQL Error Detected - Possible SQL Injection	2	
Medium		(2 found)
URL Injection	2	
Low		(None found)
Info		(9 found)
Character Set Not Specified	6	
HTTP Error Detected	2	
Cookie HttpOnly Flag Not Set	1	

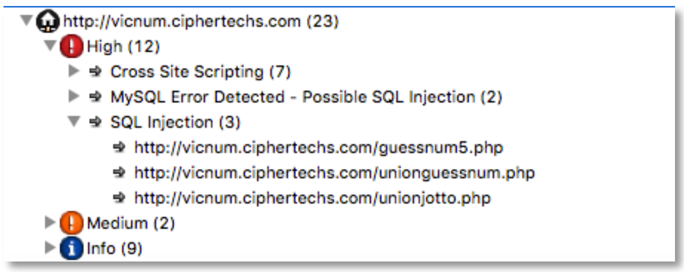
Figure 5: JuiceShop's Vulnerabilities


Scan Alert Summary

High	(None found)
Medium	(None found)
Low	(None found)
Info	(None found)

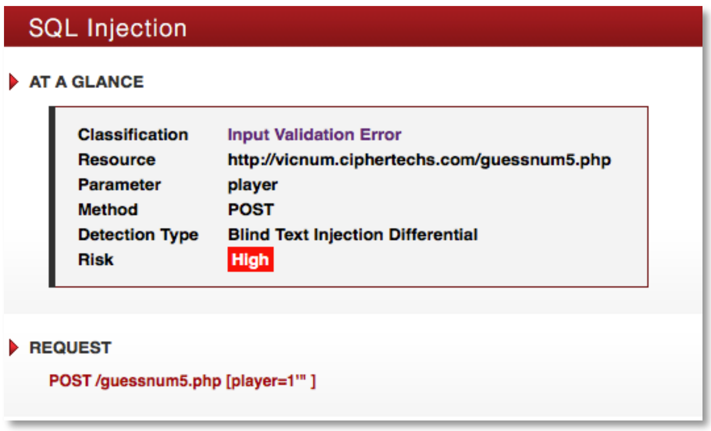
As you can see in figure 4, it shows all the vulnerabilities it scanned for and found in the web application. The vulnerability that this research focused on was SQL Injection and it did come back with three SQL Injections from that website. The next figure, figure 5, however contains no vulnerabilities, which did happen with some of the web applications. These tables also categorized the vulnerabilities from high to low depending on the seriousness of the attack on the website. Although that part did not contain much information, VEGA did provide a little more information when going into each of the vulnerabilities. This provided where the SQL Injection might be found and the type of injection it could be.

Figure 6: VEGA Scan details



```
▼ http://vicnum.ciphertechs.com (23)
  ▼ High (12)
    ▶ Cross Site Scripting (7)
    ▶ MySQL Error Detected - Possible SQL Injection (2)
    ▼ SQL Injection (3)
        http://vicnum.ciphertechs.com/guessnum5.php
        http://vicnum.ciphertechs.com/unionguessnum.php
        http://vicnum.ciphertechs.com/unionjotto.php
  ▶ Medium (2)
  ▶ Info (9)
```

Figure 7: VEGA scan of 1 SQL Injection from Vicnum



SQL Injection

AT A GLANCE

Classification	Input Validation Error
Resource	http://vicnum.ciphertechs.com/guessnum5.php
Parameter	player
Method	POST
Detection Type	Blind Text Injection Differential
Risk	High

REQUEST

POST /guessnum5.php [player=1'"]

After statically analyzing the websites, the next part was to dynamically analyze the web applications. The Burp Suite tool was used for dynamic analysis. Dynamic analysis is when the program is executed and the tool is going through each step helping to find any possible vulnerabilities. [16] This step was done by going through each page of the websites and trying different SQL injections to see where in the website was vulnerable to attacks while using the tool to help. Burp Suite helped to see different error messages and other mistakes made by programmers. This part took the longest since the process required going through each website thoroughly.

Figure 8: Hack Yourself First website page that contains a vulnerability

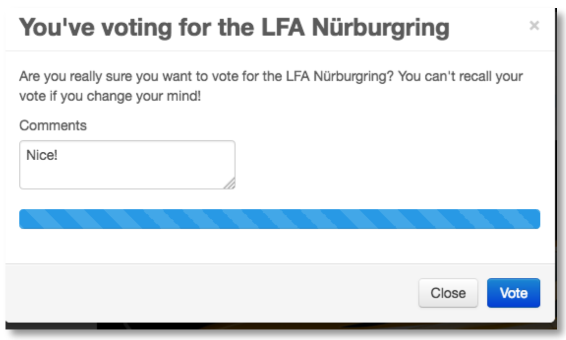
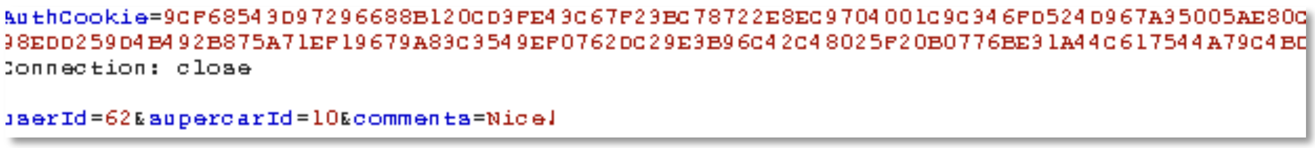

