

Detect Kernel-Mode Rootkits via Real Time Logging & Controlling Memory Access

Slide 1 Hello

Hi, I'm Igor, thanks for coming. The title is quite long, but actually it means just "We Protect the Computer Memory".

Slide 2 Hello

The ideas I'm going to present you are based not only on my work but also on the research made by Satoshi Tanda.



Slide 3 "Intro"

About a year-and-half ago I found his blog, where he carried out a huge research about how Patch Guard can be bypassed. He is really good at reverse engineering, hypervisors, and also he is a professional developer. I found his results very interesting and suggested him an idea of how to protect PatchGuard from being defeated.

And now you'll see the result of our collaboration. Accesses to the computer memory can be illustrated as a road traffic and this traffic looks like the left picture.



Slide 4 "Memory accesses look like driving without rules"

Here we can see no traffic lights, no traffic signs, no road marking. Actually in computer memory we have a similar situation.

My idea is to protect the memory by monitoring and controlling memory accesses. As a result, it will look like the right picture, where each car is counted and any violations /ˌvaɪəˈleɪʃ(ə)n/ will be recorded.



Slide 5 “Agenda”

I’d like to start with experts’ view on some new malware trends.

I’ll tell you about how to detect malware infections at early stages.

Next, I’ll demonstrate you how I managed it by controlling memory access.

I’ll show you the advantages of new memory monitoring tool.

Finally I’ll describe my ideas about applying The Internet of Things in digital security.

Agenda

- Malware avoids detection: trends & experts’ views
- Intercepting memory access attempts: methods & projects
- The new memory interceptor MemoryMonRWX: idea & prototype
- Demos
- Future plans with IoT & Digital Security

Slide 6 “Quotation”

The researchers from these two universities raise their concerns about modern malware rootkits. We can see that memory dump analysis is becoming useless.

What protection mechanisms do we have to solve this issue?

“... malware, or more specifically, a kernel rootkit, can often tamper with kernel memory data, putting the trustworthiness of memory analysis under question”¹



1. Prakash, A., Venkatesan, E., Yin, H., & Lin, Z. (2015, October 31). On the Trustworthiness of Memory Analysis - An Empirical Study from the Perspective of Binary Execution. IEEE Transactions on Dependable and Secure Computing (TDSC), 12(5), 1545-1571. <http://dx.doi.org/10.1109/TDSC.2015.2366661>

Slide 7 “What do we have now?”

Windows has several protection mechanisms to deal with malware attacks. For example, a Driver Signature Enforcement prevents loading an unsigned driver. Another one is PatchGuard, which checks the integrity of the Windows kernel and in case of any modification PatchGuard crashes the system and we see a blue screen of death.

Unfortunately, a new malware can avoid all these protection mechanisms.

What do we have now?

Windows security features	What do we have now?
Driver Signature Enforcement	
PatchGuard (Kernel Patch Protection)	

1. McAfee. (2016, September). Threats Report. McAfee Labs. Retrieved from <http://www.mcafee.com/aboutreports/reports/mcafee-annual-threats-report-2016.pdf>
2. Singh, A. (2015, April 8). Dissecting Turtle Rootkit Malware Using Dynamic Analysis. Retrieved from <https://www.burpbox.com/about/updates/turtle-rootkit-malware-using-dynamic-analysis>

Slide 8 “What do we have now?”

3 million of new malware binaries with digital certificates show the weakness of the Driver Signature Enforcement. These two rootkits are able to defeat PatchGuard even on newest Windows. There are a lot of examples of defeating Windows PatchGuard. I’d like to show you one of them.

What do we have now?

Windows security features	What do we have now?
Driver Signature Enforcement	3 million of signed malicious binaries ¹ 
PatchGuard (Kernel Patch Protection)	New malware is able to bypass PatchGuard: <ul style="list-style-type: none">• ‘Turla’ rootkit²• TD/L4/TDSS

1. McAfee. (2016, September). Threats Report. McAfee Labs. Retrieved from <http://www.mcafee.com/us/resources/reports/threat-report-sep-2016.pdf>
2. Singh, A. (2015, April 8). Dissecting Turla Rootkit: Malware Using Dynamic Analysis. Retrieved from <https://www.sans.org/whitepapers/dissecting-turla-rootkit>

Slide 9 “Defeat and Protect PatchGuard”

I’ll show you these three scenarios.

Firstly, I will load a rootkit, which hide a process and PatchGuard reacts on this manipulation via Blue Screen of Death (BSOD).

Secondly, I will load an exploit which disables PatchGuard and hides a process.

In this case we won’t see a BSOD.

Finally, I’ll present you a new memory protector, which stops this exploit and protects the operating system.

Now we will see 3 videos. Let’s start.

Second video: This infection was the starting point for my research. I thought how to prevent this or similar exploit by monitoring memory access. Let’s see what I achieved.

