IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 1, 2018, accepted October 18, 2018, date of publication November 13, 2018,
date of current version December 18, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2881070

A Web Second-Order Vulnerabilities Detection Method

MIAO LIU AND BIN WANG
School of Computer Science and Educational Software, Guangzhou University, Guangzhou 510006, China
Corresponding author: Miao Liu (liumiao@gzhu.edu.cn)

This work was supported by Guangzhou Municipal Universities under Project 1201620342.

ABSTRACT Second-order vulnerabilities are more subtle and more destructive than the first-order vulnerabilities. After researching and analyzing the principles of web penetration testing and second-order attack principles, this paper proposes a method to detect web second-order security vulnerabilities. The method detects web second-order security vulnerabilities through two crawl scans. It crawls the website URL for the first time, sends anchor points, crawls URLs of the storage anchor point for the second time, and detects second-order web security vulnerabilities specifically for these suspicious URLs. The approach greatly reduces the time complexity of detecting second-order web security vulnerabilities and makes up for the lack of methods to detect web security second-order vulnerabilities.

INDEX TERMS Web security, second-order attacks, penetration testing, second-order vulnerability detection.

I. INTRODUCTION

With the rapid development of the Internet, web applications provide people with more and more functions, corresponding web vulnerabilities are gradually increasing. Many websites harbor malware because of web vulnerabilities. According to the Symantec's 2018 Internet Security Threat Report, Symantec WebPulse URL classification and reputation analysis service scanned 1.07 billion URLs per day in 2017, the number of malicious URLs grew by 2.8 percent, with 7.8 percent (1 in 13) of all URLs identified as malicious. The number of URLs resulting from bot-related traffic, such as that used for command and control, grew by 62.3 percent, accounting for 14.7 percent of all malicious URLs in 2017 [1].

Security incidents due to web vulnerabilities have also occurred frequently, such as the large-scale Internet incidents in the United States in 2016, Jingdong 12G user data leakage incidents, Intercontinental hotels Group (IHG) credit card data breaches in 2017, and WannaCry file extortion incidents on a global scale. These web security incidents are shocking, but these are only the tip of the iceberg of many web security incidents. Web security has an important impact on people's daily lives and needs constant attention.

We know that regular web security vulnerabilities are almost always triggered by the user's input. SQL (Structured Query Language) injection vulnerability is that user input causes SQL statements to be executed incorrectly. XSS (Cross Site Scripting) vulnerability is that user input causes page parsing error. User input leads to error OS command execution is RCE (Remote Command Execution) vulnerability. We refer to these categories of vulnerabilities that cause web application errors by one time input as web first-order security vulnerabilities, and corresponding two times inputs triggered vulnerabilities are collectively referred to as web second-order security vulnerabilities. An attacker exploits a first-order vulnerability to input attack vectors and immediately gets attack results from responses of web applications. Second-order web security vulnerabilities are hidden deeper and not easily detected, because second-order web attacks belong to multi-step attacks, and are accomplished by combining two step inputs [2]. Attackers exploit second-order vulnerabilities to attack in two steps. The first step is to input attack vectors in some injection points, web applications respond normally and store tainted data in databases, sessions or source code files. The second step is to access tainted data or input attack vectors in different points, and then attackers get attack results from responses this time.

II. RELATED WORKS

The second-order code injection attack was proposed in [3] in 2004, and relevant research results were few in the following years. In 2010, Bau [4] from Stanford University compared mainstream penetration testing tools by experiments, and one of the results showed that detection rate of the second-order SQL vulnerability was zero. Common vulnerability scanning tools only involve the analysis of

