**Windows built-in Sandbox Disables Microsoft Defender and other EDR/AVs:**

**Attack Detection and Prevention via MemoryRanger**

|  |  |
| --- | --- |
| **Slide 1 Hello**  😊 Hi everyone! 😊  Thanks for inviting me. I am happy to be here.  Today I`ll show you how malware *can* bypass ... *can* disable Microsoft Defender by abusing system mechanisms and how to avoid this kind of attacks.  I think this topic is of crucial importance for endpoint security and anti-viruses’ experts. |  |
| **Slide 2 “WhoamI”** (fast: speed=3)  • This work was done in collaboration with one of my students from National Research Nuclear University.  Dénis investigated attacks on Microsoft Defender in his bachelor thesis under my supervision.  • I’m Igor. I’ve been exploring **/ˌəʊ ˈes/** OS security for more than 10 years.  I’ve been **curious** about discovering new attacks on the Windows kernel and find the ways to prevent them. |  |
| **Slide 3 “TITLE”**   * Cybercriminals actively use various money-making malware, including Banking Trojans, cryptojacking malware and Ransomware. * They use various Defense Evasion techniques to maintain a persistent and undetectable presence on the infected machine. For example, attackers can improve malware resiliency using code obfuscation. * New malware samples can bypass or disable endpoint security solutions, such as Microsoft Defender. * This is very important. |  |
| **Slide 4 “Microsoft Defender is running on over 500 000 000 PCs”**  You know that, Microsoft Defender is installed as a primary anti-virus on more than half a billion devices.  The partner director at Microsoft underlined that Microsoft Defender is facing a **/hjuːdʒ/** **hu-uge** number of various attacks. Security experts are discovering and discussing new techniques to disable or bypass Microsoft Defender. |  |
| **Slide 5 “Attack vectors on Microsoft Defender”**  Let me show you the current attack vectors on Microsoft Defender. |  |
| **Slide 6 “Thread modeling Microsoft Defender”**   * Here we can see the scope of user-mode and kernel-mode attacks on Microsoft Defender. * For example, attackers can abuse legitimate apps, modify the Microsoft Defender settings, and also load kernel driver to disable callback notify routines. * Here we have five user-mode attack vectors and three kernel-mode ones. * Regarding user-mode attacks. |  |
| **Slide 7 “User-mode Attacks: 1, 2”**   * In technique number 1 – attackers abuse Windows built-ín apps as well as whitelist apps to run malware code.   For internal techniques see number two, attackers can use the following:   * Packing and obfuscation to avoid signature-based detection, * Token Impersonation to abuse TrustedInstaller in order to stop Microsoft Defender. * Custom loader that bypasses kernel callbacks. |  |
| **Slide 8 “User-mode Attacks: 3, 4, 5”**   * Technique 3 is based on modifying the settings of Microsoft Defender and its components. You can see the details in the table below. * Technique number 4 results in unloading Microsoft Defender driver by abusing NT symbolic links. * The technique number 5 uses syscall functions to modify the Token of Microsoft Defender and finally sandbox or isolate Microsoft Defender apps from the rest of the OS. * Microsoft has rolled out a patch to block this user-mode sandboxing and now it doesn’t work. |  |
| **Slide 9 “Kernel-mode Attacks”**   * However, Denis Pogonin independently discovered a similar kernel-mode attack. Today you will see the details of this attacks and its prevention. * In technique number 7 hackers can disable various kernel callbacks, such process and thread callbacks. As a result, the Microsoft Defender stops receiving notifications about new events. You can see the details in the table below. * Finally, attackers can modify ETW data structures to kill ETW sessions. * All the links are clickable. |  |
| **Slide 10 “”**  A few words about Microsoft Defender components. |  |
