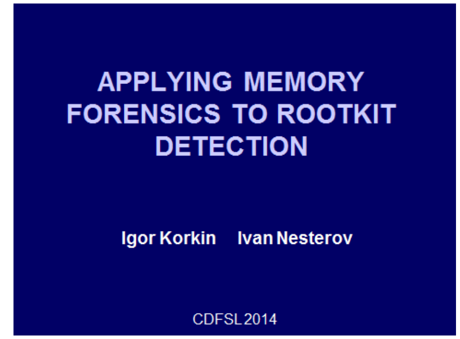
Slide 1 Hello

And the last talk for today. Can you hear me?

Hi again! Thanks for staying to listen to me. {PAUSE}

¿Do you know what your computer <u>memory</u> contains? {PAUSE} Of course operation system, office programs, working documents etc. ¿Are you sure there are no hidden processes or drivers? {PAUSE}

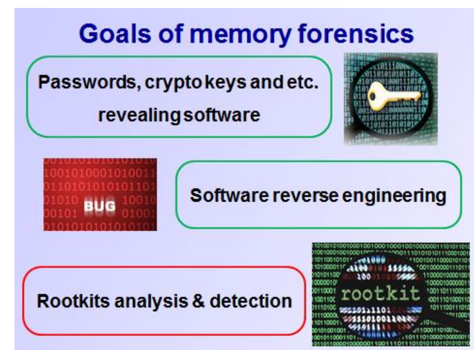
Today I'll tell you how to find hidden objects in virtual memory.



Slide 2 Area of memory forensics

First I'd like to define memory forensics and its goals. Memory forensics is memory analysis which is made to achieve cyber security goals, for example work with sensitive information in memory, reverse engineering of software, hidden programs detection.

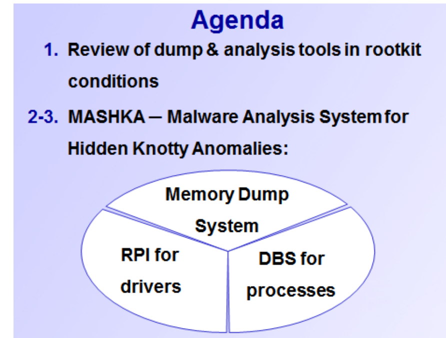
In this talk I'd like to focus on róotkits detection.



Slide 3 Agenda

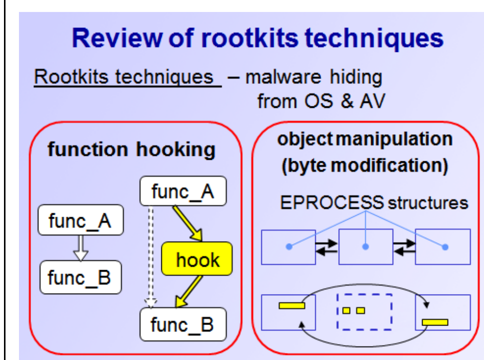
This talk consists of three parts. The first part covers [*cuvez*] existing memory dump and detection approaches [*apro-ochez*]. In the second part, I'll go on to the new memory dump system. And the third part deals with two detection approaches which are resilient to an intruder.

Now I'm going to present current approaches not because I want to criticize them but because I want to avoid their dráwbacks. {PAUSE}



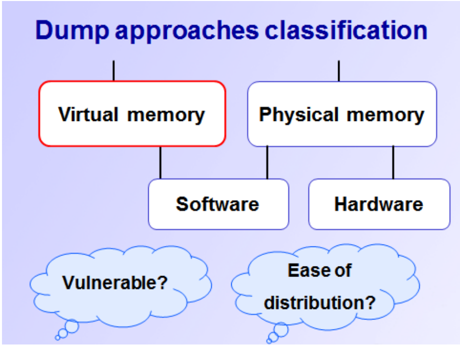
Slide 4 Róotkit Technologies

Modern málware can prevent its dumping and further analysis by using róotkit techníques. {PAUSE} Róotkit techníques are generally classified by two approaches: function hooking and object manipulations. Function hooking causes módification of function results. Changes are highlighted by yellow color. By unlinking structures from lists {PAUSE} process can be hidden. {See yellow rectangles} And in some cases this hidden structure might be additionally módified. {See yellow squares}. ¿Why does it occur? I'll give you answers [*ansez*] in my second part.



Slide 5 Dump approaches tree

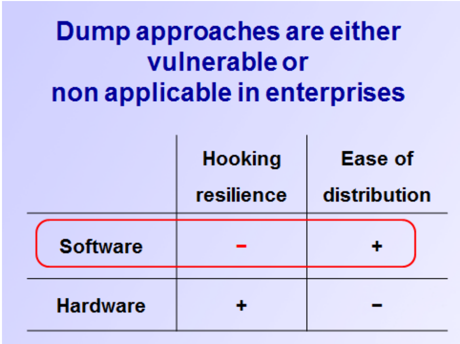
It is possible to dump virtual and physical memory with software and hardware approaches. We want to get a dump approach, which is resilient to hooking and easy to distríbute. ¿Can we do it?