2169-3536 © 2018 IEEE. Translations and content mining are permitted for academic research only.
Personal use is also permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

one step attack, cannot complete the analysis of multiple associated steps attacks, and so cannot find effective attack paths [5]. Later, Korscheck [6] proposed a strategy based on iStar, a workflow-based detector created by Daimler. The strategy is to separate the penetration test from the attack phase, so that the detector can perceive the behavior of the web application and analyze unexpected web application behavior. The method improved storage XSS detection rate. Rocha and Souto [7] developed ETSSDetector, a generic and modular web vulnerability scanner that automatically analyzes web applications to find XSS vulnerabilities. The scanner includes an Extraction component, a Qualification component, a Test component, an Analysis component, and a Database. Extraction component is responsible to identify, collect and analyze the necessary information to evaluate the web application. Qualification component makes the analysis of the pages of the web application being tested identifying each vulnerable point. The Test component is responsible to inject and execute XSS attacks that will be assigned to each vulnerable point. Analysis component performs the detection of XSS attacks through the analysis of the results from previous modules. All information collected during ETSSDetector execution are stored in a database like links, forms and parameters. Finally, experiments show that ETSSDetector is presented as a good alternative to assist information security professionals in the fight against the vulnerabilities that are presented in web systems. The scanner uses a traditional crawl detection algorithm, and it is inefficient when dealing with large websites [8].

The use of white-box testing methods to detect web security second-order vulnerabilities began in 2014. In [9], a source code analysis tool named RIPS was used for semantic analysis, combined with database storage status detection of web security second-order vulnerabilities. Johannes Dahse and Thorsten Holz tested some popular web applications at the time and detected 159 second-order security breaches. Yan [10] also proposed an approach to analyse source code to find the suspicious data columns, and then test the suspicious data columns by dynamic methods. This approach uses static analysis to understand the internal information of the web application, narrows the scope of detection and then fills the gap of the high false positive rate of static analysis by dynamic testing. Li *et al.* [11] proposed a detection method combined with static smut analysis and fuzz testing, and the method could detect some second-order SQL injection vulnerability. Marashdih and Zaaba [12] proposes a method which begins from Pixy tool to analyze the source code, in a way to draw the Control Flow Graph (CFG) and to find the whole paths of the PHP source code. Two different experiments were conducted and experimental results showed that the method can effectively remove the XSS vulnerability from source code. Reference [13] proposed a static analysis method for detecting second-order DoS vulnerabilities in web applications, and implemented the analysis in a tool called Torpedo.

Hu [14] established a defense resistance and remedy model of SQL injection attack, and the related code is implemented. By applying the program on actual projects, it is proved that the proposed approach is able to protect the network from SQL injection. Le *et al.* [15] proposed a series of new second-order SQL attack techniques. They are blind second-order SQL injection, second-order SQL injection attacks operating systems and client second-order SQL injection. The author didn't propose any corresponding second-order vulnerability detection method. Reference [16] proposed a middleware plus proxy server model to defend against second-order SQL injection attacks. The model protected second-order SQL injection by intercepting random standard SQL additions, deletions, and other keywords in middleware and proxy servers, and didn't mention how to find second-order SQL injection vulnerabilities.

III. SECOND-ORDER VULNERABILITIES DETECTION

A. SECOND-ORDER VULNERABILITIES ATTACK PRINCIPLE

Web second-order attacks must have two time attack inputs or access. The first attack input does not cause web application vulnerabilities to be triggered, then the second attack input causes the web application to produce an error, or attackers access tainted data in the second step. Before the second attack input is entered, the web application must have a web security vulnerability, such as SQL injection, stored XSS, RCE or LFI (Local File Include). Otherwise, the second attack input will not cause the web application to perform incorrectly. The reason is that only two inputs join together to form a effective attack vector. The state diagram of web application is illustrated in Fig. 1.

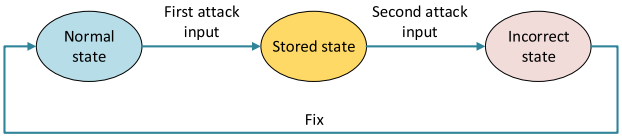


FIGURE 1. Web application state transition diagram.

