КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 2 (14), 2021
ISSN 2663 - 4023
DOI 10.28925/2663-4023.2021.14.107117
УДК 004.03

Ilyenko Anna
Candidate of Technical Sciences, assistant professor , assistant professor of Information Security Systems Department National Aviation University of Kyiv, Faculty of Cyber Security, Computer and Software Engineering, Ukraine
ORCID ID: 0000-0001-8565-1117
ilyenko.a.v@nau.edu.ua

Ilyenko Sergii
Candidate of Technical Sciences, assistant professor , assistant professor of Automation and Energy Management Department National Aviation University of Kyiv, Aerospace Faculty, Ukraine
ORCID ID: 0000-0002-0437-0995
ilyenko.s.s@nau.edu.ua

Kulish Tatiana
Student Information Security Systems Department
National Aviation University of Kyiv, Faculty of Cyber Security, Computer and Software Engineering, Ukraine
ORCID ID: 0000-0001-8413-9154
teti98kulish@gmail.com

PROSPECTIVE METHODS OF PROTECTING THE FRAMEWORK WEB APPLICATION ON THE GRADLE AUTOMATIC ASSEMBLY SYSTEM

Abstract. The article considers the problem of providing protection of the web framework application in the system of automatic build gradle and defines perspective methods of providing protection. This article defines that the basic and generally accepted approach to ensuring the security of a web application is a properly constructed test framework. During research the analysis of modern protection methods of web application of the framework based the automatic assembly gradle system is made. A comparative analysis of methods is also included.The basic approaches and methods concerning the organization of application protection are defined on the basis of the modern framework analysis. During planning a test framework, the automator is faced with the task of choosing methods that will solve the problem, will be flexible to change, easy to read and are fast for finding application vulnerabilities. At the stage of developing a test framework, it is planned to choose a suitable method for the specific category. Choose to check the client, web server or both at once; write api and ui tests to implement in individual or project or projects, choose the test data to use; define how to generate and transmit user tokens, which patterns to use; define whether they are needed. Implement parallelization for api tests or for ui as well; define how to check the models that come in the answers. This article outlines the methods that cover these issues and makes their brief analysis. The research of perspective methods and means of web application protection of the framework on the automatic assembly gradle system allows to state that correctly constructed test framework is one of effective and complex approaches to provide security information, which allows detection of vulnerabilities and correction of violations on the early stages of product development at the right time.

Keywords: api; ui; framework; tests; test data; automation; client; web server; gradle.

1. INTRODUCTION

Formulation of the problem. Nowadays, one of the most popular software development methodologies is Scram. Scrum is a development framework through which people can quickly solve emerging problems if productive producing of products if highly important. At the fulfilled process of Scrum testing has already a continuous cycle. Automation is often used to speed up the process, which significantly optimizes the verification of tasks, becaus, it is necessary to check (cover) the product as much as possible even for checking a single feature.

And this is quite a time-consuming procedure. If you look for weaknesses on each feature manually, then even a two-week iteration will be catastrophically not enough to include it in the current sprint, especially if you need to fix bugs (bug-fix).Sprint test results are one of the dominant values in application development. After the testing phase you can get quality product results. Before to testing the web application behaves like Schrödinger code. You do not know if everything works as intended. Tests before releasing or updating a product will help make sure that there are no vulnerabilities or defects in the code. And the sooner the potential and existing shortcomings of the application are identified, the lower error will cost. To find errors successfully and quickly, the automator first of all faces the task correct implementation of the test framework. Properly constructed test architecture ensures fast application coverage by tests, and also a quick search for its vulnerabilities. The correct presentation of the code provides easy understanding and further use by those who did not write it. It is important to search the vulnerability of the application from the beginning of its creation, so it provides a higher degree of protection and minimizes potential problems. Properly created test framework will ensure a quick search for incorrect implementation and easy understanding of the code by team members, and therefore fast writing, where the tests will be standardized for all.

Analysis of recent researches and publications. As in the books [1] UI Testing with Puppeteer: Implement end-to-end testing and browser automation using JavaScript and Node.js, (Dario Kondratiuk), [2] Automating and Testing a REST API: A Case Study in API testing using: Java, REST Assured, Postman, Tracks, cURL and HTTP Proxies (Alan J Richardson), this article discusses promising methods for protecting the framework's web application on a gradle automated build system. Namely, such issues are raised: choosing the client check, web server or both at once; what test data to use; how to generate and transmit tokens to users; implement parallelization or not; implement parallelization for api tests or for ui as well; how to check the models that come in the answers.

