String Analysis for Vulnerability Detection and Repair*

Tevfik Bultan

Department of Computer Science, University of California, Santa Barbara
bultan@cs.ucsb.edu

Abstract. String manipulation errors in input validation and sanitization code are a common source for security vulnerabilities in web applications. This short survey summarizes the string analysis techniques we developed that can automatically identify and repair such vulnerabilities. Our approach (1) extracts client- and server-side input validation and sanitization functions, (2) models them as deterministic finite automata (DFA) using symbolic fixpoint computations, and (3) identifies errors in input validation and sanitization code by either checking them with respect to manually specified attack patterns, or by identifying inconsistencies in input validation and sanitization operations at the client and server-side. Furthermore, we developed automated repair techniques that strengthen the input validation and sanitization checks in order to eliminate identified vulnerabilities. We implemented these techniques in two tools: Stranger (STRing AutomatoN GEneratoR) and SemRep (SEMantic differential REPair), which are available at: http://www.cs.ucsb.edu/~vlab/tools.html. Our experimental evaluation demonstrates that these techniques are very promising: when applied to a set of real-world web applications, our techniques are able to automatically identify a large number of security vulnerabilities and repair them.

1 Motivation

According to the Common Vulnerabilities and Exposures (CVE) repository, web application vulnerabilities form a significant portion of all reported computer security vulnerabilities [8]. Additionally, Open Web Application Security Project (OWASP) compiles a top ten list to identify the most critical security flaws in

* This material is based on research sponsored by NSF under grants CCF-1423623, CNS 1116967, CCF 0916112 and by DARPA under agreement number FA8750-15-2-0087. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government. Part of this research was conducted while Tevfik Bultan was visiting Koç University in İstanbul, Turkey, supported by a research fellowship from TÜBİTAK under the BİDEB 2221 program.

web applications [10]. According to the OWASP top ten lists compiled in 2007, 2010 and 2013, Cross-site Scripting (XSS) and SQL Injection (SQLI) are always among the top three web application vulnerabilities.

XSS and SQLI vulnerabilities are due to improper input validation and sanitization. Errors in validation and sanitization of user input can lead to vulnerabilities for web applications, and since web applications are globally accessible, these vulnerabilities can be exploited by malicious users all around the world. Different layers of a web application interact through commands that often embed user input and are written in many languages, such as XML, SQL, and HTML. Hence, programs that propagate and use malicious user inputs without validation and sanitization, or with improper validation and sanitization, are vulnerable to attacks such as XSS and SQLI.

2 String Analysis

In order to provide a general framework for eliminating vulnerabilities related to input validation and sanitization we have to address multiple issues, such as, automated extraction of validation and sanitization operations from a given web application, detecting if there is a vulnerability by automatically analyzing extracted string operations, and automatically generating a repair that eliminates the identified vulnerability. During last several years, we worked on various aspects of this problem at the Verification Laboratory of the University of California, Santa Barbara (`http://www.cs.ucsb.edu/~vlab/`). A summary of the techniques we developed in this domain is provided below.

Automata-based String Analysis: First, we developed an automata-based approach for the verification of string operations in PHP programs based on symbolic string analysis [15, 12, 14]. String analysis is a static analysis technique that determines the values that a string expression can take during program execution at a given program point. This information can be used to verify that string values are sanitized properly, and to detect programming errors and security vulnerabilities. In our string analysis approach, we encode the set of string values that string variables can take as Deterministic Finite Automata (DFA). We implement all string functions using a symbolic automata representation (MBDD representation from the MONA automata package [7]) and leverage efficient manipulations on MBDDs, e.g., determinization and minimization. Particularly, we developed a novel algorithm for language-based replacement. Our replacement function takes three DFAs as arguments and outputs a DFA. Finally, we apply a widening operator defined on automata to approximate fixpoint computations [6]. If this conservative approximation does not match any attack patterns (specified as regular expressions), we conclude that the program does not contain any errors or vulnerabilities. Our experimental results demonstrate that our approach works quite well in checking the correctness of sanitization operations in real-world PHP applications. We implemented these automata-based string analysis techniques in a tool called Stranger (STRing AutomatoN GEneratoR) which is available at: `http://www.cs.ucsb.edu/~vlab/stranger/`.

