Software cannot protect software: An argument for dedicated hardware in security and a categorization of the trustworthiness of information

Matthew Judge, Paul Williams, Yong Kim, and Barry Mullins

Air Force Institute of Technology
2950 Hobson Way
Wright Patterson AFB OH 45433, USA
{matthew.judge,paul.williams,yong.kim,barry.mullins}@afit.edu

Abstract. There are many current classifications and taxonomies relating to computer security. One missing classification is the *Trustworthiness of Information* being received by the security system, which we define. This new classification along with *Timeliness of Detection* and *Security level of the Security System* present motivation for hardware-based security solutions. Including hardware is not an automatic solution to the limitations of software solutions. Advantages are only gained from hardware through design that ensures at least *First-hand Information*, dedicated monitors, explicit hardware communication, dedicated storage, and dedicated security processors.

1 Introduction

As security takes on ever increasing importance in today's connected, digital world; security solutions incorporate new, dedicated hardware at an increasing rate [1–13]. Though these works and many others investigate the incorporation of hardware into designs to gain different advantages, little work has been dedicated to understanding what precisely can be accomplished with hardware that cannot be accomplished solely with software solutions. Though many people believe a hardware-based solution is necessary to achieve effective security, little or no work exists demonstrating that this is true. The first and most obvious question to be asked is whether hardware solves the shortcomings and vulnerabilities of software based solutions. Exploring this question leads to a critical answer: *Not necessarily.* This work then, attempts to capture the necessary design elements for creating hardware that overcomes the weaknesses of purely software-based solutions. To aid in defining these requirements, we propose a classification for the *Trustworthiness of Information* and show that the necessary level of trust, *First-hand Information*, can *only* be achieved by properly designed hardware. Complete security systems will integrate these key hardware components with security software as needed.

2 Current Security Classifications

Significant work has been published on categorizations, classifications, and taxonomies for computer security. Bazaz and Arthur present a taxonomy of vulnerabilities [14]. Axelsson develops a taxonomy of detection methods [15], that Williams extends [16]. Kuperman classifies both the goals of detection and the timeliness of detection [17], and Stakhanova et al. work towards a taxonomy of intrusion detection system responses [18]. Mott presents work into classifying the level of security that the *security system* maintains for itself [5]. All of these different classifications provide valuable insight for working in the security field. One critical classification missing from these is the *Trustworthiness of Information*, which we develop in Sec. 4.3. Mott's work and Kuperman's timeliness of detection classification are both integral to the discussion of why hardware is necessary both on their own and how they relate to and are influenced by the *Trustworthiness of Information*. We discuss each in greater depth here.

Kuperman's notation categorizes time into an ordered sequence of events [17]. He defines the set of all events that can occur in the system, E, the subset of all malicious events, B, $B \subseteq E$, and three events a, b, c such that $a, b, c \in E$ and $b \in B$. Given the notation t_x to represent the time of event x occurring and $x \rightarrow y$ representing a causal dependence of y upon x we assume the three events are related such that $a \rightarrow b \rightarrow c$ yielding the relationship, $t_a < t_b < t_c$ must be true. Note that although $x \rightarrow y$ represents a causal dependence it does not necessarily mean that x is the direct cause of y. Kuperman uses $D(x)$ to represent the detection of an event x.

With this notation defined, Kuperman presents four main timeliness categorizations: real-time detection, near real-time detection, periodic detection, and retrospective detection. We discuss the first two here, which represent detection categories we hope to improve through our research.

Real-time Detection The detection of a bad event, b, occurs while the system is operating and before events dependent on b occur, requiring the order

$$t_b < t_{D(b)} < t_c \tag{1}$$

Near Real-time Detection The detection of a bad event, b, occurs within some predefined time step δ, either before or after t_b.

$$|t_b - t_{D(b)}| \leq \delta \tag{2}$$

Kuperman comments that this timeliness categorization should be independent of the underlying hardware and the rate of event occurrence. Although this goal is desirable for a software-based solution, it relies on assumptions of trustworthiness and lack of vulnerabilities in this underlying hardware. With today's computer hardware this independence is unobtainable. Rutkowska's attack, discussed in Sec. 3, provides a specific example of why hardware cannot be blindly trusted. If hardware cannot automatically be trusted it *must* be considered in security measurements.

