

CONFERENCE SPEECH “ACCELERATION OF STATISTICAL DETECTION OF ZERO-DAY MALWARE IN THE MEMORY DUMP USING CUDA-ENABLED GPU HARDWARE”

<p>Slide 1 Hello</p> <p>Hi everyone! I’m Igor. Thanks for coming. This is the topic of my annual presentation.</p> <p>The title is rather long but actually it means just speedy detection of unknown malware using statistics.</p> <p>Let’s start.</p>	<p>Acceleration of Statistical Detection of Zero-Day Malware in the Memory Dump Using CUDA-Enabled GPU Hardware</p> <p>Igor Korkin, Ph.D. Iwan Nesterow Independent Researchers 2016</p>
<p>Slide 2 “Agenda”</p> <p>You know, nowadays the vast majority of cyber-attackers are using various malware drivers to infect systems and it is very important to reveal kernel-mode infections. Today, I will outline the existing ways of drivers detection and how they can be bypassed. I’ll present the idea of “Highest Stealth Malware” or “HighStem”, which includes common anti-forensic techniques and their future improvements. Then, I’ll give you some ideas how to detect suspicious drivers under these difficult conditions. And finally, I’ll show you how to speed up the analysis of memory dump.</p>	<p>Agenda</p> <ol style="list-style-type: none">1. Motivation2. Analysis of drawbacks of drivers detection3. HighStem prototype4. Drivers detection in the memory by separating code from data5. GPU & CPU powered dump analysis

Slide 3 “5 Years”

During last 5 years various companies from 69 countries were severely attacked by sophisticated malware. The attackers collected information from hundreds of high-profile victims from government agencies and embassies, institutions involved in nuclear and energy research, oil, gas and even aerospace industries. The most awful thing is that nobody was aware of the breach. I believe that it is possible to detect such malware attack much earlier and these are my findings. First of all, let's look at the modern malware, which is usually well-targeted and well-prepared.

5-year cyber espionage attack



Slide 4 “2 wells”

You know, for example, ‘BlackEnergy’ trojan infiltrated into the power grid and water distribution systems. Another example is ‘Havex’ malware, which caused significant damage to a steel mill plant in Germany two years ago. You know even in the US now there is about 20% increase in computer security incidents in the nuclear power stations. The most dramatic thing is that hackers can plant malware, which can be simultaneously executed to disrupt various systems and cause cascading failures.

By and large such malware code is always well-prepared to its detection. The point is that hackers know very well how anti-viruses work and they apply various anti-forensic techniques to avoid malware

Modern malware in modern world

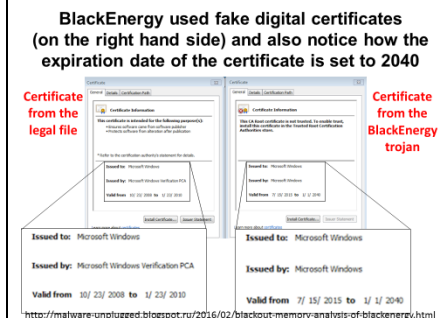
Well-targeted	Well-prepared
BlackEnergy trojan	— Uses 0-day exploits
Havex malware	— Contains fake digital certificates
+20% rise of incidents	— Applies anti-forensics tricks

Detection becomes too time-consuming

detection. Nowadays we need to detect hidden malware, under these new conditions, under the deliberate countermeasures. These three aspects of malware, are indicative of a higher level of adversary's involvement.

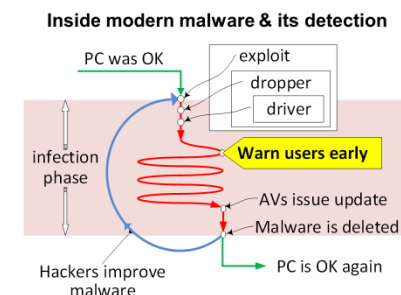
Slide 5 “fake-digital-certificates”

For example, malicious binaries can contain fake digital certificates even from Microsoft, so this kind of ‘authentication process’ becomes absolutely senseless. You can see that malware authors are clever enough to be one step ahead. We can say modern antiviruses have no ability to react swiftly against highly sophisticated malware attacks. Let’s look inside the modern malware and see what’s what.



