Steal this Movie

Automatically Bypassing DRM Protection in Streaming Media Services

Ruoyu ‘Fish’ Wang\textsuperscript{12}, Yan Shoshitaishvili\textsuperscript{1},
Christopher Kruegel\textsuperscript{1}, Giovanni Vigna\textsuperscript{1}

\textsuperscript{1} UC Santa Barbara
\textsuperscript{2} Tsinghua University
Digital Rights Management
How DRM works

Encrypted Information Flow → Demultiplexing → Encrypted Media Stream → Decrypt → Decrypted Media Stream → Decode → Raw Frames → Playback

DRM Mechanisms
DRM bypasses

• What for?

• Solution-specific
  • DeCSS - DVD Jon
  • Despotify
  • HDCP master key leaking

• Analog loophole
How DRM works

DRM Mechanisms

Encrypted Information Flow → Demultiplexing → Encrypted Media Stream → Decrypt → Decrypted Media Stream

Solution-specific attacks

Analog loophole 1

Analog loophole 2

MovieStealer

Raw Frames → Decode → Playback
Automatic attacking
Challenges

• Complexity

• Performance

• Generality
Intuitions

Final goal:
Identify the decrypted stream and dump it!
Overview

• MovieStealer design & optimizations
• Experimental results
• Countermeasures
• Ethics and legality
MovieStealer Design
Goal: find the decrypted stream!

- Loop detection
- Buffer detection
- Data-paths
- Statistical analysis
- Dumping & rebuilding
Approach overview

Goal: find the decrypted stream!

- Loop detection
- Buffer detection
- Data-paths
- Statistical analysis
- Dumping & rebuilding
Loop detection (1)

- Based on LoopProf*

---

Loop detection (2)

• Handling unrolled loops

BasicBlock B
BasicBlock C
BasicBlock D

BasicBlock E
Unrolled

BBLs accessing the same buffer with similar patterns
Loop and call-path

```c
void crypto_loop(const char *key, void *in, void *out, int len);

void encrypt() {
    crypto_loop("key", dec, enc, len);
}

void decrypt() {
    crypto_loop("key", enc, dec, len);
}
```
Approach review

Goal: find the decrypted stream!

- Loop detection
- Buffer detection
- Data-paths
- Statistical analysis
- Dumping & rebuilding

2013/8/23
Buffer detection

- Reason about buffers based on access patterns
  - Consecutive bytes
  - Inconsecutive blocks

<table>
<thead>
<tr>
<th>Address</th>
<th>Buffer Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1000</td>
<td>Original buffer</td>
</tr>
<tr>
<td>0x1004</td>
<td>Original buffer</td>
</tr>
<tr>
<td>0x1008</td>
<td>Original buffer</td>
</tr>
<tr>
<td>0x100c</td>
<td>Original buffer</td>
</tr>
<tr>
<td>0x1010</td>
<td>Original buffer</td>
</tr>
<tr>
<td>0x1014</td>
<td>Original buffer</td>
</tr>
<tr>
<td>0x1018</td>
<td>Original buffer</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Buffer Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1000</td>
<td>Composite buffer</td>
</tr>
<tr>
<td>0x1004</td>
<td>Composite buffer</td>
</tr>
<tr>
<td>0x1008</td>
<td>Composite buffer</td>
</tr>
<tr>
<td>0x100c</td>
<td>Composite buffer</td>
</tr>
<tr>
<td>0x1010</td>
<td>Original buffer</td>
</tr>
<tr>
<td>0x1014</td>
<td>Original buffer</td>
</tr>
<tr>
<td>0x1018</td>
<td>Original buffer</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Buffer Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1000</td>
<td>Composite buffer</td>
</tr>
<tr>
<td>0x1004</td>
<td>Composite buffer</td>
</tr>
<tr>
<td>0x1008</td>
<td>Composite buffer</td>
</tr>
<tr>
<td>0x100c</td>
<td>Composite buffer</td>
</tr>
<tr>
<td>0x1010</td>
<td>Composite buffer</td>
</tr>
<tr>
<td>0x1014</td>
<td>Composite buffer</td>
</tr>
<tr>
<td>0x1018</td>
<td>Composite buffer</td>
</tr>
</tbody>
</table>
Approach review

Goal: find the decrypted stream!

- Loop detection
- Buffer detection
- Data-paths
- Statistical analysis
- Dumping & rebuilding
Data-paths

Identify *paths* through a loop which modify the input data to output data

A sensible data-path, find it!
Approach review

Goal: find the decrypted stream!

- Loop detection
- Buffer detection
- Data-paths
- Statistical analysis
- Dumping & rebuilding

Randomness

2013/8/23

Usenix Security 2013
Statistical analysis (1)

• Cipher-text indistinguishability
  • Basic requirement for secure cryptosystems

• Entropy should be pretty high, as data is from Internet
### Statistical analysis (2)

<table>
<thead>
<tr>
<th>Stage</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Entropy</td>
<td>Randomness</td>
</tr>
<tr>
<td>Download</td>
<td><img src="up.png" alt="Up" /> <img src="up.png" alt="Up" /></td>
<td><img src="up.png" alt="Up" /></td>
</tr>
<tr>
<td>Decrypt</td>
<td><img src="up.png" alt="Up" /> <img src="up.png" alt="Up" /></td>
<td><img src="down.png" alt="Down" /></td>
</tr>
<tr>
<td>Decode</td>
<td><img src="up.png" alt="Up" /></td>
<td><img src="down.png" alt="Down" /></td>
</tr>
</tbody>
</table>
Approach review

Goal: find the decrypted stream!

