Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Michał Ziemba
Nr albumu: 237954

Serwis Submit++

Praca magisterska
na kierunku INFORMATYKA

Praca wykonana pod kierunkiem
dr Aleksy Schubert
Instytut Informatyki

Czerwiec 2012

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data Podpis kierującego pracą

Oświadczenie autora (autorów) pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data Podpis autora (autorów) pracy

Streszczenie

W pracy przedstawiono opis i dokumentację narzędzia JBSC, służącego do analizy statycznej bajtkodu Javy. Jest to wtyczka do środowiska programistycznego Eclipse, korzystająca z frameworku Umbra. Umożliwia proste wykrywanie potencjalnie niebezpiecznych konstrukcji w kodzie bez jego uruchamiania.

Słowa kluczowe

Serwis, zadanie, rozwiązanie, MIMUW

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.3 Informatyka

Klasyfikacja tematyczna

D. Software

Tytuł pracy w języku angielskim

The Submit++ service

Spis treści

Wprowadzenie

Analiza statyczna kodu polega na badaniu własności programu bez jego uruchamiania, w przecwieństwie do analizy dynamicznej.

Motywacja Ogólnie pojęta analiza statyczna ma coraz większe zastosowanie w procesie wytwarzania wolnego od błędów i bezpiecznego kodu. Analiza statyczna kodu wykonywalnego ma zastosowanie w systemach osadzonych, gdzie nie ma dostępu do źródeł. Także w przypadku systemów których źródła nie są po prostu wolnodostępne.

Dłuższy opis działania Java Bytecode Static Checker jest wtyczką do Eclipsa

Rozdział 1

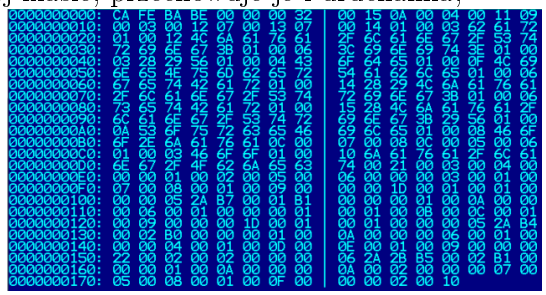
Podstawowe pojęcia

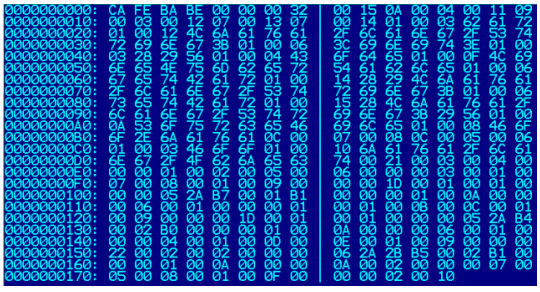
Rozdział 2

Kod bajtowy

2.1. Wprowadzenie

Kod bajtowy (*Java Byte Code*) jest formą instrukcji, wykonywanych przez maszynę wirtualną Javy. Programy napisane w języku Java są kompilowane do przenośnego kodu, przeważnie w postaci plików *.class*. Zadaniem maszyny wirtualnej jest dostarczenie środowiska, umożliwiającego uruchamianie takiego kodu. Maszyna wirtualna, wczytując plik zawierający ciąg bajtów, dostaje kod bajtowy dla wszysztkich metod w danej klasie, przechowuje je i uruchamia, gdy konkretna metoda tej klasy zostanie wywołana. r8cm Selbstgebaute „Bluesniper“ um Bluetooth-Geräte aus über 1 km Entfernung anzugreifen. (Stand: 2004)





Każdy kod instrukcji (ang. *opcode*) ma długość jednego bajta, przy czym niektóre instrukcje wymagają parametrów. Każdemu kodowi odpowiada pewne znaczenie (*mnemonic*), przypisane w specyfikacji maszyny wirtualnej (zob. [JVMSpec])

Na przykład dla ciągu bajtów: 03 3b 84 00 01 1a 05 68 3b a7 ff f9 Otrzymujemy po deassemblacji:

mnemonic	corresponding codes
iconst_0	03
istore_0	3b
iinc 0, 1	84 00 01
iload_0	1a
iconst_2	05
imul	68
istore_0	3b
goto −7	a7 ff f9

Ponieważ maszyna wirualna Javy nie posiada rejestrów, większość operacji używa stosu do przechowywania wartości.