Slide 6 “Inside Modern Malware”

Malware usually includes an exploit, a dropper and a malware driver itself. The exploit is aware of vulnerabilities and gains control of a computer system. The dropper installs driver in the operating system. After that the malware driver is loaded and the system becomes infected. After-a-while, anti-virus companies create new signatures or new updates. And only after the installation of these updates the malware will be deleted and finally our computers will be okay again. But, during this infection phase, users do not know that their computers are infected. And, no-one-knows what to do until antivirus guys



prepare their reports. Revealing of modern malware is very time-consuming. You see, sometimes it can take even several months.

Look at the slide, the blue curve shows the malware “life cycle”. Hackers always improve their ‘products’.

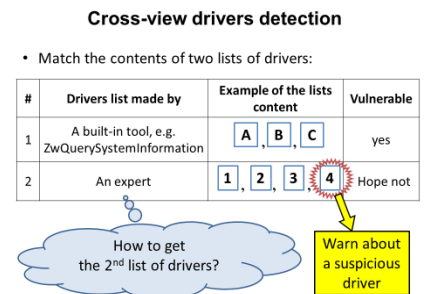
My idea is to warn users about the suspicious driver as early as possible. Here! Look at the yellow arrow.

This early notification of a suspicious driver can help users to prevent losing data, which is the most important thing. And, it could initiate the ‘incident response’ much earlier.

Slide 7 “Cross View Drivers Detection”

To clarify the problem with drivers-detection let me tell you about a cross-view-detection-approach.

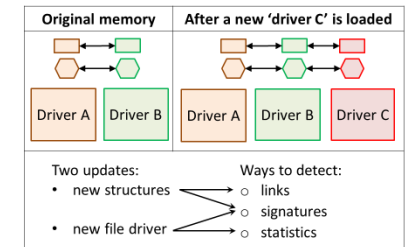
This well-known approach is based on comparing two lists of drivers information. These lists have to be based on two different sources of drivers data. An anomaly could be found by periodically comparing the content of these two lists. The first list is usually filled-in by the Windows built-in tool. A malware driver usually bypasses this list, because otherwise it can be easily detected by any user. We usually assume that hackers don’t know about all other lists, but actually clever hackers succeed in bypassing all the lists. So this is the main question – how to fill-in this hypothetical second list, which will be resilient to common anti-forensic techniques?



Slide 8 “After a driver is loaded”

To answer this tough question let’s look at what has been changed or added in the operating system, after some new driver is loaded. The information about it is automatically added into several system lists and its file is loaded from the hard disk into the memory. As a result, we’ve got new structures in the lists and new file in the memory. Both of these changes can be used to create various lists of drivers.

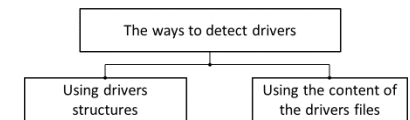
Using updates in the memory content as a source for drivers detection



Slide 9 “Classification of methods to detect drivers”

According to these changes all detection approaches can be classified into two groups: using drivers structures and using driver’s files in the memory.

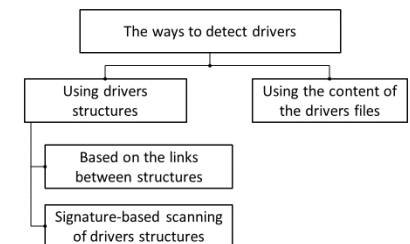
Classification of drivers detection



Slide 10 “Classification of methods to detect drivers”

Structures can be detected using links and by signature based scanning.

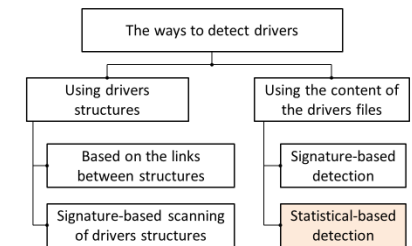
Classification of drivers detection



Slide 11 “Classification of methods to detect drivers”

Files can be revealed also by using signature-based approach and a statistical-based approach. For me the statistical-based detection is the most perspective way. But, now let’s start with the lists based on the links between structures.

