How to Detect Parent PID (PPID) Spoofing Attacks

Huseyin Can YUCEEL & Eren KARABACAK | January 19, 2022

Parent Process ID (PPID) spoofing attacks are commonly used for defense evasion and privilege escalation. Although default Windows security logs are able to detect some cyberattacks, detecting Parent Process ID spoofing attacks requires utilizing additional log types that are not enabled in default configuration. In this article, we will talk in detail about:

- Process ID and its types
- Event Tracing For Windows (ETW)
- Parent Process ID spoofing attacks and example
- Detection of Parent Process ID spoofing attacks using Kernel-Process Logs

1. What is Process ID?

Process ID (PID) is a unique number that identifies any process currently running on the operating system. In Windows, Process ID can be viewed using various tools and commands such as tasklist and wmic commands or Process Explorer in Sysinternals. In Linux and Unix, the ps command lists PIDs. PID can be used to call or kill a process.

When a process spawns another process, it becomes a parent process, and the spawned process becomes a child process. Process IDs of these processes are recorded in event logs. For example, Kernel-Process logs of *Process A* keep three types of PID records:

- Process ID: PID of *Process A*
- Parent Process ID: PID of the parent process of *Process A*
- Execution Process ID: PID of the process that executes *Process A*

1.1 Viewing Process ID in Windows

Let's show how to view Process ID in Windows using tasklist and wmic command-line tools. Also, Process Explorer shown in Figure 1 is an easy-to-use alternative to view Process ID.

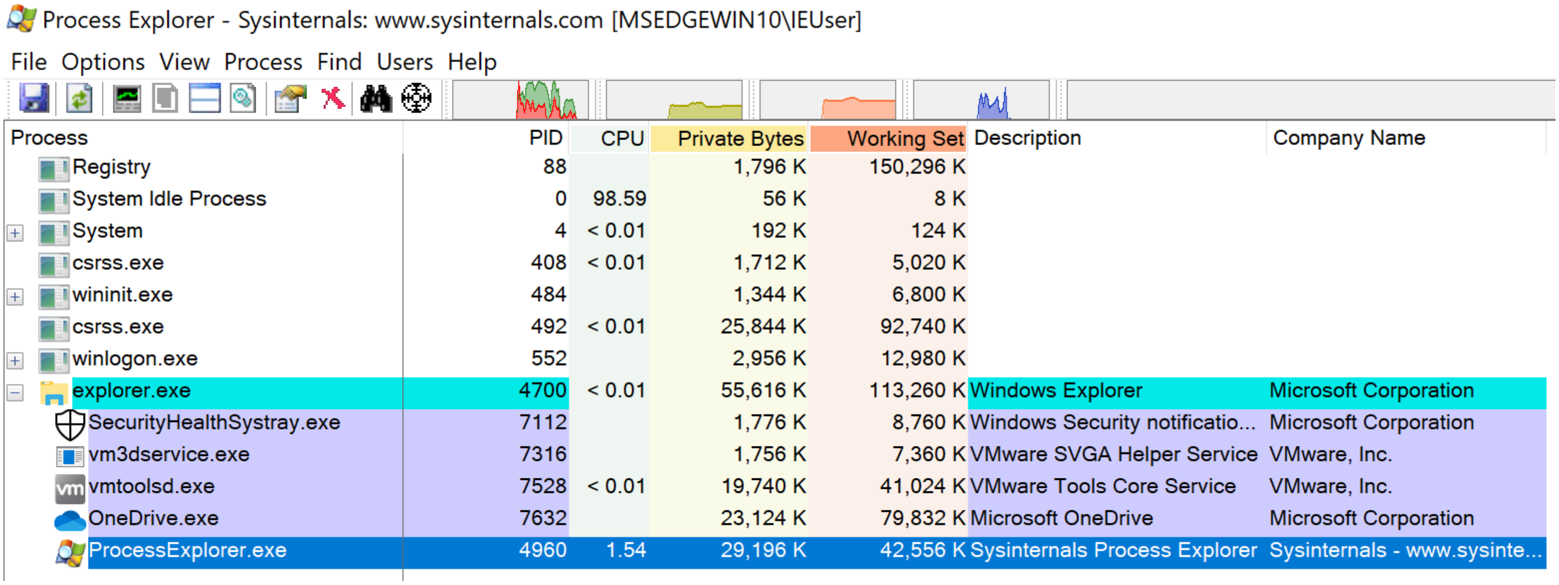
```
> tasklist /fi "IMAGENAME eq explorer.exe"

Image Name                     PID Session Name        Session#    Mem Usage
========================= ======== ================ =========== ============
explorer.exe                  4700 Console                    1    267,936 K

> tasklist /fi "IMAGENAME eq ProcessExplorer.exe"
```

```
Image Name                     PID Session Name        Session#    Mem Usage
========================= ======== ================ =========== ============
ProcessExplorer.exe           4960 Console                    1     55,880 K

> wmic process where (processid=14324) get parentprocessid
ParentProcessId
4700
```