The purpose of the article. Nowadays there are many tools for protecting web applications with frameworks on the automatic build system gradle, which means that you can increase the level of security of the software at the stage of its development. Even during planning a framework test, the automator faces with the task of choosing methods that will solve the problem, will be flexible to change, easy to read and fast when searching for application vulnerabilities. The purpose of this article is to study the known methods of protecting web applications with frameworks on the automatic build system gradle, as well as to present their own solutions that will help to find client-server vulnerabilities at the stage of software development.

2. POTENTIAL VULNERABILITIES THAT MAY BE DETECTED BY THE TEST FRAMEWORK

To understand what vulnerabilities can be detected through the test framework, types of testing that can be implemented through automated tests should be considered. All types of work performed by the tester can be divided into two types.

First type, functional testing and non-functional testing. The main purpose of this type is to verify the implementation of the functional application requirements, i.e. the ability of the application to solve the tasks assigned to it in the given conditions. Requirements include: security, compliance with standards, ability to interact with other applications, functionality and clarity [6].

Second type, the main purpose of the second one is, first of all, to check for compliance with non-functional requirements: convenience (mainly user convenience assessment);

scalability (checks both vertical and horizontal scalability of the tested program) [7]; productivity (ability to run the program at different loads); security (user data protection, program data protection, burglary resistance); portability (compatibility and portability of the application for and under different environments, platforms, etc.); reliability (system behavior in various unexpected situations, the ability to handle non-standard user actions).

Also there are many different subtypes from these basic types. According to the objects that can be tested, there are localization and internationalization testing (detailed and in-depth verification of a certain application functionality, for example, only the network part, or only songs verification, or one page verification); interaction testing (how this application interacts with others or with the database); configuration testing (checking the accuracy of configuration files of both the application and the database it cooperates with);

Performance testing: Stability testing (how the application will behave during long-term operation); Stress testing (checking how the application will work under unexpected conditions, for example, if the power supply to the server is turned off, whether user data is lost or if incorrectly entered data is added database); Load testing (how the application will behave if, for example, a large number of users are logged in at the same time); Or usability testing (how user-friendly the application is); user interface testing (whether the user interface is correct in terms of UX design).

Security testing, search for vulnerabilities such as: XSS (Cross-Site Scripting), XSRF / CSRF (Request Forgery), Code injections (SQL, PHP, ASP, etc.), Server-Side Includes (SSI) Injection, Authorization Bypass [8].

3. MODERN METHODS OF FRAMEWORK WEB APPLICATION PROTECTION ON GRADLE AUTOMATIC ASSEMBLY SYSTEM

Modern computer systems are usually developed using a multi-layered architecture approach. [9] Covering the automation of the api component has the highest priority. The importance of the API is that it allows different organizations to create software applications that depend on other applications and services with no need to constantly update them during changing of the internal components of dependent applications or services. Ui automation is a bit more difficult to implement. It is often not automated and the manual is tested. In the case of automation coverage of the client part together with the tests that check the backend, it is a good practice to create them in a separate project or in another package. Best practice is to cover both parts, since it scales the application check and increases the search area for its vulnerabilities.

The test data is actually the input to the program. They represent data that affect or depend on the execution of a particular module. Some data can be used for positive testing, usually to verify that a given set of input data for a given function gives the expected result. Other data can be used for negative testing to test the program's ability to handle unusual, extreme, exceptional, or unexpected input. Poorly designed test data may not test all possible test scenarios. That will degrade the quality of the software.

There are some other approaches for creating test data:

- **Manual test data generatio:** In this approach, test data is entered by testers manually according to the requirements of the test case. This is a time consuming process and also prone to mistakes.
- **Automated test data generation:** It is done by using data generation tools. The main advantage of this approach is its speed and accuracy. However, this is more expensive than generating test data manually.

ISSN 2663 - 4023

- **Internal data entering:** This is done using SQL queries. This approach can also update existing data in the database. It is fast and efficient, but it should be used very carefully so that the existing database is not damaged.
- **Using the third-party tools:** There are public tools that understand your test scenarios at first and then generate or enter data accordingly to ensure broad testing coverage. These tools are accurate because they are set to business needs. But they are quite expensive.

