




## Research Article

# RW-Fuzzer: A Fuzzing Method for Vulnerability Mining on Router Web Interface

Longjuan Wang <sup>1,2</sup> Chunjie Cao,<sup>1,2</sup> Jun Ye <sup>1,2</sup> and Wenjie Zhong <sup>3</sup>

<sup>1</sup>School of Cyberspace Security, Hainan University, Haikou 570228, China

<sup>2</sup>Key Laboratory of Internet Information Retrieval of Hainan Province, Haikou 570228, China

<sup>3</sup>Hatlab, Hangzhou Dbappsecurity Co., Ltd, Hangzhou 310051, China

Correspondence should be addressed to Jun Ye; yejun@hainanu.edu.cn

Received 21 February 2022; Accepted 23 March 2022; Published 25 April 2022

Academic Editor: Deepak Kumar Jain

Copyright © 2022 Longjuan Wang et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

As the main routing device of the network, most routers can be set up and managed through their web enabled admin portal. This paper proposes a new method for router admin portal vulnerability mining fuzzing test (RW-fuzzer: Router Web fuzzer). The mutation samples that generated by Boofuzz are used to construct the test sample set for fuzzy testing. The constructed mutation test cases are more suitable for the attack load or critical value of the router's Web interface. They can cause unexpected errors for the devices more easily, which achieves the goal of discover potential vulnerabilities, and the practicality is excellent. Based on the proposed RW-fuzzer method, this work conducted fuzzing tests on 4 widely sold router models from manufacturers. Four nday vulnerabilities and one 0day vulnerability have been found. Experiment results show that the proposed RW-fuzzer method is effective.

## 1. Introduction

A router is a network communication device that connects various local area networks and wide area networks in the Internet. It has the functions of network interconnection, data processing, and network management and plays a vital role in the Internet. The router itself has a large number of exploitable security vulnerability. These security vulnerabilities can present a range of risks to users, including intelligence gathering, identity theft, data theft, damage or destruction of computer equipment, congestion and destruction of network traffic, firmware removal, and botnet infection. The Mirai IoT (Internet of Things) malware DDoS attack that broke out in 2016 caused the DNS server and the websites of many mainstream Internet companies to be paralyzed, which included Twitter, payment website PayPal, and music website Spotify. In this case, the botnet of the DDoS attack reached more than 300,000 units, of which routers accounted for the majority. What is more, the mal-

ware used Telnet scanners to identify router devices exposed online and used a combination of vulnerabilities and default credentials to infect insecure routers. Hence, the routers can be added to a massive botnet. In September 2019, FortiGuard Labs of Fortinet, an integrated automation network security solution provider, discovered and reported an unauthorized command injection vulnerability in router D-Link products (FG-VD-19-117/CVE-2019-16920). An attacker can use this vulnerability to achieve remote code execution (RCE) on the device without authentication, and this vulnerability is marked as a high-severity vulnerability. Additionally, in 2019, the 360 Security Research Institute team conducted an in-depth analysis of the vulnerability [1]. After extracting the vulnerability identification pattern, the self-developed FirmwareTotal (IoT firmware automatic security analysis solution) was used to analyze the firmware of more than 200,000 routers in the entire network. This scan found that the vulnerability exists in 58 versions of firmware in 13 different D-Link models of routers.

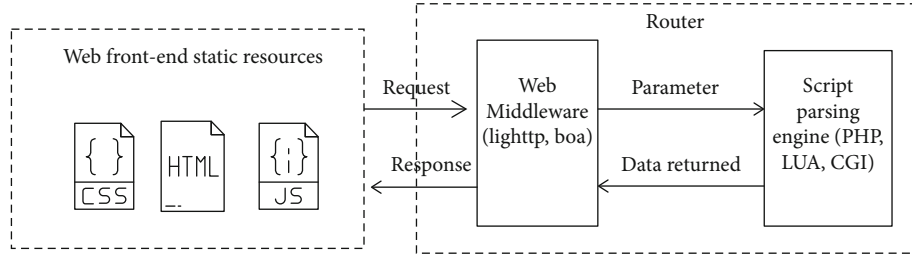


FIGURE 1: Router Web architecture.

TABLE 1: Common router Web vulnerabilities.

