Automated unpacking of executables using DBI

Tadas Vilkeliskis <tvilkeli@stevens.edu>
Department of Computer Science, project advisor Sven Dietrich

STEVENS Institute of Technology

Technogenesis Undergraduate Scholars' Program, 2009

Introduction

What:

- Find a better generic way to automatically unpack executables with multi-layer packing.
- Be able to handle multi-threaded applications.

Why:

- To make analysis of malicious software faster and easier.

How:

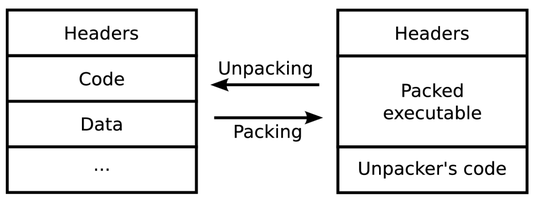
- Analyzed malicious software and various packer formats.
- Unpacking by code injection using dynamic binary instrumentation.

Dynamic Binary Instrumentation (DBI): A runtime binary analysis method where instrumentation code is inserted into the running binary application.

Packing

Packing: A process of compressing or encrypting an executable file and embedding it into a newly created executable.

Example:



Goal: Protect executable from reverse engineering.

Protection types:

- Multiple packed layers.
- Debugger, emulator and tracer checks.
- Multi-threaded unpacking algorithms.
- Control transfer transformations and code obfuscation.
- Unpacking on demand.
- Anti-dumping.
- Virtual machines and custom instruction set.

Commercial application: A lot of commercial software, such as *Wolfram's Mathematica*, use packing to protect intellectual property. Even commercial packing solutions exist.

Design

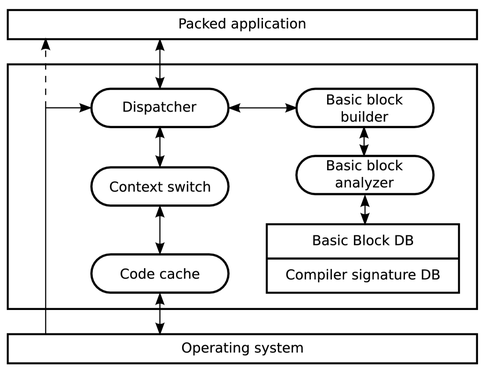
The generic unpacker consists of two modules: injector module and unpacker module.

Injector module

Creates suspended process and injects unpacker into the process address space.

Unpacker module

Unpacker module works as a proxy between the operating system and a packed application.



Dispatcher: Makes decisions and is entered from control retention code.

Context switch: Switches between the process context and unpacker's context by saving CPU state and creating stack transparency.

Code cache: Executes code natively and retains control.

Basic block builder: Creates a basic block and places it in code cache.

Basic block analyzer: Performs analysis of a basic block to find out whether the code is original entry point of the program.

Portability

The code targets Windows OS. The injector, dispatcher and code switch should be modified in order to execute the code on other OS.

Multi-Threading

Same Instrumentation Code Multiple Data design.

Real-world application

The unpacker should be able to instrument most of the malicious software such as *Conficker* or *Waledac.*

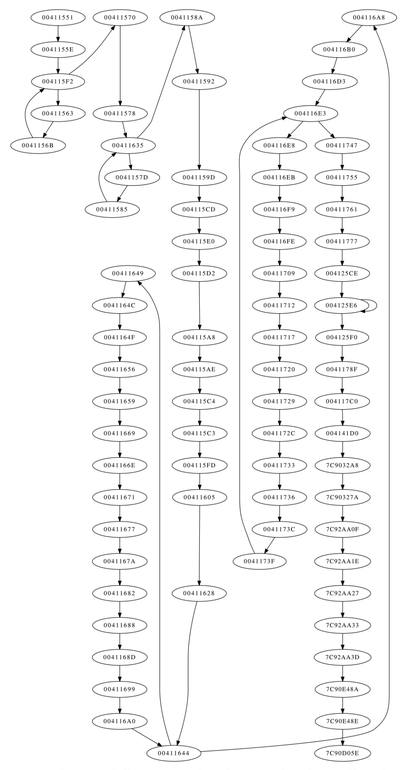
Possible unpacker evasion techniques include: a recently discovered vulnerability in the system that could allow indirect control transfer, hook detection or unpacker's code/data corruption.

Results

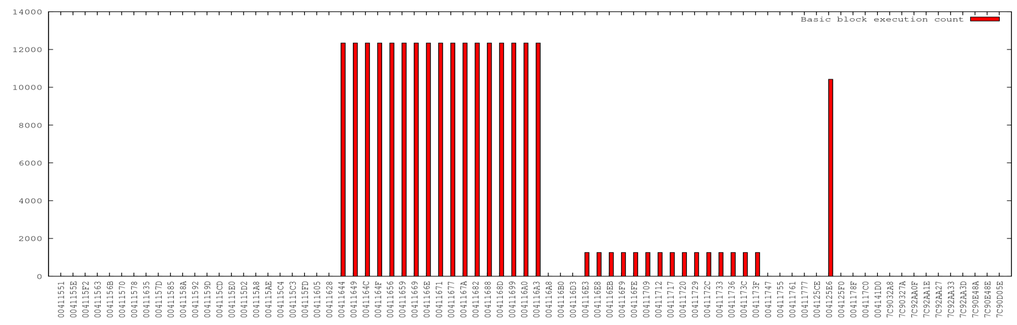
The unpacker is at the early prototype stage. Currently it can perform basic block traces via:

- Calls, jumps and returns.
- Structured exceptions (SEH).
- *NtContinue*.

Stack and CPU transparency has also been achieved–the unpacker does not leave any CPU or stack memory modifications.



Control flow graph of a packed application.



Basic block execution frequency histogram.

Future work

- Implement basic block analyzer.
- Build signature database.
- Add support for multi-threaded applications.