You've voting for the LFA Nürburgring
×
Are you really sure you want to vote for the LFA Nürburgring? You can't recall your vote if you change your mind!
Comments
Nice!
Close
Vote

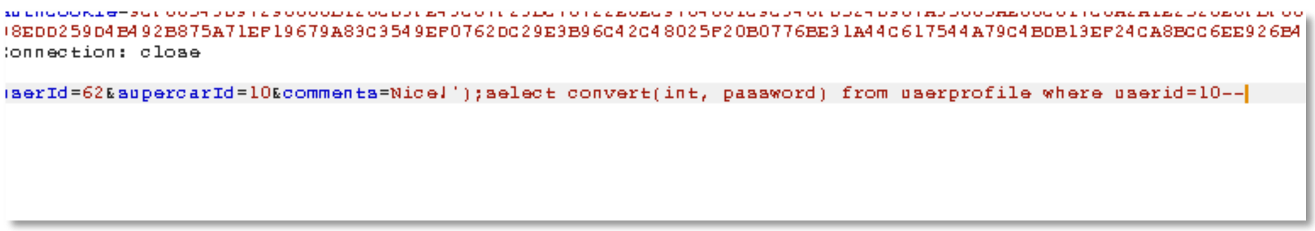
Figure 9: Intercepted Page in Burp Suite



```
AuthCookie=9CF68543D97296688B120CD3FE43C67F23BC78722E8EC9704001C9C346FD524D967A35005AE80C
98EDD259D4B492B875A71EF19679A83C3549EF0762DC29E3B96C42C48025F20B0776BE31A44C617544A79C4BD
onnection: close

serId=62&supercarId=10&comments=Nice!
```

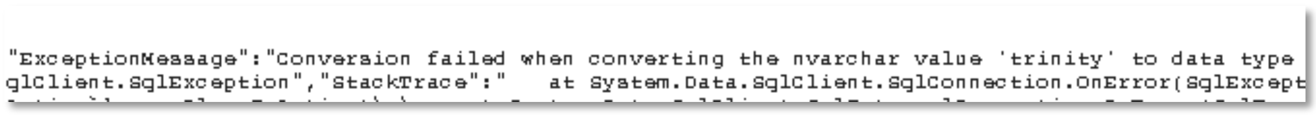
Figure 10: Adding SQL Injection to comment made on website page



```
8EDD259D4B492B875A71EF19679A83C3549EF0762DC29E3B96C42C48025F20B0776BE31A44C617544A79C4BDB13EF24CA8BCC6EE926B4
onnection: close

serId=62&supercarId=10&comments=Nice!');select convert(int, password) from userprofile where userid=10--
```

Figure 11: Error message on Burp Suite showing the password of a user



```
"ExceptionMessage":"Conversion failed when converting the nvarchar value 'trinity' to data type
qlClient.SqlException","StackTrace":"   at System.Data.SqlClient.SqlConnection.OnError(SqlExcept
```

5. RESULTS

Each website had to be first scanned using the VEGA static analysis tool and then using the Burp Suite tool, go through each web page testing for vulnerabilities. The process took a while considering for some websites the static analyzer took hours to go through the website and after to physically going through the websites dynamically took from days to weeks.

The first website Hack Yourself First took the longest to go through the static analyzer. Once the results were back the VEGA tool returned a total of 4 SQL injection vulnerabilities. The four that were returned were detected as type Blind Text Injection Differential. It showed exactly what pages in the application these attacks came from. The dynamic analysis was next. For this website, all the vulnerabilities that showed up in VEGA was also found using Burp Suite. Once those were tested, an exam through the rest of the website was done to check for other possible SQL injections. Burp Suite was able to find one other one through dynamic analysis. The one that VEGA could not detect was an injection of type Incorrect Queries.

The second website tested was Vicnum and it also took a while when using the static analyzer. After this web application was statically tested it came back with a results of 3 SQL injection vulnerabilities. These three injection vulnerabilities came back with two as a type of Blind Text Injection Differential and the last one a type of Blind Arithmetic Evaluation Differential. When going through using Burp Suite, the three detected using VEGA was also the three SQL injections detected. The SQL injections were using a search text field to get back the usernames and all the users' attempted tries to finding the answers to the games on the website Vicnum.

The third website Altoro Mutual came back with a results of 0 SQL Injection vulnerabilities from VEGA. This could not be right considering this website says from the beginning that it is vulnerable. Using Burp Suite to dynamically test the website, there were a few SQL injections that were found. The types of the SQL injections that were detected were tautologies, incorrect queries and union queries. While Vega had a hard time with detecting any injections, Burp Suite found them fairly easy.