Device name	CWE (list of common defects)	Describe
D-Link/DIR-300	CWE-352	Cross-site request forgery
Netgear	CWE-601	URL redirects to untrusted site
D-Link DIR-600/300	CWE-200	Information leakage
D-Link 850L	CWE-319	Sensitive information transmitted in clear text
D-Link/DIR8xx	CWE-295	Improper certificate validation
Billion 7700NR4	CWE-798	Use hardcoded certificates
Link 850L	CWE-798	Use hardcoded certificates
TOTOLINK	CWE-20	Improper certificate validation
TP-Link	CWE-284	Improper access control

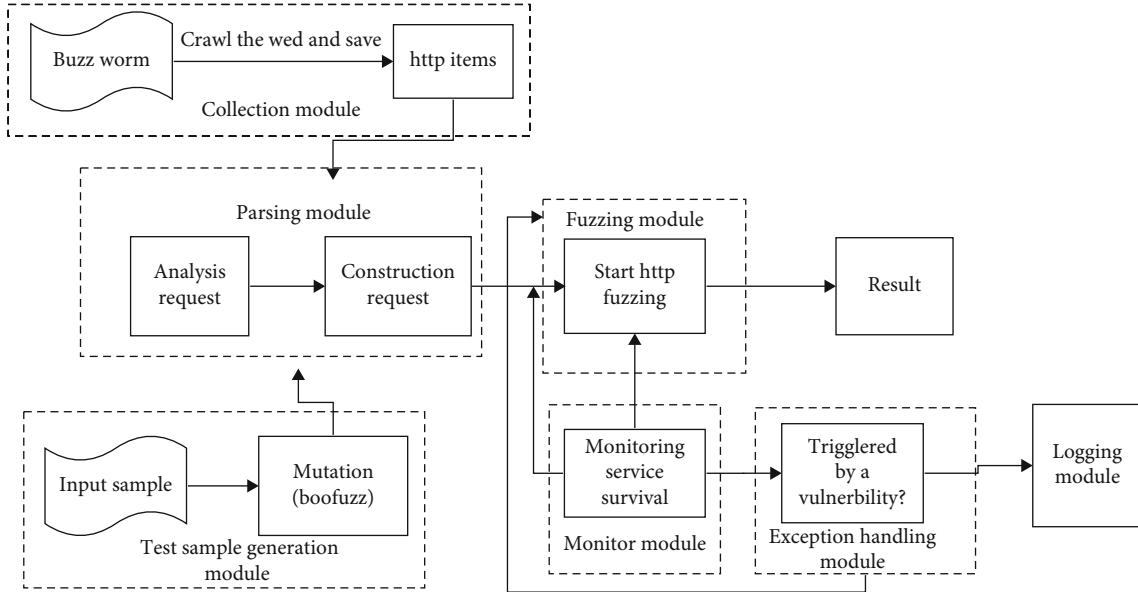


FIGURE 2: RW-fuzzer overall structure.

The existence of router vulnerabilities poses a threat to user security, so the mining of router device vulnerabilities has become a research hotspot in the field of security research.

Most routers develop Web applications based on common Web technologies such as PHP and Javascript for the convenience of users. Then, the applications are used to configure and manage routers' Web interface. This paper pro-

poses a fuzzing test method for router Web-side vulnerability mining in view of the vulnerability risks existing on the router Web interface. The main work and contributions are as follows:

- (1) Export test targets in batch processing, which has higher operational efficiency than the original single import method

---

Algorithm 1 Parse http item from burp items

---

Input: *BurpItems*  
Output: *HTTPItemlist*

```

1: function PARSEHTTPREQUEST(burpitem) file ← open(burpitem)
3:  doc ← XMLParse(file)
4:  while doc do
5:    httpitem ← ParseHttpContent(doci)
6:    httpitemlist ← append(httpitem)
7:  end while
8:  return httpitemlist
9:
10: function PARSEHTTPCONTENT(item) item.header ← getheader(item)
12: item.body ← getbody(item)
13: return item
14: end function = 0

```

---

FIGURE 3: Parsing module pseudocode.

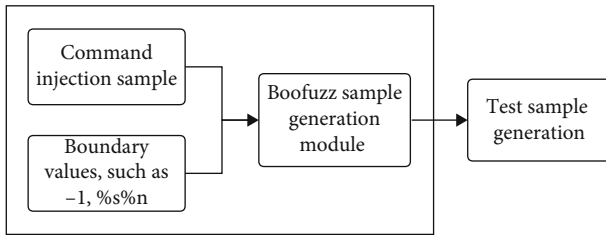


FIGURE 4: Test sample generation flowchart.

