

Kernel Hijacking Is Not an Option: MemoryRanger Comes to the Rescue Again

Slide 1 Hello

😊 Hi everyone! 😊

Thank you, Mr. Cooper, for introducing me.

I am happy to be a part of this event and I really appreciate your interest in my /w3:(r)k/ work!

Today I'd like to present you ...to show you ... three hypothetical, but possible attacks on kernel data in the newest Windows 10 and a way to win this kernel-mode battle.

I think this topic is of critical importance for all security experts who are dealing with /əʊ 'es/ OS security.



Slide 2 “WhoamI”

A few words about mé. I've been exploring /əʊ 'es/ OS security for more than 10 years.

I'm always curious about attacks on the OS kernel and how to prevent them.

You can find my results in my blog.

WHOAMI

- PhD, speaker at CDFSL since 2014 and BlackHat
- Windows OS Kernel Security Researcher:
- Rootkits and anti-rootkits
- Bare-Metal Hypervisors vs. Attacks on Kernel Memory
- Fan of cross-disciplinary research - igorkorokin.blogspot.com
- Love traveling and powerlifting - @igor.korokin

Slide 3 “Agenda and general questions”

Today I am going to talk about three new attacks on kernel data.

Two of them show a way how to read and overwrite files opened without shared access.

AGENDA

- Three attacks on kernel memory data:



Slide 4 “Agenda and general questions”

And another one shows how to elevate process privileges.

All these attacks are based on patching memory data and they do not trigger any Windows security mechanisms. I have discovered a technique that can ... prevent this kind of attacks.

AGENDA

- Three attacks on kernel memory data:

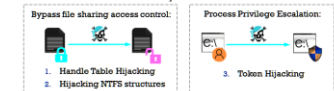


Slide 5 “Agenda and general questions”

I'll show you my MemoryRanger, the {slow} tool {slow} which is designed to prevent attacks on kernel memory. One of the key features of updated MemoryRanger is a new /dateonly/ data-only enclave.


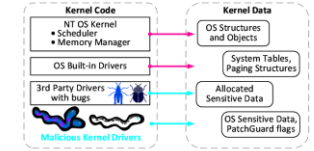






AGENDA

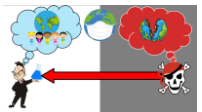


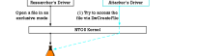


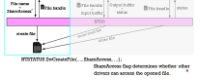
- Three attacks on kernel memory data:

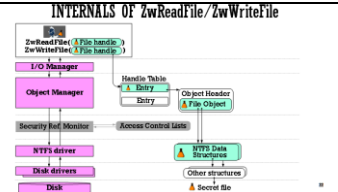
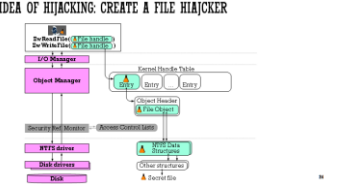
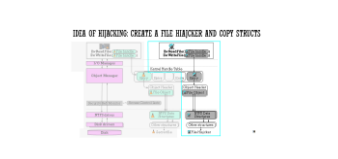







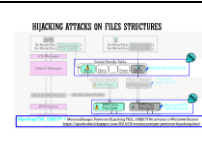


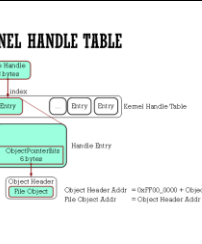
- MemoryRanger blocks kernel attacks:
- It runs drivers in isolated kernel enclaves
- It includes a new feature: Data-Only Enclave

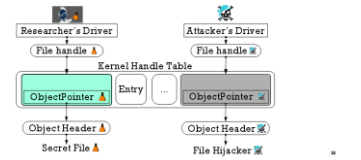
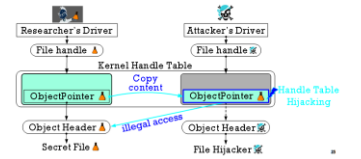
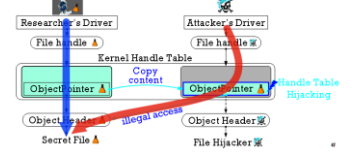
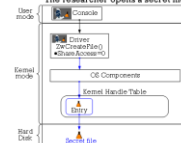
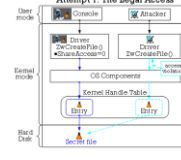
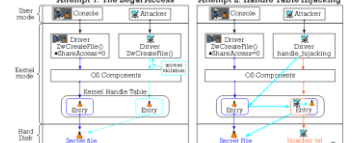