A token is a unique sequence of data. The service must have a function for authorization, for example. In this case, the token will be the login and password, or what its content, this function should validate the user. The token can be generated using the GUID of the object or just MD5 from an inaccurate unique string. This token is returned during authorization and stored in the secrets' place in the database. Then this token is sent instead of passwords for login when accessing the service. And it is also checked if it contains the correct login or password or this token is available in the database. Accordingly, if there is a suspicious activity in a certain token, it can be removed from the database, i.e. "blocking" the token. So the session ends and the user remains active and can raise other sessions.

The standard solution for generating user tokens is to use third-party libraries. For example, the SoapUI Groovy script to generate a JWT token for ZAPI. The JWT consists of three parts: a header, payload and a signature. Methods that generate tokens are usually provided with a public access modifier. This solution appeared due to the need of using this method at the level of requests called in classes by the tests. Public access causes a vulnerability to the test framework. Tokens must be hidden and not available at the test level.

TestNG is a Java testing platform that helps organize tests in a structured way and improves the ease and convenience of reading scripts. Due to a wide range of features, TestNG has simplified the work of automation testers. One of them is parallel testing or parallel execution. TestNG provides an automatically defined XML file, where you can set a parallel method attribute/tests/classes, and with the concept of Java multithreading you can set the number of threads you want to create for parallel execution. There is the structure for defining this attribute in XML TestNG below [11]: <suite name = "Parallel_Testing" parallel = "methods" thread-count = "2" >.

The parallel attribute can be extended to several values, as shown below: **methods**: Helps to run methods in separate threads; **tests**: Helps run all methods which belong to one one tag in thread; **classes**: Helps to run all methods which belong to a one class in thread; **instances**: Helps to run all methods in one instance in one thread

Along with the parallel attribute, the count-count attribute helps to determine the number of threads you want to create for running tests at the same time. For example, if threads of one of the three methods are equal to two, then at runtime two threads start at the same time with the corresponding methods. While the execution of the first method is completed and the thread is released, it accepts the next method in the queue. The standard method of implemented parallelization in tests is written in the code snippet below. In one case, a chrome browser opens, in another – firefox. These methods are performed at the same time if the number of threads in the TestNG XML file is equal to two.

```
public class ParallelTestWithMultiThread {
    WebDriver driver;
    @Test()
    public void testOnChromeWithBrowserStackUrl()
    {System.setProperty("webdriver.chrome.driver", ".\\Driver\\chromedriver.exe");
```

ISSN 2663 - 4023

Initialization of the Chrome driver, setting the boot timeout, window scale and basic url to the test application. The maximize () method is used to fully deploy the window

```
        driver=new ChromeDriver();
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
        driver.get("https://www.browserstack.com/");
        driver.manage().window().maximize();
        System.out.println("this is the test related to chrome browserstack
homepage"+ " " +Thread.currentThread().getId());}
```

The next test is equivalent to the previous one, except for raising another browser

```
    @Test()
    public void testOnChromeWithBrowserStackSignUp()
    {System.setProperty("webdriver.gecko.driver", ".\\Driver\\geckodriver.exe");
        driver=new FirefoxDriver();
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
        driver.get("https://www.browserstack.com/users/sign_up");
        driver.manage().window().maximize();
```

The duplicate code, understanding how drivers open and work (the stage of the driver initialization is at the level of tests) is the problem of using these methods. This problem can be solved by dividing duplicate content in a separate method or in the base class. Then the class inheritance is implemented by the classes where the tests are located. But if the methods are static, after running the tests only one driver will rise. The driver will be reinitialized and as the result there are two failed tests. The first test will fail at the stage of raising the second driver [12]. OpenAPI is often used nowadays: specifications as a basis for acceptance tests, namely, building the test models; using setters and generating entire json, xml files.

Let's consider how the generation of the client works in more detail. OpenAPI is used more often nowadays. Opensource has two large quite popular projects. This is Swagger Codegen, which is currently supported by SmartBear. It has 11,000 stars on GitHub. And OpenAPI Generatoris also an open source project but it is supported by the community [13].

