Protected Process Light is not Protected: MemoryRanger Fills The Gap Again

Igor Korkin
Independent Researcher
Moscow, Russia
igor.korkin@gmail.com

***Abstract*— Windows OS issued a newly updated security mechanism to prevent illegal access to the memory of critical processes as well as for Digital Rights Management (DRM) requirements. It is Protected Process Light (PPL). Intruders can disable PPL to access the memory content of protected processes using a kernel driver. Also, they can illegally enable PPL for the malware apps to provide self-protection and access memory of protected processes, without disabling their PPL. PatchGuard does not check the integrity of PPL. This kind of attack is crucial for OS security and has to be prevented. This paper presents some undocumented internals of PPL during the creation of the protected process as well as accessing the protected process memory to analyze how the PPL can be tampered with. In this contribution, the hypervisor-based solution called MemoryRanger is applied to prevent such type of kernel attacks on PPL. MemoryRanger can prevent both types of attacks on PPL: disabling and enabling PPL in run time. MemoryRanger has been successfully tested on the recent Windows 10, version 20H2 Build 19042.631 x64.**

Keywords— content protection, OS Security, attacks on data, Protected Process Light, security enhancement for Windows OS.

I. INTRODUCTION

Protection of process memory is vital for various areas, including the Digital Rights Management (DRM) market, Game and Anti-Virus industries as well as credentials protection.

To fulfill these requirements Windows expanded its security model [1] and introduced the protected process model (PP) to provide increased protection for high-value content. The model provides several new security features including restriction of read and write access to protected process memory from other processes running even with administrative privileges. In order to be loaded as PP the image file on disk has to be signed using Microsoft certificate [2, 3, 4].

Protected Process Light (PPL) is an extension to this model. PPL gives an additional dimension to the protection using `Protection Type` and `Signer` values. Various combinations of these two parameters provide different protection levels among PPL-protected processes [2, 5]. Protected processes have the following constraints: a typical process cannot access the virtual memory of a protected process and inject a thread into a protected process [6].

PPL is initially designed to protect Windows built-in apps and currently it allows anti-malware user-mode services to be launched as a protected service. Anti-malware vendors with the help of Early Launch Anti-Malware (ELAM) driver can launch their anti-malware services as protected ones. Now, all critical Windows processes are protected using PPL, including Local Security Authority Subsystem Service (LSASS), Windows Defender Process.

Authors admit that with the help of kernel-mode malware, the PPL protection can be disabled by clearing the flag which indicates the protected process.

The PPL feature is implemented using a new PS_PROTECTION byte added in the EPROCESS structure. This byte is set for PPL processes and checked in Windows API routines. Malware driver can disable the PPL protection by clearing this byte, which stops restricted access to this process. Such DKOM attacks are critical for platform security.

Microsoft experts consider preventing such type of attacks by prohibiting the digital signing of malicious code and recognize such attacks using Kernel Patch Protection (KPP/PatchGuard) and Protected Environment Authentication and Authorization Export Driver (PEAuth).

However, research papers and experimental results prove that these measures are not enough and PPL can be easily disabled even on the newest Windows 10.

Benjamin Delpy creates Mimikatz, a console application that loads a kernel-mode driver, and can be used to demonstrate the weakness of the Windows authentication subsystem [7]. Mimikatz is not malicious software, but it can disable PPL by patching the field Protection PS_PROTECTION. This patching occurs transparently for users and for the OS, without causing any security reaction, such as a BSOD with a bugcheck critical_structure_corruption (0x109).

Windows Defender process (MsMpEng.exe) protected by PPL can be terminated after resetting the process protection field using Mimikatz. The corresponding attacks have been shown by [8, 9]. Boonen implemented a similar attack using his own tool called AquaWrench to disable PPL for Windows Defender [10].

The security issues with disabling PPL are widely discussed in the video game industry because PPL plays an important role in game cheating engines [11].

Local Security Authority Subsystem Service (LSASS) process is protected by PPL to prevent unauthorized access to users' password hashes stored in its memory. Mimikatz can disable PPL for LSASS, and after that it can extract password

hashes from its memory. The users' passwords from the stolen hashes can be gained by running hashcat [12, 13].

Microsoft security experts issued Microsoft Defender Credential Guard (WDCG), which is designed to prevent extracting credentials from LSASS. This protection mechanism sometimes cannot be enabled and "in these cases, attackers can use tools like Mimikatz to scrape cleartext passwords and NTLM hashes from LSASS" [14].

Finally, Windows security experts add Mimikatz app and driver to the malware list. This measure helps to block Mimikatz itself but does not prevent this type of attacks on PPL globally. Also, Mimikatz can be obfuscated to prevent its detection [15].

A. Examples of Disabling PPL

Attackers can load their own kernel drivers to patch the PPL flag, for example, Mimikatz by Delpy [7], Blackbone by DarthTon [11]. Disabling PPL can also be achieved by exploiting a vulnerable signed driver:

- MSI driver by RedCursorSecurityConsulting [16]
- CPU-Z bug FireF0X [17];
- MalwareFox by Harakirinox [18];
- Gigabyte driver by Bui [19, 20].

Using a similar manipulation, other non-protected processes, even malicious ones, can be elevated up to a PPL-protected level, even without a special Windows signature.

B. Examples of Escalating PPL level

Intruders can escalate PPL level for the malware app and finally access memory of the protected process, without disabling their PPL. These attacks can be implemented in the following way:

- loading a kernel driver [21];
- exploiting user-mode vulnerabilities [22, 23].

Windows does provide any API to modify PPL level. Experimental results show that these memory manipulations are not registered by Windows security monitoring features [24]. Therefore, it is crucial for the OS security to prevent this kind of attacks. Any attempts to modify the PPL level have to detected as a misuse of the system.

C. Problem Statement

Attackers can modify some areas of Windows kernel memory without triggering any security alerts such as BSOD from PatchGuard. The current paper considers attacks on PPL, which are part of the attacks on dynamically allocated data.

Intruders can disable PPL for the critical processes by clearing the corresponding EPROCESS structures, which helps them to access the sensitive process memory. Apart from that, they can escalate PPL for the malware process by modifying its EPROCESS structure; thus, restricting access to the malware process and also granting access to the memory of PPL process memory.

This paper shows a trustworthy solution, which guarantees the integrity of the PPL protection against attacks based on modifying the PS_PROTECTION byte. Windows 10, version 20H2 x64 is under the test.

This research presents MemoryRanger, a virtualization-based solution, which has been updated to prevent DKOM attacks on PPL. The key feature of MemoryRanger is its ability to run newly loaded drivers in isolated kernel memory areas, called enclaves, by leveraging VT-x and EPT.

D. Limitations

Lagrasta [25] shows how to extract password hashes by hooking MsvpPasswordValidate in NtlmShared.dll, which is out of the scope of this paper. Ciholas, Such, Marnerides, Green, Zhang, & Roedig [26] reveal how to gain handles for the protected process including anti-malware and anti-cheat protection solutions. The attack is out of the scope of this paper. Forshaw [27] injects arbitrary code into a PPL using a feature of the COM technology. The attack is out of the scope of this paper. Another technique it to obtain credentials by dumping the content of the LSASS process using the legitimate comsvcs.dll library [28]. The attack is out of the scope of this paper.

