I.J.Computer Network and Information Security, **2014, 1, 26-33**
Published Online November 2013in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2014.01.04

Textual Manipulation for SQL Injection Attacks

Hussein AlNabulsi
Department of Computer Engineering, Yarmouk University, Irbid, Jordan
hsnasn@yahoo.com

IzzatAlsmadi
Department of Information Systems, Prince Sultan University, Riyadh, KSA
ialsmadi@cis.psu.edu.sa

Mohammad Al-Jarrah
Department of Computer Engineering, Yarmouk University, Irbid, Jordan
jarrah@yu.edu.jo

***Abstrac*—**SQL injection attacks try to use string or text manipulations to access illegally websites and their databases. This is since using some symbols or characters in SQL statements may trick the authentication system to incorrectly allow such SQL statements to be processed or executed. In this paper, we highlighted several examples of such text manipulations that can be successfully used in SQL injection attacks. We evaluated the usage of those strings on several websites and web pages using SNORT open source. We also conducted an extensive comparison study of some relevant papers.

***Index Terms*—**Network security, vulnerability, Intrusion detection systems, SNORT, vulnerability assessment, rule-based detection.

I. INTRODUCTION

Websites face an enormous amount of possible attacks through the Internet. Attackers may try to access a particular website for one of several possible reasons. The major reason behind such attacks includes trying to retrieve sensitive data for identity theft purposes. Websites can also be accessed for spam purposes. Spammers try to inject their links or codes in websites to get higher traffic or popularity and hence be more visible by users and search engines. Such market goal may also include trying to spy on users, their machines or websites and their search behavior in order to develop guided advertisements or marketing campaigns. Websites and machines can be also accessed by friends, relatives or lovers looking for personal sensitive information. They may be also accessed by disgruntled employee or ex-employee looking for a revenge for employer. Political or international crime reasons can also be a factor in attacking websites. Finally, some individuals may try to access websites to be popular among their rivals or to only use their skills and long available time.

Intrusion Detection is the operation of detecting actions that attempt to perform data theft, policy violations or network misuse. The Intrusion Detection System (IDS) tries to detect possible network attacks and inform network administrator accordingly. The concept of IDS was initially appeared in James Anderson's technical report (Aickelin et al 2008). This work founded the first generation of IDSs. Such systems monitor audit logs of a single machine after the intrusion. The main task of the first IDS generation is to search the audit logs for predefined patterns of a suspicious activity (Roesch 1999).

Most IDSs are reliable in detecting suspicious actions by evaluating TCP/IP connections or log files, when the IDS finds a suspicious action, it will create an alert which contains information about the source, target, and preview type of the attack.

SNORT is one of effective intrusion detection tools. SNORT is a popular Network Intrusion Detection System (NIDS) tool which is a rule-based system to identify attacks. SNORT is an open-source, lightweight IDS written by Martin Roesch in 1999. It was bought by the company SourceFire. SourceFire was then bought by the firewall giant CheckPoint in 2005. SNORT supports three protocols explicitly – TCP, UDP, and ICMP. It also supports the IP protocol.

The rest of the paper is organized as the following: Several papers relevant to the subject of this paper is presented and compared with this paper in the next section; section two. Section three presents experiments and analysis related to SQL injection and detection based on SNORT rules for detection and prevention. Paper is then concluded in section 4.

II. A COMPARISON STUDY

In this section, several related papers to the subject of this paper will be analyzed. We will compare our approach with each one of those papers in terms of: methodology, experiments or case study and findings or conclusions.

1. SNORT Rules to Detect Network Attacks.

If we evaluate the SNORT results of this paper with our methods of SQL Injections Attack, we will find that these SNORT rules did not detect all types of the SQL Injection Attacks, our methods of SQL Injection Attacks

would not detected using these SNORT rules (Dabbour et al 2013).

- Methodology: This paper is almost followed the same approach we followed in this thesis. The paper used SNORT to detect some examples of vulnerabilities related to web attacks such as SQL injections. Our approach is more comprehensive and thorough. We tried to evaluate all possible types of SQL injection attacks.
- Experiment and Case Study: In this paper, the experiment presented several SNORT rules and then evaluated their ability to detect network attacks. The Authors used SNORT IDS under Linux and they focused on the following types of attacks: SQL injection, XSS, and command execution attacks. They evaluated and testing their study using DVWA (Damn Vulnerable Web Application), they used SNORT Intrusion Detection System (IDS) and defining several examples for simulating these types of attacks, and they wrote and evaluated SNORT rules that can detect these types of attacks.