Defeat and Protect PatchGuard

No	Pre-emptive Actions	Malware actions	Results & Comments
1		Rootkit is hiding the process	
2		Exploit is disabling PatchGuard Rootkit is hiding the process	
3	Memory protector limits memory access	Exploit is disabling PatchGuard Rootkit is hiding the process	

Demos 1-2-3

Slide 10 “Defeat and Protect PatchGuard”

Here are the results of these 3 scenarios. We can see that only in the final video the operating system has been protected. The new memory interceptor blocks the exploit and prevents malware infection at early stage.

Defeat and Protect PatchGuard

No	Pre-emptive Actions	Malware actions	Results & Comments
1		Rootkit is hiding the process	OS has crashed ❌ (PatchGuard has generated 0x109 BSOD)
2		Exploit is disabling PatchGuard Rootkit is hiding the process	OS has been infected ❌ (PatchGuard has been disabled, no BSOD)
3	Memory protector limits memory access	Exploit is disabling PatchGuard Rootkit is hiding the process	OS has been protected ✅ (Exploit has failed)

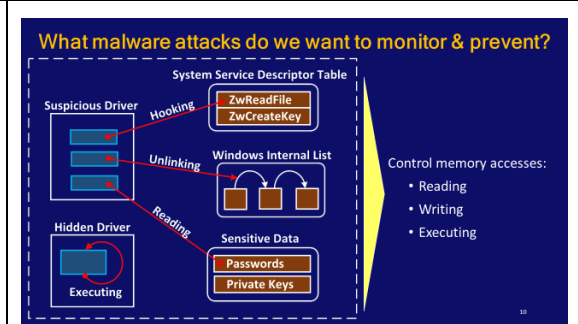
Slide 11 “What malware attacks do we want to monitor and prevent”

What sort of malware attacks can we prevent?

Let's think of some scenarios that this memory interceptor will face and try to specify its requirements. Modern malware can read sensitive data from the memory.

It can change the memory content, for example, by hooking, unlinking, or patching.

And also the unknown code execution can create a new thread.

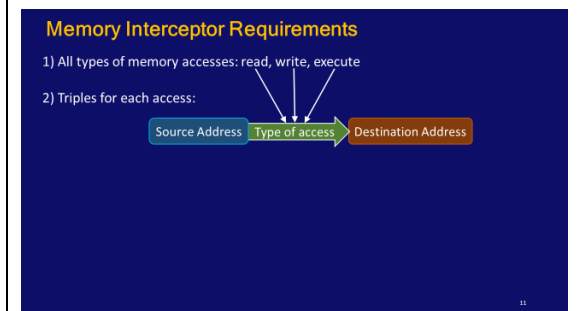


Slide 12 “Memory Interceptor Requirements”

Well, we need to control all possible memory accesses: reading, writing, and executing.

So, for each memory access, we will log the following triple: source address, destination address, and the type of accessing.

It will be also good to filter all these accesses, for example to monitor memory access only from a suspicious driver or to limit access to the sensitive data in the memory.



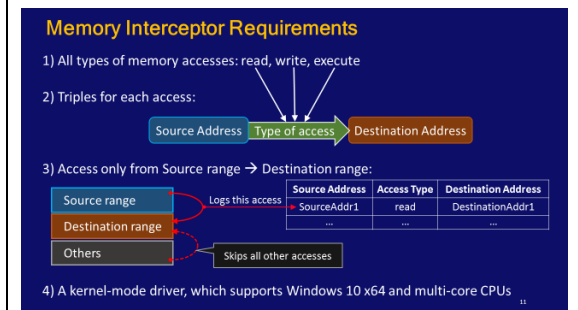
Slide 13 “Memory Interceptor Requirements”

I propose to divide memory into three parts: Source range, Destination range and others. Memory protector controls memory accesses only from the source range to the destination range and skips all the rest.

Source range can include, for example, the beginning and ending addresses of the entire driver in memory. Destination range can contain the range of addresses and can also include just one address to reveal a patching attack.

What else do we need? Compile and load this memory interceptor like any other driver.

And keep it up to date. This tool has to support the newest Windows 10 64 bit and multi-core CPUs. That's it. But how to achieve this?

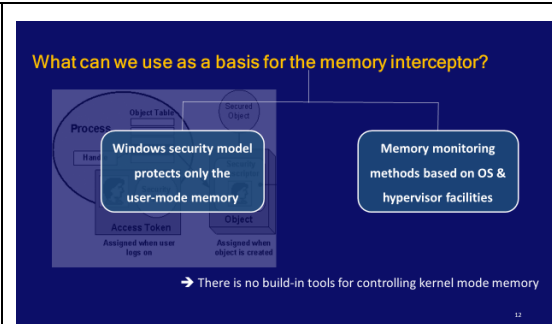


Slide 14 “What can we use as a basis for the memory interceptor?”

Let’s move on to the question – what tools and technologies are available at the moment? Windows protection mechanisms such as a process security and access rights regulate only the user-mode memory.

There are no built-in tools for controlling kernel-mode memory so far.

