Automated Evaluation of Digital TV Middleware: Using Fuzzing and Pairwise for Test Generation

Sergillam B. Oliveira[†‡], Eddie B. de Lima Filho[†‡], Orlewilson B. Maia[†], and Lucas C. Cordeiro[‡*]
[†]*TPV Technology, Manaus, Brazil*
[‡]*Federal University of Amazonas, Manaus, Brazil*
[*]*University of Manchester, Manchester, UK*
{sergillam.oliveira, eddie.filho, orlewilson.maia}@tpv-tech.com
lucas.cordeiro@manchester.ac.uk

***Abstract*—Testing is paramount in evaluating DTV software quality while developing digital TV (DTV) receivers. It is about executing tests and looking for the most effective and efficient approaches, i.e., the ones that present high coverages in short running times. However, in addition to the system software modules usually present in DTV environments, there is also the middleware element, which surely involves application programming interfaces and specific execution environments, leading to a thorough conformance evaluation. Moreover, as a complex entity, DTV middleware may require broader verification, even with scenarios going beyond what is exposed in standards. The present work addresses these aspects and introduces an automated test generation methodology using fuzzing and pairwise testing tailored for the middleware DTV Play used in the Brazilian DTV system. It speeds up testing processes, enabling quicker cycles and more thorough evaluation steps.**

***Index Terms*—Representational State Transfer, Application Programming Interface, Software Testing, Fuzzing, Pairwise, Digital TV.**

I. Introduction

Software testing is essential to verify and validate modules, thus ensuring they function as intended [1]–[4]. Beyond functional verification, testing plays a key role in identifying potential vulnerabilities and improving the security and performance of software modules [1].

In the context of digital TV (DTV) receivers, interactive middleware modules must undergo rigorous evaluation. A compromised middleware module could expose sensitive user information, including financial data [1], [4]. Additionally, such modules must conform to specific industry standards to ensure consistent functionality across different devices and environments [1].

It is common practice to address these requirements using test suites and verification tools during checking processes [5], [6]. When properly configured and organized, such resources can provide a structured approach to testing, leading to a comprehensive evaluation of a module's functionality, reliability, and robustness [5], [7]. Besides, test suites can be manually or automatically created, even employing fuzzing techniques specifically crafted to find edge cases and security vulnerabilities [1], [5].

For DTV middleware, one significant risk is the infiltration of malicious code through applications or application programming interface (API) requests. This is particularly concerning for the middleware used in the Brazilian digital television system (SBTVD), known as DTV Play, which includes a comprehensive representational state transfer (REST) API for peripheral devices, such as smartphones [8]. Therefore, automated and in-depth testing of such interfaces is critical to ensure robustness.

However, generating automated tests for REST APIs, particularly using fuzzing, presents a unique challenge. Fuzzers' effectiveness is often measured by how many APIs they can activate or how much code coverage they achieve within a specific timeframe or budget. To generate valid requests that explore a target API thoroughly, a fuzzer must be capable of automatically providing valid parameters by resolving API dependencies.

This study builds upon the work of Oliveira *et al.* [9], addressing the challenges outlined above by introducing a methodology for automated test generation specifically targeting the DTV Play's REST API. It uses fuzzing techniques to generate test scenarios that traditional approaches, such as official test suites, may overlook. This enhanced methodology aims to improve the robustness and reliability of DTV Play, ultimately contributing to the overall performance of DTV environments.

The methodology improves robustness and reliability by utilizing smart fuzzing to generate a wide variety of unpredictable and edge-case test inputs that are often missed by traditional testing approaches. Fuzzing systematically explores the input space of the API, identifying potential vulnerabilities, crashes, or unexpected behaviors that may occur under real-world conditions. By simulating diverse and anomalous usage patterns, the method uncovers hidden issues that might affect the stability and security of the DTV Play system. This automated, thorough exploration enhances the overall performance of DTV environments by ensuring they are resilient to a broader range of inputs and operational scenarios.

The proposed tests were applied to the official DTV Play test suite, created by an SBTVD's special group. They focused on the REST API provided by its common core web services (CCWS) submodule. The results suggest that these new tests effectively complement the existing test suite by identifying unaddressed scenarios that can reveal hidden errors.

The methodology developed in this study verifies and expands a suite's current test coverage through an exploratory approach. Identifying new error conditions and weaknesses during the experiments demonstrates its potential to significantly enhance the quality and security of DTV middleware modules.

