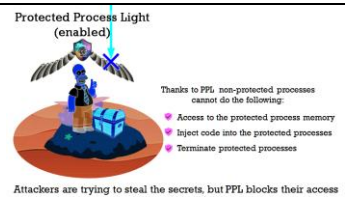
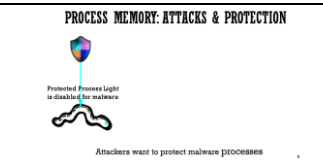
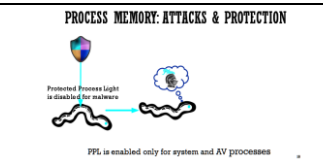
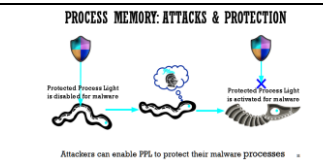
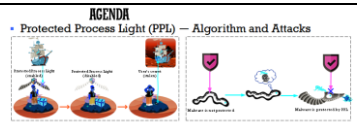
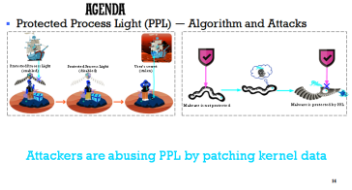
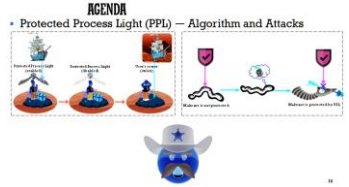
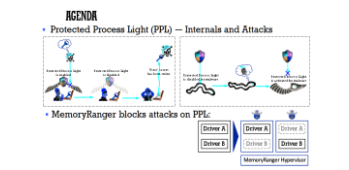
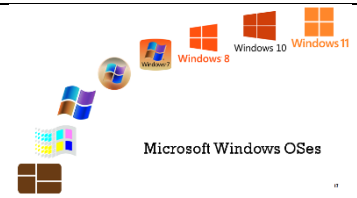
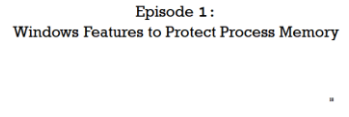
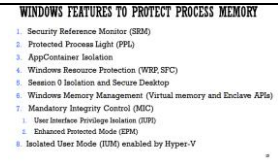
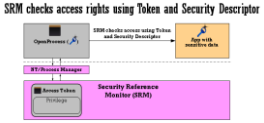
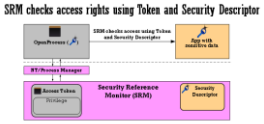
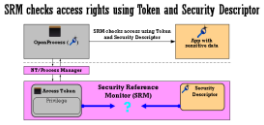
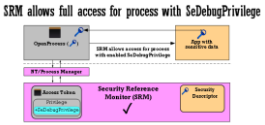
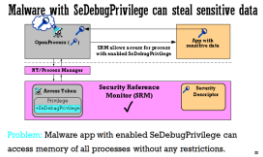
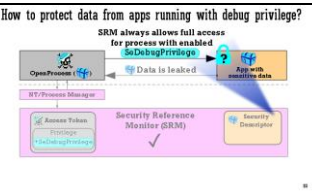
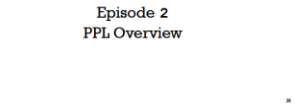
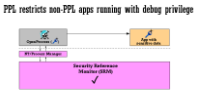
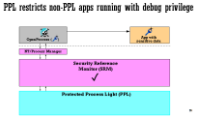
Protected Process Light will be Protected – MemoryRanger Fills the Gap Again

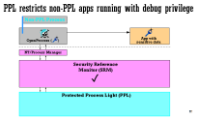
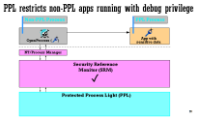
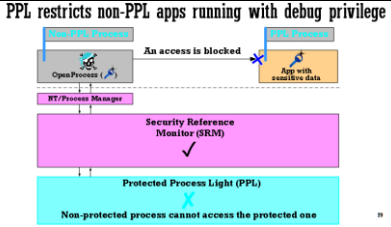
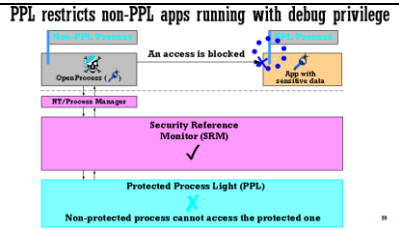
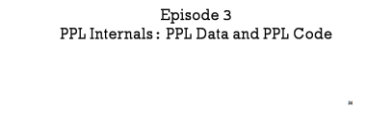
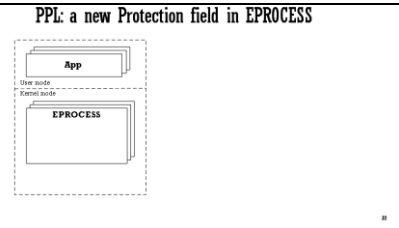
Slide 1 Hello 😊 Hi everyone! 😊 Thanks for inviting me. I am happy to be here. Today I will be talking about undetected kernel attacks on one of Windows mechanisms called Protected Process Light and how to avoid this kind of attacks. I think this topic is of crucial importance for all experts dealing with application security.	Protected Process Light is not Protected - MemoryRanger Fills the Gap Again Igor Korkin Independent Researcher 2021
Slide 2 "WhoamI" (fast: speed=3) A few words about me. I've been exploring **/ əʊ ˈes/** OS security for more than 10 years. I've been **curious** about applying both theoretical and practical expertise to discover new attacks on the OS kernel and find the ways to prevent them. You can find all the information in my blog.	WHOAMI • PhD, speaker at the ADFSL, BlackHat, HITB, IEEE SPW • OS Security Researcher: • Rootkits, Anti-rootkits and EDRs • Memory Forensics for user- and kernel- modes • Bare-Metal Hypervisors against Attacks on Kernel Memory • Fan of cross-disciplinary research — igorkorkin.blogspot.com • Love traveling and powerlifting — @ igor.korkin
Slide 3 "WhoamI" (fast: speed=2) Today, I will be showing you my analysis of a Windows mechanism called Protected Process Light. We'll be seeing its algorithm and its vulnerabilities.	AGENDA • Protected Process Light (PPL) — Internals and Attacks
Slide 4 "Agenda: PeePeeL" You know {pause}, a great **amóunt** of sensitive data {pause} is located in process memory. Providing data protection at run time is always a challenge.	AGENDA  Users secrets are stored in process memory
Slide 5 "Agenda: PeePeeL disabling" Protected Process Light is a Windows mechanism designed	AGENDA Protected Process Light (enabled)  PPL is enabled for the process to protect its memory