Process Explorer - Sysinternals: www.sysinternals.com [MSEDGEWIN10\IEUser]
File
Options
View
Process
Find
Users
Help

Process	PID	CPU	Private Bytes	Working Set	Description	Company Name
Registry	88		1,796 K	150,296 K		
System Idle Process	0	98.59	56 K	8 K		
System	4	< 0.01	192 K	124 K		
csrss.exe	408	< 0.01	1,712 K	5,020 K		
wininit.exe	484		1,344 K	6,800 K		
csrss.exe	492	< 0.01	25,844 K	92,740 K		
winlogon.exe	552		2,956 K	12,980 K		
explorer.exe	4700	< 0.01	55,616 K	113,260 K	Windows Explorer	Microsoft Corporation
SecurityHealthSystray.exe	7112		1,776 K	8,760 K	Windows Security notificatio...	Microsoft Corporation
vm3dservice.exe	7316		1,756 K	7,360 K	VMware SVGA Helper Service	VMware, Inc.
vmtoolsd.exe	7528	< 0.01	19,740 K	41,024 K	VMware Tools Core Service	VMware, Inc.
OneDrive.exe	7632		23,124 K	79,832 K	Microsoft OneDrive	Microsoft Corporation
ProcessExplorer.exe	4960	1.54	29,196 K	42,556 K	Sysinternals Process Explorer	Sysinternals - www.sysinte...

Figure 1: Process Explorer

2. Event Tracing for Windows

Event Tracing for Windows (ETW) is a kernel-level monitoring tool to monitor kernel or application-level logs in real-time [1]. These logs can be used to debug an application or detect performance issues.

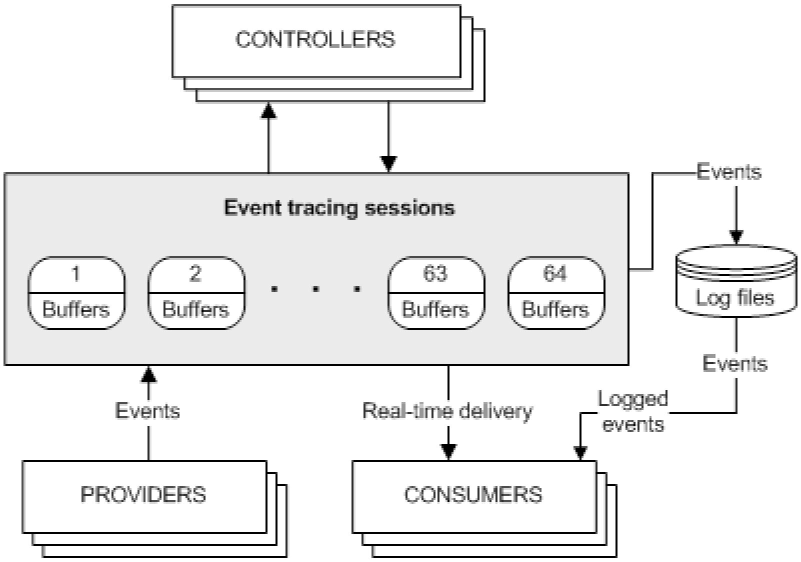


Figure 2: Event Tracing Model

ETW consists of 3 components;