The remainder of the paper proceeds as follows.

Section 2 provides the internals of PPL: its configuration; the analysis of triggering PPL during the opening process and why the process protection can be disabled and enabled illegally.

Section 3 explains Mimikatz as an example of bypassing PPL and extract password hashes. Analysis of the existing approaches and tools designed to reveal and prevent Mimikatz is in Section 3.

Section 4 presents MemoryRanger, a hypervisor-based solution designed to protect kernel memory, and how to use MemoryRanger to protect PPL for the LSASS process from being disabled and finally prevents users' passwords hashes from being leaked by Mimikatz.

Section 5 and Section 6 focus on the main conclusions and further research directions respectively.

II. PPL Internals

This section provides the details regarding PPL configuring, some cases of using PPL, PPL internals, including the updates of EPROCESS and OpenProcess routine algorithm. Finally, it is shown why PPL is not protected.

A. Introduction to the Protected Process Light

Windows security experts introduced a Protected Process (PP) to host Digital Rights Management (DRM) content and prevent read and write access to the content of protected processes even from admin-level non-protected processes.

Protected Process Light (PPL) is a re-design of the (PP) which creates an access hierarchy for protected processes using three elements: protected signer, protected type, and auditing mode [2, 29]. Achilles [30] gives all the details about their combinations.

PPL is used to protect memory for critical OS apps as well as for various security vendors: Bitdefender [31], Cisco [32], ESET [33], Kaspesky [34], SolarWinds [35], McAfee [36].

Windows introduced the Early Launch Anti-Malware (ELAM) mechanism, which makes it possible to register a kernel-mode driver that is guaranteed to execute very early in the boot process and launch an anti-malware service as a protected service. Finally, PPL will defend the anti-malware services against malicious attacks.

PPL limits non-protected processes activity against the protected ones:

- non-protected processes are not able to inject threads and they are not allowed to write into the virtual memory of the protected process;
- non-protected processes are neither able to debug an active protected process nor to duplicate a handle from a protected process. But debugging any anti-malware protected processes is allowed using a kernel debugger.

In addition, any non-Windows DLLs which get loaded into the protected process must be also signed with the same certificates.

A newly introduced Secure Event Tracing for Windows (Secure ETW) can be consumed only by PPL processes, while other non-protected processes cannot listen to these events [37].

It is crucial to analyze how well PPL is protected.

B. Activation and Checking PPL

PPL is activated automatically but for some cases, for example, to activate Local Security Authority (LSA) protection for the LSASS process the following steps have to be taken [38].

There are several Windows API routines to check the process protection level: ZwQueryInformationProcess with ProcessProtectionInformation flag, PsGetProcessProtection, PsIsProtectedProcess, and PsIsProtectedProcessLight [39, 40].

These functions gain information by reading the content of the EPROCESS structure, the details are in the next section.

C. PPL Internals

This section includes some internals of PPL.

EPROCESS Updates. Field Protection PS_PROTECTION has been added to the EPROCESS structure to flag PPL processes.

PS_PROTECTION Protection is a one-byte structure, which includes three members *Type*, *Audit*, and *Signer*. The *Type* field contains the type of protected process. The *Audit* field is reserved, and the *Signer* field contains the protected process signer [39].

With disabled PPL the Protection byte is zero, see Fig. 1 *a)*. After activating PPL this byte is non-zero. For example, the LSASS process is running with the following protection values, see Fig. 1 *b)*. The Type equals 0x1 (PsProtectedTypeProtectedLight), the Signer field has 0x4 value, which corresponds to PsProtectedSignerLsa [39, 41, 42].

Process Manager API routines have been updated to consider the Protection field in the EPROCESS structure. The details of the creation and opening process will be shown further.

As a result, the virtual memory of the PPL process is protected from being accessed by all non-protected applications, even if they have a debug privilege [43].

Creating the PPL process. The protected process creation has some features. The first is that the binary must have a special signature, which is provided by Microsoft but currently available only for Microsoft binaries.

Processes can be created as protected during startup (PspInitPhase) as well as in run time by calling NtCreateProcess, see Fig. 2. All these functions use the same Process Manager routine SepSetTrustLevelForProcessToken to update the field Protection PS_PROTECTION [44].

Windows provides a ChangeServiceConfig2 routine with SERVICE_CONFIG_LAUNCH_PROTECTED flag to run services as PPL using ELAM, the service protection type is stored in the SERVICE_LAUNCH_PROTECTED_INFO structure [29, 45, 46].

The next section describes how to open protected processes.

Accessing Process Memory. To access a process memory from any other processes the following sequence of functions has to be called:

1) OpenProcess
2) ReadProcessMemory\WriteProcessMemory
3) CloseHandle

OpenProcess routine uses the process ID as one of the input parameters and returns an open handle to the specified process. This handle is used in ReadProcessMemory (WriteProcessMemory) routines to read (write) the memory content of the opened process. CloseHandle routine is designed to finish work with the process and resource deallocation.

OpenProcess Internals for disabled PPL. According to the MSDN, the OpenProcess function checks access rights using the security descriptor for the caller process. At the same time, "if the caller has enabled the SeDebugPrivilege privilege, the requested access is granted regardless of the contents of the Security Descriptor". In the recent Windows 10, this check is implemented in nt!PsOpenProcess routine, see Fig. 3.

To enable SeDebugPrivilege the caller can use the RtlAdjustPrivilege(SE_DEBUG_PRIVILEGE) [43].

Enabling this privilege makes it possible to get a process handle, but for the PPL processes, an additional security check is implemented. This new check will be covered in the next section.

OpenProcess Internals for enabled PPL. OpenProcess routine implements an additional check for accessing protected processes, but for non-protected processes, this check is skipped.

During opening the process OpenProcess routine checks whether the protected process or non-protected process is going to be opened. For this purpose, the OpenProcess routine reads the value of the field Protection PS_PROTECTION [44]. This value is used by nt!RtlTestProtectedAccess and nt!PspCheckForInvalidAccessByProtection routines to check caller privileges, see Fig. 4.

If the caller does not have enough privileges, the OpenProcess routine returns STATUS_ACCESS_DENIED.

Research papers and experimental results [7, 8, 9, 10, 11, 16, 17, 18, 19, 20, 48] prove that PPL can be disabled even on the newest Windows 10 by resetting the corresponding Protection field to zero. Now OpenProcess routine successfully returns an open handle to the protected process, recognizing it as non-protected by reading the zero value.

D. Protected Process Light is not Protected

PPL security feature is based only on checking the Protection field from EPROCESS during OpenProcess call.

At the same time, this field can be modified in order to disable PPL for OS built-in critical processes and vice versa to elevate a non-protected process to a protected one.

```
Protection    struct _PS_PROTECTION
  Level       0x00 ''
  Type        0x00 ''
  Audit       0x00 ''
  Signer      0x00 ''
```

a)

```
Protection    struct _PS_PROTECTION
  Level       0x41 'A'
  Type        0x01 ''
  Audit       0x00 ''
  Signer      0x04 ''
```

b)

Fig. 1. The content of the EPROCESS.Protection _PS_PROTECTION structure for the LSSAS process: *a)* with disabled PPL and *b)* with enabled PPL: Level equals 1 (PsProtectedTypeProtectedLight) and Signer equals 4 (PsProtectedSignerLsa)

Attackers can easily kill AV solutions and steal users' credentials from LSASS memory by disabling PPL protection. Also, they can protect their malicious apps using PPL.

