Evaluation of Ginga's CC-Web-Service Module

Sergillam Barroso Oliveira
Andre Ricardo da Silva Conceição
Eddie Batista de Lima Filho
TPV Technology, Manaus, Brazil
{sergillam.oliveira, andre.conceicao, eddie.filho}@tpv-tech.com

Lucas C. Cordeiro
University of Manchester, UK
Federal University of Amazonas, Brazil
lucas.cordeiro@manchester.ac.uk

***Abstract*—Software testing has become a key factor in software development processes in the last decades. Ideally, it should check programming errors, assure conformance, and prevent weaknesses that attacks can exploit. However, at the same time, the ability to increase the coverage of testing approaches regarding expected features or source code also turned out to be quite a challenge. The present work considers tests applied to evaluate interactive middleware modules for digital TV receivers. To enhance their resulting quality assessment, we have developed a methodology that verifies the current test coverage provided by a test suite and executes an exploratory study to expand it. Finally, we have conducted experiments considering test suites for commercial versions of the middleware Ginga-D used in Brazil. As a result, new error conditions and weaknesses were identified, which can effectively improve middleware modules.**

***Index Terms*—Software Testing; Fuzzing; Security; Digital TV.**

I. Introduction

Software testing is the verification process that checks if a program behaves as expected, including overall correctness, conformance, and absence of weaknesses. In addition, in academic research and industry, it is usual to use test suites and verification tools for that process [1]. Specifically, the former are usually evaluated according to a coverage criterion, aiming to ensure their quality and comprehensiveness [2]–[4].

As a practical example of software evaluated with test suites, we can cite interactive middleware modules for digital TV (DTV) receivers [1]. They are often based on open standards and must conform to them, given that any manufacturer can develop its version. For instance, DTV middleware may be attacked by malicious code in applications loaded from external devices (e.g., memory drives) or requests made to application programming interfaces (APIs). Indeed, the latter can happen to the middleware used in the Brazilian digital television system (SBTVD), namely Ginga-D, which provides a representation state transfer (REST) API [5].

Test teams may receive testing resources for middleware modules if a certification must be acquired. They may even develop their test suites if no entity is responsible for conformance or correctness [1]. However, one may argue that such tools may not suitably tackle specific scenarios and unforeseen conditions on user premises. Certification tests only assure conformance, while proprietary ones usually check basic behaviors. However, both rarely stress technologies or consider robustness and uncommon conditions.

This work addresses the problem raised in the last paragraphs and proposes a methodology to enhance test suites for DTV middleware. The main goal is to provide conditions and scenarios not tackled by original testing resources. In addition, this study also provides:

- an evaluation regarding the effectiveness of the proposed methodology;
- results that give a snapshot of commercial middleware versions.

Experiments were performed with the SBTVD's test suite that is applied to the middleware Ginga-D, more specifically, the part related to its component common core (CC) web services. The obtained results show that the original SBTVD test suite was indeed enhanced. In addition, the new test set revealed new failures in software structures.

II. Methodology

In this section, we first describe the techniques we have used in our research. Then, the methodology itself is presented. Its purpose is to evaluate test suites, identify gaps, and enhance their resources and structures.

A. Testing Infrastructure

1) Function Point Analysis (FPA): It is a method used to measure the functional size of software from the user's point of view. This approach was chosen because it measures complexity in development and maintenance projects, regardless of the technology employed. With such a complexity estimate for a given software element, one can point out where coverage gaps are probably located.

2) API Fail Injection Tests (AFIT): It is a technique to detect security vulnerabilities in applications that use APIs. It sends malicious requests to target APIs to obtain confidential information or control the underlying system. Such tests are performed to simulate attacks and identify security weaknesses, which can be fixed.

In this work, two techniques were used:

- parameter injection, which sends malicious parameters in a request to obtain confidential information or gain control over a system;
- script injection, which sends malicious scripts in a request to gain access to a system.

During exploratory tests, these techniques can be used to identify new conditions, weaknesses, and test elements to enhance a given test infrastructure.

3) The Proposed methodology: As mentioned, the proposed methodology aims to enhance existing test suites for DTV middleware modules. It is summarized in Fig. 1, while Fig. 2 details its third step, i.e., test identification and creation.

In Fig. 1, the evaluation of an original test suite begins with its analysis, which mainly involves FPA. Elements receive weights, which are then ranked to reveal the most complex ones. Then, we perform a match to choose the elements that, at the same time, present the highest FPA figures and the lowest number of initial tests. These elements are checked based on their specifications, so new scenarios are raised. Next, tests related to these new conditions are identified and implemented via an exploratory approach. Finally, they are applied to a real system with the goal of validation. As a result, an enhanced test suite is provided, with new tests based on ignored aspects.

Fig. 2 shows the detailed approach to creating new tests. Possible vulnerabilities are explored based on elements ignored by an original test suite (test gaps). It employs AFIT based on the previously identified test gaps in an exploratory fashion. Next, a conformant way of creating them is defined, again based on their specifications, which leads to implementations with the available system resources (e.g., calls to APIs, request transmission, etc.).

The expected outcome is an enhanced test suite, which has the potential to reveal new problems hidden in software layers.