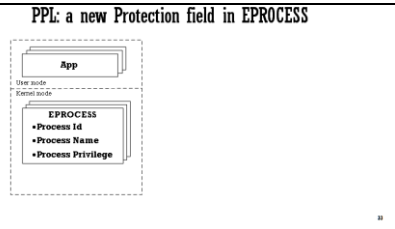
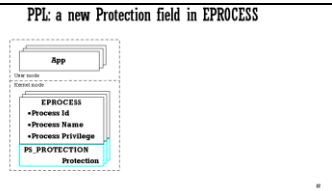
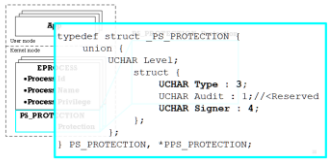
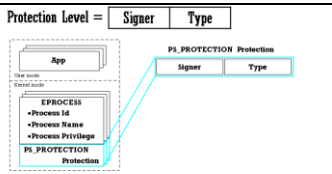
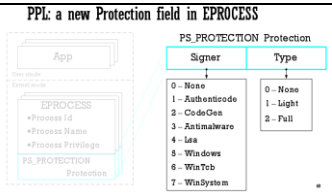
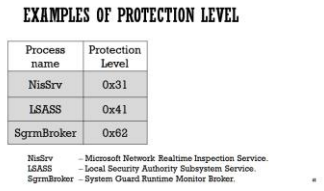
Slide 6 “Agenda: PeePeeL disabling” to protect … to guard sensitive data against malicious attacks.	
Slide 7 “Agenda: PeePeeL disabling” For example, protected processes cannot be dumped or terminated by non-protected processes.	
Slide 8 “Agenda: PeePeeL disabling” But attackers can disable PeePeeL for the target processes and	
Slide 9 “Agenda: PeePeeL disabling” steal users’ secrets {pause}.	
Slide 10 “Agenda: PeePeeL illegal enabling” At the same time, attackers are always looking for the ways to protect their own malware from being detected.	
Slide 11 “Agenda: PeePeeL illegal enabling” PeePeeL seems very promising for attackers, but there are no PeePeeL functions to enable protection for the third-party apps.	
Slide 12 “Agenda: PeePeeL illegal enabling” But attackers can illegally enable PeePeeL for malware processes to protect their apps from being detected.	

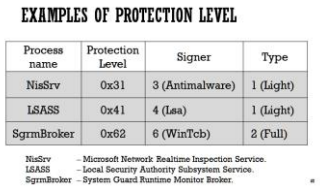
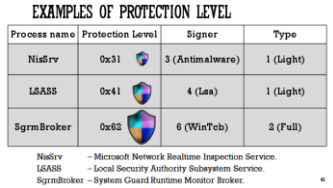
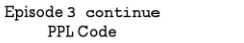
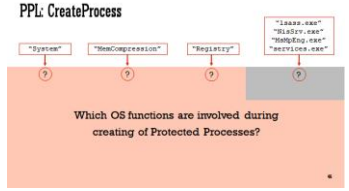
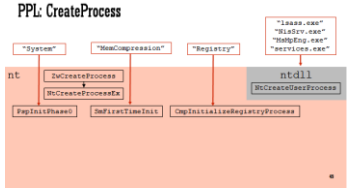
Slide 13 "Agenda: PeePeeL disabling and illegal enabling" Both these attacks can be implemented ..	
Slide 14 "Agenda: PeePeeL disabling and illegal enabling" by modifying kernel memory, without triggering Windows security features. However, {pause} these attacks can be blocked ..	
Slide 15 "Agenda: PeePeeL disabling and illegal enabling" by my MemoryRanger {pause}.	
Slide 16 "Agenda: MemoryRanger protects PeePeeL" MemoryRanger is **{very slow}** the solution, **{very slow}** the tool, **{very slow}** the utility, which I designed to prevent attacks on kernel memory and **we {p} will be seeing** how my MemoryRanger can successfully block **all the attacks** on PeePeeL.	
Slide 17 "Agenda: MemoryRanger protects PeePeeL" Windows experts have developed various process protection mechanisms.	
Slide 18 "Windows Process Protection Mechanisms" Episode 1 {pause-music}	

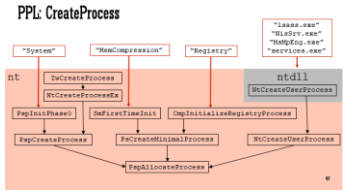
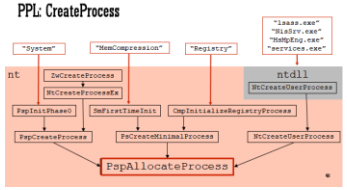
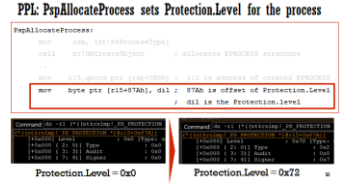
Slide 19 "Security Reference Monitor" One of the essential Windows component**s** is	
Slide 20 "Security Reference Monitor" Security Reference Monitor, which performs access check. Let's see how it works.	
Slide 21 "Security Reference Monitor Principles" OpenProcess() routine **calls** {p} Security Reference Monitor to **check** whether a process has enough privilege to access memory of another process.	
Slide 22 "Security Reference Monitor Principles" • Each process has its own Access Token, which identifies the privileges.	
Slide 23 "Security Reference Monitor Principles" • The Security Descriptor stores the process security attributes. Security Reference Monitor performs access check by comparing..	
Slide 24 "Security Reference Monitor Principles" .. Access Token and Security Descriptor.	
Slide 25 "Security Reference Monitor Principles does not restrict processes with the debug privilege" But for the process running with a debug privilege, Security Reference Monitor always allows **fúll** access without any checks.	

