Creating Modern Blue Pills and Red Pills

Asaf Algawi[1], Michael Kiperberg[4], Roee Leon[1], Amit Resh[3] and Nezer Zaidenberg[2]
[1]University of Jyväskylä, Finland
[2]College of Management Academic Studies, Rishon LeZion, Israel
[3]Shenkar College, Ramat Gan, Israel
[4]Holon Institute of Technology, Holon, Israel
asaf@trulyprotect.com
michael@trulyprotect.com
roee@trulyprotect.com
amit@trulyprotect.com
nezer@trulyprotect.com

Abstract: The blue pill is a malicious stealthy hypervisor-based rootkit. The red pill is a software package that is designed to detect such blue pills. Since the blue pill was originally proposed there has been an ongoing arms race between developers that try to develop stealthy hypervisors and developers that try to detect such stealthy hypervisors. Furthermore, hardware advances have made several stealth attempts impossible while other advances enable even more stealthy operation. In this paper we describe the current status of detecting stealth hypervisors and methods to counter them.

Keywords: virtualization, forensics, information security

1. Introduction

The blue pill was introduced by Johanna Rutkowska in Blackhat 2006 (Rutkowska 2006b). There were others who used hypervisor for security it was with the introduction of the concept of the blue pill and red pill that the virtualization concept became so closely related to cyber security.

The blue pill is a rootkit that takes control of a victim host computer. Unlike other rootkits the blue pill is actually a malicious hypervisor. The original blue pill starts after the OS has already booted and through a series of hardware instruction, the blue pill gains control of the victim host. In fact, after the blue pill is deployed it has gained higher privileges then the OS that started it. Of course, like all rootkits the blue pill must camouflage its existence, or it will be removed by the user. Using a hypervisor assists in camouflage because the blue pill can now start tasks outside the OS scope. In order to counter the blue pill the red pill was invented. The red pill is a hardware or software tool that is designed to detect such malicious blue pill rootkit.

The red pill is a special case of the related "trusted computing" and the attestation concept (Zaidenberg et al. 2015d), In Trusted computing attestation a remote 3rd party or even local software tries to ensure the integrity of the local machine in terms of software (mainly) and hardware (sometimes).

When hypervisors are concerned the attestation of lack of hypervisor or blue pill was first researched by Kennell et al. (Kennell et al 2003) in order to establish the "genuinity of the host" (i.e. ensure that the host is a physical machine running the correct software as opposed to an emulator or a virtual machine or a physical machine running non-genuine software) Kennell proposes running a series of tests that will pass only if the inspected system is genuine. The test will fail if a hypervisor is running due to side effects involved with the running of hypervisors, mainly more expensive memory traversal.

Today the original Kennell hypothesis are questionable due to the availability of specific hardware instructions and features designed to remove memory access side effects (Second Level Address Translation instructions such as EPT™ in Intel's case and RVI™ in AMD's case) However the core idea of using side effects is still in use in many modern red pills.

Since Rutkowska introduced the blue pills malware concept, multiple attempts to create red pills that detect such blue pills have also been proposed. However, more advanced blue pills have been designed to avoid detection. Which led to more advanced red pills and so on and so forth. So, the goals of the blue and red pills are conflicting and naturally technology advances in one front requires advancement by the other front in order

to keep up. This paper describes multiple modern methods in which one can construct a blue pill or red pill defeating most blue pills.

2. Modern hardware capabilities

Nowadays, modern CPUs by Intel, AMD and ARM feature hardware-assisted virtualization. Hardware-assisted virtualization provides new capabilities for implementing virtual machines and emulator software. Thus, hardware-assisted virtualization makes several "red pill" attempts futile, for example by eliminating several virtualization side effects(Zaidenberg et al 2015c).

However, hardware-assisted virtualization provides new forensics methods. Therefore, many new openings to create new red pills now exist and many side effects that were previously used are eliminated or are much subtler.

This paper describes the situation of red pills and blue pills primarily on Intel virtualizations architecture circa 2019 and the ninth generation of intel's core CPUs.