We evaluated the Precision rate and Recall rate for this paper because it is not evaluated and the results are:
The Precision Rate is = True Positive/ (True Positive + False Positive)

True Positive = 9 SQL injection detected.

False Positive = 0 websites gave a false alarm about it.

⇨ 9/(9+0) = 9/9 = 1

The Recall Rate is = True Positive/ (True Positive + False Negative)

False Negative = 37 detected all SQL injection attacks.

9/ (9+37) = 9/46 = 0.1956

- Result: These SNORT rules were good for detection but not comprehensive for detection attacks and it based on many frequent repeated processes.

SNORT RULES:

Rule number 1: [alert $EXTERNAL_NET any → $HOME_NET $HTTP_PORT (msg:"[SQL Injection attack has been detected--1]"; flow:to_server,established; pcre:"(((\?id=)(\?id\%3D))(\w*)((\')|(\%27)))/ix"; classtype:web-application-attack; sid:10000015;rev:5;)]

Rule number 2: [alert $EXTERNAL_NET any → $HOME_NET $HTTP_PORT (msg:"[SQL Injection attack has been detected--2]"; flow:to_server,established; pcre:"(((\?id=)|(\?id\%3D)).{0,}(\%3b)|(\;).{0,}((#)|(\%23)))/ix"; classtype:web-application-attack; sid:10000016; rev:5;)]

Rule 1 could not detect SQL injection attack unlike rule two that was able to detect the same attack.

Figure. 1: Results of applying Rule Number 1

The SNORT rule that successfully detects the injection attack in Figure 1 is shown in Figure 2:

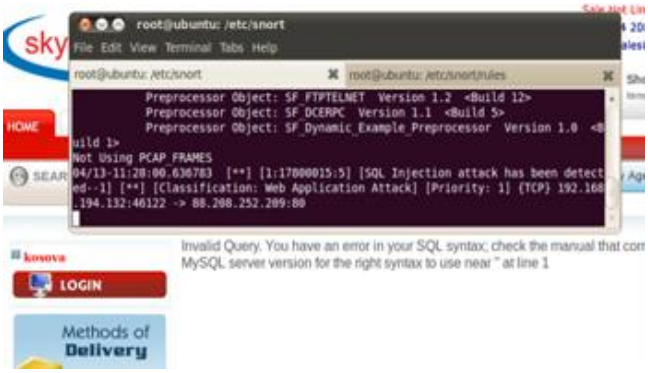

Figure. 2: Results of applying Rule Number 2

2. The Security Onion for Detecting Web Application Attacks

If we evaluate the SNORT results of this paper with our methods of SQL Injections Attack, we will find that these SNORT rules didn't detect some of SQL injection methods (Deuble 2012).

- Methodology: In this paper, the author used Security Onion Security (it is a live Xubutnu based distribution containing many of the tools required to perform the detection and prevention of these exploits). He investigates about how to Alert and detect on SQL Injection (SQLi), and how to detect Cross Site Scripting (CSS), and query Command Injection attacks on web applications, he used SNORT tool under Linux for detection these types of attacks.
- Experiment and Case Study: Weused Samurai WTF (Web Testing Framework) distribution, Damn Vulnerable Web Application (DVWA) for testing, and Security Onion instances for SNORT were configured for analyzing traffic in between the vulnerable web applications in Damn Vulnerable Web Application (DVWA), and the attacking machine Samurai WTF (Web Testing Framework). One of the main goals of the Damn Vulnerable Web Application (DVWA) distribution is to help security

professionals in testing their tools in a legal environment.

We evaluated the precision rate and recall rate for this paper because it is not evaluated and the results are:

The precision Rate is = True Positive/ (True Positive + False Positive)

True Positive = 42 SQL injection detected

False Positive = 2 websites gave a false alarm about it.

⇨ 42/(42+2) = 42/44 = 0.9545

The Recall Rate is = True Positive/ (True Positive + False Negative)

False Negative = 4 detected all SQL injection attacks.

⇨ 42/(42+4) = 42/46 = 0.9130

- Results: These SNORT rules were good for detection but there are a lot of missing alarms (false negative), and that make it harder for the analyst to know what is the actual attack.

