Host-Based Intrusion Detection Using an FPGA

MICHAEL SEVILLA (CSE)
MENTOR: IAN HARRIS

Abstract
Malicious computer attacks by invasive software programs have led to the emergence of intrusion detection techniques and research. Traditional anti-malware systems used software to combat attacks but in the project I used hardware. I extracted the program counter from the LEON3 processor and used that information to create an FPGA. The FPGA was used to test the processor because it imitates hardware chips. The system accurately monitored the program counter of the processor and raised a flag if the display of the FPGA when the program counter went out of the specified range. This intrusion detection system did not interrupt the execution of the FPGA and successfully extracted the signal.

Introduction
A computer system under attack can result in malicious activities that can alter important files and transfer control of a computer system to an unauthorized user. For example, the zer0 day worm caused serious damage in 2000 when it quickly spread across Microsoft operating systems [1].

Several prevention systems such as firewalls have been used to fight in malicious computer attacks. Unfortunately, these attacks have evolved in parallel with software firewall systems as they deal in constantly being outdated as a result of the next attack. One example of a popular prevention system is an intrusion detection system (IDS), which aims to recognize attacks.

There are two types of IDSs:

- Network Intrusion Detection Systems (NIDS): focuses on traffic such as firewalls, network packets, etc.
- Host-Based Intrusion Detection Systems (HIDS): focuses on events, such as the access, application execution, processes, etc.

The problem with many of these techniques is the approach. Software methods stop the processor and gather information on each clock cycle.

Stopping the processor negatively impacts the performance of the system because it:
- slows down the system
- wastes resources
- can be detected by the malware

To avoid these problems, my research focuses on utilizing hardware instead of software for an HIDS. Hardware checks behavior as programs are running rather than storing and predicting behavior. Although using hardware to track events in software is difficult, their coherency can relax design constraints because speed and flexibility are optimized.

Methods
Materials
FPGA: Digilent Spartan-3 Xilinx FPGA xc3s1000 board
Processor: LEON3
Environment: Xilinx ISE Design Suite
Cable: Xilinx Platform Cable USB II
Compiler: Bare-C Cross-Compiler (BCC)
GRMON and GRLIB

Results
The LEON3 was successfully synthesized onto the FPGA and monitored the PC. The running programs resulted in different results.

Experimental Results
As Table 1 demonstrates, the experiment results are mixed. The c programs mike1.c and mike2.c produced visible results because the display flashed faster than the human eye can see. To make sure the program was indeed running, mike3.c was executed so that the flag was displayed. mike4.c tests a boundary case to ensure that the attached HIDS does not affect the stability of the processor. mike5.c adequately demonstrates the effectiveness of the system as the execution of the program shows a flashing "F" indicating that "F" and "I" are being intermittently displayed.

Analysis
The results seem mixed but the tests programs show that the system accurately monitors the program counter. It is difficult to see feedback for a computer system, especially one that emulates a processor. The speed makes it difficult for the tester but it solidifies the effectiveness of the approach. Although the programs are small and simple, it is encouraging to see the system behave in a way that is consistent with the hypothesis that hardware solutions do not interrupt the processor.

Conclusions
A successful intrusion detection system was implemented and tested with hardware. The results were minimal but prove that such a system is feasible. Using the hardware and software on an FPGA was shown that an effective monitor can be added to a processor and can raise a flag in real time. This research can help form a basis for future intrusion detection because it has greater speed and efficiency than software implementations of similar systems. With the advent of higher quality malware and worms, anti-malware techniques will need to be more sophisticated and hardware implementations may pave the way for techniques that halt even the most complex attacks.

Further Work
- Fun and more complex programs through the processor
- Attach entities that monitor system calls, branches, and jumps
- Check the state machine at compile time so that expected behavior can be compared to observed behavior
- Emulate attacks by simulating a buffer overflow using a web server

Table 1: The results of executing programs through the processor.

<table>
<thead>
<tr>
<th>Test ID</th>
<th>Function</th>
<th>Display Output</th>
<th>Result</th>
<th>Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>mike1c</td>
<td>1 output pulse</td>
<td>Flag raised only</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mike2c</td>
<td>2 output pulse</td>
<td>Flag raised only</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mike3c</td>
<td>3 output pulse</td>
<td>Flag raised only</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mike4c</td>
<td>4 output pulse</td>
<td>Flag raised only</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mike5c</td>
<td>5 output pulse</td>
<td>Flashing &quot;F&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Reference