Formale Methoden im Reverse Engineering

Sebastian Porst (sebastian.porst@zynamics.com)
TU Dortmund – Januar 2010

Ich

- Master of Science Informatik, FH Trier 2007
- Softwareentwickler für Reverse Engineering Tools bei der zynamics GmbH
- Fast 12 Jahre Reverse Engineering Erfahrung
- Zwei Jahre Vortragserfahrung über Reverse Engineering (CanSecWest, Hack in the Box, SOURCE Barcelona, …)

Meine Motivation

- Unzufrieden mit Fortschritten beim Reverse Engineering in den letzten 10 Jahren
- Möchte akademische Forschung mit industrieller Anwendung verknüpfen
- Studenten für Reverse Engineering begeistern

zynamics

- IT-Sicherheitsfirma aus Bochum
- Spezialisiert auf das Entwickeln von RE Tools
- 5 Jahre alt
- 12 Mitarbeiter
- Deutscher IT-Sicherheitspreis 2006

Übersicht

- Diskussion der Problematik
- Abstrakte Interpretation
- Metasprachen
- Dynamische Instrumentierung
- Taint Tracking
- BitBlaze

Reverse Engineering

- Analyse von Binärcode ohne Zuhilfenahme von Quellcode
- Ermitteln der Programmfunktionalität oder bestimmter Eigenschaften des Binärcodes
- Optionale Rückübersetzung in Hochsprachencode

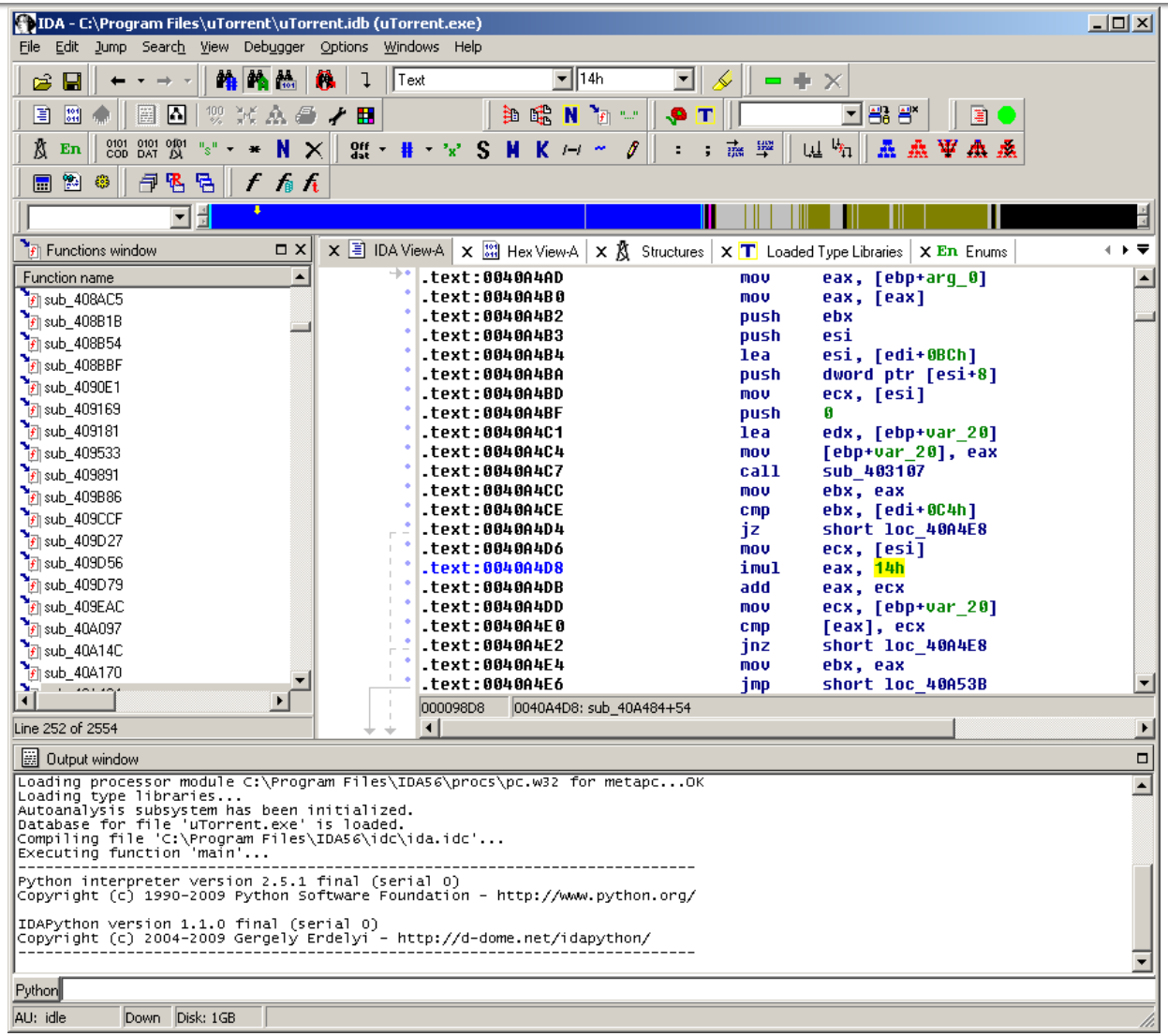
Anwendungen

- Schwachstellenermittlung
- Malwareanalyse
- Behebung von Kompatibilitätsproblemen
- Überprüfung von Code auf Vertrauenswürdigkeit
- Entdeckung von Codediebstahl

