RoViM: Rotating Virtual Machines for Security and Fault-Tolerance

Dorottya Papp*†, Zhendong Ma†, Levente Buttyan*

*CrySyS Lab
Budapest University of Technology and Economics, Hungary
{dpapp, buttyan}@crysys.hu

†Digital Safety & Security Department
AIT Austrian Institute of Technology, Austria
zhendong.ma@ait.ac.at

***Abstract*—Nowadays, the field of embedded system experiences a number of changes. On one hand, recent cyber attacks against safety-critical systems demonstrate that malware can force safety-critical systems to endanger human lives and harm the environment. Therefore, a new requirement of security have arisen for safety-critical and embedded systems. However, security should be designed hand in hand with safety to resolve conflicts between the two fields. On the other hand, the emerging trend of virtualization has significant impact on the embedded market. The isolation and protection mechanisms of virtualization contributes to both safety and security via redundancy and the prevention of one virtual machine affecting another.**

In this paper we present RoViM, a system of rotating virtual machines providing *proactive security* for embedded devices. RoViM uses multiple virtual machines in the system which increases redundancy as a safety measure. Our design satisfies reachability, liveness and safety requirements and we present a proof-of-concept implementation with use case of an Internet Protocol Security (IPsec) gateway. We evaluate our design with formal verification and show that rotating virtual machines cause no significant change in the performance of the IPsec gateway.

***Keywords*—embedded systems, proactive security, virtual machines, self-cleansing intrusion tolerance (SCIT)**

I. Introduction

Embedded systems are special-purpose computer systems which are dedicated to a single function and are tightly constrained with respect to cost, power, size and storage. Embedded systems react to changes in the environment. In some appliances, they must compute results in real-time. They are present in many field of our daily life, from electronic stethoscopes to automotive applications and railway. They are also the main driving force behind the concept of the Internet of Things, where the majority of the connected devices will be embedded computers instead of traditional PCs [1].

Safety-critical systems are special kinds of embedded systems whose malfunction may endanger human life or the environment. Therefore, their design needs special emphasis on safety among the traditional design requirements (reliability, availability, etc.). However, recent cyber attacks such as Stuxnet [2], and the threats discussed in [3] demonstrate that certain pieces of malware can prevent safety-critical systems from conforming with traditional requirements. Thus, a new requirement have risen: security. However, in many cases, existing security mechanisms affect traditional requirements negatively. As a result, safety and security should be designed hand in hand but the methodology required is still an area of active research. Existing results include integrating security analysis to existing tools [4] and modeling techniques [5], [6].

Not only the requirements, but also the technology of the embedded field is changing too. The emerging trend of virtualization in particular has significant impact on the embedded market [7], [8]. For example, the ability to run multiple virtual machines on the same physical board enables certified legacy applications to be run on modern hardware while emulating the outdated hardware the application was written for. Virtualization also contributes to fault-tolerance and reliability with its isolation and protection mechanisms preventing a fault in one virtual machine to affect another. What is more, the same isolation and protection mechanisms are also security measures. Can virtualization become a basis on top of which safety and security are designed together?

In this paper we aim to find the answer by designing a system of rotating virtual machines, RoViM, that provides proactive security for embedded devices. RoViM uses multiple virtual machines in the system which increases redundancy as a safety measure. Hence, our design satisfies security, as well as reachability, liveness and safety requirements. In addition, we present a proof-of-concept implementation where we realize an Internet Protocol Security (IPsec) [9] gateway using our RoViM approach. Our design is evaluated by formal verification and the test results of our proof-of-concept implementation shows no significant change to the use-experience.

II. Related Work

Our design of RoViM was inspired by the concept proposed in [10] of Self-Cleansing Intrusion Tolerance (SCIT). Instead of using reactive approaches to security, SCIT is a proactive risk management approach that uses virtual machines. A device implementing SCIT enjoys deletion of malware in every minute, restoration to a pristine state, recovery from software deletion attacks and cooperation with reactive approaches to security. SCIT has been implemented in prototypes for several use-cases in traditional IT environment such as Single Sign On (SSO) [11] and Service-Oriented Architecture (SOA) [12].

