ISSN 1330-3651 (Print), ISSN 1848-6339 (Online)
DOI: 10.17559/TV-20150310110158

IMPLEMENTATION AND ANALYSIS OF WEBSITE SECURITY MINING SYSTEM, APPLIED TO UNIVERSITIES' ACADEMIC NETWORKS

Ying-Chiang Cho

Original scientific paper
It is becoming increasingly common for web application and data storage services to be handled by cloud computing; therefore, more and more people are putting their private information on the internet, motivating research into cloud computing, database security and authority encryption. In the Open Web Application Security Project (OWASP) assessment, SQL injection is one of the most dangerous attack vectors in internet security. With this in mind, we have implemented a system named the website security mining system, which leverages a web crawling algorithm to analyze web URL and e-mail address leaks through black-box testing of 20 well-known universities' websites. Based on our data, academic website maintainers can be clearly informed about what kind of danger they are exposed to, which URLs are highly in danger, and the need to patch the website to protect against vulnerabilities and prevent academic resources from attacks. We hope that in the future, academic networks will gain more attention in the information security community, just like commercial and government networks today.

Keywords: *academic networks; black-box testing; database security; search engine; SQL injection*

Uvođenje i analiza sustava za probijanje sigurnosti web mjesta s primjenom na sveučilišne akademske mreže

Izvorni znanstveni članak
Sve je uobičajenije za web aplikacije i poslužitelje za pohranu podataka rukovanje putem programskog rješenja u oblaku; stoga je sve veći broj ljudi koji svoje privatne podatke stavljaju na internet, motivirajući istraživanje mogućnosti programskog rješenja u oblaku, sigurnosti baza podataka i kodiranih nadležnosti. U procjeni Open Web Application Security Project (OWASP)-a, ubacivanje SQL-a jedan je od najopasnijih napadnih vektora na sigurnost interneta. Imajući to u vidu, uveli smo sustav nazvan sustav za probijanje sigurnosti web mjesta, koji pokreće algoritam za pretraživanje weba kako bi analizirao propuste na zaštiti URL-a i adresa e-pošte ispitivanjem crnih kutija web mjesta 20 poznatih sveučilišta. Na temelju naših podataka, održavatelji akademskih web mjesta mogu saznati kakvoj su opasnosti izloženi, kojim URL-ovima prijeti veća opasnost i što učiniti kako bi uredili web stranicu za zaštitu od ranjivosti i sprijećili napade na akademske resurse. Nadamo se da će se u budućnosti veća pažnja posvetiti sigurnosti informacija na akademskim mrežama, kako se to danas čini s komercijalnim i vladinim mrežama.

Ključne riječi: *akademske mreže; sigurnost baza podataka; stroj za pretraživanje; testiranje crne kutije; ubacivanje SQL*

1 Research motivation

Due to the broad availability of web applications, securing a web application (particularly one backed by a database) is much more challenging than securing a traditional application [1]. Some security researchers have attempted to convey the concepts of information security to web application programmers [2]. However, these concepts are often not successfully applied. After all, human thoughts are much more uncontrollable [3, 4] than computer software; therefore, humans still need to use external programs to help analyze security issues in source code, web application services, and even the entire operating environment. For example, some researchers have used bounded model checking to analyze web application security [5] but this approach still fails to account fully for the dynamic environment that real web applications must deal with. SQL injection attacks have been known for decades, but due to improper handling of user input, these attack modalities are still leveraged towards nefarious purposes such as executing arbitrary database queries to, for example, delete data or steal user account information [6].

Academic networks are typically smaller and simpler than commercial and government networks, where the issue of information security is given more attention. However, academic networks still face serious risks. Commercial networks contain much private data and government networks contain government documents, which motivates strong security measures as compared to academic networks where malware attacks are more common [7, 8]. However, academic networks do contain valuable information, such as proprietary academic material and private personal data.

This research applies web crawling and keyword searching, the core elements of a search engine, with automated vulnerability detection to implement the Website Security Mining System (WSMS), which we applied to websites of the top 20 universities (ranked by Quacquarelli Symonds) [9]. Comparing the difference among these well-known universities' security protection schemes reveals some interesting phenomena which deserve more attention in the web security community.

2 Introduction of the core system and techniques

The WSMS is designed to combine search engine technology with vulnerability testing to automatically spider and assess the security of a target website. Below, the key technologies involved are discussed: search engines, web vulnerability mining, and SQL injection attacks.

2.1 Search engines

Although there are differences in the designs of every search engine, the key components can be broken into four categories: Web Crawler, Indexer, Searcher and User Interface. The detailed functions are as follows:

- Web Crawler: A web crawler moves throughout the internet, collecting web pages and metadata for use in building the other components of a search engine

listed below. A web crawler typically starts at one URL and then follows links on that web site to find other URLs. Because content can rapidly change, a web crawler needs to repeatedly revisit sites that it has already visited previously.

- Indexer: An indexer analyzes web pages that have been downloaded by the web crawler, then sifts out useless information and keeps useful information to make a query index, which allows for fast information retrieval.
- Searcher: A searcher uses the query index database created by the indexer to determine which results to display to the user. Searchers can be based on a variety of metrics such as quality, relevance, and popularity.
- User Interface: A search engine is of little use if users cannot access it. Therefore, a user interface is necessary to allow the user to query the search engine and to display results in a human-friendly manner. A search engine's basic framework is depicted as Fig. 1.