SNORT RULES:

Rule number 3: alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"ET WEB_SERVER Possible SQL Injection Attempt UNION

SELECT"; flow:established, to_server; content:"UNION"; nocase; http_uri; content:"SELECT"; nocase; http_uri; pcre:"/UNION.+SELECT/Ui";

reference:url,en.wikipedia.org/wiki/SQL_injection; reference:url,doc.emergingthreats.net/2006446; classtype:web-application-attack; sid:2006446; rev:11;)

Rule number 4: alert http $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"ET WEB_SERVER Possible SQL Injection Attempt UNION SELECT"; flow:established,to_server; uricontent:"UNION"; nocase; uricontent:"SELECT"; nocase;

pcre:"/UNION.+SELECT/Ui"; reference:url,en.wikipedia.org/wiki/SQL_injection;

reference:url,doc.emergingthreats.net/2006446;classtype: web-application-attack; sid:2006446; rev:11;).

This SNORT rule will detect the SQL injection attack, if we wrote the (UNION) and (SELECT) words capital letters as shown in Figure 3.

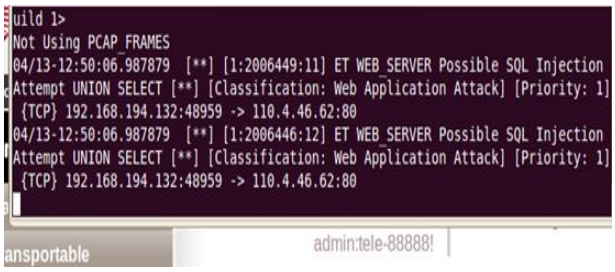

Figure. 3: Results of applying Rule Number 3

But if we wrote one of these words (union, select) in small letters it will not detect the SQL injection attack, as shown in Figure 4.

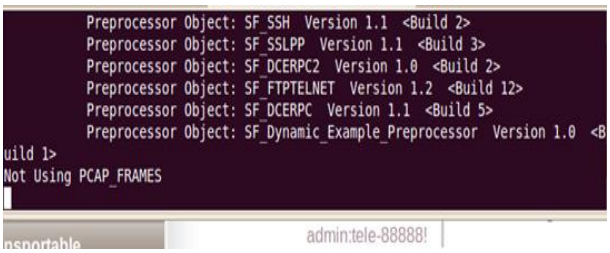

Figure. 4. Results of applying Rule Number 4

3. IDS Evasion.

If we evaluate the SNORT results of this paper with our methods of SQL Injections Attack, we will find that these SNORT rules didn't detect some of our SQL injection methods (Warneck 2007).

- Methodology: In this paper, the author used almost same approach we followed in this thesis about SQL injection attacks. The paper used many ways for Defeating the SQL Injection attack to prevent the vulnerabilities related to web attacks such as SQL injections, and the author used SNORT under Linux as IDS (Intrusion Detection System) for detecting SQL injection attacks.
- Experiment and Case Study: In this paper, the experiment presented several types of SQL injection attacks and the author defeating SQL Injection attacks using many tools that depend on the analysis, and the database level, and the web application level, and he used SNORT as IDS (Intrusion Detection System) for detecting SQL injection attacks.

We evaluated the precision rate and recall rate for this paper because it is not evaluated and the results are:
The Precision Rate is = True Positive/(True Positive + False Positive)
True Positive = 7 SQL injection detected
False Positive = 0 websites gave a false alarm about it.

⇨ 7/(7+0) = 7/7 = 1

The Recall Rate is = True Positive/(True Positive + False Negative)
False Negative = 39 detected all SQL injection attacks.

⇨ 7/(7+39) = 7/46 = 0.1521

- Results: The best way to defend against SQL injection is from Defense in Depth. There is no method that will alone defeat the SQL injection attacks, but when they combined together, they will provide a good web based application against SQL injection attacks, and the SNORT rules could detect many types of SQL injection attacks.