The attacker submits some well-conceived attack inputs in the HTTP request. The web application server stores some or all of inputs. The attacker sends the second attack input, web first-order vulnerability status changes, and results in incorrect execution. Second-order attacks require two steps to achieve attack effects, and so second-order vulnerabilities have strong concealment.

For example, a code segment of PHP that simulates users posting blogs is as follows.

```
<?php
if(isset($_POST["blog"]))
{
$content=mysql_real_escape_string($_POST["content"]);
$owner = $_SESSION["username"];
if($content != "")
   {
```

```
$sql = "INSERT INTO blog (date, content, owner) VALUES (now(),'". $content . "','". $owner . "')";
        $recordset = mysql_query($sql);
        if(!$recordset)
        {
          die("Error: " . mysql_error(). "<br />");
        }
         $message = "<font color= "green ">message has added!</font>";
      }
   }
   echo "  ". $message;
   ?>
```

The code uses the mysql_real_escape_string function to prevent SQL injection attacks on the blog content. The next step is to store the content to the database, and the code segment has no problem. If the user enters some dangerous SQL injection characters, these characters will not cause the blog program to execute incorrectly, and these special characters can also be output to some web pages in the application.

Then the following code segment is a popular statistic for the content published by a user.

```
<?php
$id=$_GET['id'];
$secsql="SELECT content FROM blog WHERE id='$id'";
$st=mysql_query($secsql);
$cont=mysql_fetch_row($st);
$finsql="SELECT * FROM blog WHERE content='$ cont[0]'";
$st2=mysql_query($finsql);
echo mysql_num_rows($st2);
?>
```

The code segment first accepts an id parameter to query popular degree of the content posted by the user's blog. We assume that the id parameter is an anti-SQL injected parameter. The code first extracts the contents of the id, and then queries the number of the same content in the blog table. The code seems to be safe. However, there is a potential threat that the statistical code will be executed incorrectly when the information stored in the database is a dangerous SQL injection attack string, such as ''; drop table blog;- -''. If the injected code is executed, the blog table will be deleted.

The second-order vulnerability does not appear at all when the user submits normal content. Only special SQL injection attack strings can trigger it, if they are stored in the database before. Therefore, ordinary tools can hardly detect these second-order vulnerabilities.

B. SECOND-ORDER VULNERABILITY DETECTION PRINCIPLE

If common black box detection tools are extended according to first-order vulnerabilities detection approach to detect second-order web security vulnerabilities, there will be few vulnerabilities to be detected. The experimental results of [4] and [15] showed this problem.

If you use black box testing for web security second-order vulnerabilities, the first problem to face is time complexity. The ordinary black box detection tool first performs a comprehensive scan of the web application to obtain a URL table, and then conducts various first-order vulnerability tests on each URL in this table. If we modified these tools in order to detect web security second-order vulnerabilities, the general process is as follows:

1) Crawl the testing web application to get a list of URLs;
2) Send an attack input to each URL in the URL table;
3) Crawl the testing web application again to get a new URL table;
4) Perform various vulnerability tests on the new URL table.

The above steps seem to be no problem, but it has two important deficiencies. The first deficiency is that in the second step, due to types of vulnerabilities are not known, various attack inputs need to be sent, making the number of URLs in the third step much larger. The second deficiency is that the new table has no relationship with attack inputs in the second step, so that the fourth step of the vulnerabilities test requires a comprehensive vulnerabilities testing. It is necessary to consider all combinations of two tables. As a result, the above process will take a great deal of time.

C. A WEB SECOND-ORDER VULNERABILITY DETECTION ALGORITHM

According to the storage characteristics of web security second-order vulnerabilities, this paper presents a black box second-order vulnerability detection algorithm. The algorithm flow chart is shown in Fig. 2.

The algorithm detects web second-order vulnerabilities through two time crawls. The algorithm greatly reduces the number of URLs that detect suspicious web second-order vulnerabilities by using special strings and numeric ''anchors'', and also establishes relationship between the URL of the first time crawling and URLs of suspicious web second-order vulnerabilities.