3. Background

Blue pill technology relies on hardware assisted virtualization ("hypervisor instructions") and hardware assisted virtualization technology. Recent advances in x86 hardware-assisted virtualization allow for better blue pills and red pills. These new instruction families are called VT-x, VT-d, EPT on the intel architecture. AMD-v, IOMMU and RVI are the corresponding names on AMD architectures. The new instructions enable the blue pill and red pill capabilities.

3.1 Hypervisors and thin hypervisors

A hypervisor is a computer software concept that is designed to run multiple operating systems on the same hardware.

As its name implies, a hypervisor has higher privileges (hyper= "above") than the operating system (i.e. The operating system = the supervisor).

The operating system supervises memory and hardware resources for the processes that runs on top of it. Likewise, the hypervisor supervises the hardware resources for each operating system that runs on top of it.

Hypervisors research started with Popek et al. (1974) who classify hypervisors into two main categories:

- 1. Type I hypervisors, or boot hypervisors, are hypervisors that the machine starts from on boot. The machine starts the guest operating system after booting the hypervisor. VMWare ESXi is an example of a modern Type I hypervisor.
- 2. Type II hypervisors or hosted hypervisors are hypervisors that start only after the operating system has started. A modern example for a Type II hypervisor is VMWare Desktop or Oracle Virtual Box hypervisors.

The original blue pill that was described by Rutkowsa was a Type II hypervisor, however type I "blue pill" hypervisors (boot kits) are also possible.

Regular hypervisors reside logically between the hardware and the supervisor (OS) layers. The hypervisors catch interrupts and deliver them to the correct operating system and control memory addresses. The hypervisor uses its own translation tables for deciding which operating system owns each memory address and which operating system should handle each hardware interrupt. This is analogues to the MMU in OS environment where each memory address is assigned to a different process.

However, there also exists a special case of hypervisors that do not attempt to run multiple operating systems. Instead, these hypervisors, called "thin hypervisors", supports running only one operating system on the target hardware. All interrupts and memory accesses are either blocked or transferred to the operating systems for handling. Indeed, Thin hypervisors act as a microkernel that provides certain services to the underlying operating system. The thin hypervisor includes very little memory management and relies on the single guest OS system for both memory management and interrupt handling.

3.2 Hypervisors, forensics and cyber security

Zaidenberg (Zaidenberg 2018) summarized hypervisor usage in cyber security. Hypervisors can be used to inspect the target system. Such forensics effort can be used to assist developers (Khen 2011), profile the code (Khen 2013) or directly to obtain the memory of the inspected system (Kiperberg et al 2019a) or detect malware (Zaidenberg et al 2015b). Malicious software may desire to inspect the hypervisor presence before deployment.

Furthermore, the hypervisor may provide security services to the guest system. Microsoft's Deviceguard, TrulyProtect hypervisor for protection against reverse engineering (Averbuch et al. 2013, Kiperberg et al 2019b) and Execution Whitelisting (Kiperberg et al. 2017) are examples of thin hypervisors. Other thin hypervisors monitor Video DRM (David et al 2014), provide forensics data or provide end point security (Leon et al 2019) Virtually all blue pills are thin hypervisors.

3.3 x86 virtualization

On intel's x86 CPU architecture, hardware-assisted virtualization is provided by unique instruction families. Intel architecture and AMD architecture each provide three such instructions families for handling hypervisors.

These instructions are further optimized with each new processor generations. Newer generations include new hardware assisted virtualization capabilities such as shadow VMCS. Furthermore, virtualization instructions take less CPU cycles to complete in newer CPU generations.

The x86 instruction families are presented in Table 1.