Computing Vulnerability Signatures: Based on automata-based string analysis, we developed techniques that, given a program and an attack pattern (specified as a regular expression), generate string-based vulnerability signatures, i.e., a characterization that includes all malicious inputs that can be used to generate attacks [11, 13]. Using forward reachability analysis, we compute an over-approximation of all possible values that string variables can take at each program point. Intersecting these with the attack pattern yields the potential attack strings if the program is vulnerable. Using backward analysis we compute an over-approximation of all possible inputs that can generate those attack strings. In addition to identifying existing vulnerabilities and their causes, these vulnerability signatures can be used to filter out malicious inputs. Our approach extends the prior work on automata-based string analysis by providing a backward symbolic analysis that includes a symbolic pre-image computation for DFAs on common string manipulating functions such as concatenation and replacement.

Relational String Analysis: String analysis techniques based on standard single-track automata cannot precisely represent relational constraints that involve multiple variables. To address this problem, we developed novel relational string verification techniques based on multi-track automata [18, 19]. Multi-track automata recognize tuples of strings by reading multiple inputs concurrently. Value of each string variable is represented as one of the input tracks of the automata, which enables us to represent relational constraints on multiple variables. Using this symbolic representation we are able verify relational properties that involve multiple string variables.

Verifying string manipulating programs is an undecidable problem in general [18, 19] and any approximate string analysis technique has an inherent tension between efficiency and precision. We developed sound abstraction techniques for strings and string operations that allow for both efficient and precise verification of string manipulating programs [16]. We first defined an abstraction called regular abstraction which enables us to perform string analysis using multi-track automata as a symbolic representation. We then introduced two other abstractions—alphabet abstraction and relation abstraction—that can be used in combination to tune the analysis precision and efficiency. We showed that these abstractions form an abstraction lattice that generalizes the string analysis techniques studied previously in isolation, such as string length analysis or non-relational string analysis. Finally, we empirically evaluated the effectiveness of these abstraction techniques with respect to several benchmarks and an open source application, demonstrating that abstraction techniques can improve the performance without loss of accuracy of the analysis when a suitable abstraction class is selected.

Handling Operations on String Length: Another interesting class of properties involve relationships among the string and integer variables. The lengths of the strings in a regular language form a semilinear set. Since we use automata to encode possible values of string variables, we can use this observation to encode

the lengths of string variables as semilinear sets. Moreover, we can construct automata that recognize these semilinear sets, i.e., recognize the possible lengths of a string variable. We developed techniques that construct length automata that accept the unary or binary representations of the lengths of the strings accepted by string automata [17]. These length automata can be integrated with an arithmetic automaton that recognizes the valuations of the integer variables at a program point. We developed a static analysis technique that uses these automata in a forward fixpoint computation with widening [6], and is able to capture relationships among the lengths of the string variables and the values of the integer variables. This composite string and integer analysis enables us to verify properties that cannot be verified using string analysis or integer analysis alone.

Client-side String Analysis: Client-side computation in web applications is becoming increasingly common due to the popularity of powerful client-side programming languages such as JavaScript. Client-side computation is commonly used to improve an applications responsiveness by validating user inputs before they are sent to the server. We developed string analysis techniques for checking if a client-side input validation function conforms to a given policy [2]. In our approach, input validation policies are expressed using two regular expressions, one specifying the maximum policy (the upper bound for the set of inputs that should be allowed) and the other specifying the minimum policy (the lower bound for the set of inputs that should be allowed). Using our analysis we can identify two types of errors 1) the input validation function accepts an input that is not permitted by the maximum policy, or 2) the input validation function rejects an input that is permitted by the minimum policy. We implemented our analysis using dynamic slicing to automatically extract the client-side input validation functions from web applications, and using automata-based string analysis to analyze the extracted functions. Our experiments demonstrate that our approach is effective in finding errors in input validation functions that we collected from real-world applications and from tutorials and books for teaching JavaScript.

