Mastering Security Anomalies in Virtualized Computing Environments via Complex Event Processing

Lars Baumgärtner, Pablo Graubner, Matthias Leinweber, Roland Schwarzkopf, Matthias Schmidt, Bernhard Seeger, Bernd Freisleben
Department of Mathematics and Computer Science, University of Marburg
Hans-Meerwein-Str. 3, D-35032 Marburg, Germany
{lbaumgaertner,graubner,leinweberm,rschwarzkopf,schmidtm,seeger,freisleb}@informatik.uni-marburg.de

***Abstract*— To protect computer systems and their users against security attacks, all potential security related incidents should be detected by monitoring system behavior. In this paper, a novel approach to detect, analyze and handle security anomalies in virtualized computing systems is presented. Adequate sensors on different virtualization layers monitor relevant events, a Complex Event Processing engine is used to aggregate and correlate events on the same or different layers to find genuine attacks and eliminate false positives, and corresponding actions are performed if a security anomaly is detected. To enhance the quality of the results, machine learning techniques are used to analyze a historical database of recorded events offline to generate new or modify existing queries on the monitored event stream automatically. Furthermore, sensors can be activated and deactivated during runtime to gather interesting events, reduce the false alarm rate and ensure the system's responsiveness when a sudden increase of monitored event data occurs. In this way, a flexible, minimally-invasive approach for detecting, analyzing and reacting to a broad variety of security anomalies in a virtualized environment is provided.**

***Index Terms*— security; malware; virtual machine monitoring; complex event processing; intrusion detection.**

I. Introduction

Computers exposed to the Internet are at constant risk of being attacked. To protect them against security attacks, all security related incidents should be detected by monitoring system behavior. To detect security anomalies, Intrusion Detection Systems (IDS) or Intrusion Prevention Systems (IPS) are typically used; their combination is known as Security Information and Event Management (SIEM). However, most SIEM systems only monitor events on the infrastructural layer, need human assistance in case of error recovery, raise a high number of false alarms, and do not scale well with an increasing number of events.

In this paper, a new approach to detect, analyze and handle security anomalies is presented. The anomalies include both known and yet unknown security vulnerabilities with a particular focus on systems based on operating system virtualization, such as Infrastructure-as-a-Service Cloud computing systems. Hence, this paper proposes a novel SIEM system especially for virtualized computing resources.

The proposed architecture monitors security anomalies on different layers of a virtualized computing system. Monitoring is based on installing adequate sensors in the hypervisor (also called virtual machine monitor), in a virtual machine itself and in any kind of application runtime environment, such as an web-application container, to continuously report all relevant activities. A combination of out-of-VM monitoring using virtual machine introspection [6] and in-VM monitoring is used to keep the usual monitoring overhead low. To facilitate horizontal and vertical correlation and aggregation of monitored events, Complex Event Processing (CEP) is used. CEP enables robust cross-layer monitoring by leveraging Event Processing Agents (EPAs). EPAs are continuous queries on event streams that are able to analyze basic events and look for security anomalies. Based on the gathered results, the system can react autonomously and intelligently, i.e., it is able to repel attacks and circumvent security vulnerabilities. Occurring anomalies, even on different layers, will be detected at an early state and appropriate actions are launched. To enhance the quality of the results, machine learning is used to analyze archived, offline data. This allows us to generate new EPAs automatically through behavioral models derived from a historical database of recorded events. Furthermore, it is possible to activate and deactivate sensors during runtime in order to gather interesting, individual events, eliminate false positives and keep the system responsive when a sudden increase of monitoring data occurs. n this way, a flexible, minimally-invasive approach for detecting, analyzing and reacting to a broad variety of security anomalies in a virtualized environment is provided.

The core functionality is maintained in a special, trusted virtual machine (called ACCEPT-VM). It receives and processes all sensor data and triggers actions if necessary. In order to increase the processing speed and handle a larger amount of events, the benefits of multi-core architectures are leveraged, thus, EPAs can be scheduled between CPU cores. Furthermore, to make use of intra-EPA-parallelism, it should be possible to offload certain EPAs to General Purpose Graphic Processing Units (GPGPUs).

This paper is organized as follows. Section II discusses related work. Section III presents the approach for detecting, analyzing and handling security anomalies in virtualized computing systems. Section IV presents examples. Section V concludes the paper and outlines areas for future work.