<p>Slide 6 “Kernel-mode Threats: Problem and Consequences”</p> <p>I’ve been developing my MemoryRanger for several years, here are the corresponding research projects. You can follow the links below to read more about the MemoryRanger.</p>	<p>PREVIOUS RESEARCH ON MEMORYRANGER: PAPERS + SLIDES + DEMOS</p> 
<p>Slide 7 “Kernel-mode Threats: Problem and Consequences”</p> <p>Nowadays kernel-mode OS components and malware drivers share the <u>same</u> memory space with the rest of the OS kernel. All drivers can read and overwrite any part of kernel-mode memory without <u>any</u> hardware restrictions. Therefore, modern operating systems are /prə-ʊn/ prone to kernel attacks.</p>	<p>KERNEL DRIVERS CAN COMPROMISE THE OS SECURITY</p> 
<p>Slide 8 “Kernel-mode Threats: Problem and Consequences”</p> <p>Look at the recent examples of kernel malware attacks. <u>/stress/ RobbinHood Ransomware</u> that exploits a legal vulnerable driver to load a malware driver and disables security products.</p>	<p>KERNEL DRIVERS IN RECENT MALWARE ATTACKS ON WINDOWS</p> 
<p>Slide 9 “Kernel-mode Threats: Problem and Consequences”</p> <p><u>/stress/ Crypto-miner</u> that has infected over 50,000 servers. It installs a malware driver to protect itself and abuses the computing /rɪˈzɔː(r)z/ resources for quite a long time.</p>	<p>KERNEL DRIVERS IN RECENT MALWARE ATTACKS ON WINDOWS</p> 
<p>Slide 10 “Kernel-mode Threats: Problem and Consequences”</p> <p><u>/stress/ Glupteba</u> malware that uses a set of kernel drivers: to disable Windows security and finally hides its footprints. Our world is in danger.</p>	<p>KERNEL DRIVERS IN RECENT MALWARE ATTACKS ON WINDOWS</p> 
<p>Slide 11 “Scenario of attacks on files”</p> <p>Who can save the world?</p>	
<p>Slide 12 “Scenario of attacks on files”</p> <p>Let’s imagine <u>{1 sec}</u> that one researcher has invented <u>{1 sec}</u> a /vɪˈvɪfɪk/ vivific elixir, <u>{1 sec}</u> an effective solution that</p>	
<p>Slide 13 “Scenario of attacks on files”</p> <p>That can save the world.</p>	

<p>Slide 14 “Scenario of attacks on files”</p> <p>But here is the hacker-attacker, who wants to destroy the world. He is plotting to steal and modify the formula. It is very likely, that he can succeed. That is why I have developed my MemoryRanger ...</p>	
<p>Slide 15 “Scenario of attacks on files”</p> <p>... to prevent attacks on kernel data. My MemoryRanger guarantees that the secret data remains safe.</p>	
<p>Slide 16 “Scenario of attacks on files”</p> <p>Let’s see how it works.</p>	<p>Episode 1</p> <p>Hijacking File Structures: Bypass File Sharing Access Control</p>
<p>Slide 17 “Scenario of attacks on files”</p> <p>It is obvious that the formula should be /kæpt/ kept in secret. The researcher opens a file without shared access. No one can read and overwrite it.</p>	<p>SCENARIO OF ATTACKS ON FILES</p> 
<p>Slide 18 “Scenario of attacks on files”</p> <p>And here is a “<u>hacker-attacker</u>”, that wants to steal the formula and overwrite it.</p>	<p>SCENARIO OF ATTACKS ON FILES</p> 
<p>Slide 19 “Scenario of attacks on files”</p> <p>Usually, Windows /,gærən'ti:/ guarantees an exclusive access to the open file and promises to block all other access attempts. But it is not quite so, simple.</p>	<p>SCENARIO OF ATTACKS ON FILES</p> 
<p>Slide 20 “Scenario of attacks on files”</p> <p>The attacker can bypass this Windows barrier and gain access to the secret data. Today I’ll show you two possible attacks on this kind of files. Let’s have a look at the details of file system routines in Windows kernel.</p>	<p>SCENARIO OF ATTACKS ON FILES</p> 
<p>Slide 21 “Inside FileSystem”</p> <p>Drivers use ZwCreateFile routine to create or open a file. Setting ShareAccess to zero gives an exclusive access to the open file.</p>	<p>FILE SYSTEM KERNEL API ROUTINES</p> 

<p>Slide 31 “FileSystem Internals”</p> <p>Functions ZwWriteFile and ZwReadFile use the same Windows subsystems, but they are skipping Security Reference Monitor.</p> <p>OS uses file handle as a key to traverse through the following structures:</p>	
<p>Slide 32 “FileSystem Internals”</p> <ul style="list-style-type: none"> Kernel Handle Table; 	
<p>Slide 33 “FileSystem Internals”</p> <ul style="list-style-type: none"> Object Header with File Object; 	
<p>Slide 34 “FileSystem Internals”</p> <ul style="list-style-type: none"> NTFS Data Structures. <p>For each read and write operations OS <u>gets through</u> all these structures automatically, without any security checks, the Security Reference Monitor is not involved.</p>	
<p>Slide 35 “Basic Idea of Hijacking”</p> <p>We have two outcomes:</p> <ul style="list-style-type: none"> ZwCreateFile always performs access checking. ZwReadFile and ZwWriteFile walkthrough the file structures without any security checks. <p>This fact can be used by attackers in the following way.</p>	<p>SUMMARY</p> <ul style="list-style-type: none"> • Only for ZwCreateFile we have checks regarding shared access permissions • ZwWriteFile and ZwReadFile do not bother about access permissions
<p>Slide 36 “Basic Idea of Hijacking”</p> <p>So, the secret file has been created in an exclusive mode and it is remaining open at the moment.</p>	<p>IDEA OF HIJACKING: CREATE A FILE HIJACKER</p> 
<p>Slide 37 “Basic Idea of Hijacking”</p> <p>The attacker can create a file, I named it a file hijacker. OS creates all the corresponding file structures automatically. Then, the attacker can modify the file structures / <u>inorderto</u> / in order to redirect the control / <u>flow</u> / flow to the secret file.</p>	<p>IDEA OF HIJACKING: CREATE A FILE HIJACKER AND COPY STRUCTS</p> 