Slide 6 Dump approaches table

Not by current approaches, because software approaches are vulnerable to róotkits techníques. Hardware approaches are not suitable [*sutable*] for use in énterprises. {PAUSE} We cannot improve hardware approaches.

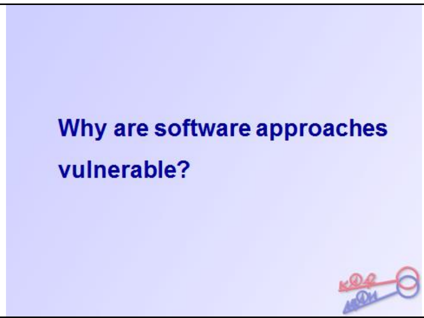
¿Can we improve software ones?



Slide 7 Q

Let's think why are software approaches vulnerable [*vaLnerbl*] ?

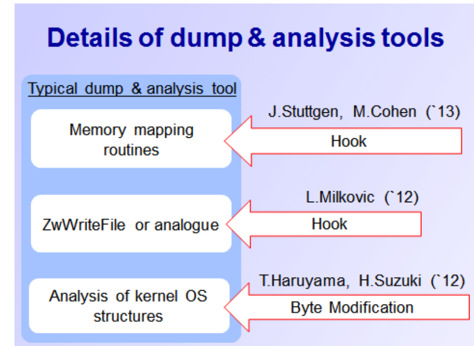
To answer this question let's look at a typical tool for memory dump and analysis.



Slide 8 Details of dump & analysis tools

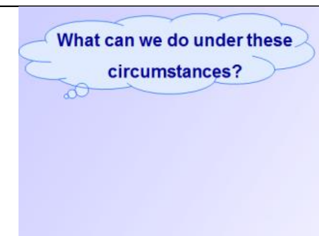
¿What are the main components of such a tool? {PAUSE} This tool usually consists of three components: memory acquisition, its saving and analyzing. {PAUSE} These authors describe methods to disrupt each component. For example Lúka Milkovic's approach is based on hooking acquisition routine and replacing its buffer content. As a result memory pages will be saved without information about málware.

We cannot use operation system functions, because they can be intercepted.



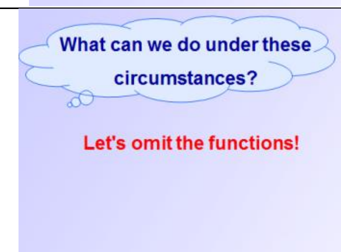
Slide 9 Q

¿What can we do under these circumstances? {FASTER}



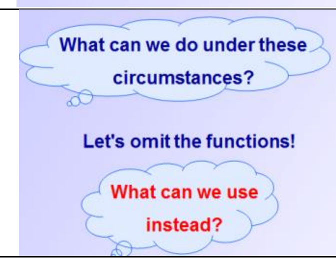
Slide 10 Q

Let's omít the functions! {PAUSE}



Slide 11 Q

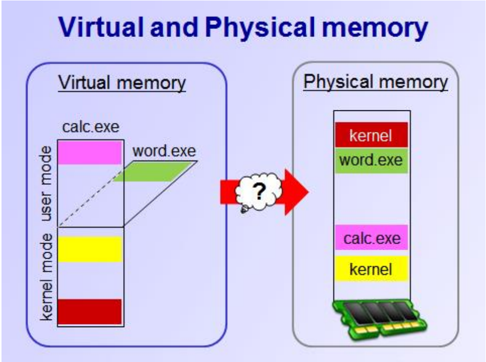
¿What can we use instead?



Slide 12 Virtual and Physical memory

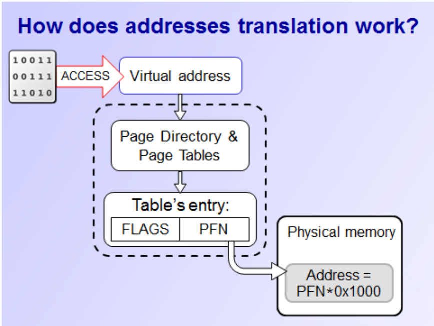
Let's look at memory addréssing in protected mode. In this mode each process [*pro-oses*] uses [*uzez*] a séparate memory context, with user mode and kernel mode. Here we have two processes [*pro-osesez*] Calculator and Word. They contain pages, colored [*colod*] pink and green. Roughly, kernel mode includes two pages yellow and brown. And here they áre in physical memory.