Classification of drivers detection

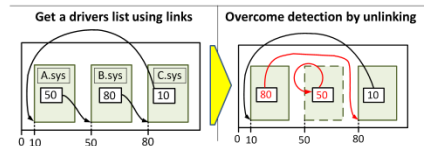


Slide 12 “Lists based detection”

Here you can see a set of popular lists and corresponding structures, which store drivers information. Various detection functions acquire information about drivers by walking through different lists, but their structures are very similar. Each driver has its own structure, which contains information-about-it, such as the name of the driver, its size etc. All structures are linked in the list. This is a common way to store data in computers. Here you can see an example of memory content with three linked structures. I’ve marked the links between structures by the following numbers – 50, 80 and 10. The Windows function gets the list of drivers by walking through these numbers. It’s just an example. But a malware driver can bypass its detection by unlinking its structure from the list. Unlinking means just modifying pointer values between neighboring structures, it is the typical operation in the lists. This unlinking attack yields two results. The built-in tool and other links-based tools are not able to find this driver; and this unlinking doesn’t cause the

Detect drivers using drivers lists

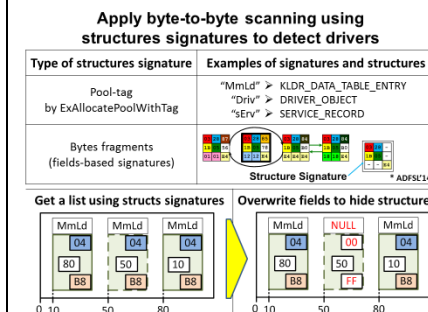
Drivers list names	Name of structure
PsLoadedModuleList	➤ KLDL_DATA_TABLE_ENTRY
ObjectDirectory	➤ DRIVER_OBJECT
Service record list by SCM	➤ SERVICE_RECORD
Threads from 'System'	➤ ETHREAD
Recently unloaded drivers	➤ UNLOADED_DRIVERS



reboot, its hidden for common user.

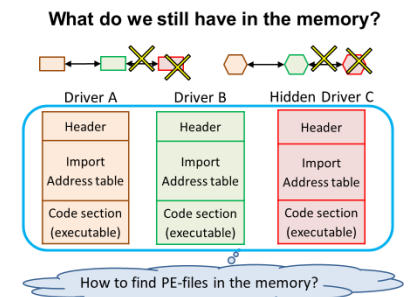
Slide 13 “Structures signatures”

But, if a structure is hidden-just by using unlinking it can be revealed. Drivers structures have typical fragments or signatures, which are the same for all structures from one list. We can reveal all structures by using byte-to-byte signature scanning and it's not important whether they are linked or unlinked. There are two types of signatures, based on the pool-tag and based on the typical bytes fragments. The pool-tag includes 4 bytes and is added to the structure automatically by the OS, using this Windows function. Look! Here are the examples of pool-tag signatures, and they are quite readable. Apart from the pool-tag, signatures can be based on the fact that some bytes in all structures are the same. These fields-based signatures include only the bytes, whose values are the same for all structures, which are in one list. Here are the examples of these kind of signatures. But a malware driver can bypass all these signatures by partially overwriting the content of the unlinked structure. And as a result signature based scanning is not able to recognize or detect the modified structure. Thus, simultaneously applying unlinking attack and overwriting makes drivers' structures absolutely useless for creating that second list of drivers.



Slide 14 “What do we still have in the memory”

So what can we use instead? What do we still have in the memory? Neither structures nor links. But we still have the drivers files loaded in the memory and we can find them. All drivers files like Portable Executable or PE-files include typical fragments. Using byte-to-byte signature scanning we can reveal all drivers files. We can use the following parts of PE-file for detection: the file header, the import table and the code section.



Slide 15 “PE signatures”

The first signature is based on ASCII strings, which are located in the file header and the import table. The most famous ASCII strings are "MZ" “PE” and “This program cannot be run in DOS mode”. Also, by searching names of Windows functions we can reveal the import table and then the whole driver file in the memory. And the last one is based on the fact that drivers files include a code section. This code has got fixed byte fragments, for example, the beginning of function – prologue and ending of function – epilogue. Here is an example of byte patterns for prologue and epilogue. But, after a driver has been loaded, it can overwrite the ASCII fragments and also use obfuscation technique to bypass the bytes signature.