<p>Slide 38 “Basic Idea of Hijacking”</p> <p>As a result, all read and write access attempts to the file hijacker will be traversed to the secret file.</p>	
<p>Slide 39 “Three hijacking attacks”</p> <p>So, the hacker can use three different hijacking techniques. /+2 seconds/</p>	
<p>Slide 40 “Three hijacking attacks”</p> <p>Handle Hijacking /+2 seconds/</p>	
<p>Slide 41 “Three hijacking attacks”</p> <p>FileObject Hijacking /+2 seconds/</p>	
<p>Slide 42 “Three hijacking attacks”</p> <p>Hijacking of NTFS data structures /+2 seconds/.</p>	
<p>Slide 43 “Three hijacking attacks”</p> <p>Today I'll present you the details of two attacks: Handle Hijacking and Hijacking of NTFS data structures. The link with the hijacking attack on FILE_OBJECT was shown /'z:(r)liə(r)/ earlier.</p>	
<p>Slide 44 “Handle Table Hijacking”</p> <p>Now let's move on to the handle hijacking.</p>	
<p>Slide 45 “Handle Table Hijacking”</p> <p>To understand the idea of handle hijacking let me show some details of the kernel handle table.</p>	
<p>Slide 46 “Handle Table and its entries”</p> <p>This table is used to store the mapping from the handles to the corresponding object structures. Windows Kernel allocates a new entry for each created file. The entry includes <u>ObjectPointerBits</u>. This field points to the object header structure. OS does not <u>check</u> the integrity of this field, and this fact causes the possibility of handle hijacking.</p>	

<p>Slide 47 “Hijacking Handle Table”</p> <p>Here we have a secret entry for the secret file and an entry for the file hijacker.</p>	<p>HANDLE TABLE HIJACKING</p> 
<p>Slide 48 “Hijacking Handle Table”</p> <p>The attacker can overwrite its ObjectPointerBits {slow} using a value of ObjectPointerBits from the secret entry {slow}.</p>	<p>HANDLE TABLE HIJACKING</p> 
<p>Slide 49 “Hijacking Handle Table”</p> <p>Now <u>all</u> read and write <u>access attempts</u> from the attacker will be redirected to the secret file. The legal access will also be forwarded to the secret file. Let's see how it can happen.</p>	<p>HANDLE TABLE HIJACKING</p> 
<p>Slide 50 “Handle Hijacking Demo Scheme Overview”</p> <p>A researcher will open a secret file without shared access.</p>	<p>DEMO: HANDLE TABLE HIJACKING</p> <p>The researcher opens a secret file</p> 
<p>Slide 51 “Handle Hijacking Demo Scheme Overview”</p> <p>The attacker will try to access the opened file twice. First, by calling ZwCreateFile routine</p>	<p>DEMO: HANDLE TABLE HIJACKING</p> <p>Attempt 1: The Legal Access</p> 
<p>Slide 52 “Handle Hijacking Demo Scheme Overview”</p> <p>and second by using the proposed handle hijacking attack. Let's see how it can happen.</p>	<p>DEMO: HANDLE TABLE HIJACKING</p> <p>Attempt 1: The Legal Access Attempt 2: Handle Table Hijacking</p> 

Slide 53 “Handle Hijacking Demo: The Attack”

Let’s check the Windows version. We have the final one.

The researcher is launching his app, which loads a driver.

The researcher is opening a new file.

He is writing down the invented formula.

He is reading the data to check it.

The formula is saved in the secret file but the file is still open.

While the researcher is being proud of his job, for example, having a tee break, the attacker has a chance to steal the formula.

The attacker is launching his app, which loads a driver.

First, he tries to open the file using Windows function. And he fails.

Windows blocks illegal access to the file.

He is starting the hijacking attack and he needs to have a file handle for the secret file.

/ without stress / Let us assume that he’s **got** this handle value.

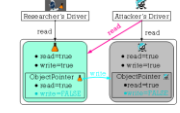
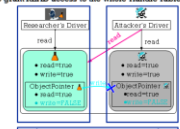
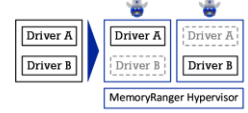

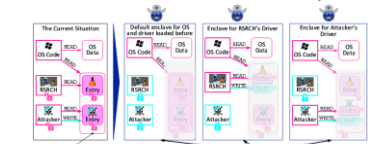
We can see that the attacker creates a file hijacker and patches the corresponding Handle Table.