The leak of the protection of PPL has been proved by research papers and experimental results. The authors state that by clearing the Protection flag in the EPROCESS structure the corresponding application loses its "protection process" status transparently without triggering any security alerts such as BSOD from PatchGuard [7, 8, 9, 10, 11, 16, 17, 18, 19, 20, 48].

This patching of PS_PROTECTION value from the EPROCESS is critical for the OS protection and it must be controlled and blocked.

The next section describes Mimikatz as an example, which can disable PPL and the existing ways to fight against Mimikatz.

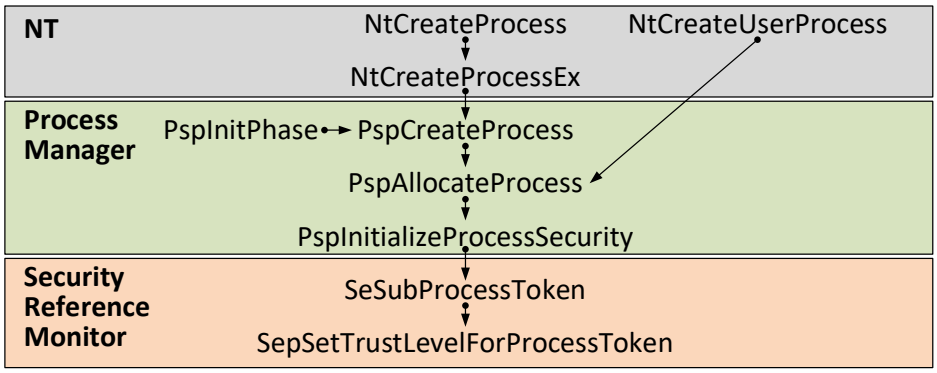


Fig. 2. The Internals of CreateProcess

```
NTSTATUS PsOpenProcess()
{
   NTSTATUS Status;
   SeCaptureSubjectContextEx()
   SepCreateAccessStateFromSubjectContext()
   if (SePrivilegeCheck(SeDebugPrivilege,PreviousMode))
      {...}
   PsLookupProcessByProcessId()
   status = ObOpenObjectByPointer()
   return status;
}
```

a)

```
Stack fragment:

   nt!PsOpenProcess
   nt!NtOpenProcess
   nt!KiSystemServiceCopyEnd
   ntdll!NtOpenProcess
   KERNELBASE!OpenProcess
```

b)

Fig. 3. The Internals of OpenProcess implementing Privilege checking: a) the pseudocode of nt!PsOpenProcess and b) the corresponding call stack fragment

```
NTSTATUS PspProcessOpen(EPROCESS Process)
{
   BYTE level = Process->Protection.Level;
   PspIsParentProcess();
   PsTestProtectedProcessIncompatibility();
   if (!level)
   {
      ...
      return STATUS_ACCESS_DENIED;
   }
   return STATUS_SUCCESS;
}
```

a)

```
Stack fragment:
   nt!RtlTestProtectedAccess
   nt!PspCheckForInvalidAccessByProtection
   nt!PsTestProtectedProcessIncompatibility
   nt!PspProcessOpen
   nt!ObpIncrementHandleCountEx
   nt!ObpCreateHandle
   nt!ObOpenObjectByPointer
   nt!PsOpenProcess
   nt!NtOpenProcess
   nt!KiSystemServiceCopyEnd
   ntdll!NtOpenProcess
   KERNELBASE!OpenProcess
```

b)

Fig. 4. The Internals of OpenProcess implementing PPL check: a) the pseudocode of nt! PspProcessOpen and b) the corresponding call stack fragment

III. Mimikatz Can Disable PPL

This section describes how to apply Mimikatz to extract users' passwords with disabled (enabled) PPL for LSASS and the analysis of the existing approaches fighting Mimikatz.

A. *Mimikatz can access LSASS process memory with disabled and enabled PPL*

Mimikatz can extract various types of sensitive data from memory [7]. An overview of Mimikatz features and its internals are provided by Mulder [49], Hand [50], Chester [51, 52].

To dump users hashes with disabled PPL attackers can run the following two Mimikatz commands:

1. privilege::debug
2. lsadump::lsa /inject

The first command adds SeDebugPrivilege for the Mimikatz process, while the second command extracts the password hashes. The details of the second step are out of the scope of this research and discussed by Patil and Meshram [53].

After enabling PPL for LSASS process these two commands fail to extract users' credentials.

Mimikatz has been updated with a new feature, which can disable PPL by patching the EPROCESS structure. Now attackers have to run the following four commands, see Fig. 5:

1. !+
2. !processprotect /process:lsass.exe /remove
3. privilege::debug
4. lsadump::lsa /inject

The first command loads a Mimikatz driver, while the second one disables PPL for LSASS by zeroing the Protection field, see Fig. 6. The last two commands are the same as the previous case. Finally, password hashes are dumped.

A similar attackers' technique is based on accessing memory of the protected process without disabling its PPL. Intruders can maximize the PPL level for the attackers' application, e.g., Mimikatz app. Experimental results prove that this technique works well without triggering any OS security mechanisms. To access LSASS process running with 0x41 PPL level, see Fig. 1., the Protection level of Mimikatz application has to be 0x41 or higher. Finally, Mimikatz app can access LSASS memory, without disabling PPL for LSASS.

The next part is focused on the analysis of how to reveal Mimikatz and prevent disabling PPL.

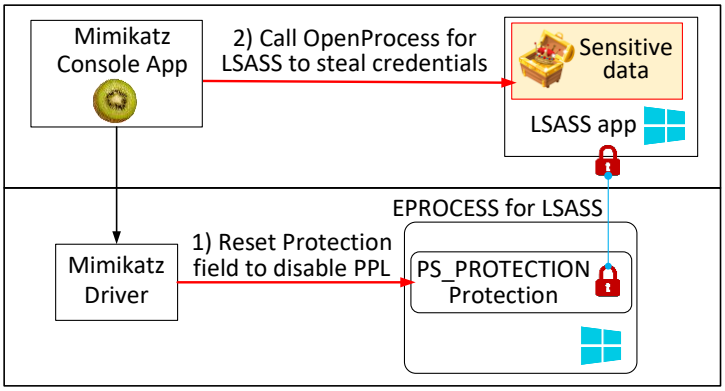
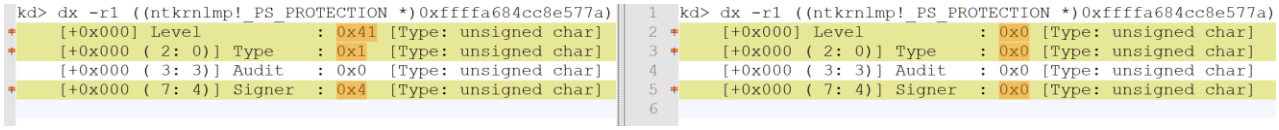


Fig. 5. Mimikatz disables PPL protection for the LSASS process in order to acquire the password hashes stored in LSASS process memory



a) *b)*

Fig. 6. The content of the EPROCESS.Protection _PS_PROTECTION structure for the LSSAS.exe process with enabled PPL: a) before and b) after patching by Mimikatz driver. As a result, PPL for LSASS has been disabled.