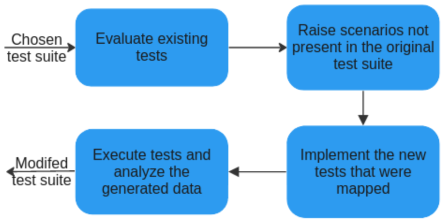
Chosen test suite
Evaluate existing tests
Raise scenarios not present in the original test suite
Implement the new tests that were mapped
Execute tests and analyze the generated data
Modifed test suite

Fig. 1. Our methodology to evaluate Ginga's CC-Web-Service Module.

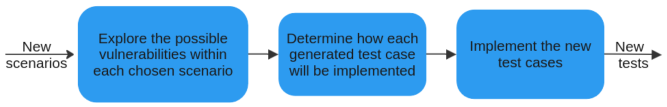
New scenarios
Explore the possible vulnerabilities within each chosen scenario
Determine how each generated test case will be implemented
Implement the new test cases
New tests

Fig. 2. The detailed view of the third step of the proposed methodology.

III. Experiments

We have conducted experiments to validate our methodology, using commercial versions of the SBTVD's middleware Ginga-D. The latter is a platform that provides two distinct environments: nested context language (NCL) and hypertext markup language (HTML) version 5 applications. Besides, it also offers a REST API via its element CC web services, which allows access to DTV resources.

This work focuses on the CC web services' API, which accepts requests from NCL applications, HTML5, and peripheral devices connected to the same local network. Consequently, it configures as a target for attacks, and as such, it must be deeply evaluated during the testing phases.

The test suite that was enhanced is the official release provided by the SBTVD's committee. Its tests and scenarios were analyzed to map vulnerabilities not previously considered. During the exploratory step, parameter and script injections were performed via hypertext transfer protocol (HTTP) requests (in their bodies) to the CC web services' REST API.

The enhanced test suite was then evaluated during test batteries performed by the development team of the company TPV, with three different versions of the middleware Ginga-D: the own TPV's version and two other commercial modules from various providers. The latter are referred as commercial Ginga-D 1 and commercial Ginga-D 2.

Two groups of the original test suite were tackled: authorization/authentication, which includes routes 8.1.1, 8.1.2, 8.7.1, and 8.7.2, and application environment control, which includes route 8.3.10. All of them can be checked in the associated standard [5]. The first group originally had XX tests, to which 21 new elements were added, while the second one had YY, which was increased by 15 new tests. The first group involved only calls to obtain authorization, which then gives access to DTV data and resources. Specifically, JavaScript injection was employed to collect information regarding variables that could be used as parameters in other routes, ultimately allowing access to the underlying system. The second group, in turn, employed a route to change the status of NCL applications. Specifically, it involved modifying an application's state using incorrect command dispatch and interruption of relevant processes (e.g., an application performing download), for instance. In addition, it also required the transmission of an NCL application to be controlled, aiming to analyze its behavior after receiving status-changing requests.

The new suite was then run with the three mentioned Ginga-D versions, whose results are shown in Tables I and II. The first one reveals that authentication/authorization routes are robust, in the chosen middleware versions, given that no weaknesses were identified and exploited. All routes, in the three middleware versions, returned suitable error messages (e.g., forbidden access and wrong parameter), and the middleware modules' integrity was not affected. However, the second table shows that the status-changing route can be used for attacks. The three middleware versions presented anomalous behaviors, which include frozen interfaces, software crashes, and reboots. Consequently, one may argue that this part is inherently weak and deserves more attention during the verification phases.

Note that the middleware versions: commercial Ginga-D 1 and commercial Ginga-D 2 are used in many receivers from different Brazilian manufacturers. It indicates that many products present such weaknesses, which should lead to changes in current verification processes.

TABLE I
Test results for the authentication/authorization routes.

Group	*Quantity of test applied*	*Tests passed*
TPV's version	21	21
Commercial Ginga-D 1	21	21
Commercial Ginga-D 2	21	21

TABLE II
Tests results for the status-changing route.

Group	*Quantity of test applied*	*Tests passed*
TPV's version	15	14
Commercial Ginga-D 1	15	11
Commercial Ginga-D 2	15	10

IV. Conclusion

This study proposed a methodology for enhancing test suites to evaluate DTV middleware modules. It includes a test-set evaluation followed by additional exploratory tests that consider the identified test gaps, based on the associated standards. To validate our methodology, we ran it on an official test suite provided by the SBTVD's committee. Its result was then applied to three different Ginga-D middleware versions: one developed by TPV and two obtained from different providers, focusing on application status-changing and authorization/verification. The results show that the proposed methodology is effective and that the chosen middleware versions are robust regarding authentication/authorizations. However, they may face frozen interface, crash, and reboot events when attacked using routes that change the status of a running application. Note that commercial Ginga-D 1 and commercial Ginga-D 2 are used by many Brazilian DTV-receiver manufacturers, which reveals a great number of defective devices that may suffer successful attacks.

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] R. Gopinath, C. Jensen, and A. Groce, "Mutations: How close are they to real faults?," in *2014 IEEE 25th International Symposium on Software Reliability Engineering*, pp. 189–200, IEEE, 2014.
[3] G. Gay, M. Staats, M. Whalen, and M. P. E. Heimdahl, "The risks of coverage-directed test case generation," *IEEE Transactions on Software Engineering*, vol. 41, no. 8, pp. 803–819, 2015.
[4] R. Niedermayr, E. Juergens, and S. Wagner, "Will my tests tell me if i break this code?," in *2016 IEEE/ACM International Workshop on Continuous Software Evolution and Delivery (CSED)*, pp. 23–29, IEEE, 2016.
[5] 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.