In fact, OpenAPI Generator is a Swagger Codegen Fork. It withdrew from this project in 2018 [14]. This happened because of the independent development of Swagger Codegen 3.X and Swagger Codegen 2.X. As a result, backward compatibility was violated. Many customers have gone and were not supported. And another reason is the instability of the release cycle. Releases in Swagger Codegen were quite rare, tests often failed. And that was unsatisfying for the community [17]. The use of third-party plugins and specifications has its disadvantages: it is not clear what is inside, whether it is suitable for the project and application; there is a redundant code that is not used and only overloads the project; there is a risk of incorrect use of ready solutions due to ignorance of the library specifics [15-16].

4. FURTHER DIRECTIONS FOR RESEARCH

Test data generation. A separate interface [17, 18] can be an alternative to standard methods and your own solution. In this interface can be implemented [19] two classes GenerateUIData and GenerateAPIData, in which the redefinition of the method getDate () is used. This solution can be used to find functional errors, check the mandatory input/transfer of requisites/all fields. The process of validation: checking the presence of the correct element with the correct name, which can be used as preconditions for other tests. This significantly clears the code from unnecessary repetitions and copypastes. The interface looks like this:

```
public interface DataFormat {String getDate (int minusFromCurrentDate);}
```

ISSN 2663 - 4023

This interface is implemented in the GenerateAPIData class. It overrides the data, which differ for two types of tests in the GenerateUIData class it is done in the same way:

```
@Override
public String getDate(int minusFromCurrentDate) {
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");
return LocalDate.now().minusYears(minusFromCurrentDate).format(formatter);}
```

The implemented interface redefines data that differ for two types of tests:

```
@Override
public String getDate(int minusFromCurrentDate)
{ DateTimeFormatter.ofPattern is responsible for bringing the data to the desired
type. api tests have the format yyyy-MM-dd, and ui MM / dd / yyyy
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("MM/dd/yyyy");
return LocalDate.now().minusYears(minusFromCurrentDate).format(formatter); }
```

At the level of api tests of data usage look like this:

```
SubrogationBase postInvalidSubrogationData =
SubrogationData.createSubrogationData(new GenerateAPIData());
```

At the ui level tests of data usage look like this:

```
SubrogationBase invalidSubrogationData =
SubrogationData.createInvalidSubrogationData(new GenerateUIData());
navigateToSubrogationPage()
    .editSubrogation(invalidSubrogationData)
```

As a result, it depends on the choice of new GenerateAPIData () or new GenerateUIData (), whether the generated data, which is used to test the client and web server, differ.

Token generation and transmission to users. Using standard methods of token generation and transmission to users may be an alternative way to use your own static generator. Token generation has a static access modifier, which provides one-time generation of tokens for all users after starting the build. Unlike standard methods this one significantly speeds up the execution of tests. This method can be used to look for potential vulnerabilities in the application, such as security errors, checking for peers, and user rights. All ui tests can be written and the search of the all vulnerabilities on the client side can be done on the basis of the own decision using a pool of drivers for implementation of parallelization in ui tests. Implementation of parallelization for api tests is fairly straightforward. To do this,let's add parallel = "tests" thread-count = "4" to the xml file:

```
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >
<suite name="All api test suite" data-provider-thread-count="4" parallel="tests"
thread-count="4">
```

The implementation of parallelization for ui tests is more complicated. Let's consider own representation of the solution of parallelization for ui tests.Calling of one driver instance is used to implement it in the WebDriverFactory class. Using static methods to open the driver complicates the task because it means that only one driver can be raised. A driver pool is initialized in the next way (ThreadLocal <WebDriver> poolOfDrivers = new ThreadLocal <> ()) to raise several drivers. The default constructor is executed immediately after calling this class. There is a separate instance in the class for raising the driver and the getter that receives drivers from the object pool, and the setter, which enters the drivers into the object pool.

ISSN 2663 - 4023

```
public class WebDriverFactory extends AbstractWebDriver {
    private WebDriverFactory() {}
    private static WebDriverFactory instance = new WebDriverFactory();
    public static WebDriverFactory getInstance() {
        return instance; }
    ThreadLocal<WebDriver> poolOfDrivers = new ThreadLocal<>();
    public void setDriver(WebDriver driverData) {
        poolOfDrivers.set(driverData);}
```

The driverSet (String browser) method is responsible for raising a new driver and adding it to the driver pool. Implemented parallelization guarantees doubling of the ui tests speed. It is known that the execution time of these tests is much longer compared to api tests. If the first ones last for 40 minutes, the second ones – 15-20 minutes. The ability to choose a browser almost doubles the testing scope of the application.

