MANDIANT®

EVTXtract

Recovering EVTX Records from Unallocated Space

PRESENTED BY: Willi Ballenthin

OCT 6, 2013

What do we have here today?

- *A technical presentation on a novel forensic technique for recovering past user activity*
- Beginning
  - Introductions
  - LfLe project and results
- Middle
  - EVTX files & theory
  - EVTXtract tools
- End
  - Questions
  - Answers

Willi Ballenthin

- **Professionally**: Incident response & malware analysis @ Mandiant

- **Personally**: IR & computer forensic tool dev

LfLe

LfLe.py and Friends

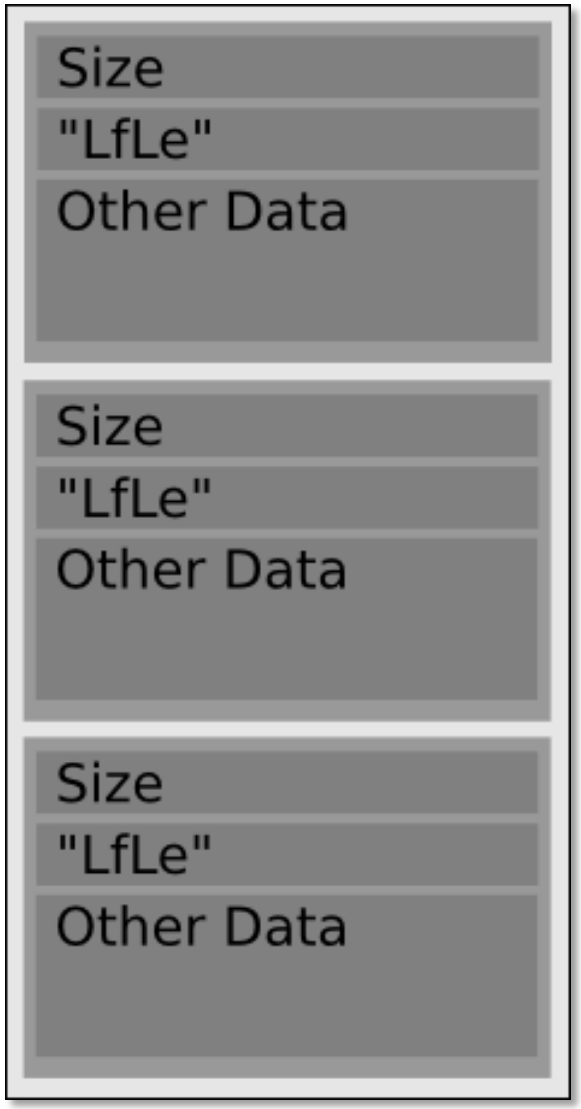
- Carve EVT event log records
  - input: arbitrary binary data (memory, disk, VM)
  - output: reconstructed EVT log file, CSV, etc.

- Sources
  - https://github.com/williballenthin/LfLe
  - *Carve for Records (Not Files)* - Jeff Hamm @ SANS DFIR Summit 2012

- Easter egg: Python module for EVT manipulation

LfLe.py in 6 steps

1. Scan for “LfLe”
2. Seek back 0x4 bytes
3. Read record size
4. Extract buffer
5. Sanity check
6. Dump to file



LfLe.py in 6 steps

1. Scan for “LfLe”
2. Seek back 0x4 bytes
3. Read record size
4. Extract buffer
5. Sanity check
6. Dump to file

```
0000000: 3000 0000 4c66 4c65 0100 0000 0100 0000  0...LfLe........
0000010: d032 0000 142f 0000 9321 0000 0f00 0000  .2.../...!......
0000020: 0000 0100 0600 0000 8051 0100 3000 0000  .........Q..0...
0000030: 4100 5400 4100 3e00 3c00 4300 5500 5200  A.T.A.>.<.C.U.R.
0000040: 5200 4500 4e00 5400 4700 5200 4f00 5500  R.E.N.T.G.R.O.U.
0000050: 5000 3e00 5500 7300 6500 7200 7300 3c00  P.>.U.s.e.r.s.<.
0000060: 2f00 4300 5500 5200 5200 4500 4e00 5400  /.C.U.R.R.E.N.T.
0000070: 4700 5200 4f00 5500 5000 3e00 0000 0000  G.R.O.U.P.>.....
0000080: a804 0000 a804 0000 4c66 4c65 8921 0000  ........LfLe.!..
```