An often overlooked aspect of a computer security monitor is the security of the monitor itself. This security is a critical aspect of a security system, since compromising the monitors can effective render the security system useless. Mott presents a classification of the security of the monitors creating eight levels of monitoring system security [5] presented here.

Open This worst case scenario occurs when the monitored system has knowledge of the monitor and shares information with the monitor without any security mechanisms present.

Soft Security This level of monitor security is equivalent to *open* with software used to secure the monitor. Both of these levels tend to contain monitors on a uniprocessor host-based intrusion detection system.

Passive Security The monitor operates without the monitored system necessarily knowing it is there. To compromise such a system, information about how the monitor analyzes gathered state data must be known. Prime examples of this level of security include most network Intrusion Detection Systems (IDSs) where only network traffic is monitored. Specific information passed over the network has the potential to disable the system, but there are no direct avenues of attack.

Self Security Similar to both *open* and *soft* security systems, the monitored system shares information with the monitor. The manner in which the monitor operates provides it with security, requiring the monitored system to be compromised before the monitor can be compromised. An example of this level of security is Williams' CuPIDS [16].

Loose-hard Security The monitored system again has knowledge and coordinates with the monitor, sharing information, but dedicated hardware mechanisms protect key portions of the security system from compromise such as with hardware-based return address stacks [19].

Semi-hard Security The monitored system's knowledge of the monitor is extremely limited. To provide this level of security the monitor cannot execute on the same processor core as the monitored software and communications happens through mechanisms like unmaskable interrupts that are kept to a minimum. Compromise can only occur via code controlling synchronization signals to the monitor, which would cause the monitor to operate in a diminished capacity.

Strict-hard Security This security level adds to the requirements of *semi-hard* security by requiring only hardware connections to the monitor and removing synchronization signals to the monitor. The monitor must be able to gather its own state information to remove dependence of the monitor on the monitored system. Two examples of this level of security are CoPilot [6] and Independent Auditors [4].

Complete Security This level of security is the ideal secure case, used as a theoretical comparison point. In reality, such a monitoring system would have no contact with the production system, negating it's usefulness.

Mott notes that with many of these levels of security, there is a tradeoff between the security of the monitor and the ease with which state information

can be gathered from the monitored system [5]. One critical piece of information overlooked by these categories is the trustworthiness of the information that the monitor is receiving. Although technically the monitor itself is not corrupted, the effects can be equivalent. For example, a Supervisory Control And Data Acquisition (SCADA) System controlling critical infrastructure such as the electrical grid, could be manipulated to perform undesirable actions, without ever compromising the SCADA System. This can still be accomplished by an attacker who can only manipulate the information being received by the SCADA System. For instance, if an attacker can manipulate the information feeding the SCADA System, telling it that there is a massive overdraw on the electrical grid, they can affect SCADA System responses such as causing a rolling blackout. This is accomplished without specifically corrupting the SCADA system to do so. The SCADA System would respond correctly to the environment it believes exists, not the actual environment. A simpler exploit corrupting the information being passed to monitors is a denial of service (DoS) attack. If the SCADA system does not receive readings from sensors monitoring critical sections of the system, it will be unable to respond to parameters out of acceptable ranges. This could quickly compound into catastrophic failure.

Although this issue is acknowledged in a number of works [6, 16, 17], we have not found research that investigates this aspect. Our research explores this aspect of the monitor's security. Rutkowska presents methods for corrupting the memory access of the PCI Bus without affecting the processor's access to memory [20] which is discussed in more detail in Sec. 3. This exploit highlights the importance of this aspect of classification for the security of the monitoring system. CoPilot [6], one of the examples Mott identifies as being *strict-hard* security, is defeated by this attack because of its security weakness on this new axis of categorization. We present an independent axis for categorizing the security of the monitor relating to the trustworthiness of the monitored data: how far removed the monitor is from what it is monitoring. This classification is defined in Sec. 4.3.

3 Defeating Hardware-based RAM Acquisition

As the previous section began to develop, the ability to falsify the information a security monitor receives corrupts the integrity of the security system. One prominent example of this exploitation is Rutkowska's defeat of hardware based random access memory (RAM).