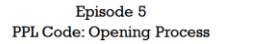
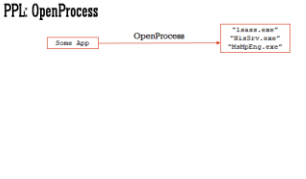
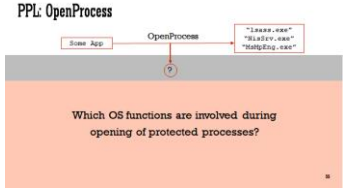
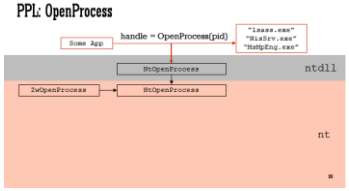
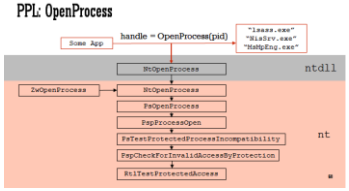
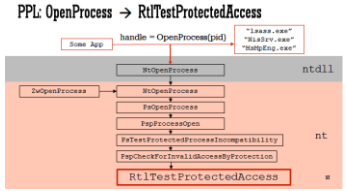
Slide 26 “SRM does not restrict processes running with administrative privileges” Attackers can enable the debug privilege by using a Windows API routine. Therefore, the **malware** process {p} **can get** **ácccess** to the sensitive data. That’s {p} the way how Security Reference Monitor works.	
Slide 27 “PeePeeL mechanism: Stop Access Private Data” How to fill this /gæp/ gap with data protection and prevent the access attempts to the critical process memory? Windows experts had faced this problem, and they introduced **one more** security mechanism.	
Slide 28 “Episode 2: PPL Overview” {Music}-{Pause}	
Slide 29 “PeePeeL mechanism: Stop Access Private Data” It is called Protected Processes Light, or PeePeeL.	
Slide 30 “PeePeeL mechanism: Stop Access Private Data” PeePeeL adds	
Slide 31 “PeePeeL mechanism: Stop Access Private Data” an additional independent layer to protect process data.	

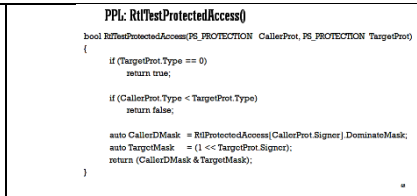
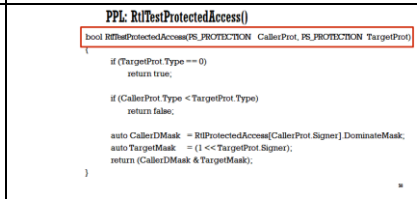
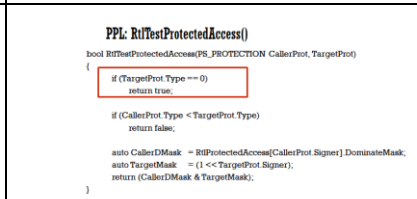
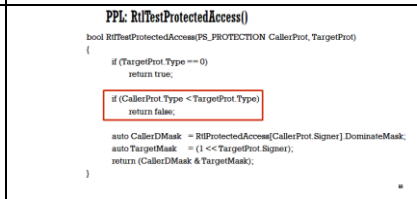
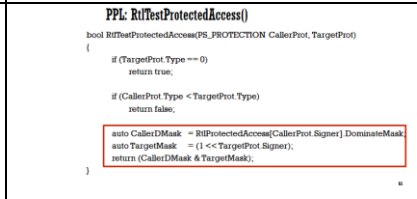
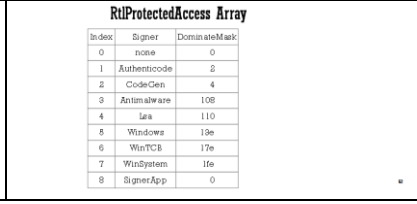
Slide 32 "PeePeeL mechanism: Stop Access Private Data" OS **marks** some apps as protected or PeePeeL processes	
Slide 33 "PeePeeL mechanism: Stop Access Private Data" while other apps are marked as not protected.	
Slide 34 "PeePeeL mechanism: Stop Access Private Data" Now, any non-protected process**es** cann**o**t get access to the data of protected one.	
Slide 35 "PeePeeL mechanism: Stop Access Private Data" The illegal access is blocked. It seems that the data protection can be performed by PeePeeL, but let's analyze its algorithm carefully.	
Slide 36 "Episode 3 PPL Internals: PPL Data and PPL Code" {Pause-music}	
Slide 37 "PeePeeL Internals: New Fields in EPROCESS" As you know **e-each** Windows process is represented by a kernel structure called EPROCESS.	