SNORT RULES:

Rule number 1: alert tcp any any -> $HTTP_SERVERS $HTTP_PORTS (msg: "SQL Injection SELECT statement"; flow: to_server, established; pcre:"/(s|%73|%53)(e|%65|%45)(l|%6C|%4C)(e|%65|%45)(c|%63|%43)(t|%74|%45).*(f|%66|%46)(r|%72|%52)(o|%6F|%4F)(m|%6D|%4D).*(\-\-|\/*|\#)/i"; sid: 29; rev: 3;)

Rule number 2: alert tcp any any -> $HTTP_SERVERS $HTTP_PORTS (msg: "SQL Injection attempt"; flow: to_server, established; content: "' or 1=1 --"; nocase; sid: 17; rev:1;)

Rule number 3: alert tcp any any -> $HTTP_SERVERS $HTTP_PORTS (msg: "SQL Injection attempt"; flow: to_server, established; pcre: "/(and|or) 1=1 (\-\-|\/*|\#)/i"; sid: 19; rev:2;)

Rule number 4: alert tcp any any -> $HTTP_SERVERS $HTTP_PORTS (msg: "SQL Injection SELECT statement"; flow: to_server, established; pcre:"/select.*from.*(\-\-|\/*|\#)/i"; sid: 2; rev: 1;)

Rule number 5: alert tcp any any -> $HTTP_SERVERS $HTTP_PORTS (msg: "SQL Injection UNION statement"; flow: to_server, established; pcre:"/union.*(\-\-|\/*|\#)/i"; sid: 30; rev: 8;)

Rule number 6: alert tcp any any -> $HTTP_SERVERS $HTTP_PORTS (msg: "SQL Injection UPDATE statement"; flow: to_server, established; pcre:"/update.*set.*\=.*(\-\-|\/*|\#)/i"; sid: 7; rev: 1;)

Rule number 7: alert tcp any any -> $HTTP_SERVERS $HTTP_PORTS (msg: "SQL Injection DROP TABLE statement"; flow: to_server, established; pcre:"/drop table.*(\-\-|\/*|\#)/i"; sid: 3; rev: 1;)

Rule number 8: alert tcp any any -> $HTTP_SERVERS $HTTP_PORTS (msg: "SQL Injection WAITFOR DELAY statement"; flow: to_server, established; pcre:"/waitfor delay \'[0-9]{1, 3}: [0-9]{1,2}:[0-9]{0,2}\'.*(\-\-|\/*|\#)/i"; sid: 4; rev: 1;)

Rule number 9: alert tcp any any -> $HTTP_SERVERS $HTTP_PORTS (msg: "SQL Injection SELECT statement"; flow: to_server, established; pcre:"/(s|%73)(e|%65)(l|%6C)(e|%65)(c|%63)(t|%74).*(f|%66)(r|%72)(o|%6F)(m|%6D).*(\-\-|\/*|\#)/i"; sid: 2; rev: 2;)

In this SNORT rules, the SQL injection attack will not be detected, if we don't use [#], [/*], or [--]. Because these characters are not necessary for SQL injection, the SQL injection will be successful success without them. As in Figure 5:

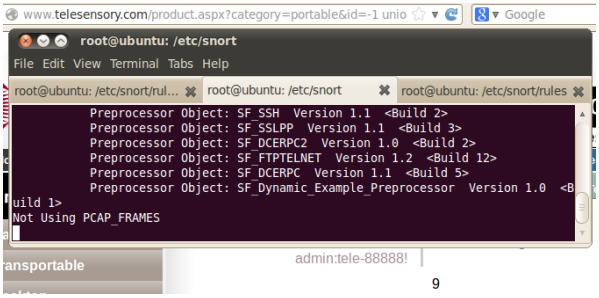

Figure. 5. Results of applying Rule Number 5

However if we use these characters [#], [/*], or [--]. The SNORT rule will detect the SQL injection attack such as in the Figure 6:

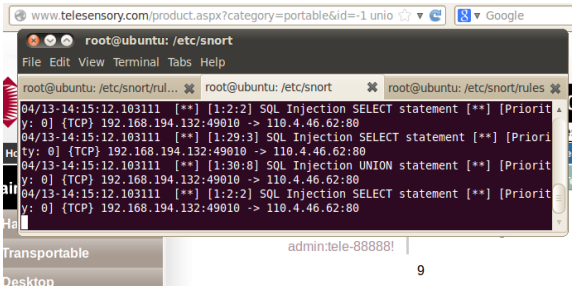

Figure. 6. Results of applying Rule Number 6

4. Regular Expressions for SQL Injection

If we evaluate the SNORT results of this paper with our methods of SQL Injections Attack, we will find that these SNORT rules didn't detect some of our SQL injection methods (Mookhey andBurghate 2010).