The SCIT Architecture consists of three core components. Firstly, a virtualization platform is needed for the virtual machines but the architecture remains independent of the chosen platform. Secondly, the SCIT controller is tasked with controlling the rotation of the virtual machines. It is installed on a secure machine within the internal network and acts as a central component. Thirdly, a persistent short term memory is required for processing data.

To our knowledge, the design principles of SCIT have never been studied in the context of safety-critical systems. The domain poses interesting challenges for the original concept because devices run smaller and more limited applications compared to standard PCs and servers. Plus, many devices are not located inside a protected internal network but are deployed out in the field. This renders the usage of a central component inefficient in this context. In this paper, we replace the central controller with a distributed solution. What is more, handling persistent data in SCIT is a challenge. The difficulty in the original concept arises from the practice of destroying the virtual machine exposed to the network and replacing it with a new one. This process destroys the temporary memory resulting in the loss of persistent data. The authors overcame this problem by using a Network Attached Memory which acts as a shared memory between virtual machines. However, a shared memory can be used as a stepping stone for the attacker from one virtual machine to another. In this paper, we present another solution to data propagation between virtual machines that enables the close monitoring of persistent data and can be used to efficiently detect possible compromises.

III. High-level Overview of The System

Our designed system, RoViM, follows the principles of [10] and provides proactive security for embedded devices. However, the design of RoViM takes into account not only security, but the potentially safety-critical nature of the embedded device as well. The system consists of multiple virtual machines, each of which is capable of performing the same task as the embedded device. The usage of multiple virtual machines provides redundancy and thus contributes to the overall safety of the embedded device.

Before the high-level overview of the system and our assumptions can be discussed, some definitions must be made. The *active* virtual machine is the virtual machine that performs the task of the embedded device and is connected to and communicating with the outside world. *Standby* virtual machine(s) provide redundancy and are on cold standby, waiting to take the place of the active virtual machine. The standby virtual machine that will become the active virtual machine in the rotation is called the *next active* virtual machine. The *cleansing* virtual machine previously acted as the active virtual machine and is being restored to its compromise-free state. In practice, the compromise-free state can be a snapshot taken before the deployment of the embedded device. Rotations between virtual machines happen periodically.

Our assumptions are as follows. We assume that the virtual machines communicate via internal communication channels (e.g., virtual LANs), non visible outside of the embedded device. We anticipate communication failures between virtual machines but require that the communication channel notifies the system about such a failure. In our threat model, the attacker can interact with the system and compromise the active virtual machine just like he could compromise the embedded device. However, we expect rotations to happen frequently enough so that the attacker is unable to compromise standby and cleansing virtual machines via the internal communication channel used by the virtual machines.

The rotation of virtual machines should be as transparent as possible to outside entities and services. However, devices on the network must know or at least must be notified about the changes in the address of the active virtual machine. Otherwise, packets on the network would not be received by the next active virtual machine. Therefore, nodes on the network to accept updates to the network address of the embedded device. For example, in case of Address Resolution Protocol (ARP) on the data link layer, all devices in the local network must process unsolicited ARP replies - which acts as the notification of a rotation - and update their ARP caches.

Shown in Figure 1, one rotation is depicted with the high level interaction needed for a single rotation to complete. The unnumbered arrow between the active virtual machine and the outside world highlights that the communication between the active virtual machine and the outside world is not disrupted by the rotations.