 ISBN: 978-1-61208-181-6

II. Related Work

Teixera et al. [11] present Holmes, an implementation of a monitoring solution for integrating a CEP engine with machine learning. The CEP engine generates alerts using hand-crafted continuous queries to detect known abnormalities and deviations from expected behavior. Furthermore, it normalizes the asynchronous events for analysis with the machine learning algorithm, i.e., it joins different streams to be analyzed together and generates time series with equidistant intervals. A machine learning algorithm detects unknown anomalies in time series, without manual rule creation and anticipation of problem conditions and thresholds.

Holmes utilizes infrastructure level sensors and can thus only detect hard- and software issues as well as attacks such as Distributed Denial-of-Service (DDoS) attacks. The proposed architecture is not hierarchical, i.e., it consists of a single message bus, where all sensors publish their information to and the central CEP engine and machine learning modules subscribe to. This architecture does not scale well, neither for an increasing number of events nor for an increasing number of machines to monitor. Historical data is not used for anomaly detection, which limits the potential to detect anomalies as well as increases the risk of false positives.

Ficco [5] presents an approach to detect and respond to attacks by using event correlation. The approach is described using a DDoS attack as an example. Different information sources on several architectural levels, such as network, operating system and application, are deployed in strategic points of the system. In the example, these sources are the number of connections from a single IP, the length of the backlog queue of TCP and the number of application requests. Agents deployed together with the sensors analyze, filter, normalize and forward messages to the so called Decision Engine, consisting of a correlator, a diagnoser and a reaction module. Specialized modules, called Remediators, are used to remediate a specific attack or intrusion. An ontology is used to map all symptoms and possible effects of an attack. This ontology is used for the correlation of events and the decision about the right remediation strategy.

Although the proposed solution uses sensors on several architectural levels, it is targeted mainly at detecting different types of DoS attacks. The detection is based on the information about known attacks stored in the ontology. Detection of unknown anomalies is not possible with this solution. Since a central decision engine is used, scalability is also a problem of the architecture for growing network size or an increasing number of events. Finally, historical data is not taken into account in the detection process, missing another opportunity to eliminate false positives.

Cugola and Margara [3] present research about low latency CEP and general purpose GPUs. The work is based on the T-Rex CEP Engine and TESLA [4] as the language for defining rules. The authors assume that there are two major approaches for complex event processing: an automaton and a column based approach. Their main goal is to evaluate performance differences between them. Furthermore, they additionally compare them with a GPU variant. Automaton-based Incremental Processing (AIP) is the algorithm used to translate a CEP rule into a linear, deterministic finite state machine, which is fed with the incoming events while temporal results are stored. The counterpart approach is based on Column-based Delayed Processing (CDP), where events become more or less replicated for each rule and are stored in a column based structure. This algorithm is also used as a basis for their GPU implementation.

In all of their test setups, it is obvious that a CDP approach outperforms automaton based processing. They also implemented the CDP approach with CUDA on nVidia Graphics cards. Their first test of the GPU implementation results in a speedup of 25 compared with CPU CDP. Their evaluation leads to an average speedup of 40 with their hardware configuration. Additionally, their results are very varying depending on the particular configuration. For example, a large window size in which events are aggregated can lead to a speedup of 100. Their closing recommendation is to use GPU aided CEP only for large and complex rules, because there is a tradeoff between speedup and the overhead generated by using this technology.

Gorton [7] argues that the usage of a diversity of sensors on several architectural levels raises the chance to detect an attack, because the sensors may reinforce each other. However, this requires to manage and correlate the higher number of events and alerts. Different solutions have been developed in the area of intrusion correlation, targeted to the reduction of alerts a security officer must address. The potential to detect anomalies using these different information sources, however, is not the focus of these solutions.

III. Proposed Architecture

To detect attacks in a virtualized computing environment, it is useful to know as many anomalies in the behavior of the system as possible. Theoretically, every event that occurs in one of the different layers of a virtualized system can be an indicator for an anomaly: for example, established network connections, creation or termination of processes or even user or process activities beyond regular working hours.

The decision about what is a normal or unknown system behavior cannot be made by the sensors of a monitored environment. Instead, a CEP engine is responsible for processing all the informations sent by the sensors and is then able to decide what can be viewed as normal system behavior. Through dynamic deployment of further sensors, it is possible to eliminate false positives and verify findings.