- Methodology: This paper used SNORT to detect some examples of SQL injection attacks and Cross Site Scripting (CSS). In our approach we tried to evaluate all possible types of SQL injection attacks, so it is more comprehensive and thorough.
- Experiment and Case Study: In this paper, the experiment presented several SNORT rules and then evaluated their ability to detect network attacks. Authors focused on the following types of attacks: SQL injection, CSS. They evaluated and testing their study by observing, they used SNORT (Intrusion Detection System) and defining several examples for simulating these types of attacks.

We evaluated the precision rate and recall rate for this paper because it is not evaluated and the results are:
The precision Rate is = True Positive/(True Positive + False Positive)
True Positive = 9 SQL injection detected
False Positive = 0 websites gave a false alarm about it.

⇨ 9/(9+0) = 9/9 = 1

The Recall Rate is = True Positive/(True Positive + False Negative)

False Negative = 37 detected all SQL injection attacks.

9/(9+37) = 9/46 = 0.1956

- Result: These SNORT rules were good for detection but didn't detect all methods of SQL injection attacks.

SNORT RULES:

Rule number 10: alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"SQL Injection - Paranoid"; flow:to_server, established;

uricontent:".pl"; pcre:"/(\%27)|(\')|(\-\-)|(%23)|(#)/i"; classtype: Web-application-attack; sid:9099; rev:5;)

Rule number 11: alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"SQL Injection - Paranoid"; flow:to_server, established; uricontent:".pl";

pcre:"/\w*((\%27)|(\'))((\%6F)|o|(\%4F))((\%72)|r|(\%52))/ix"; classtype: Web-application-attack; sid:9101; rev:5;)

In this SNORT rules, the SQL injection attack will not be detected from SNORT IDS, if the attacker did not use [‘], [--], or [#]. And the SQL injection will success if we don't use these characters, because these characters are not necessary for SQL injection, the SQL injection will success without them, and if the attacker didn't use [‘or], the SNORT IDS would not detect it. As in Figure 7:

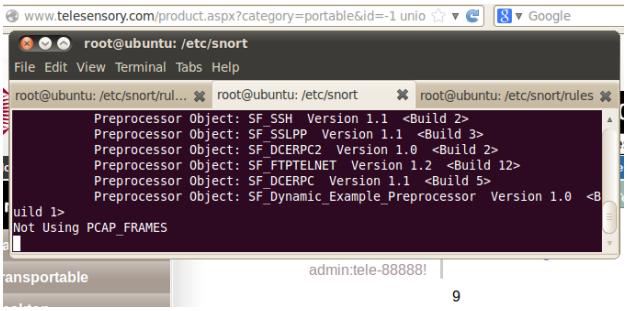

Figure. 7. Results of applying Rule Number 10

But the SNORT IDS will detect it, if the attacker use one of the following [‘], [--], [#], or [‘or]. As in Figure 8:

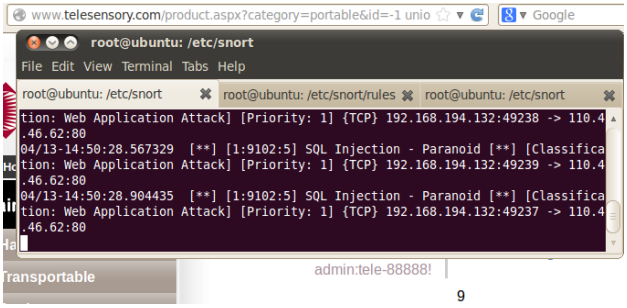

Figure. 8. Results of applying Rule Number 1

5. GreenSQL Database Firewall.

If we evaluate the SNORT results of this paper with methods of SQL Injections Attack, we will find that these SNORT rules didn't detect some methods of SQL injection attacks (Veerman andOprea 2012).

- Methodology: The paper used SNORT under Linux as an Intrusion detection system (IDS) to detect SQL injection attacks and detect SQL injection tools such as Havij, BobCat tools, Brute Force, XSS reflected, and XSS stored, Rather than constructing SQL queries by hand, these type of tools have built-in several attack methods (POST, GET, blind, cookie attack). The author used Database Protection Solutions such as GreenSQL Database Firewall, Oracle Database Firewall, and Application Security dbProtect. Building the Testing Environment For the first point we used a combination of Xen and VMware machines for testing, running Ubuntu 12.04 GNU/Linux and Microsoft Windows XP Professional SP3. He applied the latest security patches everywhere and for the victim host he used the latest versions of Apache, PHP and MySQL available in the official / default repositories: Apache 2.2.22, PHP 5.3.10 and MySQL 5.5.22.
- Experiment and Case Study: In this paper, the experiment presented several SNORT rules and then evaluated their ability to detect many examples of SQL injection attacks. They evaluated and testing their study using DVWA (Damn Vulnerable Web Application), and exploit.co.il Vulnerable Web app. They used SNORT (Intrusion Detection System) and defining several examples for simulating these types of attacks, and they wrote and evaluated SNORT rules that can detect these types of attacks.
  We evaluated the precision rate and recall rate for this paper because it is not evaluated and the results are:
  The precision Rate is = True Positive/(True Positive + False Positive)
  True Positive = 1 SQL injection detected
  False Positive = 0 websites gave a false alarm about it.