This paper is organized as follows. Section II presents related studies, which help define the associated scientific basis. Then, Section III provides the basic knowledge employed in our work. Next, Section IV introduces the proposed architecture. After that, Section V presents an implementation of it, while Section VI exposes our experiments and discusses application aspects, with the goal of proposal validation. Finally, the associated conclusions are shown in Section VII.

II. Related Studies

The work developed by Silva *et al.* [2] presents an automated testing architecture that provides conformance and robustness test scenarios for multimedia processing infrastructures employed in DTV receivers. It highlights the challenges posed by consumer electronics devices, which lack optimized and flexible infrastructures to support multimedia services. Their methodology aims to verify the current test coverage provided by a test suite and expand it through an exploratory study. Experiments with a real implementation, focusing on DTV receivers conforming to the SBTVD standards, demonstrated the methodology's effectiveness by revealing robustness and conformance issues in an already mature and tested multimedia stack. Additionally, based on fuzzing techniques, the test generation engine successfully created an uncommon scenario that was not detected by popular tools, underscoring the proposed solution's importance.

In connection with this, Maia *et al.* [5] address the recent enhancements to the open standards related to middleware DTV Play adopted in Brazil, which now incorporate integrated broadcast-broadband concepts. These advancements allow for greater cooperation within local DTV networks and access to broadband content. Indeed, traditional testing of interactive applications may now prove inadequate due to these new aspects. Moreover, manually validating network scenarios involving CCWS, which relies on the hypertext transfer protocol (HTTP), can be time-consuming and error-prone. To address this issue, they propose a real-time analysis methodology for automatic testing of CCWS, leading to test procedures that are, on average, 66.84% faster than manual testing. They also highlight the efficiency of this automated approach in handling the expanded functionality of DTV systems.

Expanding on this theme, Júnior *et al.* [3] explores an innovative methodology focused on DTV Play, using flowcharts and user interfaces to detail its test. Validation through automated test equipment (ATE) reports error rates and quality metrics during implementations, delving into code integration, testing methods, and conclusions drawn from evaluating DTV devices with DTV Play. This paper concludes by reinforcing the impact of automated testing methodologies, emphasizing their role in improving the reliability and quality of DTV middleware systems. As such, this research provides a solid basis for further exploring automated testing frameworks in DTV middleware, contributing to ongoing advancements in this field.

Overall, the studies mentioned here collectively contribute to the growing body of knowledge on automated testing methodologies for DTV systems, highlighting both the challenges and the potential of automated solutions to improve conformance, robustness, and efficiency in this critical area of digital broadcasting.

III. Background

The next sections present the basic knowledge used in our work. Specifically, we tackle REST APIs, fuzzing, and pairwise.

A. REST API

REST is a design approach for network-based software typically implemented via HTTP [10]. However, it is not strictly tied to any specific protocol [11], and an API designed using it is referred to as a REST API.

When employed for communication between clients and servers, a REST API should offer a format that allows valid requests and guidelines for interpreting their responses [11]. Requests are composed by the following elements:

- $endpoint$, which indicates the uniform resource locator (URL) of an API and may consist of several tokens or arguments separated by "/", e.g., */token/:arg* and */token/arg*;
- $method$, which regards HTTP methods, i.e., POST for creating, GET for retrieving, PUT for updating, PATCH for partially updating, and DELETE for removing data;
- $parameter$, which is accepted as key-value pairs and can be either required or optional.
- $return\ response$, which is a status code indicating the outcome of clients' requests to a server.

B. Fuzzing

Fuzzing is an automated software testing method that generates random inputs to induce errors in target systems [12]. It aims to detect program anomalies, logical errors, or design flaws, using data that seem correct but may implicitly carry non-conformances or wrong elements.

Fuzzers can also be smart and possess some knowledge of the target entity, based on usual implementations, standards, or even code descriptions. Besides, they usually employ special strategies, such as grammars, to enhance effectiveness and code coverage.

C. Pairwise Testing

Pairwise testing is a technique that employs a combinatorial approach, where every possible unique pair of input parameters is tested for all conceivable discrete combinations [13].

Besides, it is widely known that bugs usually result from a single parameter or interaction among parameters [13], the latter being rare. Consequently, it offers a trade-off without significantly sacrificing coverage [14].