Differential String Analysis: Developers typically perform redundant input validation in both the front-end (client) and the back-end (server) components of a web application. As we mentioned above, client-side validation is used to improve the responsiveness of the application, as it allows for responding without communicating with the server, whereas server-side validation is necessary for security reasons, as malicious users can easily circumvent client-side checks. We developed a differential string analysis technique that (1) automatically extracts client- and server-side input validation functions, (2) models them as DFAs, and (3) compares client- and server-side DFAs to identify and report the inconsistencies between the two sets of checks [3]. Our initial evaluation of the technique is promising: when applied to a set of real-world web applications, our technique was able to automatically identify a large number of inconsistencies in their input validation functions.

Automated Test-Case Generation: Automata-based static string analysis techniques we described above can be used to automatically compute vulnerability signatures (represented as automata) that characterize all the inputs that can exploit a vulnerability. However, there are several factors that limit the applicability of static string analysis techniques in general: 1) undecidability of static string analysis requires the use of approximations leading to false positives, 2) static string analysis tools do not handle all string operations, 3) dynamic nature of the scripting languages makes static analysis difficult. As a complementary approach to static string analysis techniques, we developed automated testing techniques for checking string manipulating code [4]. In particular, we showed that vulnerability signatures computed for deliberately insecure web applications (developed for demonstrating different types of vulnerabilities) can be used to generate test cases for other applications. Given a vulnerability signature represented as an automaton, we developed algorithms for test case generation based on state, transition, and path coverage. These automatically generated test cases can be used to test applications that are not analyzable statically, and to discover attack strings that demonstrate how the vulnerabilities can be exploited.

Automated Vulnerability Repair: Based on automata-based static string analysis, we developed techniques that automatically generate sanitization statements for patching vulnerable web applications [13]. Our approach consists of three phases: Given an attack pattern we first conduct a vulnerability analysis to identify if strings that match the attack pattern can reach the security-sensitive functions. Next, we compute vulnerability signatures that characterize all input strings that can exploit the discovered vulnerability. Given the vulnerability signatures, we then construct sanitization statements that 1) check if a given input matches the vulnerability signature and 2) modify the input in a minimal way so that the modified input does not match the vulnerability signature. Our approach is capable of generating relational vulnerability signatures (and corresponding sanitization statements) for vulnerabilities that are due to more than one input.

Although attack patterns are useful for characterizing possible attacks, they need to be written manually, which means that they may contain errors, and they may not be available for newer attacks. We developed an automated differential repair technique for input validation and sanitization functions that does not require manual specification of expected behavior [1]. Differential repair can be used within an application to repair client and server-side code with respect to each other, or across applications in order to strengthen the validation and sanitization checks. Given a reference and a target function, our differential repair technique strengthens the validation and sanitization operations in the target function based on the reference function. It does this by synthesizing three patches: a validation, a length, and a sanitization patch. Our automated patch synthesis algorithms are based on forward and backward symbolic string analyses that use automata as a symbolic representation. Composition of the three automatically synthesized patches with the original target function results in the repaired function, which provides stronger validation and sanitization than both the target and the reference functions. We implemented these automata-based

differential repair techniques in a tool called SemRep (SEMantic differential REPair), which is available at: `https://github.com/vlab-cs-ucsb/SemRep`.

Model Counting for String Constraints: Symbolic execution has become one of the most widely used automated bug detection techniques. In order to apply symbolic execution to analysis of string manipulating programs, it is necessary to develop constraint solvers that can check satisfiability of string constraints. Our automata-based string analysis techniques can be used to build a string constraint solver. To facilitate this, we developed a string analysis library called LibStranger (available at `https://github.com/vlab-cs-ucsb/LibStranger`) based on our tool Stranger. In a recent independent empirical study for evaluating string constraint solvers, Stranger was determined to be the best in terms of precision and efficiency for symbolic execution of Java programs [9].

However, for quantitative and probabilistic program analyses, checking the satisfiability of a constraint is not sufficient. In addition to checking satisfiability, it is also necessary to count the number of satisfying solutions. Recently, we developed a string constraint solver that, given a constraint, 1) constructs an automaton that accepts all solutions that satisfy the constraint, 2) generates a function that, given a length bound, gives the total number of solutions within that bound [5]. Our approach relies on the observation that, using an automata-based constraint representation, model counting reduces to path counting, which can be solved precisely. We demonstrated the effectiveness of our approach on a large set of string constraints extracted from real-world web applications. We implemented these techniques in a tool called ABC (Automata Based model Counter for constraints) (see `http://www.cs.ucsb.edu/~vlab/ABC/`).