- **Controllers**: The modules that
  - define size and location of the log file
  - start and stop event tracing sessions
  - enable providers to log events to the session
- **Providers**: The modules that produce event logs
  - generate events
  - produce event logs when enabled by a controller
- **Consumers**: The modules that use the generated event logs

2.1. How to Start an Event Tracing Session?

ETW structure allows dynamic management of event tracing sessions. Event tracing sessions can be started and managed by built-in controllers such as the logman command and perfmon tool.

2.1.1. Starting Event Trace Session with logman

logman is a built-in Windows command-line tool included in the controller group of ETW components. logman can start and control event trace sessions. Let's see some examples of usage of logman.

- List available providers on Windows

```
> logman query providers

Provider                              GUID
--------------------------------------------------------------------------------
.NET Common Language Runtime          {E13C0D23-CCBC-4E12-931B-D9CC2EEE27E4}
ACPI Driver Trace Provider            {DAB01D4D-2D48-477D-B1C3-DAAD0CE6F06B}
Active Directory Domain Services: SAM {8E598056-8993-11D2-819E-0000F875A064}
Active Directory: Kerberos Client     {BBA3ADD2-C229-4CDB-AE2B-57EB6966B0C4}
Active Directory: NetLogon            {F33959B4-DBEC-11D2-895B-00C04F79AB69}
ADODB.1                               {04C8A86F-3369-12F8-4769-24E484A9E725}
ADOMD.1                               {7EA56435-3F2F-3F63-A829-F0B35B5CAD41}
Application Popup                     {47BFA2B7-BD54-4FAC-B70B-29021084CA8F}
Application-Addon-Event-Provider      {A83FA99F-C356-4DED-9FD6-5A5EB8546D68}
ATA Port Driver Tracing Provider      {D08BD885-501E-489A-BAC6-B7D24BFE6BBF}
AuthFw NetShell Plugin                {935F4AE6-845D-41C6-97FA-380DAD429B72}
. . .
```

- List the providers that are currently tracing and working in Windows

```
> logman query -ets

Data Collector Set                      Type                 Status
--------------------------------------------------------------------------------
Circular Kernel Context Logger          Trace                Running
AppModel                                Trace                Running
DiagLog                                 Trace                Running
Diagtrack-Listener                      Trace                Running
EventLog-Application                    Trace                Running
EventLog-System                         Trace                Running
iclsClient                              Trace                Running
```

```
iclsProxy                  Trace              Running
Microsoft-Windows-Kernel-Process     Trace              Running
SgrmEtwSession             Trace              Running
...
```

- Look into details of any provider with their GUID

```
> logman query providers {22FB2CD6-0E7B-422B-A0C7-2FAD1FD0E716}

Provider                         GUID
-------------------------------------------------------------------------------
Microsoft-Windows-Kernel-Process     {22FB2CD6-0E7B-422B-A0C7-2FAD1FD0E716}

Value              Keyword            Description
-------------------------------------------------------------------------------
0x0000000000000010  WINEVENT_KEYWORD_PROCESS
0x0000000000000020  WINEVENT_KEYWORD_THREAD
0x0000000000000040  WINEVENT_KEYWORD_IMAGE
0x0000000000000080  WINEVENT_KEYWORD_CPU_PRIORITY
0x0000000000000100  WINEVENT_KEYWORD_OTHER_PRIORITY
0x0000000000000200  WINEVENT_KEYWORD_PROCESS_FREEZE
0x0000000000000400  WINEVENT_KEYWORD_JOB
0x0000000000000800  WINEVENT_KEYWORD_ENABLE_PROCESS_TRACING_CALLBACKS
0x0000000000001000  WINEVENT_KEYWORD_JOB_IO
0x0000000000002000  WINEVENT_KEYWORD_WORK_ON_BEHALF
0x0000000000004000  WINEVENT_KEYWORD_JOB_SILO
0x8000000000000000  Microsoft-Windows-Kernel-Process/Analytic

Value              Level              Description
-------------------------------------------------------------------------------
0x04               win:Informational    Information

PID                Image
-------------------------------------------------------------------------------
0x00000000
```