Rutkowska discusses both software and hardware approaches to memory acquisition with the claim that the hardware based approaches are superior to that of software based solutions [20]. She cites non-persistent malware as motivation for needing memory acquisition and she presents a number of known exploits of software memory acquisition by code running at the same privilege level as the acquisition software. One specific example of such an exploit is the FU Rootkit [21]. Rutkowska notes that these software memory acquisition tools require additional software on the target machine, which she claims violates the

forensic tool requirement not to write data to the targeted machine. She then extols the virtues of hardware based solutions, setting her readers up for her defeat of this "superior" memory acquisition method.

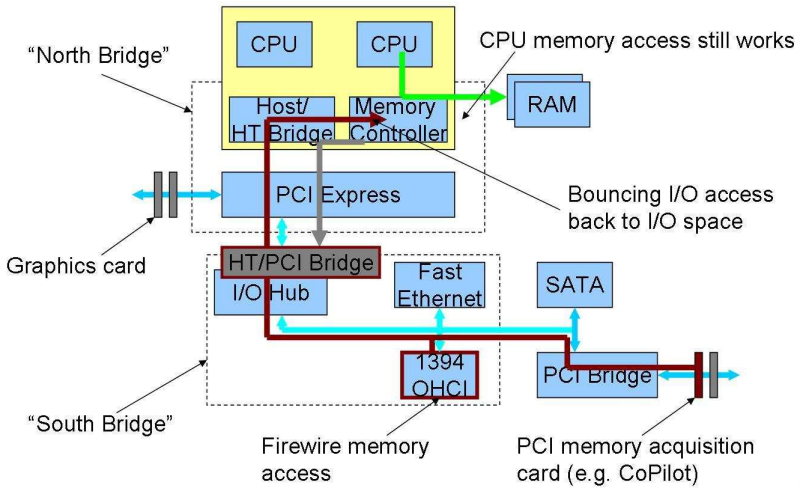

Fig. 1. Rutkowska's Defeat of Hardware Based RAM Acquisition [20]

Rutkowska delivers three levels of compromise to hardware based memory acquisition devices such as CoPilot [6] and Tribble [1]; each building upon the same basic exploit with increasing levels of damage. This exploit, depicted in Fig. 1 involves configuring the north bridge on a system to map arbitrary ranges of physical memory to I/O space. This remapping denies memory access to peripheral devices for the specified physical memory range while not affecting the memory access of the processor(s). This allows an exploit to execute correctly, while hiding the exploit's presence from the hardware-based acquisition tool. These levels range from a denial of service to an attack that provides the monitor with false data, completely masking the compromise from the monitor.

The exploits Rutkowska presents show definitively that current hardware based memory acquisition devices, such as those that plug in to a firewire port or as a PCI device, are not reliable. The lesson to be taken from her work is not that hardware cannot do a better job of providing security features, rather that hardware is not a magic bullet; it does not automatically improve security. This work highlights that many current hardware solution are missing an important aspect of the capability and security of the monitoring system. This provides substantial motivation to explore the trustworthiness of the information being received by a security monitor. This critical axis of security for a monitor, though acknowledged in numerous works [1, 6, 16, 17] is not well understood and not clearly defined. Section 4.3 provides definitive categorization along this axis to

aid future work in security related fields understand what is required to provide truly reliable security monitoring.

4 Why Hardware?

Most current security systems for computers are based largely on software systems. Numerous flaws and vulnerabilities have been exposed and even exploited in these different software solutions. Compromise of protected code via rootkits [22] represents one of the most prevalent exploits. Recent work has begun exploring different hardware based approaches to security [5, 6, 12] with many people coming to believe that we cannot solely use software to protect software and only hardware, coupled with software, can do that job successfully [1–13]. Though a number of advantages to hardware over software have been suggested, we found no research discussing what precisely makes hardware a significant improvement over software and just what capabilities hardware provides that software cannot. We present here a number of key advantages achievable through the use of hardware.

Reduced Avenues of Attack Separate monitoring hardware can strengthen the security of the monitor by reducing the extent of the coupling between the security and production systems.

Trustworthiness of Information Correctly designed hardware guarantees that the monitor receives valid data from the production system, something that we show software incapable of doing.

