DIGITAL FORENSIC RESEARCH CONFERENCE

Inception: Virtual Space in Memory Space in Real Space - Memory Forensics of Immersive Virtual Reality with the HTC Vive

By

Peter Casey, Rebecca Lindsay-Decusati, Ibrahim Baggili, & Frank Breitinger

From the proceedings of

The Digital Forensic Research Conference

DFRWS 2019 USA

Portland, OR (July 15th - 19th)

Inception: Virtual Space in Memory Space in Real Space - Memory Forensics of Immersive Virtual Reality with the HTC Vive

Peter Casey, Rebecca Lindsay-Decusati, Ibrahim Baggili, & Frank Breitinger

Graduate Research Assistant & UNHcFREG Member

Presenting @ DFWRS 2019, Portland, OR, USA

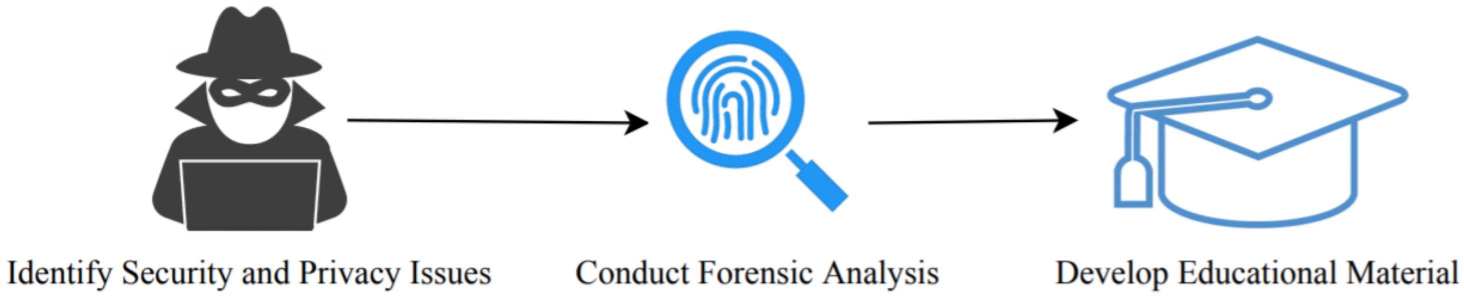
University of New Haven
Cyber Forensics Research & Education Group

UNHcFREG

Acknowledgment

- This material is based upon work supported by the National Science Foundation under Grant No. 1748950. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Background – Attacks against VR



- Overlay Attack
- Chaperone Attack
- Disorientation Attack
- Human Joystick
- Camera Exfiltration

Background – Forensics

- Surveyed nonvolatile artifacts
- Analyzed network traffic
- Steam (VR) and Oculus (Home)
- Select social VR applications
- Goal
  - Establish knowledge base of artifact location and content
  - Recreate a timeline of events

Problem

- Substantial amount of information resides only in memory.
  - Tracked device location / state
  - Virtual environment (VE) geometry and orientation

- What if a crime is committed in VR or a user is harmed while immersed?
- Chaperone, Disorientation, HJ attacks manipulate the live copy of the VE
- Can we recreate the VE from a memory dump?

Contributions

- To the best of our knowledge this is the primary account for specifically examining the memory forensics of VR systems.
- We share our analysis and findings that may impact future investigations involving VR systems.
- We employ and share a reusable methodology that may be adopted by others to create similar plugins.
- We construct an open source tool Vivedump, that may be used in the analysis of memory dumps of HTC Vive VR systems and share related datasets. The tool is a plug-in for the widely adopted Volatility framework.

Objective - Apparatus

UNHcFREG

- Can we recreate the VE from a memory dump?
- HTC Vive
  - 2 x Controllers
  - 2 x Base stations
- Windows 10 Workstation
  - 8/16 GB Ram
  - Steam VR
  - DumpIt

Methodology - Recon

UNHcFREG

- OpenVR - not quite open source…
- But we do have the function interfaces.
- **Step 1**: Review documentation and header files for relevant data sources.

