boa

Buffer Overrun Analyzer

Edo Cohen	Tzafrir Rehan	Gai Shaked
039374814	039811880	036567055
sedoc@t2	tzafrir@cs	gai@tx

March 1, 2011

Chapter 1

Introduction

1.1 Goal

Given a C program that performs buffer manipulations, statically (at compile time) identify whether the program may perform array access out of the array bounds.

1.2 Previous work

Chapter 2

boa

2.1 Overview

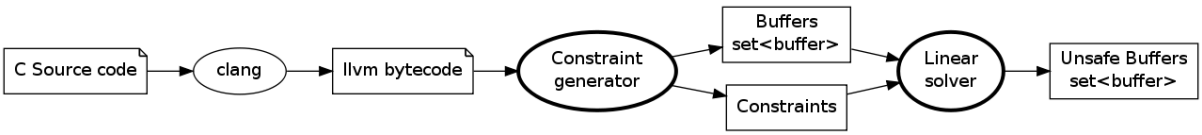

Figure 2.1: Main components and stages

2.2 Constraint Generator

2.2.1 Integers

2.2.2 Direct array access

```
1  char buf[10];
2  buf[10] = 'a';
```

2.2.3 String manipulation functions

```
1 #include "string.h"
2
3 int main() {
4   char *str1 = "longer than ten", *str2 = "short";
5   char buf1[10], buf2[10];
6   strcpy(buf1, str1);
7   strcpy(buf2, str2);
8 }
```

2.2.4 Buffer aliasing

2.3 Linear Solver

2.3.1 GLPK

2.3.2 Elastic filter

2.3.3 Blame system

2.4 Implementation

Chapter 3

-To be named-

3.1 Test system

3.2 Version control

3.2.1 Code reviews

Chapter 4

Results

We tested boa on several widespread real world programs. We tested to see whether boa discover real buffer overruns, and also to evaluate the number of false alarms and their main causes. The source files used in all of the experiments is availble in boa git repository[2].

Table 4.1 summerize the performence of boa on the programs we present in this document, the reported runnig times are the results of experiments on Dell vostro 1310 laptop, with Intel Core2 Duo CPU T8100 2.10GHz and 2GB RAM running Debian GNU/Linux Wheezy (7.0.0), clang 2.9, llvm 2.9 and GLPK 4.43. On this humble configuration boa can analyze few thouthends lines of code within seconds, thus the use of elastic filter did pay off and boa can be used to efficiently analyze any reasonable piece of C code.

	fingerd	flex	syslog
Source lines	230		332
Constraints	2894		1206
Running time	2.508s		1.304s
Buffers	34		15
Overruns reported	6		8
Real overruns	1		1

Table 4.1: boa performence on various real world examples

4.1 fingerd

We tested boa using *fingerd*, unix finger deamon. We altered the current source code to reflect the well known buffer overrun, used by the Internet worm in 1988. The overrun is caused by using the unsafe function *gets* to read data into the 1024[1] bytes buffer *line*. As far as we know, this is the only real buffer overrun in the 230 lines source code.

Running on *fingerd* source, boa report overruns on 6 out of the 34 buffers. Next we present boa's blame for each of them, and analyze the reason for the reported overrun -

line is the only real overrun in *fingerd*

line tests/realworld/fingerd/fingerd.c:85
- unsafe function call gets [tests/realworld/fingerd/fingerd.c:121]
- unknown function call realhostname_sa [tests/realworld/fingerd/fingerd.c:128]
- memchr call might read beyond the buffer [tests/realworld/fingerd/fingerd.c:139]
[... 10 more lines ...]

The overrun discovered by boa, and the real cause reported briefly.

[1] Back in 1988 *line* was 512 bytes, but it does not matter for the analysis.

rhost is a char buffer ment to hold the host name

rhost tests/realworld/fingerd/fingerd.c:86
- unknown function call realhostname_sa [tests/realworld/fingerd/fingerd.c:128]
- unknown function call realhostname_sa [tests/realworld/fingerd/fingerd.c:128]
- buffer alias with offset [tests/realworld/fingerd/fingerd.c:128]
- buffer alias with offset [tests/realworld/fingerd/fingerd.c:128]

4.2 flex

Bibliography

[1] Buffer Overrun Detection using Linear Programming and Static Analysis

[2] boa git repository - https://github.com/tzafrir/boa/