---

Algorithm 2 Fuzzing algorithm

---

Input: *HttpItem*  
Output: *Nothing*

```

1: function MAINFUZZING(srcitem, testcase)
2:  while testcasei do
3:    srcitem.parameter ← tesecasei
5:    res ← request(srcitem)
6:  end while
14: end function

```

---

FIGURE 5: Fuzzing module pseudocode.

- (2) The first to apply is the fuzzing method commonly used in software vulnerability testing to the vulnerability detection of the router Web interface. The reconstructed mutation samples and its testing process are close to the request of the real browser. It is more suitable for the attack load on the web interface of the router. It is easier to trigger the exception of the device, showing good operability
- (3) The paper verifies the proposed method. The fuzzing test is carried out on 4 real devices including DP-Link, B-Link, ASUS, and other router equipment

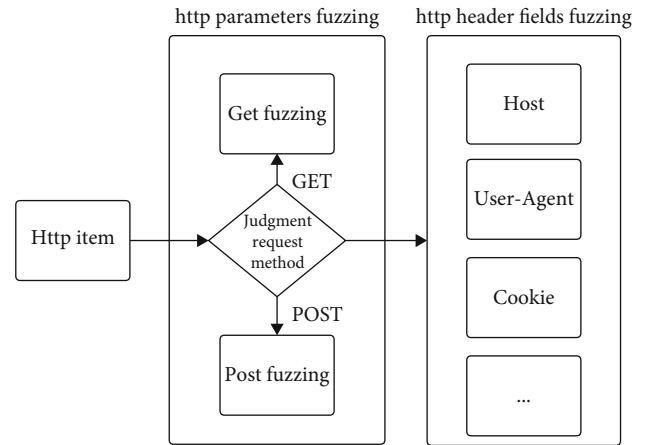


FIGURE 6: Fuzzing module flowchart.

manufacturers, and 4 nday vulnerabilities and a 0day vulnerability are found. The results indicate the validity and applicability of the method

Section 2 of this paper will describe the related work; Section 3 will introduce the router Web-side vulnerability mining fuzz test method (RW-fuzzer); Section 4 uses the method proposed in this paper to test several routers and analyze the test results and summary; Section 5 summarizes the work of this paper and looks forward to the next work.

## 2. Relative Work

**2.1. Routers.** According to the “China Internet Development Report (2021),” by the end of 2020, the number of netizens in my country was 989 million. In the second half of 2020, as many as 15.698 million home routers were sold in the domestic market. The core position of the router makes it play an important role in both personal and corporate

---

Algorithm 3 Monitor algorithm

---

Input: *SourceItem*, *FuzzItem*

Output: *If there is a vulnerability*

```

1: function MONITORSERVER (srcitem, fuzzitem)
2:   if crash then //Detecting memory corruption vulnerability
3:      $i \leftarrow 0$ 
4:     while  $i < \text{threshold}$  do
5:        $res \leftarrow \text{requests}(\text{srcitem})$ 
6:        $i \leftarrow i + 1$ 
7:     end while
8:     return True
9:   end if
10:  sock  $\leftarrow \text{sock}(\text{server.ip}, \text{server.port})$  // detection command injection
11:  state  $\leftarrow \text{sock.connect}()$ 
12:  if state then
13:    return True
14:  end if
15:  if request(srcitem) = request(fuzzitem) then
16:    return True //detecting over permission
17:  end if
18: end function

```

---

FIGURE 7: Monitor module pseudocode.

networks. Most of the routers on the market use the Web terminal for the convenience of users. Reference [2] studied 11,482 kinds of routers and found that the proportion of devices with Web interface was 83.6%. This data shows that most routers are controlled and managed through the Web. The router's Web architecture is shown in Figure 1.

**2.2. Vulnerabilities on the Web Interface of Routers.** While the application of the router Web side brings convenience to users, it also brings many security risks due to the security loopholes in the Web interface. Table 1 lists the common router Web-interface vulnerabilities in recent years. Hackers use these vulnerabilities to attack routers, which will bring great security threats to users.