Table 1: x86 Virtualization Instructions

	Intel Architecture Name	**AMD Architecture Name**	**Usages**
Virtualization instructions	VT-x	AMD-v	This family is required for starting a hypervisor
SLAT (second-level address translation)	EPT (Extended page tables)	RVI (Rapid virtualization indexing)	Allows the HV to run Multiple MMUs for multiple guest operating system
IO MMU	VT-d	IOMMU	Allow the HV to assign IO memory to specific guest operating systems
VM data structure	VMCS	VMCB	Holds all the VM information (one per VM)

3.4 Rootkits and the blue pill

The recommended and common system administrator's cyber incident-response protocol once an attack is detected on any networked server, is to format and reinstall the effected server's operating system. If the operating system is reinstalled (and fully patched with security patches), then the hacker may find herself locked outside of the compromised system. It follows that the hackers have a desire to hide thier tracks. Thus, if the hack was not detected, the hacker can maintain a persistent access to the hacked servers. Therefore, hackers frequently install software packages is known as a "rootkit." A rootkit is a software package that allows the hacker access to the victim system resources. Furthermore, the rootkit hides itself and all the processes that the hacker runs on the infected system in an effort to mask its existence. Therefore, the rootkit has two goals. First the rootkit provides the hacker ease of access to victim computer resources. Second the rootkit provides measures to hide the hack and its own existence on the victim system.

There are many ways to build rootkits from hijacking system calls and library functions and installing *setuid* programs to replacing innocent-looking binaries.

The blue pill is a special type of rootkit. Unlike normal rootkits that modify the operating system in order to hide files and gain access to the system, the blue pill starts a hypervisor. Thus, the blue pill gains more permissions than the operating system. The blue pill can run processes in the hypervisor address space. Thus, the processes or their address spaces are not visible by the operating system.

3.5 Bootkits

One of the ways that the rootkit can operate is as a "bootkit". A bootkit is a special type of rootkit that boots (from the hard drive master boot record, UEFI, PXE, or other means) before the operating system starts. The bootkit runs its own software (i.e. the hypervisor in the blue pill case) before the OS starts and later boots the OS (by calling the OS boot). The bootkit may also patch the OS system calls in order to hide its processes and files.

3.6 The original blue pill and subverting attacks

Rutkowska (2006a) has introduced the original blue pill concept in 2006. The blue pill approach was an innovative rootkit approach that was relatively unresearched at the time. Since Rutkowska introduced the first blue pill, there have been several suggestions on how a blue pill can be detected (for example, the memory location of the IDT vector) However advances in x86 virtualization technology make such detection more difficult or even impossible. For example, it was initially proposed to check the address of the IDT in order to detect hypervisors. However, using EPT (new virtualization technology) it is completely possible to maintain addresses that appear to be genuine IDT address by the guest but point to different address.

3.7 Blue chicken

Rutkowska et al. (2007) introduced the blue chicken blue pill one year after the original blue pill. The blue chicken blue pill detects the red pill inspection attempts and unloads itself, and then reinstalls itself after the detection attempts are over. The blue chicken method is not recommended for two reasons**.**

- A hypervisor can be loaded after the blue pill is unloaded. Then, if the blue pill tries to reload itself, the new hypervisor can detect it and prevent the blue pill from loading. The new hypervisor can even act as a blue pill, allowing the original blue pill to load in order to perform forensics inspect its operational logic).
- Most red pills do not consume too much time and may be able to detect hypervisor present before the blue chicken manages to unload.

We summarize that the blue chicken method is not a threat today.

4. Local red pills

Local red pills are tests performed by the tested machine and contained within the tested machine. These tests cannot be considered reliable as computation is performed on an untrustworthy machine (the very machine that is being inspected). However, in many real-life scenarios there is no TPM chip available or similar third-party chip or server that one can use for attestation.

If no third party, trustworthy, root of trust is available, then one is left with running local code on an untrustworthy machine to provide attestation in a best effort attempt to detect blue pills on the inspected machine.