2.2. Opis bajtkodu

Specjalne i zarezerwowane instrukcje:

-
-

Instrukcje kodu bajtowego można podzielić na następujące grupy:

-

Wyróżniamy podstawowe typy, na których operują instrukcje:

oznaczenie	typ	definicja
b	byte	one-byte signed two's complement integer
si	short	two-byte signed two's complement integer
i	int	4-byte signed two's complement integer
l	long	8-byte signed two's complement integer
f	float	4-byte IEEE 754 single-precision float
d	double	8-byte IEEE 754 double-precision float
c	char	2-byte unsigned Unicode character

Każda instrukcja określa, jakiego typu jest argument, zatem sama maszyna wirtualna nie musi znać typów samych wartości, po prostu przekazywane one są jako ciąg bajtów, w porządku *big-endian*, na przykład: ciąg bajtów: 17 01 00 odpowiada: sipush 256; // 17 01 00

2.3. Struktura pliku *.class*

Constants pool

2.4. Przykład generowania kodu bajtowego

Dla prostego programu w języku Java:

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

kod bajtowy wygenerowany przy pomocy deasemblacji (polecenie $java - d$) wygląda następująco:

```
    Compiled from "HelloWorld.java"
public class HelloWorld extends java.lang.Object
 SourceFile: "HelloWorld.java"
 minor version: 0
 major version: 50

{
public HelloWorld();
 Code:
  Stack=1, Locals=1, Args_size=1
  0: aload_0
  1: invokespecial #1; //Method java/lang/Object."<init>":()V
  4: return

 LineNumberTable:
  line 1: 0

public static void main(java.lang.String[]);
 Code:
  Stack=2, Locals=1, Args_size=1
  0: getstatic #2; //Field java/lang/System.out:Ljava/io/PrintStream;
  3: ldc #3; //String Hello, world!
  5: invokevirtual #4; //Method java/io/PrintStream.println:(Ljava/lang/String;)V
  8: return
 LineNumberTable:
  line 5: 0
  line 6: 8
}
```

Rozdział 3

Analiza statyczna

Analiza statyczna polega na badaniu kodu źródłowego bez jego uruchamiania. Pozwala wykryć potencjalnie niebezpieczne konstrukcje w kodzie. Jednak fakt niewykrycia błędu na poziomie analizy statycznej nie czyni kodu bezbłędnym.

Możliwości i ograniczenia (zalety) Zalety korzystania z narzędzi do analizy statycznej:

- Sprawdzenia wykonywane przez narzędzie, w odróżnieniu od sprawdzenia programisty, pozwalają wykryć potencjalne błędy w miejscach mniej interesujących, które mogły zostać pominięte lub przeoczone przez programistę.
- Analiza statyczna daje możliwość wykrycia problemów u ich źródła. Pozwala uniknąć sytuacji, kiedy w trakcie uruchamiania programu otrzymujemy błąd przepełnienia buforu, a nie wiemy dokładnie kiedy, gdzie i dlaczego to przepełnienie nastąpiło.
- Błędy są znajdowane na wczesnym etapie rozwijania kodu. To daje możliwość uniknięcia być może kosztownego usuwania tego błędu na późniejszym etapie.

Najpoważniejszym zarzutem, który pojawia się pod adresem narzędzi do analizy statycznej jest fakt, że produkują zbyt wiele ostrzeżeń, z których większość jest nieprzydatnych. Takie fałszywe ostrzeżenia nazywamy *false negatives*.

Zakres analizy statycznej Przy pomocy analizy statycznej można wykonywać następujące sprawdzenia:

-

Etapy analizy
Co można sprawdzać
False positives, false negatives
Przykładowe narzędzia i ch krótki opis:

- lint
- findbugs
- escjava2

Rozdział 4

Integracja

Co to jest Umbra?
    Połączenie z frameworkiem Umbra

Rozdział 5

Funkcjonalność systemu

5.1. Wprowadzenie

Rozdział 6

Środowisko systemu

6.1. Serwer

Rozdział 7

Budowa systemu

7.1. Architektura systemu

Rozdział 8

Podsumowanie

Dodatek A

Listingi kodu bajtowego

Dodatek B

Dostarczone dokumenty

Obiekty dostarczone w ramach pracy licencjackiej to:

- dokument z treścią pracy licencjackiej
- płyta CD

Dodatek C

Spis zawartości płyty CD

Bibliografia

[JVMSpec] "http://docs.oracle.com/javase/specs/jvms/se7/html/index.html".

[Fif01] Filigran Fifak, *O fetorach* σ-ρ, Acta Fetorica, 2001.