Providers may produce multiple log types categorized with different keywords. When setting up a data collector, log types to be collected are selected using these keywords.

The example log shown below can be retrieved when the WINEVENT_KEYWORD_WORK_ON_BEHALF keyword from Microsoft-Windows-Kernel-Process is assigned to an event trace sessio

```
-<Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
-<System>
   <Provider Name="Microsoft-Windows-Kernel-Process" Guid="{22fb2cd6-0e7b-422b-a0c7-2fad1fd0e716}" />
   <EventID>21</EventID>
   <Version>0</Version>
   <Level>4</Level>
   <Task>18</Task>
   <Opcode>0</Opcode>
   <Keywords>0x8000000000002000</Keywords>
   <TimeCreated SystemTime="2022-01-10T13:01:49.527905100Z" />
   <EventRecordID>20</EventRecordID>
   <Correlation />
   <Execution ProcessID="1068" ThreadID="3716" ProcessorID="1" KernelTime="1" UserTime="1" />
   <Channel>Microsoft-Windows-Kernel-Process/Analytic</Channel>
   <Computer>MSEDGEWIN10</Computer>
   <Security />
  </System>
-<EventData>
   <Data Name="OldWorkOnBehalfThreadID">0</Data>
   <Data Name="NewWorkOnBehalfThreadID">1432</Data>
  </EventData>
</Event>
```

- Start a new event trace session

```
> logman create trace "Sample-trace" -ets
The command completed successfully.
```

- Assign the provider and select logs to be collected for the new event trace session

```
> logman update Sample-trace -p Microsoft-Windows-Kernel-Process 0x70 -ets
The command completed successfully.

> logman query Sample-trace -ets


Name:             Sample-trace
Status:           Running
Root Path:        C:\
Segment:          Off
Schedules:        On


Name:             Sample-trace\Sample-trace
Type:             Trace
Output Location:     C:\Sample-trace.etl
Append:           Off
Circular:         Off
Overwrite:        Off
Buffer Size:      8
Buffers Lost:     0
Buffers Written:  2
Buffer Flush Timer: 0
Clock Type:       Performance
File Mode:        File


Provider:
Name:             Microsoft-Windows-Kernel-Process
Provider Guid:    {22FB2CD6-0E7B-422B-A0C7-2FAD1FD0E716}
Level:            255
KeywordsAll:      0x0
KeywordsAny:      0x70 (WINEVENT_KEYWORD_PROCESS,WINEVENT_KEYWORD_THREAD,WINEVENT_KEYWORD_IMAGE)
Properties:       64
Filter Type:      0


The command completed successfully.
```

Event Trace Session logs are recorded in the Output Location in .etl format. The logs can be examined in the Event Viewer. Keywords of the collected logs are given as KeywordsAny. In the example, 0x70 parameter in KeywordsAny represents the sum of the selected keywords' parameter given below.

```
0x0000000000000010 WINEVENT_KEYWORD_PROCESS
0x0000000000000020 WINEVENT_KEYWORD_THREAD
0x0000000000000040 WINEVENT_KEYWORD_IMAGE
```

2.1.2. Starting Event Trace Session with Performance Monitor

Windows Performance Monitor is a Microsoft Management Console (MMC) snap-in that provides tools for monitoring application and hardware performance in real-time and managing data collections. Event Trace Sessions can be managed by using Data Collector Sets on the Performance Monitor.