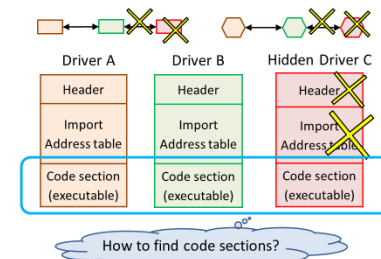
Apply byte-to-byte scanning using features of PE-file to detect drivers

Driver as a PE-file includes:	PE-file features		Countermeasures
	Type of signature	Examples	
Header	ASCII Strings	'MZ', 'PE', 'This program cannot be run in DOS mode'	Data overwriting
Import Address table	ASCII Strings	'ZwOpenFile'	
Code section (executable)	Bytes combination (prologue & epilogue)	BBFF MOV EDI,EDI 55 PUSH EBP 8BEC MOV EBP,ESP 8BE5 MOV ESP,EBP 5D POP EBP C20400 RET 4	Code obfuscating & packing

Slide 16 “What do we still have in the memory?”

Ok, What can we use instead? Look! What do we still have in the memory? No structures, no links, no signatures in PE-file. But we still have the executable code in the memory, because malware needs to be executed. If we are able to distinguish the executable code from other memory fragments it will really help us. So, the challenge of completing the second drivers list comes down to the recognition task between the executable code and other data fragments in the memory.

What do we still have in the memory?



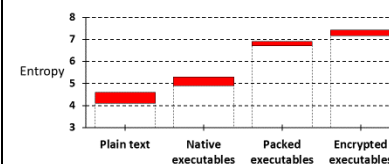
Slide 17 “Entropy separates data types?”

To solve this task, we will operate with statistics or secondary values, which are calculated from the memory content. The best-known statistic to evaluate memory content is the binary entropy, which can be calculated in the following way. Apart from binary entropy, which uses only one byte, we can also calculate entropy using two bytes or bigram analysis, as well as using three bytes or trigrams. You know that by comparing entropy values we can distinguish different data types, like a plain text, encrypted data and native executables. Higher entropy values can indicate the encrypted fragments or the executable fragments. Lower entropy corresponds to data structures. Entropy is usually calculated for the whole file to check if this file is packed or protected. Also by using byte-to-byte or a sliding-window approach we can

Using binary Entropy to separate data types

Definition:
$$S = - \sum_{i=1}^{255} p_i * \log_2 p_i$$

 p_i – the frequency of each byte value in the file.

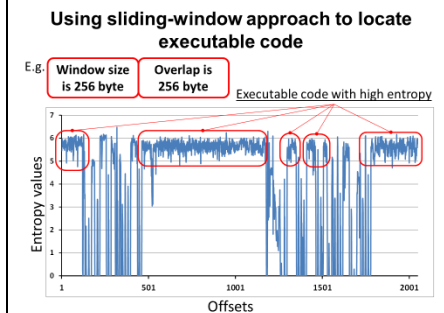


*Using Entropy Analysis to Find Encrypted and Packed Malware by R. Lyda & J. Hamrock

locate the encrypted fragments and the executable exploits in various sources, like PDF-files.

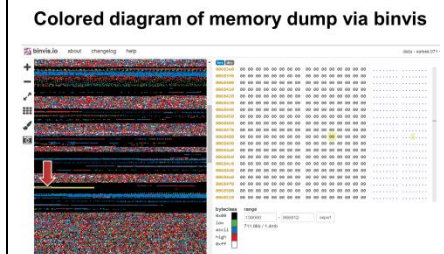
Slide 18 “Sliding-window approach”

We can apply this idea for analysis of memory dump. We can calculate entropy values in a sliding window using various windows size and their overlapping and can obtain a regional information density as a function of the byte offset. This is an example of such a function for memory dump fragment. We can see that the executable fragments have high entropy values. Using this distinguished feature we can easily separate an executable code from other memory fragments.