¿How does addréss tránslátion wórk?



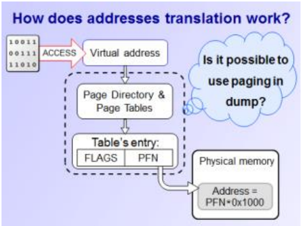
Slide 13 Q

When a program accesses the virtual addréss, the C-P-U is walking through the sýstem tables to find the corresponding page entry. Its P-F-N Page Frame Number, corresponds to the page physical addréss.



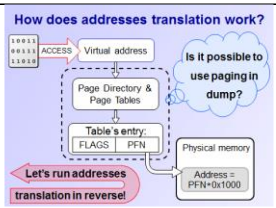
Slide 14 How does

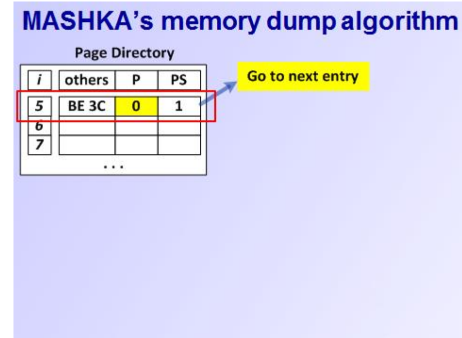
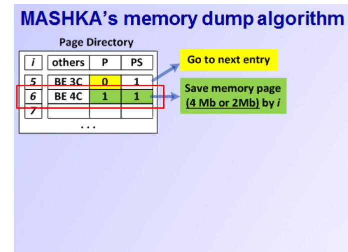
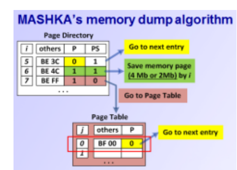
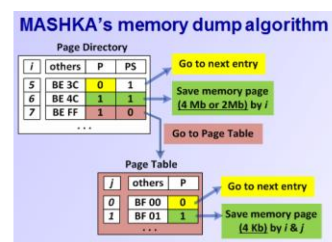
Let's focus on the dashed line rectángle. ¿Is it possible to use this fragment in memory dump? {PAUSE}



Slide 15 How does

Yes it is! Let's run addrésses translation in reverse! {PAUSE}



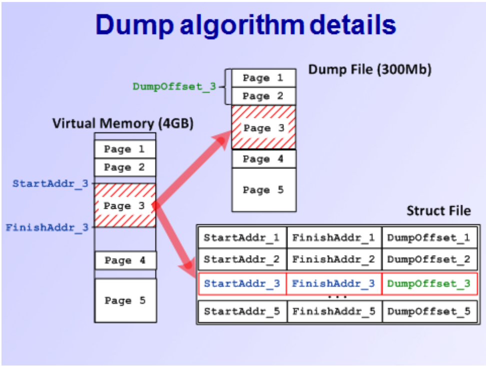
Slide 16 Memory dump algorithm Let me demonstrate [*demonstra-ate*] how to use paging for memory dump. {PAUSE} Walk successively [*succeessively*] through the Page Directory entries and check the P flag of each entry. {PAUSE} If this flag is 0, go to the next entry;	
Slide 17 Memory dump algorithm Otherwise check the Page Size flag. If PS flag is 1, save the corresponding memory page.	
Slide 18 Memory dump algorithm If PS flag is 0 {PAUSE} this entry corresponds to the Page Table. Go to this Table.	
Slide 19 Memory dump algorithm → Slide 20 Memory dump algorithm In a similar way walk successively [*succeessively*] through the Page Table and save memory pages.	 
Slide 21 Memory dump algorithm As a result we acquire complete dump of virtual memory from one process, without memory mapping routines.	

Slide 22 Dump algorithm details

Here is what we get after applying memory dump algorithm.

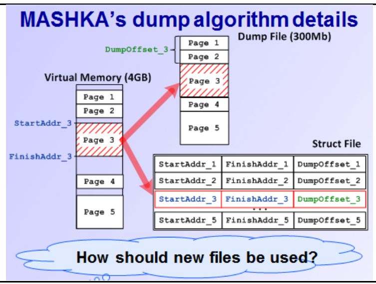
We save virtual memory context in two files. The first file contains only memory pages without gaps. The second file contains the connection between the page addrésses in the virtual memory and its óffset in the dump file. {PAUSE}

For example, we copy page number three from the memory to dump file and save its óffset, start and finish addresses to the struct file.