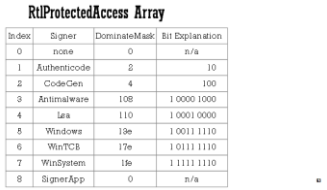
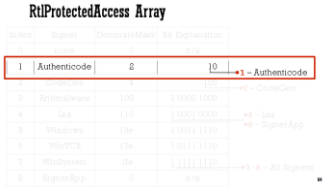
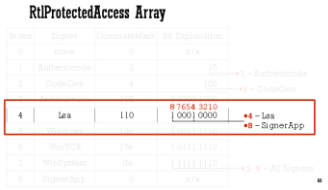
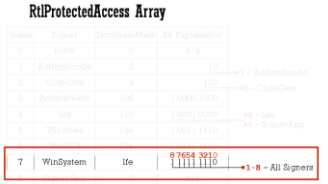
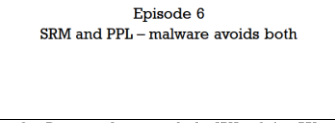
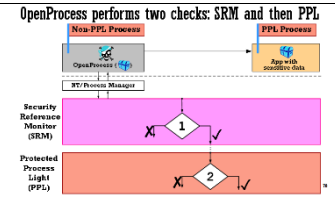
Slide 38 “PeePeeL Internals: New Fields in EPROCESS” It includes information about the process, like process ID, full name, process privileges and other related structures. To **/səˈpɔːt/** support PeePeeL the EPROCESS structure has been updated.	
Slide 39 “PeePeeL Internals: New Fields in EPROCESS” A new field named Protection has been added.	
Slide 40 “PeePeeL Internals: New Fields in EPROCESS” The protection level is stored in a PS_PROTECTION structure, which is here. All the information is stored in a single byte in the two parts.	
Slide 41 “PeePeeL Internals: PS_PROTECTION structure” A signer of protected process and a type.	
Slide 42 “PeePeeL Internals: PS_PROTECTION structure” Signer can have 9 different values, while type just three. The protection level is defined by a combination of these two fields.	
Slide 43 “PeePeeL Internals: PS_PROTECTION structure” Various system processes have different Protection Level values. For example, LSASS process is running with the Protection value 41.	

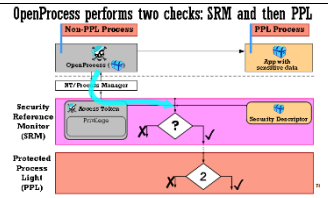
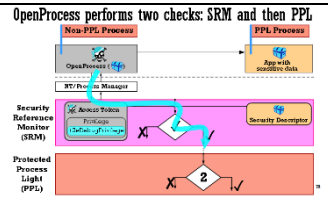
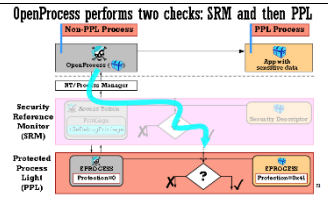
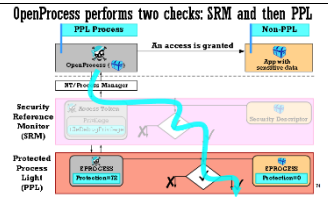
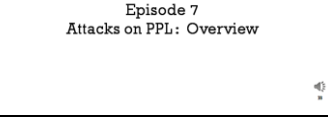
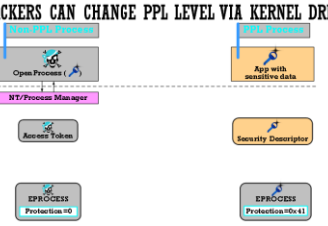
SLIDE 44 “PeePeeL Internals: PS_PROTECTION structure” As we know, each protection level is a combination of the Signer and Type.	
Slide 45 “PeePeeL Internals: PS_PROTECTION structure” Briefly speaking, processes with higher protection levels are more protected. They cannót be áccessed by processes with lower protection levels. How does Windows initiate the Protection level for new launched processes?	
Slide 46 “Episode 4: PPL Code: Creating Protected Processes” {Pause-music}	
Slide 47 “PeePeeL Internals: PS_PROTECTION structure” The following protected processes are launched during Windows startup.	
Slide 48 “PeePeeL Internals: PS_PROTECTION structure” Which OS functions are involved to launch them?	
Slide 49 “PeePeeL Internals: PS_PROTECTION structure” We can see that various API functions ..	

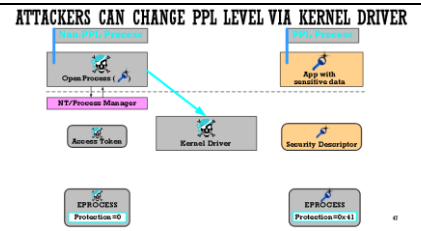
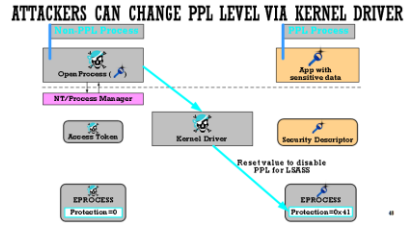
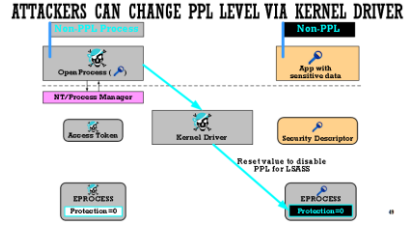
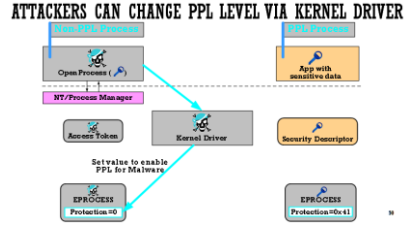
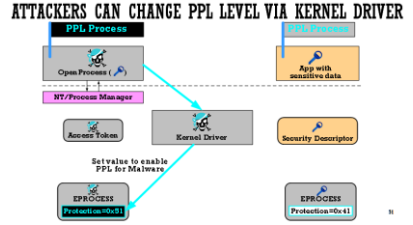
Slide 50 “PeePeeL Internals: PS_PROTECTION structure” are involved in launching protected processes. And finally, all of them call ..	
Slide 51 “PeePeeL Internals: PS_PROTECTION structure” the same function PspAllocateProcess. This very function sets the Protection Level, let see how.	
Slide 52 “PeePeeL Internals: PS_PROTECTION structure” This function is here. It allocates memory for the EPROCESS structure by calling ObCreateObject().	
Slide 53 “PeePeeL Internals: PS_PROTECTION structure” The address of EPROCESS is saved in R15-register.	
Slide 54 “PeePeeL Internals: PS_PROTECTION structure” Finally, the value of DIL is copied to the Protection Level of EPROCESS structure.	
Slide 55 “PeePeeL Internals: PS_PROTECTION structure” Here we can see the content of the Protection Level before setting the level and after that. This is the way how Windows sets the Protection Level. Well, how the Protection Level is used during OpenProcess routine?	