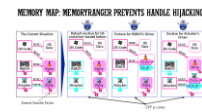

He is trying to read the secret file. And he succeeds. He has read the secret formula.




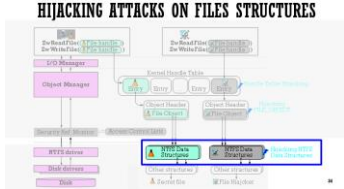
And look, he is trying to modify the secret file.

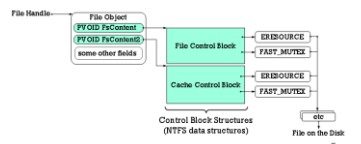


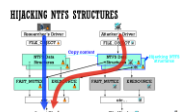

And **When** the researcher decides to check the formula, he sees that the formula has been illegally modified.

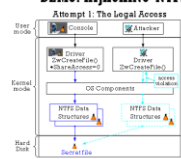
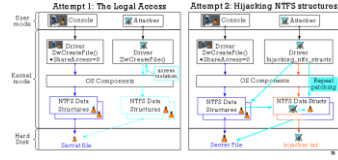
Let’s wait for PatchGuard reaction, which is designed to prevent any illegal memory modifications.



<p>We've been waiting for 10 hours it is quite a long time and ... Look! Nothing has happened. The OS has not been crushed. The OS became infected and the secret elixir is in danger.</p>	
<p>Slide 54 “Handle Hijacking Analysis”</p> <p>Let's think about how to prevent this attack. We have to restrict WRITE access to the ObjectPointerBits, without blocking WRITE access to the other fields. Also, we cannot block READ access to this memory area.</p>	<p>HOW TO PREVENT THE HANDLE HIJACKING?</p> <ul style="list-style-type: none"> • We have to block WRITE access to the ObjectPointer-part • We have to grant READ access to the whole Handle Table for all drivers 
<p>Slide 55 “Handle Hijacking Analysis”</p> <p>My MemoryRanger has been updated to take into account this attack.</p>	<p>HOW TO PREVENT THE HANDLE HIJACKING?</p> <ul style="list-style-type: none"> • We have to block WRITE access to the ObjectPointer-part • We have to grant READ access to the whole Handle Table for all drivers 
<p>Slide 56 “MemoryRanger”</p> <p>Actually, it is a hypervisor-based project designed to block kernel-mode attacks. The MemoryRanger can <i>/træp/</i> trap the loading of new drivers and move them to the isolated kernel <i>/'ɛnklɛɪvz/</i> énclaves. My MemoryRanger can allocate isolated énclaves in run-time with different memory access restrictions.</p>	<p>MEMORYRANGER RUNS DRIVERS IN ISOLATED KERNEL SPACES</p> 
<p>Slide 57 “MemoryRanger blocks Handle Hijacking Analysis”</p> <p>The MemoryRanger allocates <u>the default énclave</u> for the OS and drivers loaded before.</p>	<p>MEMORY MAP: MEMORYRANGER BLOCKS HANDLE HIJACKING</p> 
<p>Slide 58 “MemoryRanger blocks Handle Hijacking Analysis”</p> <p>To prevent Handle Hijacking my MemoryRanger moves two newly loaded drivers into separate énclaves.</p>	<p>MEMORY MAP: MEMORYRANGER BLOCKS HANDLE HIJACKING</p> 

<p>Slide 59 “MemoryRanger blocks Handle Hijacking Analysis”</p> <p>It hooks ZwCreateFile routine to locate the created file structures and restricts any illegal access to them. Red lines indicate the illegal access attempts.</p>	
<p>Slide 60 “MemoryRanger blocks Handle Hijacking Analysis”</p> <p>This <u>scheme</u> helps to prevent Handle Hijacking Attack by blocking illegal WRITE access attempts. Let’s see how it can happen.</p>	
<p>Slide 61 “Demo: MemoryRanger prevents Hijacking Handle Table”</p> <p>Let’s check the Windows version. We’ve got the final one.</p> <p>The MemoryRanger <u>hypervisor</u> is loaded first.</p> <p>The researcher is launching his app, which loads a driver.</p> <p>The researcher is opening the file.</p> <p>He is writing down his genius ... secret formula.</p> <p>He is reading the file to check its content.</p> <p>The formula is saved<u>u</u> in the secret file, but the file is still open.</p> <p>While the researcher is being proud of his job, and for example, having a tee break, the attacker has a chance to <u>steal</u> the formula.</p> <p>An attacker is launching his app, which loads a driver.</p> <p>First, he tries to open the secret file using Windows API function. And he fails.</p> <p>Windows kernel prevents an illegal access to the open files.</p> <p>He is starting the hijacking attack and he needs to have a file handle for the secret file.</p> <p>Let us assume that he’s got this handle value.</p>	

