The Design and Analysis of a Hardware-based Anomaly Detection Scheme

JinLong Piao and Seong Baeg Kim*

Dept. of Computer Education, Jeju National University, Jeju, Korea
herodragon107@hotmail.com, sbkim@jejunu.ac.kr

Abstract

We propose and analyze a novel approach for security based on an execution behavior of an application program, which aims at detecting previously unknown anomaly execution patterns. Our scheme, which is a sort of hardware-based approach, uses the basic block information of an application program for the purpose of detecting and preventing unknown anomaly execution behaviors effectively. Furthermore, we present a possibility to connect our scheme with an existing branch prediction scheme such as BTB (Branch Target Buffer).

Keywords: *Security, Branch prediction, Basic block, Anomaly execution behavior, BTB*

1. Introduction

Recently, computer system security is getting more important as an end user's private information, which is required to be protected such as electronic money, grows large. Malicious programs generally perform anomaly operations to invade computer system. To detect and prevent the malicious operations of an application program, there has been much research on both software and hardware approaches. However, unlike well-known security attacks, there are still many problems in dealing with unknown security attacks.

The vulnerability of security is being reduced as the patch to solve the holes at system level is prompt and new systems, which are frequently updated, are more secure. So, we focus on application-level security scheme, based on the patterns of end user's computer usage. Unlike the previous computer system, the recent computer system has a new trend. For example, in most cases including embedded systems, end users use only the small number of applications, which are less than 10. Also, updating or installing a new application is not frequent. Therefore, we presented a novel scheme for detecting anomaly behavior executions [6]. Based on our scheme for security considering these characteristics, we examined the basic block information through simulation using SPEC benchmarks.

The rest of the paper is organized as follows. Section 2 briefly summarizes the theoretical background of the system security. Section 3 presents a new framework for the system security. Section 4 examines how to combine efficiently with the existing instruction prediction scheme. Section 5 presents the experimental results of our scheme. Section 6 gives a description of performance analysis considering overheads. Section 7 makes a conclusion of the paper.

* Corresponding author

2. Related Work

To attack a system, there should be at least malicious two operations. First, there should be an operation to inject an attack code into a system. Second, it is required to hijack a program execution control. For example, stack smashing is a well-known approach to be used to attack a system. There has been several research related to stack smashing. The basic solution to prevent stack smashing is to prohibit a stack overflow that can occur from the execution code stored in data segments. However, some programs attempt to generate an executable code at run time using just-in-time compiler, which is required for executing instructions stored in a data segment.

There are two hardware-based approaches to solve stack overflow attacks. [2] uses a secure cache called *SCache Architecture*. [3] uses an LIFO small memory stack called *SRAS (Secure Return Address Stack)*. Also, there is a compiler-based approach [4]. This approach is based on a combination of static and dynamic monitoring and proposes models using context-free grammar and automata. However, these approaches do not provide a solution to detect any unknown anomaly execution behaviors except stack smashing. [5] discussed various anomaly detection techniques developed in the huge literature and provided an structured and comprehensive overview on them.

3. The Structure for Basic Block Information

Our scheme uses the basic block of an application program, which is a sequence of instructions with one entry point and one exit point within it.

3.1 Basic Idea

Our scheme aims at detecting anomaly execution behaviors using the basic block information of an executable program. It is required to maintain the pairs of the starting and ending address of each basic block in a table called *EBCI* (*Execution Boundary Check Information)*, as shown in Table 1.

Table 1. The Table Structure of EBCI

Starting Addr.	Offset or Ending Addr.

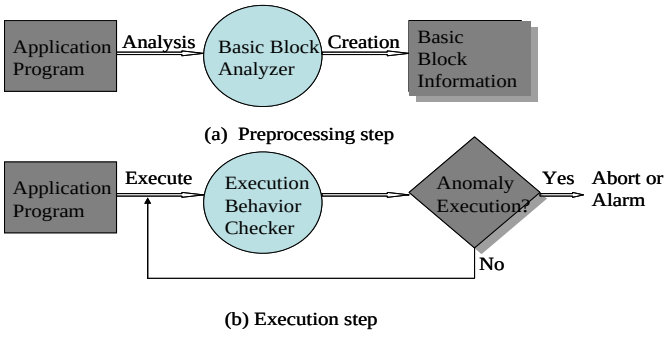

Figure 1. Two steps for Security

There are two steps to enable the operation of our scheme as shown in Figure 1. As the first step, it is necessary to construct EBCI from an application program using a basic block analyzer. The next step is to detect anomaly execution behaviors by checking EBCI using an execution behavior checker during execution.