B. *Analysis of Existing Approaches Fighting Mimikatz*

This section provides the analysis of the ways, which can be applied to detect or prevent Mimikatz [54, 55].

1) *Mimikatz as Malware?*

A straightforward approach to prevent Mimikatz is to add its application and driver to the blacklist and detect them as malware. Mimikatz has been detected as malicious software by more than 70% of AV products, while many experts believe that it is not a virus [56]. The debate is continued.

2) *Via Windows PowerShell*

Windows standard tools can be applied to reveal the Mimikatz attack consequences [57]. After disabling PPL, the corresponding PPL attributes for LSASS are not returned. PowerShell can be used to check this feature, but the author admits that this is a poor man's solution.

3) *Revoke SeDebugPrivilege*

Another approach is based on revoking administrator debug privileges (SeDebugPrivilege) by configuring a group policy [55]. A process with this privilege can open almost every process [56], while this privilege it is very rarely used [57]. Malware with local admin rights can restore the config [58].

4) *Disable WDigest protocol*

One more approach prevents storing users' credentials in memory [51]. Activation of this feature requires adding the registry key [58, 59]. Finally, attackers fail to retrieve the credentials. Malware with local administrator privileges can restore the configuration [60, 61].

5) *Enable Restricted Admin Mode*

Microsoft update provides a Restricted Admin mode [62, 63]. This mode is disabled by default. To enable it, the registry key has to be added [55]. At the same time, Restricted Admin mode can be disabled by restore the configuration [64].

6) *Windows Defender Credential Guard (WDCG)*

Microsoft issued a Windows Defender Credential Guard (WDCG), which is based on Virtual Secure Mode (VSM). VSM creates a set of Virtual Trust Levels (VTLs), so that processes running in one VTL cannot access the memory of another VTL. VSM supports only two VTLs: VTL0 with a normal kernel and the higher privileged VTL1 with Secure Kernel and trustlets. With enabled VSM, LSASS runs as a trustlet and its memory is protected from any code running in VTL0. However, WDCG is integrated only in Windows 10 Enterprise and Windows Server 2016, while other Windows editions are becoming susceptible to attacks on memory [65, 66].

7) *Conclusion*

We can see that analyzed protection methods cannot reliably protect users' credentials from being accessed by Mimikatz-type attacks.

To access memory of PPL protection process, such as stealing users' credentials from the LSASS, intruders can implement one of the following modifications:

- Malware driver can disable PPL protection by clearing the PS_PROTECTION byte in the EPROCESS structure, which stops restricted access to this process.
- Also, intruders can escalate PPL level for the malware process so that they can access the memory of critical processes without disabling PPL for them.

Mimikatz implemented only the first technique, while the experimental results prove that Mimikatz process with maximized PPL level can access LSASS memory, without disabling PPL for LSASS. Both of these Mimikatz-type attacks are critical for platform security.

To protect process memory, we have to guarantee the integrity of the Protection field of the EPROCESS structures for all processes. Any attempts to modify these fields can be used as indicator of compromise (IoC), because Windows does provide any documented API to modify PPL level in run time. At the same time revealing the abnormal values of PPL levels can be used as a attack footprint during memory forensics.

The next section demonstrates how MemoryRanger can be updated to isolate the mentioned field of EPROCESS in order to prevent attacks on PPL.

IV. MemoryRanger Provides Trustworthy Protected Process

This section describes the details of how updated MemoryRanger can prevent both kinds of attack of PPL: illegal disabling PPL for the OS critical processes and illegal escalation PPL for the normal processes. The novelty and scope of MemoryRanger will be given.

A. *MemoryRanger Intro*

MemoryRanger (MR) is a software-based platform security solution designed to protect Windows OS kernel data and code from kernel driver attacks. MR has been originally designed by Igor Korkin and presented at several conferences [67, 68, 69].

MR includes two main parts: a kernel-mode driver and a bare-metal hypervisor (type 1 hypervisor).

MR driver registers several callback routines to be notified about various OS events: loading (unloading) drivers, creation (termination) processes. The corresponding MR driver dispatching routines locate the sensitive areas in kernel memory, which have to be protected.

Being a key part of MR, the bare-metal hypervisor leverages Intel hardware-assisted virtualization technology (VT-x) and Extended Page Tables (EPT) feature to protect Windows OS kernel memory. Using EPT we can get an additional or second level of address translation (SLAT). When EPT is enabled, guest physical addresses are translated to host physical addresses by traversing a set of EPT paging structures, which is called a *kernel enclave*. These structures determine the mapping between the guest memory and the host memory. MR can trap read, write, execute access to the memory page by resetting the corresponding access bits on the page. Using EPT violations MR is notified of memory access attempts and it can protect sensitive data by redirecting access to the fake null memory page.

MR can allocate several sets of EPT paging structures with various memory access configurations and, by switching between them, MR organizes drivers' execution in isolated enclaves. MR allocates the default EPT paging structures, called the *default kernel enclave*, for OS kernel and all drivers loaded before. MR allocates a separate set of EPT paging structures, called an *allocated kernel enclave* for each newly loaded driver. MR updates memory access bits in the default enclave and in a newly allocated enclave so that the newly loaded driver executes only in its enclave. MR provides switching between enclaves so that all drivers and OS kernel can be executed.

MR can hook kernel API routines, such as ZwCreateFile, ExAllocatePoolWithTag, and locate the corresponding sensitive data in memory. Windows OS does not provide any built-in facilities to hook such routines and direct hooking will cause a BSOD from PatchGuard.

A recent feature of MR is a special data only enclave, called *data only enclave*, which includes sensitive data and a limited number of OS core drivers to manage them. This enclave makes it possible to protect data from drivers loaded before MR and after it [69]. A key restriction of this enclave is that it is applicable to protect data, which are accessed quite rarely. Storing the frequently accessed data inside such an enclave will cause significant time degradation due to switching between enclaves, which is time-consuming.

MR has been chosen as a basic platform to safeguard PPL from the mentioned DKOM attacks, due to MR facilities to monitor OS events as well as its memory enclave protection.

B. PPL Safeguarding: MemoryRanger Updates

Due to the fact that there is no Windows API to modify the process Protection level in run time, MR has to prevent all write access to the Protection field of EPROCESS structures, without restricting read access. To achieve this, MR driver and MR hypervisor have been updated.

An updated MR driver is able to do the following:

- locate EPROCESS structures for all running processes both protected and non-protected and create the list of addresses called Active Processes List (APL).
- monitor creation (termination) processes, locate the corresponding addresses of EPROCESS structures, and update the APL.
- for each item of APL locate the address of PS_PROTECTION structure, which stores process protection level and signer, and send it to the MR hypervisor.

MR hypervisor's duty is to manage memory enclaves and dispatch the commands from MR driver to restrict and allow access to the sensitive memory areas. MR also implements memory access policy to decide whether or not the restricted access is allowed.

An updated MR hypervisor can do the following:

- allow or restrict access to the corresponding memory area for each enclave after processing an update from MR driver;
- trap memory access violations, decide whether or not this is write access to the PS_PROTECTION structure.