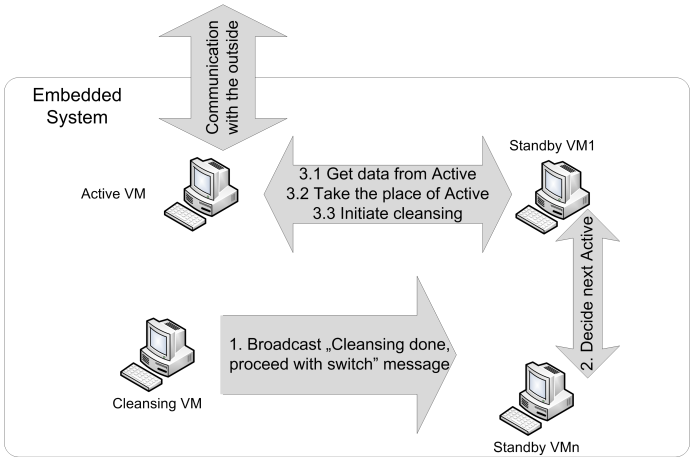

Fig. 1. High-level Overview of the Design

In the second phase, a standby virtual machine becomes the next active virtual machine. Depending on the number of standby virtual machines used, two cases are possible for a standby virtual machine to become the next active virtual machine. If there is one standby virtual machine (apart from the previously restored virtual machine), that virtual machine will automatically be the next active virtual machine. If there are multiple standby virtual machines, they must agree on the standby virtual machine that should become the next active virtual machine. This problem translates to the well-known leader election problem which has been discussed in literature several times [13].

To become the active virtual machine, an interaction of three subphases is needed between the active and the next active virtual machines. A high-level description of the three subphases is presented here and more details are discussed in Section IV.

1) The next active virtual machine must acquire all required data to perform the task of the embedded device correctly. As the active virtual machine is connected to the outside and may be compromised, the data on it may become corrupted and malware may be installed. Our designed system can be extended to ensure that no malicious content is propagated to other virtual machines via validation of the application data. Also, while the data from the active virtual machine is being transmitted, the active virtual machine must make no changes to the application data. Otherwise, the application running on the next active virtual machine and the entities in the outside world become out of sync. In a sense, time must freeze for the application but this may be against the safety mechanisms implemented by the application. Therefore, the implementation of the rotation must specify a time limit during which the next active virtual machine can take the place of the active virtual machine. If the next active virtual machine does not succeed within the time limit, the rotation should be aborted
2) The next active virtual machine must notify all nodes on the local network to route packets currently destined to the active virtual machine to the next active virtual machine instead
3) The next active virtual must initiate the restoration of the active virtual machine into a compromise-free state. The active virtual machine is connected to the outside world and may be compromised. We can assume that it is not in the interest of the attacker to restore compromised virtual machines to their compromise-free state. Therefore, the procedure of cleansing must be forced by the next active virtual machine. One such cleansing procedure can be a reverting to a snapshot taken before the deployment of the embedded device

A. Applications and RoViM

To complete the three-subphase protocol of taking the place of the active virtual machine, the application is required to provide all data necessary for its correct functioning in a form that can be transmitted to other virtual machines. It is also required to be able to restore that data when it is provided. As a result, existing applications require some kind of adaptation or extension to work in this paradigm.

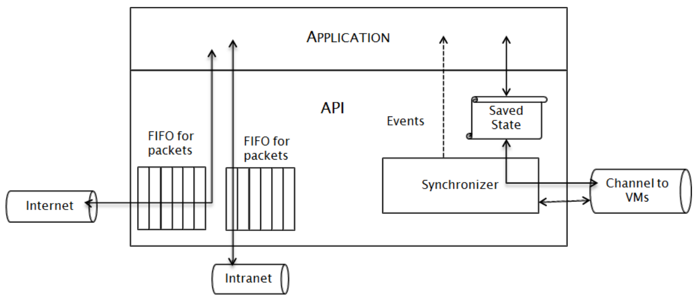

Fig. 2. Interaction of Existing Application and Rotation API

We propose making applications aware of the rotation. While new application can be developed with the rotation in mind, existing applications can also be tailored to the rotating environment with little effort from developers. As shown in Figure 2, the rotation is implemented by a software layer we refer to as the API. The API should control all resources the application uses to be able to freeze time for the application. In Figure 2, FIFO buffers are shown for all network interfaces the application communicates through. Therefore, the reception of packets can be delayed. The API communicates with the application via events to signal different phases of the rotation. When the data used by the application is needed to be copied to the next active virtual machine, the application should provide the data in a serialized form, for example, a file. The data may consists of variables, configuration files, etc. The serialized form is transmitted to the next active virtual machine where the application reads the serialized form and restores the contents. The serialized form of the data used by the application can also be subjected to validation and be used to efficiently detect a compromise. This approach is advantageous from the security point of view as the API does not interact with the possibly compromised memory of the application.