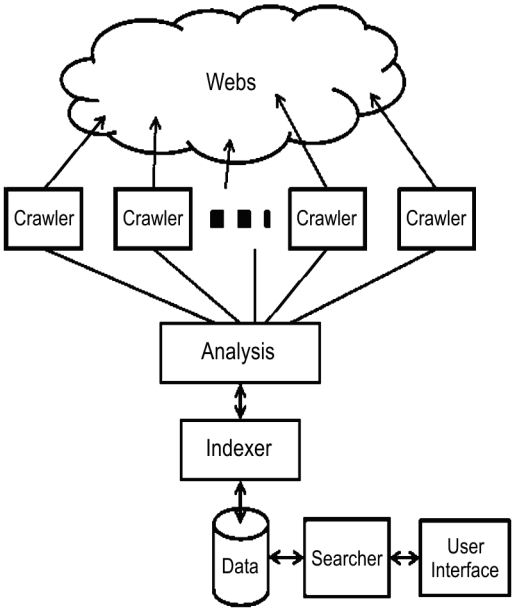

Figure 1 Framework for a search engine

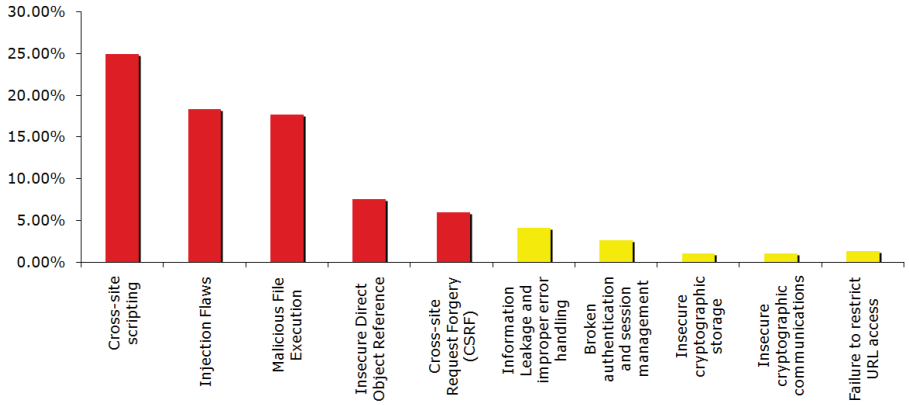

Figure 2 Top ten web application threats

2.2 Web vulnerability mining

According to OWASP's [53, 54] statistics in 2010, SQL injection attacks are one of the most dangerous attack vectors among web applications' top ten security threats, accounting for 18 % of all attacks. These statistics are shown in Fig. 2 [53].

A weakness in a system's security is also called a vulnerability, which has been defined in RFC2828 [16] as following: "A flaw or weakness in a system's design implementation or operation and management that could be exploited to violate the system's security policy".

There are two ways to analyze vulnerabilities:

- Static Analysis: Analyzes a web page's source code to attempt to find vulnerabilities. If one can control all of a program's behaviors, then one can find all the possible interactions between user input and internal program logic, which can reveal functional parameters that may have problematic security properties. By comparing known classes of vulnerabilities against source code, static analysis can identify potential flaws for developers to repair [17].
- Dynamic Analysis: Identifies security problems by directly interacting with a functioning web site. In other words, dynamic analysis relies on simulating user interactions with web pages, including interactions designed with potentially malicious intent. Because dynamic analysis uses a real website to find vulnerabilities in real time, found vulnerabilities are much more likely to be real than with static analysis, which has problems with detecting false positives [18, 19].

Based on the above two general analysis technique categories, many more specific analysis methods have been created, some of which even combine static and dynamic analysis [20]. For instance, black box testing, fuzz testing and penetration testing are all growing increasingly common.

Black box testing determines whether a web application has vulnerability by inputting testing data to the application and analyzing its response [21], as opposed to white box testing which focuses on source code parsing and analysis. White box testing tends to have lower efficiency because it does not factor in the dynamic interplay between the web server, application server, and database server [22]. Therefore, it is more common to use black box testing to more holistically analyze web application's vulnerability [23]. Fuzzing [24] is automatic software testing technique that is based on vulnerability

injection which uses lots of invalid or even random data as input to analyze whether vulnerabilities can be detected in these abnormal scenarios. Fuzzing does not have the False Positive problem caused by the static analysis, and it also does not need lots of people to do the reverse engineering because it can be highly automated. Therefore, fuzzing is a technique that has high efficiency and low cost [14, 15]. Fuzzing is widely used. Many companies and organizations use it to improve the quality of software; vulnerability analyzers use it to discover and report vulnerabilities; and hackers use it to discover and exploit vulnerabilities. Penetration testing is a method to estimate the security of computer system or internet security by actively simulating attacks [24, 25]. This method analyzes all possible injection weaknesses in the system, so the testing result is very valuable and convincing. The end product is not simply potential vulnerabilities, but verified vulnerabilities and exploits. Honest testing result can form a bridge between developer and information security communities [26, 27]. The WSMS was created by combining several of these concepts discussed above.

2.3 SQL injection attacks

SQL injection attacks [28] take advantage of the process of web applications accessing databases with queries based on improperly-validated user input. The WSMS finds SQL injection attacks which can bypass firewall and identity authorization to control the database [29]. SQL injection can penetrate any type of database that relies on SQL, regardless of whether the underlying web application is written in ASP, PHP or JSP as long as the program has a severe yet common logic error. Although there are well-known techniques to combat SQL injection attacks [30, 31], they are still quite common and therefore there has been much interest in developing methods to inspect web applications and detect these vulnerabilities [29, 32, 33].

3 System implementation

In order to inspect whether information stored on the web presents a security risk, this research combines a web crawler, like those used in search engines, with the concept of application vulnerability inspection, specifically black box and penetration testing. The end product is the WSMS, a tool to evaluate a website's security [36, 37]. This system can be separated into two main modules which are the Static Mining Module and the Dynamic Scanning Module. The Static Mining Module inspects a specific website's robots.txt, e-mails, potential SQL injection URLs, files, and broken links. The Dynamic Scanning Module uses the system's vulnerability-inspecting function by typing keywords into a search engine's query box to inspect many websites.

