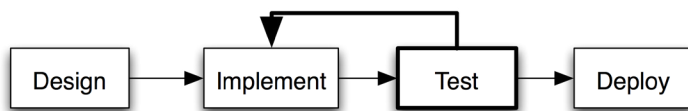


DEADBOLT

Adaptive testing for PCS software robustness and security

Overview

Vulnerabilities in PCS and SCADA software threaten availability and integrity of control and data acquisition services provided by these systems. The DEADBOLT tool suite provides PCS and SCADA software developers with state-of-the-art technology that facilitates automatic discovery of buffer overflow and resource exhaustion vulnerabilities. Buffer overflow errors can be exploited by a malicious attacker to make unauthorized changes to system configuration, learn confidential information or force process control software to fail. Resource exhaustion conditions can result in software failing to support legitimate operations (denial of service) and can be triggered by causes ranging from simple equipment malfunction to a worm infection or a malicious attack. Since availability and integrity of control system operations are of critical importance, finding and addressing these errors in PCS and SCADA software before it is deployed is paramount.



“Software bugs, or errors, are so prevalent and so detrimental that they cost the US economy an estimated \$59.5 billion annually. Approximately 60% of the costs are borne by software users and 40% by software developers and vendors. Although all errors cannot be removed, more than a third of these costs (an estimated \$22.2 billion) could be eliminated by an improved testing infrastructure that enables earlier and more effective identification and removal of software defects.”

Source: “The Economic Impacts of Inadequate Infrastructure for Software Testing”, NIST 2002.

The DEADBOLT tool suite focuses on helping software developers discover these errors during *implementation* and *testing* phases of the software development life-cycle. DEADBOLT supplements functional testing by

- Statically analyzing program source code to identify potential vulnerabilities,
- Automatically generating test cases that trigger buffer overflows, memory corruption errors or resource exhaustion, and
- Providing detailed information about discovered errors, including exact location of faulty code and input that triggers erroneous behavior.

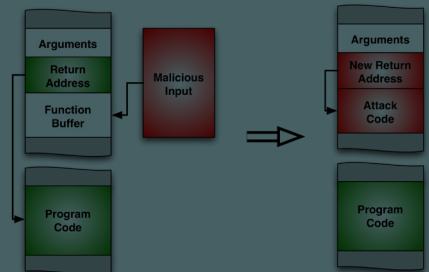
DEADBOLT’s error reports greatly facilitate debugging and enable software developers to create more secure and robust implementations of PCS and SCADA software.

What is a Buffer Overflow?

Memory in C and C++ programs is managed by creating and destroying memory objects. Objects can be created and destroyed explicitly by the programmer or automatically by the system. A buffer overflow is any access to an object that ventures beyond the bounds of its allocated memory. Writing beyond the bounds of an object frequently enables the attacker to modify program logic by changing operating parameters or subvert the program altogether and execute arbitrary code.

A stack-smashing attack

“Stack-smashing” is a common attack that exploits an overflow in a stack buffer. Upon entry to a function the program saves a “return address” specifying code that should be executed after the function is completed. This important value is stored on the stack, next to function arguments and buffers used by the function for processing inputs. An attacker can craft an input that is too large for the buffer, and, if appropriate checks are not made, the function will continue to write beyond the bounds of the buffer, overwriting the saved return address. A buffer overflow thus allows an attacker to insert attack code into the buffer and change the return address such that this code is executed when execution of the current function is completed. In effect, the attacker takes over the process and can execute arbitrary code with the privilege level of the process that has been exploited.



Key Features and Benefits

- Bugs found and fixed before software is deployed
 - Increased security from malicious attacks
 - Increased robustness for day-to-day operations
- Reduced cost and increased effectiveness of testing process
 - Automated testing framework enables developers to spend more time fixing bugs, less time finding them
- Reduced cost of maintenance
 - Fixing defects in the field costs 13 times more than fixing them during development*
- Low cost of adoption
 - Automatic code instrumentation via source-to-source transformation enables DEADBOLT to work with any standards-compliant C/C++ compiler
- More robust and secure software means safer, more efficient operations for your customer

About MIT Lincoln Laboratory



Mr. Michael Zhivich • mzhivich@ll.mit.edu

MIT Lincoln Laboratory has pioneered research and development in advanced electronics since its origin in 1951 as a Federally Funded Research and Development Center of the Massachusetts Institute of Technology. The scope of the problems has broadened from the initial emphasis on air defense to include information assurance, communications, space surveillance, missile defense, tactical surveillance systems, and air traffic control. The Laboratory’s fundamental mission is to apply science and advanced technology to critical problems of national security.

Functional Description

DEADBOLT is a suite of developer tools that complement functional and regression testing during the *implementation* and *testing* phases of the software development life-cycle. DEADBOLT tools require source code for the application under test and a collection of sample valid inputs. Final version of the tool suite will be incorporated into an integrated development environment (IDE), such as Eclipse, which provides developers with easy navigation and editing capabilities, debugger, performance profiling tools, etc. This integration will ensure that DEADBOLT tools are easy to use and leverage an environment already familiar to developers.