**2.3. Research on Router Security.** The role of routers in the network makes the study of router security one of the key research directions of security researchers in recent years. Some researchers start from improving router security. Reference [3] maximizes the authentication capability of a router by embedding a device-based MAC address application to monitor user entries. The method periodically changes the password of the router, sends the encrypted password to the user, and at the same time informs the administrator about the change. The router transmits data through the Internet to improve its security. Reference [4] proposes the IDRBT-IoT Secure Router (IISR), which can resist a large number of network attacks. IISR protects the IoT ecosystem by providing a one-step solution, which therefore protects the network from malicious activity. Ref-

erence [5] presents an evaluation study of Manufacturer Usage Descriptions (MUDs), proposing a mechanism to identify and eliminate configuration vulnerabilities before creating a device's MUD profile, which minimizes the attack surface.

More researchers invest in how to detect router vulnerabilities. Literature [6] is mainly used to discover the vulnerabilities of SOHO router devices and realize automatic pattern recognition of multiple types of vulnerabilities in SOHO routers. For the D-CONF operation in popular SOHO routers, the CONF-READ communication model is extended, and it is carried out. In order to improve, an improved SRFuzzer method based on DCONF mode, fuzzification mechanism is proposed, focusing on detecting read operation problems. Reference [7] developed the WNV detector, a general and extensible framework for detecting vulnerabilities in wireless networks, which enables automated and fine-grained device identification and effectively detects security vulnerabilities in wireless routers. Reference [8] proposed the first gray-box fuzzy framework EWWHunter based on embedded device Web front-end for security analysis of embedded devices. EWWHunter can automatically detect authentication bypass vulnerabilities and command injection vulnerabilities in embedded devices by using information from official firmware and logic programs that control the terminal's Web front-end. Reference [9] designed a black-box fuzzing tool named SIoTFuzzer, which can detect the vulnerabilities of IoT devices, design, and implement for automatic discovery of IoT devices' loopholes through device monitoring deployed in the system or built

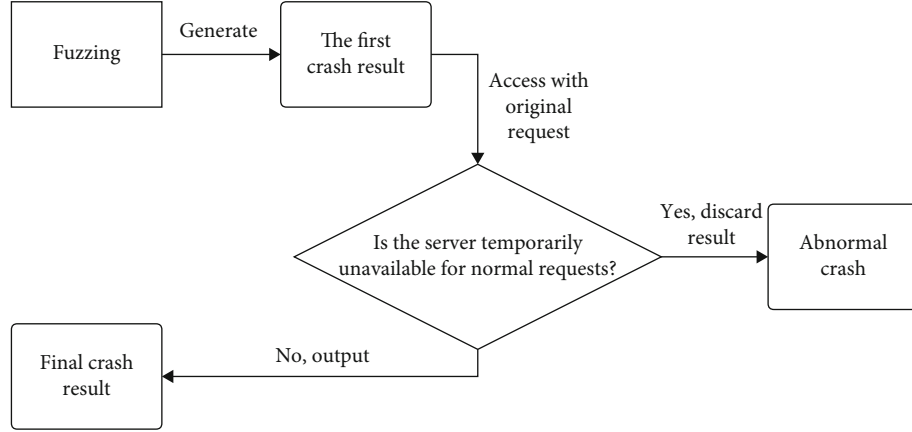


FIGURE 8: Anomaly filtering module flowchart.

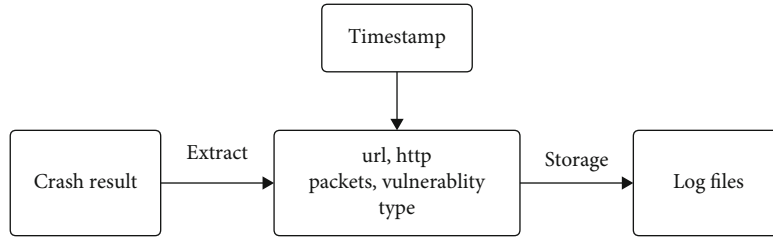


FIGURE 9: Logging module.

```

[2020-04-18 12:45:06] [buffer/stack overflow]
==> [GET] http://192.168.1.1/userRpm/PingIframeRpm.htm
  
```

FIGURE 10: Vulnerability URL logging.

```

{'ping_addr': '123`wget+http://192.168.1.100:8000', 'doT
ype': 'ping', 'isNew': 'new', 'sendNum': '-1', 'pSize': '
64', 'overTime': '800', 'trHops': '20'}
  
```

FIGURE 11: Vulnerable packet logging.

in simulators. The test targets are mainly routers and IP cameras. Reference [10] designed an IoT firmware vulnerability detection platform that integrates offline static detection and online dynamic detection. The platform is able to effectively identify various security weaknesses and risks in firmware such as dangerous processes, exploitable vulnerabilities, and other attack surfaces.