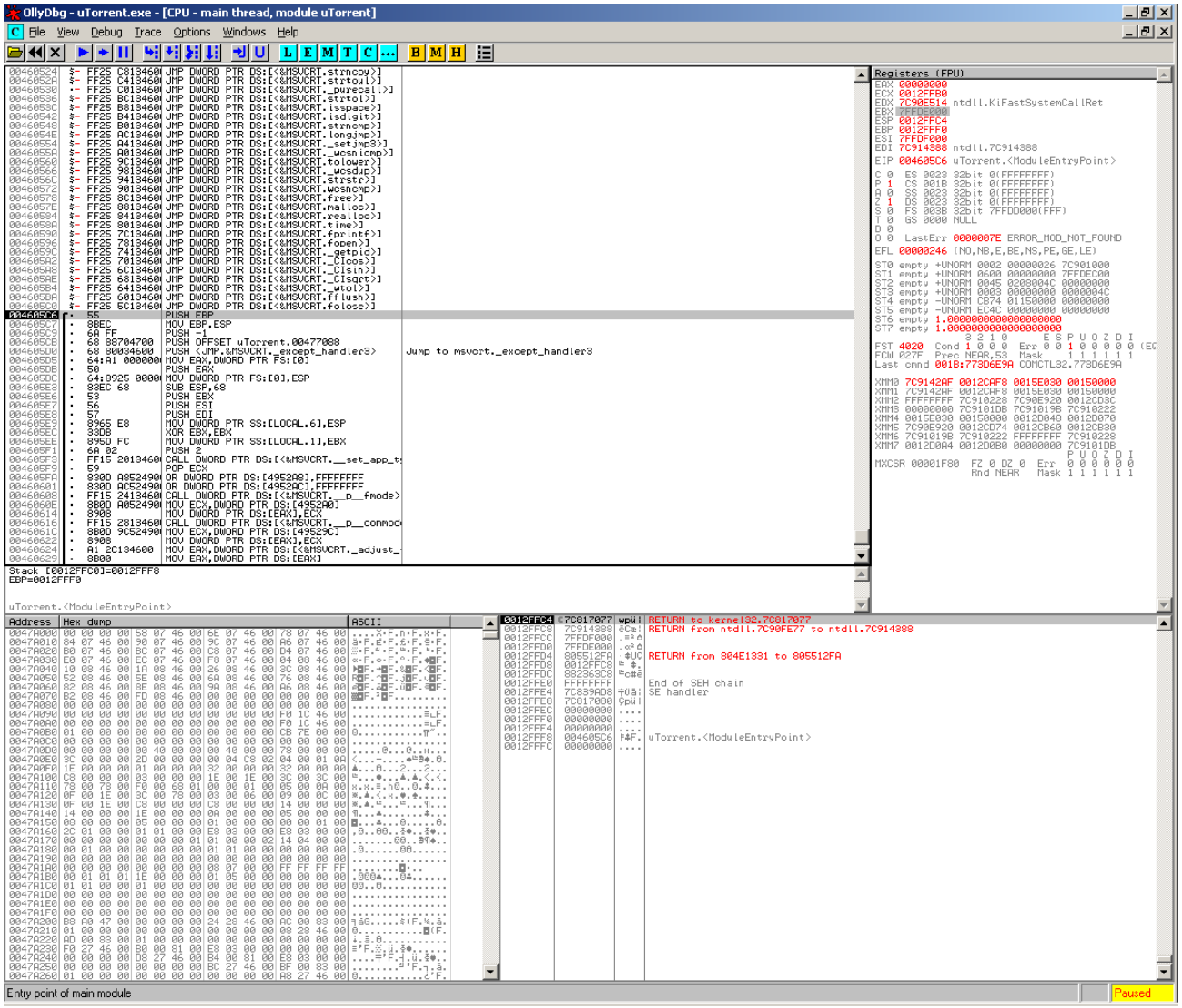
Aktuelle RE Tools

- Disassembler (IDA Pro)
- Debugger (WinDBG, OllyDbg)
- Hex-Editoren
- Fuzzer
- Dateianalyseprogramme
- Entpacker