So what can help us to prevent unauthorized access in the kernel mode?

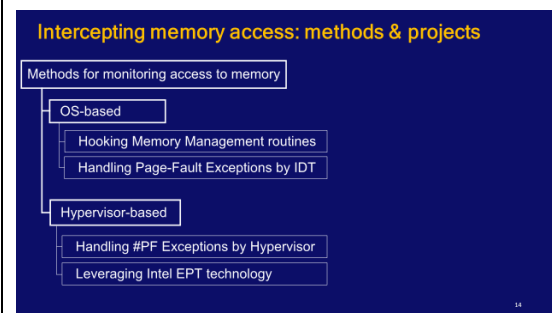


Slide 15 “Intercepting memory access: methods & projects”

The most interesting memory monitoring methods and projects are here.

All methods can be divided into two groups: based on operating system facilities and based on hardware virtualization technology. OS-based methods can be separated into two subgroups: based on hooking memory management functions, and based on handling page fault exceptions.

Hypervisor-based methods can also be divided into two groups. The first one handles page fault exceptions, while the second one applies a new virtualization extension – EPT to track all memory accesses.



Slide 16 “Intercepting memory access: methods & projects”

And here is the table of existing projects, which are based on the corresponding methods.

We can see that none-of-them controls all three memory accesses at the same time. So, I decided to create a new memory monitoring tool, which will meet all these requirements. I named it Memory Monitor of Read- Write- and eXecute access or just MemoryMonRWX. So what technology can be used as a basis for this new tool?


Intercepting memory access: methods & projects		
Methods for monitoring access to memory		
OS-based	Project title, year	Read/Write/Execute
Hooking Memory Management routines	SPIDER, 2013	+/-/-
Handling Page-Fault Exceptions by IDT	SecVisor, 2007	-/+/-
	HyperSleuth, 2010	+/-/-
Hypervisor-based	CXPIInspector, 2013	-/-/+
Handling #PF Exceptions by Hypervisor	HyperTap, 2014	-/+/-
Leveraging Intel EPT technology	DRAKVUF, 2014	-/-/+
	MemoryMonRWX, 2017 (The proposed system)	+ / + / +

Slide 17 “New Advanced Technology: Intel VT-x with Extended Page Tables (EPT)”

The EPT technology, which has recently been integrated in Intel processors, seems very perspective. It is called VT-x with Extended Page Tables (EeePeeTee). I’ll give you a brief overview of this technology, its paging structures, and how to use EPT to monitor and control the memory access.

New Advanced Technology:
Intel VT-x with Extended Page Tables (EPT)

- EPT Overview
- EPT paging structures
- Applying EPT to monitor & limit memory access

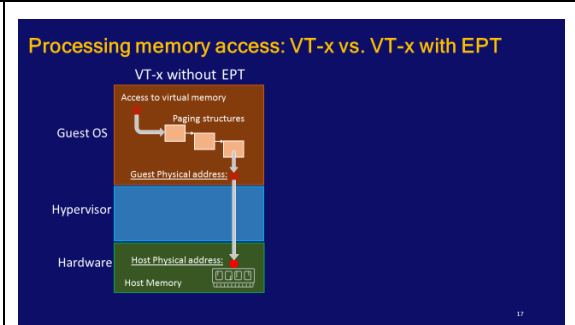


EPT plays the role of traffic lights for memory accesses

14

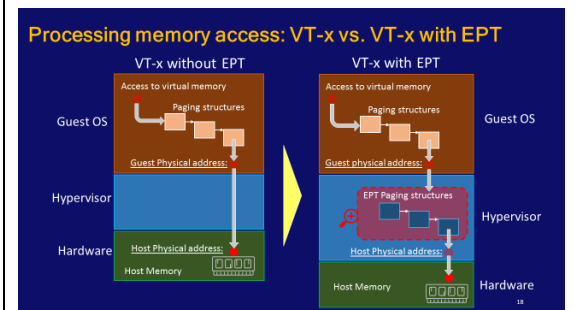
Slide 18 “Processing memory access: VT-x vs. VT-x with EPT”

Let’s look at the mechanism of address translation between the guest memory and the host memory. Without EPT once guest memory is accessed, the translation between the guest virtual address and guest physical address starts happening. Then we receive the guest physical address. Without EPT this very guest physical address is used to access the host memory.



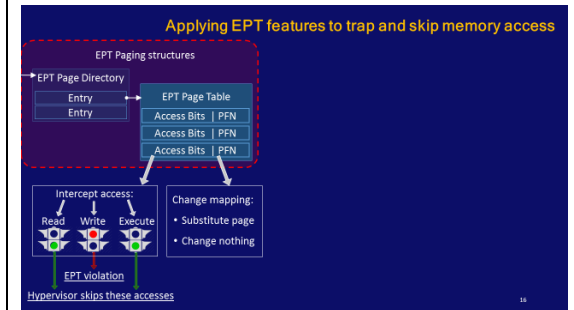
Slide 19 “Processing memory access: VT-x vs. VT-x with EPT”

Using EPT we get an additional or a second level address translation. This case is on the right. EPT plays the role of an intermediary during address translation. The guest mapping is the same. But now, we receive the host physical address by traversing the EPT structures. These structures are very similar to the paging structures in the protected mode and determine the mapping between the guest memory and the host memory. Let’s zoom in on EPT paging structures.



