Kianxali: Kieler Analyzer for Executables and Libraries

Folke Will <fwi@informatik.uni-kiel.de>

Institut für Informatik, Christian-Albrechts-Universität zu Kiel

28.04.2014

Motivation: Analyse von Programmen ohne Quellcode

Situationen ohne Quellcode

Wissenschaft und Hobby

- Schadcode-Analyse
- Linux-Treiber entwickeln
- freie Software für proprietäre Formate entwickeln
- Entfernung von Kopierschutz aus Software (Cracking)

Wirtschaft

- Security-Audits
- Industriespionage

Problemstellung

Analyse von Programmen, ohne deren Quellcode zu kennen

Welche Möglichkeiten der Analyse gibt es?

Black-Box-Analyse: Rückschlüsse aus dem Laufzeitverhalten

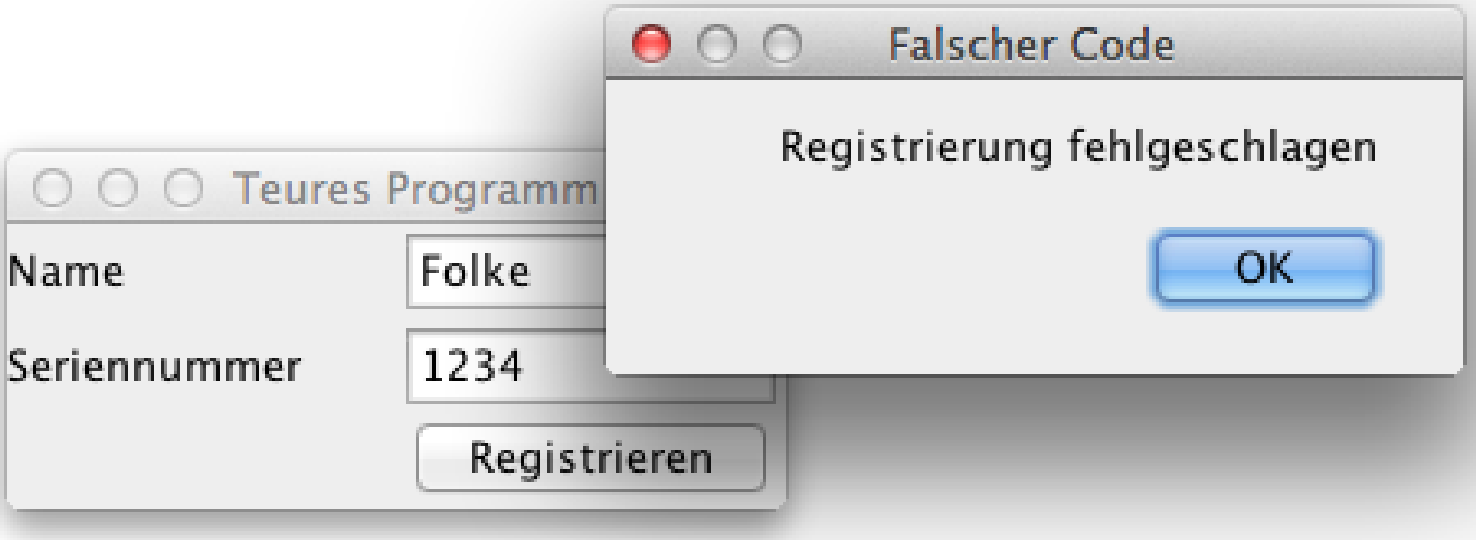
- Beobachtung der Ausführung
- Analyse von Dateiformaten

White-Box-Analyse: Rückschlüsse aus dem Maschinencode

- Programm disassemblieren
- Kontrollfluss ermitteln
- Kontrollfluss testweise manipulieren