IDA Pro


IDA - C:\Program Files\uTorrent\uTorrent.idb (uTorrent.exe)
File Edit Jump Search View Debugger Options Windows Help
Text
14h
Functions window
Function name
sub_408AC5
sub_408B1B
sub_408B54
sub_408BBF
sub_4090E1
sub_409169
sub_409181
sub_409533
sub_409891
sub_409B86
sub_409CCF
sub_409D27
sub_409D56
sub_409D79
sub_409EAC
sub_40A097
sub_40A14C
sub_40A170
Line 252 of 2554
IDA View-A
Hex View-A
Structures
Loaded Type Libraries
Enums
.text:0040A4AD mov eax, [ebp+arg_0]
.text:0040A4B0 mov eax, [eax]
.text:0040A4B2 push ebx
.text:0040A4B3 push esi
.text:0040A4B4 lea esi, [edi+0BCh]
.text:0040A4BA push dword ptr [esi+8]
.text:0040A4BD mov ecx, [esi]
.text:0040A4BF push 0
.text:0040A4C1 lea edx, [ebp+var_20]
.text:0040A4C4 mov [ebp+var_20], eax
.text:0040A4C7 call sub_403107
.text:0040A4CC mov ebx, eax
.text:0040A4CE cmp ebx, [edi+0C4h]
.text:0040A4D4 jz short loc_40A4E8
.text:0040A4D6 mov ecx, [esi]
.text:0040A4D8 imul eax, 14h
.text:0040A4DB add eax, ecx
.text:0040A4DD mov ecx, [ebp+var_20]
.text:0040A4E0 cmp [eax], ecx
.text:0040A4E2 jnz short loc_40A4E8
.text:0040A4E4 mov ebx, eax
.text:0040A4E6 jmp short loc_40A53B
000098D8
0040A4D8: sub_40A484+54
Output window
Loading processor module C:\Program Files\IDA56\procs\pc.w32 for metapc...OK
Loading type libraries...
Autoanalysis subsystem has been initialized.
Database for file 'uTorrent.exe' is loaded.
Compiling file 'C:\Program Files\IDA56\idc\ida.idc'...
Executing function 'main'...
Python interpreter version 2.5.1 final (serial 0)
Copyright (c) 1990-2009 Python Software Foundation - http://www.python.org/
IDAPython version 1.1.0 final (serial 0)
Copyright (c) 2004-2009 Gergely Erdelyi - http://d-dome.net/idapython/
Python
AU: idle
Down
Disk: 1GB

OllyDbg


OllyDbg - uTorrent.exe - [CPU - main thread, module uTorrent]
File View Debug Trace Options Windows Help
Registers (FPU)
Entry point of main module
Paused

Aktuelle Vorgehensweise

- Mühsame Handarbeit
  - Finden interessanter Stellen
  - Untersuchung interessanter Stellen
- Kaum Automatisierung
- Kaum Nutzung von Erkenntnissen aus der Informatik
- Keine Standardisierung

Problematik

- Software wird immer größer und komplexer
- Immer mehr Plattformen
  - Heimcomputer, Handys, PDAs, Router, Wireless
- Mehr schützenswerte Daten
- Sicherheitsanalysen werden teuer und aufwändiger

Lösungsansätze

- Neue Tools welche mit den Datenmengen besser umgehen
- Plattformunabhängige Tools
- Übernahme von formalen Methoden aus der Informatik

Formale Methoden

- Abstrakte Interpretation
- Metasprachen
- Dynamische Instrumentierung
- Taint Tracking
- BitBlaze

Teil 1: Statische Verfahren

- Codeanalyse ohne Programmausführung
- Vorteile:
  - Unabhängig von ausgeführtem Code
  - Keine Ausführung von gefährlichem Code
- Nachteile:
  - Statische Unentscheidbarkeit

Abstrakte Interpretation

- Entwickelt von Patrick Cousot und Radhia Cousot in den 1970ern
- Teilabschnitt der Programmanalyse
- Statisches Analyseverfahren basierend auf Verbandstheorie
- Theorie der Approximation mathematischer Strukturen in semantischen Modellen

Ziele der abstrakten Interpretation

- Aussagen treffen über Programme ohne die Programme auszuführen
- Verifikation von Behauptungen oder Entdeckung von Eigenschaften
- Reduktion der Komplexität auf das Wesentliche

Semantik einer Sprache

- Annahme der Existenz einer analysierten Programmiersprache mit Wertemenge V
- Teilmenge von V beschreibt zu jedem Zeitpunkt den Programmzustand
- Semantik der Sprache definiert Wertübergänge

$$p : v_1 \rightarrow v_2$$

Komplexitätsprobleme

- Anzahl der konkreten Werte wächst explosionsartig
- Einzelner 32 bit Ganzzahlwert hat 4294967295 Zustände
- Zustandsmenge mehrerer Variablen wächst multiplikativ

Abstraktion von V

- Ermittlung des Gesamtzustands im allgemeinen zu aufwändig und nicht nötig
- Reduktion der Werte aus V auf abstrakte Eigenschaften L

$$p : l_1 \rightarrow l_2$$

- Konkrete Gestalt der Eigenschaften ist abhängig von der gewünschten Analyse

Beispiele für Eigenschaften L

- Eine Variable ist null oder nicht null
- Wertebereich einer Variablen als Intervall
- Variable hat einen Einfluss auf den späteren Programmzustand