Slide 20 “Applying EPT features to trap and skip memory access”

Now we are looking at the fourth EPT tables or leaves of EPT tree, which are called the EPT Page Tables. This table includes a lot of entries, which contain information about a single 4 Kilobyte memory page. Each EPT entry includes access bits and page frame number PFN. This PFN is a host physical address of the corresponding memory page. EPT access bits indicate whether read- write or execute-access is allowed for this page. Intel manual says that ‘any attempts at disallowed accesses will involve EPT violation and cause VeeM exits’. Also the hypervisor can limit access to this page by changing its PFN value. In order to intercept or skip each memory access, we configure the access bits in the corresponding EPT page table entries.

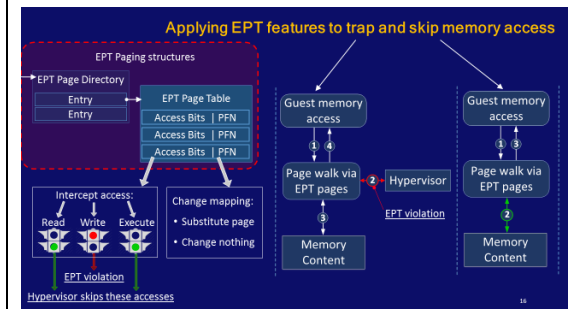


Slide 21 “Applying EPT features to trap and skip memory access”

There are two main scenarios of using EPT. If this memory access is not allowed, it involves EPT violation and causes VM exit. At this moment the hypervisor is able to log access to this memory page. After that control goes to physical memory and comes back to the guest, see central figure.

But, if this memory access is allowed according to the EPT bits, the hypervisor skips this access.

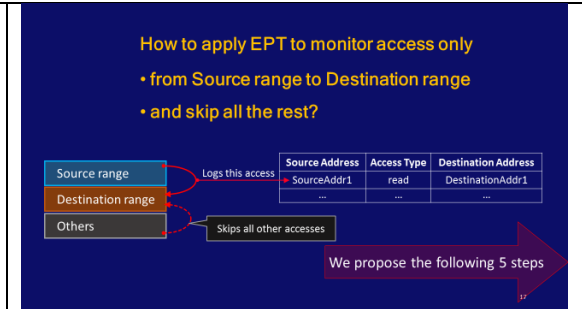
We can see, that by changing EPT setting a hypervisor can implement various algorithms. This is exactly what we’ve been looking for.



Slide 22 “How to apply EPT to monitor access”

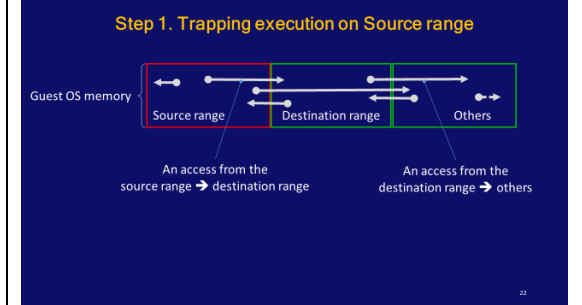
Now let’s move on to the final question how to apply these EPT features to control access only from the source range to the destination range and skip all the rest.

We can combine skipping and trapping memory access in the following five steps.



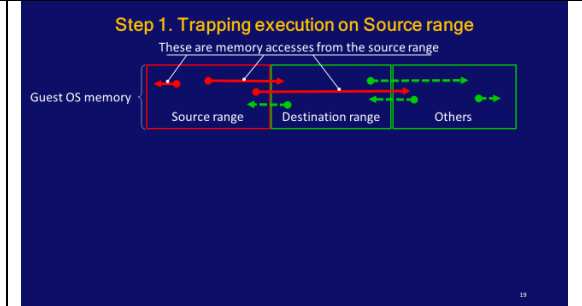
Slide 23 “Step 1. Trapping execution on Source range”

First of all, we divide the memory content into three sections: source range, destination range, and others memory addresses. The memory access attempts are shown by arrows. Here we have various accesses. For example, this is the access from the source range to the destination range, that one is from the destination range to others etc.



Slide 24 “Step 1. Trapping execution on Source range”

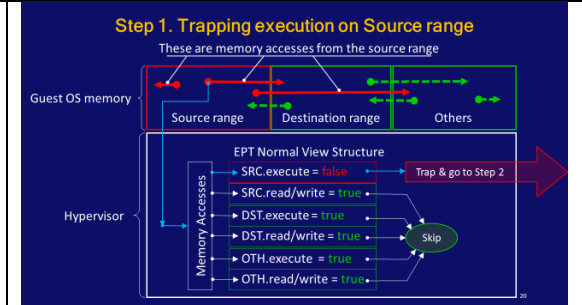
We are interested only in accesses from the source range, which are in red. We need to skip all other access, which are from the destination range and others. They are in green.