Due to the paging nature of memory, MR hypervisor can restrict access to 4 kilobytes of memory. MR traps any write access to the corresponding memory page and checks whether or not it is write access to the PS_PROTECTION structure.

An updated MR restricts access to the Protection field of EPROCESS structure by means of EPT. MR is notified when someone tries to access the Protection fields using EPT violations. There are two possible ways of implementing protection mechanisms: using a separate enclave for sensitive data and without this enclave.

1) *MR with a separate enclave for sensitive data.*

In this case, MR after its loading allocates two enclaves: a *default kernel enclave* and a new *data-only enclave*, see Fig. 8. MR changes the access restriction bits so that the Protection fields are accessible only in *data only enclave*.

Experimental results show that this approach causes huge performance degradation and blocks the OS, because of the number of switching between two enclaves: the default enclave and the special enclave for sensitive data. Performance assessment was not done due to the fact that OS had been blocked. To conclude, the *data only enclave* feature of MR works well only to protect rarely accessed data.

2) *MR without a special enclave for sensitive data*.

For the second case, MR isolates the memory content with the Protection field only for newly loaded drivers, see Fig. 9. Empirical test results show that in this case MR produces acceptable performance degradation. The details of performance evaluation will be given below.

3) *Performance Evaluation of MemoryRanger*

To evaluate performance degradation of MemoryRanger, the qualitative assessment of CPU performance was made with the help of Super PI tool. This tool measures the time it takes to calculate Pi to a specific number of digits.

The benchmarks assessment was carried out with disabled and with enabled MemoryRanger. The test bed has the following configurations:

- The computing test bed includes the host OS and VMware Workstation, which runs VM OS.
- Dell 7579 laptop with Intel i7-7500U CPU with 4 logical cores and 12 GB RAM is a host hardware platform.
- VM OS has been launched inside VMware using CPU with 2 logical cores and 4 GB RAM.
- Windows 10, version 20H2 Build 19042.631 x64 is used for both Host and VM OS.

The benchmark assessment was made for MR running inside VM OS. The results showed that performance degradation was about 50% (Fig. 7), which is expected for the proof of concept. We surmise that the reason for the higher overhead on this test is caused by the following reasons:

- MR was designed as a proof-of-concept solution to demonstrate the ability of preventing kernel attacks and its performance was not a priority. The internal dispatching algorithms of MR can be improved to speed up its overall performance.
- Limited resources were granted to the VM OS, which made performance degradation worse. Using more powerful test bed could improve the performance results. VMware Workstation emulates VMX feature, which additionally drains CPU resources.

The time overhead is affordable and can be improved.

C. *Testing MemoryRanger*

To test an updated MemoryRanger the following tools were used:

- Mimikatz version 2.2.0 20200918, which illegally disables PPL for LSASS process and shows that users' credentials can be leaked;
- An author's memory attacker driver, which illegally escalates PPL level for non-protected processes.

The corresponding results have been recorded and will be uploaded to author's YouTube channel [70].

D. *MemoryRanger vs. WDCG*

As for comparing WDCG and MemoryRanger, both of them are virtualization-based solutions and use a similar kind of EPT based isolated memory enclaves.

The major difference between extended MemoryRanger and WDCG is that MemoryRanger can prevent expanded scope of attacks on PPL: illegal disabling and illegal enabling PPL, while WDCG is focused only on LSASS memory protection, see Table I.

WDCG is based VSM, which provides a particular case of enclave-based protection with only two memory partitions for normal and secure operations [71], while MemoryRanger implements a *general case* with an infinite number of kernel enclaves. MemoryRanger has been tested before using three [68], four [67], and five separate memory enclaves [69].

The key differences are the following:

- WDCG is designed for the protection of users' credentials and memory of LSASS (LSAiso), without restricting the access to the memory data of other Protected Processes such as anti-malware services, while MemoryRanger is a general-purpose solution, which can protect all running Protected Processes, including LSASS.
- WDCG does not prevent intruders from enabling PPL on their malicious processes to prevent them from being detected, while MemoryRanger can prevent these illegal memory manipulations.

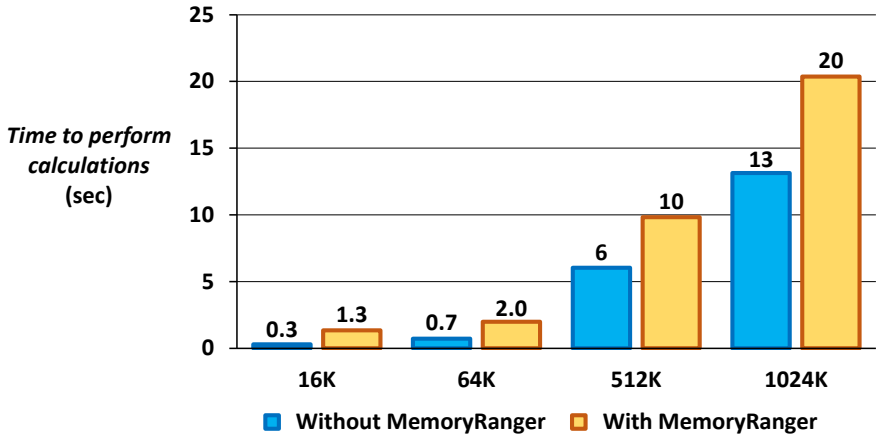


Fig. 7. MemoryRanger benchmark assessment: with enabled MemoryRanger the CPU performance degradation is about 50%

TABLE I.
COMPARISON TABLE OF WDCG AND MEMORYRANGER

Features	WDCG	MemoryRanger
Does it prevent access to the LSASS process memory?	YES	YES
Does it prevent access to the memory of other Protected Processes after disabling their PPL?	NO	YES
Does it prevent non-protected process from enabling PPL for self-protection?	NO	YES
Does it launch on startup?	YES	NO
Does it support various Windows editions?	Enterprise edition only	All editions

E. *MemoryRanger: Novelty and Scope*

MemoryRanger has the following competitive advantages:

- MR is the first solution, which protects PPL from being disabled and even illegally enabled.
- MR can notify users about modifying PPL values, which helps to reveal APT attacks on the early stage.
- MR has been tested on various Windows builds from Windows 7 and it works well on the newest Windows 10, version 20H2 Build 19042.631 x64.

V. CONCLUSION

To sum up I would like to highlight the following:

- Windows issued the Protected Process Light (PPL) to prevent illegal access to the memory of critical processes as well as for DRM requirements.
- Intruders can access the memory of protected processes by disabling PPL. Using kernel drivers, they can clear Protection field of EPROCESS structure, without triggering any security mechanisms, such PatchGuard.
- Also, intruders can protect their malware apps by escalating their level of PPL. Windows does not provide any API to modify PPL in run time, nor does it trap such illegal memory manipulations.
- MR has been extended to protect PPL from being illegally used. MR prevents disabling PPL for the protected processes and blocks enabling PPL for non-protected processes.
- MR has been successfully tested on the recent Windows 10, version 20H2 Build 19042.631 x64 with affordable performance degradation. MR works well only to protect rarely accessed data.
- MR has extended features comparing with WDCG: MR can protect illegally enabling and disabling PPL for all processes, including LSASS; while WDCG can protect only LSASS memory, leaving out attacks on PPL.