Therefore, the architecture of the anomaly management system consists of a secure and trusted virtual machine (called ACCEPT-VM), where the main analysis components of the system such as the Sensor Management, the Matchmaker, the CEP engine and various databases are located. Furthermore, a set of passive sensors and actors reside on every layer (hypervisor, operating system and application layer) of each virtual machine. All of these sensors continuously deliver a stream of information to the ACCEPT-VM, and each actor is able to execute a specific set of actions on its corresponding layer in order to respond to any detected problem.

 ISBN: 978-1-61208-181-6

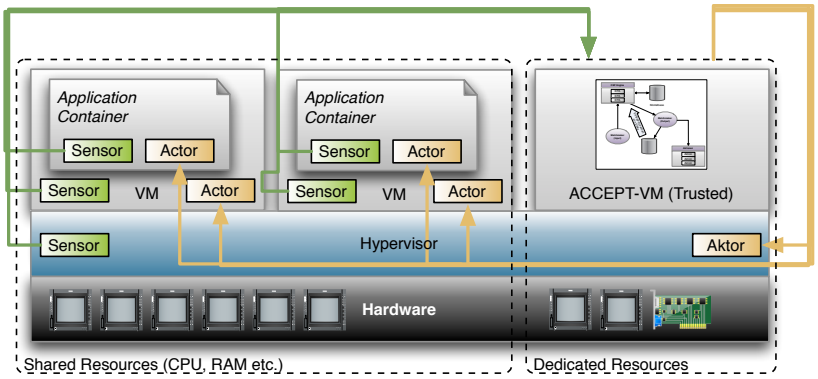

Fig. 1. Sensors and Actors in a monitored environment.

A. Monitored System

Sensors are deployed on several layers: The hypervisor, the operating system and application containers (see Figure 1). All sensors and actors are equipped with their required privileges related to the corresponding layer, and their implementation is aimed to be minimally invasive with respect to the normal functionality.

All information sent by sensors and received by actors contain important system information, which can be misused by an intruder. This is the reason why both the communication between sensors and the secure and trusted ACCEPT-VM as well as the communication between the ACCEPT-VM and the actors are secured in terms of authentication of the communication partners and the integrity/secrecy of the messages sent via the communication channels.

1) Sensors: There are several possibly interesting metrics to be gathered on each layer. Some of the expected events to be gathered can be found in the following list:

- Hypervisor level: Network traffic, system calls from within VMs, process lifecycle information.
- Operating system level: File access, network sockets, resource utilization.
- Application level: JVM information (heap utilization, thread count, library calls)

Sensors monitor traditional characteristics of resource usage (e.g., percentage of processor usage or memory consumption) as well as all data created by all processes in a system, such as system calls, network traffic, read/write memory access. This low level information greatly enhances the chances to detect more sophisticated attacks such as malicious polymorphic code, hidden processes or ongoing memory corruption exploits. On the hypervisor level, virtual machine introspection is used for acquiring monitoring data; on the operating system and application container level, the sensors are running as privileged user processes. Their security, as well as the security of the actors are ensured by hypervisor introspection. The hypervisor can monitor the running sensors and their process memory to guarantee that they have not been manipulated.

Sensors installed on the application level are used to monitor application behavior. This could be an application container such as Tomcat or JBoss or a bare Java Virtual Machine (JVM). Metrics gathered by these sensors are, e.g., changes of the memory heap, number of threads, number of Java classes, libraries or garbage collector statistics. Events occur if the code flow accesses constructors, methods and variables.

2) Actors: Actors are also installed across all layers, enabling direct countermeasures at the appropriate levels. Examples for such actions can be found in the following list:

- Hypervisor level: Start, stop or pause a virtual machine. Block or shutdown network interfaces.
- Operating system level: Start, stop, terminate processes or network connections. Delete users or files.
- Application level: Launch the garbage collector, solve deadlocks. Relaunch, terminate the application container or even remove components from the latter.