Additional/Different Information Available Mott's research explores a number of pieces of information that can be gathered through hardware primitives and leveraged to increase the overall security of the system [5]. These hardware primitives include information such as the program counter, instruction traces, and added visibility into memory.

Timeliness of Detection The ability to guarantee real-time detection, as defined in Sec. 2, requires the ability to guarantee that the monitor will execute with the ordering of (1). Dedicated monitors are necessary to accomplish this.

In the rest of this section we develop justification for needing capabilities beyond what software can provide and explore each of these advantages in greater detail. We define what is required of hardware to overcome the vulnerabilities of software and provide significantly improved performance.

4.1 Vulnerabilities of Software Security Systems

There are a number of vulnerabilities inherent in software-based security. Two critical vulnerabilities are the inability to guarantee real-time monitoring in standard commercial operating systems, even on a multiprocessor system, and the inability to protect the integrity of the security system once the production

system has been compromised. The first vulnerability is evidenced by the fact that scheduling of processes on both uniprocessor and multiprocessor systems does not make any guarantees on precise ordering or timing of when a specific process gets time on a processor. Work such as CuPIDS changes this standard paradigm to guarantee monitored processes run in lock step with the monitoring process [16] and overcome this first critical vulnerability of software security systems. Despite CuPIDS' ability to overcome this vulnerability, it cannot protect itself once the kernel has been compromised.

The specific point where software loses the ability to protect other software is when faced with exploitation of a vulnerability in privileged code. Once an attack can gain access through such a vulnerability, they have access to any piece of software in the system and can modify both data and executable code. This allows for changes in both user applications and the operating system itself, compromising the security of the security system itself. This can be accomplished through modification to the security software itself or by modifying the operating system to interact with the security software in another manner, such as reducing its privilege level. Note that exploitation of vulnerabilities in privileged code provides two main avenues of attack into the system. The more obvious method of attacking the security software itself is to degrade or interrupt its capabilities described above. The other avenue of attack is corrupting the information that is being sent to the security software.

This second issue is the general method that rootkits use to remain undetected. They interpose themselves between processes by taking control when there is a library function or system call. By controlling what information is passed back to the monitoring process the rootkit can neutralize the security software without directly modifying it [22].

4.2 Advantages of Hardware

The vulnerabilities of software discussed above show clear need for a security solution that can overcome these vulnerabilities. Does hardware provide protection from these attacks? Not necessarily. Hardware can provide increases in protection, but only if appropriately designed into the system's architecture. Two key factors in designing hardware that can enhance these areas of security are where we connect the security hardware to the system and how we make those connections. Where we connect controls the *Trustworthiness of Information* as well as influences the *Timeliness of Detection*. The next two subsections explore these advantages in greater detail. How the security hardware is connected impacts the amount of information available to the security system and defines the only avenues of direct attacks on the security system. By limiting the physical pathways between the production system and the security system to specific hardware primitives, the attack surface is significantly reduced. These primitives can also provide access to key information which is unobtainable via software-based solutions. Both aspects of hardware primitives are discussed in Sec. 4.5.

4.3 Trustworthiness of Information

Although the need for the monitor to receive accurate data is understood, there is no real framework for understanding what precisely is needed to accomplish this. Towards this end we define a new axis categorizing the trustworthiness of the information being received by the monitor. This axis of trustworthiness stands as its own contribution and should be considered when attempting to provide an accurate, secure monitoring device of any sort. By creating this categorization we set important bounds on what exactly affects the trustworthiness of the information.

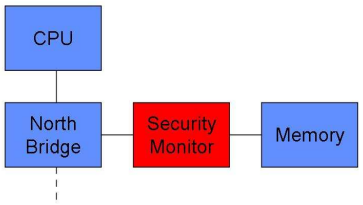

Fig. 2. Immediate Information: Security Monitor placed inline between main memory and the memory controller.

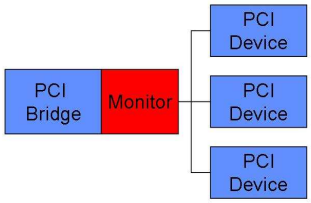

Fig. 3. First-hand Information: Security Monitor placed on a shared bus, vulnerable to Denial of Service from excessive device traffic.

