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)

DFRWS is dedicated to the sharing of knowledge and ideas about digital forensics research. Ever since it organized the first open workshop devoted to digital forensics in 2001, DFRWS continues to bring academics and practitioners together in an informal environment. As a non-profit, volunteer organization, DFRWS sponsors technical working groups, annual conferences and challenges to help drive the direction of research and development.

https://dfrws.org
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
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

• 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

- **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

<table>
<thead>
<tr>
<th>Data</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>TrackedDevicePose_t</td>
<td>Pose and status of tracked device</td>
</tr>
<tr>
<td>HmdMatrix34_t</td>
<td>Tracked device transformation matrix</td>
</tr>
<tr>
<td>ETrackedDeviceClass</td>
<td>Type of Device</td>
</tr>
<tr>
<td>ETrackedDeviceProperty</td>
<td>Static device properties</td>
</tr>
<tr>
<td>EVRState</td>
<td>Status of the overall system</td>
</tr>
<tr>
<td>EVREventType</td>
<td>Event types</td>
</tr>
<tr>
<td>EDeviceActivityLevel</td>
<td>Level of Hmd activity</td>
</tr>
<tr>
<td>VREvent_Notification_t</td>
<td>Notification related events</td>
</tr>
<tr>
<td>VREvent_Overlay_t</td>
<td>Overlay Events</td>
</tr>
<tr>
<td>VREvent_Ipd_t</td>
<td>Ipd change</td>
</tr>
</tbody>
</table>
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

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

Listing 1: TrackedDevicePose_t

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

- 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.

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

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>48</td>
<td>8b</td>
<td>05</td>
<td>25</td>
<td>d6</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>48</td>
<td>85</td>
<td>c0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>75</td>
<td>2a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>snip</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
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

```ruby
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

- Optimization – limit search to related process

Algorithm 2 Vivedump: Locate target data

1: $y \leftarrow \text{yara.compile(rule)}$  \hspace{1cm} \triangleright \text{Base address signature}
2: $r \leftarrow [\text{offset1}, \text{offset2}, \ldots]$  \hspace{1cm} \triangleright \text{Route}
3: \textbf{for} task $\in \text{processList}$ \textbf{do}
4: \hspace{1cm} \textbf{if} task = \text{vrmonitor.exe} \textbf{then}
5: \hspace{2cm} \textbf{for} vad $\in \text{getProcessVad}$ \textbf{do}
6: \hspace{4cm} $m \leftarrow y.\text{match(vad)}$  \hspace{1cm} \triangleright \text{Address of YARA match}
7: \hspace{1cm} \hspace{1cm} \textbf{for} offset $\in r$ \textbf{do}
8: \hspace{3cm} $m \leftarrow \text{dereference}(m) + \text{offset}$  \hspace{1cm} \triangleright \text{Traverse route}
9: \hspace{2cm} \textbf{end for}
10: \hspace{1cm} \textbf{return} $m$  \hspace{1cm} \triangleright \text{Address of target data}
11: \textbf{end for}
12: \textbf{end if}
13: \textbf{end for}

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

• 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

```c
Listing 1: TrackedDevicePose_t
struct TrackedDevicePose_t
{
    HmdMatrix34_t mDeviceToAbsoluteTracking;
    HmdVector3_t vVelocity;
    HmdVector3_t vAngularVelocity;
    ETrackingResult eTrackingResult;
    bool bPoseIsValid;
    bool bDeviceIsConnected;
};
```

```c
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
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.