STRANGER: An Automata-based String Analysis Tool for PHP *

Fang Yu, Muath Alkhalaf, and Tevfik Bultan

Department of Computer Science
University of California, Santa Barbara, CA, USA
{yuf, muath, bultan}@cs.ucsb.edu

Abstract. STRANGER is an automata-based string analysis tool for finding and eliminating string-related security vulnerabilities in PHP applications. STRANGER uses symbolic forward and backward reachability analyses to compute the possible values that the string expressions can take during program execution. STRANGER can automatically (1) prove that an application is free from specified attacks or (2) generate vulnerability signatures that characterize all malicious inputs that can be used to generate attacks.

1 Introduction

Web applications provide critical services over the Internet and frequently handle sensitive data. Unfortunately, Web application development is error prone and results in applications that are vulnerable to attacks by malicious users. The global accessibility of critical Web applications make this an extremely serious problem. According to the Open Web Application Security Project (OWASP)'s top ten list that identifies the most serious web application vulnerabilities [6], the top three vulnerabilities are: 1) Cross Site Scripting (XSS), 2) Injection Flaws (such as SQL Injection) and 3) Malicious File Execution (MFE). A XSS vulnerability results from the application inserting part of the user's input in the next HTML page that it renders. Once the attacker convinces a victim to click on a URL that contains malicious HTML/JavaScript code, the user's browser will then display HTML and execute JavaScript that can result in stealing of browser cookies and other sensitive data. An SQL Injection vulnerability results from the application's use of user input in constructing database statements. The attacker can invoke the application with a malicious input that is part of an SQL command that the application executes. This permits the attacker to damage or get unauthorized access to data stored in a database. MFE vulnerabilities occur if developers directly use or concatenate potentially hostile input with file or stream functions, or improperly trust input files. All these vulnerabilities involve string manipulation operations and they occur due to inadequate sanitization and inappropriate use of input strings provided by users.

We present a new tool called STRANGER (STRing AutomatoN GEneratoR) that can be used to check the correctness of string manipulation operations in web applications. STRANGER implements an automata-based approach [8, 9] for automatic verification of string manipulating programs based on symbolic string analysis. String analysis is a

* This work is supported by NSF grants CCF-0916112 and CCF-0716095.

static analysis technique that determines the values that a string expression can take during program execution at a given program point.

STRANGER encodes the set of string values that string variables can take as deterministic finite automata (DFAs). STRANGER implements both the *pre*- and *post*-image computations of common string functions on DFAs, including a novel algorithm for *language*-based replacement [9]. This replacement function takes three DFAs as arguments and outputs a DFA and can be used to model PHP replacement commands, e.g., `preg_replace()` and `str_replace()`, as well as many PHP sanitization routines, e.g., `addslashes()`, `htmlspecialchars()` and `mysql_real_escape_string()`. STRANGER implements all string manipulation functions using a symbolic automata representation (MBDD representation from the MONA automata package [2]) and leverages efficient manipulations on MBDDs such as determinization and minimization. This symbolic encoding also enables STRANGER to deal with large alphabets.

STRANGER combines forward and backward reachability analyses [8] and is capable of (1) checking the correctness of sanitization routines and proving that programs are free from specified attacks, and (2) identifying vulnerable programs, as well as generating non-trivial vulnerability signatures. Using forward reachability analysis, STRANGER computes an over-approximation of all possible values that string variables can take at each program point. If this conservative approximation does not include any attack pattern, STRANGER concludes that the program does not contain any vulnerabilities. Otherwise, intersecting these with attack patterns yields the potential attack strings. Using backward analysis STRANGER automatically generates string-based vulnerability signatures, i.e., a characterization that includes all malicious inputs that can be used to generate attack strings. In addition to identifying existing vulnerabilities and their causes, these vulnerability signatures can be used to filter out malicious inputs.

2 Tool Description

STRANGER uses Pixy [4] as a front end and MONA [2] automata package for automata manipulation. STRANGER takes a PHP program as input and automatically analyzes it and outputs the possible XSS, SQL Injection, or MFE vulnerabilities in the program. For each input that leads to a vulnerability, it also outputs the vulnerability signature, i.e., an automaton (in a dot format) that characterizes all possible string values for this input which may exploit the vulnerability. The architecture of STRANGER is shown in Figure 1. The tool consists of the following parts.

PHP Parser and Taint Analyzer. The first step in our analysis is to parse the PHP program and construct the control flow graph (CFG). This is done by Pixy. PHP programs do not have a single entry point as in some other languages such as C and Java, so we process each script by itself along with all files included by that script. The CFG is passed to the taint analyzer in which alias and dependency analyses are performed to generate dependency graphs. A dependency graph specifies how the inputs flow to a sensitive sink with respect to *string operations*. The number of its nodes is linear to the number of the string operations in the program under a static single assignment environment. Loop structures contribute cyclic dependency relations. If no tainted data

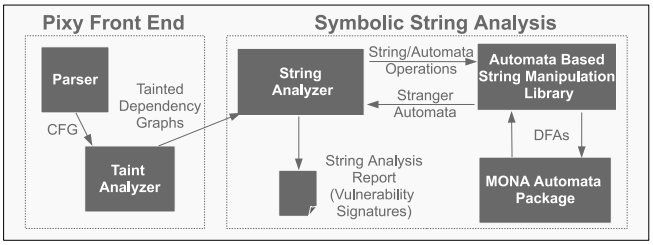

Fig. 1. The Architecture of STRANGER

flow to the sink, taint analysis reports the dependency graph to be secure; otherwise, the dependency graph is tainted and passed to the string analyzer for more inspection.