Beispiele für Algorithmen

- Verifikation eines sicheren Zustandes
- Konstantenpropagation
- Ermittlung benötigter Register
- Ermittlung der potentiellen Werteintervalle von Variablen

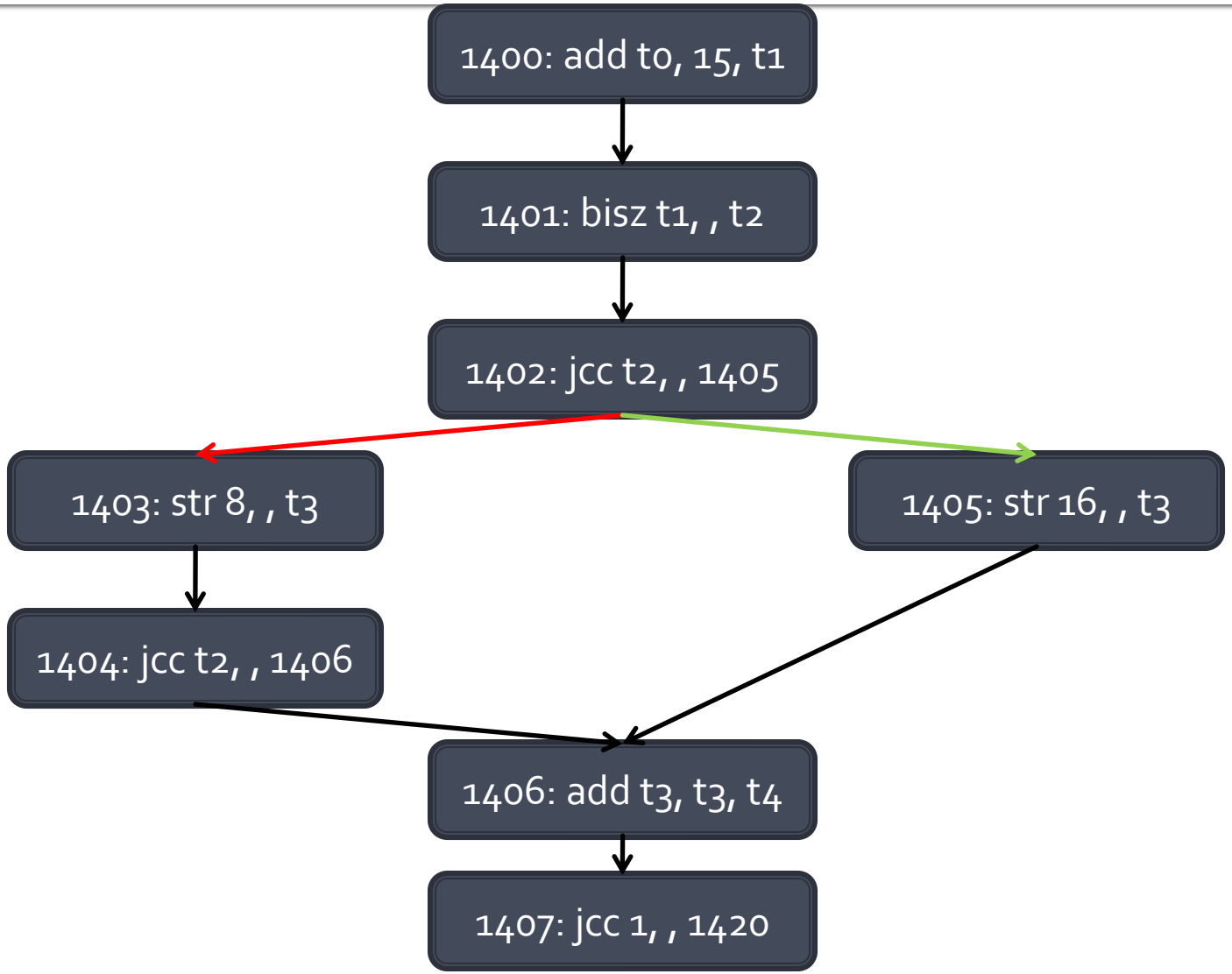
Allgemeines Vorgehen

- Definition der Eigenschaftsübergänge für alle Werteübergänge der Sprache P
- Abarbeiten des zu analysierenden Programms
- Für jeden Werteübergang einen Eigenschaftsübergang ausführen
- Fixpunktiteration durchführen bis keine weiteren Erkenntnisse gefunden werden

Schritt 1: Zerlegung des Codes

- Zu analysierender Code wird in gerichteten Graph zerlegt
- Knoten enthalten einzelne Werteübergänge
- Kanten beschreiben möglichen Programmablauf zwischen Werteübergängen

