XAVIER: Language-Based Testing for XML Injection Attacks

Paul Kalbitzer
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
kalbitzer@cispa.de

José Antonio Zamudio Amaya
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
jose.zamudio@cispa.de

Andreas Zeller
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
andreas.zeller@cispa.de

***Abstract*—Web services are essential for online interactions, supporting critical tasks like banking and shopping, but their importance also makes them prime targets for attacks. Attackers try to manipulate data injecting malicious code, potentially compromising systems. Current approaches to preventing such attacks use techniques like attack grammars, symbolic execution, or machine learning to detect vulnerabilities or manually embed malicious payloads. Still, these techniques are limited and may miss vulnerabilities in untested parts of the service.**

In this paper, we propose XAVIER, a framework for detecting XML injection vulnerabilities. By leveraging the WSDL specification of a web service, XAVIER produces test inputs specifically designed to check for XML injection vulnerabilities. By creating XML messages that reflect the service's functionality and strategically modifying them using language-based techniques, XAVIER effectively simulates XML injection attacks, thereby enabling a comprehensive examination of web services for XMLi vulnerabilities. XAVIER is open source and easily extensible, providing a platform for future research in the field.

***Index Terms*—Language-based testing, XMLi, WSDL**

I. Introduction

Web services are fundamental to the operation of countless daily applications, forming the backbone of *online banking*, *e-commerce*, *streaming platforms*, and many others. They guarantee the uninterrupted transfer of information and data between systems, typically using the Extensible Markup Language (XML) [1], as a principal format for data transmission. XML is an integral component of numerous protocols, establishing it as a cornerstone of the web infrastructure.

However, the extensive use of XML also makes web services vulnerable to attacks, particularly through *XML injection vulnerabilities* (XMLi) [2]. By manipulating the structure of XML data, attackers can insert *malicious code* into this data exchange, thereby enabling attacks such as SQL injection (SQLi) [3] or cross-site scripting (XSS) [4] within XML communication. Such an attack can result in the *execution of malicious commands*, which may lead to the unauthorized access of data, security breaches, and, in the most severe cases, the complete compromise of the system.

Code injection attacks, particularly SQL injection (SQLi), are a persistent threat to web security. SQLi attacks involve modifying database queries to achieve malicious goals. Various defenses and testing approaches have been developed, many relying on grammar-based techniques to detect attack patterns.

Specifically, *WSDL-based testing techniques* generate test inputs based on *web service specifications,* creating XML messages that cover the entire schema. SOLMI [5], one of the most well-known state-of-the-art tools, mutates valid requests by embedding malicious payloads into schema-valid XML messages using the WSDL schema of the service under test. The alternative SOAPUI PRO[1] is a security testing tool that places a paywall between developers and the full range of security testing features. However, neither SOLMI nor SOAPUI PRO are available as open source, limiting further research in the field.

In this paper, we present XAVIER, an easily extensible open-source tool developed to mitigate XML injection vulnerabilities by utilizing the WSDL specification of a web service. XAVIER enables grammar-based generation of malicious inputs, leveraging the WSDL definition to ensure coverage of the entire schema of a service. XAVIER is both modular and highly extensible, making it adaptable for advanced security testing scenarios, as well as further research in the field. Most importantly, XAVIER is *language-based* at its core, using a *grammars* for XML and SQL that users can easily extend with new attack specifications. The usage of grammars also opens up XAVIER for semantic *constraints* [6] over grammar elements that allow for further customization. In its evaluation, XAVIER performs equally compared to SOAPUI PRO, achieving similar coverage efficiently.

In summary, our contributions are as follows:

- A generic language-based approach for WSDL-based testing of web services; and
- XAVIER, an extensible tool that implements the approach.

XAVIER is available as open source (Section VI).

II. Implementation

XAVIER is available in two versions: a classic command-line interface (CLI) and a graphical user interface (GUI). Both versions provide a simple, straightforward process for generating advanced XMLi attacks.