String Analyzer. The string analyzer implements our vulnerability (forward and backward) analysis [8] on the tainted dependency graphs found by taint analysis. The dependency graphs are pre-processed to optimize the reachability analyses. First, a new acyclic dependency graph is built where all the nodes in a cycle (identifying cyclic dependency relations) are replaced by a single strongly connected component (SCC) node. The vulnerability analysis is conducted on the acyclic graph so that the nodes that are not in a cycle are processed only once. In the forward analysis, we propagate the post images to nodes in the topological order, initializing input nodes to DFAs accepting arbitrary strings. Upon termination, we intersect the language of the DFA of the sink node with the attack pattern. If the intersection is empty, we conclude that the sink is not vulnerable with respect to the attack pattern. Otherwise, we perform the backward analysis and propagate the pre images to nodes in the reverse topological order, initializing the sink node to a DFA that accepts the intersection of the result of the forward analysis and the attack pattern. Upon termination, the vulnerability signatures are the results of the backward analysis for each input node. For both analyses, when we hit an SCC node, we switch to a work queue fixpoint computation [8] on nodes that are part of the SCC represented by the SCC node. During the fixpoint computation we apply automata widening [1] on reachable states to accelerate the convergence of the fixpoint computation. We added the ability to choose when to apply the widening operator. This option enables computation of the precise fixpoint in cases where the fixpoint computations converges after a certain number of iterations without widening. We also incorporate a coarse widening operator [1] that guarantees the convergence to avoid potential infinite iterations of the fixpoint computation.

String Manipulation Library. String manipulation library (SML) handles all core string and automata operations such as replacement, concatenation, prefix, suffix, intersection, union, and widen. During the vulnerability analysis, all string and automata manipulation operations that are needed to decorate a node in a dependency graph are sent to SML along with the string and/or automata parameters. SML, then, executes the operation and returns back the result as an automaton. A Java class called $StrangerAutomaton$ has been used as the type of the parameters and results. The class follows a well defined interface so that other automata packages can be plugged

in and used with the string analyzer instead of SML. SML is also decoupled from the vulnerability analysis component so that it can be used with other string analysis tools. $StrangerAutomaton$ encapsulates $libstranger.so$ shared library that has the actual string manipulation code implemented in C to get a faster computation and a tight control on memory. We used JNA (Java Native Access) to bridge the two languages.

3 Experiments and Conclusions

We have experimented with STRANGER on several benchmarks extracted from known vulnerable web applications [9]. For each vulnerable benchmark, we also generated a modified version where string manipulation errors are fixed. STRANGER took less than few seconds to analyze each benchmark. It successfully reported all known vulnerabilities, generated the vulnerability signatures, and verified that the modified version is secure and free from the previously reported vulnerabilities. We have also conducted a case study on `SimpGB-1.49.0` - a PHP guestbook web application. `SimpGB` consists of 153 php files containing 44000+ lines of code. Using a machine with Intel Core 2 Due 2.5 GHz with 4GB of memory running Linux Ubuntu 8.04, STRANGER took 231 minutes to check XSS vulnerabilities for all entries of executable PHP scripts and concluded 304 possible vulnerabilities out of 15115 sinks. STRANGER took 175 minutes to reveal 172 possible SQL Injection vulnerabilities from 1082 sinks, and 151 minutes to reveal 26 possible MFE vulnerabilities from 236 sinks.

In sum, we presented a string analysis tool for verification of web applications, focusing on SQLI, XSS and MFE attacks. In addition to identifying vulnerabilities and generating vulnerability signatures of vulnerable applications, STRANGER can also verify the absence of vulnerabilities in applications that use proper sanitization. Compared to grammar-based string analysis tools [3, 5, 7], STRANGER features specific automata-based techniques including automata widening [1], language-based replacement [9] and symbolic automata encoding and manipulation [2]. STRANGER and several benchmarks are available at `http://www.cs.ucsb.edu/~vlab/stranger`.

References

1. C. Bartzis and T. Bultan. Widening arithmetic automata. In *CAV*, pages 321–333, 2004.
2. BRICS. The MONA project. `http://www.brics.dk/mona/`.
3. A. Christensen, A. Møller, and M. Schwartzbach. Precise analysis of string expressions. In *SAS*, pages 1–18, 2003.
4. N. Jovanovic, C. Krügel, and E. Kirda. Pixy: A static analysis tool for detecting web application vulnerabilities (short paper). In *S&P*, pages 258–263, 2006.
5. Y. Minamide. Static approximation of dynamically generated web pages. In *WWW*, pages 432–441, 2005.
6. OWASP. Top ten project. `http://www.owasp.org/`, May 2007.
7. G. Wassermann and Z. Su. Sound and precise analysis of web applications for injection vulnerabilities. In *PLDI*, pages 32–41, 2007.
8. F. Yu, M. Alkhalaf, and T. Bultan. Generating vulnerability signatures for string manipulating programs using automata-based forward and backward symbolic analyses. In *ASE*, 2009.
9. F. Yu, T. Bultan, M. Cova, and O. H. Ibarra. Symbolic string verification: An automata-based approach. In *SPIN*, pages 306–324, 2008.