Slide 19 “Using visualization of the memory”

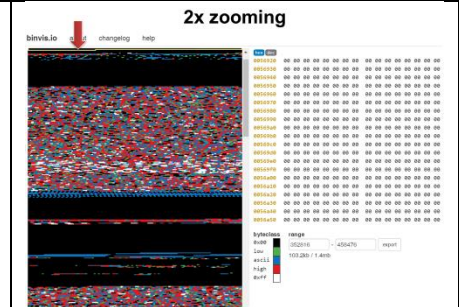
Another idea of data visualizing is coloring each byte in the dump files according to its hexadecimal value. Here is an example of visualization of a memory dump fragment using the online service binvis.io. We can easily locate null pages in the memory and distinguish particular sections of drivers files. Black blocks are corresponding to null pages. The blue values are corresponding to ASCII fragments. Solid multicolored blocks are corresponding to the executable code, which we are looking for.



Slide 20 “Using visualization of the memory”

Let’s zoom-in to specify the analysis of the memory region.

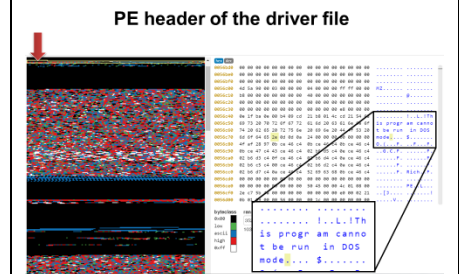
Now we can easily distinguish various sections of PE-file.



Slide 21 “Using visualization of the memory”

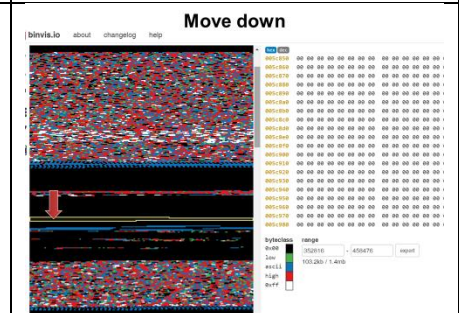
This is the beginning of PE-file with well-known ASCII signature.

Now we are scrolling down the dump file to cover all its important fragments.



Slide 22 “Using visualization of the memory”

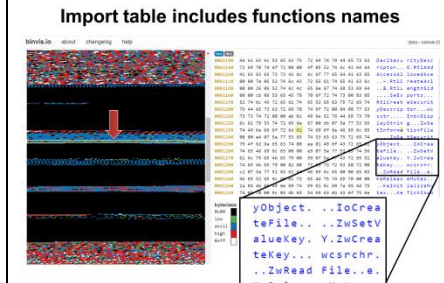
This multicolored fragment is corresponding to the executable code and we need a disassembler to analyze its logic. Let’s scroll down.



Slide 23 “Using visualization of the memory”

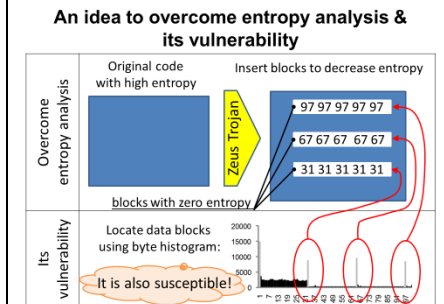
The section with blue blocks is corresponding to the import table. We can see the functions names here.

OK, you see that the analysis of entropy plots mentioned before and data visualization can really help to reveal executable code fragments. But, these statistical methods are also vulnerable to malware countermeasures. If malware driver is able to decrease its high entropy values it becomes hidden again.



Slide 24 “Countermeasures for statistical detection”

As a result, we will see in colored diagram more black blocks. The executable code which we are looking for becomes diluted. The authors of Zeus Trojan already proposed this idea of code dilution. The technique includes inserting blocks of symbols with zero entropy into the code. As a result, the code section of driver is distributed through the memory. It helps to evade entropy analysis. Fortunately, or unfortunately, this hacker's technique is also vulnerable and can be detected. The thing is that the Zeus Trojan inserts blocks with the same length and the same content. Cyber-security experts proposed an idea of extracting and deleting such data blocks. This became possible by applying byte histograms to analyze the memory content. Look at these red ovals. On the diagram these columns are corresponding to the inserted blocks of symbols, which decreased the entropy. We can easily extract these blocks and calculate the entropy



without them. But, applying this byte histogram is also vulnerable. I'll show you the appropriate anti-forensic technique a bit later.