3 Conclusion

String manipulation is an important part of modern software systems, and errors in string manipulating code continue to be a significant software dependability problem. Our results demonstrate that automata-based string analysis techniques are promising tools in eliminating software dependability problems that are due to string manipulation.

References

1. Muath Alkhalaf, Abdulbaki Aydin, and Tevfik Bultan. Semantic differential repair for input validation and sanitization. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, pages 225–236, 2014.
2. Muath Alkhalaf, Tevfik Bultan, and Jose L. Gallegos. Verifying client-side input validation functions using string analysis. In *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, pages 947–957, 2012.
3. Muath Alkhalaf, Shauvik Roy Choudhary, Mattia Fazzini, Tevfik Bultan, Alessandro Orso, and Christopher Kruegel. Viewpoints: differential string analysis for discovering client- and server-side input validation inconsistencies. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, pages 56–66, 2012.

4. Abdulbaki Aydin, Muath Alkhalaf, and Tevfik Bultan. Automated test generation from vulnerability signatures. In *7th IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 193–202, 2014.
5. Abdulbaki Aydin, Lucas Bang, and Tevfik Bultan. Automata-based model counting for string constraints. In *Proceedigns of the 27th International Conference on Computer Aided Verification (CAV)*, 2015.
6. Constantinos Bartzis and Tevfik Bultan. Widening arithmetic automata. In *Proceedings of the 16th International Conference on Computer Aided Verification (CAV)*, pages 321–333, 2004.
7. BRICS. The MONA project. `http://www.brics.dk/mona/`.
8. CVE. Common Vulnerabilities and Exposures. `http://www.cve.mitre.org`.
9. Scott Kausler and Elena Sherman. Evaluation of string constraint solvers in the context of symbolic execution. In *Proceedings of the 29th ACM/IEEE International Conference on Automated software engineering (ASE)*, pages 259–270, 2014.
10. Open Web Application Security Project (OWASP). Top ten project. `https://www.owasp.org/index.php/Category\:OWASP\_Top\_Ten\_Project`.
11. Fang Yu, Muath Alkhalaf, and Tevfik Bultan. Generating vulnerability signatures for string manipulating programs using automata-based forward and backward symbolic analyses. In *Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 605–609, 2009.
12. Fang Yu, Muath Alkhalaf, and Tevfik Bultan. Stranger: An automata-based string analysis tool for php. In *Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 154–157, 2010.
13. Fang Yu, Muath Alkhalaf, and Tevfik Bultan. Patching vulnerabilities with sanitization synthesis. In *Proceedings of the 33rd International Conference on Software Engineering (ICSE)*, pages 131–134, 2011.
14. Fang Yu, Muath Alkhalaf, Tevfik Bultan, and Oscar H. Ibarra. Automata-based symbolic string analysis for vulnerability detection. *Formal Methods in System Design*, 44(1):44–70, 2014.
15. Fang Yu, Tevfik Bultan, Marco Cova, and Oscar H. Ibarra. Symbolic string verification: An automata-based approach. In *Proceedings of the 15th International SPIN Workshop on Model Checking Software (SPIN)*, pages 306–324, 2008.
16. Fang Yu, Tevfik Bultan, and Ben Hardekopf. String abstractions for string verification. In *Proceedings of the 18th International SPIN Workshop on Model Checking Software (SPIN)*, pages 20–37, 2011.
17. Fang Yu, Tevfik Bultan, and Oscar H. Ibarra. Symbolic string verification: Combining string analysis and size analysis. In *Proceedings of the 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 322–336, 2009.
18. Fang Yu, Tevfik Bultan, and Oscar H. Ibarra. Relational string verification using multi-track automata. In *Proceedings of the 15th International Conference on Implementation and Application of Automata (CIAA)*, pages 290–299, 2010.
19. Fang Yu, Tevfik Bultan, and Oscar H. Ibarra. Relational string verification using multi-track automata. *Int. J. Found. Comput. Sci.*, 22(8):1909–1924, 2011.