Further actions include a migration of a compromised virtual machine from the productive network to a separate honeypot network in order to detect possible malware. Since actions can be triggered on all layers of the virtualized system, they must be specified in a flexible, multi-purpose way. Therefore, an easy-to-use scripting language such as JavaScript is used. Actions can be executed concurrently on a target system, with the constraint of being executed in isolation, consistently and completely to avoid interferences or unknown system behavior. To increase the expressiveness and the usability of specified actions, actions are able to view all the data monitored by the sensor of the corresponding level.

B. ACCEPT-VM

The ACCEPT-VM consists of the CEP analysis engine, a Matchmaker, a Sensor Management, a Model Database and a Historical Database (see Figure 2). The latter is stored on a dedicated server in a data warehouse. The ACCEPT-VM is a trusted virtual machine in the sense that its attack surface is minimized: The number of services is reduced to an essential set, Mandatory Access Control is implemented and integrity checks are run on the file system. Existing security approaches, such as AppArmor [2] or SELinux [8], Tripwire [13] or AIDE [1] are used. Communication with this special virtual machine avoids direct TCP/IP traffic to the other virtual machines. A virtual device within the hypervisor is responsible for all message passing between sensors and actors. This has the advantage that the ACCEPT-VM cannot be directly attacked from the network or through a compromised host.

 ISBN: 978-1-61208-181-6

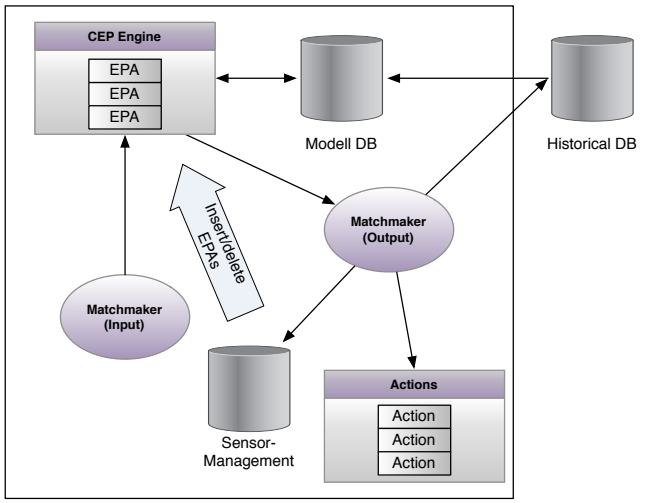

Fig. 2. Architecture of the ACCEPT-VM.

Furthermore, it provides a faster interface to pass information than classic socket communication.

The capability of performing analysis on a large amount of data on the input stream coming from the sensors is ensured by two main approaches: First, the EPAs running in the CEP engine are supported by pattern matching techniques. Second, the inherent control- and data-parallelism of EPAs are used to increase the number of events that can be processed. Furthermore, today's GPGPU technology is used to process a higher degree of events in parallel.

1) Complex Event Processing Engine: The analysis of all occurring events is performed using the CEP engine developed by the Software AG [10]. Event Processing Agents work as continuous queries on the different event streams coming from the sensors. These EPAs are verbalized in a SQL-like language and can combine information from different streams, thus different layers. Even mixing dynamic stream based queries with static data gathered, for example, from the Model Database (see III-B.2) is possible. Based on the results of the EPAs, actions are launched in order to react to a detected anomaly.

2) Model Database and Historical Database: The Model Database is built upon the information provided by the Historical Database. The Historical Database acts as a giant data store and foundation for model generation. It stores all events and actions generated by the sensors and the EPAs. Due to the high number of events generated by the sensors, it is necessary to develop new strategies to reduce the size of data. Nevertheless, it is necessary that the events and actions get recorded so that a root cause analysis is possible. The historical data can also be used to create simulations within a sandbox and replay certain scenarios. This enables both "what-if" analyses and the possibility to evaluate the effectiveness of EPAs.

With the data contained in the Historical Database, the Model Database can be created. This database contains the default behavior of a computer system expressed in statistical models. For example, a model regarding the average time of a TCP session to a web server can be created, and an EPA can be installed to detect abnormally long TCP sessions. Such a long open session might indicate a successful penetration of the web server with a shell running through the socket instead of short HTTP requests. The data from the Model Database can be used by the EPAs to detect essential anomalies differing from regular system behavior which again can trigger adequate actions.