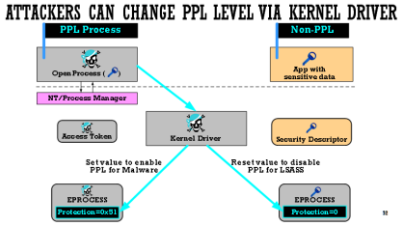
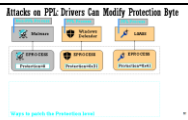
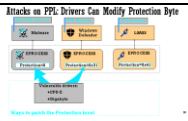
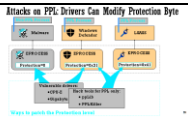
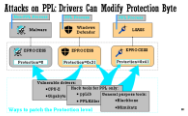
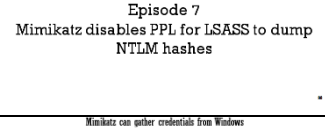
Slide 56 “Episode 5: PPL Code: Opening Protected Processes” {Pause}-{sound}	
Slide 57 “PeePeeL Internals: New values” One app is trying to open a protected process.	
Slide 58 “PeePeeL Internals: New values” Which OS functions are involved to open a process?	
Slide 59 “PeePeeL Internals: New values” The control goes to the NtOpenProcess, from ntdll.	
Slide 60 “PeePeeL Internals: New values” We have a list of function calls. And finally, the control goes to the function RtlTestProtectedAccess.	
Slide 61 “PeePeeL Internals: New values” This very function checks the process protection permissions, let’s see how.	

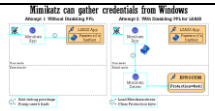
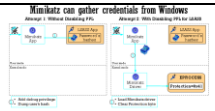
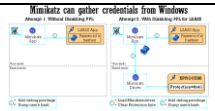
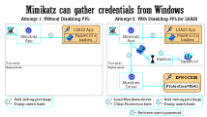
Slide 62 "PeePeeL Internals: New values" This function is here.	
Slide 63 "PeePeeL Internals: New values" The function has two **í**nput parameters – Protection values for the caller and the target processes.	
Slide 64 "PeePeeL Internals: New values" First, it checks the target protection level. If it is zero the access is granted. For example, Mimikatz. You know it is a software tool that can retrieve the credentials. Mimikatz uses this feature to **á**ccess LSASS memory. If the target process is protected, the control goes to the second check.	
Slide 65 "PeePeeL Internals: New values" Now the function is comparing the Protection types. If the Target type is bigger, it blocks the access. If the Caller type is big enough the control goes to the final check.	
Slide 66 "PeePeeL Internals: New values" Now, the function is checking whether a caller dominates the target using an especial array called RtlProtectedAccess.	
Slide 67 "PeePeeL Internals: New values" This data array is one more structure created for PeePeeL mechanism. The part of this array is here. For each Signer type, the array includes the field called "DominateMask". / **AEAEA**/. This field indicates the privilege for each Signer type.	

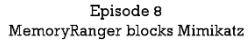
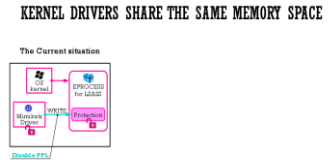
Slide 68 “PeePeeL Internals: New values” Let me explain.	
Slide 69 “PeePeeL Internals: New values” For example, /***ó****thentic code signer*/ Authenticode Signer has a DominateMask equals 2. It means that bit 1 {pause} is enabled, which corresponds to the Authenticode. Therefore, processes with /***ó****thentic code signer*/ Authenticode signer can access only /***ó****thentic code processes*/ Authenticode processes.	
Slide 70 “PeePeeL Internals: New values” We can see that LSA has DominateMask equals 110. Now bits 3 and 8 are enabled. Therefore, LSA processes can access LSA and SignerApp processes.	
Slide 71 “PeePeeL Internals: New values” WinSystem is a very interesting case. Processes with this signer level can access all processes because all bits are set. Malware can use this information to access any processes, without regarding their protection levels. Let’s see how it can happen.	
Slide 72 “Episode 6: SRM and PPL are playing together and losing” {Pause}-{Music}	
Slide 73 “Attacks on PeePeeL” To access the protected process the malware has to bypass two {*pause*} security access checks.	