XAVIER revolves around the simulation of two attack types: *Scouts*, which are used for finding potential targets, and *Invaders*, which are deployed for actual attacks. To do so,

[1]https://www.soapui.org/

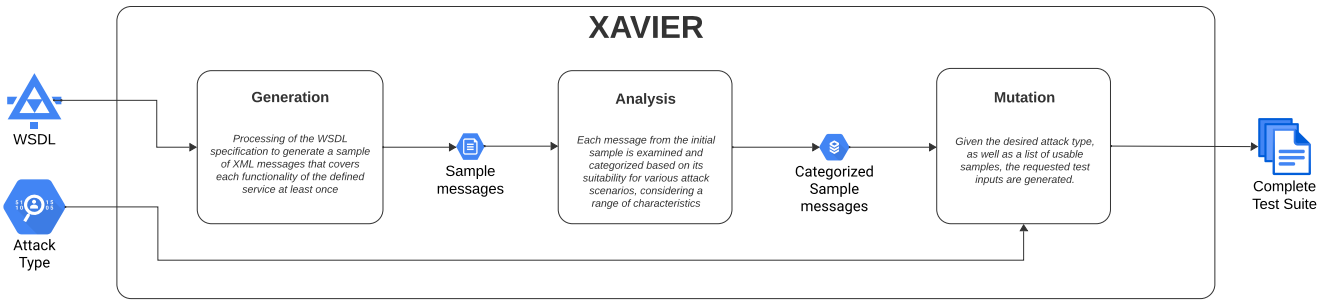


Fig. 1. How XAVIER works. XAVIER parses a WSDL web service specification to generate an initial set of XML messages. These are then categorized based on their suitability for attack scenarios and finally mutated according to the desired attack type.

XAVIER generates test inputs, derived from the WSDL description, for evaluating web services for potential XMLi vulnerabilities. Then, XAVIER parses the WSDL file and generates inputs for each function. These messages serve as the basis for subsequent tests and simulate real interactions with the service. Then, XAVIER generates both exploratory messages and targeted attacks. The workflow can be seen in Figure 1:

Generation: By processing a WSDL specification, XAVIER generates a starting example that covers each functionality of the defined service at least once.

Analysis: Every single message from the initial example is analyzed and categorized according to its suitability for different attack scenarios based on different features.

Mutation: Selected XML messages that are classified as suitable in the analysis are subject to specific mutations to simulate attack scenarios.

A. Scouts

Scouts are used to gather information about the target service without causing direct damage. They attempt to provoke unfiltered error messages that could reveal details about the implementation of the service. The implemented scout operators are:

Comment: An XML element is commented on to see how the service handles unexpected comments.

Duplicate: An XML element is duplicated to test whether the service accepts duplicate elements.

Tag: An undefined end tag is inserted into the message to provoke error messages that reveal the name of an unknown element.

Meta: XML meta characters are inserted in unexpected places to analyze the response of the service.

Delete: An XML element is removed to create a structural violation of the WSDL and gain information.

Quotation mark: An attribute of an XML message is manipulated by adding or removing quotation marks to test the service.

B. Invaders

Invaders are used to attack the service under test. To generate invaders, XAVIER leverages formal grammars to describe attack patterns, such as SQL injections. Figure 2 shows the grammar used to produce SQLi payloads, which are subsequently integrated into XML messages via our mutations. In all cases, the body of an XML element is replaced with a malicious payload, but the location is different:

```
<start>       ::= <pre> <whitespace> "OR"
                  <whitespace> <var>
                  <whitespace> <symbol>
                  <var2> <post>
                  <whitespace>
<pre>         ::= "'" | "a'" | "1"
<outer>       ::= "'" | "a" | "%27" | "1"
                  | "'%a%'"
<var>         ::= "version" | "database()"
                  | "user()"
                  | "''" | "1"
<symbol>      ::= "!=" | "=" | "LIKE"
<whitespace>  ::= " " | "/**/"
<var2>        ::= <outer> <whitespace>
<post>        ::= "#" | "--" | "#'"
```

Fig. 2. XAVIER's injection grammar. Further statements can be easily added to perform new attacks.

Direct Injection: The attacker has direct access to the desired target field.

Indirect Injection: The attacker has access to at least one field, but not the desired target field.

Sandwich Injection: The attacker has access to two input fields and his desired target field is in between them.

III. Usage Example

In this example, we are going to focus on simulating SQL injections with its invaders. First, the user needs to upload the WSDL file of the service under test. Second, the user needs to specify the type of attack desired to be performed. In our case, this attack is an SQL injection attack. XAVIER will automatically process the WSDL, producing an input suite with no attacks. Then, the inputs will be analyzed to find the best suited for an attack, based on the number of input files they provide, and the selected inputs will be mutated using an attack grammar.



Fig. 3. XAVIER workflow (GUI mode): First, the user uploads the WSDL to use. Then, a selection of attacks is shown. After the user selects the desired attack to perform, XAVIER produces a valid test suite.