3) Matchmaker: The Matchmaker component consists of two parts: The Input Matchmaker and the Output Matchmaker. The Input Matchmaker is responsible for a fully automatic interconnection between a sensor and an EPA. For example, if an EPA wants a specific input, then a request is sent to the Input Matchmaker. The Input Matchmaker is aware of the position of every sensor, including its meta-data, i.e., a flexible description of sensors and their possible actions. Analogously, the Output Matchmaker serves as a mediator that forwards EPA generated actions to actors within the monitored system. The effectiveness of these actions is measured by the Output Matchmaker. For example, if a false positive is detected, then the Sensor Management can take countermeasures by a reconfiguration of the corresponding EPA; if a static program analysis implies that a sensor is no longer useful, then it is possible to remove it from the system by means of the Sensor Management.

4) Sensor Management: The Sensor Management controls the (de-)activation and placement of sensors in a monitored system. It is able to (de-)activate sensors on-demand and during runtime, as well as it is able to scale the degree of data-granularity sent by a sensor. For example, the sampling rate of a sensor can be adapted to the needs of its corresponding EPA. Another advantage of this dynamic management system is that the number of events transmitted can be adjusted to an optimal level with respect to the system resources available to the ACCEPT-VM.

C. Performance Considerations

Due to the enormous number of sensors, a number of performance considerations have to be taken into account. Regarding the CEP engine, it is possible to optimize an EPA in order to perform multiple computations only once. While this problem is solved for simple queries, it still remains unsolved for complex ones. Hence, new methods for pattern matching must be developed. Furthermore, EPAs can be distributed to multiple, distributed and dedicated computing resources. If every EPA runs as a separate thread, it is possible to leverage the advantages of recent multi-core architecture to achieve a massive speedup (intra-EPA parallelism). It is also possible to execute the ACCEPT-VM on multiple, dedicated physical machines or an a whole compute cluster, respectively. It is likely that the system will generate huge amounts of data during runtime. Thus, the need for efficient ways to transfer events arises, both within a single virtualized node and between physical nodes in a virtualized cluster. The first problem can be solved by a lightweight secure communication channel based on para-virtualization. However, for inter-node communication, an approach is needed that reduces the amount of data, which could be achieved by technologies such as difference transmission.

 ISBN: 978-1-61208-181-6

GPGPU Aided CEP Engine: Due to the huge amount of data and probably very complex requests to detect sophisticated anomalies, it is necessary to take further technologies into account. A possible solution is the use of General Purpose Graphical Processing Units (GPGPU) with OpenCL or CUDA. Due to the architecture of this hardware, a high degree of parallelism can be achieved. But instead of a general use, only special kind of requests should be executed with this SIMD architecture due to hardware restrictions. It is imaginable that the CEP engines uses the GPGPU as a co-processor. Especially highly computation intensive tasks such as pattern matching can be outsourced with a great benefit, as indicated by the PFAC [9] library for exact string matching performed on GPUs.

If compression algorithms are required to dump the data to disk, a further application of the GPU might be possible. Due to the high speed stream processing capabilities of a GPU, it can be used to compress the event streams during runtime without generating additional load on the CPUs.

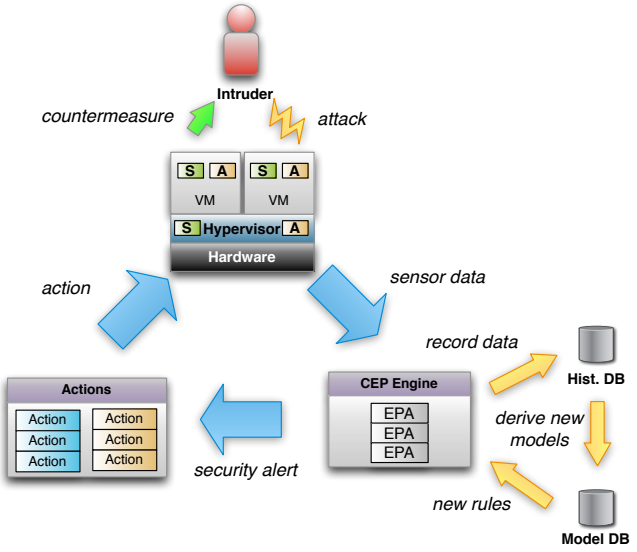

Fig. 3. ACCEPT Lifecycle

IV. Examples

The general lifecycle of the ACCEPT system is depicted in Figure 3. The ”blue” loop with the larger arrows represents the sense-detect-react-cycle and the ”yellow” loop with the smaller arrows shows the automatic rule generation process. To illustrate this new approach, two examples are presented in this section. First, a simple anomaly in double-entry accounting of the hypervisor and operating system layer port list is associated with a network based backdoor (see Subsection IV-A). Second, a more complex scenario shows an anomaly in the correlation of sensor data on the application container and operation system level (see Subsection IV-B), which is associated with a common attack scenario: An SQL injection attack [14].