Both of the Static Mining and Dynamic Scanning modules can lever the system's vulnerability inspecting function, which has two parts: known website vulnerability inspection and SQL injection inspection. The former compiles a database of open source website vulnerabilities into an XML file which is used to inspect the website to see whether it has the same vulnerability. Fig. 3 is the format of an XML file. The bug file parameter is a base64 hash and other parameters are converted from the open source website vulnerabilities database. Our system updates its vulnerability database by adopting new vulnerabilities that have been announced on the Exploit Database regularly [34]. By updating the vulnerability database, we can ensure that the vulnerability samples are always updated, similar to how antivirus software regularly updates its virus database.

```
<?xml version="1.0" encoding="utf-8"?>
<config>
    <name>Name of Vulnerability</name>
    <date>Releasing Date of Vulnerability</date>
    <author>Author</author>
    <version>Version Number</version>
    <type>Type of Vulnerability</type>
    <description>Description of Vulnerability</description>
    <file>File of Causing Vulnerability </file>
    <bugfile>The URL that used for Vulnerability Testing</bugfile>
    <successkeyword>Successful keyword shown on the page after error
appears</successkeyword>
</config>
```

Figure 3 XML Format

The WSMS can find vulnerabilities in a variety of database engines, specifically MS-SQL, MySQL, Access, Oracle, Postgresql, and Sqlite. The steps to identify SQL injection vulnerabilities are as follows. First, an injectable point must be identified by inspecting the website for places where user input may be used in SQL queries. If such an injectable point is found, then further tests are conducted to identify the specific type of database engine. To do this, we take advantage of how different databases use different function names for certain tasks. For example, MS-SQL and MySQL use len( ) to calculate length, while Oracle uses length() to do this. In other words, when you use len('s')=1 to test and receive a normal response, the target website uses MS-SQL or MySQL. On the other hand, if this does not work, then the database might be Oracle or other database type. There are several other functions that can help us determine what the database is. After getting the database's type, we can obtain table and column names and finally get the database's data.

The WSMS can run on any operating systems which are supported by Java. We describe the two basic modules in more detail below.

3.1 Static mining module

The Static Mining Module runs depth mining on a specific website. There is an option to determine whether you want to follow the website's robot.txt rules. Robot.txt [35] is an ASCII-encoded file stored in the website's root directory that lists files that can and cannot be accessed by search engine crawlers. There is no official standards body or RFC for the robots.txt format. It was created by consensus and ultimately serves only as a

recommendation for crawlers, so it cannot protect the website's private data completely. Other functions of the Static Mining module are identifying e-mail information, potentially injectable URLs, downloadable files, and broken links, which may contain private information.

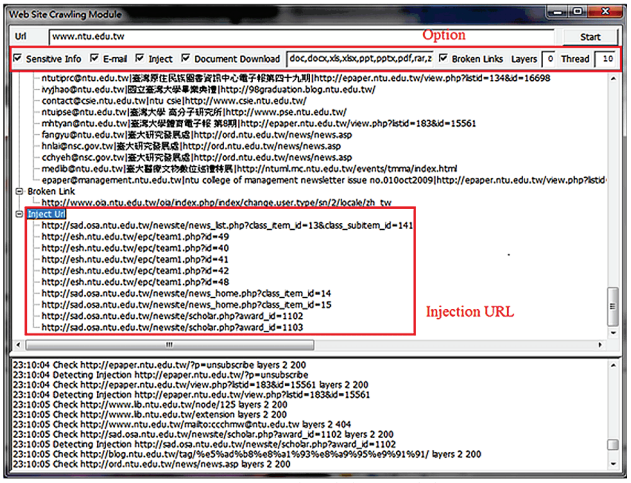

Figure 4 Static mining module

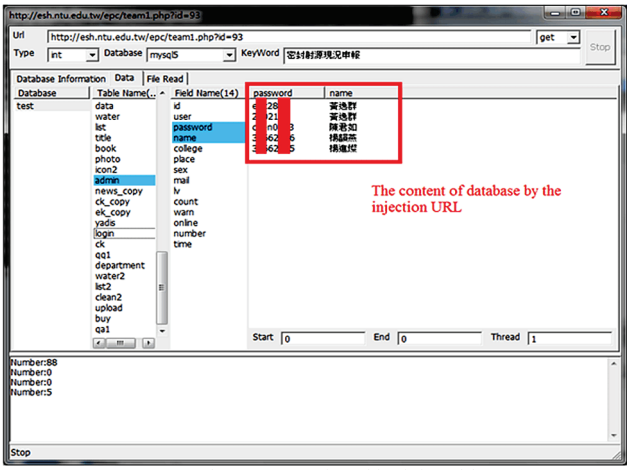

Figure 5 Database content found by injectable URLs

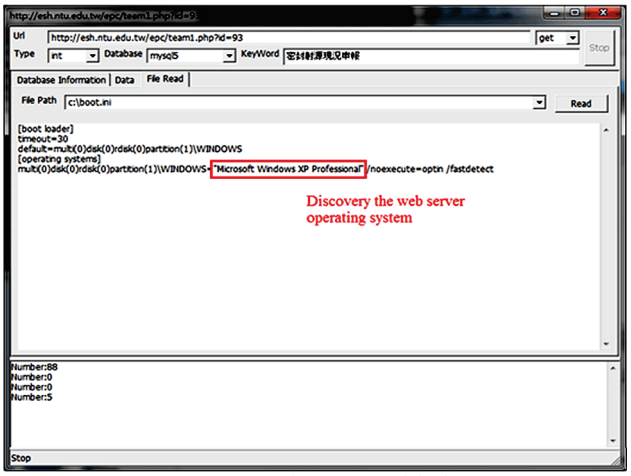

Figure 6 Operating system of the host

The Static Mining Module starts with a specific web site and then collects all the related pages from it using a breath-first search algorithm. The system assumes that web pages have close relations to the original web page if the link distance is within a certain range [38, 39], so it will fetch all links inside the original page then iterate through all of those URLs to fetch all links within them. This type of method can be easily parallelized to improve fetching speed. After files are downloaded by the web crawler, an HTML parser process extracts pages' URLs and then adds it into the URL queue. Also, the system will call vulnerability inspecting process to inspect URLs, checking whether it has potential vulnerabilities or not.