4.1 Paranoid fish and other modern red pills

Paranoid fish (Pafish) (Ortega 2016) is currently the *de facto* standard in "red pill for blue pill detection." software Pafish includes multiple tests capable of detecting most known blue pills when running under Linux or Windows. Many of these tests to detect a hypervisor assume the hypervisor is not really attempting to hide by looking up specific values in memory. (For example, VMWare routinely reports 440BX (1990's era hardware) on all machines. These chipsets are over 20 years old components that are virtually none-existent on real physical modern systems)

However, some of Pafish tests are specifically geared toward hypervisor that do try to hide. These methods relay on timing and side effects (Zaidenberg 2015c) of running hypervisors. Local timing methods are employed to try to flush out these blue pills. The local timing tests perform the following steps:

- Take local time (for example, RDTSC instruction).
- Execute an operation that must be intercepted by a hypervisor (for example, CPUID instruction).
- Take local time again to obtain elapsed time.

The underlying assumption is that step (2) takes significantly longer to execute when a hypervisor is active.

4.2 Paranoid fish tests

Paranoid Fish (Pafish) tests include the following:

- Specific and paravirtualization tests – These tests are designed to detect specific commercial hypervisors (that do not try to hide)
- Timing tests (Pafish includes two timing tests)
- *RDTSC, CPUID, RDTSC < 1000 cycles. This test runs in user space. It runs the CPUID instruction, which must be intercepted by a hypervisor. As a result of the required context switch, the time required for the three instructions acts as a tell-tale sign for hypervisor presence (the red pill).*
- *RDTSC, RDTSC < 750 cycles - This test involves calling two RDTSC instructions and measuring the response times. It is designed to ensure that the hypervisor is not intercepting the RDTSC instruction.*
- *In both tests, an irregular high result will be obtained due to an interrupt. To avoid false positives due to irregular time required for the context switch, Pafish performs an average over 10 runs.*
- Sandbox tests – These tests detect if the system is some sort of sandbox. The tests measure if there is no user interaction or mouse movement.

4.3 Defeating the paranoid fish timing tests

Most Pafish tests can be defeated easily using the following methods:

- Instruction replacement – The CPUID operating can be detected and replaced by the hypervisor. In that method, the first time that CPUID is called a context switch will occur and, indeed, take a long time; however, further instructions can be replaced with another instruction that generates a similar response (such as generating an interrupt and returning with iret).
- Tampering with RDTSC - The hypervisor is allowed to intercept calls to RDTSC and similar calls and provide its own responses. The hypervisor may run its own code and return values that are smaller than the values obtained from the CPU. These values can take into consideration time spent in the hypervisor. However, RDTSC time measurements are needed for the operating system to function properly. Tampering with RDTSC return values may cause system instability.
- Intel allows the time measurements (in CPU cycles) on the VM to be slower than on the actual machine; for example, every two CPU cycles on the physical PC can be considered as only one CPU cycle on the guest. This method can offset several of the red pill tests as operations no longer take “too much time.” However this test works with a constant multiplication factor. This is not the case in real life. Some CPU opcodes require context switches and are much slower when a hypervisor is present while other operations take exactly the same amount of time regardless of hypervisor presence. Thus, these tests can be offset by performing operations that do not take longer on a VM (such as ADDing, ORing, and NOPing). It is possible to create a sequence of very fast operations that do not run slower on a VM. If the VM is running some sort of clock adjustment were VM cycles are slower then real world cycles, then these operations may seem to take too little time on the VM. (creating yet another type of red pill)

4.4 Augementing the paranoid fish tests

The Pafish timing test runs in user mode and examines only a limited set of instructions (for example, only RDTSC and not RDTSCP). By augmenting Pafish timing tests to run in kernel mode and trying multiple instructions (RDTSC, RDTSCP etc.), the Pafish tests can become more potent and harder to avoid.

Other well-known tests not found in Pafish include verifying the IDT pointer. (that is easily avoided in modern systems with EPT)

4.5 Nested virtualization tests

One of the tests not performed by Pafish is performing virtualization instructions (calling VMXON, VMREAD etc.) and performing complex virtualization operations such as EPT or IOMMU operations.

These instructions require a significant effort to support nested virtualization. Running these instructions efficiently poses extra complexity on the blue pill hypervisor, which is not even implemented by some commercial hypervisors such as Oracle's Virtual Box.