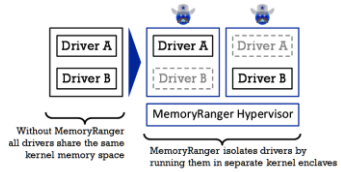
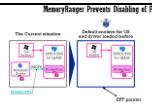
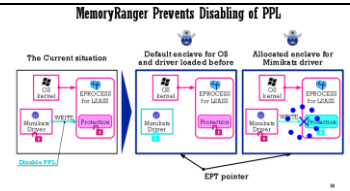
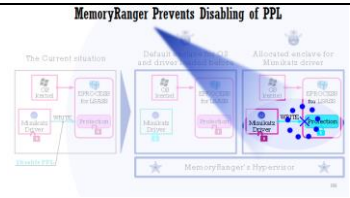
Slide 74 "Attacks on PeePeeL" First, the function verifies Access Token and Security Descriptor. Attackers can bypass the first check ..	
Slide 75 "Attacks on PeePeeL" by enabling the debug privilege. The first check is done. Now the OpenProcess is performing the second check, by comparing ..	
Slide 76 "Attacks on PeePeeL" the Protection levels of the caller and the target processes. OS does not provide any API routine to modify the Protection level.	
Slide 77 "Attacks on PeePeeL" But attackers can illegally update the Protection level to bypass PPL and finally get access to the sensitive data. Let's see what is going to happen.	
Slide 78 "Episode 7: Attacks on PPL: Overview" {Music}-{Pause}	
Slide 79 "Attacks on PeePeeL" To modify the kernel memory,	

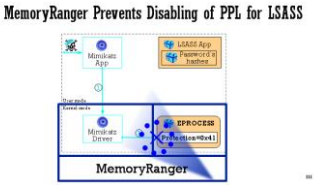
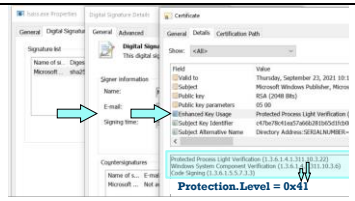
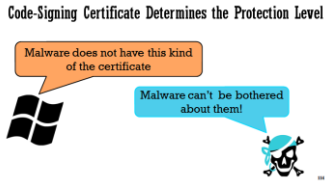
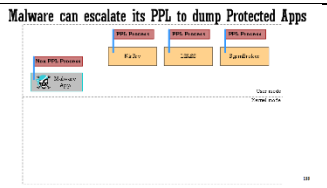
Slide 80 "Attacks on PeePeeL" an attacker has to use a kernel driver.	
Slide 81 "Attacks on PeePeeL" The attacker can reset or clear the Protection field for the target process, which disables PPL to this process.	
Slide 82 "Attacks on PeePeeL" After that, the attacker can access the process memory easily, because PeePeeL has been disabled.	
Slide 83 "Attacks on PeePeeL" At the same time, the attacker can illegally enable PeePeeL for his own malicious process by setting the Protection level.	
Slide 84 "Attacks on PeePeeL" Now, PeePeeL is protecting the **malware** process.	

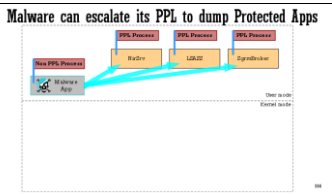
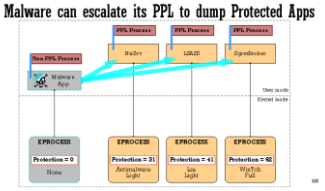
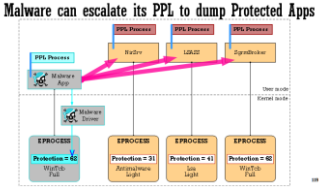
Slide 85 “Attacks on PeePeeL” Both these kernel data modifications never trigger any security features, like PatchGuard. Probably, Windows developers didn’t take into account this attack on PeePeeL. But they **should’ve**. (~~But they **should** have **taken** them.~~) Let’s see some examples of the attacks on PeePeeL.	
Slide 86 “Attacks on PeePeeL” To perform this kind of attacks, intruders have to use kernel drivers.	
Slide 87 “Attacks on PeePeeL” • They can exploit vulnerable drivers;	
Slide 88 “Attacks on PeePeeL” • They can use specially crafted drivers;	
Slide 89 “Attacks on PeePeeL” • They can apply even a general-purpose hacker’s toolkit, such as Blackbone and even Mimikatz. Let me show you an example with Mimikatz.	
Slide 90 “Episode 7: Mimikatz disables PPL for LSASS to dump NTLM hashes” {Music}-{Pause}	
Slide 91 “Mimikatz” You know, Mimikatz is an open-source toolkit designed to make some experiments with Windows Security.	
Slide 92 “Mimikatz” An attacker can use Mimikatz to extract users’ credentials from LSASS twice.	
Slide 93 “Mimikatz”	

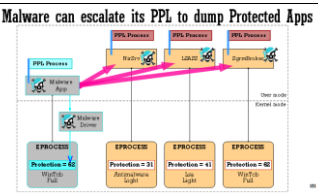
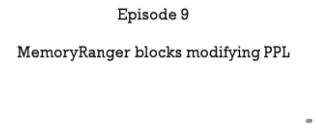
First, an attacker adds the debug privilege for Mimikatz App and after that, he tries to extract the password hashes. We will see if he gains the access.	
Slide 94 "Mimikatz" If he fails, he will load a driver,	
Slide 95 "Mimikatz" which disables PeePeeL.	
Slide 96 "Mimikatz" After that, he repeats the commands to extract hashes,	
Slide 97 "Mimikatz" and finally, he uses a special tool called hashcat to recover user's passwords from the leaked hash. Let's see how it can happen.	
Slide 98 "Demo 1: Attack on PeePeeL" Let's check the Windows version. We've got the final one. An attacker is **launching** Mimikatz app. This app is loaded. He is copying the commands to add privilege and extract hashes. And look – he has failed. PeePeeL prevents illegal access to the protected process. Okay{p}, but, the attacker is trying again {p} using more commands in order to disable PeePeeL and extract hashes. Look – the hash has been gained. The attacker is copying the gained hash to the hashcat's config file. Now, the hashcat is ready to start. The attacker is **launching** cmd to load hashcat.	