Slide 25 “Summary”

So, we can say that all analyzed detection methods are vulnerable. Any malware driver can be hidden by many ways but the results will always be the same – running an uncontrolled code in the privileged memory area and this code will have a low entropy value. Now let's integrate all anti-forensic techniques and see how they are improving. The purpose is to imagine the most difficult scenario for detection.

All detection methods are vulnerable

The ways to detect drivers		Anti-forensic technique
Using drivers structures	Using links between structures	Unlinking
	Signature-based scanning	Overwriting
Using content of drivers files	Signature-based scanning	Overwriting & PE packing
	Statistical-based detection	Inserting data blocks

Let's consider the most difficult case for detection - HighStem

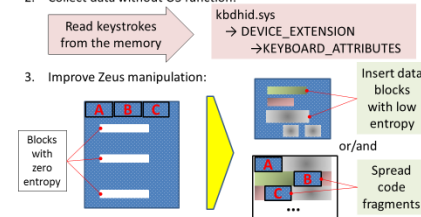
Slide 26 “HighStem”

In a nutshell the Highest Stealth Malware – HighStem is just the kernel-mode driver. The main idea of the HighStem functionality is using as few Windows functions as possible and acquire information straight from the memory. For example, we can get keystrokes just by reading specific memory regions, without using any Windows functions or installing a filter driver. My preliminary research reveals that the codes of keystrokes are included in the structure keyboard attributes, which could be located in this way.

Also, I've got some ideas of how to improve Zeus trojan's manipulations to hide its code from the Δ byte histograms analysis. The first thing is to insert blocks of symbols, with low but not zero entropy value. The

Highest Stealth Malware (HighStem) imitates the most difficult case for detection

1. Apply Atsiv or Turla Driver Loader to load a HighStem driver
2. Collect data without OS function:



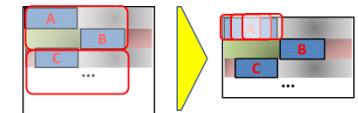
second idea is to insert different size blocks. And also we can significantly increase the size Δ and the number of these data blocks. As a result, it will look like using camouflage for executable code. Next I'll present some ideas how to locate such a super hidden code. I'll mark the memory fragments with this code as a suspicious one, without deciding if this code is a real malware or not.

Slide 27 “How to reveal all parts of diluted executable?”

The task to detect a malware driver comes down to the recognition task between the executable code and other memory fragments. But in our situation this task is hindered by the code dilution. The obvious thing is to apply existing a sliding window approach using smaller window size. This helps us to find separate pieces of executable code but unfortunately it will probably cause a lot of false positives.

How to reveal all parts of diluted executable?

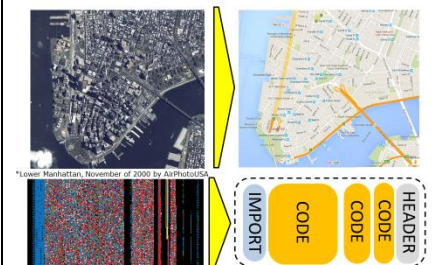
- Calculate entropy using smaller window size



Slide 28 “Photogrammetry”

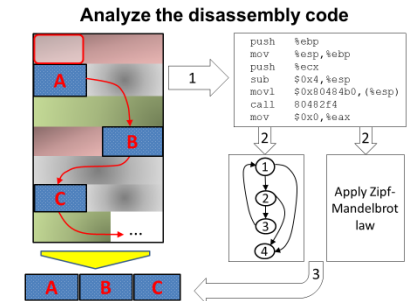
A bit crazy idea is to use best practices from different technical fields, for example, digital photogrammetry. The methods of digital image recognition can distinguish faces in the iPhones, recognize houses and roads in the map using various approaches, such as artificial neural networks. And why not apply this for drivers detection? In our case we need to distinguish anomalies in entropy figures or find executable fragments in colored diagrams.