**2.4. Fuzzing Test Technology.** Fuzzing test [11] is a software automation testing method originally developed by Miller in 1989. This method has been widely used in software vulnerability mining and is also used to mine router equipment vulnerabilities, which is still in the exploratory stage [12]. Fuzzing is a simple and effective method for vulnerability testing, and commonly used fuzzing tools include SPIKE, Peach, Sulley, Autodafe, and GPF. Boofuzz [13] is a python-based network protocol fuzzing framework, which is a fork and subsequent version based on the Sulley fuzzing framework. The idea of the Boofuzz framework is to perform fuzz testing based on mutation and can customize the length

of the data block. It also includes excellent session management and monitoring modules, which can efficiently mine potential vulnerabilities of common communication protocols in IoT devices, including http, ftp, and udp. Boofuzz is suitable for constructing fuzz format data packets of network protocols, so this paper improves the framework to be suitable for vulnerability mining of router devices.

Fuzzing testing method is widely used in software vulnerability testing and has shown good performances in software vulnerability testing. In this paper, combining fuzzing testing technology, we propose RW-fuzz method for vulnerability mining on the Web interface of routers.

### 3. Fuzzing Test Method for Router Web-Side Vulnerability Mining

RW-fuzzer is divided into seven modules: collection module, parsing module, test case generation module, fuzzing module, monitoring module, exception handling module, and logging module. RW-fuzzer is implemented based on the python programming language. First, the Burp Suite crawler is used to grab the http items on the Web side of the router device and manually export them to the local and then use the python parsing module to parse the data packets. All packets are then put on a queue, and their web function point interfaces are automatically fuzzed one by one. RW-fuzzer can avoid problems such as repeated data packet capture, repeated definition, and repeated field analysis for each function point. The overall structure of RW-fuzzer is shown in Figure 2.

```

text:00451A4C
text:00451A4C loc_451A4C:          # CODE XREF: sub_451674+3C4↑j
text:00451A4C          la      $t9, httpGetEnv
text:00451A50          la      $a1, aSendnum          # "sendNum"
text:00451A58          jalr   $t9 ; httpGetEnv
text:00451A5C          move   $a0, $s5
text:00451A60          lw      $gp, 0x70+var_58($sp) |
text:00451A64          nop
text:00451A68          la      $t9, atoi
text:00451A6C          nop
text:00451A70          jalr   $t9 ; atoi
text:00451A74          move   $a0, $v0          # nptr
text:00451A78          lw      $gp, 0x70+var_58($sp)
text:00451A7C          la      $a1, aPsize          # "pSize"
text:00451A84          la      $t9, httpGetEnv
text:00451A88          move   $a0, $s5
text:00451A8C          jalr   $t9 ; httpGetEnv
text:00451A90          move   $s4, $v0
text:00451A94          lw      $gp, 0x70+var_58($sp)

```

FIGURE 12: httpd disassembly code.

```

:ext:0047389C
:ext:0047389C loc_47389C:          # CODE XREF: ipAddrDispose+1C0↑j
:ext:004738A0          addu   $v0, $s1, $t0
:ext:004738A4          lb      $v1, 0($v0)
:ext:004738A8          nop
:ext:004738AC          bne    $v1, $s0, loc_4738CC
:ext:004738B0          addiu   $v0, $v1, -0x30
:ext:004738B4          addiu   $v0, $a0, -1
:ext:004738B8          sltiu   $v0, 3
:ext:004738BC          beqz    $v0, loc_473978
:ext:004738C0          move    $a1, $zero
:ext:004738C4          addiu   $t1, 1
:ext:004738C8          b       loc_47395C
:ext:004738CC          move    $a0, $zero

```

FIGURE 13: Illegal memory access disassembled code match.

TABLE 2: Router Web vulnerability discovered.

Equipment manufacturer and model	Vulnerabilities found
D-Link (DIR-816)	CNVD-2020-42655
B-Link (WR-9000)	CVE-2016-6563, CVE-2016-6563
ASUS (RT-68U)	CVE-2018-6636
TP-Link	Buffer overflow

**3.1. Collection Module.** When manually browsing and traversing each function point on the Web side of the router device, turn on the monitoring of the Burp Suite proxy, view some historical data packets, and export them. The collection module uses the proxy monitoring function of Burp Suite to monitor the traffic of each function point on the Web side of the router device, record the detailed information of each http data packet, and finally export all the data packets (http items) to local storage.