Why you should always use LfLe

- LfLe.py can recover event log records from unallocated space
  - when a bad person clears the log
  - when the log rolls over
- Records recovered during recent investigation: 30,000
- Time to run: 10 minutes

EVTX Event Logs

- Used in Vista and above

```
working-private/complete_evtx - [master●] » xxd complete_evtx.evtx| head -n 2
0000000: 456c 6646 696c 6500 0000 0000 0000 0000  ElfFile.........
0000010: 0700 0000 0000 0000 cb02 0000 0000 0000  ................
```

- Event log divided into 64KB chunks

```
working-private/complete_evtx - [master●] » xxd -s $((4096)) complete_evtx.evtx|
 head -n 2
0001000: 456c 6643 686e 6b00 0100 0000 0000 0000  ElfChnk.........
0001010: 6700 0000 0000 0000 0100 0000 0000 0000  g...............
```

- Logical records are XML, records on disk are binary XML

```
0001200: 2a2a 0000 1806 0000 0100 0000 0000 0000  **..............
0001210: 7994 3161 7386 cd01 0f01 0100 0c01 6dca  y.1as.........m.
0001220: 72c7 2602 0000 0000 0000 6dca 72c7 cdb8  r.&.......m.r...
```

EVTX Tools

- **Parse-Evtx**
  - Andreas Schuster
  - ★★★★★ (five stars)!
  - Perl
- **libevtx**
  - Joachim Metz
  - C++
- **MS Log Parser**
  - standalone, Windows
- **Event Log Explorer**
  - standalone, Windows

Technical Details

EVT Event Log “Compression”

- EVT files use external resources
  - DLL PE resources contain “templates”
  - EVT records contain a list of strings
  - template + substitutions == record

- Great for localization
- Good for file size
- Easy for investigators

EVTX Event Log “Compression”

- EVTX files use “structural compression”
  - The Event Log system recognizes repeated patterns of data and “factors them out”

- Much better flexibility for log messages
- Log is now self contained*
- Not great for file size
- Bad for investigators (until now?)

```
<Event>
  <EID>100</EID>
  <EventData>
    <LogonService>Willi logged in</LogonService>
  </EventData>
</Event>

<Event>
  <EID>101</EID>
  <EventData>
    <LogonService>Willi logged out</LogonService>
  </EventData>
</Event>

<Event>
  <EID>200</EID>
  <EventData>
    <iPodService>Music Transferred</iPodService>
  </EventData>
</Event>
```

```
<Event>
  <EID>100</EID>
  <EventData>
    <LogonService>
      Willi logged in
    </LogonService>
  </EventData>
</Event>

<Event>
  <EID>101</EID>
  <EventData>
    <LogonService>
      Willi logged out
    </LogonService>
  </EventData>
</Event>

<Event>
  <EID>200</EID>
  <EventData>
    <iPodService>
      Music Transferred
    </iPodService>
  </EventData>
</Event>
```

Template 1:

```
<Event>
  <EID>[Substitution #1]</EID>
  <EventData>
    [Substitution #2]
  </EventData>
</Event>
```

Template 2:

```
<LogonService>
  [Substitution #1]
</LogonService>
```

Template 3:

```
<iPodService>
  [Substitution #1]
</iPodService>
```

```
<Event>
  <EID>100</EID>
  <EventData>
    <LogonService>
      Willi logged in
    </LogonService>
  </EventData>
</Event>

<Event>
  <EID>101</EID>
  <EventData>
    <LogonService>
      Willi logged out
    </LogonService>
  </EventData>
</Event>

<Event>
  <EID>200</EID>
  <EventData>
    <iPodService>
      Music Transferred
    </iPodService>
  </EventData>
</Event>
```

Template 1:

```
<Event>
  <EID>[Substitution #1]</EID>
  <EventData>
    [Substitution #2]
  </EventData>
</Event>
```

Template 2:

```
<LogonService>[Substitution #1]
</LogonService>
```

Template 3:

```
<iPodService>[Substitution #1]
</iPodService>
```

```
TMPT-1(100, TMPT-2(Willi logged in))

TMPT-1(101, TMPT-2(Willi logged out))

TMPT-1(200, TMPT-3(Music transferred))
```