The hashcat is starting **to crack** the password hash. Let's wait until the hash is cracked. It usually takes a while, and in this case, it is about {p} seven minutes. The password "honeypot" has been retrieved. Is the password correct? The attacker is checking it. Oh no! The gained password is correct. The OS is in danger.	
Slide 99 "Nutshell: MemoryRanger Prevent PeePeeL disabling" Is it possible to prevent this kind of attacks? Yes, it is /itís/.	
Slide 100 "Episode 8: MemoryRanger blocks Mimikatz" {Music}-{Pause}	
Slide 101 "Nutshell: MemoryRanger Prevent PeePeeL disabling" Currently, all kernel drivers share the same memory space. If we are able to prevent illegal access to the Protection field of ERPOCESS structure, we can protect PeePeeL. The **ke-e-y** feature of PPL is that only Windows kernel needs to access the Protection field, {pause} all other access attempts can and must be {1 sec} blocked.	
Slide 102 "Nutshell: MemoryRanger Prevent PeePeeL disabling" How to prevent this access? I suppose that my MemoryRanger {*pause for the sound on the next slide*}	
Slide 103 "MemoryRanger" is the solution to control access in kernel memory.	

Slide 104 "MemoryRanger" My MemoryRanger, it is a **hypervísor-básed** {small-pause} **software {pause}** designed to block kernel-mode attacks. It can **/træp/** trap the loading of new drivers and move them to the {slow} isolated kernel /ˈɛnkleɪvs/ énclaves in run-time with different memory access restrictions.	
Slide 105 "MemoryRanger blocks PPL disabling" To prevent any illegal access to the sensitive data,	
Slide 106 "MemoryRanger blocks PPL disabling" My MemoryRanger must be loaded first.	
Slide 107 "MemoryRanger blocks PPL disabling" It allocates the default énclave for the OS and previously loaded drivers.	
Slide 108 "MemoryRanger blocks PPL disabling" MemoryRanger can trap the loading of Mimikatz driver and move it to the separate énclave. This enclave includes only Mimikatz driver and the limited number of OS drivers. The Protection field of LSASS will be excluded from this enclave.	
Slide 109 "MemoryRanger blocks PPL disabling" This **sche-e-me** helps to prevent disabling PeePeeL mechanism by trapping and blocking illegal access to the Protection field. Let's see how it works.	
Slide 110 "Demo2: MemoryRanger prevents Hijacking Handle Table" Let's check the Windows version. We've got the final one. The MemoryRanger hyper**v**isor is loaded first. Now, an attacker is **launching** Mimikatz App. This app is loaded. The attacker is copying the commands to add privilege and extract hashes. And look – He has failed. PeePeeL prevents illegal access to the protected process.	

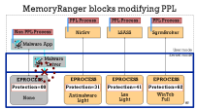
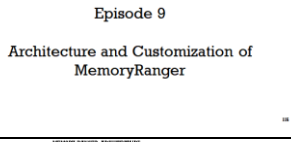
Okay{p}, but the attacker is trying again {p} using more commands to disable PeePeeL and extract hashes. Look – He gets nothing. He has failed again. MemoryRanger has prevented disabling of PeePeeL. Thanks to MemoryRanger the OS and user's data are protected.	
Slide 111 "Episode 9: Malware escalates its own PPL to attack protected processes" MemoryRanger prevents modifying of Protection Level for LSASS. Well, but what about escalating PPL level for malware?	
Slide 112 "Episode 10: Malware escalates its own PPL to attack protected processes" {music}-{pause}	
Slide 113 "MemoryRanger" As we know the OS runs the process as protected only if its image file has a special digital certificate. Here is the certificate for LSASS process and corresponding Protection Level.	
Slide 114 "MemoryRanger" Malware doesn't have this certificate, and it doesn't care. Malware can escalate the Protection level in run time to access the protected processes, let's see how.	
Slide 115 "MemoryRanger" Here we have a malware app and three system processes	

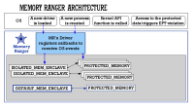
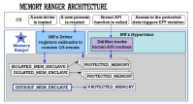
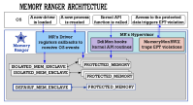
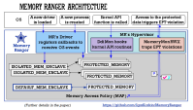
Slide 116 “MemoryRanger” The attacker wants to dump these processes, but PPL prevents these attempts.	
Slide 117 “MemoryRanger” Here we have corresponding EPROCESS structures.	
Slide 118 “MemoryRanger” Malware can load its driver to modify the Protection level for its own malware process and dump protected memory. Let’s see if it causes a PatchGuard reaction.	
Slide 119 “Demo3: Attacker’s App Escalates Privileges” Let’s check the Windows version. We’ve got a newest Windows 11. An attacker is **launching** its app, which loads a driver. Process Hacker is launched in order to see the Protection Levels. Windows built-in processes are protected by PPL. But attacker’s app is not protected, which is expected. The attacker is dumping **Realtime Inspection Service**, and he fails. OS prevents illegal access. The attacker is setting debug privilege and setting the Protection Level, which is 31 (thirty one). Let’s check the Protection Levels again and relaunch the Process Hacker.	