Serialization, deserialization. For the process of serialization and deserialization, own models, which generated by using Plain Old Java Object (POJO) files [28], should be used . This solution can be used to search for security errors, check the models obtained in the requests, in functional tests. This allows to skip the transmission of unnecessary variables that can have a negative impact on the application. In addition, it also checks whether no system variables are visible to the user in the requests, minimizes the possibility of penetration and influence of third-party users on the application. POJO class is a Java object that is not bound by any restrictions other than those ones caused by the Java language specification. POJO is characterized by: extension of the specified classes; implementation of the specified interface; contains installed annotations. Compared to other standard solutions, this solution is easy to operate and implement. And the run time of the test is the same as in other methods.

Cloud testing. Due to the need of having access to multiple platforms and browsers for running parallel tests, the cost of testing compatibility with parallel testing increases. In addition, access to all browsers and versions may not be possible. BrowserStack provides access to numerous platforms and browsers with relative versions in the cloud. Also running of automatic parallel tests can be used and using multiple browsers and versions.

Artificial Intelligencem Another further area of testing is the use of artificial intelligence for testing. Nowadays there some programs: Applitools. Helps to find errors inside the user interface quickly. Applying some machine learning developments, testers can easily find inaccuracies in the interface. The application allows to adjust the format of the tests to the required display forms (adaptive view) quickly. If a product uses animation, the utility to find it could be created [20-23]. Sauce Labs. One of the first programs that allowed launching the tests in the cloud. The service runs up to 1 million automated tests every day. Based on the machine learning researching, the Sauce labs developers are working to create a powerful tool for analyzing the quality of applications. Test.AI. Helps to combine AI and Selenium. The tests are created in a simple format that is visually very similar to Gherkin. There is no need to write anything and understand all of the locators. The parameters dynamically define the window of utility and elements, as well as automatically start the work to check the functionality of the application. Also it is easy to record all the verification of the created tools. Mabl. Allows to run functional tests on the application. Created tests can be easily self-taught. There are tools for combating flaky tests. There are parameters for checking the dynamic change of elements and shapes. The history of runs can be also checked in the logs to find changes in the utility. ReTest. Utility for functional testers. The application allows you to create tests even without programming skills. There are test recording functions, ACCERT standardization and areas of

application that require special attention. But these products are not frequently used due to common mistakes during operation.

5. CONCLUSIONS

Summing up the results, it can be concluded that an important place in the search for application vulnerabilities is occupied by a properly constructed test framework. It ensures fast coverage of the application by tests and quick searching of its vulnerabilities. The correct code presentation provides easy understanding and further use by those who did not write it. To ensure the highest level of security of the developed application, it is necessary to focus on security methods that will make the code understandable and pure: project structure; selection of test data; implementation of a pool of drivers for ui tests parallelization; the process of generating tokens that are transmitted only to a certain level; pattern builder used in test data; POJO files for ensuring serialization, deserialization.