Calling these instructions introduces additional complexity, which the normal blue pill hypervisor may not meet or may not meet within time and complexity limits.

Like the CPUID instructions, all of Intel's VMX instructions induce an exit from guest to host mode, making it mandatory to emulate and, thus, extending timing attacks to new instruction types and even mixing instructions from the family. Additionally, VMX instructions also modify the flags register in order to notify VMM software of success or failure of the instructions.

4.6 Undocumented opcode-based attacks

The x86 opcodes vary in length from 1 to 19 bytes. This massive address space leaves huge room for opcodes.

Domas (2017) mapped the opcodes and compared them with the x86 specs. Domas found many (~10^5) undocumented opcodes in the x86 architecture. It is speculated that these instructions are used by Intel's engineers for internal testing. While the behaviour of these opcodes is undefined, by definition it is not unlikely that by calling one of these opcodes one could generate a different behaviour if a red pill hypervisor is running. Further research on this subject is required.

5. Remote red pills

"Remote red pills" refers to situations in which there is a trusted third party that is available to test the inspected system.

These tests make more sense than local test cases as they do not rely on a local system that is currently considered untrustworthy.

If such a remote system is available, then these tests can provide responses with greater confidence than local tests. Unfortunately, there are many real-world cases and scenarios were such reliable third party systems do not exist.

5.1 Kennell's timing method and derived attacks

Kennell et al. (Kennell 2003) proposed a method to perform a remote hypervisor red pill based on computation side effects on the inspected system. Such side effects include TLB and cache hits and misses as well as real world time that preforming the computation consumes.

According to Kennell's method, the attested computer receives a "challenge" (computation request) from the trustworthy remote server.

The computation of the challenge causes several computational side effects such as TLB hits, TLB misses, cache hits, cache evictions, etc. The challenge contains several stages in each of these stages the side effects of the prior stage are added to the computation result. Thereby the prior stages result, and side effects affect the result of the next phase computation. To pass the test, the tested computer must not only produce the same results for the computation itself but also compute accurate results for the side effects as well. Furthermore, the entire computation (of all stages) must be completed in a short time (the time it would take a non-virtual machine to calculate). The kennell test relies on the fact that if a blue pill or an emulator is running then the computation side effects are bound to be different. Thus calculation of side effects must be done separately and consume more time. Therefore, It is impossible for the response to the challenge on an emulated machine or virtual machine to be right and arrive on time. Furthermore, since the challenge is constructed from many stages that must be computed in the correct order the Kennell test cannot be emulated on parallel machines. The Kennell test will declare the machine is not genuine if the answer is wrong or arrives too late.

Kennell's method came under direct attack the following year. Shankar et al. (2004) claimed that performance side effects are not sufficient as a method for software detection.

Kennell et al (Kennell 2004) has answered these claims and the matter rested until virtualization became commonly available in modern PCs.

Kennell's method cannot be emulated directly on modern system as modern systems are more complex than the model assumed by kennell with multiple caches. Also EPT provides much faster memory traversal even if an hyper visor is present.

Kiperberg et al (Kiperberg et al 2013 and Kiperberg et al 2015) claimed that the Kennell method can be replicated on modern PCs with hardware virtualization. This result was short-lived as Intel changed their caching algorithm the following year (between 2nd and 3rd generation of core processor). Furthermore, Intel has not shared their caching algorithms.

However, Kennell tests rely on the availability of certain algorithms, such as CPU caching algorithms, which are not commonly available. These algorithms are considered trade secrets. Furthermore, Intel has changed the caching algorithms of their core platform between the second and third generations and changed them again to combat the "meltdown" (Lipp et al. 2018) and "specter" (Kocher et al. 2018) weaknesses. Thus supporting the Kennell tests on modern hardware require reversing the architecture of the caching algorithm. It is difficult and extremely time consuming. Supporting the Kennell tests on all recent intel/AMD architectures can be a menial task that will require further research.