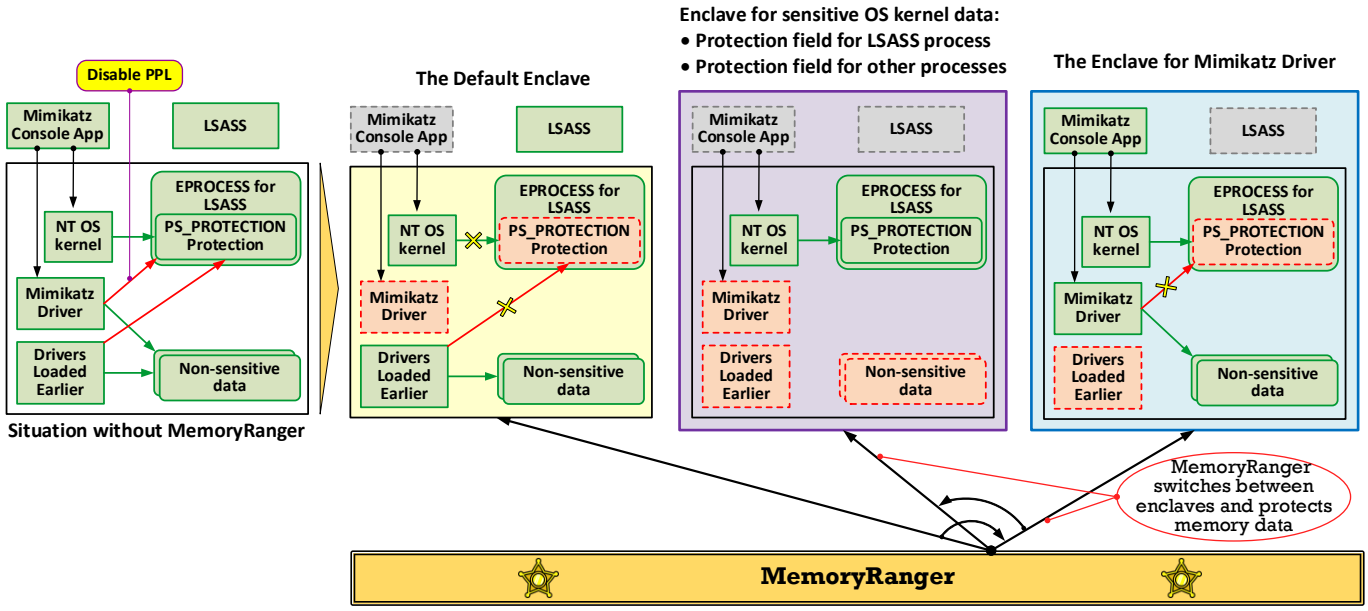


Fig. 8. Case 1: MemoryRanger protects the EPROCESS structure of the LSASS process from being patched by Mimikatz driver using three enclaves: the default one for drivers loaded earlier, the enclave for sensitive OS kernel data, and a separate enclave for newly loaded Mimikatz driver

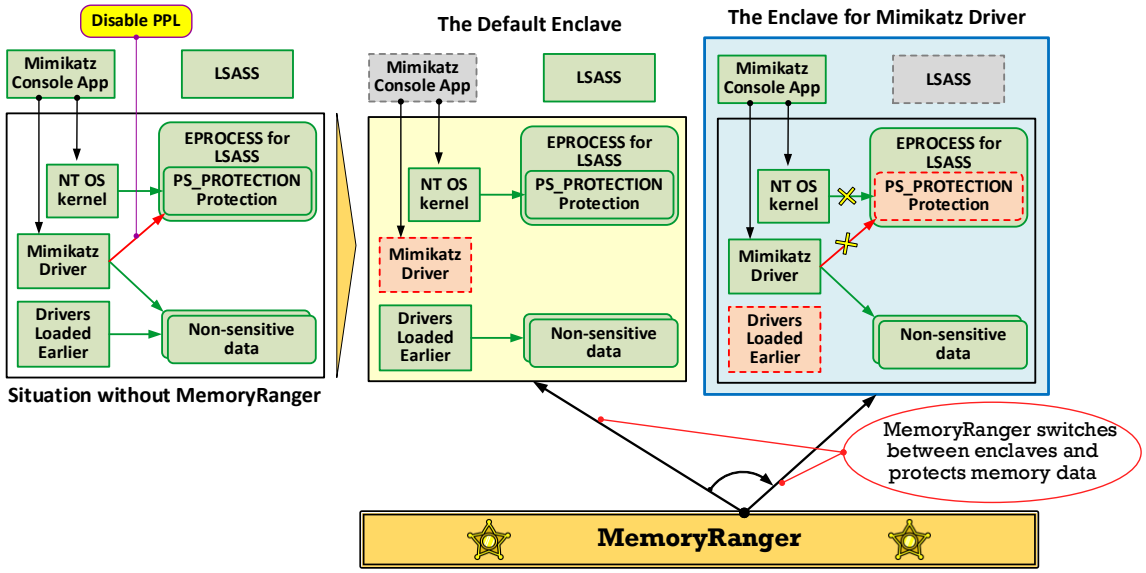


Fig. 9. Case 2: MemoryRanger protects the EPROCESS structure of the LSASS process from being patched by Mimikatz driver using two enclaves: the default one for drivers loaded earlier and a separate enclave for newly loaded Mimikatz driver

VI. Future Plans

Windows Process can use various flags to resilient itself:

- Flags3 and DisallowUserTerminate flags, which prevents process from being terminated;
- ControlFlowGuardEnabled,
- VmDeleted,

MemoryRanger implements kind of Next-Generation Secure Computing Base (NGSCB).

VII. References

[1] Pearson Education, Microsoft Windows Security, [Online]. Available: https://www.microsoftpressstore.com/articles/article.aspx?p=2228450&seqNum=2

[2] P. Yosifovich, M. Russinovich, D. Solomon, and A. Ionescu, "Windows Internals, Part 1: System architecture, processes, threads, memory management, and more (Developer Reference)", 7th Edition, 2017.

[3] B. Blunden, "The Rootkit Arsenal: Escape and Evasion: Escape and Evasion in the Dark Corners of the System", 1st Edition, 2009.

[4] A. Tanenbaum, and H. Bos, "Modern Operating Systems 4th Edition", 4th edition, 2014.