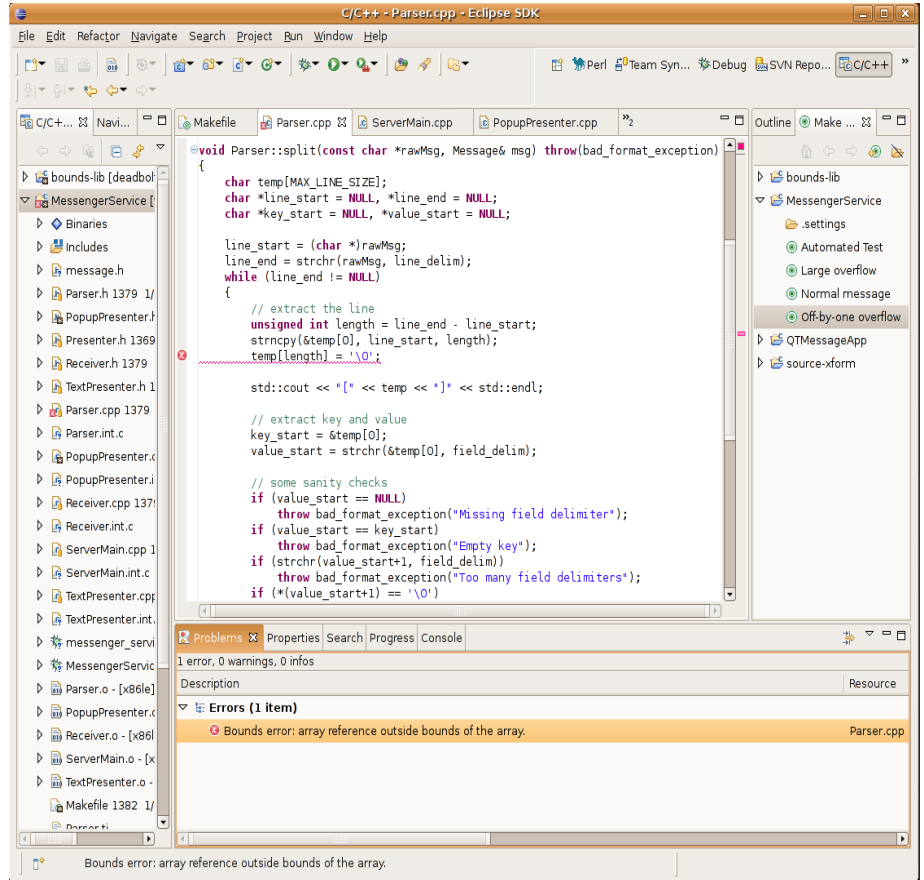
Technical Description

DEADBOLT tools employ a state-of-the-art hybrid approach to find software vulnerabilities. By performing static analysis of source code first and using results to guide dynamic testing, the tool suite leverages the strengths of both approaches, while mitigating the high false alarm rates that frequently plague pure static analysis.

To discover resource exhaustion vulnerabilities, a static analysis is used to identify sites where consumable resources (e.g. memory, file descriptors, network sockets, locks) are allocated and released. This analysis can also identify *greedy* paths through the code that allocate resources, but never release them. A dynamic testing component then runs the application under instrumentation to generate an input that forces the program down the *greedy* path that exhibits erroneous behavior.

To discover buffer overflows, DEADBOLT uses source-to-source transformations to insert instrumentation that keeps track of all allocated objects and checks all memory accesses for buffer overflow violations. Automated testing component uses information gathered by this instrumentation at runtime to generate test cases that trigger erroneous behavior in the program or verify that the software is implemented correctly.

The screenshot above shows the Eclipse integrated development environment with output of DEADBOLT buffer overflow detection tool pin-pointing the line of source code containing the out-of-bounds memory access as well as the error description.



Summary of Costs

	Low	Medium	High
Component	<\$1,000	\$1,000-\$10,000	>\$10,000
Engineering	Drop-in	Moderate modification	Complete System Design Lifecycle
Bandwidth/Network Burden	None-Passive Only	Moderate Traffic, Medium Overhead	Heavy Traffic, Large Overhead
Training	No training required	Moderate training	Intensive training. Additional staffing may be required.
Maintenance and Operation	< 1 hour per week	< 5 hours per week	> 5 hours per week

Technology Transfer and Readiness: Bench-tested.

This material is based upon work supported by the U.S. Department of Homeland Security under Grant Award Number 2006-CS-001-000001, under the auspices of the Institute for Information Infrastructure Protection (I3P) research program. The I3P is managed by Dartmouth College. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the U.S. Department of Homeland Security, the I3P, or Dartmouth College.



The I3P is managed by Dartmouth College.