Oberbegriff: *reverse engineering*

Black-Box-Analyse



Erkenntnis

Erkenntnis: Programm zeigt Fehlermeldung bei falschem Code

White-Box-Analyse

```
    push ebx                    ; Parameter der Funktion 200
    push ecx                    ; Parameter der Funktion 200
    call 200                    ; Funktionsaufruf
    cmp eax, 0                  ; Pruefen, ob Ergebnis == 0
    jne label1                  ; Falls Nein: Sprung
    push offset "Vielen Dank"
    push offset "Registrierung erfolgreich"
    call 400                    ; Funktionsaufruf
    call 500                    ; Funktionsaufruf
    ret                         ; Ende der Prozedur
label1:
    push offset "Falscher Code"
    push offset "Registrierung fehlgeschlagen"
    call 400                    ; Funktionsaufruf
    ret                         ; Ende der Prozedur
```

White-Box-Analyse

Erkenntnis

Kontrollfluss eines Programms ist sichtbar und manipulierbar

Kianxali?

Was ist Kianxali?

Kianxali ist...

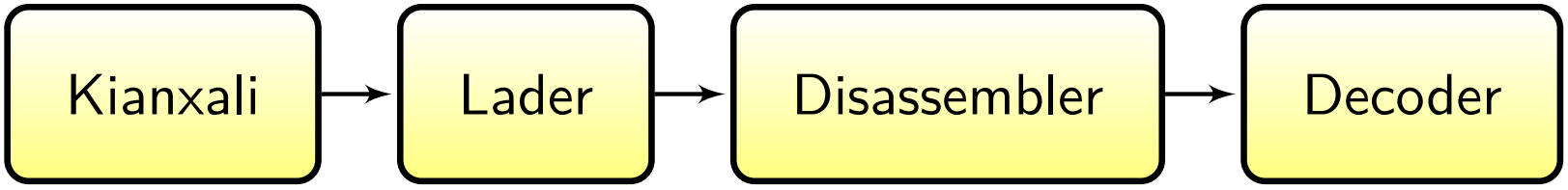
- ein Disassembler
- interaktiv
- skriptbar

Merkmale

- geschrieben in Java
- unterstützt x86- und x64-Opcodes
- verarbeitet PE, ELF und Mach-O

Komponenten

Aufteilung der Aufgaben in Komponenten



Wesentliche Komponenten Kianxalis

- Lader: lädt Programme im Binärformat
- Decodierer: decodiert einzelne Instruktionen
- Disassembler: analysiert Programm als Ganzes

Lader: Simuliert Betriebssystemlader

Komponente: Lader

Aufgaben des Laders

- Laden der Meta-Informationen
- Bestimmung des Einstiegspunkts
- Importe aus Bibliotheken nachvollziehen
- Umrechnung von Speicher- und Dateiadressen

Unterstützte Formate

- PE (EXE- und DLL-Dateien unter Windows)
- ELF (Linux)
- Mach-O (OS X) und *fat binaries*

Decodierer: Aufgaben

Komponente: Decodierer.

Aufgaben des Decodierers

- Decodieren eines einzelnen Befehls
- Decodieren der Operanden
- Decodieren von Daten (z.B. Zeichenketten)

```
... | 90 EB FE ... → nop
... 90 | EB FE ... → jmp 0
```

Einfachste Schnittstelle, schwierigste Umsetzung.

Decodierer: x86-Architektur

x86-Architektur

Eigenschaften

- Ursprung: Intel 8086 aus 1978 mit 133 Instruktionen
- Heute: mehr als 700 Instruktionen (Intel Core i7)
- i7 ist abwärtskompatibel zum 8086
- dadurch 42 Typen von Operanden
- und 32 Arten zur Codierung
- sauber und vollständig spezifiziert

Decodierer: x86-Architektur

x86-Architektur

Eigenschaften

- Ursprung: Intel 8086 aus 1978 mit 133 Instruktionen
- Heute: mehr als 700 Instruktionen (Intel Core i7)
- i7 ist abwärtskompatibel zum 8086
- dadurch 42 Typen von Operanden
- und 32 Arten zur Codierung
- sauber und vollständig spezifiziert