Fig. 4 shows the process of mining a college's website [40] by our system. Several injectable URLs were found and by exploiting these vulnerabilities we were able to retrieve the database information shown in Fig. 5.

Additionally, we determined that the operating system (OS) of the host was "Microsoft Windows XP Professional", as shown in Fig. 6, which could open up the possibility for further OS-based exploits.

3.2 Dynamic scanning module

The most popular search engines today are Google, Yahoo, Baidu, Microsoft Bing, NHN, eBay, Ask.com, Yandex and Alibaba. With the help of search engines, we can find billions of web pages and their URLs. Our system inspects these websites to determine whether they have vulnerabilities by analyzing the results retrieved from search engines. Our system supports the kinds of query syntaxes used in modern search engines. After you input the keywords, the system can find all related web pages and inspect whether they are at risk for vulnerabilities or not.

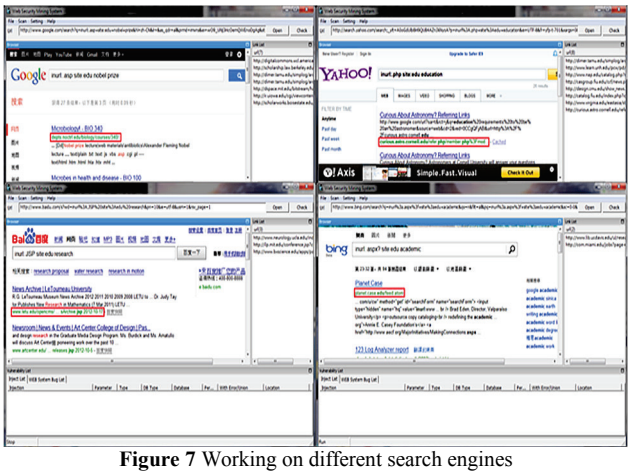

Figure 7 Working on different search engines

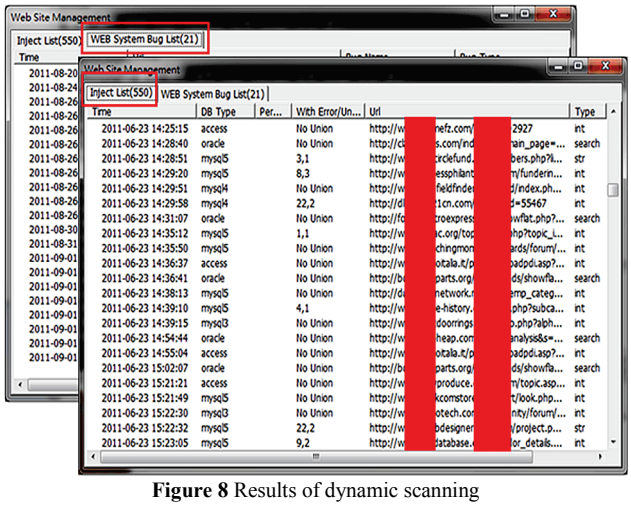

Figure 8 Results of dynamic scanning

Fig. 7 shows the different query syntaxes used in different search engines. For Google, it is "inurl:.asp? site:edu nobel prize"; Yahoo is "inurl:.php? site:edu education"; Baidu is "inurl:.jsp? site:edu research"; Bing is "inurl:.aspx? site:edu academic".

This research used the same command, inurl:.asp? |.jsp?|. php?|.aspx?site:com new, to search the ten most popular search engines. 800 web pages were retrieved from each search engine. We found 550 SQL injectable URLs and 21 known website vulnerabilities out of this total of 8000 web pages, which are shown in Fig. 8 below. This highlights the fact that SQL injection problems are still very severe on the internet.

4 Real experimental analysis

This research tested websites of the top 20 universities based on the Quacquarelli Symonds 2012 ranking by applying the Static Mining Module to gather e-mail addresses and find injectable URLs. This analysis was done on a single computer running Windows XP on an Intel Core I3-540 processor and 1 GB of RAM. Each university's website was allotted a maximum of 72 hours for analysis, although some analyses terminated before the time limit.

Tab. 1 shows the number of e-mail addresses and injectable URLs found after 72 hours' mining the websites of 20 universities. Six universities exposed over ten thousands e-mail addresses and nine universities had URLs that could be injected. In total, there were 152523 e-mail addresses and 34 injectable URLs detected in this experiment. Fig. 9 below shows an example of the direct output of the system in terms of e-mail addresses, injectable URLs, and broken links.

Table 1 Statistics of mail numbers and Injectable URLs in every university

School	Website	Number of Mail	Number of Injectable URL
Massachusetts Institute of Technology (MIT)	web.mit.edu	12122	0
University of Cambridge	www.cam.ac.uk	6075	1
Harvard University	www.harvard.edu	8081	0
University College London (UCL)	www.ucl.ac.uk	4589	4
University of Oxford	www.ox.ac.uk	15324	2
Imperial College London	www3.imperial.ac.uk	978	0
Yale University	www.yale.edu	9056	7
University of Chicago	www.uchicago.edu	2888	0
Princeton University	www.princeton.edu	7434	1
California Institute of Technology (Caltech)	www.caltech.edu	6822	2
Columbia University	www.columbia.edu	16732	4
University of Pennsylvania	www.upenn.edu	7967	1
ETH Zurich (Swiss Federal Institute of Technology)	www.ethz.ch	778	2
Cornell University	www.cornell.edu	545	5
Stanford University	www.stanford.edu	8733	9
Johns Hopkins University	www.jhu.edu	10067	4
University of Michigan	www.umich.edu	310	1
McGill University	www.mcgill.ca	5056	10
University of Toronto	www.utoronto.ca	11746	14
Duke University	www.duke.edu	17427	6