When the fourth website Acunetix(Forum ASP) was tested the results from the static analyzer was just 1 SQL injection vulnerability. The type of this vulnerability listed by VEGA is Blind Arithmetic Evaluation Differential. Not only was this SQL injection found once doing dynamic testing using Burp Suite but another one was also found. The VEGA tool never picked up the possible SQL injection from the login page of the Acunetix(Forum ASP) website. This was a simple SQL injection of the type tautologies.

The last website statically tested was JuiceShop. This website came back with a results of 0 SQL injections from the VEGA tool. Knowing previously that this web application contained vulnerabilities, when testing dynamically using Burp Suite a few SQL

Injections did show up. The types of these SQL Injections were tautologies and Blind SQL injection. This website contained different levels of hacks so you could tell that it did involve some SQL injection. Surprisingly the VEGA tool never detected any of the vulnerabilities for this website.

Table 1: Results from Testing Tools
SQL Injections Detected

Website Name	VEGA Results (Static analyzer)	Burp Suite (Dynamic analyzer)
Hack Yourself First	4	5
Vicnum	3	3
Altoro Mutual	0	4
Acunetix(Forum ASP)	1	2
JuiceShop	0	5

6. CONCLUSION

6.1 Summary

Today, the main concern with all of this technology being used is the security part behind it all and if it is affective. With all these news reports of big companies being hacked and so many people's information being leaked, it is no wonder security is our focus. The Internet is a big part of our daily lives where most of us use it everyday so protection is key. Web applications can contain many vulnerabilities but the top one is SQL injection. Web Application developers must learn how to use security to keep information in databases secure. This paper provides information on types of SQL injections, how to prevent them and testing of two open-source testing tools. Burp Suite was used for dynamic analysis and VEGA for static analysis. To conclude these results in this paper, for most of the websites Burp Suite had a better outcome than the VEGA analyzer. VEGA did find some of the vulnerabilities as well but Burp Suite over all did a much better job detecting. Also while VEGA could only give where the vulnerability was located in the website, when using Burp Suite you can actually test out the different SQL queries to see which ones work and which do not. This provides you with much more information than the VEGA static analysis. Usually static analyzers give results with false positives but for this results with VEGA there were none. However VEGA did have many false negatives for many of the websites. The dynamic testing also produced false negatives considering there are still other SQL Injections that were not found. Burp Suite is definitely a tool that would be great for testing but VEGA would not be recommended considering the results.

6.2 Testing Difficulties

For the testing stage I came across some difficulties concerning the websites and tools. Some of the vulnerable websites are not live online so it would require setting up. I tried some of those types of websites but could not get them working on my computer so I stuck to the live websites. Then I noticed some of the websites that said you could hack into it did not allow you to use tools such as the proxy to find the vulnerabilities. Once it was connected to be intercept, the website automatically logged you off. Last problem I ran into was at the last stage. The VEGA tool has a scanner and a proxy, that allows you to do static and dynamic analysis but at the end when I went to use the proxy it would not connect in order to intercept. Because the tool has not been updated there might be some bugs that need fixing.

REFERENCES

[1] Ceccato, Mariano & Scandariato, Riccardo. Static Analysis and Penetration Testing from the Perspective of Maintenance Teams. ACM. September 2016.

[2] McQuade, Kinnaird. Open Source Web Vulnerability Scanners: The Cost Effective Choice?.2014 Proceedings of the Conference for Information Systems Applied Research. 2014 EDSIG.

[3] Garn, B., Kapsalis, I., Simos, D. & Winkler, S. On the Applicability of Combinatorial Testing to Web Application Security Testing: A Case Study. ACM. July 2014. http://dx.doi.org/10.1145/2631890.2631894

[4] Aliero, M., Ardo, A., Ghani, I., Atiku, M. Classification of SQL Injection Detection and Prevention Measure. IOSR Journal of Engineering. Vol.06, Issue 02. February 2016.

[5] Dehariya, H., Skukla, P., Ahirwar, M. A Survey on Detection and Prevention Techniques of SQL Injection Attacks. International Journal of Computer Applications. Volume 137- No.5. March 2016.

[6] Kaur, Navdeep & Kaur, Parminder. Modeling a SQL Injection Attack. IEEE. 2016 International Conference on Computing for Sustainable Global Development (INDIACom). 2016.

[7] Burp Suite: https://portswigger.net/burp/

[8] VEGA: https://subgraph.com/vega/

[9] Hack Yourself First: https://hackyourselffirst.troyhunt.com

[10] Vicnum: http://vicnum.ciphertechs.com

[11] Acunetix(Forum ASP): http://testasp.vulnweb.com

[12] JuiceShop: https://github.com/bkimminich/juice-shop

[13] Altoro Mutual: http://demo.testfire.net/default.aspx

[14]OWASP.SQLPreventionCheatSheet: https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

[15] OWASP Top 10 Web Vulnerabilities: https://www.owasp.org/index.php/Top_10_2013-Top_10

[16] Static Analysis vs. Dynamic Analysis: https://www.veracode.com/blog/2013/12/static-testing-vs-dynamic-testing