**3.2. Parsing Module.** The parsing module is used to process http item packets exported from Burp Suite. First, use python's xmlparse module to parse the http item data to obtain a complete data packet object, including the http request header and http request body. Then, the request parameters and http headers of each data packet are parsed into dictionary format, so that when using python's requests module, it can be easily made http requests. The pseudocode of the parsing module is shown in Figure 3.

**3.3. Test Sample Generation Module.** The test samples are generated by the use samples of the Boofuzz framework, and some basic test samples that can be used to detect command execution and unauthorized vulnerabilities are added. Using the boundary value generation method based on fuzzy heuristics when generating test samples can effectively increase the probability of triggering vulnerabilities in the testing process. The flowchart of test sample generation is shown in Figure 4.

The mutation rules generated by the test cases refer to the IoTfuzzer [14] framework. Typical mutation-based mutation rules include the following: changing the length of the string to test for buffer overflow and randomly filling in the format character "%s%n" to test the format String vulnerability; change all fields that accept numbers to negative numbers or 0xFFFFFFFF to test integer overflow and array out-of-bound vulnerabilities, etc.

**3.4. Fuzzing Module.** First, convert the mutated test cases into keys and values in http packet parameters, such as {"addr": "127.0.0.1", "size", 0xFFFFFFFF}. Secondly, use python's requests module to convert the data in dictionary format to the http packet body. Then, parameter fuzzing is performed on the data packets of the get method or post-method with http parameters (using the control variable method to perform mutation-based fuzzing test on the value of each variable). The pseudocode is shown in Figure 5.

After the parameter fuzzing test is completed, the fuzzing module automatically fuzzes each field of the http packet header and uses different parsing methods for different header fields during testing. For example, for the user-



agent header, only the characters before and after the slash are fuzzed, including Mozilla/5.0 and AppleWebKit/537.36. At this time, the fuzzing test also adopts the control variable strategy and successively replaces the parameters of each variable, thereby increasing the stability of the fuzzing test. The flow of the fuzzing module is shown in Figure 6.

**3.5. Monitoring Module.** In the vulnerability mining of router devices, monitoring of network protocols is relatively easy to implement, and it is only necessary to detect whether remote services are available by sending normal data packets of the corresponding protocol. If the input length is too long, it will cause a buffer overflow, which will cause the service to crash and become temporarily inaccessible (if the device has a watchdog service, it will restart after detecting an abnormality in the service). When testing, first send a request that can cause an exception, then make a normal access request, and set a threshold for the number of requests. If the service is still unavailable after the number of normal access requests exceeds the threshold, it means that the device has a buffer overflow, memory corruption vulnerabilities such as pointer out-of-bounds.

For command injection vulnerabilities, the only need is to inject a malicious command execution string into the data packet during testing and inject the command enclosed in backticks into the payload, for example: "telnetd -p 2309 -l/bin/sh." When the server receives the test command, it will splice the received payload into the system() function parameter string. Due to the characteristics of Linux system commands, the server will execute the injected content first. Therefore, the black-box fuzz method can efficiently detect command injection vulnerabilities. In the above example, the injected command is to start the telnetd service operation. If the server has a command injection vulnerability, the attacker will be able to connect to the corresponding port directly through the socket. The pseudocode of the monitoring module is shown in Figure 7.

**3.6. Exception Filtering Module.** The exception filtering module is used to exclude false positives in fuzzing. In some cases, after getting the result of a service crash, the results of fuzzing may have false positives. For example, some http function interfaces in the background of the router device were triggered during the test, causing the router device to restart normally, and the test result was falsely reported as a service crash caused by a memory corruption vulnerability. Therefore, RW-fuzzer needs to perform secondary detection on the crash report in the final test result and eliminate false positive results by using automated or semimanual analysis methods. The flowchart of the exception filtering module is shown in Figure 8.

**3.7. Logging Module.** The logging module is responsible for recording the http packet information (http item) that caused the device to crash, as well as the timestamp, url, get or post method parameter packet information that triggers the crash, etc., and export the above information to local storage. The flowchart of the logging module is shown in Figure 9.