Probleme

- verfügbare Decoder liefern nicht genug Informationen
- Spezifikation hat 6.000 Seiten

Decodierer: Beispiel für Decodierung

Einfacher Befehl

```
8B C2 mov eax, edx
```

8B: mov-Instruktion
C2: codierte Operanden

Befehl mit Präfix

```
66 8B C2 mov ax, dx
```

66: Operand-Size-Prefix
8B: mov-Instruktion
C2: codierte Operanden

Decodierer: Beispiel für Decodierung

Scheinbar einfacher Befehl

```
0F 6F C1 movq mm0, mm1
```

0F 6F: movq-Instruktion
C1: codierter Operand

Anderer Befehl

```
66 0F 6F C1 movdqa xmm0, xmm1
```

66 0F 6F: movdqa-Instruktion

Mit Präfix

```
44 66 0F 6F C1 movdqa xmm8, xmm1
66 44 0F 6F C1 movdqa xmm8, xmm1
```

44: Register-Extension-Prefix

Decodierer: Nochmal genauer

66 44 0F 6F C1=

01100110 01000100 00001111 01101111 11000001

Legende:

Präfix

Operand

Instruktion

Dies ist nur einer von vielen Stolpersteinen

Weitere

- Teile der Instruktion im Operandenfeld
- Teile des Operanden im Instruktionsfeld
- Hartcodierte Operanden
- ...

Decodierer: Lösungsidee

Probleme für die Datenstruktur

- Präfix-Byte ist mal Präfix, mal Teil der Instruktion
- Tabellen und Bäume fallen weg
- Zustandsautomat wäre zu groß

Idee inspiriert durch LR-Parsing

- lege mögliche Präfix-Bytes auf Stack
- nutze dort Zustandsautomaten
- sobald Opcode beginnt, nutze Baum und Zustand
- Blätter können mehrere Instruktionen entscheiden
- endgültige Entscheidung erst nach Operanden (push, test)

Aufgaben des Disassemblers

Komponente: Disassembler.

Aufgaben des Disassemblers

- Analyse und Folgen des Programmflusses
- Erkennung von Prozeduren
- Bereitstellung einer Datenstruktur mit Kontrollflussgraph

1. Ansatz: Linear Sweep

Disassembler-Strategie: Linear Sweep

Linear Sweep

- decodiere sequentiell alle Instruktionen ab Einstiegspunkt

Nachteil

Lässt sich von Schadcode leicht austricksen:

```
4000h       jmp 4010h
4004h       ; Bytes, die keinem Opcode entsprechen
4010h       ...
```

Disassembler analysiert die Bytes und schlägt fehl.

2. Ansatz: Recursive Traversal

Disassembler-Strategie: Recursive-Traversal

Recursive Traversal

toVisit $\leftarrow \{\text{programEntryPoint}\}$
while toVisit $\neq \emptyset$ **do**
    address $\leftarrow$ takeFromSet(toVisit)
    instruction $\leftarrow$ decodeAt(address)
    if instruction has branch addresses **then**
        toVisit $\leftarrow$ toVisit $\cup$ branch addresses
    end if
    if execution flow continues after instruction **then**
        toVisit $\leftarrow$ toVisit $\cup\{$address + size(instruction)$\}$
    end if
end while

Nachteil

Erreicht keinen indirekt angesprungenen Code.

Erweiterungen

Kianxali erweitert den Ansatz des Recursive-Traversal

Erweiterungen

- Decodierung von Daten
- Aufdeckung von indirekt aufgerufenen Prozeduren
- Aufdeckung von Sprungtabellen (switch-case-Anweisung)

Datenstruktur

- Repräsentation des virtuellen Speicherlayouts in RS-Baum
- Kontrollflussgraph: Querverweise zwischen Knoten

Demonstration

(Demonstration des Projekts)