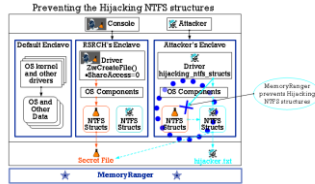


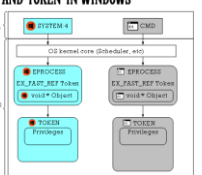
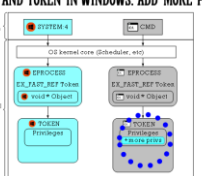
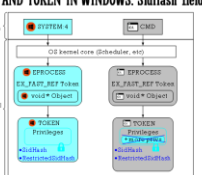
<p>We can see that now the attacker <u>is failing</u> to patch the handle table.</p> <p>He is trying to the secret file and he can read nothing.</p> <p>He is trying to overwrite the secret file.</p> <p>Then. The researcher has finished his tea and wants to check the formula.</p> <p>And the formula has not been changed. MemoryRanger prevents all illegal access attempts.</p> <p>Now let's close both files to compare their content.</p> <p>We can see that the secret file contains only the secret formula and all attacker's input data is saved inside its file hijacker.</p> <p>Thanks to MemoryRanger the OS is protected.</p>	
<p>Slide 62 “MemoryRanger blocks Handle Hijacking Analysis”</p> <p>As a result, we have the following picture.</p>	
<p>Slide 63 “MemoryRanger blocks Handle Hijacking Analysis”</p> <p>MemoryRanger restricts WRITE access to the handle entry and prevents Handle Hijacking.</p>	
<p>Slide 64 “NTFS data hijacking”</p> <p>Now let me show you one more way, an absolutely new technique to access exclusively opened files.</p>	
<p>Slide 65 “NTFS data hijacking”</p> <p>It is the NTFS data hijacking.</p>	

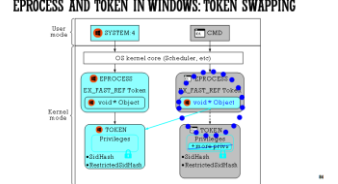
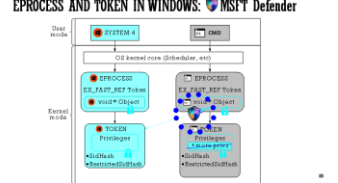
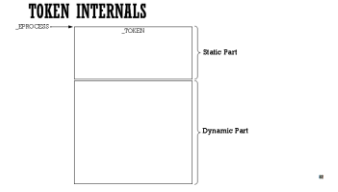
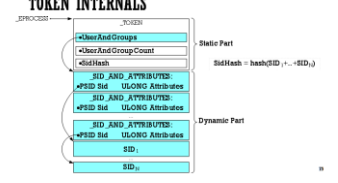
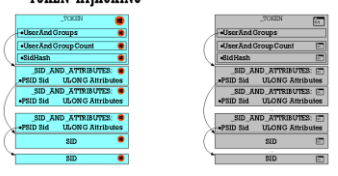
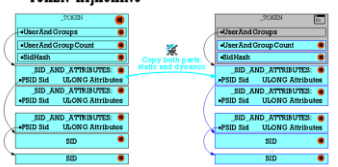
<p>Slide 66 “Overview: NTFS data hijacking”</p> <p>Here we have FILE_OBJECT, which fields point to the control block structures. Let me call them NTFS data structures. These NTFS structures point to other structures as well. OS does not check the integrity of these structures, which causes the possibility of Hijacking /N-Tee-F-S/ NTFS structures.</p>	<p>FileObject fields point to the NTFS Data Structures</p> 
<p>Slide 67 “Overview: NTFS data structures internals”</p> <p>Here we have NTFS data structures for the secret file and for the file hijacker.</p>	<p>HIJACKING NTFS STRUCTURES</p> 
<p>Slide 68 “Overview: NTFS data structures internals”</p> <p>The idea of attack is to overwrite the NTFS structures for the file hijacker using the data from the secret file.</p>	<p>HIJACKING NTFS STRUCTURES</p> 
<p>Slide 69 “Overview: NTFS data structures internals”</p> <p>This can help to redirect read and write illegal access to the structures related to the secret file. The legal access will also be forwarded to the secret file. While I was checking this idea, I got the following BSOD.</p>	<p>HIJACKING NTFS STRUCTURES</p> 
<p>Slide 70 “Overview: NTFS data structures internals”</p> <p>This bug check indicates that a someone tried to <u>release</u> a <u>resource</u> he did not <u>own</u>. Actually, this is true. I tried to use the resource, which doesn't belong to me. / 3 sec / How Windows OS has revealed this? The reason is here.</p>	<p>BSOD - RESOURCE_NOT_OWNED (0xE3)</p> 
<p>Slide 71 “Overview: NTFS data structures internals”</p> <p>Windows OS <u>checks</u> /no-stress/ whether the current thread ID matches the value stored in the structure.</p>	<p>BSOD: THE REASON AND THE WAY TO BYPASS</p> <pre>void ExReleaseResourceLite(PERESOURCE Resource) { ... CurrentThread = KeGetCurrentThread(); if (!KeOwnedExclusive(Resource)) { if (Resource->OwnerThreadId != CurrentThread) { KeBugCheckEx(RESOURCE_NOT_OWNED, ...); } } }</pre>
<p>Slide 72 “Overview: NTFS data structures internals”</p> <p>If they <u>are</u> not the same the OS calls KeBugCheckEx, which brings down the system.</p>	<p>BSOD: THE REASON AND THE WAY TO BYPASS</p> <pre>void ExReleaseResourceLite(PERESOURCE Resource) { ... CurrentThread = KeGetCurrentThread(); if (!KeOwnedExclusive(Resource)) { if (Resource->OwnerThreadId != CurrentThread) { KeBugCheckEx(RESOURCE_NOT_OWNED, ...); } } }</pre>