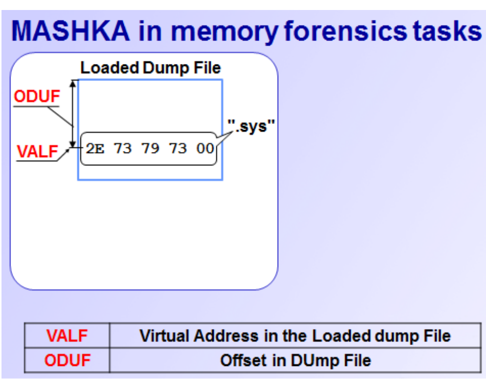
Slide 23 Q

¿Why are there two files: dump and struct? ¿How should they be used?



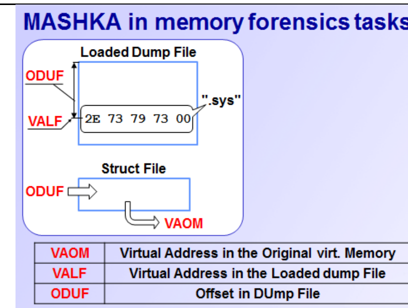
Slide 24 MASHKA in memory forensics tasks

By using MASHKA we can search for binary fragments, strings and do other typical forensics tasks. To understand how it works, let's find the string "dot sys". Before analyzing we load the dump file completely. After searching we receive its dump's óffset O-D-U-F and addréss in this memory V-A-L-F {SLIDE}.



Slide 25 MASHKA in memory forensics tasks

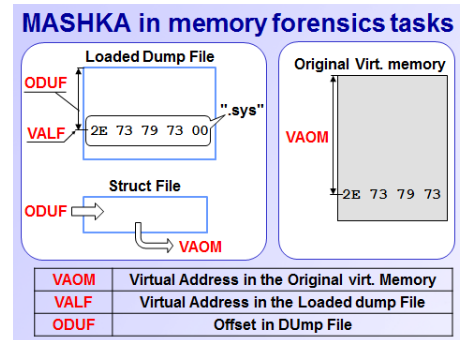
By using struct file we get its original addréss – V-A-O-M.



Slide 26 MASHKA in memory forensics tasks

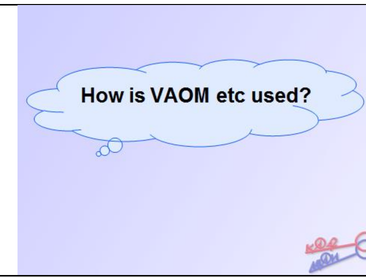
By using VAOM it is possible to find objects, which refer to this string.

For example this allows us to reverse structures with the help of the fragments we know. In a similar way we can search for various objects.



Slide 27 Q

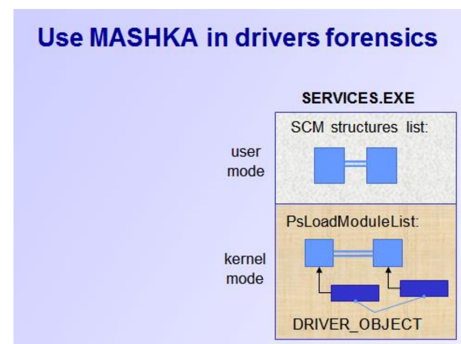
Let's see how to use this system in drivers forensics.

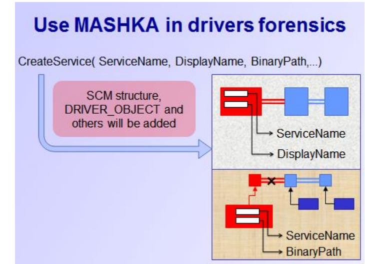
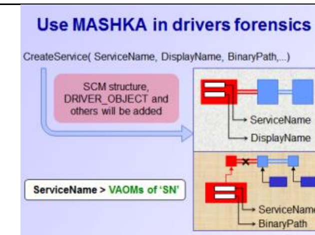
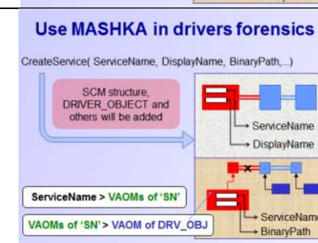
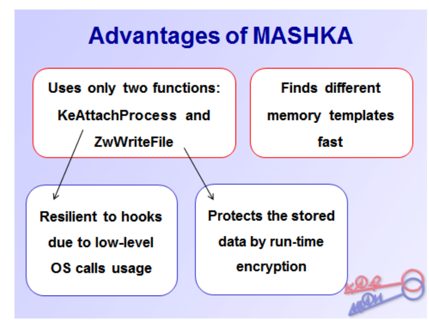


Slide 28 Use MASHKA in drivers forensics

We are going to find information in memory about driver, as if it was hidden. This is just a demo.

Before starting a driver let's look at two lists: list of services in services dot exe and list of loaded drivers in kernel memory, or PsLóadModuleList.