IV. Taking the Place of the Active Virtual Machine

As mentioned before, the active and the next active virtual machines must complete a three-subphase interaction before the cleansing procedure. The interaction can be achieved by the following protocol of three subphases.

1) Subphase 1 - Transferring the Application State: During Subphase 1, the next active virtual machine receives the data of the application running on the active virtual machine. The Subphase starts with a trigger message from the next active virtual machine and acts as a request for the application data in serialized form. The trigger causes the active virtual machine to transmit the requested data to the next active virtual machine. Note, that the transmission relies on no specific protocol, the details of transmitting the serialized form is left to the implementation. After receiving the trigger message, the active virtual machine must not process incoming packets. A processed packet at this time could change the application data, rendering the transmitted serialized form out-of-date. Instead, to avoid packet loss, packets are put on hold in buffers until one of the two virtual machines is ready to process them.

As we anticipate communication failures, virtual machines set a timeout during which messages must arrive. If a message is not received within the time limit, it is considered lost. After suffering specific amount of lost messages at either virtual machine, that virtual machine assumes the channel to be broken and aborts the protocol without further notice. The time limit and the maximum number of lost messages should be configured with respect to the safety requirements of the embedded device.

2) Subphase 2 - Configuration of Network Interfaces: Subphase 2 is attempted only if the application running on the next active virtual machine has the data for the application. After all, the next active virtual machine has no means of performing the task of the embedded device if it is unable to process packets because of the absence of the application data. During Subphase 2, the active and next active virtual machine configure their network interfaces and notify the networks about the change in the address of the embedded device.

At the beginning of Subphase 2, the next active virtual machine tries to bring its interfaces up through which the application expects packets. If the process is successful, the virtual machines continues the protocol; if unsuccessful, the parties need to abort the protocol. In either case, the active virtual machine is notified about the outcome. In case of success, the next active virtual machine notifies the network about the change in the address of the embedded device and instructs the active virtual machine to bring its interfaces down.

Virtual machines might experience communication failures during Subphase 2 as well. To recover from losing the message containing the outcome of bringing up the interface of the next active virtual machine, the active virtual machine sets a time limit. When the time limit is exceeded, the active virtual machine must poll the next active virtual machine for the outcome. Why not abort the protocol? Let us assume for a moment that after a specified amount of polling for status, the active virtual machine deems the communication channel broken and aborts the protocol. At this point both virtual machines are capable of processing incoming packets: both have the data for the application, the correct networking configuration and are accepting packets from the outside. Now, we have two virtual machines as the active virtual machine. Depending on the timing of their packets sent, nodes in the network might repeatedly update the address of the embedded device and transmit packets to one of the virtual machines. However, the embedded device and the outside world would lose synchronization as the virtual machines would update their data based on different packet flows.

3) Subphase 3 - Optional Buffering: During Subphase 3, packets buffered during the interaction are relocated to the virtual machine capable of processing them. If Subphases 1 and 2 finished successfully, buffered packets are transmitted to the next active virtual machine and processed there. Before relocating the buffered packets, the next active virtual machine requests information about the size of the buffer. Depending on the received size, it decides whether or not the time needed for processing the packets is within the safety requirements. If the protocol was aborted, the buffered packets are processed by the active virtual machine. Buffered packets suffer latency which depends on the size of the application data, the number of packets arriving to the active virtual machine during Subphases 1 and 2 and the network throughput between virtual machines. Depending on the safety requirements, the introduced latency may or may not be acceptable. In some cases, where packet loss is acceptable (e.g. communication using UDP) the buffering of packets should be disabled and packets arriving during Subphases 1 and 2 should be dropped by the active virtual machine.