More on EVTX Templates

- Templates
  - are local to a 64KB chunk
    - If we have an entire chunk, we can completely recover it. Checksums verify contents.
  - **may be** created dynamically
  - are defined “resident” within the first related record
  - can be referenced from subsequent records by a relative pointer
- Substitutions
  - values have a specific type (23 available)
  - can be *conditional* or *required*

Without Templates: typed values, but **no context!**

```
WstringTypeNode(offset=0x99) --> Microsoft-Windows-Security-Auditing
GuidTypeNode(offset=0xdf) --> {54849625-5478-4994-a5ba-3e3b0328c30d}
WstringTypeNode(offset=0xef) --> Security
BXmlTypeNode(offset=0xff) --> RootNode(offset=0xb477, length=0x112)
  RootNode(offset=0xff)
    StreamStartNode(offset=0xff)
    TemplateInstanceNode(offset=0x103, resident=False)
    Substitutions(offset=0x10d)
      SIDTypeNode(offset=0x161) --> S-1-0-0
      WstringTypeNode(offset=0x16d) --> -
      WstringTypeNode(offset=0x16f) --> -
      Hex64TypeNode(offset=0x171) --> 0x0000000000000000
      SIDTypeNode(offset=0x179) --> S-1-5-7
      WstringTypeNode(offset=0x185) --> ANONYMOUS LOGON
      WstringTypeNode(offset=0x1a3) --> NT AUTHORITY
      Hex64TypeNode(offset=0x1bb) --> 0x0000000000021661
      UnsignedDwordTypeNode(offset=0x1c3) --> 3
      WstringTypeNode(offset=0x1c7) --> NtLmSsp
      WstringTypeNode(offset=0x1d7) --> NTLM
```

Key Intuition

Templates may be dynamically created, but…

in a given configuration, templates do not often change.

Therefore, we can build up a collection of templates relevant to a case and *try* to reuse them.

EVTXtract

EVTXtract

- EVTXtract is
  - a technique for building a database of templates
  - a tool for extracting potential records
  - a method for fitting the parts back together
  - open source, Python, available on Github

What can we expect?

- Best Case
  - fully reconstruct all EVTX records in an image
  - result is a single XML file
- Worst Case
  - extract only substitution arrays, no structure/context
  - analyst must manually interpret values
  - result is a JSON file containing all known content & metadata
- Average Case
  - fully reconstruct most EVTX records to an XML file
  - analyst must review a few remaining values