IV. The Proposed Methodology

The proposed methodology aims to generate tests in an automated way, as illustrated in Fig. 1. It is based on a process structured into four main steps: (1) input parameter definition, (2) test case generation, (3) test case execution, and (4) result evaluation.

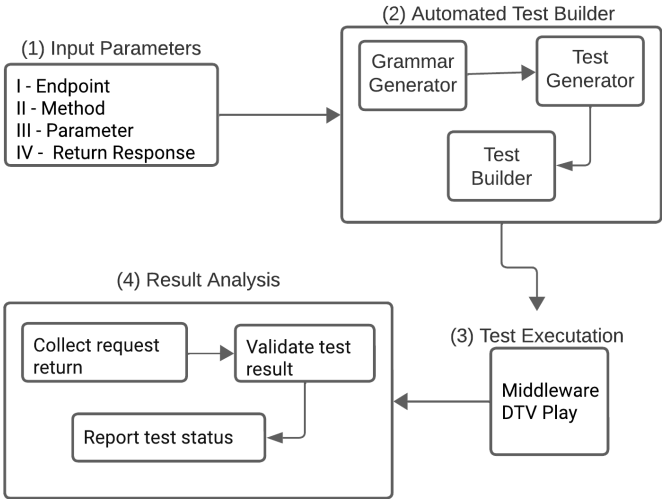

Fig. 1. A detailed overview of the proposed methodology.

The first step involves an initial mapping of all information that can be changed, including data domain, dependencies between variables, specific behaviors of values, and business requirements. The pairwise technique identifies the number of combinations for each parameter, ensuring the generation of relevant and valid tests during the fuzzing phase.

In the second step, once all parameters have been properly mapped, the test construction process begins. Firstly, in the *Grammar Generator* stage, the grammatical and logical rules that guide test creation are defined, including variable interaction and restrictions. Next, *Test Generator* applies pairwise and fuzzing, combining the inputs of the previously defined variables according to the existing rules. Then, *Test Builder* sets up and validates tests, ensuring that the pre-established quality and functionality standards are met.

The third step tackles test execution. It checks the specific route to be evaluated and the set of previously generated test cases, following the basic approach presented by Oliveira *et al.* [9]. Moreover, it executes the generated tests, interacting with a REST API according to the specified inputs. This procedure not only verifies whether the implementation respects the established rules and conditions but also provides an objective evaluation of a system's performance and compliance.

The fourth step gathers essential data on how a REST API responds to inputs. During its analysis, it examines whether the responses are technically correct and whether they comply with the defined functional and non-functional requirements. This includes checking for accurate data and appropriate behavior in different scenarios and observing possible deviations or anomalies. A deep understanding of these results is crucial to identify and correct any faults or irregularities that may compromise the performance or integrity of a REST API.

V. An Implementation of the Proposed Architecture

The proposed methodology was evaluated by implementing a tool based on it, performed on a personal computer running the operating system Ubuntu 22.04. The programming language Python [15] was employed, along with the framework Flask[1] and the library *pywebview* [2], which resulted in the initial interface responsible for extracting information from the target API, as illustrated in Fig. 2.

Fig. 2. The software interface for registering API parameters.

When collecting data, the tool's initial step detects the HTTP method used in a request. Then, it identifies parameters in the associated URL and classifies them based on their data. Finally, fuzzing is employed to create variants.

Next, it concentrates on filling in the parameters sent in the request body. They are presented as variable names along with their corresponding data types.

Fig. 3 illustrates another part of the URL data records, providing a detailed view of the process used to log both success messages and error codes returned by a route. The associated parameters and their respective values are systematically presented alongside the returned messages for successful responses. This enables a clear understanding of how the data flows during successful transactions. In contrast, error codes are carefully cataloged when errors occur, offering detailed insight into potential failures that require further investigation. This comprehensive logging mechanism captures all possible outcomes, from successful executions to errors, providing a holistic view of a system's behavior.

Such an approach not only aids in identifying and troubleshooting issues but also facilitates performance analysis and system optimization by offering precise data for future improvements. This level of detailed logging is crucial for ensuring a system's robustness and reliability, particularly in

[1]https://flask.palletsprojects.com/en/3.0.x/
[2]https://pywebview.flowrl.com/

complex environments where multiple parameters and error conditions must be monitored.