Table 3: OpenVR Data Structures and Enumerators

Data	**Contents**
TrackedDevicePose_t	Pose and status of tracked device
HmdMatrix34_t	Tracked device tranformation matrix
ETrackedDeviceClass	Type of Device
ETrackedDeviceProperty	Static device properties
EVRState	Status of the overall system
EVREventType	Event types
EDeviceActivityLevel	Level of Hmd activity
VREvent_Notification_t	Notification related events
VREvent_Overlay_t	Overlay Events
VREvent_Ipd_t	Ipd change

Challenges – Plan

- Memory forensics: The path forward (Case and Richard, 2017)
- Future directions
  - Application specific analysis
  - Rapid updates – 17 versions of SteamVR in June
- Complex and robust system, solely RE effort not practical.

Methodology – Scenario Creation

UNHcFREG

- How do we know what we are searching for in the first place?
- OpenVR Background application

Listing 1 : TrackedDeviePose_t

```
struct TrackedDevicePose_t
{
    HmdMatrix34_t mDeviceToAbsoluteTracking;
    HmdVector3_t vVelocity;
    HmdVector3_t vAngularVelocity;
    ETrackingResult eTrackingResult;
    bool bPoseIsValid;
    bool bDeviceIsConnected;
};
```

```
/** enum values to pass in to VR_Init to identify whether t
* draw a 3D scene. */
enum EVRApplicationType
{
    VRApplication_Other = 0,          // Some other kind of a
    VRApplication_Scene = 1,          // Application will sub
    VRApplication_Overlay = 2,        // Application only int
    VRApplication_Background = 3,     // Application should n
                                      // keep it running if e
    VRApplication_Utility = 4,        // Init should not try
                                      // interfaces (like IVR
    VRApplication_VRMonitor = 5,      // Reserved for vrmonit
    VRApplication_SteamWatchdog = 6,// Reserved for Steam
    VRApplication_Bootstrapper = 7, // Start up SteamVR

    VRApplication_Max
};
```