5.2 Network-based attacks

Because Kennell has shown that several operations take a significantly longer time when a blue pill is running, it is possible to construct a network-based rootkit using other network servers instead of a Kennell-based attestation server.

For example, if a network-based NTP (network time) server is available and can be considered trustworthy, then it is possible to construct a "network-based blue pill."

The network-based red pill scheme is as follows:

- Query a network time server.
- Perform instructions that may take significantly longer if a blue pill is present (for example CPUID) * N times.
- Query the network time server and obtain the difference.
- Compare the time required to perform CPUID against a threshold.

The time measurements can be done more accurately by pinging other computers in the network as well as the default gateway.

6. TPM-based red pills

The Trusted Platform Module (TPM) is a device that is found on most modern computer systems. The TPM includes several performance counters that take measurements during the system boot.

The TPMs are developed using hardware obfuscation methods that cannot be reversed easily.

One of the main goals of the TPM is to attest the hardware and software currently running.

This is performed using the following steps:

- Establish a trust nexus.
- Maintain a chain of trust.
- Perform TPM attestation (remote and local).

6.1 Establishing a trust nexus and chain of trust

The TPM chip itself is the trust nexus. The TPM is built using hardware obfuscations and its internals cannot be easily reversed. Despite attacks on older TPM models, there are no known attacks on recent TPM modules. If

the risk that a modern TPM chip will be hacked is accepted (or considered negligible), then the TPM can serve as a root of trust (or trust nexus) for the application. Once the TPM itself has been attested the new modern technologies such as secure boot can be used to create a chain of trust.

6.2 Maintain the chain of trust

Once the TPM has attested the BIOS or initial hypervisor it is up to the next phase to maintain trust. For example, the hypervisor instructions may be disabled in BIOS and the OS can ensure that only trusted software is loaded. Alternatively a hypervisor can be started from UEFI or MBA and ensure that only trusted OS is loaded. The trusted OS will load only trusted applications and so on and so forth.

6.3 Perform TPM attestation

The TPM can be attested directly or through trusted third party thus assuming 6.1 and 6.2 holds the system can be considered attested.

7. Conclusion and recommendation

As demonstrated above, there are numerous ways in which a hypervisor can hide and avoid detection and numerous ways in which a hypervisor can be detected.

It is recommend taking the following precautions as a malicious blue pill must be avoided at all costs:

- Install a TPM on the system and attest the UEFI BIOS.
- Start an attested trusted hypervisor that will prevent any subversion attacks.
- Update your code if new subverting attacks are discovered and published (0-days).

If a red pill is required and a primordial hypervisor cannot be installed, then the recommended red pill method is to use remote attestation methods whenever possible.

Local attestation by the local system should be avoided, especially if the malicious hypervisor's designers can review the local attestation code and design the hypervisor so that it passes the local test with high probability.

However, if local attestation cannot be avoided, then it is recommended to use the multiple cores available in all modern CPUs for time-based attestation. Such methods should be used as a last resort and remote attestation or network timing-based methods are preferred. A malicious blue pill hypervisor can block these methods; however, tampering with NMIs between cores will usually introduce noticeable latency to the system.

It is also possible that unknown opcodes can be researched to provide new red pills.

References

Averbuch, A., Kiperberg, M., & Zaidenberg, N. J. (2013). Truly-Protect: An efficient VM-based software protection. *IEEE Systems Journal*, *7*(3), 455-466.

David, A. and Zaidenberg, N., (David et al 2014), October. Maintaining streaming video DRM. In *Proc. Int. Conf. Cloud Security Manage (ICCSM)* (p. 36).

Domas, C. (2017). Breaking the x86 ISA Blackhat, USA

Garfinkel, T., Pfaff, B., Chow, J., Rosenblum, M., & Boneh, D. (2003, October). Terra: A virtual machine-based platform for trusted computing. In *ACM SIGOPS Operating Systems Review* (Vol. 37, No. 5, pp. 193-206). ACM

Kennell, R., & Jamieson, L. H. (2003). Establishing the Genuinity of Remote Computer Systems. In *USENIX Security Symposium* (pp. 295-308).

