Possibilities of Using Fuzz Testing in Smart Cities Applications

Lubomir Almer, Josef Horalek, and Tomas Svoboda(✉)

Faculty of Management and Informatics, University of Hradec Kralove, Hradec Kralove, Czech Republic
{lubomir.almer,josef.horalek,tomas.svoboda}@uhk.cz

Abstract. In recent years, technological advances in various fields of human activity have enabled the development of smart city applications that can help improve life in modern cities. In order to validate that the functional requirements of the applications are met and, above all, to ensure security and resilience against vulnerabilities and constantly evolving cyber threats, it is imperative that these applications are adequately tested before being put into operation. The aim of this paper is to present an analysis of the use of fuzz testing to test the stability, correctness and security of applications and information systems that are applicable in the smart city domain. The paper presents an analysis of the possibilities of using different types of fuzz testing and maps the different fuzz testing tools that are applicable in implementation projects in different areas of smart cities. Furthermore, a testing method for the use of fuzz testing is proposed and presented. This method is then validated using a set of proposed tests and outputs for a selected project.

Keywords: testing · fuzz testing · application security · information systems · security · smart cities · smart application

1 Smart Cities Application Testing

In the last more than a decade, the smart city concept has spread worldwide and influences the design of city strategies regardless of their geographical location, population and existing models and processes for providing services to residents [1]. The concept of smart cities is based on the principle of using modern ICT services, including the areas of IoT and cloud computing. The use of modern information technologies and applications brings several problems and challenges, which are analyzed in the article Privacy in the Smart City-Applications, Technologies, Challenges, and Solutions [2]. These are challenges related to ensuring cybersecurity of applications and technologies in smart cities and comprehensive protection of data based on their confidentiality or sensitivity [3]. Globally, the number and severity of cyber-attacks have been increasing in recent years, and they no longer target only IT technologies, but increasingly also industrial and OT technologies [4]. It cannot be assumed that cyber-attacks will not also target

P. Cong Vinh and H. Mahfooz Ul Haque (Eds.): ICTCC 2023, LNICST 586, pp. 85–97, 2024.
https://doi.org/10.1007/978-3-031-59462-5_6

smart city applications and solutions in the future, and it is therefore imperative to address the issue of having adequate or designing new methods and approaches on how to implement adequate protection for smart city applications and IT solutions [5]. The basic requirements for cybersecurity of applications in smart cities are the issues of ensuring secure software development and its adequate testing before putting it into production operation to ensure software quality [6].

2 Related Work

Cybersecurity Curriculum Design: A Survey [7] addresses the issue of effective software quality assurance with emphasis on the necessary involvement of the programmers of these applications in particular, who can ensure an adequate level of application security through properly chosen techniques and design patterns. Application programmers are complemented by the article Processes, Systems & Tests: Defining Contextual Equivalences. [8] to include application testers who are responsible for the actual testing of the application, i.e., proving that the required functionality is consistent with the software specification. The idea of software quality assurance in smart cities is addressed in the paper Testing Strategies for Smart Cities applications [9], where approaches and documented testing strategies in smart cities are analyzed. These include testing via test-bed, testing in the lab and testing on a simulator. However, the authors state that unit tests can also be used for application testing. A similar perspective on testing is then presented in The Campus as a Smart City: University of Málaga Environmental, Learning, and Research Approaches [10]. A specific approach to application-specific testing, based on testing for the smart traffic control domain through unit tests of smart city applications, is presented by the authors of the article Design and Implementation of a Smart Traffic Signal Control System for Smart City Applications [11]. A new approach to application testing using fuzz testing (fuzzing) for information and cybersecurity assurance, with an emphasis on detecting application vulnerabilities in smart cities and reducing the testing time, is presented in the paper Integrating Fuzz Testing into the Cybersecurity Validation Strategy [12]. A similar approach is then presented in [13], which considers fuzz testing as part of application robustness and stability validation. Fuzzing is a testing technique for finding vulnerabilities in software applications by sending unexpected input data to target systems and applications and then monitoring the results [14]. It is an automatic or semi-automatic process that involves sending and repeatedly manipulating data to an application [15]. Fuzzers can be broadly classified into two categories: mutation-based fuzzers and generation-based fuzzers. Mutation-based fuzzers apply mutations to existing data samples to create a test space. Generation-based fuzzers create test cases from scratch by modeling the target protocol or file format.