  ⇨ 1/(1+0) = 1/1 = 1

  The Recall Rate is = True Positive/(True Positive + False Negative)
  False Negative = 45 detected all SQL injection attacks.

  1/(1+45) = 1/46 = 0.0217
- Results: The result of the research is mixed. They found that it could use SNORT to detect the mostly of SQL injection attacks, but it is also concluded that it is not very easy against such SQL injection attacks. The SNORT's default

rules are not enough to detect some of SQL Injection attacks.

SNORT RULES:

Rule number 12: alert tcp any any -> any $HTTP_PORTS (msg:"SQL Injection - StartW - ebcruiser"; content: "%27%20and%201=2%20union%20all%20select%201,1"; classtype: Web-application-attack; sid:9301; rev:1;)

Rule number 13: alert tcp any any -> any $HTTP_PORTS (msg:"SQL Injection - StartS - QLmap"; pcre:"/(%27%20UNION%20ALL%20SELECT%20NULL%2CNULL%2C)/i"; classtype: Web-application-attack; sid:9302; rev:1;)

Rule number 14: alert tcp any any -> any $HTTP_PORTS (msg:"SQL Injection - StartT - The Mole"; content: "%27%20and%201%3D0%20UNION%20ALL%20SELECT%200%2C1%2CCONCAT%28"; classtype: Web-application-attack; sid:9303; rev:1;)

Rule number 15: alert tcp any any -> any $HTTP_PORTS (msg:"SQL Injection - StartH - avij"; content: "%27+union+all+select+"; classtype: Web-application-attack; sid:9304; rev:1;)

In this SNORT rules, the SQL injection attack will be detected, if the attacker write the SQL injection attack at this order [‘ union all select], and the SNORT rule detect it as shown in Figure 9:

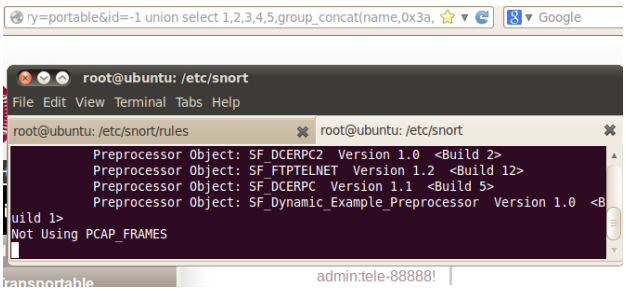

Figure. 9. Results of applying Rule Number 2

Also it will not be detected, if the attacker did not write the SQL injection in previous way such as [union select]. As shown in Figure 10:

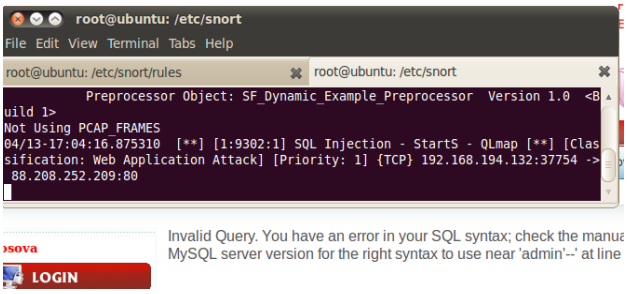

Figure. 10. Results of applying Rule Number 13

Here, it is the summary table of comparing our result of study with the other works in the same field of study.