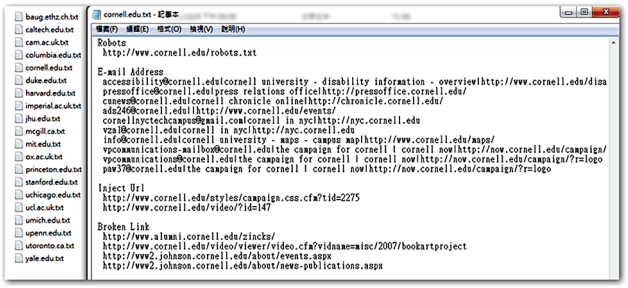

Figure 9 Program output, showing-mail address, injectable URLs, and broken links

The data from Tab. 1 is further summarized in Fig. 10 and 11 below, showing the distributions of e-mail and injectable URL counts by university. From these two figures, we observe that the universities which have over a thousand leaked e-mail addresses account for 80 % of the total in the experimental samples. This experiment shows most universities do not take any extra steps to process the "@" symbol, such as changing @ to "at" or replacing it with a @ picture. As for the injectable URL's inspection, we found that nine universities have howintion vulnerabilities which might let hackers gain access to underlying databases for a variety of malicious purposes.

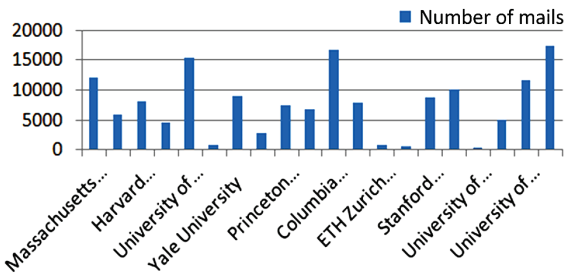

Figure 10 Bar chart of the amount of e-mail addresses found for each university

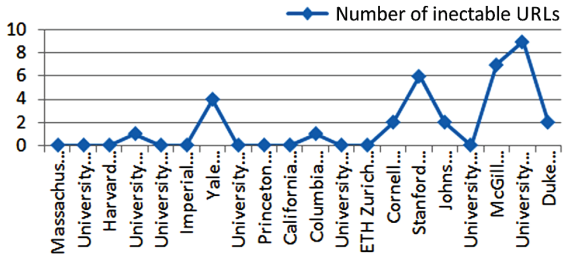

Figure 11 Line chart of the number of injectable URLs found for each university

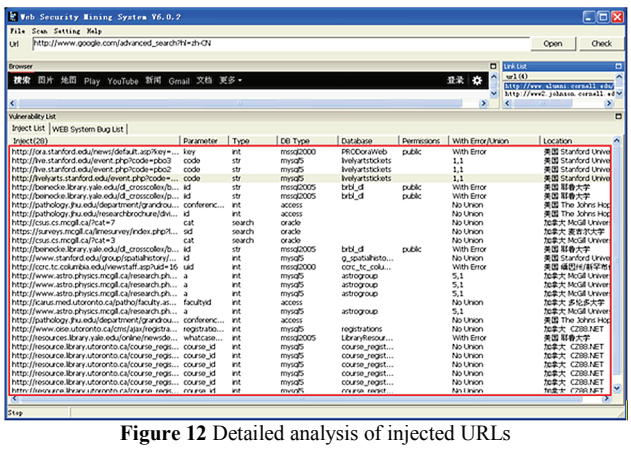

Figure 12 Detailed analysis of injected URLs

Fig. 12 below shows details of the 20 universities' websites, including database type, database name, and the specific formats used for injection attacks. For further research to explore the databases, we were able to identify the content of databases, for instance as is shown in Fig. 13. Additionally, we found some databases that stored user account passwords in clear text rather than hashing them, shown in Fig. 14.

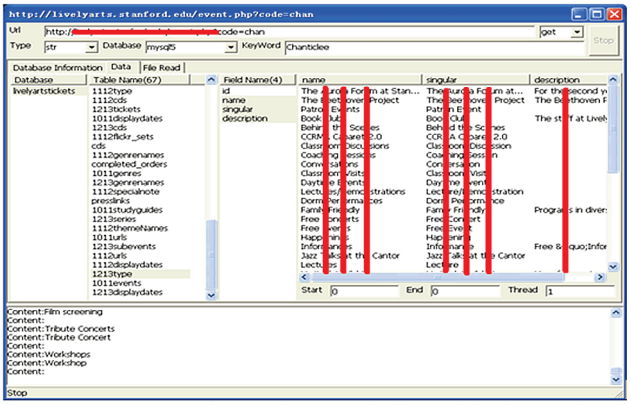

Figure 13 Content in a database revealed by an injection attack

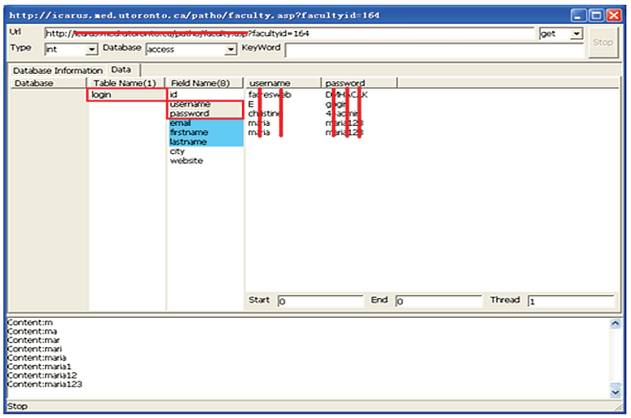

Figure 14 Accounts and passwords stored in the database

5 Response strategies

SQL injection has been a dangerous yet common attack vector for many years. However, during our penetration testing, we identified many cases of privacy leakage, most commonly through the exploitation of SQL injection vulnerabilities that should be fixed. Consider this example in PHP:

$sql="SELECT id,name,mail,cv,blog,twitter FROM register WHERE id= ".mysql_real_escape_string($_GET['id']);

When an attacker uses a URL such as:

http://vuln.example.com/user.php?id=12,AND,1=0,union,select,1,concat(user,0x3a,password),3,4,5,6,from,mysql.user,where,user=substring_index(current_user(),char(64),1)

This attack may bypass the intended protection from "mysql_real_escape_string" and instead allow for injected code, since "mysql_real_escape_string" only creates characters overflow of ", \r, \n, NULL, and Control-Z. In the URL above, these characters are totally avoided in the payload. Though the programmer may add in more characters to protect against such as "space", "parentheses", "SELECT" or "INSERT" to expand the scope of prevention, it is not sufficiently effective, but a temporary solution. For instance, consider the following queries:

*SELECT/**/passwd/**/from/**/user*
SELECT(passwd)from(user)
SELECT passwd from users where user=0x61646D696E
/ Hexadecimal encoding 0x61646D696E =admin*/*

Despite the number of vulnerabilities we found, the majority of websites were secure to our attempts thanks to sound security practices. Practical approaches towards protecting against SQL injection attacks are summarized as follows:

A. Pre-prepared Statements

Replacing the parameters in SQL statements with prepared statements. Consider this prepared statement written in Java:

String Stuname = request.getParameter("studentName");
//This should REALLY be validated too
//perform input validation to detect attacks
String query = "SELECT student_score FROM user_data WHERE user_name = ? ";
PreparedSttatement pstmt = connection.prepareStatement(query);
pstmt.setString(1, stuname);
ResultSet results = pstmt.executeQuery( );

From the programmer's perspective, the semantics of the SQL query remain the same using prepared statements. However, attackers will fail to change the structure of SQL when we use "?" as the parameter in SQL statement. As is shown below in PHP, even if attackers input string like tom' or '1' = '1, database will take it as username to query.

$query = "INSERT INTO myData (Name, Height, Weight) VALUES (?,?,?)";
$stmt = $mysqli->prepare($query);
$stmt = $bind_param("sdd", $val1, $val2, $val3);
$val1 = 'Fred';
$val2 = '177.7';
$val3 = '77.7';
/ Execute the statement */*
$stmt->execute();

Different methods are adopted for prepared statements in each language:

- Java EE – use PreparedStatement() with bind variables
- NET – use parameterized queries like SqlCommand() or OleDbCommand() with bind variables
- PHP – use PDO with strongly typed parameterized queries (using bindParam())
- Hibernate – use createQuery() with bind variables (called named parameters in Hibernate)
- SQLite – use sqlite3_prepare() to create a statement object

B. Using Stored Procedures

The difference between stored procedure and prepared statement is that stored procedure is defined and stored at the database level, rather than the application level. Also, a stored procedure itself might have an injection problem if it is built from user input as a dynamic SQL query. In that case, one should first apply strict input filtering to deal with input data from users. In the code below, sp_getStudentScore is a stored procedure:

```
String stuname =
request.getParameter("studentName");
//This should REALLY be validated too
try {
  CallableStatement cs =
connection.prepareCall("{call
sp_getStudentScore(?)}");
  Cs.setString(1, stuname);
ResultSet results = cs.executeQuery();
//... result set handling
} catch (SQLException se) {
//... logging and error handling}
```

Sometimes it is impractical to use prepared statements or stored procedures, for instance due to working with legacy code. In those scenarios, a strong focus must be placed on filtering input data.

C. Data Type Inspection

Checking the data type of input data can prevent SQL injection efficiently. The following PHP code restricts the input data type to be an integer, which makes it un-injectable:

```
settype($offset, 'integer');
$query = sprint("SELECT id, name FROM products
ORDER BY name LIMIT 20 OFFSET %d ;",$offset);
```

Checking the data format and data type is very beneficial. For example, when users input e-mail addresses, time, and date, these formats can all be checked to ensure that they are strictly adhered to. But checking data type is not invulnerable, as if the request needs the user to send arbitrary strings, for example, one must take care to appropriately handle these strings.

D. Using Safety Functions

Nearly every web programming language has built in functions to properly escape variables for use in SQL queries. When that is not available, each database vendor has its own strategy to deal with it. Taking MySQL for example, it can use the following rules to encode characters:

```
NUL (0x00)   \0
BS  (0x08)   \b
TAB (0x09)   \t
LF  (0x0a)   \n
CR  (0x0d)   \r
SUB (0x1a)   \z
"   (0x22)   \"
%   (0x25)   \%
'   (0x27)   \'
\   (0x5c)   \\
_   (0x5f)   \_
```

All other non-alphanumeric characters with ASCII values less than

```
256--> \c
```

where 'c' is the original non-alphanumeric character. Also, we can use the functions of OWASP ESAPI which are written by security experts, for instance:

```
ESAPI.encoder().encodeForSQL( new OracleCodec(),
queryparam);
Codec ORACLE_CODEC = new OracleCodec();
String query = "SELECT user_id FROM user_data WHERE
user_name = ' " + ESAPI.encoder().encodeForSQL(
ORACLE_CODEC, req.getParameter("userID"))
+ " ' and user_password = ' "
+ ESAPI.encoder().encodeForSQL( ORACLE_CODEC,
req.getParameter("pwd") )
+ " ' ";
```

Another fundamental principle of security is that the web application should have the minimal amount of power it needs to accomplish its task, and no more; this is known as the principle of least privilege. If there are different applications using the same database, then each application should use a separate account that cannot access the data from other applications. The database account used by web application should not have the authority to create functions or operate on files. If the principles discussed above are followed, then SQL injection attacks will become increasingly rare.