Apply digital photogrammetry to locate a code



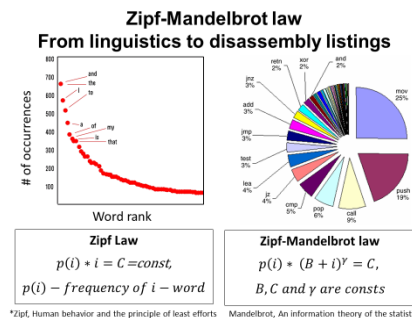
Slide 29 “Analyze the disassembly code”

Apart from this, we can locate the executable code using a disassembler for each code fragment and analyze the output in different ways. For example, we can do it by drawing a control flow graph and analyze the logic for each fragment. The purpose is to find the links between various fragments, and put together all three blocks A-B-C. Another idea is to apply a linguistic law to decide if this assembler code belongs to the executable or it is just data. This Zipf-Mandelbrot Law is not so trivial and I’d like to tell you more about it.



Slide 30 “Zipf-Mandelbrot Law”

The American linguist George Zipf found that the frequency of words is inversely proportional to their rank in frequency lists. For example, if we rank all words in Romeo and Juliet we will see the following hyperbolic curve. The most frequent words such as “the” and “and” are on the top, while rare words are in the right tail. This law works well in all existing languages. This is the law of how languages work. The mathematician Benoit Mandelbrot expanded this law by adding two more constants. And his law works well on texts with high logic. The point is the opcodes of executable codes have similar frequency ranks stability. The most popular opcode is “MOV”, next is “PUSH” “CALL” and their ranks are so close to

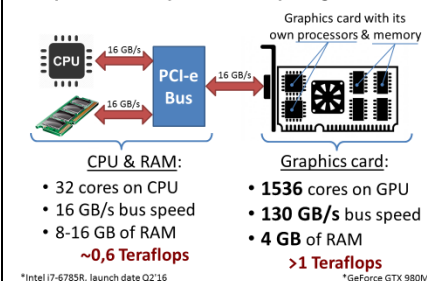


human language. My idea is to check opcodes ranks for each fragment – if these ranks are close to Zipf-Mandelbrot principle.

Slide 31 “Graphics card - a powerful PC in a PC”

All ideas mentioned before require a lot of computational power and we have hardware, which has feasible capabilities. Let’s look at the modern graphics card. GPU has more than one and a half hundred cores, while modern CPU has only thirty cores. GPU bus speed is about 10 times higher than the bus between CPU and RAM memory. Why don’t we use GPU for processing a sliding windows algorithm?

Graphics card - a powerful computing unit in a PC

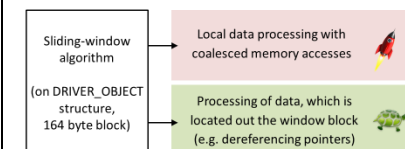


Slide 32 “Straightforward sliding-window detector on GPU”

We tested drivers detector from paper ‘Applying memory forensics to rootkit detection’, which uses a sliding window type algorithm to detect hidden drivers structures. This algorithm includes two distinctive features. The first part corresponds to processing the data locally inside the window block. GPU shows up the best performance on such operations. The second part is processing the memory fragments outside the window block. Unfortunately, GPU has rather low speed in such out of block operations. So, the best performance occurs when memory accesses can be coalesced into consecutive byte chunks.

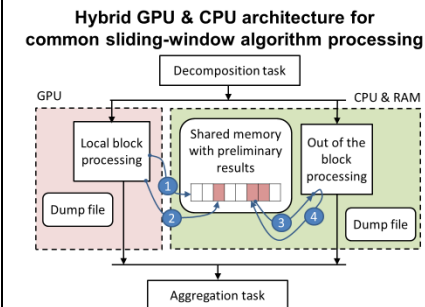
Porting issues of common sliding-window algorithm to GPU

- We tested drivers detector from the paper ‘Applying memory forensics to rootkit detection’ ADFS’2014, Richmond, VA
- GPU works efficiently on 128-byte size coalesced memory
- GPU operates much slower on distinct memory fragments



Slide 33 “Issues of porting of common sliding-window algorithm to GPU”

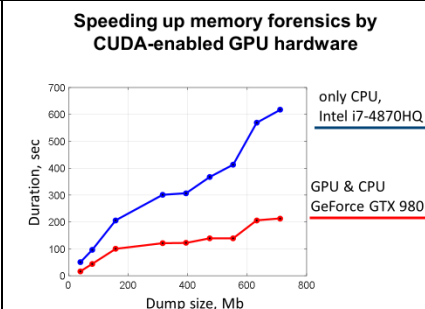
Taking these two features into account, we can split the sliding window algorithm in GPU and CPU parts, which can run concurrently. Graphics video card computes its part of analysis and sets the score of each memory region into the table with preliminary results. CPU uses this table and handles only the memory regions with appropriate metrics. Thus the GPU part performs only local memory block analysis, transferring further processing to CPU. Operating system API functions calls are also processed by CPU, e.g. iswprint.