2.1 Fuzz Testing Types

Blackbox Fuzzing: This is a technique where the fuzzer has no information about the internal state and implementation of the application. The application is a black box - it has only inputs and outputs visible. Test files are inserted on the application input. Fuzzer monitors the output and behavior of the program. Blackbox fuzzing is not used

to find specific vulnerabilities but is used to identify conditions that create exceptions in the code and cause the target application to crash [16]. In other words, it is used to find unknown and undiscovered vulnerabilities. Blackbox testing not only allows the emulation of the view of cyber attackers but is an essential tool when the source code of an application is not available.

Whitebox Fuzzing: White box fuzzers analyze the internal structure of a program and learn to monitor and maximize code coverage or extreme value testing with each successive execution. White box fuzzers typically use adaptive algorithms and intelligent instrumentation during fuzz testing, which makes them more efficient and accurate in detecting vulnerabilities [17].

Greybox Fuzzing: A fuzzer takes a lightweight approach to test generation. This approach effectively reveals vulnerabilities and weaknesses. Greybox fuzzers randomly mutate program inputs to execute new paths. However, this makes it difficult to cover code that is guarded by tight controls [18].

2.2 Fuzz Testing Phases

Fuzzing testing consists of the following phases [19, 20]:

1. **Target identification** - an optional phase used only by potential attackers. Testers already have their target identified.
2. **Input identification** - the goal is to analyze and describe the area for a possible attack. Exis-sues many forms of inputs to the application: network, files, registries, environment variables, command line commands, and more. There are several tools that can be used to identify inputs. These include Command Line Arguments, Environment Variables (ShareFuzz), Web Applications (WebFuzz), File Formats (FileFuzz), Network Protocols (SPIKE), and many others.
3. **Fuzz Data Generation** - The purpose of the fuzzer is to test for vulnerabilities that are accessible through inputs in the application. Therefore, the fuzzer must generate test data that should delineate the target input space, which can then be passed to the target application's input. Test data can either be generated before testing, or more often iteratively generated on demand at the beginning of each test suite. A general approach to fuzz testing is to iteratively provide test instances to the target and monitoring the response. If, during testing, a test case is found to have caused an application failure, the combination of the specific test case and information about the nature of the failure it caused constitutes a defect report. Defect reports can be considered as the final output of fuzz testing for forwarding to developers [13].
4. **Exception Monitoring** - The generation and execution of test data that trigger manifestations of software defects must be detected. This is achieved by using an oracle, a generic term for a software component that monitors a target and reports a failure or exception. An oracle can take the form of a simple check, or it can be a debugger running on the target that monitors for exceptions and collects detailed logging information [13].

5. **Determining exploitability** - Once defects on the application are identified, it is necessary to process this report and forward it to the developers for fixing. The tester can also investigate the defects in question and determine the level of exploitability risk.

2.3 Testing Environment and Methodology

To demonstrate this procedure, a web application with a microservice architecture will be tested. JAVA 8 was chosen for the backend of this application, the frontend is rendered in Coffee script, Oracle database is used for data. This is an application that is used in practice for banking system. Partial fuzzers will be applied to the above application to scan the application. For testing purposes, a test environment was set up and the following activities were performed:

- Identification of entry points.
- Generation/creation of random data.
- Running Fuzz testing.
- Analysis of testing outputs.
- Repeating fuzz tests if necessary.

2.4 Fuzz Test Design

There will be white-box testing, as the tested application is known, and the source code is available. The goal is to verify the functionality of the application and find any undetected bugs. During fuzz testing, several types of responses to requests sent by fuzzers can be encountered. These responses are shown in Table 1.

Table 1. Response code descriptions.

Code	Description
200	Successful response - server to process request
204	Successful response without content, with header only
301	Permanent redirection to the URL specified in the response header
302	Request was received and redirected to another URL
303	Redirection to a new URL, usually a POST request
401	Unauthorized access
403	URL was not recognized
405	The method used (GET, POST, PUT, DELETE) is not allowed
500	Server did not process the request

The tested application contains a set of forms that could contain potential errors. It is also designed to scan URLs and hidden folders that could be exploited by an attacker. Cookies will also be fuzzed.