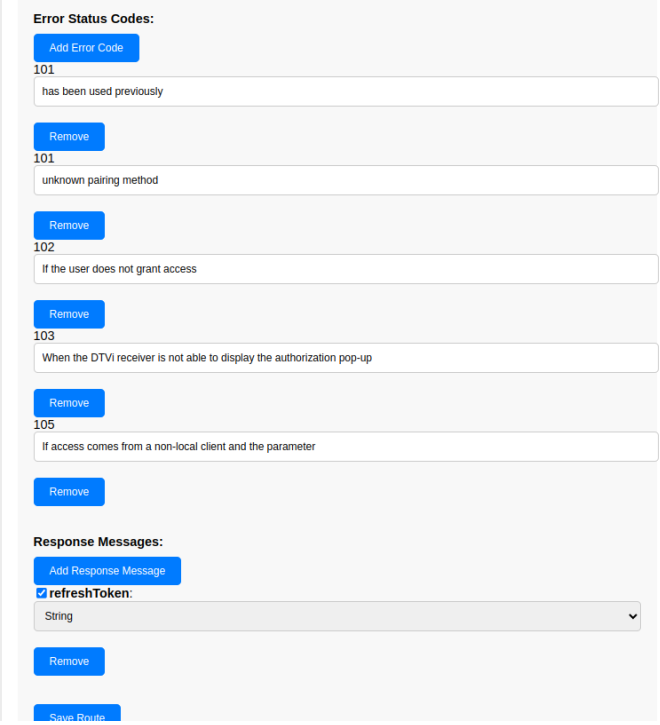

Fig. 3. The software interface for logging the API's return messages.

VI. Simple Experiments and Some Discussion

The methodology proposed in this work introduces a novel approach to automated API testing using smart fuzzing techniques. Unlike traditional fuzzing, which randomly generates inputs to test API robustness, smart fuzzing focuses on intelligently creating input variations based on the data types and parameter structures extracted from API endpoints. This ensures that the generated tests are more targeted and relevant, leading to improved coverage and the discovery of otherwise overlooked vulnerabilities.

The developed tool was used to evaluate the client authorization/authentication and broadcaster security APIs: CCWS's route groups 8.1.1, 8.1.2, 8.7.1, 8.7.2, and 8.7.3 [8], whose results are detailed in Table I. Our tool surpassed the official suite, producing 94 additional tests, with its smart fuzzing method that tackles missed data, thus enhancing coverage. This was achieved by exploiting parameter variants, which resulted in a more comprehensive evaluation. Moreover, 3 new weaknesses were found, which would otherwise go unnoticed.

Indeed, these tests tackled missed data from the official test suite and improved coverage by exploring edge cases and parameter variations. Each new test case was constructed based on the parameter types identified during API analysis, ensuring meaningful and valid inputs.

The key innovation of our methodology lies in its modular nature, particularly in Step 3, which automates the creation of test cases. As illustrated in Fig. 1, this step can be adapted to test any REST API, provided the necessary data is supplied in Step 1. This enables the generation of a customized grammar for the target API, ensuring that the automated test builder in Step 2, also shown in Fig. 1, produces valid and context-aware test cases.

TABLE I
The performance of the implemented tool, including original, new, and failed new tests.

Route	Original tests	New tests	Failed new tests
8.1.1	21	72	0
8.1.2	11	27	0
8.7.1	6	15	1
8.7.2	6	15	1
8.7.3	6	15	1
Total	50	144	3

This adaptability demonstrates the tool's broader applicability, enabling it to test other APIs while maintaining its robust capabilities in finding vulnerabilities through smart fuzzing.

VII. Conclusion

This study proposes a methodology for test creation using fuzzing and pairwise testing to evaluate middleware modules conforming to the DTV Play's specifications, focusing on identification, application authorization, and broadcaster security features. Such tests were created following the respective standard [8]. The efficiency and effectiveness of this approach were demonstrated as it was able to automatically create 94 new tests that resulted in 3 new violations without the need for human intervention.

This outcome is considered positive, as the methodology not only efficiently generated a significant number of new tests but also identified new violations, proving its effectiveness in detecting issues that might not have been discovered through manual testing. This improves test coverage while reducing time and effort, making the approach both cost-effective and scalable. Therefore, the proposed methodology adds significant value to the testing process, highlighting its potential for further use and optimization.