| **Slide 11 “TITLE”**   * Microsoft Defender includes about a dózen of user-mode apps and services and 6 kernel mode drivers. * In this research we analyzed these two components Microsoft Defender Core Service (~~named Microsoft Malware Protection Engine~~) and Microsoft Defender file filter driver. |  |
| **Slide 12 “Microsoft Malware Protection Engine”**  Microsoft Defender Core Service is a is just a normal windows app that can detect the malware   * on the disk * and in the memory. |  |
| **Slide 13 “Microsoft Malware Protection Engine (MpEngine.dll)”**  One of its DLL modules named MpEngine implements the signature based-detection. It is the brain of anti-virus engine that decides whether the input is malicious or not. |  |
| **Slide 14 “TITLE”**  Thanks for this engine Microsoft Defender is able to remove malware from the disk and block malicious apps in memory. |  |
| **Slide 15 “”**   * An idea of the attack is to disable the brain {pause} and block it from receiving any input data. * As a result, Microsoft Defender cannot access files on the disk and apps memory. * This attack can be divided into two parts. |  |
| **Slide 16 “”**   * Blocking access to the files system. |  |
| **Slide 17 “”**   * Blocking access to the memory. |  |
| **Slide 18 “”**   * First of let’s look at how to prevent inspection of files on the disk. * ~~I will show how Windows built-in mechanism can be used to disable Microsoft Defender.~~ * ~~Let’s look at the Mándatóry Integrity Control (MIC)~~ |  |
| **Slide 19 “”**  Each time any app for example Microsoft Defender tries to access the object for example open a file. |  |
| **Slide 20 “”**  the Security Reference Monitor checks whether this access is allowed. |  |
| **Slide 21 “”**  It was originally based on Discretionary Access Control, DeeACee |  |
| **Slide 22 “”**  Mandatory Integrity Control, MICee has been added in addition to DeeACee.  MICee was designed to isolate untrusted apps from the rest of the OS. |  |
| **Slide 23 “”**  Now we have two different access checks. |  |
| **Slide 24 “”**  DeeACee checks privileges using access token and security descriptor. |  |
| **Slide 25 “”**  While MICee reads and compares Integrity Levels from access token and from object descriptor. |  |
| **Slide 26 “”**  Windows first checks an access request against MIC |  |
| **Slide 27 “”**  and if it passes, then it checks DAC. |  |
| **Slide 28 “”**  To **/allÁw/** allow access it has passed both checks. But if MIC check didn’t pass the access will be blocked. |  |
| **Slide 29 “”**  Attackers can patch integrity level for the Defender so that any Defender’s access attempt will be blocked by Windows. |  |
| **Slide 30 “”**  For this attack I have to show you two points:   * How Integrity Level values are stored in memory * And how they are processed during access check. |  |
| **Slide 31 “”**  Integrity Levels |  |
| **Slide 32 “”**  Windows supports several Integrity Levels to control access to the objects: |  |
| **Slide 33 “”**  Untrusted and Low Integrity Levels are usually used by browsers to isolate web threats from the OS. |  |
| **Slide 34 “”**  Most apps that we use {*pause*} are running with Medium Integrity Level, such as explorer, and notepad. |  |
| **Slide 35 “”**  High Integrity Level is assigned with apps launched by User Account Control, UAC. |  |
| **Slide 36 “”**  System Integrity Level is used only by system-level apps, such as Microsoft Defender, to protect them from other apps. |  |
| **Slide 37 “”**  MIC implements the Bell-LaPadula access control model that compares integrity levels of app and a file.  The access will be granted if the app’s integrity level is equal to {pause} or higher than {pause} the file’s integrity level.  Therefore, apps with Low Integrity Levels cannot overwrite the most OS objects. |  |
| **Slide 38 “”**  An idea of attack is to lower integrity level for Microsoft Defender, so that it cannot access files on the disk. |  |
| **Slide 39 “”**  Let me show you how Integrity Levels are implemented in Windows. |  |
| **Slide 40 “”**  For each app Windows allocates a separate data structure in kernel memory named Token structure.  1– |  |