[5] ERNW, "Work Package 2: Analysis of Windows 10", SiSyPHuS Win10: Study on the system structure, logging, hardening and security function of Windows 10, 2020. [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Cyber-Sicherheit/SiSyPHus/Workpackage2_Analyse_Gesamtsystem.pdf?__blob=publicationFile&v=2

[6] Microsoft. Protected Processes. Windows Vista. [Online]. Available: http://download.microsoft.com/download/a/f/7/af7777e5-7dcd-4800-8a0a-b18336565f5b/process_vista.doc

[7] B. Delpy, "A little tool to play with Windows security". Source Code of Mimikatz. GitHub. 2020. [Online]. Available: https://github.com/gentilkiwi

[8] Astr0baby, "Unloading AV from Windows 10". WordPress. 2017. [Online]. Available: https://astr0baby.wordpress.com/2017/09/11/unloading-av-from-windows-10/

[9] B. Blaauwendraad, T. Ouddeken, C. Van Bockhaven, "Using Mimikatz' driver, Mimidrv, to disable Windows Defender in Windows". 2020. [Online]. Available: https://rp.delaat.net/2019-2020/p61/report.pdf

[10] R. Boonen,. "Throwing an AquaWrench into the Kernel". IBM RedCON. 2020. [Online]. Available: https://github.com/FuzzySecurity/IBM-RedCON-2020

[11] DarthTon. "How-To Make Your External Hack 'Undetectable' (BattlEye Bypass)". UnKnoWnCheaTs - Multiplayer Game Hacks and Cheats. 2014. [Online]. Available: https://www.unknowncheats.me/forum/904106-post5.html

[12] J. Steube, G. Gristina. "Hashcat – advanced password recovery". 2016. [Online]. Available: https://hashcat.net/hashcat/

[13] K. Van Impe, "Mimikatz and hashcat in practice". 2019. [Online]. Available: https://www.vanimpe.eu/2019/03/07/mimikatz-and-hashcat-in-practice/

[14] MicrosoftDocs. "Reduce attack surfaces with attack surface reduction rules". 2019. [Online]. Available: https://docs.microsoft.com/en-us/windows/security/threat-protection/microsoft-defender-atp/attack-surface-reduction

[15] T. Hunter. "Bypassing Windows Defender cheap and cheerful: obfuscation of Mimikatz". 2019. [Online]. Available: https://translate.google.com/translate?sl=auto&tl=en&u=https://habr.com/ru/company/tomhunter/blog/454758/

[16] M. Lavrijsen. "Protected Processes Light Killer". GitHub. 2019. [Online]. Available: https://github.com/Mattiwatti/PPLKiller

[17] FireF0X. "Multi-purpose proof-of-concept tool based on CPU-Z CVE-2017-15303". GitHub. 2018. [Online]. Available: https://github.com/hfiref0x/Stryker

[18] Harakirinox. "Bypass PPL (Protected Process Light) from user-mode using a vulnerable driver". UnKnoWnCheaTs - Multiplayer Game Hacks and Cheats. 2018. [Online]. Available: https://www.unknowncheats.me/forum/anti-cheat-bypass/262766-bypass-ppl-protected-process-light-user-mode-using-vulnerable-driver.html

[19] H. Bui. "Weaponizing vulnerable driver for privilege escalation", Gigabyte Edition! Medium. 2019. [Online]. Available: https://medium.com/@fsx30/weaponizing-vulnerable-driver-for-privilege-escalation-gigabyte-edition-e73ee523598b

[20] H. Bui. "Weaponizing Gigabyte driver for priv escalation and bypass PPL", GitHub. 2019. [Online]. Available: https://github.com/hoangprod/DanSpecial

[21] Notscimmy. “Elevate a process to be a protected process” , GitHub. 2019. [Online]. Available: https://github.com/notscimmy/pplib

[22] J. Forshaw, “Microsoft Windows PPL Process Injection Privilege Escalation” 2017. [Online]. Available: https://packetstormsecurity.com/files/143946/Microsoft-Windows-PPL-Process-Injection-Privilege-Escalation.html

[23] J. Forshaw, “Injecting Code into Windows Protected Processes using COM - Part 2”, 2018. https://googleprojectzero.blogspot.com/2018/11/injecting-code-into-windows-protected.html

[24] S. Do, “Preventing Mimikatz steal Windows system password. Penetration Testing”. 2019. [Online]. Available: https://securityonline.info/prevent-mimikataz/

[25] F. Lagrasta, MsvpPasswordValidate hooking. Dumping local credentials by hooking MsvpPasswordValidate in NtlmShared.dll. 2020. [Online]. Available: https://offnotes.notso.pro/abusing-credentials/dumping-credentials/msvppasswordvalidate-hook

[26] P. Ciholas, J. Such, A. Marnerides, B. Green, J. Zhang, U. Roedig, “Fast and Furious : Outrunning Windows Kernel Notification Routines from User-Mode”. Detection of Intrusions and Malware, and Vulnerability Assessment. DIMVA. 2020. [Online]. Available: https://doi.org/10.1007/978-3-030-52683-2_4

[27] J. Forshaw, “Injecting Code into Windows Protected Processes using COM - Part 1”. 2018. [Online]. Available: https://googleprojectzero.blogspot.com/2018/10/injecting-code-into-windows-protected.html

[28] M. Baranauskas. “Dumping Lsass Without Mimikatz”. [Online]. Available: https://www.ired.team/offensive-security/credential-access-and-credential-dumping/dump-credentials-from-lsass-process-without-mimikatz

[29] A. Ionescu, “Unreal Mode: Breaking Protected Processes”. *NoSuchCon*. 2014. [Online]. Available: https://www.slideshare.net/NoSuchCon/d3-05-alexionescubreakingprotectedprocesses

[30] Achilles, “The Birth of a Process Part-1”, 2020. [Online]. Available: https://medium.com/@Achilles8284/the-birth-of-a-process-part-1-bfb4fdac070e

[31] Bitdefender, “GravityZone Administrator's Guide”. 2019. [Online]. Available: https://www.bitdefender.co.th/resources/GravityZoneEnterprise/Current/Documentation/en_US/Bitdefender_GravityZone_AdministratorsGuide_7_enUS.pdf

[32] Cisco, “AMP for Endpoints User Guide”, 2020. [Online]. Available: https://docs.amp.cisco.com/AMP%20for%20Endpoints%20User%20Guide.pdf

[33] Eset, “Easy Protection, ESET Internet Security & ESET Smart Security Premium”, 2019. [Online]. Available: https://forum.eset.com/topic/19683-easy-protection/

[34] KaspeskyLab, “About Protected Process Light (PPL) technology for Windows”, 2020. [Online]. Available: https://support.kaspersky.com/13905

[35] SolarWinds, “Protected Process Light and SentinelOne Agents”, 2020. [Online]. Available: https://documentation.solarwindsmsp.com/EDR/Liberty/en/troubleshooting-windows-agents/troubleshooting-windows-agents/protected-process-light-and-sentinelone-agents.html

[36] McAfee, “Processes that Endpoint Security installs”, 2020. [Online]. Available: https://kc.mcafee.com/corporate/index?page=content&id=KB87791

[37] Microsoft, “Windows Defender in Windows 10: System integration”, 2015. [Online]. Available: https://download.microsoft.com/download/0/B/5/0B5EC8E7-1313-4477-AE4F-8F3C9FEBC1DB/MMPC%20Threat%20Intelligence%20August%202015.pdf

[38] MSDN. “Configuring Additional LSA Protection. Security and Assurance. Credentials Protection and Management’, 2016. [Online]. Available: https://docs.microsoft.com/en-us/windows-server/security/credentials-protection-and-management/configuring-additional-lsa-protection

[39] MSDN. “ZwQueryInformationProcess function. Process and Thread Functions”. 2016. [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/procthread/zwqueryinformationprocess

[40] Ntopcode. “Anatomy of the Process Environment Block (PEB) (Windows Internals)”. 2018. [Online]. Available: https://ntopcode.wordpress.com/2018/02/26/anatomy-of-the-process-environment-block-peb-windows-internals