As illustrated in Figure 2, the algorithm crawls the target site first, and stores the URL of the entire site in a table A. The next step is to read the contents of table A, and sends a request for the string ''SecCheckBy'' + n (n is the storage order of the URL in table A) to every URL. Then the algorithm crawls the target website again, this time only accesses the URL of the web page content containing the string ''SecCheckBy'' and stores into the table B.

The core part of the algorithm is reading contents of table B. When reading a URL record ordered as ''k'', the algorithm gets the contents of the URL and takes the number behind the string ''SecCheckBy'' as ''n''. According to ''n'', the algorithm finds the URL of the order n in table A. Then the algorithm sends a series of attack vectors to this URL, and then perform regular first-order vulnerability detection on the 'k'th URL in table B. If there is a detection result, the vulnerability information will be stored in the database, otherwise the next record in table B will be read until

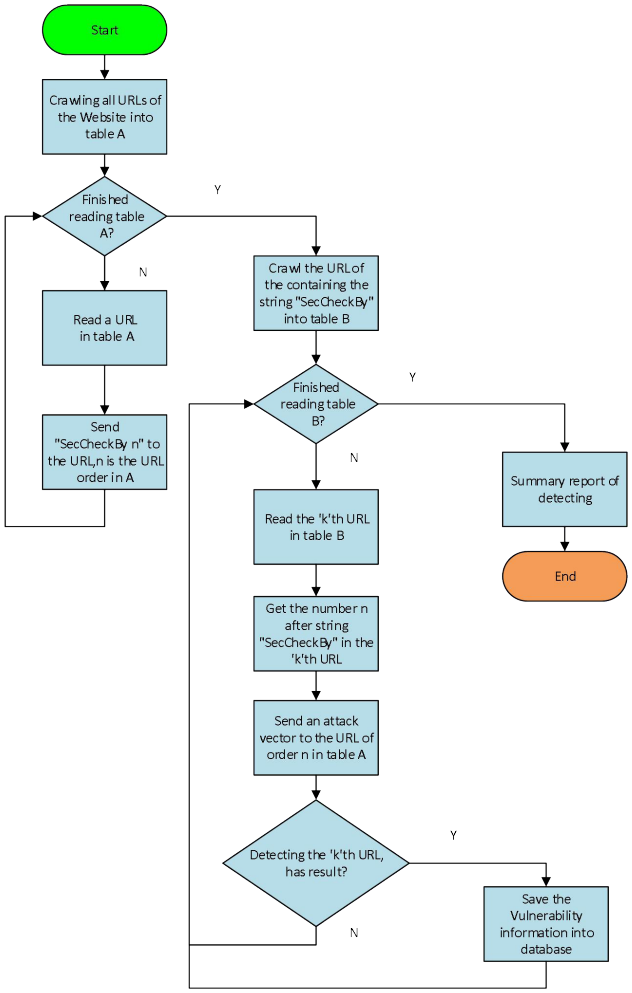


FIGURE 2. Flowchart of second-order vulnerability detection algorithm.

table B is finished. The last step is summarizing the detecting results.

From the above algorithm description, we can see that the detection number of the algorithm is directly proportional to the number of records in table B. The number of detecting for each type of web security second-order vulnerability equals to the record number of table B. If the record number of table B is m and the type number of second-order vulnerabilities to be detected is t, the total number of detecting is ($t * m$).

Reviewing the process of detection by using common penetration testing tools, assuming that the number of total site URLs crawled in the first step is n, and then the number of new total site URLs scanned in the third step is n+c, where c is the number of new added URLs that the web application responds to attack inputs in the second step. Because the two obtained global URL tables are not related, it is necessary to consider all combination cases of the two tables. In this way, the detection number of every type of web security second-order vulnerability is $n*(n+c)$, and the total number of detecting is $(t*n*(n+c))$. Comparing the algorithm proposed, because the record set in the table B is subset of all new global url record set,and it also means that m is less than or equals to n+c, the detection algorithm is better than the above process.