Kennell, R., & Jamieson, L. H. (2004). An analysis of proposed attacks against genuinity tests. *CERIAS, Purdue Univ., West Lafayette, IN, USA, Tech. Rep*, *27*, 2004.

Khen, E., Zaidenberg, N. J., & Averbuch, A. (2011, June). Using virtualization for online kernel profiling, code coverage and instrumentation. In *2011 International Symposium on Performance Evaluation of Computer & Telecommunication Systems* (pp. 104-110). IEEE.

Khen, E., Zaidenberg, N. J., Averbuch, A., & Fraimovitch, E. (khen et al 2013). Lgdb 2.0: Using lguest for kernel profiling, code coverage and simulation. In *2013 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)* (pp. 78-85). IEEE.

Kiperberg, M., & Zaidenberg, N. (Kiperberg et al 2013) Efficient Remote Authentication. In *Proceedings of the 12th European Conference on Information Warfare and Security: ECIW 2013*(p. 144). Academic Conferences Limited.

Kiperberg, M., Resh, A., & Zaidenberg, N. J. (Kiperberg et al 2015). Remote Attestation of Software and Execution-Environment in Modern Machines. In *2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing* (pp. 335-341). IEEE.

Kiperberg M, Algawi A, Leon R, Resh A. & Zaidenberg N. J. Hypervisor-assisted Atomic Memory Acquisition in Modern Systems (Kiperberg et al2019a) in Proceedings of 5th international conference on information system security and privacy ICISSP 2019

Kiperberg, M., Leon, R., Resh, A., Algawi, A. and Zaidenberg, N.J., (Kiperberg et a; 2019b). Hypervisor-based Protection of Code. In *IEEE Transactions on Information Forensics and Security*.

Kocher, P., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., ... & Yarom, Y. (2018). Spectre attacks: Exploiting speculative execution. *arXiv preprint arXiv:1801.01203*.

Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Mangard, S., ... & Hamburg, M. (2018). Meltdown. *arXiv preprint arXiv:1801.01207*

Ortega, A. (2016) Paranoid Fish https://github.com/a0rtega/pafish

Popek, G. J., & Goldberg, R. P. (1974). Formal requirements for virtualizable third generation architectures. *Communications of the ACM*, *17*(7), 412-421.

Resh, A., Kiperberg, M., Leon, R., & Zaidenberg, N. J. (2017). Preventing Execution of Unauthorized Native-Code Software. *International Journal of Digital Content Technology and its Applications*, *11*.

Rutkowska, J. (Ruthkowska 2006a). Introducing blue pill. *The official blog of the invisiblethings. org*, *22*, 23

Rutkowska, J. (Ruthkowska 2006b). Subverting VistaTM kernel for fun and profit. *Black Hat Briefings*.

Rutkowska, J., & Tereshkin, A. (2007). IsGameOver () anyone. *Black Hat, USA*.

Shankar, U., Chew, M., & Tygar, J. D. (2004). Side effects are not sufficient to authenticate software. In *USENIX Security Symposium* (Vol. 8, No. 3).

Zaidenberg, N. J. (Zaidenberg 2018). Hardware Rooted Security in Industry 4.0 Systems. *Cyber Defence in Industry 4.0 Systems and Related Logistics and IT Infrastructures*, *51*, (pp 135-151).

Zaidenberg, N. J., & Khen, E. (Zaidenberg 2015b). Detecting Kernel Vulnerabilities During the Development Phase. In *2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing* (pp. 224-230). IEEE.

Zaidenberg, N. J., & Resh, A. (Zaidenberg et al 2015c). Timing and side channel attacks. In *Cyber Security: Analytics, Technology and Automation* (pp. 183-194). Springer, Cham.

Zaidenberg, N., Neittaanmäki, P., Kiperberg, M., & Resh, A. (Zaidenberg et al 2015d). Trusted Computing and DRM. In *Cyber Security: Analytics, Technology and Automation* (pp. 205-212). Springer, Cham.