## 4. Experiments and Result Analysis

In this chapter, the proposed RW-fuzzer method is used to conduct actual fuzzing experiments on several home routers to verify the vulnerability mining effect of the RW-fuzzer method. The following is a detailed description of a certain experimental process. The specific experimental environment is as follows: OS: Ubuntu 16.04, software environment: Burp Suite and Boofuzz, and router equipment: a certain model of TP-Link home router.

**4.1. Procedure.** The IP address of the background Web management service of the experimental router is 192.168.1.1, and the proposed RW-fuzzer is used to mine the Web service for vulnerabilities during the experiment. First, use the collection module to monitor traffic and export the monitored HTTP packets. The exported packets are then parsed using the parsing module, and fuzzing is performed using the test case generation module and the fuzzing module. After the fuzzing test, there are the following vulnerability records in the log file recorded by the logging module (as shown in Figure 10), indicating that the experimental router has a buffer overflow vulnerability. The URL of the vulnerability point is <http://192.168.1.1/userRpm/PingIframeRpm.htm>.

For further vulnerability analysis, we search in the log file for the parameter list of the get method in the data package that causes the vulnerability, which is shown in Figure 11.

The triggering of this vulnerability can cause the router Web service to crash, making the router background service inaccessible for a short period of time. At this time, the browser is used to access the background Web service of the experimental router, and the browser prompts that the Web service on port 80 cannot be connected. Therefore, it is judged that the experiment has triggered a memory corruption vulnerability.

**4.2. Analysis of Results.** By analyzing the fields in the post data packet of the http protocol, it can be concluded that the mutation fields of the vulnerability are ping\_addr and sendNum. The test value of the sendNum field is a boundary value, and it can be determined that this field triggers a memory corruption vulnerability. Next, the firmware of the experimental router was obtained from the official service support of TPLINK [15] and disassembled using binwalk software. After the disassembly is successful, use the IDA software to load the httpd service and analyze the disassembly code (0x00451A4C) of the sendNum string, as shown in Figure 12. It can be seen from the disassembly code that the httpd service uses the atoi function to convert the sendNum parameter in the http protocol received from the Web side.

Further analysis of the execution flow of the return value of the atoi function shows that at the address of 0x004738A0, the value of the v0 register is the result of the addition of the previous addu instruction, as shown in Figure 13. In the addu instruction, the value of the s1 register is a legal address, and the value of the t0 register is the result converted by the atoi function. Since the value of the t0

register during fuzzing is -1, the result of the addu instruction (v0 register) is an invalid address. The experimental router caused an illegal memory access when using this illegal address, causing the service to crash.

In addition to testing the device, fuzzing tests were also performed on 4 real-world devices including D-Link, B-Link, and ASUS router equipment manufacturers, and 4 nday vulnerabilities were found (as shown in Table 2). The effectiveness and applicability of the method are verified by experiments.

**4.3. Comparison.** IoTFuzzer [14] is an application-based fuzzing framework designed to discover memory corruption in physical IoT devices without firmware images. It leverages fuzzing and taint analysis, focusing only on moving to the Web interface and detecting only through activity check memory corruption, and the framework is not strong enough to find Web server vulnerabilities. This article refers to the mutation rules generated by test cases in its framework to find Web-side vulnerabilities. Reference [16] uses qemu to simulate the web interface of some Linux-based devices. This method does not work with Web interfaces that contain special hardware-related operations such as Wi-Fi configuration. The most popular solution for finding vulnerabilities is to obfuscate the SOHO router's Web server module. However, its effectiveness is limited by the lack of input specifications, the router's internal operational state, and a test environment recovery mechanism. The RW-fuzzer method proposed in this paper is effective and applicable to the router vulnerability mining on the Web side.

## 5. Conclusion and Outlook

This paper proposes a RW-fuzzer method, which uses the http packets collected by the Burp Suite tool as the input of the fuzzing test. And the algorithm rules of the Boofuzz framework are improved using a mutated use sample generation algorithm, which automatically generates mutated samples. Thus, we can mine the vulnerabilities on the Web side of the router. Based on RW-fuzzer, the paper conducted fuzzing tests on 4 actual devices including D-Link, B-Link, ASUS, and other router equipment manufacturers and found 4 nday vulnerabilities and one 0day vulnerability. Experiments show that the RW-fuzzer method is effective and applicable. However, this method still has the defects of high tool false positive rate, low single-threaded request rate, and long time spent traversing each function point on the Web page of the device. In the future work, we will study to optimize the request speed of Web fuzzing and improve it into a multithreaded way in the main fuzzing module request and monitoring module. And we will improve the data parsing module and optimize the parsing rate, so that it supports more http protocol packet formats such as xml and json.