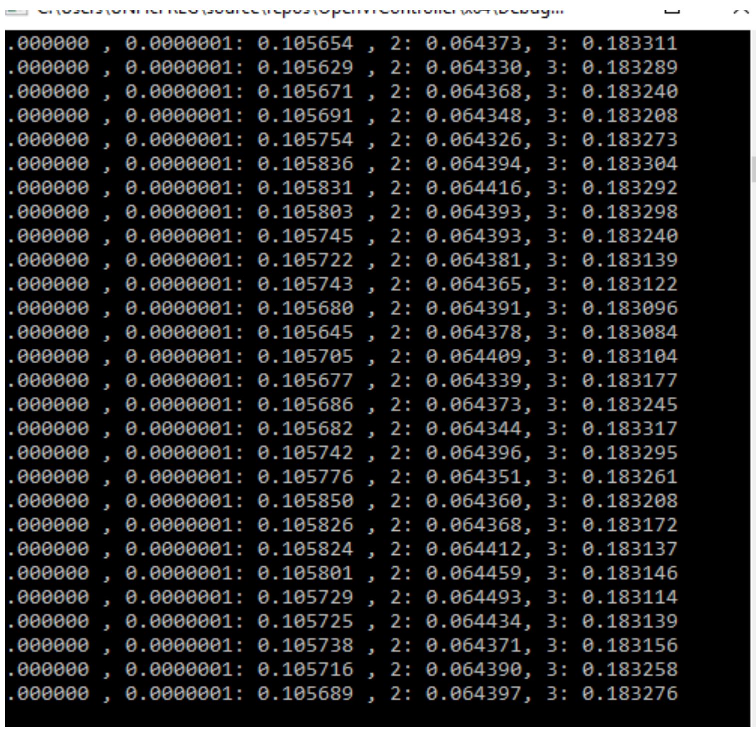
Methodology – Memory Scanning

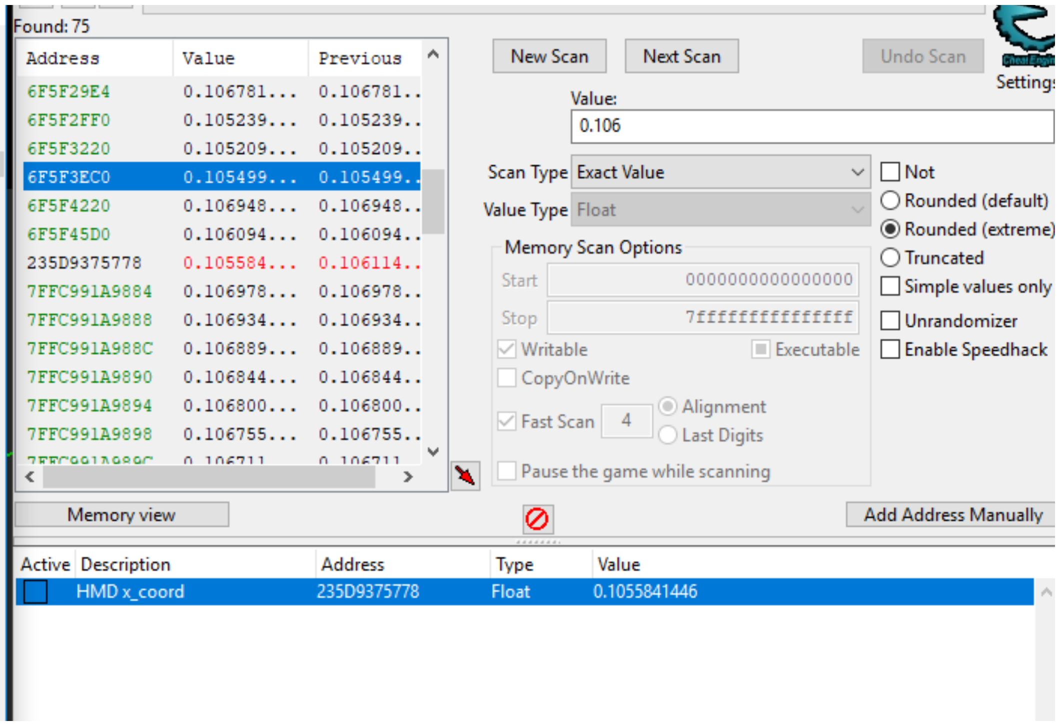
- Tracking system information is rapidly updated and has high precision.
- Cheat Engine – *"Cheat Engine is an open source tool designed to help you with modifying single player games…"*
  - **Powerful Memory Scanner**
  - Extensive Lua scripting support

Methodology – Memory Scanning

UNHcFREG

- Search
- Reduce
- Modify
- Repeat





Methodology – Static Reference

- How to reliably locate data in a memory dump?
- Work backwards from data to a static reference.
- Two strategies:
  - Attach debugger and monitor reads and writes to address
  - Search for pointer chains.

Base Address	Offset 0	Offset 1	Offset 2	Offset 3	Offset 4	Points to:
"vrmonitor.exe"+001F78A0	68					235D9375778
"THREADSTACK1"-00000...	2C0	0	50	8	68	235D9375778
"THREADSTACK1"-00000...	2C0	0	50	8	68	235D9375778
"THREADSTACK1"-00000...	2C0	0	50	8	68	235D9375778
"vrclient_x64.dll"+00220F...	60	30	50	8	68	235D9375778
"vrclient_x64.dll"+002278...	38	90	50	8	68	235D9375778
"vrmonitor.exe"+001F7798	668	90	50	8	68	235D9375778
"vrclient_x64.dll"+00227B...	18	C0	50	8	68	235D9375778
"vrclient_x64.dll"+00227B...	20	C0	50	8	68	235D9375778
"vrclient_x64.dll"+00227B...	18	C0	50	8	68	235D9375778
"vrclient_x64.dll"+002221...	D8	C0	50	8	68	235D9375778
"THREADSTACK1"-00000...	428	C0	50	8	68	235D9375778
"vrclient_x64.dll"+002250...	18	C0	50	8	68	235D9375778
"vrclient_x64.dll"+00223C...	618	C0	50	8	68	235D9375778
"vrclient_x64.dll"+002250...	20	C0	50	8	68	235D9375778
"vrclient_x64.dll"+00223C...	620	C0	50	8	68	235D9375778
"vrclient_x64.dll"+002250...	28	C0	50	8	68	235D9375778
"vrclient_x64.dll"+00223C...	628	C0	50	8	68	235D9375778
"vrclient_x64.dll"+002250...	30	C0	50	8	68	235D9375778
"vrclient_x64.dll"+00223C...	630	C0	50	8	68	235D9375778