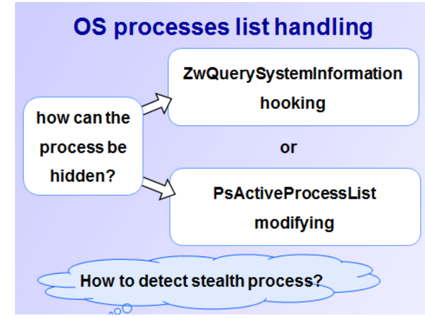


Slide 29 Use MASHKA in drivers forensics After we have loaded a driver {SLIDE} new structures are added to memory. They contain links to strings.	
Slide 30 Use MASHKA in drivers forensics By searching known 'ServiceName' we can find its VAOM.	
Slide 31 Use MASHKA in drivers forensics After that we can use VAOM value to find all the required structures and lists.	
Slide 32 Advantages of MASHKA This approach it's fast and resilient [*resileeyent*] to typical attacks like hooking. It gives various opportunities for solving memory forensics tasks. Let's look at how to use MASHKA to solve two of them.	
Slide 33 Q ¿How to apply MASHKA to processes detection?	

Slide 34 Q

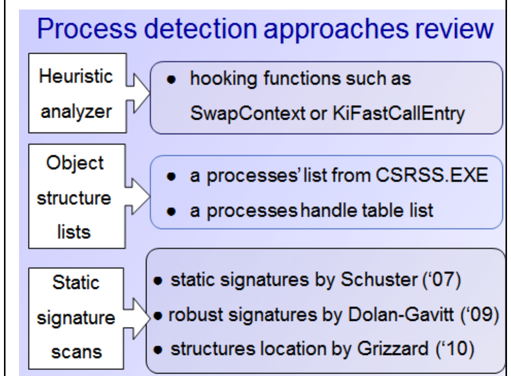
I've always had some reservations about windows task manager for good reason.

Process can be hidden with the help of function hooking or process list módificátion. ¿How to detect a hidden process?



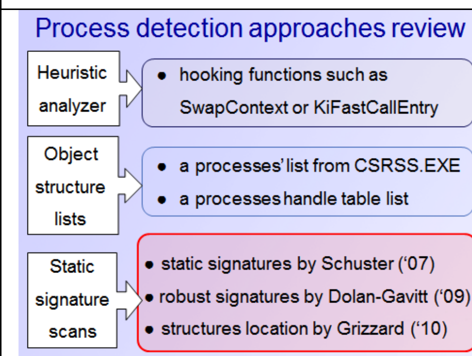
Slide 35 Process detection approaches review

Let's analyze popular cross-view detection approaches. The heurístic analyzer has to collect enough information about activity of a program, which isn't reliable. For example a hidden program can send data to server once a week. We have to wait of a week. The second method uses information from additional objects lists. This method is vulnerable to unlinking a target structure from all these lists. The third method uses signatures of processes structures to search them in memory dump.



Slide 36 Process detection approaches review with red square

Let's analyze static signature scans

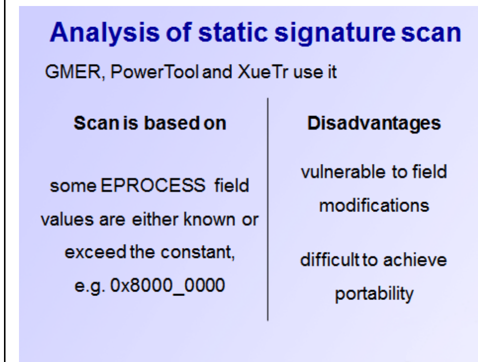


Slide 37 Analysis of static signature scan

Static scan is implemented in well-known anti-róotkits.

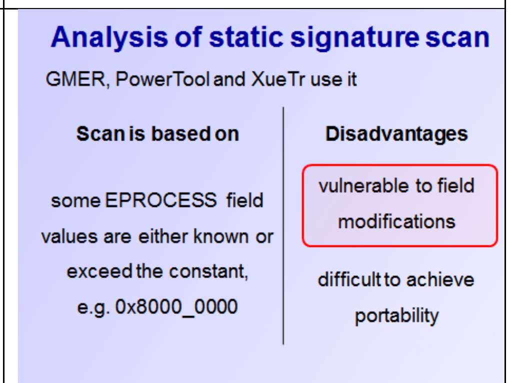
It is based on the fact that values [*valyooz*] of some fields are known. That’s why it is possible to check their values in byte to byte search. We decide whether or not a structure is in memory {PAUSE} if all checks are true at the same time.

This method is vulnerable and difficult to port.