3 Fuzz Testing Outputs

This chapter describes the individual tests and their outputs. The tests include form, URL, cookie, and hidden page and folder tests. Since this was a known application, the first phase of testing (target identification) was skipped and only the following phases were implemented in the testing.

3.1 Fuzz Testing - Forms

The wfuzz tool will be used to test the forms. An internet banking application is identified as the target. After the necessary installation of python, pycurl and colorama modules the tool can be run. First, it is necessary to get through the authorization. This can be done using several methods, such as invoking curl with a POST request and providing login credentials. Then the focus can be on the forms themselves. In the case of this test, the forms for changing the username, submitting a transaction, and changing the password will be fuzzied.

Test A: Changing Your Username

- *Input identification*: text input in the form - field for username.
- *Data generation*: created dictionary usernames.txt, which contains non-valid values.
- *Running the test*:

```
Wfuzz -c -z file,wordlist/general/usernames.txt -d
"alias=FUZZ" url_adresa
```

- *Test result*: the result shows an attempt to redirect. In this case it is a redirect to the authorization component and then the request is processed on the server. The result of this test cannot be considered completely valid. Due to the security of the application, the final tests had to be done manually (Fig. 1).

Test B: Sending the Transaction

- *Identification of inputs:* form inputs for beneficiary account, variable symbol, part ku and message.
- *Data generation:* a dictionary bankccounts.txt was created for the beneficiary accounts, varsymbols.txt for the variable symbol and amount.txt for the amount. The character length t-testing for the message will only take place in a command where a disproportionately long character string is entered.

```
=====================================================================
ID           Response   Lines    Word       Chars       Payload
=====================================================================

000000009:   303        7 L      23 W       256 Ch      "%%"
000000006:   303        7 L      23 W       256 Ch      "5555"
000000001:   303        7 L      23 W       256 Ch      "usernames"
000000003:   303        7 L      23 W       256 Ch      "ff"
000000007:   303        7 L      23 W       256 Ch      "vyvoda"
000000011:   303        7 L      23 W       256 Ch      "12345678912345678912345678 9"
000000002:   303        7 L      23 W       256 Ch      ".."
000000010:   303        7 L      23 W       256 Ch      "qwertzuiopasdfghjhjklyc"
000000004:   303        7 L      23 W       256 Ch      "12"
000000008:   303        7 L      23 W       256 Ch      "==="
000000012:   303        7 L      23 W       256 Ch      "MmM"
000000005:   303        7 L      23 W       256 Ch      "123"

Total time: 0.501946
Processed Requests: 12
Filtered Requests: 0
Requests/sec.: 23.90694
```

Fig. 1. Fuzzing output for username changes.

- *Running the test:*

```
wfuzz -f output.txt -c -z file,wordlist/gen-
eral/bankaccounts.txt -z file,wordlist/gen-
eral/varsymbols.txt -z file,wordlist/gen-
eral/amount.txt -d "accountNumberCreditIn-
put=FUZZ&bankCodeCredit=0100&variableSymbol=FUZ2Z&a
mount=FUZ3Z&textarea=longlongtexttextextetxtextetxt
etextetxtettxtettxfttextextextextextextextetxe-
txtextetxetxtextetxtetxtetxtext" Url_adress
```

- *Test result:* in this case it is a redirect to the authorization component and then the request is processed on the server (Fig. 2).

Test C: Change Password

- *Identification of inputs*: input for new password and input for re-entering this password.
- *Data generation*: a dictionary passwords.txt was created to change the password when the password violates the mentioned conditions.
- *Running the test*:

```
wfuzz -f outputPasswords.txt -c -z file,word-
list/general/passwords.txt -d "currentPass-
word=Heslo123&newPassword=FUZZ&reNewPassword=FUZZ"
```

