

Windows 7/2008 Event Log forensic and reversing analysis  
eseugutroP Reversed  
2011/03/16 by ar1vr

This text refers to the 32bit version of Windows, unfortunately I don't have access to a 64bit development environment.

Opening an .evtx log is pretty straightforward, the only problem is that the *eventlog* service opens this files exclusively and if we try to make a copy, we'll get "Access is denied" error. So, we need to get them with the machine offline.

One of the reasons beyond the handles to this files being exclusively open, is that the *eventlog* service memory maps chunks of this files, holding and manipulating some housekeeping metadata information directly in the memory mapped files.

Performance and security issues also come to the discussion: the files aren't always in sync with the data in memory for example, and each file has an associated timer that CRCs the headers and flushes the mapped views. Also this metadata information allows for file recovery in case of unexpected shutdown or crash.

What this means for a "interested" person, is that we need to be careful if trying to change the logs content online, because not everything is what it seems. But it is indeed possible to manipulate the logs directly in memory: remove and change log entries, insert new entries, stealthily clear the logs, add log garbage, etc.

I wrote a small POC tool that shows this being done, I called it *Elchomp*. *Elchomp* basically clears the last log record entry in the event log of choosing. *Elchomp* is available with this document or at <https://www.filesanywhere.com/fs/v.aspx?v=8a6b66865967747da3a5>.

The .evtx files are binary files, to where the *eventlog* service streams log records using XML templates to format each entry. This makes easier switching between the Event Viewer general and details panels.

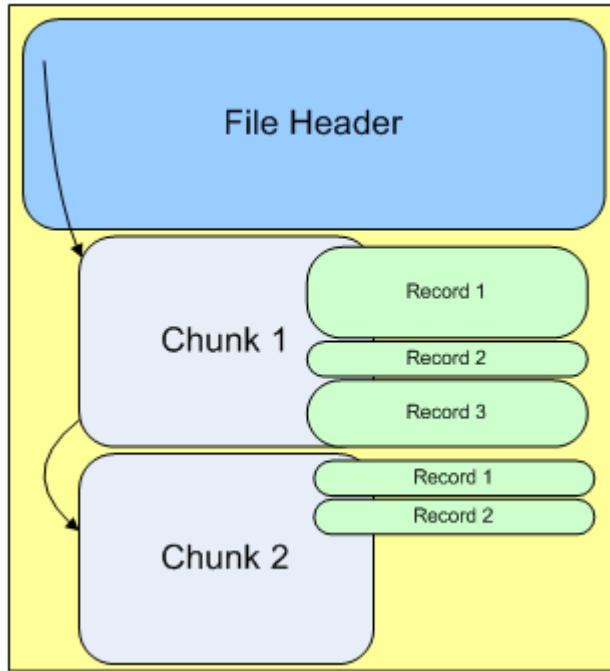
These binary files are structured as follows:

- The file starts with a file header that describes the chunks used by the file. The stored information pertains to the number of chunks, the chunk size, and some duplicated information for failover purposes. A chunk is a container block of event records.

- N Chunks serialized. Each chunk points to a set of event log entries.
- N records. Each record represents an event log entry.

Let's view in more detail each of these structures. I'll cover only the most important data, so keep this in mind while reading. I'll open a real .evtx log for analysis. Figure 2 shows the first bytes of the file.

Figure 1: Event log layout



As can be seen, there are a couple of signatures in the file, the first signature "ElfFile" at position zero in the file, marks the beginning of the file, and is used to detect if this is an event log file type.

Figure 2: File Header

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
000000000	45	6c	66	46	69	6c	65	00	00	00	00	00	00	00	00	00
000000010	1c	00	00	00	00	00	00	00	ea	0d	00	00	00	00	00	00
000000028	80	00	00	00	01	00	03	00	00	10	1d	00	00	00	00	00
000000030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000050	00	10	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000000070	00	00	00	00	00	00	00	00	01	00	00	00	7d	8a	51	72

FileHeader   
 CurrentChunkCount   
 SizeOfHeader   
 NumberOfChunks

At offset 0x10 we have the *CurrentChunkCount*. This DWORD value doesn't show up in the *eventlog* file, I just mention it here because I wanted to enforce the differences between the offline and online layouts. When the log is opened by the *eventlog* service, the *CurrentChunkCount* is set to the chunk that the *NumberOfChunks* references, a new chunk is created and the *ChunkOffset* is updated accordingly.