XAVIER provides built-in grammars to mutate the inputs, but it is also possible to provide a custom grammar. After this process, XAVIER creates the corresponding test files based on the specified attack scenario and mutations. Figure 3 shows the entire process in XAVIER.

IV. Preliminary Evaluation

To analyze our prototype, we have compared the SQL injection generation functionality of XAVIER against SOAPUI PRO. The comparison has been based on eight different publicly accessible WSDL files. A guide on how to reproduce the results can be found in the XAVIER Gitlab repository (Section VI).

A. Research Questions

We aim to evaluate the effectiveness of XAVIER in comparison to SOAPUI PRO across their throughput, diversity of inputs, and how well each tool leverages the WSDL specification for web service testing.

RQ1: Throughput. How quickly can XAVIER produce valid inputs compared to SOAPUI PRO?

RQ2: Completeness. How well does XAVIER explore the WSDL compared to SOAPUI PRO?

RQ3: Diversity. How diverse are the inputs produced XAVIER compared to SOAPUI PRO?

B. Evaluation Setup

We conducted two iterations with timeouts of 1 and 5 seconds, respectively. To ensure a fair comparison, the configuration of SOAPUI PRO was adjusted to generate the next input immediately after creating a test input, without waiting for a response from the tested service.

C. Results

Table I shows the results for our eight subjects. In each case, we have measured the number of inputs generated by each tool, the average and maximum Levenshtein distance for each test suite, and the evenness of the Shannon index to check how well each tool explores the WSDL definition of each service.

D. RQ1: Throughput

On average, XAVIER generates approximately 2737.4 test inputs per second, which is less than SOAPUI PRO with an average of 6450.9 test inputs. The reason is that SOAPUI PRO uses a *fixed* list of SQL statements, which it simply has to run, whereas XAVIER generates a new SQL statement for each test input generated using a grammar. This deterministic behavior of SOAPUI PRO makes it produce a fixed amount of inputs, which XAVIER can quickly reach in every case.

E. RQ2: Completeness

We have determined the relative and absolute frequency for each element type to determine the diversity of the occurring element types, and we have used Shannon Entropy [7] to measure the structural diversity of the generated test inputs.

Looking at Table I, we can see that XAVIER performs equally to SOAPUI PRO, in terms of covering the functionality specified in the respective WSDL.

About the results relating to WSDL_7, after 1 second, it is noticeable that SOAPUI PRO failed to create test inputs for each function in this time window, which means that at least one functionality described in the WSDL could not be tested.

F. RQ3: Diversity

We use the Levenshtein Distance [8] to evaluate the *diversity* of the generated test suites. We assume that the more different two inputs are from each other, the higher their probability of triggering other execution paths.

The results show that the entries on the list used by SOAPUI PRO are more different than the statements generated by our grammars. The reason for this may be that grammars are often designed to generate the smallest possible inputs. However, this is a well-known problem, and there are methods to adjust the fuzzer to produce longer and better-sampled inputs [9].

In comparison, XAVIER performs better on WSDL_7, which describes functions with a large number of parameters.

V. Limitations

Since WSDL is XML-based, it is also possible to define not only a data type for an element but also other constraints such as a pattern to be adhered to. When generating values, XAVIER restricts itself exclusively to the consideration of the specified data type and not to any other constraints.