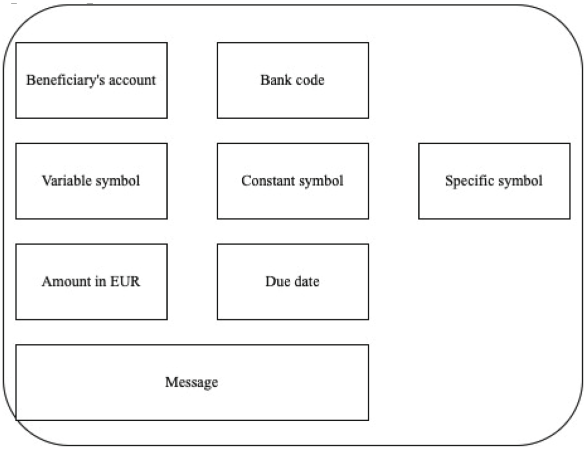

Fig. 2. Transaction form

- *Evaluation of results:* wfuzz seems to be a very useful tool for form fuzzing, but there are some limitations. If the application is already deployed and includes an authorization component for successful form submission, the test results cannot be considered. At this point, it is already much more convenient to take advantage of the implemented automated tests and test the extreme values of the forms. It is more advisable to use wfuzz during initial development, when testers can generate dictionaries with extreme values (Fig. 3).

Fig. 3. Fuzzing result for password change

3.2 Fuzz Testing – URL

URL fuzzing was also performed using the wfuzz tool. The target application this time was the operator console, which does not have two-factor authentication.

- *Input identification*: URL of the operator console.
- *Data generation*: default dictionary common.txt, containing keywords.
- *Running test A:* in the first step, the link was fuzzed before authentication.

```
Wfuzz - z file,wordlist/general/common.txt --hc
404,403 url_adresa/FUZZ
```

- *Test results A*: the result for this test is 3 potentially exploitable addresses:

  – https://url_adresa/client
  – https://url_adresa/css
  – https://url_adresa/js

However, the keyword addresses have been checked and are redirects that are subsequently blocked due to insufficient permissions (Fig. 4).

Fig. 4. Fuzzing URL – test A

- *Running test B*: next, the test focused on URL discovery after authorization to the apli-cache. Before the actual test, it is necessary to find out the valid sessionid of the logged in user and then use it.

```
Wfuzz -z file,wordlist/general/common.txt --hc
404,403 -b "sessionid=566d78ce-998f-458f-a0cd-ee-
bec0bb4f8b" url_adresa/FUZZ
```

- *Test result B*: in this case no potential address for abuse was detected (Fig. 5).

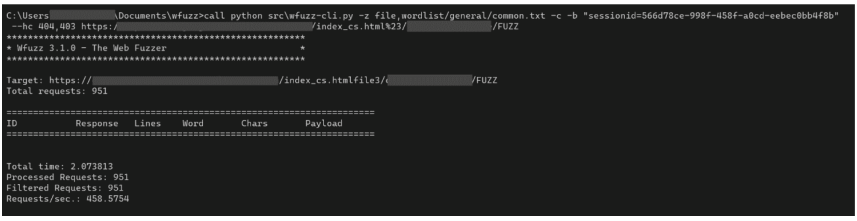

Fig. 5. Fuzzing URL for authorization

3.3 Fuzz Testing – Cookies

The wfuzz tool will again be used for testing cookies. The tool contains a default wordlist that contains the generated values.

- *Input identification:* Application URL
- *Data generation:* default dictionary common.txt, containing keywords
- *Running the test*:

```
wfuzz -z file,wordlist/general/common.txt -b
"cookie=FUZZ" url_adresa
```

- *Test result:* when used on the app, the answer 200 can be seen. So the server answered the call. Valid cookies values were found, but other factors such as https encryption and cookie expiration time need to be taken into account (Fig. 6).