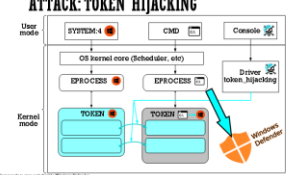
<p>Slide 73 “Overview: NTFS data structures internals”</p> <p>To bypass this check malware driver needs to copy structures and patch the thread ID. Malware needs to repeat these two steps each time before reading and writing the secret file.</p>	<p>BSOD: THE WAY TO BYPASS (FOR HACKERS ONLY)</p> <ol style="list-style-type: none"> 1. Overwrite control block structures 2. Patch ThreadID-related fields using attackers ThreadID: <ul style="list-style-type: none"> • Resource->OwnerEntry.OwnerThread = PsGetCurrentThread(); • PagingIoResource->OwnerEntry.OwnerThread = PsGetCurrentThread(); 3. Repeat steps 1 and 2 before each read and write call
<p>Slide 74 “Scenario of Demo: NTFS data structures internals”</p> <p>Here is the scenario of the demo. The attacker will try to access this file twice: using OS API function</p>	<p>DEMO: HIJACKING NTFS STRUCTURES</p> 
<p>Slide 75 “Scenario of Demo: NTFS data structures internals”</p> <p>and using the Hijacking NTFS structures. Let’s see how it can happen.</p>	<p>DEMO: HIJACKING NTFS STRUCTURES</p> 
<p>Slide 76 “Demo with Hijacking NTFS structures”</p> <p>Let’s check the Windows version. We’ve got the final one.</p> <p>The researcher is launching his app, which loads a driver. The researcher is opening the file. He is writing the formula. He is reading the data to check it. The formula is saved in the secret file <u>but</u> the file is still opened.</p> <p>While the researcher is being proud of his job, and for example, having a tee break, the attacker has a chance to steal the formula.</p> <p>An attacker is launching his app, which loads a driver. First, he tries to open the secret file using Windows API function. And he fails.</p>	





<p>Windows kernel prevents an illegal access to the open files.</p> <p>Then, he starts the hijacking attack on NTFS data structures and he needs to have a file object for the secret file. Let us assume that he has this value.</p> <p>We can see that the attacker creates a file hijacker and successfully patches the corresponding structures. He is trying to read the secret file. The secret formula has been stolen. The attacker is trying to overwrite it.</p> <p>And when the researcher decides to check the formula, he sees that his formula has been illegally modified.</p> <p>What about PatchGuard reaction?</p> <p>7 hours have passed and nothing has happened. The OS became infected and the formula is in danger.</p>	
<p>Slide 77 “MemoryRanger prevents NTFS data hijacking”</p> <p>To prevent this attack my MemoryRanger implements a similar technique. It runs drivers inside isolated énclaves. The MemoryRanger grants access to the NTFS structures, that have been allocated by each driver.</p>	
<p>Slide 78 “MemoryRanger prevents NTFS data hijacking”</p> <p>The MemoryRanger restricts access to the NTFS structures, that have been created for other drivers. Let’s see how it can happen.</p>	
<p>Slide 79 “Demo: MemoryRanger prevents NTFS data hijacking”</p> <p>MemoryRanger hypervisor is the first to be loaded.</p> <p>The researcher is launching his app, which loads a driver.</p> <p>The researcher is opening the file.</p>	

<p>He is writing down the secret formula. He is reading the data to check it. The formula is saved<u>u</u> in the secret file, but this file is still opened.</p> <p>While the researcher is being proud of his achievements and for example having a cup of tea, the attacker has chance to steal the formula.</p> <p>The attacker is launching his app, which loads a driver. First, he is trying to open the secret file using API function. And he fails. Windows prevents all illegal access attempts.</p> <p>He is starting the hijacking attack and he needs to have a file object for the secret file. Let us assume that he's got this value.</p> <p>We can see that now an attacker is failing to patch the NTFS data structures. He is trying to read the secret file. He reads nothing. He fails again. The attacker is trying to overwrite the secret file.</p> <p>Then. The researcher has finished his break and wants to check his formula. The formula has not been changed. MemoryRanger protects the file structures. Now let's close both files to compare their content. We can see that the secret file contains only the secret formula and attackers' <u>i</u>nput is in the file hijacker. Thanks to the MemoryRanger the OS is protected.</p>	
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--