Slide 25 “Step 1. Trapping execution on Source range”

To achieve this, we can use the following EPT structure as a trap. This is EPT normal view structure. In this structure only execution access bits on the source range are disallowed. All other accesses are allowed and will be skipped.

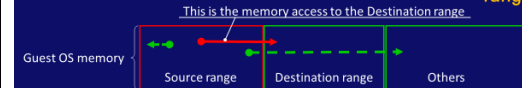
Now any code execution on the source range will involve EPT violation, which causes VM Exit. Hypervisor will trap it and we go to Step 2.



Slide 26 “Step 2. Process VM-Exit to separate access to the Destination range”

Control goes to the hypervisor. At this point we have accesses only from the source range. We are focusing only on the accesses to the destination range. We are filtering out all other accesses.

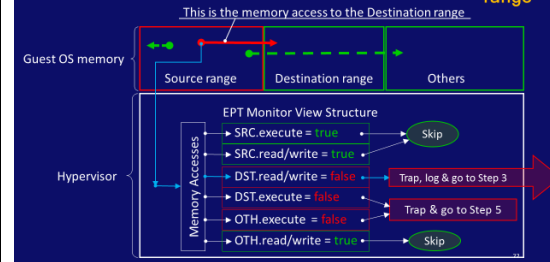
Step 2. Process VM-Exit to separate access to the Destination range



Slide 27 “Step 3. Process VM-Exit, because of access on Destination range”

In order to achieve it we use another EPT structure as a second trap. This is EPT monitor view structure. This structure disallows accesses to the destination range and skips all other accesses. Any access to the destination range generates a VM Exit again and we move on to Step 3.

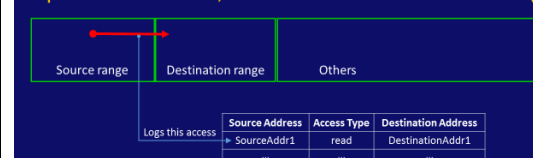
Step 2. Process VM-Exit to separate access to the Destination range



Slide 28 “Step 3. Process VM-Exit, which occurs because of an access on DST”

At this point the control goes to the hypervisor again. Now we can log all related information: the source address, the destination address, and the type of access. Also we can optionally protect the destination data from being read {RED} or overwritten. In order to achieve it we apply the following 3 procedures:

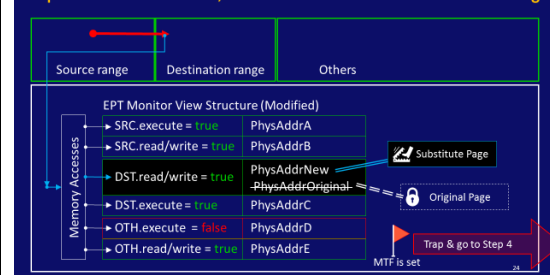
Step 3. Process VM-Exit, because of access on Destination range



Slide 29 “Step 3. Process VM-Exit, which occurs because of an access on DST”

(1) We change PFN value of the secret page to the substitute one, for example, to the null page. This helps to protect the original data. (2) We allow read- and write- access to this page. (3) We set Monitor Trap Flag (MTF). Setting MTF flag enables the system to generate VM Exit after executing each instruction. As a result, after the guest operating system reads the replaced page by

Step 3. Process VM-Exit, because of access on Destination range



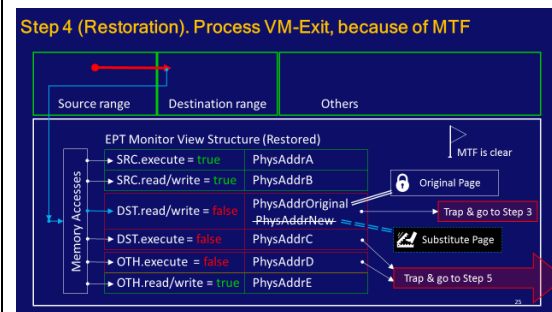
executing just one instruction, the control goes automatically to the hypervisor again and we move on to Step 4.

Slide 30 “Step 4 (Restore setting). Process VM-Exit, because of MTF”

We’ve just avoided an access to the secret data. We need to restore the original settings.

We apply the following 3 procedures: (1) We set PFN value to the original page with secret data. (2) We block read- and write- access to this page. (3) We reset Monitor Trap Flag. This is the original EPT monitor view structure.

After that any access to the destination range will generate VM-Exit, and we move on to Step 3 again. Any execute access on other range will also generate VM-Exit, and we move on to Step 5. We’ve already considered Step 3. Now we go to the final Step 5.