Performance Monitor can create new Event Trace Sessions or manage currently running sessions. It can also monitor other providers running Event Trace Sessions using Data Collector Sets.

Let's start an Event Trace Session with Performance Monitor. Microsoft has a similar guide to start a session [2]. Under the Data Collector Sets and User Defined section,

- Right Click > New > Data Collector Set
- Create manually
- Create Data Logs > Event Data Trace
- Add Providers > Microsoft-Windows-Kernel-Process
- Edit Properties > Add 0x70 Manually

Figure 3: Performance Monitor

2.1.3 Viewing collected logs

Recorded Event Trace Session logs can be examined using Event Viewer. To view in more detail,

- Enable “Show Analytic and Debug Logs” under the View tab
- Open Saved Log under the Action tab
- Select saved log file In the logman example. The saved log was stored at “C:\Sample-trace.etl”.

It should be noted that logs opened in the Event Viewer are not in real-time. If the event trace session is still ongoing, a copy of the log file will be created and the copy will be opened by the Event Viewer. In Figure 4, Sample-trace.etl is opened in Event Viewer and a single log is viewed in XML format.

Figure 4: A sample log examined in Event Viewer

3. Parent Process ID Spoofing

Parent PID Spoofing is a technique that allows attackers to run processes under any parent process they want. Effectively, the child process can have the Parent Process ID selected by the attackers. This technique enables malicious processes to evade detection methods based on the parent-child relationship and inherit access tokens from the parent process to elevate privilege.

Let’s give examples about defense evasion and privilege escalation using Parent PID spoofing.

- Assume that a malicious payload is delivered using a Word document to create a PowerShell session. In this scenario, PowerShell becomes the child process of winword.exe. However, this is not normal behavior of the Word and is easily detected by defense mechanisms. To evade detection by the defense mechanisms, a payload can be created with a different parent PID of a parent process that usually spawns PowerShell.
- Using the PROC_THREAD_ATTRIBUTE_PARENT_PROCESS attribute of CreateProcess Win32 API call, a malicious process may inherit the access token of the parent process [3]. If the parent process has a SYSTEM level access token, the malicious process gets a SYSTEM level access token as well.

Parent PID Spoofing attacks are categorized as a sub technique under the T1134 Access Token Manipulation techniques in the MITRE ATT&CK framework [4]. It is a widely used technique and detection of Parent PID spoofing is essential to the secure operation of the assets.

4. Detecting Parent PID Spoofing Attacks Using Kernel-Process Logs

So far, we have talked about process ID types, event tracing sessions, log collection, and examination. Let’s see how Parent PID spoofing can be detected using Kernel-Process logs step by step in the Windows command line.

4.1 Starting an event trace session

Using logman, start an event tracing session and start recording Kernel-Process logs.

```
> logman create trace "Sample-trace" -ets
The command completed successfully.

> logman update Sample-trace -p Microsoft-Windows-Kernel-Process 0x70 -ets
The command completed successfully.
```

4.2 Starting a notepad process

Using cmd.exe, start notepad.exe

- Child Process: notepad.exe (PID: 5756)
- Parent Process: cmd.exe (PID: 5428)
- Executing Process: cmd.exe (PID: 5428)

4.3 Starting a notepad process with Parent Process ID spoofing

Using PPID-spoof.ps1, start notepad.exe [5].