Methodology – Static Reference

- References originating from the stack, unreliable.
- DLL functions operated on data passed to them
- Most reliable candidate pointer chains originate directly from executable – **vrmonitor.exe**

Listing 2 : Disassembly of Base Address Access

```
48 8b 05 25 d6 10 00        MOV RAX, [RIP+0x10d625]
48 85 c0                    TEST RAX, RAX
75 2a                       JNZ + 2a
snip
```

Methodology – Yara Signatures

- YARA - *"is a tool aimed at (but not limited to) helping malware researchers to identify and classify malware samples"*
- Develop signatures based on referencing op-codes.
- [Volatility Plugin & malware analysis]() – Thomas Chopitea

```
YARA_HMD = {
    'hmd_pointer': 'rule hmd_pointer { strings: $p = {48 8b 05 25 D6 10 00} condition: $p}',
}

YARA_HMD_ACTIVITY = {
    'hmd_activity': 'rule hmd_activity { strings: $p = {48 8b 05 55 A4 13 00} condition: $p}',
}
```

http://virustotal.github.io/yara/

Plugin Development

Algorithm 2 Vivedump: Locate target data

```
y ← yara.compile(rule)                         ▷ Base address signature
r ← [offset1, offset2, ...]                    ▷ Route
for task ∈ processList do
    if task = vrmonitor.exe then
        for vad ∈ getProcessVad do
            m ← y.match(vad)                   ▷ Address of YARA match
            for offset ∈ r do                  ▷ Traverse route
                m ← dereference(m) + offset
            end for
            return m                           ▷ Address of target data
        end for
    end if
end for
```

- Optimization – limit search to related process

http://virustotal.github.io/yara/

Data Interpretation

UNHcFREG

- C++ Standard – *"Non-static data members of a (non-union) class with the same access control are allocated so that later members have higher addresses within a class object."*
- Enumerators – if not explicitly set, assume default implementation

Listing 1 : TrackedDeviePose_t

```
struct TrackedDevicePose_t
{
      HmdMatrix34_t mDeviceToAbsoluteTracking;
      HmdVector3_t vVelocity;
      HmdVector3_t vAngularVelocity;
      ETrackingResult eTrackingResult;
      bool bPoseIsValid;
      bool bDeviceIsConnected;
};
```

```
enum ETrackingResult
{
    TrackingResult_Uninitialized            = 1,
    TrackingResult_Calibrating_InProgress   = 100,
    TrackingResult_Calibrating_OutOfRange   = 101,
    TrackingResult_Running_OK               = 200,
    TrackingResult_Running_OutOfRange       = 201,
};
```

Visualization

Data Reliability

- TrackedDevicePose array always available
- TrackedDevice Class route may be misleading
- Correlate device states between structures

Data Structure	YARA Signature	Route
TrackedDevicePose Array	48 8b 05 25 D6 10 00	0x10D62C
—- HmdMatrix34_t		+ 0x5C
—- ETrackingResult		+ 0xA4
—- bPoseIsValid		+ 0xA8
—- bDeviceIsConnected		+ 0xAA
HMD State	48 8b 05 55 A4 13 00	0x13A45C, 0x1E0, 0x 0B8, 0x68
—- HMD Activity		+ 0x10
TrackedDevice Class	48 8b 05 55 A4 13 00	0x13A45C, 0x190, 0x8, 0x510, 0x3E0, 0x120

Future Work

- Expand supported recoverable features
- Version Detection
- Application update recovery
- Emulation

Questions

Peter Casey

(pgrom1@unh.newhaven.edu)

Thanks to the National Science Foundation,

University of New Haven,

DFRWS, and MITRE.