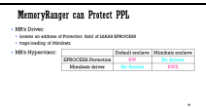
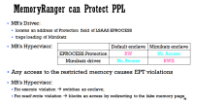
Now the attacker App is protected. The attacker is dumping Realtime Inspection Service again, and he succeeds, the dump file is here. The attacker is focusing on LSASS. The attacker is dumping this process, and he fails. OS prevents illegal access. The attacker is escalating the Protection Level using 62 (sixty-two) value. Let's check the Protection Levels and relaunch the Process Hacker once more. The attacker's app is protected with increased /**encreSt**/ Protection Level. The attacker is dumping LSASS again, and he succeeds, the dump file is here. The attacker is focusing on SgrmBroker. The attacker is dumping this process, and he succeeds again, the dump file is here. Let's wait for Patch Guard reaction, which is designed to prevent any illegal memory modifications. 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 is in danger.	
Slide 120 "MemoryRanger" We have seen that attacker has successfully dumped protected processes without triggering Windows security features. Let me show you how MemoryRanger can prevent this attack.	
Slide 121 "Episode 10: MemoryRanger blocks modifying PPL" {music}-{pause}	

Slide 122 “MemoryRanger” MemoryRanger is launched first	
Slide 123 “MemoryRanger” and it moves all EPROCESS structures into the separate enclave. MemoryRanger can trap the launching of the malware app	
Slide 124 “MemoryRanger” and moves its EPROCESS into this enclave. MemoryRanger intercepts the launching of the malware driver	
Slide 125 “MemoryRanger” and moves it to the separate enclave. Let me show how this scheme helps to block PPL escalation.	
Slide 126 “Demo4: MemoryRanger Prevents Attacker’s App from Modifying the PPL Level” Let’s check the Windows version. We’ve got a newest Windows 11. First, we launch a DebugView to see the kernel debug prints. MemoryRanger hypervisor is loaded. An attacker is **launching** its app, which loads a driver. MemoryRanger traps creation of attacker’s app and restricts memory access to its Protection Level. Memory Ranger also traps the loading of attacker’s driver and isolates this driver. Process Hacker is launched in order to see the Protection Levels. Windows built-in processes are protected by PPL. Attacker’s app is non-protected. The attacker is dumping **Realtime Inspection Service**, and he fails. OS prevents illegal access.	

The attacker is setting debug privilege and is setting the Protection Level, which is 31 (thirty one). Let's check the Protection Level again and relaunch the Process Hacker. The attacker's app is still non-protected. Let's see the debug output. MemoryRanger traps an access attempt to the sensitive memory and {*pause*} redirects it to the fake page. The attacker is dumping Realtime Inspection Service once more. He dumps nothing. He fails again. OS prevents illegal access. The attacker is dumping LSASS and he fails. OS prevents illegal access again. The attacker is escalating the Protection Level using 62 (sixty-two) value. Let's check the Protection Levels and relaunch the Process Hacker once more. The attacker's app is still non-protected. MemoryRanger traps this illegal access and blocks it as well. The attacker is trying to dump LSASS again. He dumps nothing. He fails again. The attacker is dumping SgrmBroker and he fails once more. The OS is protected! Thanks to the MemoryRanger the Protected Processes are actually Protected now.	

Slide 127 “MemoryRanger” We’ve just seen that MemoryRanger can successfully block both attacks on PPL: disabling PPL and escalating PPL level. Now let’s move on to the MemoryRanger architecture.	
Slide 128 “Episode 11: Architecture and Customization of MemoryRanger” {Pause}-{Music}	
Slide 129 “MemoryRanger overview” MemoryRanger processes the following four events:	
Slide 130 “MemoryRanger overview” loading new drivers	
Slide 131 “MemoryRanger overview” launching new processes	
Slide 132 “MemoryRanger overview” calling kernel API functions and	
Slide 133 “MemoryRanger overview” processing memory access violations, which occur due to the access to the memory with restricted access.	
Slide 134 “MemoryRanger overview” After loading, MemoryRanger allocates a default enclave for the OS and previously loaded drivers.	
Slide 135 “MemoryRanger overview” • MemoryRanger traps loading drivers and creates a separate memory énclave for each of them. These enclaves have different memory access configurations. The information about each memory configuration is saved into ISOLATE_MEM_ENCLAVE structure.	

Slide 136 “MemoryRanger overview” • MemoryRanger can also trap launching new users’ apps. It helps to locate sensitive kernel data and block access to them by modifying PROTECTED_MEMORY structures.	
Slide 137 “MemoryRanger overview” MemoryRanger can trap any kernel API routine. DdiMon component provides these hidden hooks using EeePeeTee. It helps to locate sensitive data, allocated dynamically. Any access to the restricted memory areas causes EPT violations.	
Slide 138 “MemoryRanger overview” MemoryMonRWX is designed to process all these violations. To process execute violations MemoryMonRWX changes enclaves, so another driver continues execution inside its own enclave. Read or write violations mean that a driver is accessing restricted memory data. This case is redirected to the Memory Access Policy (MAP), which	
Slide 139 “MemoryRanger overview” decides whether to block or allow this access. MemoryRanger is a proof-of-concept which can monitor and block access to the kernel data and code.	
Slide 140 “MemoryRanger: Previous Research” Here are some of my research projects. That demonstrate various types of MemoryRanger customization to prevent different types of kernel attacks.	
Slide 141 “MemoryRanger: Customized” To protect PPL MemoryRanger has been customized in the following way:	
Slide 142 “MemoryRanger: Customized” MemoryRanger’s driver {p} lócates the address of the Protection field of EPROCESS.	
Slide 143 “MemoryRanger: Customized” and traps loading of kernel drivers.	
Slide 144 “MemoryRanger: Customized” MemoryRanger’s hypervisor provides various memory access restrictions for the default enclave and	

Slide 145 “MemoryRanger: Customized” for the enclaves allocated for the kernel drivers.	
Slide 146 “MemoryRanger: Customized” MemoryRanger’s hypervisor blocks illegal access to the Protection field. This is the way how MemoryRanger protects PeePeeL.	
Slide 147 “Conclusion” Let me /ˈ**ri**:**kap**/ ré-cáp very briefly on what we have discussed so far. First of all, Windows Security Model is based on User’s Access Tokens and Object’s Security Descriptors. This model does not restrict processes running with the debug privilege. To fill this gap a new mechanism called Protected Process Light has been released. But it can be disabled just by modifying a byte in kernel memory.	
Slide 148 “Thank you” I have presented my MemoryRanger, which can prevent this kind of attacks by restricting memory access to the Protection fields. Thank you!