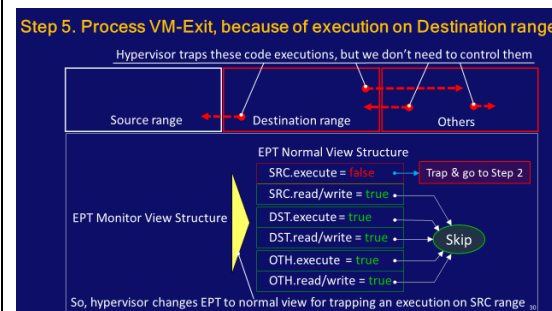


Slide 31 “Step 5. Process VM-Exit, because of execution on Destination range”

Now we trap an execution on destination range and others memory addresses.

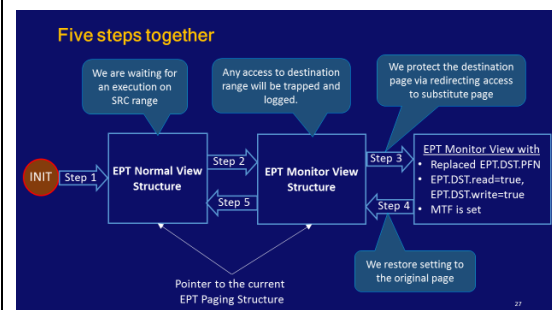
We double check if this VM Exit address belongs to the source range. If it doesn’t, it means that this code is out of our control and we don’t need to control it.

So we change EPT from monitor view to normal view to be ready for trapping a new code execution on the source range. That’s it.



Slide 32 “Five steps together”

Here is the summary of the interaction between two EPT structures: normal view and monitor view. Firstly, we set EPT normal view to intercept any access from the source range. Secondly, after trapping an execution on the source range, we change EPT pointer to the EPT monitor view. It helps us to log accesses only to the destination range. Thirdly, we can optionally protect the page by redirecting access to another

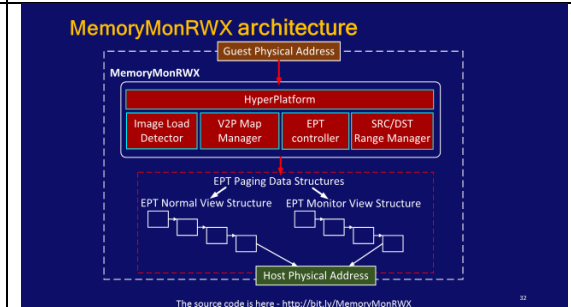


page. And after an access to the substituted page we restore setting, this is step 4 and go back to the EPT monitor view. If we trap the execution, which we don't need to control, we change EPT to the normal view.

Slide 33 “MemoryMonRWX architecture”

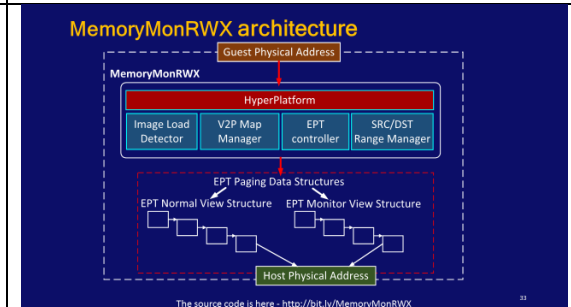
We checked all these ideas by developing a MemoryMonRWX hypervisor.

Now let's look at its architecture. MemoryMonRWX includes the following five components: HyperPlatform, Image Load Detector, Virtual-to-Physical Map Manager, EPT controller, and Source/Destination Range Manager.



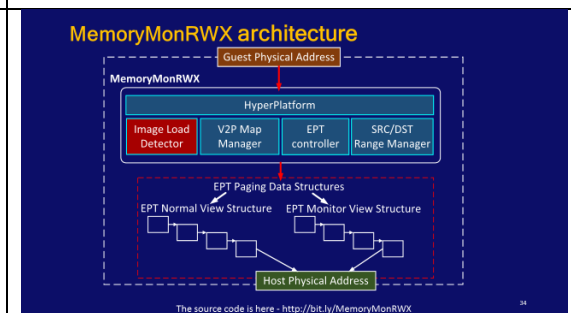
Slide 34 “MemoryMonRWX architecture”

HyperPlatform is the main component of this system, which is a bare-metal hypervisor and it is designed for intercepting various events in the operating system.



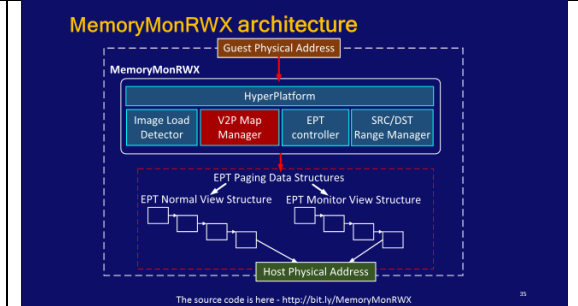
Slide 35 “MemoryMonRWX architecture”

Source and destination address ranges are formed by *Image Load Detector*. We set the addresses of recently loaded drivers as a source range. Destination range includes, for example, address of PoolBigPageTableSize, which we've seen in the video at the beginning.