WSDL	Timeout	Tool	#Inputs	Avg. Levenshtein	Max. Levenshtein	Shannon Index	Evenness
HelloService (WSDL_0)	1	XAVIER	**2791**	20.19	41	2.08	1.00
		SOAPUI PRO	813	**66.29**	**477**	2.08	1.00
	5	XAVIER	**13967**	20.29	43	2.08	1.00
		SOAPUI PRO	813	**66.29**	**477**	2.08	1.00
TempConvert (WSDL_1)	1	XAVIER	2812	47.24	95	2.43	0.98
		SOAPUI PRO	**4878**	**92.32**	**514**	2.43	0.98
	5	XAVIER	**13917**	47.22	97	2.43	0.98
		SOAPUI PRO	4878	**92.32**	**514**	2.43	0.98
DataFlex ISBN (WSDL_2)	1	XAVIER	2804	21.36	43	2.25	0.98
		SOAPUI PRO	**3252**	**66.79**	**500**	2.25	0.98
	5	XAVIER	**13863**	21.27	45	2.25	0.98
		SOAPUI PRO	3252	**66.79**	**500**	2.25	0.98
DataFlex Country (WSDL_3)	1	XAVIER	2774	52.54	100	**3.04**	0.83
		SOAPUI PRO	**11842**	**95.74**	**549**	3.02	0.83
	5	XAVIER	11234	52.37	102	3.03	0.83
		SOAPUI PRO	**11842**	**97.49**	**549**	3.03	0.83
DNE Calculator WS (WSDL_4)	1	XAVIER	2754	59.07	99	**2.58**	0.93
		SOAPUI PRO	**10349**	**86.92**	**495**	2.57	0.93
	5	XAVIER	**13599**	59.04	101	2.58	0.93
		SOAPUI PRO	13008	**87.05**	**495**	2.58	0.93
DNE Number Conversion (WSDL_5)	1	XAVIER	2791	28.21	57	2.43	0.98
		SOAPUI PRO	**3252**	**73.79**	**508**	2.43	0.98
	5	XAVIER	**13802**	28.23	59	2.43	0.98
		SOAPUI PRO	3252	**73.79**	**508**	2.43	0.98
DataFlex Temp (WSDL_6)	1	XAVIER	2755	**137.02**	259	**2.74**	**0.95**
		SOAPUI PRO	**9756**	111.01	**573**	2.72	0.94
	5	XAVIER	**13807**	**151.07**	297	**2.74**	**0.95**
		SOAPUI PRO	9756	111.01	**573**	2.72	0.94
ndfdXML (WSDL_7)	1	XAVIER	2418	**419.69**	664	**3.56**	0.96
		SOAPUI PRO	**7465**	145.83	**739**	3.27	0.96
	5	XAVIER	**12254**	**457.76**	**798**	3.56	0.96
		SOAPUI PRO	7465	166.43	739	3.56	0.96

TABLE I
ANALYSIS OF LEVENSHTEIN DISTANCE AND SHANNON INDEX FOR VARIOUS WSDL EXAMPLES.

VI. CONCLUSION AND FUTURE WORK

XAVIER provides a comprehensive solution for the detection of XML injection vulnerabilities in web services by leveraging the WSDL definition to generate test cases. Preliminary results show that it explores the WSDL definition as thoroughly as the alternative. By identifying potential vulnerabilities, XAVIER adopts a proactive stance in securing web services against malicious attacks.

After identifying vulnerabilities, XAVIER could propose mitigation strategies or suggest fixes to developers. Implementing such functionality would make XAVIER not only a detection tool but also a component in the vulnerability-fixing process.

Existing fuzzers, [6], [9], are capable of handling syntactic structures while satisfying constraints. Incorporating such fuzzers would enable XAVIER to generate valid, constraint-aware inputs even in scenarios where the grammar alone may be insufficient.

These fuzzers would allow XAVIER to read the constraints from the WSDL definition and feed them alongside the grammar to produce valid inputs even in the more complex scenarios.

XAVIER is available as open source at

https://gitlab.com/P.K-98/x-avier

REFERENCES

[1] N. A. Nordbotten, "XML and web services security standards," *IEEE Communications Surveys & Tutorials*, vol. 11, no. 3, pp. 4–21, 2009.

[2] V. R. Mouli and K. Jevitha, "Web services attacks and security-a systematic literature review," *Procedia Computer Science*, vol. 93, pp. 870–877, 2016.

[3] M. Lawal, A. B. M. Sultan, and A. O. Shakiru, "Systematic literature review on SQL injection attack," *International Journal of Soft Computing*, vol. 11, no. 1, pp. 26–35, 2016.

[4] U. Sarmah, D. Bhattacharyya, and J. K. Kalita, "A survey of detection methods for XSS attacks," *Journal of Network and Computer Applications*, vol. 118, pp. 113–143, 2018.

[5] S. Jan, C. D. Nguyen, and L. C. Briand, "Automated and effective testing of web services for XML injection attacks," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*, pp. 12–23, 2016.

[6] D. Steinhöfel and A. Zeller, "Input invariants," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2022, (New York, NY, USA), p. 583–594, Association for Computing Machinery, 2022.

[7] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[8] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet physics doklady*, vol. 10, pp. 707–710, 1966.

[9] J. A. Zamudio Amaya, "Shaping test inputs in grammar-based fuzzing," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2024, (New York, NY, USA), p. 1901–1905, Association for Computing Machinery, 2024.