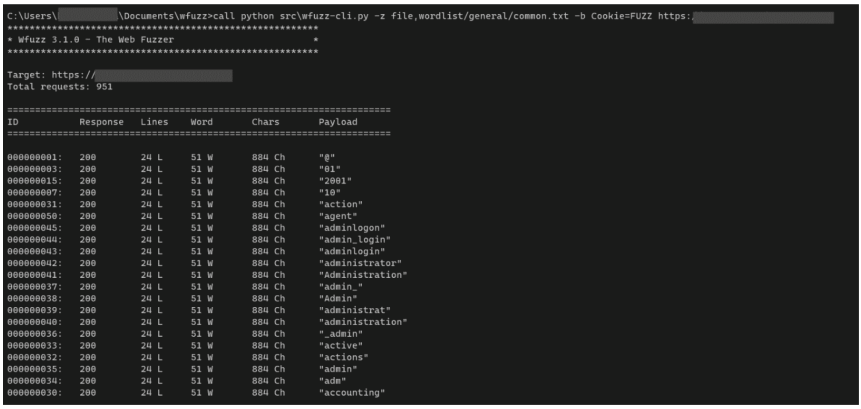

Fig. 6. Fuzzing cookies

3.4 Fuzz Testing - Hidden Pages and Folders

The ffuf tool was chosen for this type of testing and is easy to install. It works by generating requests from the dictionary.

- *Input identification*: application URL
- *Data generation*: dictionary admin-panel, which can be found from the attachments.
- *Running the test*: `ffuf -c -w admin-panels.txt -u url_adresa`
- *Test results*: two potential vulnerabilities were found in this case. It should be noted that this is a 302 status. The pages were then manually scanned and both were redirected to the default 404 page (Fig. 7).

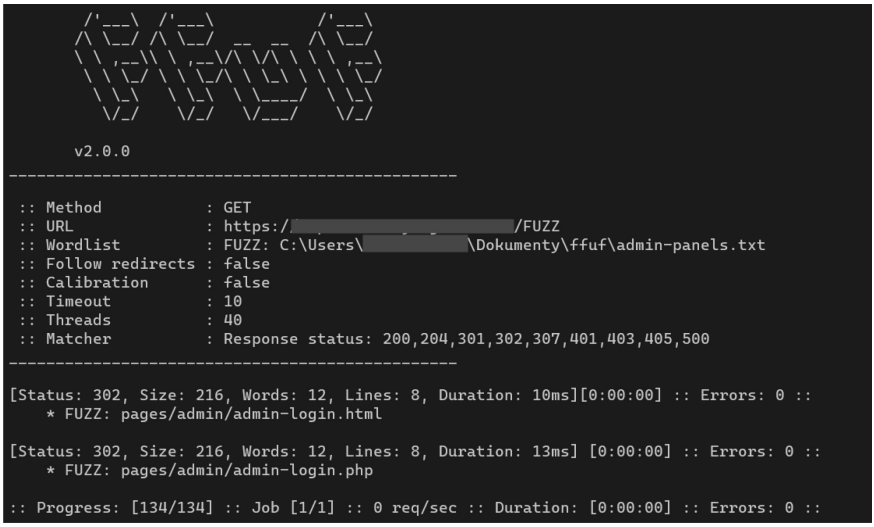

Fig. 7. Fuzzing hidden pages and folders

4 Outputs Analysis

In the first set of tests, forms were validated using the fuzz tool. The setup of the tool is timesaving and the commands are intuitive. The tool also includes a set of already-generated dictionaries that can be used. In testing, it was necessary to get through the initial authentication and then fuzz the forms. However, the forms are secured by an additional authorization component. Therefore, they had to be retested with automated tests and manually.

URL testing was done with fuzz on an Internet banking operator application where there is no need to go through two-factor authentication. Testing revealed 3 sites that were not rejected by the server but were nevertheless redirected to a secure site. The sites found were verified and the error was ruled out.

URL testing is more specific to black box testing to scan for vulnerabilities. For URL testing, it would be more appropriate to use specialized penetration testing tools such as OWASP ZAP. The cookie test revealed interesting results, where any value can be inserted into cookies. Here, however, HTTPS encryption must be considered.

For tests of differently generated request headers, the tool is efficient and can detect non-valid inputs that pass within a short while. In combination with the Burp Suite tool, which is used in security testing and can be used to send requests with modified headers, it is possible to identify errors in the headers. The fun tool was used in testing hidden folders. The tool can be put into operation very quickly and testing can begin. However, it offers very similar capabilities to fuzz and does not contain basically generated dictionaries. This test revealed two potential hidden pages, which were immediately investigated. However, the links have redirect to a 404 page and there is no way to get anywhere through them. Again, it would be more efficient to use a dedicated scan tool

to scan the folders. The dictionaries generated in this case cannot be sufficient to cover all sites and it would be more appropriate to scan instead of fuzzing (Table 2).