<p>Slide 80 “Token Hijacking”</p> <p>As a result, we have the following picture. My MemoryRanger restricts READ access to the NTFS data structures and prevents Hijacking NTFS structures.</p>	
<p>Slide 81 “Token Hijacking”</p> <p>That is all about attacks on the file system and now let’s move to the process privilege escalation attack. Let me briefly introduce this kind of attacks.</p>	<p>Episode 2 Token Hijacking: Escalation Process Privileges</p>
<p>Slide 82 “Token Hijacking”</p> <p>For each created process OS kernel allocates several internal structures.</p>	
<p>Slide 83 “Token Hijacking”</p> <p>One of them is the EPROCESS structure, which includes the process information and pointers to other structures.</p>	
<p>Slide 84 “Token Hijacking”</p> <p>One of the fields of EPROCESS structure points to the /'təʊkən/ TOKEN structure. Information about process privileges is stored in the /'təʊkən/ TOKEN structure. Malware authors invented various techniques to escalate process privileges.</p>	
<p>Slide 85 “Token Hijacking”</p> <p>Ten years ago, attackers could easily <i>{slow}</i> add more privileges to the process by directly patching the /'təʊkən/ TOKEN structure.</p>	
<p>Slide 86 “Token Hijacking”</p> <p>Windows security experts prevent this attack by adding such fields as SidHash and RestrictedSidHash into the TOKEN structure. This guarantees the integrity of the TOKEN structure.</p>	

<p>Slide 87 “Token Hijacking”</p> <p>Hackers made one step ahead and escalate process privileges by copying the address of a Token structure from a more privileged process. This technique is called token stealing or token /swap/ swapping.</p>	
<p>Slide 88 “Token Hijacking”</p> <p>Newest Microsoft Defender prevents this token swap attack by checking the integrity of these pointers. I will show you how attackers can escalate process privileges without triggering any security features. Now let me show you the details of the TOKEN structure.</p>	
<p>Slide 89 “Token Hijacking: Token Structure”</p> <p>The TOKEN structure includes two parts: static part and a dynamic or /variab(ə)l/ variable part.</p>	
<p>Slide 90 “Token Hijacking: Token Structure”</p> <p>The static part has a fixed size and has a link to the dynamic part. The dynamic part includes the process privileges, which are stored <i>{slow}</i> using the set of SID_AND_ATTRIBUTES structures.</p>	
<p>Slide 91 “Token Hijacking: Token Structure”</p> <p>Here we have two TOKEN structures: for a high privileged system process and for the low privileged console application.</p>	
<p>Slide 92 “Token Hijacking: Token Structure”</p> <p>The proposed Token Hijacking Attack is to copy the whole TOKEN structures of the privileged process including /bəʊθ/ both parts: static and dynamic. Copying the static part is simple. To copy the dynamic part attackers need to consider the interconnections between the structures. Let’s check this idea.</p>	

<p>Slide 93 “Token Hijacking: Attack scenario overview”</p> <p>Here is the scenario of the demo with Token Hijacking.</p> <p>The attacker will start the CMD and his application, which loads a driver to escalate privileges.</p>	
<p>Slide 94 “Token Hijacking: Attack scenario overview”</p> <p>He will check that the privileges have been escalated by trying to disable Windows Defender. The commands to disable Windows Defender are /'borou/ borrowed from a banking trojan, called Trickbot. Let's have a look.</p>	
<p>Slide 95 “Token Hijacking: Attack Demo”</p> <p>Let's check the Windows version. We've got the final one.</p> <p>The attacker is launching the CMD.</p> <p>We can see that CMD has user's privileges.</p> <p>The attacker is trying to disable Windows Defender using the commands from the trojan.</p> <p>We can see a lot of /'erə(r)/ error /'mesɪdʒz/ messages. They are the same and all of them mean that CMD doesn't have enough privileges to disable Windows Defender.</p> <p>The attacker needs to escalate CMD privileges to disable Windows Defender.</p> <p>The attacker is launching his app, which loads a driver.</p> <p>To start the token hijacking attack, he needs to know a CMD's process ID.</p> <p>He is getting and copying the target process ID. We can see that the /'təʊkən/ token structure has been patched.</p> <p>The attacker is checking the privileges again.</p>	

<p>We can see that privileges have been elevated. The CMD has the SYSTEM privileges.</p> <p>The attacker is trying to disable Windows Defender again and he /'knpi - peists/ copy-pastes the commands from the trojan once more.</p> <p>He does not receive any error messages. This time he succeeds.</p> <p>We've just received a notification, which confirms that Windows Defender has been disabled. The attacker can also disable even this notification.</p> <p>But let's see the PatchGuard reaction, which is designed to prevent illegal memory modifications. We've been waiting for 10 hours it is quite a long time and nothing has happened. The OS has not been crushed. It means that PatchGuard hasn't prevent this invasion. The OS became infected and our data is in danger.</p>	
<p>Slide 96 “Token Hijacking: Idea and its Implementation using separate énclave for data only”</p> <p>How to prevent Token Hijacking?</p> <p>The key feature of the Token structure is that only Windows kernel needs to access them, all other access attempts can and must be {0.5sec} blocked.</p>	
<p>Slide 97 “Token Hijacking: Idea and its Implementation using separate énclave for data only”</p> <p>My idea is to create a special isolated énclave for sensitive data.</p> <p>This énclave will include only Token structures and Windows OS kernel core.</p>	
<p>Slide 98 “Token Hijacking: Idea and its Implementation using separate énclave for data only”</p> <p>All other drivers will be excluded from this énclave.</p> <p>The Token structures will be excluded from all other énclaves.</p>	
<p>Slide 99 “Token Hijacking: Idea and its Implementation using separate énclave for data only”</p> <p>This idea helps to prevent illegal access to the TOKEN structures from all drivers loaded before MemoryRanger and after it. Let's see how it works.</p>	

