# CODE ANALYSIS AND TRANSFORMATION

**Jakub Křoustek**
Bachelor Degree Programme(3), FIT VUT
E-mail: xkrous00@stud.fit.vutbr.cz

Supervised by: Alexander Meduna
E-mail: meduna@fit.vutbr.cz

## ABSTRACT

This paper describes methods and procedures used for code analysis and transformation. It contains basic information of a science discipline called reverse engineering and its use in information technologies. The primary objective is a construction of tool that can disassemble from binary form to symbolic machine code. This operation is highly dependent on the concrete instruction set, and it has to be used for a beforehand known processor architecture. This problem is solved with patterns, plug-ins, and modularity of disassembler. These features provide users the ability to add new instruction sets into this disassembler. The output is the text representation of instructions and is functionally equivalent to the input. The thesis demonstrates usual methods of disassembly as well as the methods made by the author.

## 1. ÚVOD

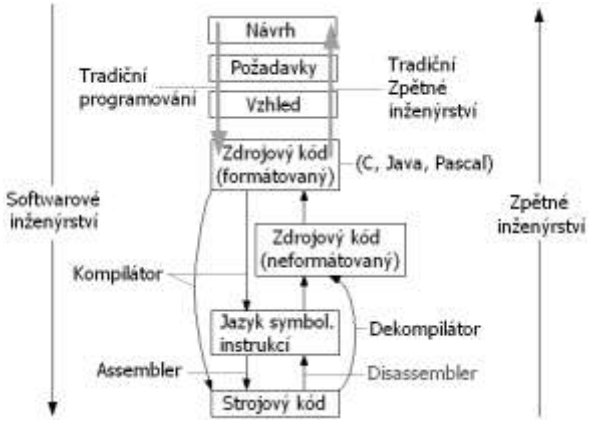
Snaha o zkoumání a poznávání neznámých věcí byla člověku vždy vlastní. Proto není divu, že se lidská zvídavost rozšířila i v moderních vědních oborech výpočetní techniky. Již řadu let můžeme sledovat rozvoj odvětví zvané Zpětné inženýrství (z anglického Reverse engineering). Jedná se o celou řadu disciplín, které vedou k úplnému pochopení vlastností a vazeb zkoumaného objektu a jejich přetvoření do jiné formy.

Současná doba si žádá algoritmy a nástroje, které jsou schopny rychle a efektivně analyzovat spouštěné programy či přenášená data. To může být potřeba z bezpečnostních důvodů (antivir, firewall) či ve snaze o optimalizaci výkonu (překladače, vestavěné systémy, …). Je tedy potřeba vytvořit prostředek pro analýzu neznámého kódu a jeho transformaci do srozumitelné podoby. To je hlavním cílem této práce.

Zpětnému inženýrství bude věnována následující kapitola. Další kapitola pak bude popisovat tvorbu nástroje zvaného *disassembler*.

## 2. ZPĚTNÉ INŽENÝRSTVÍ

Proces Zpětného inženýrství si můžeme představit jako proces inverzní ke klasickému softwarovému inženýrství. V obou metodách nacházíme totožné kroky (návrh, implementace, zdrojový kód, výsledný binární produkt, …), s tím rozdílem, že jsou chronologicky řazeny opačně. Více je patrno z obrázku 1.




**Obrázek 1:** Jednotlivé metody Zpětného inženýrství.

Některé z metod Zpětného inženýrství ukáži detailněji, další metody jsou popsány v [1].

### 2.1. DEKOMPILACE

Dekompilace je postup získání zdrojového kódu vyššího programovacího jazyku z již zkompilovaného programu. Svojí podstatou se tedy jedná o opak překladače. Je nutné si však uvědomit, že kvůli optimalizacím a úpravám kódu při překladu většinou není možné získat totožný zdrojový kód.

### 2.2. DISASSEMBLER

Jedná se o obrácený postup než je assembler, tedy ze zkompilovaného programu se vytvoří jeho reprezentaci v jazyku symbolických instrukcí. Výstup se někdy též nazývá mrtvý kód. Vyplývá to z jeho povahy, jelikož se po vytvoření již nijak nemění. Tento proces je silně závislý na architektuře procesoru a na typu instrukční sady, ale oproti dekompilátoru je možné ho sestrojit poměrně snadno pro každou architekturu.