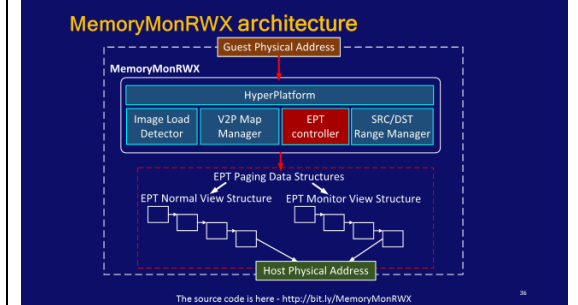
Slide 36 “MemoryMonRWX architecture”

Virtual-to-Physical Map Manager maintains mapping between virtual and physical addresses. Actually it is a supplementary component.



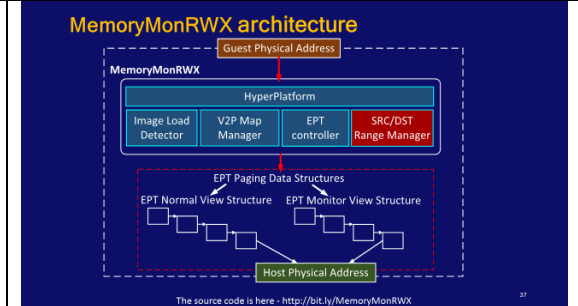
Slide 37 “MemoryMonRWX architecture”

EPT controller is responsible for updating EPT Structures and handling EPT violations.



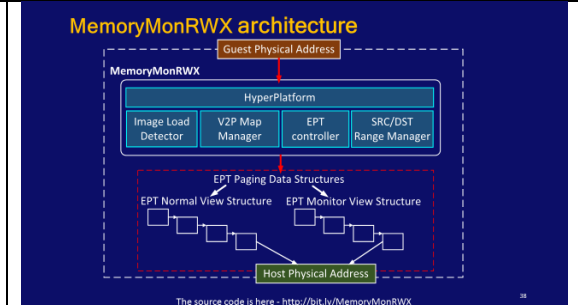
Slide 38 “MemoryMonRWX architecture”

And the last one is the *Source/Destination Range Manager*. This manager asks the EPT controller to update EPT access bits for the source and destination ranges.



Slide 39 “MemoryMonRWX architecture”

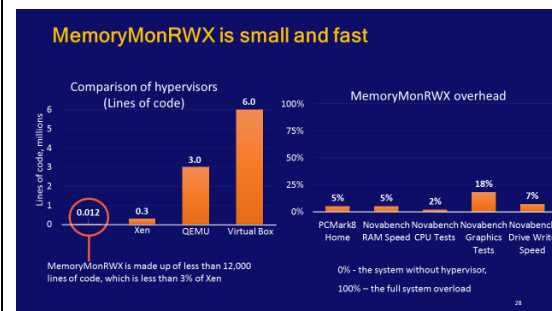
Architecture is quite simple, the source code is uploaded here.



Slide 40 “MemoryMonRWX is small and fast”

MemoryMonRWX is designed to be small. It is made up of less than 12,000 lines of code, which is less than 3% of XEN, for example. Also, it can be compiled on Visual Studio without any problems. It can be debugged with WinDbg just like a common Windows driver.

MemoryMonRWX is very fast. The results showed that performance degradation was less than 10% in all tests except only one. It is quite positive.



Slide 41 “Conclusions”

MemoryMonRWX is a very promising project, it works in the newest Windows 10 and supports multi-core CPUs. It provides integrity and confidentiality for both code and data in memory. It can prevent data leakage attacks, unauthorized software copying, and protect PatchGuard from being disabled. MemoryMonRWX is just a fishing rod to catch memory disclosure attacks, I’m planning to develop a fast fishing boat to catch more and more. Also I’d like to show you my future plans.

Conclusions

- MemoryMonRWX logs & controls all memory accesses in a real time
- It is a hypervisor, which supports newest Windows 10 x64
- MemoryMonRWX can be used in various tasks:
 - Trace malware activity
 - Protect memory of 3rd party drivers

Illustration of a fishing boat catching a large fish, symbolizing the project's goal of catching memory disclosure attacks.

Slide 42 “Apply Raspberry Pi to Acquire Physical Memory Dump & Detect Hidden Software”

During my research I usually ignore hardware tools for digital forensics and incident response, but now look at this device.

This hardware can acquire physical memory dump by connecting to the ExpressCard slot. It’s produced by WindowsSCOPE company and it costs about \$8000.

I suppose, that the device, which only dumps memory, can cost less.

Acquire Physical Memory & Detect Hidden Software by Raspberry Pi

Illustration of the WindowsSCOPE CaptureGuard Physical Memory Acquisition Hardware, an ExpressCard device.