Immediate Information (Fig. 2) With immediate access to what is being monitored we can insure the monitor is receiving true data. This immediate categorization represents a specific form of first-hand information where the monitor is inline, directly between what is being monitored and its interaction with the system. While this level of trustworthiness is certainly the most definitive method for ensuring the monitor's security, it leads toward a design with individual monitors on every single hardware component, thus requiring a complete redesign of all aspects of a system.

First-hand Information (Fig. 3) This level of trustworthiness represents a monitor that has direct access to the data being output from some device. Depending on the specific design of the architecture being monitored, this level of trustworthiness will likely be equivalent to *Immediate Information*. However, a shared bus architecture could be vulnerable to a denial of service (DoS) exploit. This would be accomplished in much the manner that someone would have trouble listening to another's conversation in a crowded room.

Second-hand Information This level of trustworthiness encompasses any monitor that relies on some intermediary mechanism, such as hardware or software components, to pass it the data it is attempting to monitor. Although each additional mechanism relied upon reduces the trustworthiness into third-hand information and so forth with a continually lessening level of

trustworthiness. For simplicity we group all levels of trustworthiness that cannot guarantee accurate monitoring into this category of second-hand information. Unless any and all mechanisms being relied upon to pass the monitor data can be guaranteed secure, this presents an avenue of attack for corrupting the monitor be feeding it false data. Figure 1, on page 5, shows a PCI-based memory acquisition tool, such as CoPilot [6], that must trust the PCI bridge, the south bridge, and the north bridge; trust which Rutkowska's research demonstrates as unwarranted [20].

It is this previously undefined axis of the monitor's security that is being exploited by Rutkowska's attack. Our research defines the requirement to protect against this attack: monitors must be capable of receiving at least *First-hand Information*. Two important things to note about this axis of security are that 1) all software based security systems on a uniprocessor system are inherently unable to achieve a level of trustworthiness better than *Second-hand Information* since they must rely on data controlled by the operating system and 2) even software based solutions designed to operate within a multiprocessor system, such as CuPIDS [16], must still rely on the trustworthiness of main memory and therefore receive no better than *Second-hand Information*. In order to ensure accurate monitoring, the monitor needs to have access to at least *First-hand Information* of the data being produced, any intermediate devices provide the possibility of the data being manipulated before reaching the monitor. Therefore at very least we need monitoring or interaction points at each of the bridges in the system, i.e. any device that passes information from one part of the system to another.

4.4 Timeliness of Detection

Another aspect of monitor placement is the speed with which a monitor can detect an attack. One of the areas where the speed of a device far exceeds the speed of the buses that pass information to and from it is the processor(s). To accomplish real-time monitoring as defined in Sec. 2, monitors will need to be closer to the main processor than system bridges will allow. One such example of this is a hypothetical purely cache based attack [6]. Such an attack will be able to do its damage before detection, since detection is only possible with access to a present view of cache. Even if we accept near real-time monitoring capabilities, Kuperman's δ value in (2) will be significantly smaller for a monitor that is located on-chip.

4.5 Hardware Primitives

The manner in which we connect monitors to the system plays a significant role in enhancing both the security of the production system and the security of the security system. By limiting connections between the monitor and production systems and remaining within Mott's *Semi-hard* security level the only avenues of directly attacking the security system are the hardware primitives that bridge

the monitors and production system. As long as no primitives allow for modification of the monitoring system's code, we maintain a greatly reduced attack footprint for the security monitor. At the same time, these hardware primitives can offer direct access to information previously difficult to obtain and even provide access to information not accessible through any software methods. Mott presents a number of hardware primitives that can be leveraged in [5]. The two main areas of interest for creating hardware security (and security in general) have been attempts to monitor processes running on the production system, mainly through various memory introspection techniques [1–3], and monitoring the incoming network traffic as it enters the system [8–13].