Even though Subphase 3 is optional, chance of recovery from possible communication failures is added to the protocol. When the active virtual machine receives the command to bring its interfaces down and Subphase 3 is enabled, it sets a timeout during which the request for sending information about the buffered packets must arrive. If the request does not arrive, the active virtual machine forcibly sends the information to the next active virtual machine and waits for the decision. If the decision does not arrive in time, it is treated as a refusal. At the next active virtual machine, after the request for information is sent, a timeout is set during which the requested information has to arrive. If it does not arrive, the next active virtual machine can retry and then ultimately abandon Subphase 3 deeming minimizing the latency introduced more important than avoiding packet loss.

A. Formal Verification

As Subphase 3 is optional, Subphases 1 and 2 of our protocol discussed in Section IV was subjected to formal verification. Space limitations does not allow us to present the formal modal in more details. The interested reader is referred to [14]. The following requirements are demanded of Subphases 1 and 2:

- Reachability properties: Successfully reaching the end of Subphase 2 and aborting either Subphase are both possible
- Safety property: No deadlock occurs during Subphases
- Liveness property: Eventually one of the following scenarios happen: either Subphases 1 and 2 are both completed successfully or the protocol is aborted

We verified these properties with Uppaal, an integrated tool for modeling, verifying and validated real-time systems [15]. Uppaal evaluates the state-space of the modeled system. The state-space can be represented by a graph in which every node contains a possible set of states of the system and directed edges are possible changes is the state of the system. Queries to the model-checker are expressed using a simplified version of Timed Computation Tree Logic. The formal verification was not aimed at finding security issues (those are discussed in Section VII), but to check the correctness of the protocol with respect to safety requirements.

V. Proof-of-Concept Implementation

The environment of our proof-of-concept implementation is shown in Figure 3. We implemented an IPsec gateway consisting of four virtual machines (IPsecGatewayServer). The IPsec gateway is an end-point of an IPsec tunnel establishing secure communication between two end-points, the Client and the Server. The Internet itself is modeled as network between two routers.

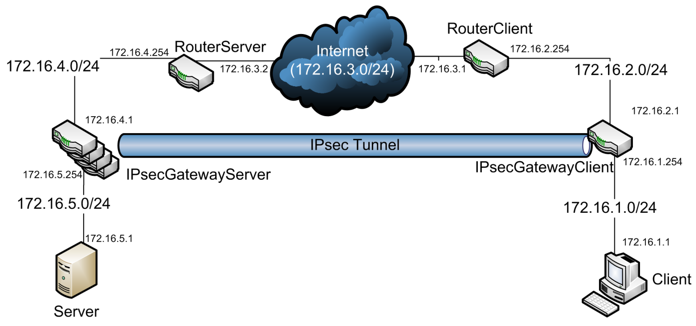

Fig. 3. Use Case of IPsec Tunnel

In our proof-of-concept implementation, all virtual machines share the same IP address to the outside world which makes rotations transparent in Layer 3 and above. However, separate addresses are used in Layer 2 to differentiate between the virtual machines. As a result, the destination address of the frame decides which virtual machine receives the frame. It must be mentioned that this solution works only if the Layer 2 address related to an IP address can be forcibly updated at nodes on the local networks. As ARP is used in the data link layer, notifications about the rotation are unsolicited ARP replies sent by the gateway.

The rotating virtual machines use two additional networks for internal communication as shown in Figure 4. The Leader Election network is used by the standby virtual machines to decide the next active virtual machine. The interaction needed for the active and next active virtual machines to complete the rotation happens in the State Exchange network. The separation of internal networks ensures that standby virtual machines are separated from the exposed active virtual machine. Virtual machines keep their unnecessary network interfaces down.