6 Conclusions and future work

Static and dynamic analysis cannot produce a full picture of web site security; therefore, we can use code review to make up for the deficiencies of static analysis and penetration testing to make up for the deficiencies of dynamic analysis. Penetration testing is invaluable in improving the security of us applications. Deep penetration testing, which executes human-like actions, can make up the deficiencies of dynamic analysis. Similar to how code review needs some specialized software to do code inspection, penetration testing also requires some domain-specific tools which can vastly improve efficiency. Even though automated analysis tools are growing more and more sophisticated, the human element inherent in code review and penetration testing is still of critical importance.

These days, there are many automated programs to aim at data collection from the internet [55]. If harvested e-mail addresses are sent malware, it could lead to many security problems [41 ÷ 43]. From the inspection we ran on university web sites, we found lots of academic websites do not cover the characteristic of "@" to mask email addresses and they also do not optimize robot.txt settings [35, 44, 45], which let us easily find thousands of mail addresses. However, there are also many academic websites with more strict security procedures that do not expose e-mail addresses and do not have injection vulnerabilities [46, 47]. Furthermore, our research showed that many databases on academic servers contained valuable data (including passwords) in plaintext, which could obviously be of great value to malicious hackers who could exploit similar vulnerabilities to access database content. This research reminds us that web security is an important consideration in many phases of web development, including database design and

application development. Software written with the best of intentions may be vulnerable to creative attacks made by humans with very different thought processes than the software creators [48, 49]. The future of this research will emphasize the comparative efficiency of vulnerability detection [21, 50, 51, 52], keep expanding the functionality of the WSMS, and work with the Exploit Database [34] to use knowledge gained from our research to improve vulnerability detection performed on other websites. Then, as other researchers improve the state of knowledge about vulnerabilities and thus improve our work, a positive feedback loop will lead to enhanced vulnerability analysis that can be used for any website.

7 References

[1] Scott, D.; Sharp, R. Abstracting application-level web security. // Proceedings of the 11th International Conference on World Wide Web, ACM. (2002), pp. 396-407.

[2] Curphey, M.; Endler, D.; Hau, W.; Taylor, S.; Smith, T.; Russel, A. et al. A guide to building secure web applications: The open web application security project, OWASP. 2002.

[3] Laribee, L.; Barnes, D. S.; Rowe, N. C.; Martell, C. H. Analysis and defensive tools for social-engineering attacks on computer systems. // Information assurance workshop. IEEE. (2006), pp. 388-389.

[4] Kotenko, I.; Stepashkin, M.; Doynikova, E. Security analysis of information systems taking into account social engineering attacks. // PDP, Euromicro international conference, IEEE. (2011), pp. 611-618.

[5] Huang, Y. W.; Yu, F.; Hang, C.; Tsai, C. H.; Lee, D. T.; Kuo, S. Y. Securing web application code by static analysis and runtime protection. // Proceedings of the 13th International Conference on World Wide Web, ACM. (2004), pp. 40-52.

[6] Lin, J. C.; Chen, J. M.; Liu, C. H. An automatic mechanism for sanitizing malicious injection. // ICYCS, The 9th International Conference, IEEE. (2008), pp. 1470-1475.

[7] Pan, C. C.; Yang, K. H.; Lee, T. L. Secure online examination architecture based on distributed firewall. // IEEE. (2004), pp. 533-536.

[8] Riden, J. Responding to security incidents on a large academic network-a case study. // IEEE, 2006.

[9] Priyadarshini, R.; Aishwarya, S.; Ahmed, A. A. Search engine vulnerabilities and threats-a survey and proposed solution for a secured censored search platform. // INCOCCI, IEEE. (2010), pp. 535-539.

[10] Kleinberg, J. The CLEVER project, IBM. http://www.almaden.ibm.com/projects/clever.shtml, 2010. Accessed: August 15, 2013.

[11] Wikimedia Foundation. Direct hit technologies. Wikipedia Foundation. http://en.wikipedia.org/wiki/Direct_Hit_Technologies, 2011. (15.08.2013).

[12] N. L. Northern Light Search, NLG, LLC. http://www.nlsearch.com/home.php, 2012. (23.08.2013).

[13] Phillips, F. Scirus topic pages: New publication possibilities for Moti researchers. // Management of Engineering and Technology. (2009), pp. 2184-2189.

[14] Chakrabarti, S.; Van den Berg, M.; Dom, B. Focused crawling: A new approach to topic-specific web resource discovery. // Computer Networks. 11, 31(1999), pp. 1623-1640. DOI: 10.1016/S1389-1286(99)00052-3

[15] Lawrence, S.; Giles, C. L.; Bollacker, K. Digital libraries and autonomous citation indexing. // Computer. 6, 32(1999), pp. 67-71. DOI: 10.1109/2.769447

[16] Shirey, R. Internet security glossary. // IETF The Internet Society. http://www.ietf.org/rfc/rfc2828.txt, 2002. (15.08.2013).

[17] Xu, P.; Liu, W. A research of on-line static security analysis based on WEB services. // APPEEC, IEEE. (2011), pp. 1-4.

[18] Weiser, M. Program slicing, IEEE transactions on software engineering. // SE10. 4, (1984), pp. 352-357. DOI: 10.1109/TSE.1984.5010248

[19] Phalgune, A. Testing and debugging web applications: An end-user perspective. // VLHCC, IEEE Symposium, 2004.

[20] Antunes, N.; Vieira, M. Comparing the effectiveness of penetration testing and static code analysis on the detection of SQL injection vulnerabilities in web services. // Dependable Computing. PRDC, IEEE Symposium. (2009), pp. 301-306.

[21] Bau, J.; Bursztein, E.; Gupta, D.; Mitchell, J. State of the art: Automated black-box web application vulnerability testing. // SP, IEEE Symposium. (2010), pp. 332-345.

[22] El Ioni, N.; Sillitti, A. Open web services testing. // SERVICES, IEEE World Congress. (2011), pp. 130-136.