4.6 What Do We Mean By Hardware Security?

To this point we have left the definition of hardware security somewhat up in the air. All computer systems contain a mix of hardware and software and only a limited amount is accomplished with purely hardware. To create a security system purely in hardware would significantly hamper the flexibility and modifiability of such a system reducing the number of future attacks to which a system could potentially respond. Solutions such as a field programmable gate array (FPGA) can be used to extend software flexibility into hardware, though it does require performance tradeoffs and is not pivotal to this aspect of our discussion. However, a pure hardware solution is not our goal when we talk about hardware-based security. The key component of hardware-based security is the *communication between the production system and the security system.* Whether a specific monitor is pure hardware, a FPGA, or software running on some combination of hardware that remains separate from the production system hardware, what qualifies a security component as hardware-based is that connection back to the production system. Note that an important result of this definition is that a hardware-based security solution requires physically separated memory. This is not to say that pure hardware or at least FPGA solutions will not be required in some instances to provide fast enough response. Areas where high-speed detection is crucial will almost certainly benefit from pure hardware solutions. One predominant example of this is the network IDS field where research has shown benefits from hardware solutions [10, 11, 13, 23].

4.7 Hardware/Software Interaction

With the key component of using hardware being the communication between the production system and the security system, software can be employed on a separate security processor. This allows a full-fledged software security operating system to run on such a dedicated security processor. Mott et al. explore this interaction, pointing to the hardware monitors as decoupling production and security software [24]. This software can perform management and communication roles between elements of the security system so long as there is no access to modify the software via the production system. With the inclusion of dedicated

security system I/O, via some variant of a communications port, the software can be modified and updated as needed to respond to future threats.

5 Specific Requirements for Achieving Benefits from Hardware

So far we have discussed the different advantages of dedicated hardware for security solutions and discussed what is required to achieve these advantages. Here we explicitly define these requirements for dedicated hardware. By designing to these requirements, it is possible to design a comprehensive security solution that achieves the advantages of hardware previously explored. These requirements are:

First-hand Information of all monitored information: This level of trusted information guarantees accurate monitoring of what is happening in the system. Without this level of trusted information security solutions are vulnerable to being denied access to the information or even fed false information. This vulnerability provides a route to compromise the effectiveness of the security system, without the need to compromise the security system itself.

Dedicated Monitors for parallel, concurrent monitoring: To protect against potential timing attacks monitors must be able to run concurrently with what they are monitoring to allow the possibility guaranteeing of Kuperman's *real-time detection* [17]. Any monitor which does not run concurrently with its target must ensure that it runs often enough to be impervious to timing attacks. In a software-based solution this becomes infeasible due to the performance penalty of continuous context switching. Dedicated hardware monitors remove the burden on production resources and keep performance degradation to a minimum [7].

Explicit Hardware Communication between the production and security systems: By limiting communication between the production and security systems to hardware pathways, we reduce avenues of attack upon the security system to these explicitly defined pathways. Without modifiable communication pathways, the ability to corrupt these pathways is reduced. These limited pathways provide a clear set of attack avenues which can be understood and protected.

Dedicated Storage of security code and data: Without dedicated, separate security storage we leave software communication pathways present in the system. These communication pathways represent a significant avenue of attack to be exploited. Any software-based separation becomes vulnerable to a root-level compromise of the production system. Separate storage which cannot be directly modified by the production system provides a more reliable method of protecting the security code and data.

Dedicated Security Processor for controlling and coordinating the security mechanisms: Though not explicitly a requirement for gaining security capabilities, a dedicated security processor is included here for the coordination

and communication abilities it can provide. This separate processor will allow for a secured security control center when coupled with these other requirements. It will provide the ability to modularly add security mechanisms into a security backplane. An important aspect of this ease of modularity is the ability to combine both network IDSs and host-based IDSs into a combined, complete IDS which can leverage combined knowledge from each to provide more flexible and effective response.

6 Conclusion

The use of hardware is necessary to provide quality security solutions. Short of verifying the trustworthiness and security of all software and hardware mechanisms in the chain from the monitored information back to the monitor, *First-hand Information*, that requires dedicated hardware to achieve, is the only way to guarantee the security monitor is not fed false data. As computer security systems become more reliant on dedicated hardware, the need for a clear understanding of the necessary design requirements to overcome inherent software security vulnerabilities is essential. This work provides a basis for this understanding by defining the advantages that can be gained from hardware, and the necessary design to achieve them.