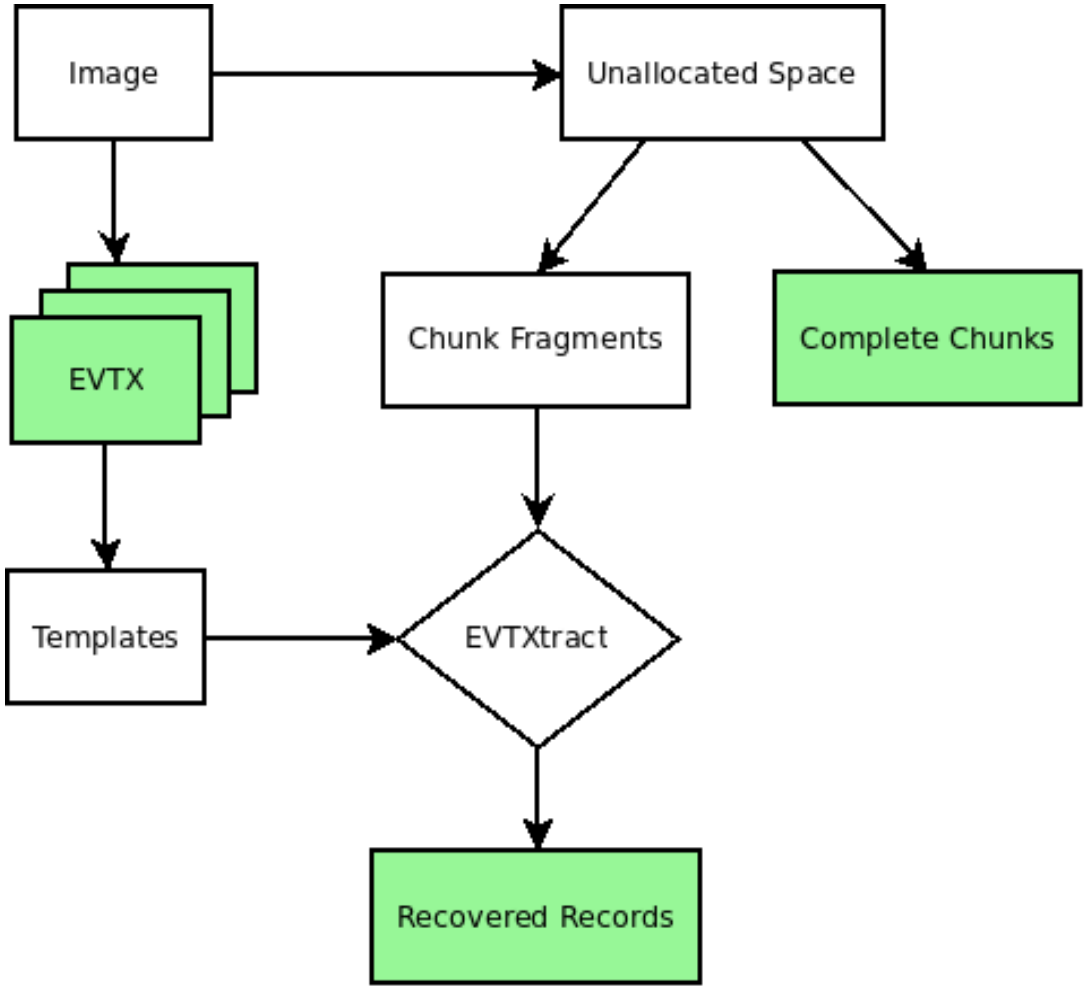
Worst case: Raw data with no context

```
WstringTypeNode(offset=0x99) --> Microsoft-Windows-Security-Auditing
GuidTypeNode(offset=0xdf) --> {54849625-5478-4994-a5ba-3e3b0328c30d}
WstringTypeNode(offset=0xef) --> Security
BXmlTypeNode(offset=0xff) --> RootNode(offset=0xb477, length=0x112)
  RootNode(offset=0xff)
    StreamStartNode(offset=0xff)
    TemplateInstanceNode(offset=0x103, resident=False)
    Substitutions(offset=0x10d)
      SIDTypeNode(offset=0x161) --> S-1-0-0
      WstringTypeNode(offset=0x16d) --> -
      WstringTypeNode(offset=0x16f) --> -
      Hex64TypeNode(offset=0x171) --> 0x0000000000000000
      SIDTypeNode(offset=0x179) --> S-1-5-7
      WstringTypeNode(offset=0x185) --> ANONYMOUS LOGON
      WstringTypeNode(offset=0x1a3) --> NT AUTHORITY
      Hex64TypeNode(offset=0x1bb) --> 0x0000000000021661
      UnsignedDwordTypeNode(offset=0x1c3) --> 3
      WstringTypeNode(offset=0x1c7) --> NtLmSsp
      WstringTypeNode(offset=0x1d7) --> NTLM
```

Best case: Fully reconstructed XML document

```
<?xml version="1.0" encoding="UTF-8"?>
<Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
  <System>
    <Provider Name="Microsoft\-Windows\-Security\-Auditing" Guid="0001010f\-010c\-
ca6d\-72c7\-260200000000" />
    <EventID Qualifiers="None">4608</EventID>
    <Version>0</Version>
    <Level>0</Level>
    <Task>12288</Task>
    <Opcode>0</Opcode>
    <Keywords>0x8020000000000000</Keywords>
    <TimeCreated SystemTime="2012\-08\-30T05\:50\:37\.361267Z" />
    <EventRecordID>1</EventRecordID>
    <Correlation ActivityID="None" RelatedActivityID="None" />
    <Execution ProcessID="460" ThreadID="464" />
    <Channel>Security</Channel>
    <Computer>WILLI-DEV</Computer>
    <Security UserID="None" />
  </System>
```

EVTXtract workflow



EVTXtract Template Matching

- Each event ID (EID) may have several associated templates, so how to pick the correct one?
  - **Substitutions are typed**, so create a signature for each template:

    ```
    EID + #subs + type(sub1) + ...
    ```

  - Given recovered record, easy to match constraints against template database
- Conflicts still arise, so manual resolution required

Thinking Bigger

- Can we build a global database of templates?
- Can we make this go faster?
- Can we integrate this technique into more tools?
  - Autopsy, EnScripts, etc.

Questions?