IV. DESIGN OF THE SYSTEM

The system adopts a B/S architecture so that users can easily perform second-order vulnerability detection on web applications through browsers. In addition, the system adopts modular design method to facilitate subsequent expansion. The overall architecture of the system is shown in Fig. 3.

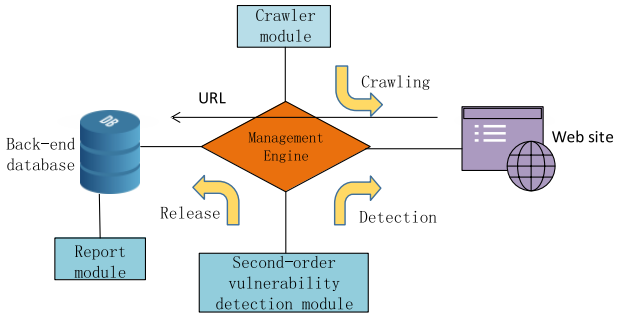


FIGURE 3. The overall architecture of system.

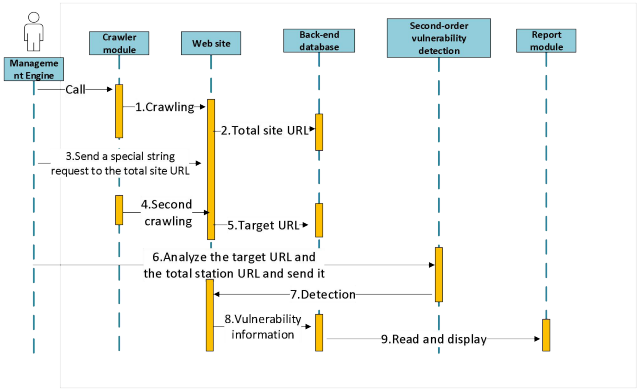


FIGURE 4. Web security second-order vulnerability detection system process.

The management engine controls the crawler module and the second-order vulnerability detection module. The crawler module crawls the URLs of the web application and stores them into the back-end database. The second-order vulnerability detection module detects the web application and stores vulnerability information to the back-end database. The reporting module reads vulnerability information and displays them in the browser.

The system process is illustrated in Fig. 4.

The flow chart of the second-order vulnerability detection module is shown in Fig. 5.

According to the algorithm, the second-order vulnerability detection module obtains two input URLs A and B. URL A is the first step attacking URL, and the URL B is the URL which stores the attacking vector of the first step. Rule matching is performed on the contents of the two URLs to determine the most likely second-order vulnerability information, then the corresponding second-order vulnerability detection module is invoked for detection, and finally the result information of the vulnerability is output to the database.

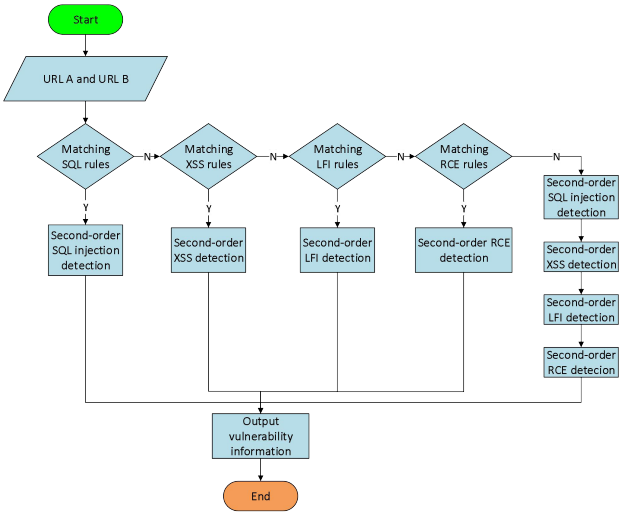


FIGURE 5. Flowchart of the second-order vulnerability detection module.

