Paradyn Parallel Performance Tools

PatchAPI

Programmer's Guide

1.0b Release
September 2012

Computer Science Department
University of Wisconsin–Madison
Madison, WI 53711

Computer Science Department
University of Maryland
College Park, MD 20742

Email `bugs@dyninst.org`
Web `www.dyinst.org`

Contents

1 Introduction

This manual describes PatchAPI, a programming interface and library for binary code patching. A programmer uses PatchAPI to instrument (insert code into) and modify a binary executable or library by manipulating the binary's control flow graph (CFG). We allow the user to instrument a binary by annotating a CFG with *snippets*, or sequences of inserted code, and to modify the binary by directly manipulating the CFG. The PatchAPI interface, and thus tools written with PatchAPI, is designed to be flexible and extensible. First, users may *inherit* from PatchAPI abstractions in order to store their own data. Second, users may create *plugins* to extend PatchAPI to handle new types of instrumentation, different binary types, or different patching techniques.

PatchAPI represents the binary as an annotatable and modifiable CFG. The CFG consists of abstractions for binary objects, functions, basic blocks, and edges connecting basic blocks, which are similar to the CFG abstractions used by the ParseAPI component.

Users instrument the binary by annotating this CFG using three additional high-level abstractions: Point, Snippet, and Instance. A Point supports instrumentation by representing a particular aspect of program behavior (e.g., entering a function or traversing an edge) and containing instances of Snippets. Point lookup is performed with a single PatchAPI manager (PatchMgr) object by Scope (e.g., a CFG object) and Type (e.g., function entry). In addition, a user may provide an optional Filter that selects a subset of matching Points. A Snippet represents a sequence of code to be inserted at certain points. To maximize flexibility, PatchAPI does not prescribe a particular snippet form; instead, users may provide their own (e.g., a binary buffer, a Dyninst abstract syntax tree (AST), or code written in the DynC language). Users instrument the binary by adding Snippets to the desired Points. An Instance represents the insertion of a particular Snippet at a particular Point.

The core PatchAPI representation of an annotatable and modifiable CFG operates in several domains, including on a running process (dynamic instrumentation) or a file on disk (binary rewriting). Furthermore, PatchAPI may be used both in the same address space as the process (1st-party instrumentation) or in a different address space via the debug interface (3rd-party instrumentation). Similarly, developers may define their own types of Snippets to encapsulate their own code generation techniques. These capabilities are provided by a plugin interface; by implementing a plugin a developer may extend PatchAPI's capabilities.

This manual is structured as follows. Section **??** presents the core abstractions in the public and plugin interface of PatchAPI. Section **??** shows several examples in C++ to illustrate the usage of PatchAPI. Detailed API reference can be found in Section **??** and Section **??**. Finally, Appendix **??** provides a quick tutorial of PatchAPI to those who are already familiar with DyninstAPI.