A. TCP Backdoor

In this scenario, an attacker has successfully installed a backdoor in a monitored virtual machine. He/she hides his/her presence through a rootkit, a modification of the operating system and its userland interfaces. Even though the backdoor is listening on an arbitrary TCP port, the process belonging to it and the listening socket are not listed by the operating system userland tools.

Sensors: The scenario including the sensors and corresponding actions is shown in Figure 4. To detect the backdoor, at least two different sensors are involved: One sensor is running within the virtual machine and utilizes standard tools such as *netstat* to check for any listening sockets. Since the backdoor is well hidden, this sensor will not report the security breach.

The other sensor is inspecting the network state of the virtual machine from the hypervisor level. Since this sensor is running outside of the guest operating system, it is not affected by the backdoor’s hiding features. On this level, an event is generated for the detection of a newly opened port in the virtual machine.

Analysis: With the help of the Model Database and the Historical Database, queries can be generated to recognize normal or regular behavior. Therefore, an alarm should be triggered when a new open port is detected. Furthermore, by comparing both listening socket sensors, inside and outside of the virtual machine, it can be concluded that this really is a security related anomaly. A regular service installed in the virtual machine should not be hidden within the system. The conflicting sensors information is a clear sign of an attack.

Action: As a result of this attack, actions should be taken to eliminate the threat as much as possible. One such action could be to block all communication from and to the backdoor’s port on the hypervisor level. This prevents the attacker of extracting information or further using the infected machine. Another step that should be taken is to isolate and possibly terminate the processes involved in the infection. For forensics purposes, taking a snapshot of the virtual machine and generating a dump is another possibility.

B. SQL Injection

Another common attack scenario is the one of a web application vulnerable to SQL injection. Input not properly sanitized might lead to arbitrary queries executed on a backend SQL server. Even though SQL vulnerabilities could easily be fixed, they are still rated as the number one vulnerability in the Open Web Application Security Project Top 10 risk list [12].

Sensors: At least one sensor is needed to intercept all SQL statements sent from the web application to the corresponding SQL server. This sensor might be a JDBC proxy server or directly located in MySQL, for example. Furthermore, a sensor inspecting all running web applications and marking potentially harmful SQL statements. Furthermore, a sensor is needed to protocol which client request caused which SQL queries.

Analysis: All queries coming from potentially harmful marked statements can then be further analyzed. Signs of potential SQL injection attempts can be identified by various syntactic specialties such as extensive use of wildcard *SELECT*s or trimmed of code through comments. In conjunction

 ISBN: 978-1-61208-181-6

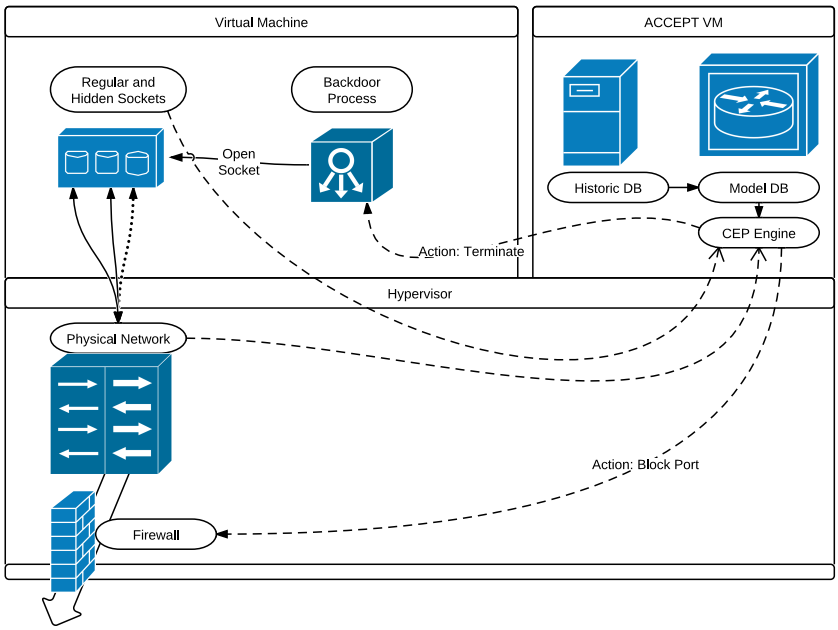

Fig. 4. Example: TCP backdoor detection.