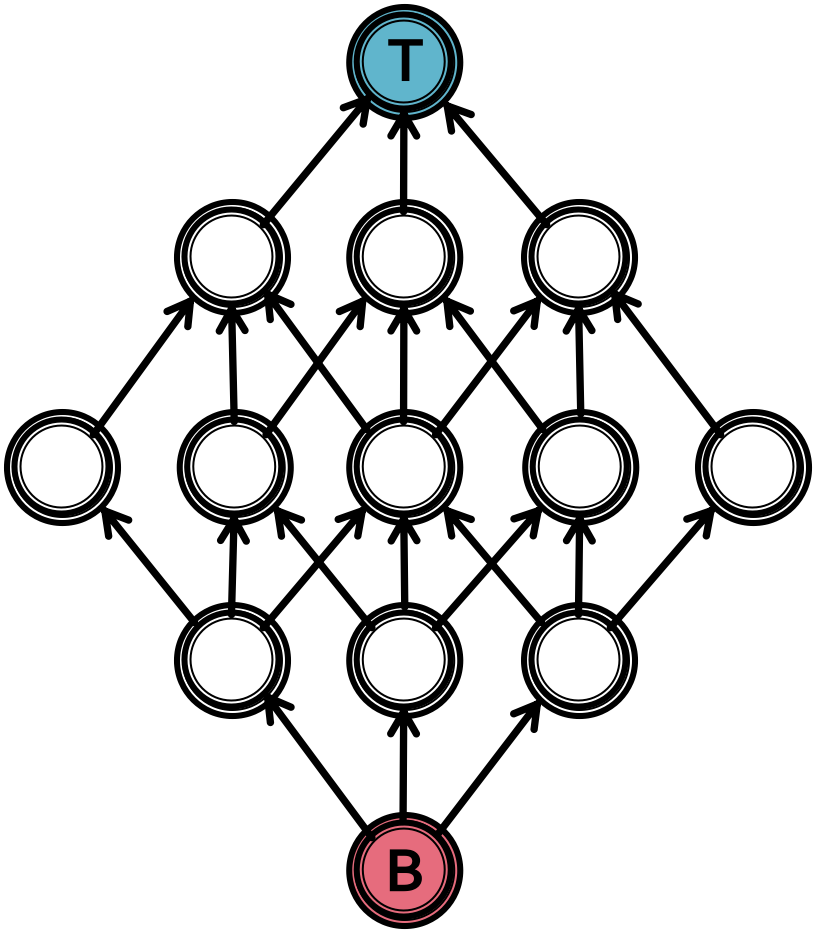
Programmablaufgraph



Schritt 2: Definition von L

- Definition der Eigenschaften L als Verband
- Definition von Infimum (Bottom) und Supremum (Top)
- Definition von monotonen Eigenschaftsübergängen anhand von Werteübergängen
- Definition eines Vereinigungsoperators

Struktur eines Verbands



Schritt 3: Initialisierung von L

- Jedem Knoten des Programmgraphen wird ein Element aus L zugewiesen
- Das Element aus L beschreibt bekanntes Wissen über das Programm an der Stelle des Knotens
- Ohne Vorkenntnisse: Bottom
- Mit Vorkenntnissen: Beliebiges Element

Schritt 4: Fixpunktiteration

- Durchführung von Eigenschaftsübergängen auf Basis der Werteübergänge
- Entweder in Programmablaufrichtung oder umgekehrt
- Vereinigung von Eigenschaften wenn Kontrollfluss zusammenläuft
- Iteration durchführen bis keine weiteren Informationen gefunden werden

Ergebnis interpretieren

- Ergebnis ist Menge der abstrakten Programmzustände
  - = Menge der Verbandselemente an den Knoten

Weiteres

- Terminierung garantiert durch Verbandsordnung und Monotonie
- Realität ist komplizierter (widening, narrowing, …)
- Laufzeitintensiv

Probleme für RE

- Kein Quelltext vorhanden
- Analyse von Assemblersprachen ist komplex aufgrund der großen Befehlsmenge
  - Definition von Eigenschaftsübergängen für Hunderte von Befehlen
- Lösung: Metasprachen

Metasprachen

- Pseudo-Assemblersprachen
- Echte Assemblersprachen können einfach in Metasprachen überführt werden
- Metasprachen sind weniger komplex
- Besser geeignet für Abstrakte Interpretation

REIL

- Speziell entwickelt für Reverse Engineering
- Designziel: Einfach zu verstehen und zu benutzen
- Nur 17 verschiedene Befehle
- Sehr reguläre Struktur mit 3 Operanden pro Befehl

REIL Beispielcode

```
40109A00    str         0x14,  , ebx
40109F04    str         0,  , byte SF
4010A100    str         0x14,  , ecx
4010A300    str         10,  , esi
4010A815    str         byte 0,  , byte CF
4010A816    str         byte 0,  , byte OF
4010A817    undef        ,  , byte ZF
4010A818    undef        ,  , byte AF
4010A819    undef        ,  , byte PF
4010A81A    str         qword 0x12C,  , eax
4010A81B    str         qword 0,  , edx
4010AA00    ldm         esp,  , t0
4010AA01    add         esp, 4, qword t1
4010AA02    and         qword t1, qword 0xFFFFFFFF, esp
4010AA03    jcc         1,  , t0
```

Vorteile von REIL

- Plattformunabhängig
- Pro Algorithmus nur 17 Eigenschaftsübergänge
- Einfaches Mapping zwischen Originalcode und REIL Code
- Framework für abstrakte Interpretation bereits verfügbar