3.2 Possibility of Connection with BTB

We present how to connect our scheme with branch history information used for branch prediction. As shown in Figure 2, we extend the existing branch target buffer (BTB) to include the additional information for checking the execution boundary. The EBCI table includes all branch instruction address.

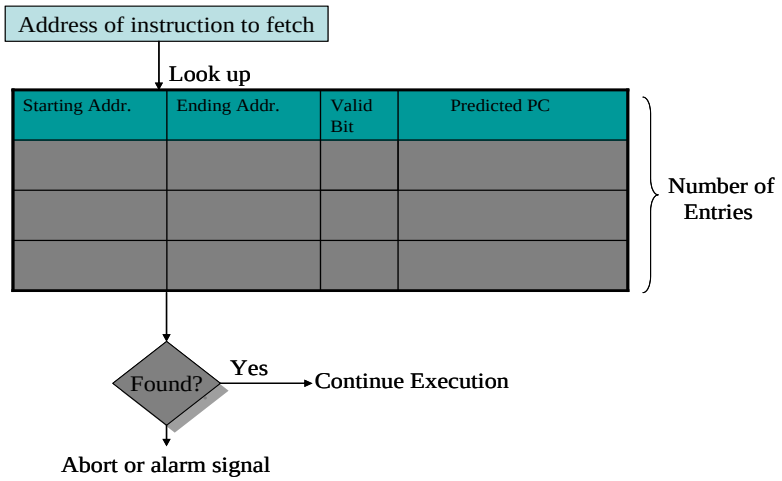

Figure 2. EBCI Internal Structure including BTB

4. Internal Architecture

There are two possible architectures to support our scheme depending on the type of the address used. Figure 3 shows the virtual address-based internal architecture. In that case, EBCI has an impact on the internal architecture of CPU. If the address from CPU could be not found in EBCI, there would be an abort or alarming signal. We propose the architecture, where EBCI and TLB are accessed concurrently to avoid performance degradation.

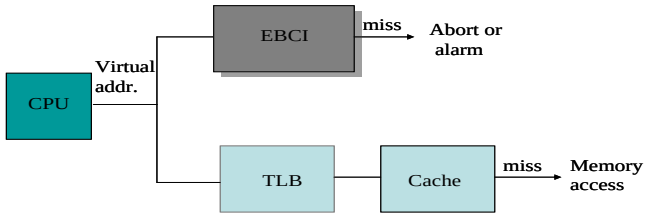

Figure 3. Virtual Address-based Structure

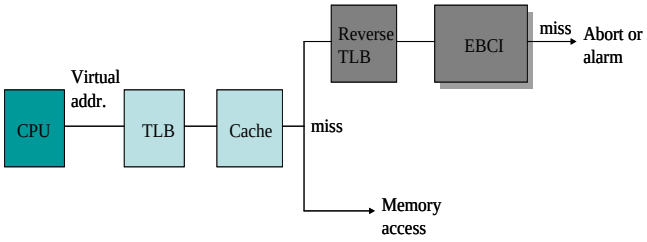

Figure 4. Physical Address-based Structure

Figure 4 shows the physical address-based internal architecture. Unlike Figure 3, it is required to do the reverse TLB to get a physical address. In that case, EBCI is accessed only when there is a cache miss. It contributes to reducing the number of the memory system accesses.

5. Experiment Results

5.1 Experimental Method

We made an experiment using SimpleScalar simulator and developed an analysis tool to obtain basic block information. We used 'objdump' program to get the instruction trace information of the program. We used two benchmarks : SPEC200 benchmarks and Mibench v1.3. We verified our approach by inserting the address of an anomaly execution behavior into a normal execution trace. The analysis tool developed collected the information such as the number of instruction per basic block, the basic block's starting address, the basic block's ending address, the distribution of the basic block size, which is the number of instruction within a basic block.

5.2 Analysis of Benchmarks

Table 2 shows the information of the benchmarks chosen. As shown in the figure, there are seven benchmarks that consist of SPEC and Mibench. The average number of instruction in a basic block ranges from 3 to 6. The number of the basic blocks of the benchmark gcc has the biggest among them.