REFERENCES

1 Kondratiuk, D. (2021). *UI Testing with Puppeteer: Implement end-to-end testing and browser automation using JavaScript and Node*.js.
2 Richardson, A. (2017). *Automating and Testing a REST API: A Case Study in API testing using: Java, REST Assured, Postman, Tracks, cURL and HTTP Proxies*.
3 Postman for API Testing — Pros, Cons, and Alternative Solutions. World Wide Web. https://dzone.com/articles/postman-for-apitesting-pros-cons-and-alternative.
4 REST API. World Wide Web. https://www.loadview-testing.com/ru/blog/.
5 Katalon Studio. World Wide Web. https://artoftesting.com/katalonstudio-features-advantages-and-disadvantages.
6 What is API Testing? World Wide Web. https://www.inflectra.com/rapise/highlights/api-testing.aspx.
7 Front-end and Back-end outlet. World Wide Web. https://skillbox.ru/media/code/frontend_i_backend_razrabotka/
8 API Testing: Why It Matters, and How to Do It. World Wide Web. https://blog.udemy.com/api-testing/.
9 SOAP API. World Wide Web. https://quality-lab.ru/blog/soap-api-testing/.
10 Satya, A. (2014). *Selenium Webdriver Practical Guide*. Paperback: Packt Publishing.
11 Testing software security. Basic understanding and value. World Wide Web. http://www.protesting.ru/testing/. (in Russian).
12 Gregory, J.; Crispin, L. (2019). Agile Testing: A Training Course for the Whole Team. "Mann, Ivanov and Ferber". (in Russian).
13 Software Testing. World Wide Web. https://www.tutorialspoint.com/software_testing/software_testing_lev els.htm.
14 Software Testing - API testing. World Wide Web. http://www.tutorialspoint.com/software testingdictionary/apitesting.htm.
15 The Value of Mixing UI and API Testing. World Wide. https://www.webomates.com/blog/api-testing/the-value-of-mixing-ui-and-api-testing/.
16 Java For Testers: Learn Java fundamentals fast 2009. World Wide Web. http://tctutorial.ru/frameworks.
17 Patton, R. (2006). *Software testing*. Pearson Education India.
18 Vinnichenko, I. (2005). *Automation of testing processes*. Publishing house " (in Russian).
19 Badgett, T., Myers, G. J., & Sandler, C. (2011). *Art of Software Testing*. Wiley & Sons, Incorporated, John.
20 Savin, R. (2007). *Testing Dot Com*. Delo.(in Russian).
21 Ilyenko, A., Ilyenko, S., & Stashevskyi, D. (2021). Software error tracking module in web applications based on the use of logger algorithm. *Cybersecurity: Education, Science, Technique*, *3*(11), 61–72. https://doi.org/10.28925/2663-4023.2021.11.6172
22 Ilyenko, A., Ilyenko, S., & Vertypolokh, O. (2020). METHOD FOR PROTECTION TRAFFIC FROM INTERVENTION OF DPI SYSTEMS. *Cybersecurity: Education, Science, Technique*, *2*(10), 75–87. https://doi.org/10.28925/2663-4023.2020.10.7587.

23 Ilyenko, A., Ilyenko, S., & Kulish, T. (2020). PROSPECTIVE PROTECTION METHODS OF WINDOWS OPERATION SYSTEM. *Cybersecurity: Education, Science, Technique*, *4*(8), 124–134. https://doi.org/10.28925/2663-4023.2020.8.124134.

КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 2 (14), 2021
ISSN 2663 - 4023

Ільєнко Анна Вадимівна
к.т.н., доцент, доцент кафедри комп'ютеризованих систем захисту інформації
Національний авіаційний університет університет, факультет кібербезпеки комп'ютерної та програмної інженерії, Київ, Україна
ORCID ID: 0000-0001-8565-1117
ilyenko.a.v@nau.edu.ua

Ільєнко Сергій Сергійович
к.т.н., доцент, доцент кафедри автоматизації та енергоменеджменту
Національний авіаційний університет університет, аерокосмічний факультет, Київ, Україна
ORCID ID: 0000-0002-0437-0995
ilyenko.s.s@nau.edu.ua

Куліш Тетяна Миколаївна
бакалавр, студентка кафедри комп'ютеризованих систем захисту інформації
Національний авіаційний університет університет, факультет кібербезпеки комп'ютерної та програмної інженерії, Київ, Україна
ORCID ID: 0000-0001-8413-9154
teti98kulish@gmail.com

ПЕРСПЕКТИВНІ МЕТОДИ ЗАХИСТУ ВЕБЗАСТОСУНКУ ФРЕЙМВОРКУ НА СИСТЕМІ АВТОМАТИЧНОЇ ЗБІРКИ GRADLE

Анотація. Стаття призначена розгляду проблеми забезпечення захисту вебзастосунку фреймворку на системі автоматичної збірки gradle та визначення перспективних методів забезпечення захисту. В даній статті визначено, що базовим і загальноприйнятим підходом що забезпечує безпеку вебзастосунку виступає правильно побудований тестовий фреймворк. В процесі дослідження зроблено аналіз сучасних методів захисту вебзастосунку фреймворку на системі автоматичної збірки gradle. Зроблено порівняльний аналіз методів, що входять у одну підкатегорію. На підставі проведеного аналізу сучасний фреймворків визначені основні підходи і методи щодо організації захисту застосунку. Ще при плануванні тестового фреймворку перед автоматизатором ставляться задача вибрати методи, що будуть вирішувати поставлені задачі, будуть гнучкими для змін, легкими для читання і швидкими при пошуку вразливостей застосунку. На етапі розробки тестового фреймворку планується вибір методу, що закриватиме свою категорію. Вибирати перевірку клієнта, вебсервера чи обох відразу; написання арі і ui тестів реалізовувати у окремих проектах чи одному, які тестові дані використовувати; як генерувати і передавати токени користувача;, які патерни використовувати, чи є у них потреба, реалізовувати паралелізацію чи ні. Реалізовувати паралелізацію для арі тестів чи для ui також; як перевіряти моделі, що приходять у респонсах. У даній статті наведено методи, що закриватимуть ці питання і зроблено їх короткий аналіз. Проведене в статті дослідження перспективних методів та засобів захисту вебзастосунку фреймворку на системі автоматичної збірки gradle дозволяє стверджувати, що правильно побудований тестовий фреймворк, є одним з дієвих та комплексних підходів щодо забезпечення інформації безпеки, що дозволить своєчасно виявляти вразливості та своєчасно виправити порушення ще на початковому етапі розробки продукту, тим самим зменшивши ціну помилки.