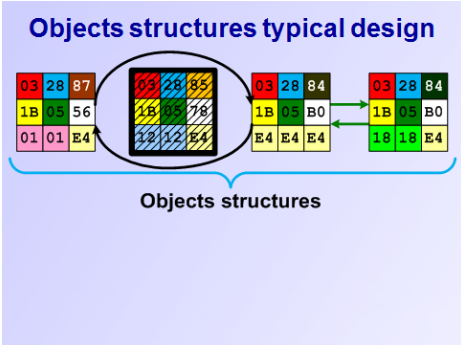
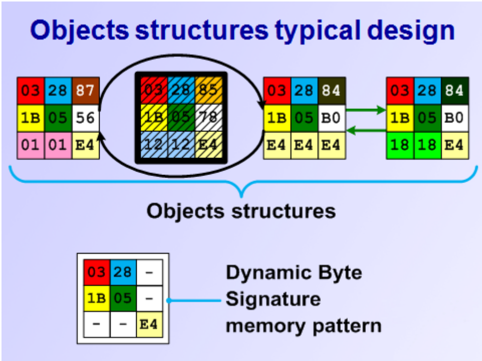
Slide 38 Analysis of static signature scan with red square

We can improve this disadvantage.



Slide 39 Q

¿How can we improve signature scans?

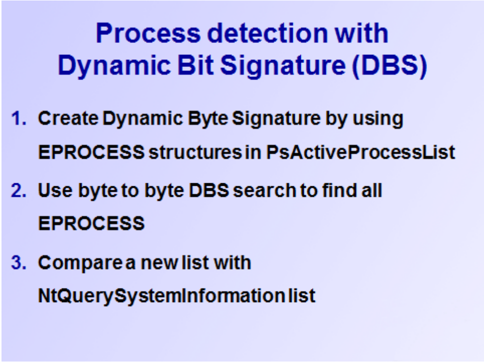
Slide 40 Objects structures typical design To answer this question let's find some common peculiarities [*pecooliaritees*] between EPROCESS structures of different processes. On this slide {PAUSE} EPROCESS structures list is shown. Different bytes are illustrated on the figure as squares [*S!*] with different colors. The corresponding squares have identical colors if the byte values [*valyooz*] are the same. We see that initial bytes of each structure are identical, but further bytes are different.	
Slide 41 Objects structures typical design Dynamic byte signature includes only those values, which are the same in all structures in this list. Look at the bottom of the slide. By using this signature it is possible to find all EPROCESS structures regardless of whether they are hidden or not. It is shown below how to do this.	

Slide 42 Dynamic Bit Signature (DBS)

First of all we create dynamic byte signature which includes the same bytes from all structures from the process list.

Due to the probabilistic nature of lookups we find all the EPROCESS structures even if they were hidden or deliberately módified.

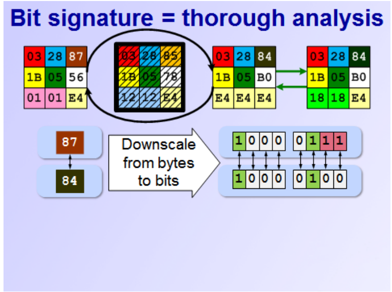
Finally we conclude about hidden processes by comparing our list with NtQuerySystemInformation list.



Slide 43 Bit signature – thorough analysis

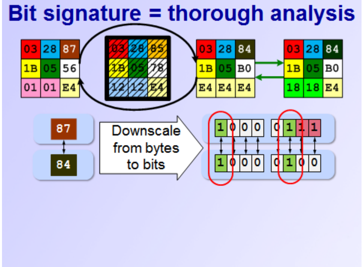
Let's have a close look at the bytes, which don't match. For example 87 and 84.

The bit mapping of these two bytes is in the centre of the slide.



Slide 44 Bit signature – thorough analysis

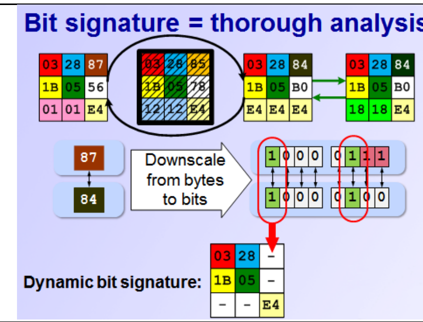
We see that some bits are the same.



Slide 45 Bit signature – thorough analysis

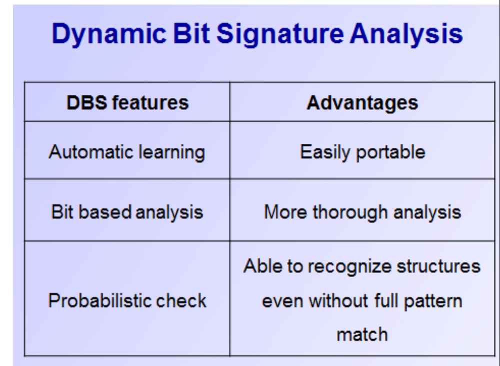
And we add these similarities to the signature.