CaptureGuard Physical Memory Acquisition Hardware
\$7,799.00

“This is an ExpressCard device capable of imaging the physical memory of the computer it’s connected to. Creates dump files in the standard WinDD format.”

1. CaptureGuard Physical Memory Acquisition Hardware - ExpressCard, WindowsSCOPE, <http://www.windowsscope.com/product/captureguard-physical-memory-acquisition-hardware-expresscard/>
2. Acquiring Windows 7 RAM kernel with disk attacks, SuperSID, http://www.idr.org.uk/Software/Tools/submitting-to-information-security-bulletin/pid_707.pdf

Slide 43 “Apply Raspberry Pi to Acquire Physical Memory Dump & Detect Hidden Software”

My idea is to apply IoT platform such as Raspberry Pi for the acquiring and processing memory dump. For example, it is possible to use FPGA devolvement platform Spartan-the-third to connect Raspberry Pi into ExpressCard slot. This Spartan platform has been successfully used in this project. Also after dumping we can apply Raspberry Pi to process the memory dump automatically as well as to send a dump via the Internet. All in all, it will cost about 10 times cheaper and have more features.

Acquire Physical Memory & Detect Hidden Software by Raspberry Pi

The diagram illustrates a hardware setup for acquiring physical memory. It shows a 'CaptureGUARD Physical Memory Acquisition Hardware' unit (labeled 'Windows SCOPE') connected to a 'CardBus FPGA dev platform' (Xilinx Spartan-3 400²) and a 'Raspberry Pi 3 Model B'. The cost breakdown is as follows:

Component	Price
CaptureGUARD Physical Memory Acquisition Hardware	\$7,799.00
CardBus FPGA dev platform (Xilinx Spartan-3 400 ²)	\$295.00
Raspberry Pi 3 Model B	\$35.00
Total	\$330.00

Lower price with more features

1. CaptureGUARD Physical Memory Acquisition Hardware - ExpressCard, WindowsScope, <http://www.winbitscope.com/usb/usbguard-physical-memory-acquisition-hardware-expresscard/>
2. A. Amelino, B. Amelino, C. Subverting Windows 7 AAE kernel with USB attacks, Sigint ECC Lab, <http://www.sigintlab.org/usb/usbguard-physical-memory-acquisition-hardware-expresscard/>, July 2016.

Slide 44 “Implantable Medical Devices as a Target of Cyber Attacks”

Another idea is to protect medical devices from being hijacked.

A lot of people need wireless devices like insulin pumps, implantable cardioverter defibrillator ICDs and many others to survive and improve the quality of their life. Patients and doctors use this wireless connection to control the devices and change their runtime parameters. Unfortunately, this wireless communication is vulnerable, here you can see some examples of resent cyber-attacks on implantable devices, performed through wireless connection. The idea is to protect this wireless communication.

In the USA in upwards of 2.5 million people depend on wireless implantable medical devices, which all can be hijacked remotely¹

The diagram shows a person's arm with a 'GLUCOSE MONITOR' and an 'INSULIN PUMP'. A red arrow points from the text 'In the USA in upwards of 2.5 million people depend on wireless implantable medical devices, which all can be hijacked remotely¹' to the devices. A box titled 'Consequences of attacks on implants:' lists the following:

1. Pacemakers by St. Jude Medical Inc. (2016)
 - manipulation of beat rates
 - battery drain
2. OneTouch Ping Insulin Pump by J&J (2016)
 - unauthorized insulin injections

1. A. A. S. (2016), Cybersecurity of wireless implantable medical devices - <http://ogfopen.proquest.com/doc/1796055559.html?MMT-ABS>

Slide 45 “Implantable Medical Devices as a Target of Cyber Attacks”

We are planning to provide complex security for these devices. We propose to choose for example length of crypto key according to the device specifications. We are thinking of a prevention system to reveal wireless denial-of-service attacks on these devices. Thank you.

Protection of Wireless Implantable Medical Devices

The diagram illustrates a secure communication channel between a 'Vulnerable Implant' and a 'Protected Implant'. The process involves the following steps:

- Input Implant's technical specifications
- Choose the lightweight crypto cipher
- Verify the firmware

The diagram also shows the team members and their roles:

- Our Team:**
 - Veronika Domova: Software developer, Sweden
 - Igor Korkin, Ph.D.: IoT and Industrial Cyber Security

Part 1. We can see that the renamed process has been hidden and in such situations PatchGuard always causes BSOD to prevent users' activity in the infected system.

Part 2. Let's check if we get a BSOD now. Usually it crashes the system in less than one hour. But can we see a BSOD now? No, it means that PatchGuard did not stop this illegal modification, though it should have.

Part 3. Let's restore the original system. Now I load the memory interceptor, next launch the disabler of PatchGuard and you can see what we get now. Now the exploit is failing. This new memory interceptor substitutes this sensitive value with a zero one. So PatchGuard is still ON, it hasn't been disabled.