## Data Availability

The data type used to support the findings of this study is included within the article.

## Conflicts of Interest

The authors declare that they have no conflicts of interest.

## Acknowledgments

This paper was supported in part by the National Natural Science Foundation of China under Grant U19B2044, the National Natural Science Foundation of China (62162020), the Hainan Province Science and Technology Special Fund (ZDYF2021GXJS216), and the Science Project of Hainan University (KYQD(ZR)20021).

## References

- [1] "360 Security research Institute," <https://mp.weixin.qq.com/s/zXcVijFVtTN13D12AzPetA>. 2019-12-03.
- [2] D. Wang, X. Zhang, T. Chen, and J. Li, "Discovering vulnerabilities in COTS IoT devices through blackbox fuzzing web management interface [J]. Security and Communication," *Networks*, vol. 2019, pp. 1–19, 2019.
- [3] D. S. Tejendra, C. R. Varunkumar, S. L. Sriram, V. Sumathy, and C. K. Thejeshwari, "A novel approach to reduce vulnerability on router by zero vulnerability encrypted password in router (ZERO) mechanism," in *Proceedings of the 2019 3rd International Conference on Computing and Communications Technologies (ICCCCT)*, pp. 163–167, Chennai, India, 2019.
- [4] A. Dua, V. Tyagi, N. D. Patel, and B. M. Mehtre, "Iisr: a secure router for iot networks," in *Proceedings of the 2019 4th International Conference on Information Systems and Computer Networks (ISCON)*, pp. 636–643, Mathura, India, 2019.
- [5] S. M. Sajjad, M. Yousaf, H. Afzal, and M. R. Mufti, "eMUD: enhanced manufacturer usage description for IoT botnets prevention on home WiFi routers," *IEEE Access*, vol. 8, pp. 164200–164213, 2020.
- [6] Y. Zhang, W. Huo, K. Jian et al., "Srfuzzer: an automatic fuzzing framework for physical SOHO router devices to discover multi-type vulnerabilities," in *Proceedings of the 35th Annual Computer Security Applications Conference*, pp. 544–556, San Juan Puerto Rico USA, 2019.
- [7] Y. Huang, F. Zhu, L. Liu et al., "WNV-detector: automated and scalable detection of wireless network vulnerabilities," *EURASIP Journal on Wireless Communications and Networking*, vol. 2021, no. 1, 2021.
- [8] E. Wang, B. Wang, W. Xie, Z. Wang, Z. Luo, and T. Yue, "EWWHunter: grey-box fuzzing with knowledge guide on embedded web front-ends," *Applied Sciences*, vol. 10, no. 11, p. 4015, 2020.
- [9] H. Zhang, K. Lu, X. Zhou, Q. Yin, P. Wang, and T. Yue, "SIoT-Fuzzer: fuzzing web interface in IoT firmware via stateful message generation," *Applied Sciences*, vol. 11, no. 7, p. 3120, 2021.
- [10] D. He, H. Gu, T. Li et al., "Toward hybrid static-dynamic detection of vulnerabilities in IoT firmware," *IEEE Network*, vol. 35, no. 2, pp. 202–207, 2020.
- [11] A. Takanen, J. D. Demott, C. Miller, and A. Kettunen, *Fuzzing for Software Security Testing and Quality Assurance*, Artech House, 2018.
- [12] J. M. Heo, J. M. Kim, C. M. Ji, and M. P. Hong, "Emulation-based fuzzing techniques for identifying web interface vulnerabilities in embedded device firmware," *Journal of the Korea*



*Institute of Information Security & Cryptology*, vol. 29, no. 6, article 1225, 2019.

- [13] “Boofuzz,” <https://boofuzz.readthedocs.io/en/stable/>.
- [14] J. Chen, W. Diao, Q. Zhao et al., “IOTFUZZER: discovering memory corruptions in IoT through App-based fuzzing,” in *Proceedings of the 2018 Network and Distributed Systems Security (NDSS) Symposium*, USA, 2018.
- [15] “TPLINK官网,” <https://service.tp-link.com.cn/>, 2020-04-30.
- [16] A. Costin, A. Zarras, and A. Francillon, “Automated dynamic firmware analysis at scale: a case study on embedded web interfaces,” in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, pp. 437–448, Xi'an China, 2016.