- Child Process: notepad.exe (PID: 7980)
- Parent Process: winlogon.exe (PID: 548)
- Executing Process: powershell.exe (PID: 6312)

```
> powershell

PS> .\tasklist /fi "IMAGENAME eq powershell.exe"

Image Name                     PID Session Name        Session#    Mem Usage
========================= ======== ================ =========== ============
powershell.exe                6312 Console                    1     72,808 K

PS> .\PPID-spoof.ps1
PS> [MyProcess]::CreateProcessFromParent(548,"C:\Windows\System32\notepad.exe","")
[+] Got Handle for ppid: 548
[+] Updated proc attribute list
[+] Starting C:\Windows\System32\notepad.exe...True - pid: 7980 - Last error: 87
PS> exit

> tasklist /fi "IMAGENAME eq winlogon.exe"

Image Name                     PID Session Name        Session#    Mem Usage
========================= ======== ================ =========== ============
winlogon.exe                   548 Console                    1     11,960 K
```

```
> tasklist /fi "IMAGENAME eq notepad.exe"

Image Name                     PID Session Name        Session#    Mem Usage
========================= ======== ================ =========== ============
notepad.exe                   7980 Console                    1     14,100 K

> wmic process where (processid=7980) get parentprocessid
ParentProcessId
548
```

4.4 Examining Event Trace Logs

In the Event Viewer, open the logs recorded.

- Enable “Show Analytic and Debug Logs” under View tab
- Click on Saved Log under the Actions tab
- Open the saved log file at “C:\Sample-trace.etl”.

Let’s look at the log created for starting notepad.exe without Parent Process ID spoofing. There might be lots of logs collected. So search for the term “Process 5756 started” from the Actions tab.

In this example, we see that Execution Process ID and Parent Process ID are the same. Usually, this is an expected log for starting a process.

```
-<Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
-<System>
   <Provider Name="Microsoft-Windows-Kernel-Process" Guid="{22fb2cd6-0e7b-422b-a0c7-2fad1fd0e716}" />
   <EventID>1</EventID>
   <Version>2</Version>
   <Level>4</Level>
   <Task>1</Task>
   <Opcode>1</Opcode>
   <Keywords>0x8000000000000010</Keywords>
   <TimeCreated SystemTime="2022-01-11T06:59:52.449063400Z" />
   <EventRecordID>37682</EventRecordID>
   <Correlation />
   <Execution ProcessID="5428" ThreadID="5636" ProcessorID="0" KernelTime="3" UserTime="1" />
   <Channel />
```

```
    <Computer>MSEDGEWIN10</Computer>
    <Security />
  </System>
 -<EventData>
    <Data Name="ProcessID">5756</Data>
    <Data Name="CreateTime">2022-01-11T07:00:04.257230300Z</Data>
    <Data Name="ParentProcessID">5428</Data>
    <Data Name="SessionID">1</Data>
    <Data Name="Flags">0</Data>
    <Data Name="ImageName">\Device\HarddiskVolume1\Windows\System32\notepad.exe</Data>
    <Data Name="ImageChecksum">307779</Data>
    <Data Name="TimeDateStamp">314954348</Data>
    <Data Name="PackageFullName" />
    <Data Name="PackageRelativeAppId" />
  </EventData>
 </Event>
```

Now, let's look at the log created for starting notepad.exe with Parent Process ID spoofing. Search for the term "Process 7980 started" from the Actions tab.