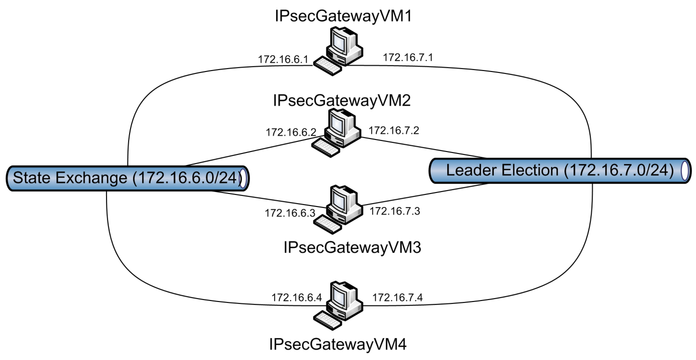

Fig. 4. Internal Networks for Rotation

Our proof-of-concept implementation uses VMware ESXi [16] as the virtualization platform. The virtual machines all run the same operating system, Ubuntu 14.04 Server LTS. Additional packages are installed for the implementation: openssh-server for transfer of application state, ulogd2-pcap for implementing the FIFO buffers, arping for sending unsolicited ARP replies and python-pip as the implementation was written in Python.

VI. Performance Evaluation of the Implementation

The proof-of-concept implementation aimed at uncovering changes in the user-experience and to measure the overall latency introduced by the rotation. A 500 Mb file was downloaded from the Server to the Client using wget. We compared the performance of the proof-of-concept implementation to the performance of the environment without rotation. During the download, one rotation was triggered manually. Table I shows the results generated by Wireshark using the captured packets at the Server.

TABLE I
Continuous Packet Flow with and without Rotation

	Without Rotation	With Rotation
Transmission time	23.680 s	23.684 s
Duplicate IP address configured	0	3
Retransmissions	55	315

The TCP connection for the file transfer did not break while the rotation was in effect. As expected, the rotation introduced latency to the transmission. However, the transmission time increased by only 4 ms, which did not have a significant influence on the user-experience.

Wireshark gave the warning of Duplicate IP address configured, realizing that while the IP address of the gateway did not change, the MAC address did. This warning was present three times in the packet flow, twice as a result of the interface management of Ubuntu and once when the unsolicited ARP reply was sent.

On the other hand, the rotation had a negative effect on the retransmission timeout and introduced a significant increase in the number of retransmissions. The algorithms used to calculate the retransmission timeout [17] adjust the retransmission timeout to the capabilities of the connection link: connections with higher throughput have lower retransmission timeouts. In our case, up until Subphase 1, the network throughput in the test environment was very high, because there was no other source of traffic and the test environment was also virtual. Then, Subphase 1 started and all incoming packets were buffered at the active virtual machine. This resulted in artificial latency introduced to the segments and acknowledgement for them were delayed. The artificial latency was high enough for the Server to assume communication failure and it retransmitted the segments affected by the latency. Therefore, we can conclude that Subphase 3 is unnecessary for TCP connections because the recovery mechanisms of TCP make up for the lost segments.

VII. Discussion on Security and Future Work

As mentioned in Section III, the threat model used during the design of the protocol assumes that the attacker does not have enough time between rotations to use the internal communication channel between the virtual machines. Nevertheless, if the next active virtual machine cannot succeed in taking over the place of the active virtual machine repeatedly, the attacker might gain the time needed to compromise standby virtual machines as well. A few implications of such a scenario and possible countermeasures are discussed here as future work.

Even though the next active virtual machine is honest and brings down its interfaces to the internal network for standby virtual machines, it may become compromised as the active virtual machine. The attacker could bring this interface up and interact with the standby virtual machines. For example, the attacker could forge messages and prevent standby virtual machines from winning the leader election. As a future work, we propose that instead of bringing down interfaces, virtual machines becoming the active virtual machine should be reconfigured without access to the network in which the leader election takes place.

Another way for the attacker to compromise the next active virtual machine is through the application data the next active virtual machine requests during Subphase 1. If the attacker provides malicious content instead of the required data for the application, the attacker might exploit a vulnerability in the application and compromise the next active virtual machine. If the attacker sends bogus data or does not transmit the data at all, the application is cut from the data needed to provide seamless execution from the outside world's point of view. While malicious content or bogus data can be detected by extensive input validation, the denial of service situation arising from the missing data is not easily handled.