Table 1: Summary Table for Comparing Study

	Authors names of the paper	Precision Rate	Recall Rate
1.	Dabbour, Alsmadi, and Alsukhni, 2013.	1	0.19
2.	Deuble, 2012	0.95	0.91
3.	Warneck, 2007	1	0.15
4.	Mookhey, and Burghate, 2007	1	0.19
5.	Veerman, and Oprea, 2012	1	0.02
6.	Our final SNORT rule	1	1

We can find from the table 20, that the papers Dabbour, Alsmadi, and Alsukhni, 2013, Warneck, 2007, Mookhey, and Burghate, 2007, Veerman, and Oprea, 2012 gave us a good value of precision rate that equal 1 which mean that the false positive (false alarm) equal 0, but the recall rate is very low which mean that the false negative (failure to detect attacks) is very high. In the other side, we can see that the paper Deuble, 2012 gave us a good values for the precision rate = 0.9545 and the recall rate = 0.9130, which mean that it gave us a good values in the false positive (false alarm) is low, and the recall rate is high which mean that the false negative (failure to detect attacks) is low. But we can see that the our final SNORT rule gave us the best values in the precision rate which equal 1, and the recall rate which equal 1, which mean that the false positive (false alarm) is equal 0, and the false negative (failure to detect attacks) is equal 0.

IV. EXPERIMENT AND ANALYSIS

In this section several experiments will be conducted. A case study of one or more websites will be assembled. We will try to evaluate vulnerabilities based on the different types and classes of SQL injection attacks. The next step will be then using SNORT and evaluate ability of different rules adding to SNORT setting to see their ability of detecting attacks.

We will also develop SNORT rules to detect against SQL injection attacks.

There are general ways of capturing SQL injections since it is not common to use the following ASCII values (in Table 2) with their corresponding Hexadecimal values in an HTTP GET function. Such symbols can be used in SQL injection attacks. One of the problems is that the chances become bigger on giving false-positives (false alarms). There are some of the general SQL attributes shown below which can be used to capture SQL injection requests. Some are also in the form of hexadecimal which is seen on the Table 2 (Veerman and Oprea 2012).

Table 2: General SQL injection symbols or keywords which can be used by attackers (Veerman and Oprea 2012).

Hexadecimal	ASCII or Meaning	Hexadecimal	ASCII or Meaning
%22	"		union
%27	’		select
%20	Space	%4f%52, %6f%72	OR, or
%3D	=		All
%23	#		concat

We will calculate the precision rate and recall rate for each rule using all of SQL injection examples that we used and anther websites to detect the false positive.

The count 46 examples of SQL injection examples that we used, and we added 107 normal websites examples. These normal websites examples are shown in table 3:

Table 3: The normal websites examples

http://www.likeboot.com/Pages.asp?id=23
http://www.like.com.my
http://us.mc1645.mail.yahoo.com/mc/welcome?.-gx=1&.tm
https://wumt.westernunion.com/info/homePage.asp?-country=JO&origination=US

See the rest of the normal websites examples table in appendix (G).

Then, we calculated the Precision Rate and Recall Rate for SNORT rules.

1. The Precision Rate and Recall Rate for the First and Third SNORT rules is = True Positive/(True Positive + False Positive)

True Positive = 46 SQL injection detected.
False Positive = 10 websites gave a false alarm about it.

⇨ 46/(46+10) = 46/56 = 0.8214

The Recall Rate for the First and Third SNORT rules is = True Positive/(True Positive + False Negative)
False Negative = 0 detected all SQL injection attacks.

⇨ 46/(46+0) = 46/46 = 1

2. The Precision Rate for the Second and Forth SNORT rules is = True Positive/(True Positive + False Positive)

True Positive = 36 SQL injection detected
False Positive = 10 websites gave a false alarm about it.

⇨ 34/(34+10) = 34/44 = 0.7727

The Recall Rate for the Second and Forth SNORT rules is = True Positive/(True Positive + False Negative)
False Negative = 12 SQL injection attacks couldn’t detect.

⇨ 34/(34+12) = 34/46 = 0.7391

3. The Precision Rate for the Fifth SNORT rule is = True Positive/(True Positive + False Positive)

True Positive = 46 SQL injection detected
False Positive = 12 websites gave a false alarm about it.

⇨ 46/(46+12) = 46/58 = 0.7931

The Recall Rate for the Fifth SNORT rule is = True Positive/(True Positive + False Negative)
False Negative = 0 detected all SQL injection attacks.

⇨ 46/(46+0) = 46/46 = 1

4. The precision Rate for the Sixth SNORT rule is = True Positive/(True Positive + False Positive)

True Positive = 46 SQL injection detected
False Positive = 2 websites gave a false alarm about it.

⇨ 46/(46+2) = 46/48 = 0.9583

The Recall Rate for the Sixth SNORT rule is = True Positive/(True Positive + False Negative)
False Negative = 0 detected all SQL injection attacks.