[41] A. Ionescu, “The Evolution of Protected Processes Part 1: Pass-the-Hash Mitigations in Windows 8.1.” Alex Ionescu’s Blog. 2013. [Online]. Available: http://www.alex-ionescu.com/?p=97

[42] A. Ionescu, “The Evolution of Protected Processes Part 2: Exploit/Jailbreak Mitigations, Unkillable Processes and Protected Services”. Alex Ionescu’s Blog. 2013. [Online]. Available: http://www.alex-ionescu.com/?p=116

[43] B. I. Aquilino, “Relevance of Security Features Introduced in Modern Windows OS”. Master’s thesis. 2019. [Online]. Available: https://aaltodoc.aalto.fi/bitstream/handle/123456789/38990/master_Aquilino_Broderick_2019.pdf

[44] A. Ionescu, and J. Forshaw, “Unknown Known DLLs and other Code Integrity Trust Violations: Breaking Signature Guarantees in Windows”. REcon. 2018. [Online]. Available: https://docplayer.net/storage/100/144657723/1609772315/rmRiBDQll2c-D7LrqAVwZg/144657723.pdf

[45] G.Tworek, “Creating Protection Service”. Github Source Code. 2020. [Online]. Available: https://github.com/gtworek/PSBits/blob/master/Services/SpoilService.c

[46] Path, “Experimenting with Protected Processes and Threat-Intelligence”. 2020. [Online]. Available: https://blog.tofile.dev/2020/12/16/elam.html

[47] zwclose7, “Set a process as critical process using NtSetInformationProcess function”, 2013. [Online]. Available: http://www.rohitab.com/discuss/topic/40275-set-a-process-as-critical-process-using-ntsetinformationprocess-function/

[48] P. Yosifovich, “Windows Kernel Programming”. 2019, CreateSpace Independent Publishing Platform

[49] J. Mulder. “Mimikatz Overview, Defenses and Detection”. SANS Institute. 2014. [Online]. Available: https://daloo.de/testwebdavhere/mimikatz-overview-defenses-detection-36780.pdf

[50] M. Hand. “Mimidrv In Depth: Exploring Mimikatz’s Kernel Driver”. Posts by SpecterOps Team Members. Medium. 2018. [Online]. Available: https://posts.specterops.io/mimidrv-in-depth-4d273d19e148

[51] A. Chester. “Exploring Mimikatz - Part 1 – Wdigest”. 2019. [Online]. Available: https://blog.xpnsec.com/exploring-mimikatz-part-1/

[52] A. Chester. ‘Exploring Mimikatz - Part 2 – SSP”. 2019. [Online]. Available: https://blog.xpnsec.com/exploring-mimikatz-part-2/

[53] D. Patil, B. Meshram, “Analysis of Windows In-Memory Structures for Extracting Digital Evidence”. 2019. [Online]. Available: http://ijcsse.org/published/volume6/issue6/p1-V6I6.pdf

[54] P. Gkatziroulis. “Preventing Mimikatz Attacks”. Medium. 2019. [Online]. Available: https://medium.com/blue-team/preventing-mimikatz-attacks-ed283e7ebdd5

[55] B. Delpy, “Mimikatz latest release”. GitHub. 2020. [Online]. Available: https://github.com/gentilkiwi/mimikatz/releases/latest

[56] B. Delpy, “Anti-Virus Software Thinks Mimikatz Is Malware”. Mimikatz Issues. 2016. [Online]. Available: https://github.com/gentilkiwi/mimikatz/issues/55#issuecomment-238336107

[57] D. Altermatt, “Detecting PPL Manipulation? A Test Using LSASS as an Example”. SCIP. 2020. [Online]. Available: https://www.scip.ch/en/?labs.20200116

[58] E. Perla, and M. Oldani, “A Guide to Kernel Exploitation: Attacking the Core” 1st Edition. Massachusetts, US: Syngress. 2010.

[59] M. Schneider, “Local Security Authority. Keeping Secrets Safe”. SCIP. 2019. [Online]. Available: https://www.scip.ch/en/?labs.20191121

[60] Woshub. How to Obtain SeDebugPrivilege when Debug Program Policy is Enabled. Windows OS Hub. 2017. [Online]. Available: http://woshub.com/obtain-sedebugprivilege-debug-program-policy-enabled/

[61] D. Kennedy, “Dumping Wdigest Creds With Meterpreter Mimikatz/Kiwi In Windows 8.1”. 2018. [Online]. Available: https://www.trustedsec.com/blog/dumping-wdigest-creds-with-meterpreter-mimikatzkiwi-in-windows-8-1/

[62] MicrosoftDocs. “Microsoft Security Advisory 2871997. Security Advisories”. 2017. [Online]. Available: https://docs.microsoft.com/en-us/security-updates/SecurityAdvisories/2016/2871997

[63] S. Anson. “Applied Incident Response.” Wiley; 1st edition. 2020

[64] E. Shlomo. “Restricted RDP for Admin (RestrictedAdmin).” Elli Shlomo Blog. 2019. [Online]. Available: https://www.eshlomo.us/restricted-rdp-for-admin-restrictedadmin/

[65] MicrosoftDocs. “Protect derived domain credentials with Windows Defender Credential Guard. Identity and access protection”. 2017. [Online]. Available: https://docs.microsoft.com/en-us/windows/security/identity-protection/credential-guard/credential-guard

[66] K. Joyce, “Defender Credential Guard: Protecting Your Hashes”. Insider Threat Security Blog. 2019. [Online]. Available: https://blog.stealthbits.com/defender-credential-guard-protecting-your-hashes/

[67] I. Korkin, “Divide et Impera: MemoryRanger Runs Drivers in Isolated Kernel Spaces”, *In Proceedings of the BlackHat Europe Conference*, London, UK, December 5-6, 2018. [Online]. Available: https://www.blackhat.com/eu-18/briefings/schedule/#divide-et-impera-memoryranger-runs-drivers-in-isolated-kernel-spaces-12668

[68] I. Korkin, “MemoryRanger Prevents Hijacking FILE_OBJECT Structures in Windows Kernel”, *In Proceedings of the 14th Annual ADFSL 2019 Conference on Digital Forensics, Security and Law*, Daytona Beach, Florida, USA, May 15-16, 2019, ISSN 1931-7379. [Online]. Available: https://igorkorkin.blogspot.com/2019/04/memoryranger-prevents-hijacking.html

[69] I. Korkin, “Kernel Hijacking is Not an Option: MemoryRanger Comes to the Rescue Again*”, Hack In The Box Security Conference (HITBLockdown002)*, July 25, 2020, Singapore. [Online]. Available: http://conference.hitb.org/hitb-lockdown002/sessions/kernel-hijacking-is-not-an-option-memoryranger-comes-to-rescue-again/

[70] I.Korkin. YouTube Channel. 2021 [Online]. Available: https://www.youtube.com/c/IgorKorkin

[71] ERNW, “Work Package 6: Virtual Secure Mode”, SiSyPHuS Win10: Analysis of Virtual Secure Mode. 2020. [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Cyber-Sicherheit/SiSyPHus/Workpackage6_Virtual_Secure_Mode.pdf?__blob=publicationFile&v=2

[72] Trusted Computing in Windows. Protected Processes. https://trustedwindows.wordpress.com/hauptseite/trusted-computing-in-windows/protected-processes/