- Loop detection
- Buffer detection
- Data-paths
- Statistical analysis
- Dumping & rebuilding
Dumping and reconstruction

* Intentionally omitted
Workflow

Start playing the video

Make *MovieStealer* begin analysis

Wait for a few minutes

*Harvest*
MovieStealer Optimizations
Problem of the basic approach

- Too much overhead!
- Won’t sniff enough data
- Media players don’t function normally
- Some media players check the performance
- Might get caught by checking systems of DRM
Optimizations

Goal: minimize overheads!
• Improved loop selection
• Efficient loop analysis
• On-Demand instrumentation
• Execution frequency
• Instruction analysis
• Bandwidth filtering
• Copying optimizations
• **Callstack key**

```
on_enter
callstack_key ^= func_addr
```

```
on_exit
callstack_key ^= func_addr
```
void crypto_loop(const char *key, void *in, void *out, int len);

void encrypt() {
    crypto_loop("key", dec, enc, len);
}

void decrypt() {
    crypto_loop("key", enc, dec, len);
}

\*on\_enter
callstack\_key ^= func\_addr

\*on\_exit

callstack\_key ^= func\_addr
Experimental Results
Implementation

• Dynamic binary instrumentation (DBI)
  • Intel PIN framework

• Under Windows 7 32-bit

• Testing
  • A common workstation
Evaluation

- *GnuPG* for testing optimizations

- Three DRM platforms
  - *Microsoft PlayReady* (Netflix)
  - *Adobe RTMPE* (*Hulu* and *Amazon Video*)
  - *Spotify's* music protection
## Results - GPG

<table>
<thead>
<tr>
<th>Optimizations</th>
<th>Loops Instrumented</th>
<th>Seconds Elapsed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Only execution frequency</td>
<td>40</td>
<td>3480 (112x)</td>
</tr>
<tr>
<td>Only bandwidth filtering</td>
<td>35</td>
<td>180 (5.8x)</td>
</tr>
<tr>
<td>Only instruction analysis</td>
<td>10</td>
<td>49 (1.58x)</td>
</tr>
<tr>
<td>All but callstack key</td>
<td>6</td>
<td>47 (1.51x)</td>
</tr>
<tr>
<td>All enabled</td>
<td>7</td>
<td>31</td>
</tr>
</tbody>
</table>
## Results - DRM

<table>
<thead>
<tr>
<th>Platform</th>
<th>Protection</th>
<th>Loops Instrumented</th>
<th>Loops Traced</th>
<th>Buffers Identified</th>
<th>Seconds Elapsed*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Netflix</td>
<td><em>Dynamic code</em></td>
<td>2274</td>
<td>58</td>
<td>80</td>
<td>110</td>
</tr>
<tr>
<td>Hulu</td>
<td>-</td>
<td>1529</td>
<td>46</td>
<td>14</td>
<td>281</td>
</tr>
<tr>
<td>Amazon Video</td>
<td>-</td>
<td>1258</td>
<td>35</td>
<td>6</td>
<td>146</td>
</tr>
<tr>
<td>Spotify</td>
<td><em>Packing and self-checking</em></td>
<td>2305</td>
<td>224</td>
<td>60</td>
<td>536</td>
</tr>
</tbody>
</table>

* seconds elapsed before MovieStealer breaks the DRM protection
Countermeasures
Countermeasures

• Attack the instrumentation
  • Anti-debugging

• Attack the loop detection
  • VM’ing those loops

• Attack the buffer detection
  • Non-consecutive buffer layouts

• Anti-piracy
  • Watermarking
Ethics and Legality

I care about copyright. I didn't know why people here are happy to buy a CD or DVD or something that is not original. I mean why didn't they care about the people who make that thing, and put the money of right notice but just "yeah you have the right, not my right"... I hope that people that don't respect the author can change their mind to buy the original if piracy has grown bigger than ever, I don't know how this world can be.
Ethics

• Responsible disclosure
  • Contacted Microsoft, Spotify, Adobe, Amazon, Netflix, and Hulu
  • Microsoft, Spotify, and Adobe responded
    • Tested MovieStealer
    • Confirmed DRM bypass
    • Provided comments and encouraged publication

• Some details omitted (e.g. reconstruction)

• No tool/source release
Legality

• We believe this work to be legal under DMCA
  • Consulted with UC counsel and the EFF
  • Thank you all
Acknowledgement

• Thanks for support from Microsoft, Adobe, and Spotify

• Thanks Kevin Borgolte, Yanick Fratantonio, Christian Kreibich, and Thorsten Holz for presentation advice
Contact info

Send us an email

fish@cs.ucsb.edu, yans@cs.ucsb.edu, chris@cs.ucsb.edu, vigna@cs.ucsb.edu
Question time

Questions?