⇨ 46/(46+0) = 46/46 = 1

5. The precision Rate for the Seventh SNORT rule is = True Positive/(True Positive + False Positive)

True Positive = 46 SQL injection detected
False Positive = 0 websites gave a false alarm about it.

⇨ 46/(46+0) = 46/46 = 1

The Recall Rate for the Seventh SNORT rule is = True Positive/(True Positive + False Negative)
False Negative = 0 detected all SQL injection attacks.

⇨ 46/(46+0) = 46/46 = 1

6. The precision Rate for all of these previous SNORT rules is = True Positive/(True Positive + False Positive)

True Positive = 46 SQL injection detected
False Positive = 22 websites gave a false alarm about it.

⇨ 46/(46+22) = 46/68 = 0.6764

The Recall Rate for all of these previous SNORT rules = True Positive/(True Positive + False Negative)
False Negative = 0 detected all SQL injection attacks.

⇨ 46/(46+0) = 46/46 = 1

Applying the SQL Injection Attacks on *Damn Vulnerable Web Application*(DVWA):

Damn Vulnerable Web Application (DVWA) is an available vulnerable web application. We will use it for testing the possible SQL injection attacks from outside because it is working as a web server and you can build your own web server using it. The command will be executed as the following: (Notice the three used symbols (;), (|), and (&)) (Dabbour et al 2013).
[192.168.194.132|cmd], [192.168.194.132;cmd], [192.168.194.132&cmd].
input-output PCB. This supports a number of experiments on computer interfacing. The 68HC11 Evaluation Board and the Input/Output Board are shown in Fig. 6.

V. CONCLUSION

In this paper, we focused on testing the usage of several strings or characters to illegally access websites or their databases under what is called SQL injection attacks. Those strings may successfully access some websites and not necessary some others. This may depend on several factors related to security and authentication in the database, website, network or even operating system.

We evaluated the usage of several examples of those strings on different web pages or websites. Those can be used to retrieve data: login, password, account information, etc. They maybe also used to delete or drop tables or databases. It should be also mentioned that the successful intrusion based on those manipulated strings are not dependent on particular websites, programming languages or database management systems. They can be all subjective to such attacks almost all equally likely.

REFERENCES

[1] U Aickelin, J Twycross and T HeskethRoberts, "Rule Generalisation using Snort", *International Journal of Electronic Security and Digital Forensics (IJESDF*), April 2008.

[2] Martin Roesch, "Snort — Light Weight Intrusion Detection for Networks", *Proceedings of LISA '99: 13th Systems Administration Conference*, November 1999.

[3] Mohammad Dabbour, IzzatAlsmadi and EmadAlsukhni," Efficient Assessment and Evaluation for Websites Vulnerabilities Using SNORT", *International Journal of Security and its Applications IJAST*, Vol. 7, No. 1, January 2013.

[4] Ashley Deuble, "Detecting and Preventing Web Application Attacks with Security Onion", *SANS Institute*, 26th July 2012.

[5] Brad Warneck, "Defeating SQL Injection IDS Evasion", *SANS Institute*, January 4th 2007.

[6] K. K. Mookhey, NileshBurghate, "Detection of SQL Injection and Cross-site Scripting Attacks", *SecurityFocusInfocus article*, Created March 2004, Updated Nov 2010.

[7] GerrieVeerman, RazvanOprea, "Database SQL Injections Detection & Protection", *University van Amsterdam*, May 30, 2012.

Hussein AlNabulsiis a recent master graduate from department of computer engineering at Yarmouk University. His research interests are largely in networks security

IzzatAlsmadiis an associate professor in the department of information systems at Prince Sultan University in KSA. He obtained his Ph.D degree in software engineering from NDSU (USA). His second master in software engineering from NDSU (USA) and his first master in CIS from University of Phoenix (USA). He had B.sc degree in telecommunication engineering from Mutahuniversity in Jordan. He has several published books, journals and conference articles largely in software engineering and information retrieval fields.

Mohammad AlJarrahis an associate professor in the department of computer engineering at Yarmouk University in Jordan.

How to cite this paper: Hussein AlNabulsi, Izzat Alsmadi, Mohammad Al-Jarrah,"Textual Manipulation for SQL Injection Attacks", IJCNIS, vol.6, no.1, pp.26-33,2014. DOI: 10.5815/ijcnis.2014.01.04