[23] Khoury, N.; Zavarsky, P.; Lindskog, D.; Ruhl, R. An analysis of black-box web application security scanners against stored SQL injection. // PASSAT, IEEE Third International Conference on Social Computing. (2011), pp. 1095-1101.

[24] Bishop, M. About penetration testing. // Security & Privacy, IEEE. 6, 5 (2007), pp. 84-87. DOI: 10.1109/MSP.2007.159

[25] Antunes, N.; Vieira, M. Enhancing penetration testing with attack signatures and interface monitoring for the detection of injection vulnerabilities in web services. // SCC, IEEE, International Conference. (2011), pp. 104-111.

[26] Hwee-Joo, K.; Pauli, J. J. Web penetration testing: Effectiveness of student learning in web application security. // FIE. 2011.

[27] Mainka, C.; Somorovsky, J.; Schwenk, J. Penetration testing tool for web services security. // SERVICES, IEEE Eighth World Congress. (2012), pp. 163-170.

[28] Junjin, M. An approach for SQL injection vulnerability detection. // ITNG, IEEE. (2009), pp. 1411-1414.

[29] Chapela, V. Advanced SQL injection, OWASP Foundation. 2005.

[30] Overstreet, R. Protecting yourself from SQL injection attacks, QuinStreet. http://www.4guysfromrolla.com/ASPScripts/PrintPage.asp?REF=%2Fwebtech%2F061902-1.shtml, 2006. (01.09.2013).

[31] Boyd, S. W.; Keromytis, A. D. SQLrand: Preventing SQL injection attacks. // Applied Cryptography and Network Security. (2004), pp. 292-302.

[32] Anley, C. More advanced SQL injection. // NGSSoftware Insight Security Research. 2002.

[33] Anley, C. Advanced SQL injection in SQL server applications. // NGSSoftware Insight Security Research. 2002.

[34] The Exploit Database, Offensive Security, http://www.exploit-db.com, 2012. (10.08.2013).

[35] Sun, Y.; Councill, I. G.; Giles, C. L. Botseer: An automated information system for analyzing web robots. // Web Engineering, ICWE Eighth International Conference, IEEE. (2008), pp. 108-114.

[36] Cho, Y. C.; Pan, J. Y. Multiple-Feature Extracting Modules Based Leak Mining System Design. // The Scientific World Journal. Article ID 704865, Vol. 2013.

[37] Cho, Y. C.; Pan, J. Y. Vulnerability Assessment of IPv6 Websites to SQL Injection and Other Application Level Attacks. // The Scientific World Journal, Article ID 946768, Vol. 2013.

[38] Shkapenyuk, V.; Suel, T. Design and implementation of a high-performance distributed web crawler. // Data Engineering, IEEE. (2002), pp. 357-368.
[39] Najork, M.; Wiener, J. L. Breadth-first crawling yields high-quality pages. // Proceedings of the 10th International Conference on World Wide Web, ACM. (2001), pp.114-118.
[40] National Taiwan University. http://www.ntu.edu.tw/english/, 2012. (20.08.2013).
[41] Hoanca, B. How good are our weapons in the spam wars? // Technology and Society Magazine, IEEE. 1, 25(2006), pp. 22-30. DOI: 10.1109/MTAS.2006.1607720
[42] Geer, D. Malicious bots threaten network security. // Computer. 1, 38(2005), pp. 18-20. DOI: 10.1109/MC.2005.26
[43] Heymann, P.; Koutrika, G.; Garcia-Molina, H. Fighting spam on social web sites: a survey of approaches and future challenges. // Internet Computing, IEEE. 6, 11(2007), pp. 36-45. DOI: 10.1109/MIC.2007.125
[44] Stassopoulou, A.; Dikaiakos, M. D. Crawler detection: a Bayesian approach. // Internet Surveillance and Protection ICISP, IEEE. (2006), pp. 16-17.
[45] Tong, W.; Xie, X. A research on a defending policy against the webcrawler's attack. // ASID, IEEE. (2009), pp. 363-366.
[46] Central Police University. http://www.cpu.edu.tw, 2012. (01.09.2013).
[47] National Defense University. http://www.ndu.edu.tw, 2012. (01.09.2013).
[48] Arnold, A. D.; Hyla, B. M.; Rowe, N. C. Automatically building an information-security vulnerability database. // Information Assurance Workshop, IEEE. (2006), pp. 376-377.
[49] Avancini, A.; Ceccato, M. Security testing of web applications: A search-based approach for cross-site scripting vulnerabilities. // Source Code Analysis and Manipulation (SCAM). IEEE International Working Conference. (2011), pp. 85-94.
[50] Fonseca, J.; Vieira, M.; Madeira, H. Testing and comparing web vulnerability scanning tools for SQL injection and XSS attacks. // PRDC, IEEE. (2007), pp. 365-372.
[51] Fong, E.; Gaucher, R.; Okun, V.; Black, P. E. Building a test suite for web application scanners. // Hawaii International Conference on Syst. Sciences. IEEE. pp. 478-478, 2008.
[52] Chao, D.; Danfeng, Y.; Yun, Y. Fangchun, Y. A domain-oriented distributed vulnerability scanning mechanism. // Broadband Network and Multimedia Technology, IC-BNMT, IEEE International Conference. (2009), pp. 832-836.
[53] OWASP Foundation. The ten most critical web application security vulnerabilities. // OWASPF. http://www.owasp.org/images/e/e8/OWASP_Top_10_2007.pdf, 2007. (27.08.2013).
[54] Williams, J. The ten most critical web application security vulnerabilities. // OWASP Foundation, 2010.
[55] Cho, Y. C.; Pan, J. Y. Design and Implementation of Website Information Disclosure Assessment System. // PLoS ONE, (2015). DOI: 10.1371/journal.pone.0117180

Author's address

Ying-Chiang Cho
Department of Electrical Engineering,
National Chung Cheng University,
Chia-Yi 62102, Taiwan
E-mail: silvergun@mail2000.com.tw