| **Slide 41 “”**  2 |  |
| **Slide 42 “”**  that includes all information about process privileges.  The set of process privileges is stored using Security identifiers, SIDees.  To support MIC two new fields have been added: |  |
| **Slide 43 “”**   * IntegrityLevelIndex and the corresponding SID\_AND\_ATTRIBUTES structure. * IntegrityLevelIndex is just an index into the array pointed by the UserAndGroups.   Let me show you how Windows gets these Integrity Levels. |  |
| **Slide 44 “”**  Burum-burum |  |
| **Slide 45 “”**  Each time any app for example Microsoft Defender tries to access the object for example open a file. |  |
| **Slide 46 “”**  The control goes to the NtOpenFile or NtCreateFile, from ntdll |  |
| **Slide 47 “”**  then to the kernel. We have a list of function calls. |  |
| **Slide 48 “”**  Finally, the control goes to the function SepMandatoryIntegrityCheck.  Let’s see inside it. |  |
| **Slide 49 “”**  The function reads the IntegrityLevelIndex value. |  |
| **Slide 50 “”**  If it is minus one, the untrusted SIDee is copied to the output. |  |
| **Slide 51 “”**  Or else, {pause} the data pointed to the Integrity Level SID are copied to the output. |  |
| **Slide 52 “”**  It means that attacker can set -1 to the IntegrityLevelIndex to mark the target process as an untrusted.  Let’s see how it can happen. |  |
| **Slide 53 “”**  Burum-burum. |  |
| **Slide 54 “”**  At first, attackers can load kernel driver to overwrite the IntegrityLevelIndex for the Microsoft Defender. |  |
| **Slide 55 “”**  We will check if Microsoft Defender is ON by extracting and launching a malware sample. |  |
| **Slide 56 “”**  The Mimikatz app will be used as a malware. We initially zipped mimikatz with password. |  |
| **Slide 57 “”**  Here is the batch script that extracts the Mimikatz from the archive file and launch it.  Let’s see how it can happen. |  |
| **Slide 58 “”**  <demo> |  |
| **Slide 59 “”**  Microsoft Defender fails to remove the file, but it is still able to block the running process. |  |
| **Slide 60 “”**  We’ve just seen some Microsoft Defender internals. It removes the malware file using pending deletion.  Also, we’ve seen a new **/ˈsté-ɪtəs/** return status that blocks running a malware app. |  |
| **Slide 61 “”**  Let me show you the driver that returns this status. |  |
| **Slide 62 “”**  My research has revealed that this status is returned by Microsoft Defender file-filter driver.  This driver registers a mini-filter to monitor file operations using pre- and post- processing callbacks. |  |
| **Slide 63 “”**  I have discovered that post-create callback function returns this status for the malware apps. |  |
| **Slide 64 “”**  It means that Microsoft Defender can still detect malware in apps memory. But how? |  |
| **Slide 65 “”**  Burum-burum |  |
| **Slide 66 “”**  We know that malware detection engine is located in the Microsoft Defender process.  Each time the Defender process gains the access to the running process. |  |
| **Slide 67 “”**  Windows checks whether the privileges are enough using |  |
| **Slide 68 “”**  access token and security descriptor. |  |
| **Slide 69 “”**  However, the Defender process has a Debug Privilege to open a process running on another user account, |  |
| **Slide 70 “”**  Processes running with Debug privilege always get /**fúll**/ full access without any checks. |  |
| **Slide 71 “”**  Attackers *can* revoke this Debug Privilege from Microsoft Defender to block access to app. |  |
| **Slide 72 “”**  The mentioned privilege is stored in Token structure in memory. And *can* be patched by attackers.  Let’s see who it can happen. |  |
| **Slide 73 “”**  Burum-burum |  |
| **Slide 74 “”**  At first, attackers load the kernel driver to perform two actions:   * overwrite the IntegrityLevelIndex * and revoke process privilege.   Let’s see whether these manipulations will be enough to disable Microsoft Defender. |  |
| **Slide 75 “”**  <demo> |  |
| **Slide 76 “”**  We *can* see that now Microsoft Defender has been completely disabled without triggering any Patch Guard reaction. |  |