For future work, additional features will be added to expand the scope of our methodology, incorporating other scenarios involving APIs for accessing the DTV context. Additionally, we are exploring ways to integrate our methodology with existing platforms and the home environment, creating a more cohesive and effective system. Such enhancements can significantly increase the usefulness and applicability of our automated testing approach.

Acknowledgement

The results presented in this paper were sponsored by ENVISION Indústria de Produtos Eletrônicos LTDA, under the terms of Brazilian federal law No. 8.387/91 (SUFRAMA). This research was conducted by Envision (TPV Group).

REFERENCES

[1] F. Izumi, E. B. de Lima Filho, L. C. Cordeiro, O. Maia, R. Fabrício, B. Farias, and A. Silva, "A fuzzing-based test-creation approach for evaluating digital tv receivers via transport streams," *Software: Testing, Verification and Reliability*, vol. 33, pp. 1–31, October 2022.

[2] A. K. Da Silva, L. C. Amorim, O. B. Maia, E. B. De Lima Filho, T. C. Da Rocha, G. C. Dos Santos, A. L. Guedes, A. R. Queiroz, and K. R. De Araújo, "An automated testing methodology to evaluate multimedia processing stacks running on digital tv receivers," in *2024 IEEE International Conference on Consumer Electronics (ICCE)*, pp. 1–6, IEEE, 2024.

[3] M. J. S. Júnior, O. B. Maia, B. Eddie Filho, R. Fabrício, and A. Silva, "An automated testing methodology for digital tv middleware implementations," in *2019 IEEE 9th International Conference on Consumer Electronics (ICCE-Berlin)*, pp. 400–403, IEEE, 2019.

[4] F. Izumi, B. Farias, E. de Lima Filho, A. Amorim, O. B. Maia, and A. Silva, "Evaluation of digital tv receivers with noncompliant mpeg-2 transport streams," in *2019 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-TW)*, pp. 1–2, 2019.

[5] O. B. Maia, A. R. da Silva Conceição, M. J. de Souza Júnior, F. Izumi, E. B. de Lima Filho, and P. Corrêa, "A real-time analyzer for testing dtv play," in *2022 IEEE International Conference on Consumer Electronics (ICCE)*, pp. 01–05, IEEE, 2022.

[6] F. R. Monteiro, M. R. Gadelha, and L. C. Cordeiro, "Model checking C++ programs," *Softw. Test. Verification Reliab.*, vol. 32, no. 1, 2022.

[7] O. B. Maia, H. C. Yehia, and L. de Errico, "A concise review of the quality of experience assessment for video streaming," *Computer communications*, vol. 57, pp. 1–12, 2015.

[8] Brazilian Association of Technical Standards, *ABNT NBR 15606-11, Digital terrestrial television – Data coding and transmission specification for digital broadcasting – Part 11: Ginga CC WebServices – Ginga Common Core WebServices specification*, 2021.

[9] S. B. Oliveira, A. R. da Silva Conceiçao, E. B. de Lima Filho, and L. C. Cordeiro, "Evaluation of ginga's cc-web-service module," in *IEEE International Conference on Consumer Electronics-Taiwan*, 2023.

[10] A. Giretti, "Introducing http and rest," in *Coding Clean, Reliable, and Safe REST APIs with ASP. NET Core 8: Develop Robust Minimal APIs with. NET 8*, pp. 1–41, Springer, 2023.

[11] swagger, "Openapi specification." urlhttps://swagger.io/specification/, Feb. 2021.

[12] H. L. Nguyen, N. Nassar, T. Kehrer, and L. Grunske, "Mofuzz: A fuzzer suite for testing model-driven software engineering tools," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, ASE '20, (New York, NY, USA), p. 1103–1115, Association for Computing Machinery, 2021.

[13] R. Kuhn, Y. Lei, and R. Kacker, "Practical combinatorial testing: Beyond pairwise," *It Professional*, vol. 10, no. 3, pp. 19–23, 2008.

[14] X. Chen, Q. Gu, J. Qi, and D. Chen, "Applying particle swarm optimization to pairwise testing," in *2010 IEEE 34th Annual Computer Software and Applications Conference*, pp. 107–116, IEEE, 2010.

[15] M. Lutz, *Programming python*. " O'Reilly Media, Inc.", 2001.