Slide 34 “Speeding up memory forensics by CUDA-enabled GPU hardware”

Here is the final current speed up achievements. We have the 2-3 times speeding up. Now we are working to process large dumps on GPU.

This demonstrates the benefits of applying GPU for memory forensics and incidents response tasks.



Slide 35 “Conclusion”

In a nutshell in this talk I’ve presented how to detect the most dangerous memory parts or an executable code. I presented the analysis of drivers detection ways and their drawbacks. I focused on HighStem driver, which includes evasion mechanisms and new ideas to overcome entropy analysis. The presented

Conclusions

- Prototype of the most hidden code – a HighStem
- Ideas to locate executable code
- Using CUDA to speed up memory dump analysis

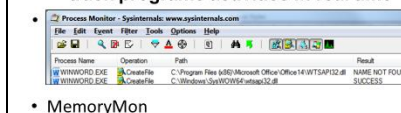


detection ideas will be able to reveal any diluted executable code. Finally, I demonstrated the architecture of a software tool, which uses video cards to speed up memory forensics. This work is now still in progress. And now I'm looking for a team who will consider this as a prospective direction. So, I've got some more ideas how to detect zero-day malware.

Slide 36 "MemoryMon"

One of my dreams is to track memory changes or programs activities in the memory in real time. We have similar tools to monitor file system, registry and network activity, but we don't have such a tool for memory. It will be so helpful for cyber security. My idea is to create a memory monitor or MemoryMon which will be hidden for malware, speedy in operation and resilient to common anti-forensics techniques. To do this I worked out in collaboration with a researcher from Canada, his name is Satoshi, and we developed a thin hypervisor using Intel VTX and EPT technologies.

#1: MemoryMon monitors memory changes to track programs activities in real time

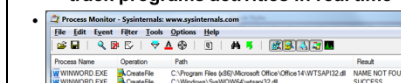


- MemoryMon

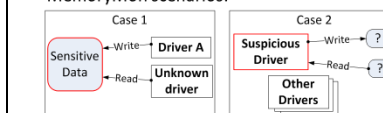
Slide 37 "MemoryMon"

MemoryMon can be used to monitor memory in two cases. The first one is monitoring access to specified memory regions to protect system critical data. We will log and control all access attempts to this memory region. The second case is recording any types of memory access from a specified memory region. We will


#1: MemoryMon monitors memory changes to track programs activities in real time



- MemoryMon scenarios:



- Details: Monitoring & controlling kernel-mode events by HyperPlatform by Satoshi Tanda @standa_t and Igor Korkin, REcon 2016.

<p>monitor memory activities of a suspicious driver. All details about implementation are in Satoshi's talk on the REcon conference.</p>	
<p>Slide 38 “VR headset”</p> <p>During an incident response process cyber security experts analyze huge amount of data, such as different systems logs and memory dumps. To do all these things, why don't we use virtual gears or headsets, which are so popular nowadays. New 3D view, instead of existing 2D, could really speed-up analysis and could make it more vivid. I hope it will become one of the future trends in cyber security.</p>	<p>#2: Apply virtual reality headset for digital forensics investigations</p>  <p>by Samsung</p> <p>by Oculus</p> <p><small>'It's like watching a 130-inch television screen from 10 feet away'</small></p> <p><small>http://www.pocket-lint.com/news/134352-suzik-iwear-720-a-vr-headset-that-s-like-putting-a-130-inch-tv-</small></p>
<p>Slide 39 “Thank you”</p> <p>Thank you!</p>	<p>Thank you!</p> <p>• igor.korkin@gmail.com</p>