| **Slide 77 “”**  Burum-burum |  |
| **Slide 78 “”**   * This attack results in sandboxing or isolating Microsoft Defender from the rest of the OS. * The key point is that we do not terminate Microsoft Defender * It’s still running but it cannot detect malware on the disk and in the memory.   Microsoft Defender has been disabled.  What about other popular anti-viruses? |  |
| **Slide 79 “”**  Burum-burum |  |
| **Slide 80 “”**  Here are the results of checking the given attack against top Windows anti-viruses.  We can see that vast majority of AVs has been successfully disabled.  Let me show how this attack can be blocked. |  |
| **Slide 81 “”**  To demonstrate the attack on AVG AntiVirus let’s look at the demo prepared by Denis.  <demo>  Thank you, Denis! |  |
| **Slide 82 “”**  Burum-burum |  |
| **Slide 83 “”**  In Windows OS kernel drivers share the same memory space with the OS kernel. |  |
| **Slide 84 “”**  There is no built-ín mechanism to restrict illegal access attempts in kernel memory. |  |
| **Slide 85 “”→ 86 “”→ 87 “”**  I designed MemoryRanger to solve this problem by allocating a separate kernel enclave for each newly loaded driver.  2→3  As a result, drivers are running inside separate enclaves with different memory access permissions. |  |
| **Slide 88 “”**  This scheme helps to block illegal access attempts. |  |
| **Slide 89 “”**  Originally, I designed MemoryRanger as a tool, as a framework to monitor access to the kernel data.  Now it can hook kernel functions; it can locate various sensitive data in kernel; and it can be easily customized.  Let’s have a look. |  |
| **Slide 90 “”**  Here are the examples of my MemoryRanger customizations that block various new attacks on Windows kernel.  In this research MemoryRanger has been customized in the following way. |  |
| **Slide 91 “”**  Without my MemoryRanger attacker’s driver can get access to the Token structure. |  |
| **Slide 92 “”**  My MemoryRanger traps loading of attacker’s driver and moves it to the separate allocated enclave.  In this enclave the Token structure cannot be accessed. Let’s see how it can happen. |  |
| **Slide 93 “”**  <demo> |  |
| **Slide 94 “”**  Let’s discuss briefly what we’ve seen.  Without my MemoryRanger attackers can disable Microsoft Defender and install malware without any problem.  Microsoft Defender fails to remove the malware. |  |
| **Slide 95 “”**  But with MemoryRanger the situation is different. |  |
| **Slide 96 “”**  MemoryRanger allocates the default enclave for the OS kernel and all previously loaded drivers. |  |
| **Slide 97 “”**  MemoryRanger traps the loading of attacker’s driver and allocates a separate memory enclave for this driver. |  |
| **Slide 98 “”**  In this enclave MemoryRanger restricts any access attempt to the Token of Microsoft Defender. |  |
| **Slide 99 “”**  As a result, attackers have failed to disable Microsoft Defender.  This is the way how MemoryRanger protects Microsoft Defender. |  |
| **Slide 100 “”**  As we know MemoryRanger hypervisor protects machines that support hardware virtualization technology.  But what about other computers?  Can we protect the computers without using hypervisor mechanism? |  |
| **Slide 101 “”**  Let’s have a look at Denis answer.  <Demo>  Thank you, Denis! |  |
| **Slide 102 “”**  Let me /ˈriːkap/ ré-cáp very briefly on what we have discussed so far.   1. First of all, driver-based attacks are crucial. 2. Intruders are plotting to bypass endpoint security solutions and their target number one is Microsoft Defender. 3. Hackers can apply M**á**ndatory Integrity Control and Token Privilege against Microsoft Defender to isolate it. 4. This is a very serious attack, because it blinds vast majority of anti-viruses without triggering any Patch Guard reaction.   I have presented my MemoryRanger, that customized version can prevent this kind of attacks.  Denis presented his tool named ProcessTokenMonitor to detect such attacks from user space. |  |
| **Slide 103 “”**  Thank you! |  |