XblAuthManagerProxy.dll Integer overflow out of bounds write

**environment**

Microsoft Windows Version 10.0.19043.1348 64-bits

**Vulnerability description**

XblAuthManagerProxy! ReadStringFromStream allocates a heap due to an integer overflow, which causes a small heap block to be allocated, which leads to a subsequent write zero to this block of memory.

**Causes of Vulnerability**

XblAuthManagerProxy.dll provides a custom Marshal proxy XblAuthTokenResult trusted by the system. The proxy clsid is 4844e347-e354-4b24-9e0c-f4ae31ad55db

Since the agent is trusted by the system, I can act as a server on any process with system permissions, such as a service process. When the server performs UnmarshalInterface (XblAuthTokenResult::UnmarshalInterface), it will first determine whether the passed interface pointer IID is the interface of IXblAuthTokenResult IID :1937ee7c-f37f-4230-abbd-4b59d8c595eb

Then verify whether the size of the subsequent stream buffer is greater than 8, and then hit the vulnerable function ReadStringFromStream

ReadStringFromStream will read the content of subsequent streams and allocate memory in this form

//IStream ->Read

v6 = (\*(\_\_int64 (\_\_fastcall \*\*)(struct IStream \*, unsigned int \*, \_\_int64, int \*))(v3 + 24))(a1, &v10, 4i64, &v11);

if (v6 >= 0)

{

v7 = (unsigned \_\_int16 \*)CoTaskMemAlloc(2i64 \* (v10 + 1)); //Integer overflow

v5 = v7;

if (v7)

{

v8 = 2 \* v10;

v6 = (\*(\_\_int64 (\_\_fastcall \*\*)(struct IStream \*, unsigned \_\_int16 \*, \_QWORD, int \*))(\*(\_QWORD \*)a1 + 24i64))(

a1,

v7,

2 \* v10,

&v11);

if (v6 >= 0)

{

v5[v10] = 0; //trigger out of bounds to write zero

**POC**

Because I encountered some problems in the poc passing the interface IID, I made a poc that is not perfect. I am not sure how long it will take to complete it. What is certain is that we can definitely pass this interface IID.

So you need to make some adjustments in the debugger after running the poc to trigger the vulnerability

Since the poc will run the XblAuthTokenResult proxy with the BITS service as the server, run the poc once, the BITS service will generate the proxy stub with XblAuthManagerProxy.dll, and there is no crash due to the need to adjust

Then you need your kernel debugger windbg to attach to the BITS service and run the following command：

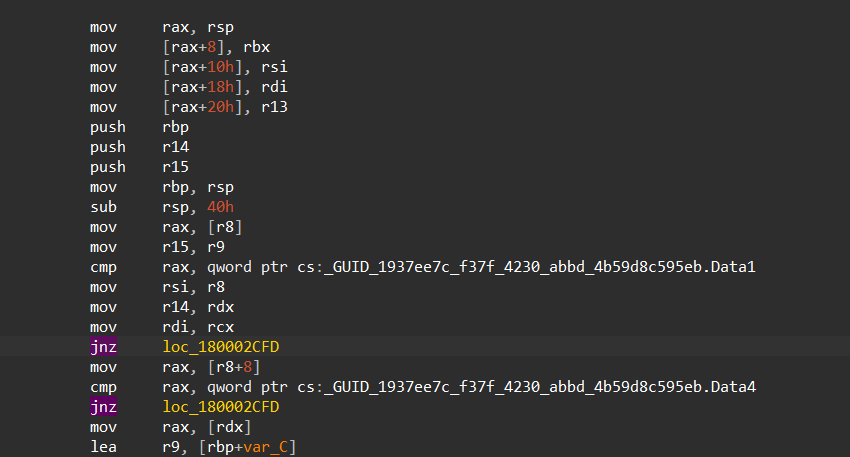
.process /p (BITS service EPROCESS)

.reload /f XblAuthManagerProxy.dll

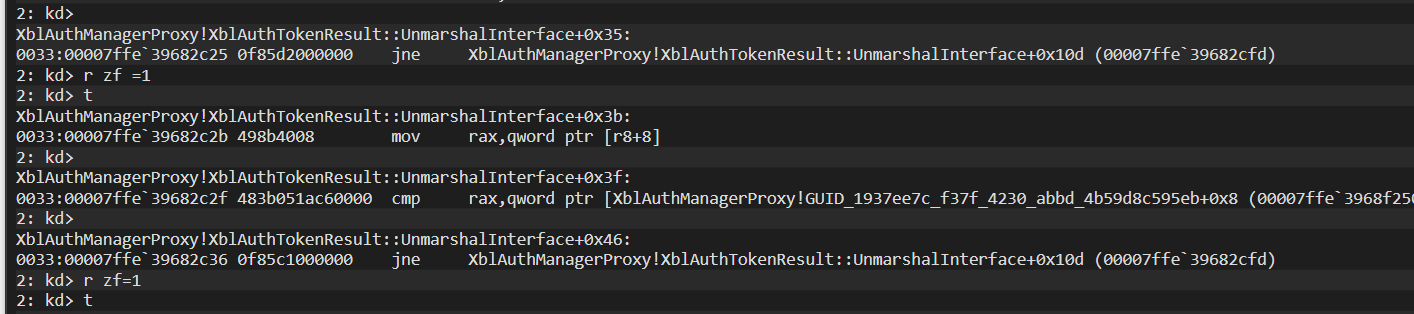
.process /p /i (BITS service EPROCESS)

bp XblAuthTokenResult::UnmarshalInterface

After doing this, run the poc again, the power will be hit, and then you need to manually skip the place where the interface IID is judged, the two Jne judges in the figure below



You can single-step to construct the zf flag when judging



Finally, the execution to the loophole produces an exception, the reason is that it is out of bounds to write 0