In this example, we see that Execution Process ID and Parent Process ID do not match. This is very unusual for starting a process and very rarely legitimate. However, this mismatch is a key characteristic of Parent PID spoofing attacks. From this Kernel-Process log, we can detect malicious Parent Process ID spoofing attacks.

```
 -<Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
 -<System>
    <Provider Name="Microsoft-Windows-Kernel-Process" Guid="{22fb2cd6-0e7b-422b-a0c7-2fad1fd0e716}" />
    <EventID>1</EventID>
    <Version>2</Version>
    <Level>4</Level>
    <Task>1</Task>
    <Opcode>1</Opcode>
    <Keywords>0x8000000000000010</Keywords>
    <TimeCreated SystemTime="2022-01-11T06:59:30.467709500Z" />
    <EventRecordID>37475</EventRecordID>
    <Correlation />
    <Execution ProcessID="6312" ThreadID="5724" ProcessorID="0" KernelTime="14" UserTime="24" />
    <Channel />
    <Computer>MSEDGEWIN10</Computer>
    <Security />
  </System>
 -<EventData>
    <Data Name="ProcessID">7980</Data>
    <Data Name="CreateTime">2022-01-11T06:59:42.276169400Z</Data>
    <Data Name="ParentProcessID">548</Data>
    <Data Name="SessionID">1</Data>
    <Data Name="Flags">0</Data>
    <Data Name="ImageName">\Device\HarddiskVolume1\Windows\System32\notepad.exe</Data>
    <Data Name="ImageChecksum">307779</Data>
    <Data Name="TimeDateStamp">314954348</Data>
    <Data Name="PackageFullName" />
    <Data Name="PackageRelativeAppId" />
  </EventData>
 </Event>
```

5. Simulating with Picus

Picus Threat Library includes multiple Parent Process ID spoofing attack simulations. These attack simulations test detection and prevention capabilities of your security controls.

5.1 Parent Process ID Spoofing Simulation with Picus

5.2 Detection with Splunk

In the attack simulation with Picus, we saw that the Parent Process ID spoofing attack was not blocked by the security controls. Splunk search queries can be designed with different methods and logs types to detect Parent Process ID spoofing.

Loading...

As an example, let’s see how we can detect the attack with Splunk using Kernel-Process logs. Since it is possible to detect the attack by comparing Execution PID and Parent PID of a child process, we can devise the following Splunk search query for detection.

```
sourcetype="test-xml" "Event.System.EventID"=1 | eval Parent_ProcessId=mvindex('Event.EventData.Data', 3) | eval
Process_Name=mvindex('Event.EventData.Data', 10) | eval Result=if('Event.System.Execution{@ProcessID}'=Parent_ProcessId,
"Not Suspicious","Suspicious") | table Process_Name, Result
```

Figure 5: Parent Process ID Spoofing Detection with Splunk

References

[1] “About Event Tracing.” [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/etw/about-event-tracing.

[2] “How to: Use Performance Monitor to Collect Event Trace Data.” [Online]. Available: https://docs.microsoft.com/en-us/dynamics365/business-central/dev-itpro/administration/monitor-use-performance-monitor-collect-event-trace-data.

[3] “UpdateProcThreadAttribute function (processthreadsapi.h).” [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-updateprocthreadattribute.

[4] “Access Token Manipulation: Parent PID Spoofing.” [Online]. Available: https://attack.mitre.org/techniques/T1134/004/.

[5] decoder-it, “GitHub - decoder-it/psgetsystem: getsystem via parent process using ps1 & embedded c#” GitHub. [Online]. Available: https://github.com/decoder-it/psgetsystem.

Share this:

EMERGING THREAT

CVE-2024-47575: FortiManager Missing Authentication Zero-Day...

LEARN MORE ▸

EMERGING THREAT

Iranian Cyber Actors' Brute Force and Credential Access Attacks: CISA Alert AA24-...

LEARN MORE ▸

◀ ▶

Email*

Email

SUBSCRIBE NOW

Platform

The Security Validation Platform

Security Control Validation

Attack Surface Validation

Cloud Security Validation

Attack Path Validation

Detection Rule Validation

Integrations

Use Cases

Breach and Attack Simulation

Penetration Testing Automation

Continuous Threat Exposure Management

Resources

Blog

Purple Academy

Webinars

Reports

Case Studies

Press Releases

Datasheets

Cyberpedia

Events

Company

About Us

Leadership

Careers

Contact

Customer Support

Trust Center

Subscribe to Our Newsletter

Email*

Email

SUBSCRIBE NOW

Contact Us

info@picussecurity.com

Schedule a meeting