MonoREIL

- Framework zur abstrakten Interpretation von REIL Code
- Implementiert in Java
- Benutzbar aus jeder beliebigen JVM-Sprache (Java, Jython, JRuby, …)
- Erleichtert das Schreiben von statischen Analysealgorithmen durch Standardisierung der Struktur

Benutzung

- Eingabe: Echter Assemblercode
- Transformation zu REIL
- Analyse des REIL Codes mit MonoREIL
- Rückbeziehung der Ergebnisse auf den Eingabecode

Teil 2: Dynamische Verfahren

- Codeanalyse durch Programmausführung
- Vorteile
  - Vollständige Kenntnisse über Programmverlauf
- Nachteile
  - Nur ausgeführter Code wird betrachtet
  - Potentiell gefährlicher Code wird ausgeführt

Dynamisches Reverse Engineering

- Zu analysierendes Programm wird echt ausgeführt
- Analyse geschieht entweder in Echtzeit oder im Nachhinein auf protokollierten Daten

Dynamische Instrumentierung

- Idee: Instrumentierungscode wird zur Laufzeit in Programmcode eingeschleust
- Möglichkeiten:
  - Echtzeitüberwachung aller Aspekte des ausgeführten Programms
  - Echtzeitmanipulation des Programmcodes und der Daten

Anwendungen

- Ermittlung von Verzweigungswahrscheinlichkeiten
- Statistiken über Fehler und Ausnahmen
- CPU Profiling
- Code Coverage
- Taint Tracking

Verfügbare Tools

- Pin
  - University of Virginia
  - Aktiv weiterentwickelt von Intel
- DynamoRIO
  - Zusammenarbeit von Hewlett-Packard und MIT

Pin vs. DynamoRIO

- Verschiedene Ansätze
- Auswirkungen auf das Zielprogramm
  - DynamoRIO ist schneller und braucht weniger Speicher
  - DynamoRIO kann Instruktionsstream verändern
- Beide Tools über C-Programme erweiterbar
  - Pin-Erweiterungen sind weniger komplex

Anwendung: Taint Tracking

- Erkenntnis I: Daten sind wichtiger als Code
- Erkenntnis II: Manche Daten sind wichtiger als andere Daten
- Taint Tracking markiert und verfolgt wichtige Daten
  - Beispiel: Unverifizierte Eingaben

Allgemeines Vorgehen

- Initiale Definitionspunkte für wichtige Daten werden ermittelt
- Ursprungsdaten werden markiert
- Immer wenn bereits markierte Daten in Berechnungen verwendet werden müssen auch Berechnungsresultate markiert werden
- Markierung von Registerwerten und Speicherzellen

