About

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

- STRANGER is an automata-based string analysis tool for finding and eliminating string-related security vulnerabilities in PHP applications.
- Uses symbolic forward and backward reachability analyses to compute the possible values that the string expressions can take during program execution.

Why Audit PHP?

- Web applications provide critical services and handle sensitive data.
- Web application development is error prone.
- Global Accessibility of web applications.
STRANGER String Analysis for x86 Binaries

Background

OWASP Top 10

1. Cross Site Scripting (XSS)
2. Injection Flaws (Such as SQL)
3. Malicious File Execution (RFI, LFI, etc.)

All of these are string vulnerabilities, stemming from a lack of sanitization.

Introduction

STRing AutomatoN GEneratoR

Used to check the correctness of string manipulation operations in web applications
- Automata-based approach
- Uses symbolic analysis

String Analysis

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

How

STRANGER encodes the set of string values that string variables can take as deterministic finite automata (DFAs).

STRANGER implements all string manipulation functions using a symbolic automata representation.

Capabilities

STRANGER combines forward and backward reachability analyses and is capable of:
- Checking the correctness of sanitization routines and proving that programs are free from specified attacks
- Identifying vulnerable programs, as well as generating non-trivial vulnerability signatures

Forward Reachability

Forward Reachability Analysis

Allows STRANGER to identify all possible values string variables can take at each program point
- If this approximation does not contain any attack pattern, STRANGER concludes the program does not have any vulnerabilities
- Intersecting the approximation with the attack pattern yields the potential attack strings
- Using Backward Analysis
  - STRANGER generates vulnerability signatures.
  - A characterization of all malicious inputs that can be used to generate attack strings

Tool Description

Credit: [STRANGER: An Automata-based String Analysis Tool for PHP]

Uses Pixy as a front end
Uses MONA automata package for automata manipulation
Tool Description

Input and Output
- **IN**: PHP program(s)
- **OUT**: Possible XSS, SQL, and MFE vulnerabilities
- **OUT**: Vulnerability signatures

The Vulnerability signatures are an automaton that characterizes all possible string values which may exploit the vulnerability.

PHP Parser and Taint Analyzer

- **Parser**: Parse PHP program and create control flow graph (CFG)
  - Process each script by itself (along with all includes)
- **Taint Analyzer**: CFG passed to taint analyzer to generate dependency graphs
  - Dependency graphs specify how inputs flow to a sensitive sink
  - If no tainted data flows to the sink, dependency graph is secure
  - Otherwise, dependency graph is tainted and passed to the string analyzer

String Analyzer

- **Purpose**: Perform vulnerability analysis (forward and backward) on tainted dependency graphs
  - Initialize input nodes to DFAs accepting arbitrary strings
  - Propagates input to nodes in topological order
- **The Intersection**: If it's empty, the sink is not vulnerable with respect to the attack pattern
  - Otherwise...
- **Backward Analysis**: Initialize sink node to DFA that accepts the language of the intersection
  - Propagates data to nodes in reverse topological order
  - At completion, vulnerability signatures are generated for each input node
String Analysis for x86 Binaries

String Manipulation Library

Credit: [STRANGER: An Automata-based String Analysis Tool for PHP]

Purpose
Handles all core string and automata operations
- Replacement, concatenation, union, intersection, etc.
- All operations during vulnerability analysis are sent to SML.
SML executes operation then returns the result back as an automaton.

Experiments and Conclusions

Case Study
Target: SimpGB-1.49.0, a PHP guestbook application
- 153 PHP files
- 44000+ lines of code
System: Intel Core 2 Due 2.5 GHz with 4GB of memory, Ubuntu 8.04

Results
- 231 minutes to check XSS vulnerabilities
  - 304 possible vulnerabilities
- 175 minutes to check SQL vulnerabilities
  - 172 possible vulnerabilities
- 151 minutes to check MFE vulnerabilities
  - 26 possible vulnerabilities

Outline

1. STRANGER
2. String Analysis for x86 Binaries

About

Authors present a static-analysis technique for recovering possible string values in an executable program, when no debug information or source code is available
- The result is a regular language
- A superset of all string values at given program point

x86sa
Introduce a tool, x86sa, a practical implementation to analyze x86 binaries. Built on top of IDA.
Why care about strings?
Aid in understanding of a program
- Contain significant high-level information about the program
- Understand undocumented functionality
- Renovating, upgrading or replacing legacy code
- Analysis of malicious code

Difficulties
A program in executable format retains little detail beyond the actual semantics of the original source code

The approach
- Build on top of existing work
- Java String Analyzer (JSA) infrastructure
The major work is the identification of strings and operations applied to them

Analysis Techniques
- Analyze a single function in a program and construct a string flow graph for C-style strings. To do this, perform 2 analysis techniques:
  - String-inference analysis
  - Stack-height analysis

String-inference analysis
- Locate possible C-style strings
- Basically doing type inference, on only the string type

Assumptions
- Strings in the x86 executable are C-style, null terminated
- All string operations use the libc string functions
**String-Inference Analysis**

Using function signatures for libc string functions, string inference can be performed. Type information is then propagated backwards through the CFG.

**Stack-Height Analysis**

**Purpose**
- Used to associate PUSH instruction with a function's argument

**Assumptions**
- The x86 executable uses the _cdecl calling convention
- Function arguments are passed from the caller to the callee using the stack

**How**
- A forward analysis is performed to locate the "cleanup" instruction
- Divide the cleanup size by x86 word size, gives the number of arguments
- This knowledge is then passed to the inference step (previous)

**Construction of the String Flow Graph**

Forward traversal of the CSG used to translate the previous information into the string flow graph:
- At creation points (strcat, malloc, etc.), an appropriate flow graph node is created (INIT, CONCAT)
- A source operand to a libc function is assigned an ASSIGN node
The C program in the previous slide generates the string \texttt{c^{100}}

After analysis, the string flow graph (previous slide) is fed to the JSA backend

JSA returns the regular expression: \texttt{c^{*}}

Safe approximation, the loop is not interpreted

The Lion Worn

Uses Linux rootkit, t0rn

One of the components is a wrapper around the \texttt{crypt} function

An interesting string is found by x86sa right before execve

'strings' does not recognize this, because the string is built at runtime, byte at a time with calls to strcat

Future Work

Provide cross-function analysis

Provide more than C-tyle strings, such as Unicode

Remove the limit of only libc modified strings