Table 2. The Information of the Benchmark

Benchmark	**Average of the number of instruction in a basic block**	**Total number of basic block (the number of EBCI entry)**	**Total number of instruction**
Consumer_jpeg_c	6	9081	60808
Consumer_jpeg_d	6	8874	59744
Consumer_tiff2bw	5	10770	62544
Consumer_tiff2rgba	5	10746	62289
gcc	3	35759	131014

gzip	4	6033	29600
bzip	5	3592	18275

Table 3 shows the distribution of the basic block size for the benchmarks. As you can see in the figure, most of the basic block size is less than 25. For 7 benchmarks, about 95 percent of the blocks have no more than 15 instructions and about 71 percent of the blocks have no more than 5 instructions. It represents that about 17 percent of the basic blocks have one instruction. Therefore, it's possible to detect abnormal execution patterns using our method.

Table 3. The Distribution of the Basic Block Size

Benchmark	Number of instruction			
	1~25	1~15	1~5	1
Consumer_jpeg_c	97.47%	91.17%	64.56%	18.8%
Consumer_jpeg_d	97.5%	91.91%	64.2%	18.35%
Consumer_tiff2bw	98.45%	94.44%	70.46%	21.18%
Consumer_tiff2rgba	98.46%	94.47%	70.6%	21.1%
gcc	99.78%	98.68%	84.77%	16.67%
gzip	99.04%	96.47%	69.95%	12.40%
bzip	98.66%	96.24%	70.85%	13.06%

6. Analysis

In our scheme, there is some overhead to maintain EBCI. Compared with BTB, our scheme is required to add the field "Starting Addr" in order to check an execution boundary. However, the number of the EBCI entry is little different because each basic block comes from a branch instruction. To reduce the size of EBCI, it's possible to maintain only the basic blocks that include a memory reference instruction, assuming that user's information can be only accessed through memory reference. To analyze the size of EBCI per application program, we examined the characteristics of the basic block of SPEC benchmarks. In case of SPEC benchmarks, each basic block consists of the number of arbitrary instructions that range typically between 1 and 25 [1]. Also, in our scheme, there are overheads related to reverse TLB and additional bandwidth between CPU and memory. However, the performance degradation is trivial because the operation related to EBCI is done concurrently while a cache miss operation is handled or TLB is accessed, depending on the type of the address.

7. Conclusion

To tackle unknown security attacks, we presented a new approach for detecting the anomaly execution behaviors of an application program, based on the basic block information of each application. We examined how to identify the boundary of normal execution behaviors effectively. The basic block information is used for finding an

anomaly execution behavior of an application program. Also, we devised the EBCI structure to provide the functionality of the branch target buffer used for branch prediction. Furthermore, we're examining how to adapt our approach to an embedded system considering low power consumption.

Acknowledgements

The authors would like to thank Hyeon-seok Kim for his assistance in getting the experimental results of this research.

References

[1] J. Huang and D. J. Lilja, "Exploiting Basic Block Value Locality with Block Reuse", Technical report HPPC-98-09, Univ. of Minnesota, **(1998)** August.

[2] K. Inoue, "Energy-Security Tradeoff in a Secure Cache Architecture Against Buffer Overflow Attacks", SIGARCH Computer Architecture News, 1, 33, pp. 81-89 **(2005)**.

[3] R. B. Lee, D. K. Kraig, J. P. McGregor and Z.Shi, "Enlisting Hardware Architecture to Thwart Malicious Code Injection", In Proceedings of the International Conference on Security in Pervasive Computing, **(2003)** March.

[4] D. Wagner and D. Dean, "Intrusion Detection via Static Analysis", In Proceedings of IEEE Symposium on Security and Privacy **(2001)**.

[5] V. Chandola, A. Banerjee and V. Kumar, "Anomaly Detection: A Survey", ACM Computing Surveys, 3, 41, **(2009)** July.

[6] H.-S. Kim, S. Je Cho, P. A. Wilsey and S. Baeg Kim, "Tackling Basic Block-based Anomaly Execution Behaviors", In Proceedings of the International Conference on Consumer Electronics, **(2010)** January.

Authors

JinLong Piao

2010 Computer Science and Statistics, Jeju National University, M.S.
2010 ~ Ph.D. course, Dept. of Computer Education, Jeju National University
Interests: Security, Computer Science Education, Edutainment, IT fusion
E-Mail: herodragon107@hotmail.com

Seong Baeg Kim

1995 Seoul National University, Ph.D.
1996~ Professor, Dept. of Computer Education, Jeju National University
Interests: Security, Computer Architecture, Computer Science Education, IT fusion
E-Mail: sbkim@jejunu.ac.kr