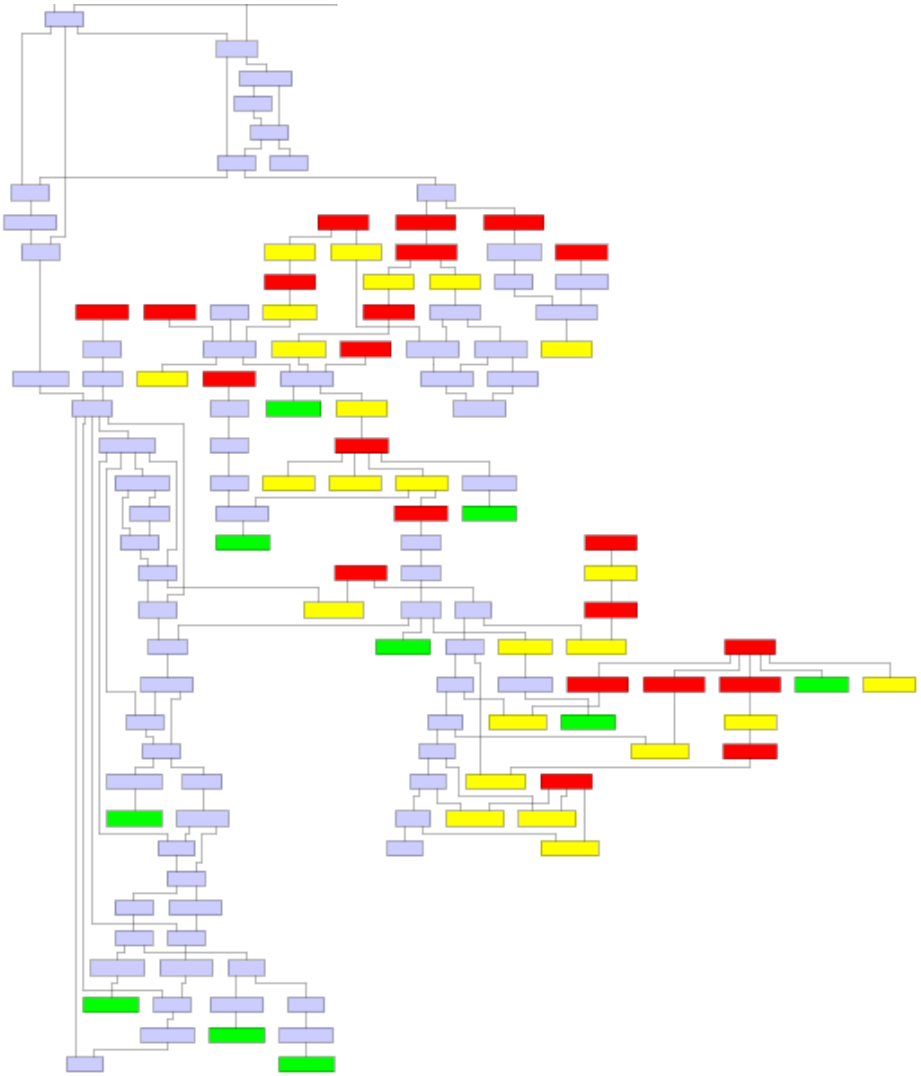
Details

- Markieren von Bits mit Hilfe von Bitmasken erweist sich als wirkungsvoller
- Markieren von Ablaufpfaden als Resultat von bedingten Verzweigungen ist nützlich
- Teilregister müssen besonders betrachtet werden
- Befehle müssen auch getaintet werden

Resultat

- Menge von disjunkten gerichteten Graphen
- Ablesbar wie Daten zusammenhängen
- Ablesbar wann welche Daten einen Einfluss auf das Programm hatten
- Ablesbar welche Befehle auf Grund der verfolgten Daten ausgeführt werden

Ergebnisgraph



Anwendungen

- Feststellen ob eingegebene Daten korrekt entschärft werden
- Feststellen ob zu einem bestimmten Zeitpunkt eingegebene Daten noch einen Einfluss auf das Programm haben
- Überprüfen der Erreichbarkeit einzelner Befehle

Probleme

- Menge an Daten in modernen Programmen bringt Komplexitätsprobleme mit sich
  - Lösung: Konzentration auf Teilmengen der Daten
- Geschwindigkeit und Qualität hängen von der Instrumentierung ab

BitBlaze

- Binary Analysis for Computer Security
- Programmanalyseprojekt der Universität von Berkeley
- Derzeit fortgeschrittenstes RE Framework
- Implementierung in O'Caml

BitBlaze Komponenten

- Vine
  - Komponente zur statischen Analyse
- TEMU
  - Komponente zur dynamischen Analyse
- Rudder
  - Komponente zur symbolischen Ausführung

Vine

- Metasprache zur Simulation von x86 Code
- Ermöglicht die Erstellung von Kontrollflussgraphen
- Stellt Standardalgorithmen zur Programmanalyse bereit
  - Konstantenpropagation
  - Datenabhängigkeiten
- Kann in C-Code übersetzt werden

TEMU

- Erweiterung von QEMU
- Ermöglicht Taint Tracking
- Ermöglicht Verhaltensanalyse vollständiger Programme

Rudder

- Symbolische Ausführung von Code
- Start mit interessanten Registern und Speicherwerten
- Werteveränderungen werden als Formeln dargestellt
- Erlaubt das mathematische Betrachten von Wertänderungen

Möglichkeiten

- Abstrakte Interpretation
- Abhängigkeitsanalyse
- Logische Analyse mit Theorembeweiser
- Emulation und dynamische Instrumentierung
- Symbolische Ausführung

Resultate

- Automatic Patch-based Exploit Generation
- Hidden Code Extraction from Packed Executables
- Deviation Detection in Binaries
- Automatic Malware Dissection and Trigger-based Behavior Analysis
- …

Zusammenfassung

- Diskussion der Problematik
- Abstrakte Interpretation
- Metasprachen
- Dynamische Instrumentierung
- Taint Tracking
- BitBlaze

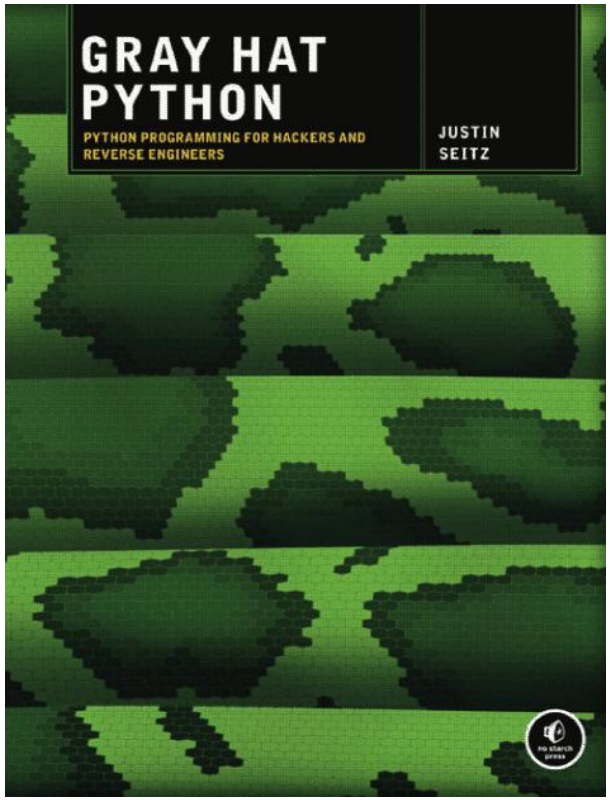
Was haben wir gelernt?