Table 2. Summary of testing results.

TEST	RESULT
TEST OF FORMS A	Critical error not detected
FORMS TEST B	Critical error not detected
FORMS TEST C	Critical error not detected
URL TEST	Critical error not detected
COOKIES TEST	Low priority finding
HIDDEN FOLDERS TEST	Critical error not detected

Fuzz testing a web application for Internet banking is very specific and many tools cannot be applied to it. Introducing new technologies would also have negative impacts on the funding of the whole project. Since the critical bug was not observed, it would be difficult to justify the decision on fuzzes. The non-discovery of a critical bug may also be related to the fact that the application has been deployed in live operation for several years and is regularly tested by penetration testing and security experts. Testing the web application using fuzzes revealed many pitfalls and pitfalls. From a project management perspective, deploying fuzz tools can be very time-consuming, and from the customer's side, deploying these types of tests can be pointless due to finances. Quality fuzz tests require skilled professionals and a lot of time to select the right tools, install and create special tests. However, some basic fuzzing and input generation can be used during development itself before the application is deployed into live operation. There are many tools for fuzz testing, but their use is very specific. Many of them are limited to one particular language or only generate data. Using them in the development process is useful, but there's no need to spend hours unnecessarily inventing the same tests. Fuzzing is also an essential part of software security.

Conclusion

This paper presents approaches and techniques for using fuzzy testing to accelerate application development and deployment. The current era calls for the rapid and dynamic development of client applications, where the speed and surpassing of development often go against the fuzziness of the developed applications both at the back end and front end. In this respect, the application support of Smart City projects is no exception, where applications form the user interface for end users to the Smart City outputs. The chosen tests were deployed on backend applications where security is one of the most important components and the presented tests have unambiguously demonstrated the effectiveness of fuzzy tests even on such critical applications. We are convinced that the extension of the fuzzy test suite to applications, not only in the area of Smart City-related products but in general all applications used.

Acknowledgment. The financial support of the project "Application of Artificial Intelligence for Ensuring Cyber Security in Smart City" (ARTISEC), n. VJ02010016, granted by the Ministry of the Interior of the Czech Republic is gratefully acknowledged.

References

1. Manville, C., Cochrane G., Kotterink, B.: Mapping Smart Cities in the EU. EPRS: European Parliamentary Research Service, Belgium (2014). CID: 20.500.12592/0scs9f. https://policycommons.net/artifacts/1339578/mapping-smart-cities-in-the-eu/1949353/. Accessed 10 Oct 2022
2. Eckhoff, D., Wagner, I.: Privacy in the smart city—applications, technologies, challenges, and solutions. IEEE Commun. Surv. Tut. **20**(1), 489–516 (2018). ISSN 1553-877X. https://doi.org/10.1109/COMST.2017.2748998. Accessed 16 Jun 2023
3. Alamer, M., Almaiah, M.A.: Cybersecurity in smart city: a systematic mapping study. In: 2021 International Conference on Information Technology (ICIT), 14 July 2021. IEEE (2021). ISBN 978-1-6654-2870-5. https://doi.org/10.1109/ICIT52682.2021.9491123. Accessed 16 Jun 2023
4. Hero, A., et al.: Statistics and data science for cybersecurity. Harvard Data Sci. Rev. **5**(1) (2023). https://doi.org/10.1162/99608f92.a42024d0. Accessed 16 Jun 2023
5. Chen, D., Wawrzynski, P., Lv, Z.: Cyber security in smart cities: a review of deep learning-based applications and case studies. Sustain. Cities Soc. **66** (2021). ISSN 22106707. https://doi.org/10.1016/j.scs.2020.102655. Accessed 16 Jun 2023
6. Lim, C., Cho, G.-H., Kim, J.: Understanding the linkages of smart-city technologies and applications: key lessons from a text mining approach and a call for future research. Technol. Forecast. Soc. Change **170** (2021). ISSN 00401625. https://doi.org/10.1016/j.techfore.2021.120893. Accessed 16 Jun 2023
7. Mouheb, D., Abbas, S., Merabti, M.: Cybersecurity curriculum design: a survey. In: Pan, Z., Cheok, A.D., Müller, W., Zhang, M., El Rhalibi, A., Kifayat, K. (eds.) Transactions on Edutainment XV. LNCS, vol. 11345, pp. 93–107. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-662-59351-6_9 Accessed 16 Jun 2023
8. Aubert, C., Varacca, D.: Processes, systems & tests: defining contextual equivalences. Electron. Proc. Theoret. Comput. Sci. **347**, 1–21 (2021). ISSN 2075-2180. https://doi.org/10.4204/EPTCS.347.1. Accessed 16 Jun 2023
9. Costa, A., Teixeira, L.: Testing strategies for smart cities applications. In: Proceedings of the III Brazilian Symposium on Systematic and Automated Software Testing, 17 September 2018, pp. 20–28. ACM, New York (2018). ISBN 9781450365550. https://doi.org/10.1145/3266003.3266005. Accessed 16 Jun 2023
10. Fortes, S., et al.: The campus as a smart city: University of Málaga environmental, learning, and research approaches. Sensors **19**(6) (2019). ISSN 1424-8220. https://doi.org/10.3390/s19061349. Accessed 16 Jun 2023
11. Lee, W.-H., Chiu, C.-Y.: Design and implementation of a smart traffic signal control system for smart city applications. Sensors **20**(2) (2020). ISSN 1424-8220. https://doi.org/10.3390/s20020508. Accessed 16 Jun 2023
12. Vinzenz, N., Oka, D.K.: Integrating fuzz testing into the cybersecurity validation strategy. In: Proceedings of the SAE WCX Digital Summit, 06 April 2021. https://doi.org/10.4271/2021-01-0139. Accessed 16 Jun 2023