References

1. Carrier, B.D., Grand, J.: A hardware-based memory acquisition procedure for digital investigations. Digital Investigation, **1** (February 2004)
2. Özdoganoglu, H., Vijaykumar, T.N., Brodley, C.E., Kuperman, B.A., Jalote, A.: Smashguard: A hardware solution to prevent security attacks on the function return address. IEEE Transactions on Computers **55** (2006)
3. Gordon-Ross, A., Vahid, F.: Frequent loop detection using efficient non-intrusive on-chip hardware. In: CASES '03: Proceedings of the 2003 international conference on Compilers, architecture and synthesis for embedded systems. (2003)
4. Molina, J., Arbaugh, W.: Using independent auditors as instrusion detection systems. Information and Communications Security: 4th International Conference (December 2003)
5. Mott, S.: Exploring hardware-based primitives to enhance parallel security monitoring in a novel computing architecture. Master's thesis, Air Force Institute of Technology (March 2007)
6. Petroni, N.L., Fraser, T., Molina, J., Arbaugh, W.A.: Copilot-a coprocessor-based kernel runtime integrity monitor. Proceedings of the 13th USENIX Security Symposium (2004) 179–194
7. Williams, P.D., Spafford, E.H.: Cupids: An exploration of highly focused, co-processor-based information system protection. Computer Networks, **51** (April 2007)
8. Song, H., Lockwood, J.W.: Efficient packet classification for network intrusion detection using FPGA. In: FPGA '05: Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays. (2005)

9. Yi, S., koo Kim, B., Oh, J., Jang, J., Kesidis, G., Das, C.R.: Memory-efficient content filtering hardware for high-speed intrusion detection systems. In: SAC '07: Proceedings of the 2007 ACM symposium on Applied computing. (2007)
10. Gonzalez, J.M., Paxson, V., Weaver, N.: Shunting: a hardware/software architecture for flexible, high-performance network intrusion prevention. In: CCS '07: Proceedings of the 14th ACM conference on Computer and communications security. (2007)
11. Hutchings, B.L., Franklin, R., Carver, D.: Assisting network intrusion detection with reconfigurable hardware. FCCM '02: 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines **00** (2002)
12. Hart, S.: APHID: Anomoly processor in hardware for intrusion detection. Master's thesis, Air Force Institute of Technology (March 2007)
13. Bu, L., Chandy, J.A.: FPGA based network intrusion detection using content addressable memories. In: Proceedings - 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM 2004, IEEE Computer Society, Los Alamitos, CA 90720-1314, United States (April 2004)
14. Bazaz, A., Arthur, J.D.: Towards a taxonomy of vulnerabilities. Hawaii International Conference on System Sciences (2007)
15. Axelsson, S.: Intrusion detection systems: A survey and taxonomy. Technical report, Chalmers University of Technology (March 2000)
16. Williams, P.D.: CuPIDS: Increasing Information System Security Through The Use of Dedicated Co-Processing. PhD thesis, Purdue University (August 2005)
17. Kuperman, B.A.: A Categorization of Computer Security Monitoring Systems and the Impact on the Design of Audit Sources. PhD thesis, Purdue University (2004)
18. Stakhanova, N., Basu, S., Wong, J.: A taxonomy of intrusion response systems. Technical Report 06-05, Department of Computer Science, Iowa State University (2006)
19. Lee, R.B., Karig, D.K., McGreggor, J.P., Shi, Z.: Enlisting hardware architecture to thwart malicious code injection. In: Lecture Notes in Computer Science: Security in Pervasive Computing. (2004)
20. Rutkowska, J.: Beyond the CPU: Defeating hardware based RAM acquisition (February 2007) `http://invisiblethings.org/papers.html`.
21. FU Rootkit: `http://www.rootkit.com/project.php?id=12`.
22. Levine, J.; Grizzard, J.O.H.: A methodology to detect and characterize kernel level rootkit exploits involving redirection of the system call table. Information Assurance Workshop, 2004. Proceedings. Second IEEE International (April 2004) 107–125
23. Tummala, A.K., Patel, P.: Distributed ids using reconfigurable hardware. In: 21st International Parallel and Distributed Processing Symposium, IPDPS 2007, Institute of Electrical and Electronics Engineers Computer Society, Piscataway, NJ 08855-1331, United States (March 2007)
24. Mott, S., Hart, S., Montminy, D., Williams, P., Baldwin, R.: A hardware-based architecture to support flexible real-time parallel intrusion detection. Proc. 2007 IEEE International Conference on System of Systems Engineering (2007)