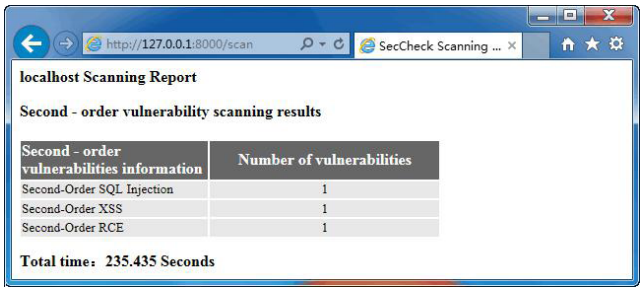


FIGURE 6. Testing result.

V. EXPERIMENT ANALYSIS

There are two environments that need to be set up in this experiment. One is the testing web application, which includes three web second-order vulnerabilities, and they are one second-order SQL injection vulnerability, one second-order XSS vulnerability, and one second-order RCE vulnerability. The other is the Python web environment, mainly to run web second-order vulnerability detection system. Both environments are built on the Win7 OS. The testing result is shown in Fig. 6.

Using scanning tools such as ZAP, three web security second-order vulnerabilities have not been detected. The overall result shows that this black box web second-order vulnerability detection method is feasible.

VI. CONCLUSION

Web vulnerabilities make attackers more versatile and pose serious threats to web security. There are usually two ways to detect web vulnerabilities: manual detection and automated detection. Detecting vulnerabilities manually is becoming more and more difficult as the scale of source code grows. The tools used for automatic detection are all based on basic characteristics of first-order vulnerabilities, and second-order vulnerabilities are rarely considered. Second-order vulnerabilities are more subtle and destructive than first-order vulnerabilities.

In this paper, we design a web second-order vulnerabilities detection algorithm by utilizing storage characteristics of theirs, and carry out experimental verification. The experimental results show that the algorithm can effectively detect web second-order vulnerabilities. In the future, the web second-order vulnerabilities detection system based on the algorithm will be further improved, and we will add more kinds and quantities of attack vectors, so that it can detect more types of web second-order security vulnerabilities, reduce false alarm rate, and improve the detection effect of web second-order security vulnerabilities.

REFERENCES

[1] Symantec Corporation. (2018). *ISTRInternet Security. Threat Report. Volume. 23*. [Online]. Available: https://www.symantec.com/security-center/threat-report

[2] P. Parrend, J. Navarro, F. Guigou, A. Deruyver, and P. Collet, "Foundations and applications of artificial Intelligence for zero-day and multi-step attack detection," *J. Inf. Secur.*, vol. 2018, no. 4, pp. 1–21, 2018, doi: 10.1186/s13635-018-0074-y.

[3] G. Ollmann, "Second-order code injection attacks," NGS Insight Secur. Res., Manchester, U.K., Tech. Rep., 2004. [Online]. Available: https://www.nccgroup.trust/uk/our-research/second-order-code-injection-attacks/

[4] J. Bau, E. Bursztein, D. Gupta, and J. Mitchell, "State of the art: Automated black-box Web application vulnerability testing," in *Proc. IEEE Symp. Secur. Privacy*, Berkeley, CA, USA, May 2010, pp. 332–345, doi: 10.1109/SP.2010.27.

[5] S. Nagpure and S. Kurkure, "Vulnerability assessment and penetration testing of Web application," in *Proc. Int. Conf. Comput., Commun., Control Autom. (ICCUBEA)*, Pune, India, 2017, pp. 1–6, doi: 10.1109/ICCUBEA.2017.8463920.