13. Lämmel, P., Tcholtchev, N., Schieferdecker, I.: Enhancing cloud based data platforms for smart cities with authentication and authorization features. In: Companion Proceedings of the 10th International Conference on Utility and Cloud Computing, 05 Dec 2017, pp. 167–172. ACM, New York (2017). ISBN 9781450351959. https://doi.org/10.1145/3147234.3148087. Accessed 16 Jun 2023
14. Rodriguez, L.G.A., Batista, D.M.: Resource-intensive fuzzing for MQTT brokers: state of the art, performance evaluation, and open issues. IEEE Netw. Lett. **5**(2), 100–104 (2023). ISSN 2576-3156. https://doi.org/10.1109/LNET.2023.3263556. Accessed 16 Jun 2023
15. Hernández Ramos, S., Villalba, M.T., Lacuesta, R.: MQTT security: a novel fuzzing approach. Wirel. Commun. Mob. Comput. **2018**, 1–11 (2018). ISSN 1530-8669. https://doi.org/10.1155/2018/8261746. Accessed 16 Jun 2023
16. Wüstholz, V., Christakis, M.: Harvey: a greybox fuzzer for smart contracts. In: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 08 November 2020, pp. 1398–1409. ACM, New York (2020). ISBN 9781450370431. https://doi.org/10.1145/3368089.3417064. Accessed 16 Jun 2023
17. Godefroid, P.: Fuzzing. Commun. ACM **63**(2), 70–76 (2020). ISSN 0001-0782. https://doi.org/10.1145/3363824. Accessed 16 Jun 2023
18. Pham, V.-T., Bohme, M., Roychoudhury, A.: AFLNET: a greybox fuzzer for network protocols. In: 2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST), pp. 460–465. IEEE (2020). ISBN 978-1-7281-5778-8. https://doi.org/10.1109/ICST46399.2020.00062. Accessed 16 Jun 2023
19. Eceiza, M., Flores, J.L., Iturbe, M.: Fuzzing the Internet of Things: a review on the techniques and challenges for efficient vulnerability discovery in embedded systems. IEEE IoT J. **8**(13), 10390–10411 (2021). ISSN 2327-4662. https://doi.org/10.1109/JIOT.2021.3056179. Accessed 16 Jun 2023
20. Beaman, C., Redbourne, M., Mummery, J.D., Hakak, S.: Fuzzing vulnerability discovery techniques: survey, challenges and future directions. Comput. Secur. **120** (2022). ISSN 01674048. https://doi.org/10.1016/j.cose.2022.102813. Accessed 16 Jun 2023