Slide 100 “Token Hijacking: Prevention attack Demo”

Let’s check the Windows version. We’ve got the final one.

The attacker is launching its app, which loads a malware driver.

Then, we are launching the MemoryRanger to protect the OS.

The attacker is launching the CMD. He is checking the privileges. CMD has user’s privileges.

The attacker is trying to disable Windows Defender again using the commands from the trojan. CMD does not have enough privileges.

To escalate process privileges the attacker is getting and copying the process ID.

The attacker is running the TOKEN hijacking attack.

We can see that now the TOKEN structure cannot be patched.

The attacker is checking the privileges again. We can see that CMD still has user’s privileges.

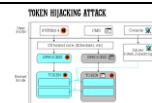
Now an attacker is trying to disable Windows Defender again.

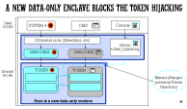


We can see a lot of error messages, that confirm that CMD does not have enough privileges.

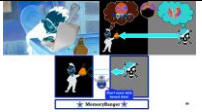


Thanks to my MemoryRanger the OS is protected.

Slide 101 “MemoryRanger: briefly overview main updates”

As a result, we have the following picture. The MemoryRanger prevents patching the Token structures



<p>Slide 102 “MemoryRanger: briefly overview main updates” by moving them to the special data-only enclave.</p>	
<p>Slide 103 “MemoryRanger: briefly overview main updates” Now let’s move on to the brief overview of my MemoryRanger and its main updates.</p>	
<p>Slide 104 “MemoryRanger overview” The MemoryRanger is a tool designed to prevent attacks on the OS kernel with the help of kernel-mode driver and a hardware-based hypervisor.</p> <ul style="list-style-type: none"> • The MemoryRanger registers a notification routine to intercept the loading of new drivers. The MemoryRanger allocates a separate enclave for each newly loaded driver. • The MemoryRanger also intercepts the launching of new applications. This helps to locate TOKEN structures and move them to the data-only enclave. • The MemoryRanger can hook kernel API functions, for example ZwCreateFile to locate the created internal file structures and restrict access to them. • Hypervisor controls access to the kernel memory by restricting access and trapping EPT-violations. The MemoryRanger provides a flexible memory access policy. <p>The MemoryRanger can be easily updated to prevent new attacks on the OS kernel.</p>	
<p>Slide 105 “Conclusion” Let me recap very briefly on what I have presented and what you learn. First of all, some Windows kernel data can be easily modified by attackers without triggering any security features. We have seen two hijacking attacks to access files opened without shared permissions and a new technique to elevate process privileges. I have presented an updated MemoryRanger, which can prevent all these attacks. One of the new features includes a special data-only enclave to isolate sensitive data from all drivers.</p>	

<p>Slide 106 “Scenario of attacks on files”</p>	
<p>Slide 107 “Thanks to MemoryRanger the world is saved” Eventually, thanks to the MemoryRanger the world is saved.</p>	
<p>Slide 108 “Thank you” Thank you!</p>	

Slide 109 “Special Edition for ADFSL Conference”

So let’s move on to the MemoryRanger updates, which can prevent illegal patching of entries. MemoryRanger is a hypervisor-based solution designed to restrict access to the kernel memory. MemoryRanger allocates an isolated memory enclave for each newly loaded driver. MemoryRanger uses hardware virtualization technologies (VT-X and EPT) to restrict access to the sensitive data by changing memory permission bits. Each time any driver tries to reads or overwrite such memory areas an violation is triggering and control goes to the hypervisor. MemoryRanger dispatches these violations. It can temporarily allow access to the memory or block it by redirected the fake memory page.

~~Well, without MemoryRanger all drivers and OS share the same memory space.
Now let’s roll back and launch MemoryRanger hypervisor the first.
After its loading MemoryRanger allocates the default enclave, which includes the OS and all loaded drivers.
The hypervisor traps the loading of the researcher driver and allocates a special memory enclave for it.
Next MemoryRanger traps that the researcher creates a file. MemoryRanger restricts access to this memory inside the default enclave and allows it inside enclave for the researcher’s driver.
In the same way MemoryRanger allocates a separate enclave for the attacker’s driver and updates memory access restrictions.
As a result, this scheme helps to prevent illegal access to the handle table entries.
Let us have a look how this scheme works in practice.~~