By replacing bytes-based with bits-based analysis we make it more thorough.



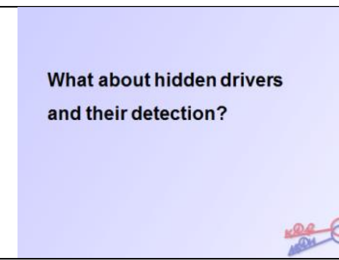
Slide 46 Analysis of DBS

Let's look at DBS. It doesn't have to be saved; DBS is automatically generated from EPROCESS structures list every time before searching. Bit analysis has just been described. DBS can recognize structures even without full pattern match. Only seventy to eighty per cent pattern matching is enough.



Slide 47 Q

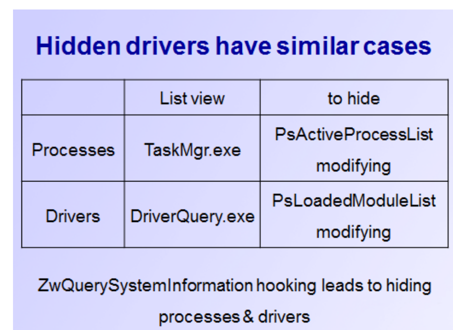
¿What about hidden drivers and their detection?



Slide 48 Hidden drivers specifics

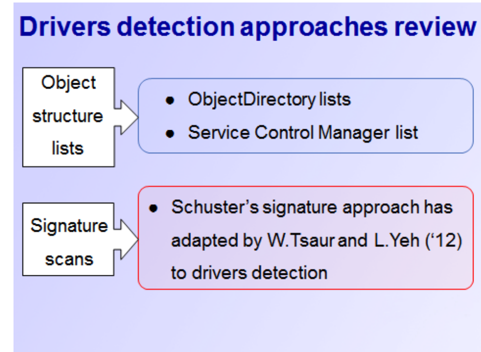
Hiding drivers and processes have a lot in common. DriverQuery.exe like TaskMgr.exe is build-in tool. DriverQuery.exe gives information about drivers.

To hide a driver we can use PsLoadedModuleList módification or hook this function.



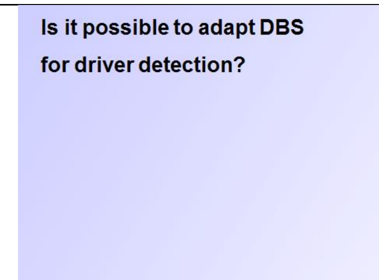
Slide 49 Drivers detection approaches review

I'm going to tell you about {PAUSE} drivers' detection approaches: {PAUSE} object structure lists and signature scans. They have the same disadvantages as those for process detection. The first one is its vulnerability to unlinking a target structure from all lists. The second one is its inability to detect módifying structures.



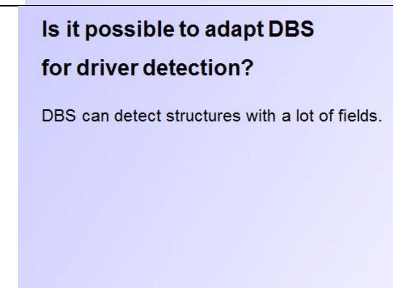
Slide 50 Q

¿Is it possible to adapt DBS for driver detection?



Slide 51 Q

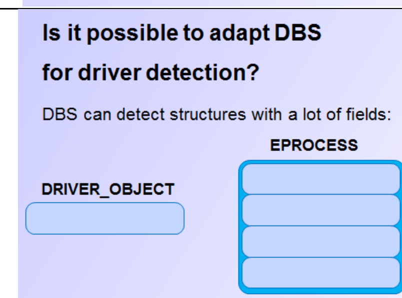
We know that DBS is good for detecting structures with a lot of fields in their definitions, because we need to have a lot of data to generate a bit signature.



Slide 52 Q

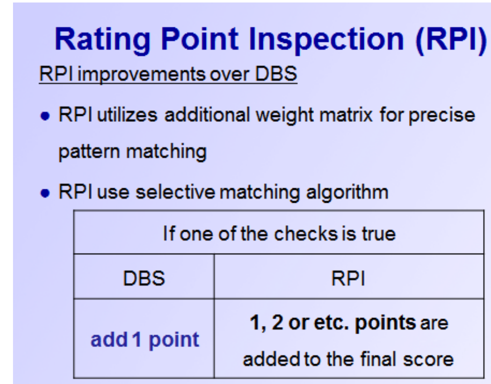
We see that DRIVER_OBJECT structure is 4 times smaller than EPROCESS.