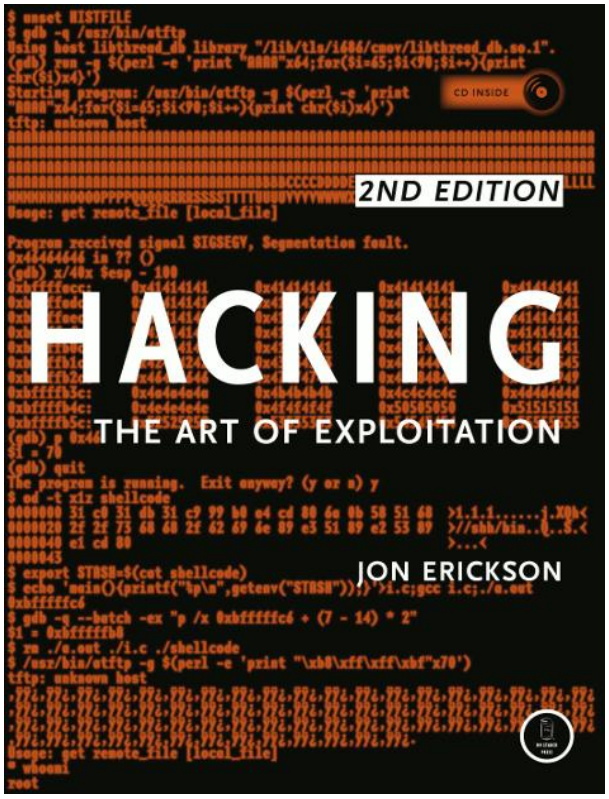
- Daten sind wichtiger als Code
- Formale Methoden werden benötigt um Komplexität entgegenzuwirken
- Es wird viel geforscht aber wenig angewand
- Reverse Engineering ist ein Feld mit anspruchsvollen und interessanten Aufgaben

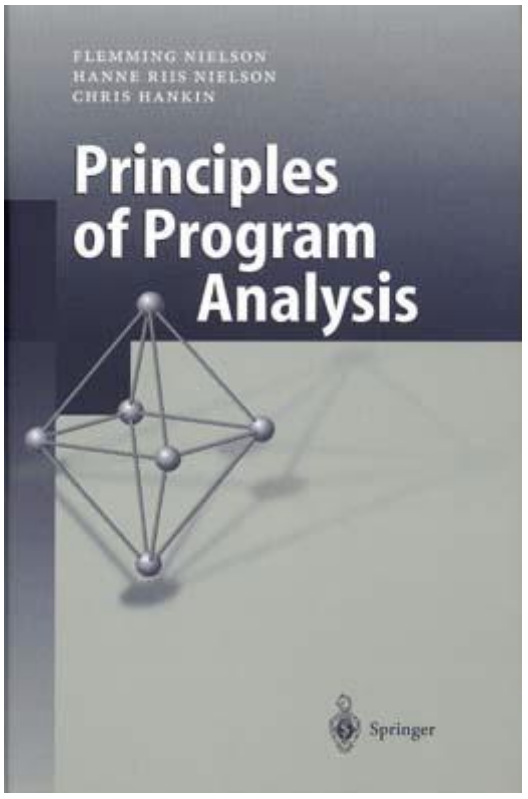
Webseiten

- Reverse Engineering Reddit
  - http://www.reddit.com/r/reverseengineering
- Lambda – The Ultimate
  - http://lambda-the-ultimate.org/
- The Program Transformation Wiki
  - http://www.program-transformation.org/

Bücher


GRAY HAT
PYTHON
PYTHON PROGRAMMING FOR HACKERS AND
REVERSE ENGINEERS
JUSTIN
SEITZ


CD INSIDE
2ND EDITION
HACKING
THE ART OF EXPLOITATION
JON ERICKSON


FLEMMING NIELSON
HANNE RIIS NIELSON
CHRIS HANKIN
Principles
of Program
Analysis
Springer

Universitäten I

- University of Berkeley
  - http://bitblaze.cs.berkeley.edu/
- University of California, Santa Barbara
  - http://www.cs.ucsb.edu/~seclab/
- Carnegie-Mellon University
  - http://www.cylab.cmu.edu/index.html

Universitäten II

- Stony Brook University
  - http://seclab.cs.sunysb.edu/seclab/
- Universität Karlsruhe
  - http://pp.info.uni-karlsruhe.de/publications.php

Diplomarbeiten

- MonoREIL Algorithmen zur Programmanalyse
- Input Crafting um Verzweigungen zu erfüllen
- Semantisches Diffen
- Trennung von Code und Daten

Fragen

http://www.flickr.com/photos/marcobellucci/3534516458/