with the information of the source of this request, an alarm can be raised. One by-product of the passive analysis is that potentially harmful code can be found before it is exploited and fixed in advance.

Action: The obvious action to take is blocking the offending IP address through a firewall. Through the use of the code inspection sensor, the faulty code sections might also be directly visible and automatic solutions to fix the problem can be used.

V. Conclusion

In this paper, a multi-level approach incorporating several state-of-the-art techniques such as virtualization, virtual machine introspection and complex event processing for detecting, analyzing and handling security anomalies has been presented. The proposed approach tries to keep management and maintenance within the virtual machines to a bare minimum by emphasizing the use of sensors on the hypervisor level. Furthermore, it does not only detect security anomalies, but is also able to react accordingly and defend or secure the system automatically. The flexible nature of the framework and the CEP backend make it especially easy to add new sensors to increase security and react to new threats or adapt to new technologies/devices. Using EPAs allows robust monitoring on all layers. With the Historical Database and techniques from machine learning, new EPAs can be automatically generated. While being able to detect not only new anomalies, the system can also verify false positives through correlation of different sensor layers. The components of the proposed framework form a reliable and adaptable security monitoring solution.

The approach is currently under development, and no significant implementation work has been started yet. Future work is devoted to detail the design of the components, implement the system and perform adequate experimental evaluations.

References

[1] AIDE Project. Advanced Intrusion Detection Environment (AIDE). http://aide.sourceforge.net/, 2011. retrieved: November, 2011.

[2] M. Bauer. Paranoid Penguin: An Introduction to Novell AppArmor. *Linux Journal*, 2006:13, August 2006.

[3] G. Cugola. Low Latency Complex Event Processing on Parallel Hardware. Technical report, Politecnico di Milano, 2011.

[4] G. Cugola and A. Margara. TESLA: A Formally Defined Event Specification Language. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, DEBS '10, pages 50–61, New York, NY, USA, 2010. ACM.

[5] M. Ficco. Achieving Security by Intrusion-Tolerance Based on Event Correlation. *Network Protocols and Algorithms*, 2(3):70–84, 2010.

[6] T. Garfinkel and M. Rosenblum. A Virtual Machine Introspection Based Architecture for Intrusion Detection. *Proceedings of the 2003 Network and Distributed System Security Symposium*, pages 191—206, Jan 2003.

[7] D. Gorton. Extending Intrusion Detection with Alert Correlation and Intrusion Tolerance. *Licentiate Thesis, Chalmers University of Technology*, 2003.

[8] National Security Agency. Security-Enhanced Linux. http://www.nsa.gov/research/selinux/, 2009. retrieved: November, 2011.

[9] PFAC open library. PFAC Open Library for Exact String Matching Performed on NVIDIA GPUs. http://code.google.com/p/pfac/, 2011. retrieved: November, 2011.

[10] Software AG. webMethods Business Events. www.softwareag.com/corporate/products/wm/events/capabilities/default.asp, 2011. retrieved: November, 2011.

[11] P. H. S. Teixeira, R. G. Clemente, R. A. Kaiser, and D. A. Vieira Jr. HOLMES: An Event-driven Solution to Monitor Data Centers through Continuous Queries and Machine Learning. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, pages 216–221. ACM, 2010.

[12] The Open Web Application Security Project. OWASP Top 10 Risks. https://www.owasp.org/index.php/Top_10_2010-A1, 2010. retrieved: November, 2011.

[13] Tripwire Inc. Tripwire. http://www.tripwire.com/, 2011. retrieved: November, 2011.

[14] W.G. Halfond J. and Viegas and A. Orso. A Classication of SQL-Injection Attacks and Countermeasures. In *In Proceedings of the International Symposium on Secure Software Engineering*, 2006.