## 3. TVORBA DISASSEMBLERU

Vstupem disassembleru je strojový kód. Snahou disassembleru je převést tento strojový kód, čitelný jen pro procesor, do sekvence instrukcí zapsaných v jazyku symbolických instrukcí. Tento zápis může být dále například formátován pro uživatele.

Stěžejní část disassembleru tvoří systém dekódování instrukcí. Velkou roli zde hraje architektura procesoru (*RISC* či *CISC*), způsob zarovnání v paměti (*alignment*) a struktura relokačních informací. Postupů pro transformaci ze strojového jazyku je několik a každý má své výhody i nevýhody. Některé z postupů jsou popsány v [2].

### 3.1. METODA – LINEÁRNÍ PRŮCHOD

Tento algoritmus prochází vstupní soubor bajt po bajtu a podle databáze instrukcí vyhodnocuje každý prvek. Jedná se o nejpoužívanější metodu. Je velmi rychlá, ale také nepřesná. Chybuje například při snaze o dekódování bajtů využitých jako výplň pro zarovnání. V tomto případě nemusí disassembler zachytit začátek instrukce a bude se snažit dekódovat nesmyslnou posloupnost binárních dat.

Nejlépe se metoda osvědčila u krátkých jednoduchých programů, kde dosahovala výborné úspěšnosti. Naopak při dekódování delších strukturovaných programů klesla úspěšnost rozpoznávání instrukcí pod 50% hranici.

### 3.2. METODA – REKURZIVNÍ ZANOŘENÍ

Při průchodu vstupním souborem jsou vyhledávány instrukce skoku a volání. Při každém nálezu se pokračuje na adrese skoku (volání) a dekóduje se normálně až do nalezení instrukce návratu, skoku či volání. To umožňuje vyhnout se chybám, které vznikají zarovnáním nebo transformováním datové sekce programu. Vzniká ale chyba při použití instrukcí typu nepřímý skok, kdy nemusí být identifikována celá návěští.

Metoda má kvalitnější výsledky než lineární postup, je časově náročnější a implementačně složitější. Úspěšnost dekódování není závislá na délce vstupu a je přibližně 80%.

### 3.3. METODA – HYBRIDNÍ ALGORITMUS

V tomto případě se jedná o kooperaci lineárního a rekurzivního algoritmu. Zdrojový soubor se dekóduje souběžně oběma algoritmy. Po dekódování se porovnávají výsledné instrukce na jednotlivých adresách. Tam, kde je nalezena shoda, je výsledek konečný. Adresy, kde jsou výsledky rozdílné, mohou být znovu zkoumány či označeny za chybné. Algoritmus je značně pomalý (prochází soubor 2krát), ale je velmi účinný (více než 90%).

### 3.4. METODA – PŘEKLADOVÉ AUTOMATY

Novinkou v této oblasti jsou překladové automaty, které používají atributované párové automaty a párovou gramatiku. Program zapsaný ve strojovém kódu je zde chápán jako věta gramaticky popsaného jazyka. Tato technika má umožnit překlad z jazyka symbolických instrukcí do binární formy, ale i překlad opačný. Metoda je použita v projektu Lissom [3]. V této práci nebyla metoda prozatím použita, ale jeví se jako velmi nadějná náhrada předcházejících metod.

### 3.5. IMPLEMENTACE

Program vytvářím v jazyce C/C++. Grafické rozhraní je realizováno pomocí tool-kitu Qt. V současné době je aplikace ve fázi ladění a testování, ale i přesto poměrně jasně ukazuje rozdíly v efektivitě jednotlivých algoritmů transformace.

## 4. ZÁVĚR

Výše nastíněné algoritmy jsou demonstrovány v programu, který je součástí mé bakalářské práce. Disassembler je modulární a umožňuje uživatelům rozšíření o nové instrukční sady. Dalšího vylepšení může být dosáhnuto zrychlením transformačních algoritmů. Následující vývoj projektu si představuji v tvorbě antiviru nebo jiného bezpečnostního programu. Půjde o vyhledávání řetězců instrukcí ve transformovaném kódu. Na základě definovaných posloupností instrukcí bude možné heuristicky odhalit škodlivý kód.

## LITERATURA


[1] Eilam E.: Reversing: Secrets of Reverse Engineering. Wiley, 2005, ISBN 978-0-7645-7481-8

[2] Schwarz B., Debray S. K., Andrews G. R.: Disassembly of Executable Code Revisited. University of Arizona, Tucson, 2002

[3] WWW stránka projektu Lissom , http://merlin.fit.vutbr.cz/Lissom (březen 2007)