Ключові слова: api; ui; фреймворк; тести; тестові дані; автоматизація; клієнт; вебсервер; gradle.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Kondratiuk, D. (2021). *UI Testing with Puppeteer: Implement end-to-end testing and browser automation using JavaScript and Node*.js.
2 Richardson, A. (2017). *Automating and Testing a REST API: A Case Study in API testing using: Java, REST Assured, Postman, Tracks, cURL and HTTP Proxies*.
3 Postman for API Testing — Pros, Cons, and Alternative Solutions. World Wide Web. https://dzone.com/articles/postman-for-apitesting-pros-cons-and-alternative.

4 REST API. World Wide Web. https://www.loadview-testing.com/ru/blog/.
5 Katalon Studio. World Wide Web. https://artoftesting.com/katalonstudio-features-advantages-and-disadvantages.
6 What is API Testing? World Wide Web. https://www.inflectra.com/rapise/highlights/api-testing.aspx.
7 Front-end and Back-end outlet. World Wide Web. https://skillbox.ru/media/code/frontend_i_backend_razrabotka/
8 API Testing: Why It Matters, and How to Do It. World Wide Web. https://blog.udemy.com/api-testing/.
9 SOAP API. World Wide Web. https://quality-lab.ru/blog/soap-api-testing/.
10 Satya, A. (2014). *Selenium Webdriver Practical Guide*. Paperback: Packt Publishing.
11 Testing software security. Basic understanding and value. World Wide Web. http://www.protesting.ru/testing/. (in Russian).
12 Gregory, J.; Crispin, L. (2019). Agile Testing: A Training Course for the Whole Team. "Mann, Ivanov and Ferber". (in Russian).
13 Software Testing. World Wide Web. https://www.tutorialspoint.com/software_testing/software_testing_lev els.htm.
14 Software Testing - API testing. World Wide Web. http://www.tutorialspoint.com/software testingdictionary/apitesting.htm.
15 The Value of Mixing UI and API Testing. World Wide. https://www.webomates.com/blog/api-testing/the-value-of-mixing-ui-and-api-testing/.
16 Java For Testers: Learn Java fundamentals fast 2009. World Wide Web. http://tctutorial.ru/frameworks.
17 Patton, R. (2006). *Software testing*. Pearson Education India.
18 Vinnichenko, I. (2005). *Automation of testing processes*. Publishing house " (in Russian).
19 Badgett, T., Myers, G. J., & Sandler, C. (2011). *Art of Software Testing*. Wiley & Sons, Incorporated, John.
20 Savin, R. (2007). *Testing Dot Com*. Delo.(in Russian).
21 Ilyenko, A., Ilyenko, S., & Stashevskyi, D. (2021). Software error tracking module in web applications based on the use of logger algorithm. *Cybersecurity: Education, Science, Technique*, *3*(11), 61–72. https://doi.org/10.28925/2663-4023.2021.11.6172
22 Ilyenko, A., Ilyenko, S., & Vertypolokh, O. (2020). METHOD FOR PROTECTION TRAFFIC FROM INTERVENTION OF DPI SYSTEMS. *Cybersecurity: Education, Science, Technique*, *2*(10), 75–87. https://doi.org/10.28925/2663-4023.2020.10.7587.
23 Ilyenko, A., Ilyenko, S., & Kulish, T. (2020). PROSPECTIVE PROTECTION METHODS OF WINDOWS OPERATION SYSTEM. *Cybersecurity: Education, Science, Technique*, *4*(8), 124–134. https://doi.org/10.28925/2663-4023.2020.8.124134