[6] C. Korscheck, "Automatic detection of second-order cross-site scripting vulnerabilities," Wilhelm Schickard Inst., Univ. Tübingen, Tübingen, Germany, Tech. Rep. 2010. [Online]. Available: http://www.vipread.com/library/item/121

[7] T. S. Rocha and E. Souto, "ETSSDetector: A tool to automatically detect cross-site scripting vulnerabilities," in *Proc. IEEE 13th Int. Symp. Netw. Comput. Appl.*, Cambridge, MA, USA, Aug. 2014, pp. 306–309, doi: 10.1109/NCA.2014.53.

[8] H. Zhao, "Research on detection algorithm of WEB crawler," *Int. J. Secur. Appl.*, vol. 9, no. 10, pp. 137–146, 2015, doi: 10.14257/ijsia.2015.9.10.12.

[9] J. Dahse and T. Holz, "Static detection of second-order vulnerabilities in Web applications," in *Proc. USENIX Secur. Symp.*, 2014, pp. 989–1003, [Online]. Available: https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-dahse.pdf

[10] L. Yan, X. Li, R. Feng, Z. Feng, and J. Hu, "Detection method of the second-order SQL injection in Web applications," in *Structured Object-Oriented Formal Language and Method. SOFL+MSVL* (Lecture Notes in Computer Science), vol. 8332, S. Liu and Z. Duan, Eds. Cham, Switzerland: Springer, 2014, doi: 10.1007/978-3-319-04915-1_11.

[11] X. Li, W. Zhang, and L. Zheng, "Vulnerability detection using second-order SQL injection combining dynamic and static analysis," *J. Huaqiao Univ. (Natural Sci.)*, vol. 4, no. 4, pp. 600–605, 2018, doi: 10.11830/ISSN.1000-5013.201606110.

[12] A. W. Marashdih and Z. F. Zaaba, "Detection and removing cross site scripting vulnerability in PHP Web application," in *Proc. Int. Conf. Promising Electron. Technol. (ICPET)*, Deir El-Balah, UAE, 2017, pp. 26–31, doi: 10.1109/ICPET.2017.11.

[13] O. Olivo, I. Dillig, and C. Lin, "Detecting and exploiting second order denial-of-service vulnerabilities in web applications," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 616–628, doi: 10.1145/2810103.2813680.

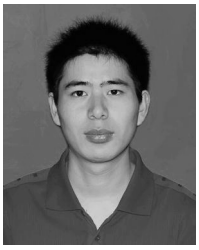
[14] H. Hu, "Research on the technology of detecting the SQL injection attack and non-intrusive prevention in WEB system," in *Proc. AIP Conf.*, 2017, p. 020205, doi: 1839.020205.10.1063/1.4982570.

[15] D. Le, X. Li, and S. Gong "Research on second-order SQL injection techniques," *J. Commun.*, vol. S1, no. Z1, pp. 85–93, Nov. 2015, doi: 10.11959/j.issn.1000-436x.2015285.

[16] C. Ping, "A second-order SQL injection detection method," in *Proc. IEEE 2nd Inf. Technol., Netw., Electron. Autom. Control Conf. (ITNEC)*, Chengdu, China, Dec. 2017, pp. 1792–1796, doi: 10.1109/ITNEC.2017.8285104.

MIAO LIU received the B.Sc. and M.Sc. degrees in computer science from Information Engineering University, China, and the Ph.D. degree in computer application technologies from the South China University of Technology, China, in 1987, 1991, and 2007, respectively.

He is currently an Associate Professor with the Computer Science Department, Guangzhou University, Guangzhou, China. He has authored or co-authored over 30 technical papers. His major research interests are network security, artificial intelligence, and e-commerce.



BIN WANG received the B.S. degree in computer science and technology from Zhoukou Normal University, China, in 2010, and the M.S. degree in computer technology from Guangzhou University, China, in 2018.

He was a Laboratory Assistant with the South China Institute of Software Engineering, Guangzhou University, from 2011 to 2015. He is currently a Software Engineer in a software development company in China. His current research interests include web security and security applications.