That is why, it is impossible to apply DBS to drivers detection.



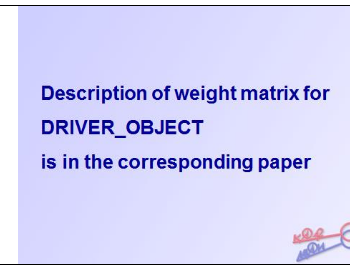
Slide 53 Rating Point Inspection (RPI)

To detect a driver I have developed a new approach RPI, which is partially based on DBS. The first difference is the utilization of additional weight matrix for precise [*pre-esise*] matching accounting. We calculate total matching points (score) but not the individual [*indivijual*] matches themselves. In the DBS case we simply summarize the numbers of matches or add 1 point to the final sum, if the check is true. In RPI if one of the checks is true, 1, 2 etc. points are added to the final score. Number of points is chosen according to the weight matrix.



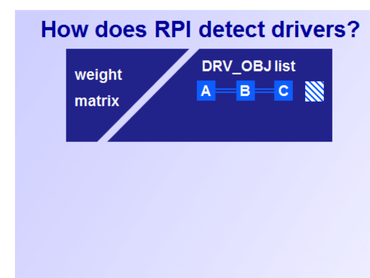
Slide 54 Weight matrix

Weight matrix is given in the corresponding paper, because it's large.



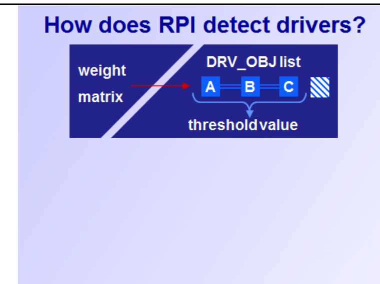
Slide 55 How does RPI detect drivers?

Let's see how RPI detects drivers. For that we use [*uze*] weight matrix and a list of DRIVER_OBJECT structures. Here we see three structures in the list and a hidden one.



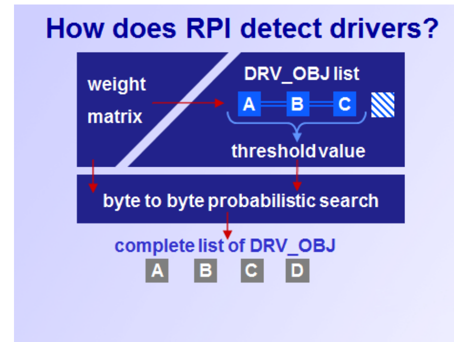
Slide 56 How does RPI detect drivers?

First we count the weight for each structure in the list. We achieve the threshold value by using all these weights, except hidden one.



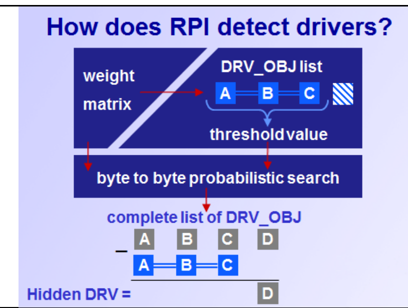
Slide 57 How does RPI detect drivers?

Second we use [uze] byte to byte search and weight matrix. We cálculate the weight for each memory fragment. {PAUSE} If the fragments weight is close to the threshold, we conclude that the driver structure is found. As a result we achieve a complete list of DRIVER_OBJECTs.



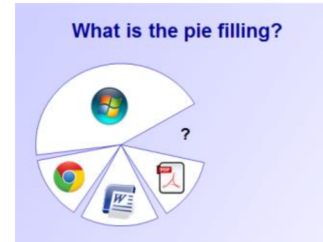
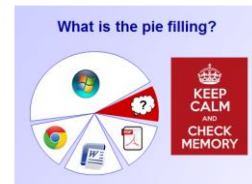
Slide 58 How does RPI detect drivers?

We detect hidden drivers by comparing the two lists.



Slide 59 MASHKA's achievements

We successfully tested MASHKA in different cases: intentionally hidden objects, málware in the wild and drivers loaded by ATSIV. All drivers, which are loaded by ATSIV are hidden.

Slide 60 MASHKA's achievements red rectangle Popular existing anti-róotkits are unable to detect the latter, but MASHKA can.	
Slide 61 Q ¿Do you know what's in your computer memory apart from windows modules, chrome, word or acrobat?	
Slide 62 Q Now you know what to do. {PAUSE} {PAUSE} {PAUSE} Thank you.	

Slide 63 Igor Korkin, Ph.D. I will answer your questions with pleasure, if I cannot, please write me an e-mail and I promise will answer all questions.