The fault-tolerance of the designed system could be improved by using fault detection. In the current design, a fault in the active virtual machine can render the system unable to function as the application state is lost with the active virtual machine. What is more, the fault may not even be detected if it occurs before the start of Subphase 1 as there is no interaction between the standby and the active virtual machines before that point in time. By adding fault detection, the standby virtual machines could monitor the performance of the active virtual machine and determine when it experiences faults. The fault detection could also be used to save the current state of the application if no fault is detected. On the other hand, the activity involved would add to the attack surface of the standby virtual machines, potentially allowing the attacker to compromise the standby virtual machines.

At the time of writing, the proof-of-concept implementation runs in a PC environment. As the next step, the code will be ported to an embedded Linux operating system running on a multi-core architecture. The use-case will demonstrate an open deterministic network with mixed-criticality.

Acknowledgment

This work is partially supported by the EU ARTEMIS project EMC^2 (contract no. 621429) and Austrian Research Promotion Agency (FFG).

References

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[2] N. Falliere, L. O. Murchu, and E. Chien, "W32. stuxnet dossier," *White paper, Symantec Corp., Security Response*, vol. 5, 2011.

[3] D. Papp, Z. Ma, and L. Buttyan, "Embedded systems security: Threats, vulnerabilities, and attack taxonomy," in *Privacy, Security and Trust (PST), 2015 13th Annual Conference on*, July 2015, pp. 145–152.

[4] M. Eby, J. Werner, G. Karsai, and A. Ledeczi, "Integrating security modeling into embedded system design," in *Engineering of Computer-Based Systems, 2007. ECBS '07. 14th Annual IEEE International Conference and Workshops on the*, March 2007, pp. 221–228.

[5] S. Zafar and R. Dromey, "Integrating safety and security requirements into design of an embedded system," in *Software Engineering Conference, 2005. APSEC '05. 12th Asia-Pacific*, Dec 2005, pp. 8 pp.–.

[6] C. Schmittner, Z. Ma, E. Schoitsch, and T. Gruber, "A case study of fmvea and chassis as safety and security co-analysis method for automotive cyber-physical systems," in *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*, ser. CPSS '15. New York, NY, USA: ACM, 2015, pp. 69–80. [Online]. Available: http://doi.acm.org/10.1145/2732198.2732204

[7] W. River, "Applying multi-core and virtualization to industrial and safety-related applications," http://leadwise.mediadroit.com/files/8535WP_Multicore_for_Industrial_and_Safety_Feb2009.pdf, February 2009.

[8] C. Main, "Virtualization on multicore for industrial real-time operating systems [from mind to market]," *Industrial Electronics Magazine, IEEE*, vol. 4, no. 3, pp. 4–6, September 2010.

[9] S. Kent and K. Seo, "Security architecture for the internet protocol," 2005.

[10] A. Bangalore and A. Sood, "Securing web servers using self cleansing intrusion tolerance (scit)," in *Dependability, 2009. DEPEND '09. Second International Conference on*, June 2009, pp. 60–65.

[11] F. Huang, C.-x. Wang, and J. Long, "Design and implementation of single sign on system with cluster cas for public service platform of science and technology evaluation," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*. IEEE, 2011, pp. 732–737.

[12] Q. L. Nguyen and A. Sood, "Improving resilience of soa services along space-time dimensions," in *Dependable Systems and Networks Workshops (DSN-W), 2012 IEEE/IFIP 42nd International Conference on*. IEEE, 2012, pp. 1–6.

[13] N. A. Lynch, *Distributed Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996.

[14] D. Papp, "Security of Safety-Critical Multicore Systems," Master's thesis, Budapest University of Technology and Economics, 2015, online: http://www.crysys.hu/~dpapp/publications/dpapp_master_thesis.pdf.

[15] G. Behrmann, A. David, and K. G. Larsen, "A tutorial on uppaal," in *Formal methods for the design of real-time systems*. Springer, 2004, pp. 200–236.

[16] VMware, "Vmware esx and vmware esxi," https://www.vmware.com/files/pdf/VMware-ESX-and-VMware-ESXi-DS-EN.pdf, 2009.

[17] R. Braden, "Requirements for internet hosts – communication layers," https://tools.ietf.org/html/rfc1122, October 1989.