At offset 0x28, *NumberOfChunks*, a WORD value, indicates how many chunks are in the file.

At offset 0x30, *SizeOfHeader* a WORD value, sets the size for the file header.

Positioning to the end of the File header, value 0x1000, we can see the first chunk

signature. (Figure 3)

Figure 3: First Chunk

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00000ff0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00001000	45	6c	66	43	68	6e	6b	00	01	00	00	00	00	00	00	lfChnk.....
00001010	76	00	00	00	00	00	00	00	01	00	00	00	00	00	00	v.....

It's time to say that the chunks are fixed sized  $0x10000 = 65536$  bytes. The last chunk is always memory mapped in the *eventlog* service process.

Multiplying size of chunk by *NumberOfChunks* we get  $0x10000 * 0x001d = 0x1d0000$ , sum the *FileHeader* and we get to last valid chunk header:  $0x1d0000 + 0x1000 = 0x1d1000$ . (Figure 4)

Figure 4: Last Chunk Header Data

001d1000	45 6c 66 43	68 6e 6b 00	ee 0d 00 00	00 00 00 00	ElfChnk.i.....
001d1013	15 0e 00 00	00 00 00 00	ee 0d 00 00	00 00 00 00	...i.....
001d1020	15 0e 00 00	00 00 00 00	80 00 00 00	40 65 00 00	.....€...@e..
001d1030	b8 66 00 00	9d e6 c9 22	00 00 00 00	00 00 00 00	,f..0æÉ".....

"ElfChnk" at position zero in the *ChunkHeader*, validates the chunk beginning.

At offset 0x10 and 0x20 we've got the same value, for failsafe purpose. It's a DWORD (0x00000e15) that represents the number of records contained in the chunk. Actually, if a new record is written it will use this value as Record counter, so the Number of records can be calculated by subtracting one to this value.

At offset 0x30 a DWORD that indicates the last offset position in the chunk that can be written to.

Figure 5: Last valid position in log

*LastChunkStart* = 0x001d1000

*LastValidOffset* = 0x000066b8

*LastValidPosition* = 0x001d1000 + 0x000066b8 = 0x001d76b8

From position 0x001d76b8 forward, there are only zeros. (Figure 5)

The last valid position also gives us the last valid record end. going backwards, we reach the signature (position 0x1d7438 in figure 6) for the last event record '\*' or 0x2A2A.

Figure 6: Event Record Data

	RecordCount	RecordsID	EventData	recordSeparator	timestamp	eventID	Keywords	SID
001d7420	00 74	00 65	00 73	00 74	00 65	00 00	00 00	00 00
001d7430	00 00	00 00	08 01	00 00	2a 2a	00 00	08 01	00 00
001d7440	14 0e	00 00	00 00	00 00	80 e9	b2 ce	58 da	cb 01
001d7450	0f 01	01 00	0c 01	f7 9f	ae 5a	8e 5f	00 00	14 00
001d7460	00 00	01 00	04 00	01 00	04 00	02 00	06 00	02 00
001d7470	06 00	02 00	06 00	08 00	15 00	08 00	11 00	00 00
001d7480	00 00	04 00	08 00	04 00	08 00	08 00	0a 00	01 00
001d7490	04 00	1c 00	13 00	00 00	00 00	00 00	00 00	00 00
001d74a0	00 00	00 00	00 00	00 00	00 00	00 00	00 00	43 00
001d74b0	21 00	02 00	00 00	e7 03	00 00	00 00	00 00	00 00
001d74c0	80 00	80 e9	b2 ce	58 da	cb 01	00 00	00 00	00 00
001d74d3	00 00	14 0e	00 00	00 00	00 00	00 01	05 00	00 00
001d74e0	00 00	05 15	00 00	00 82	[REDACTED]	[REDACTED]	[REDACTED]	[REDACTED]
001d74f0	[REDACTED]	[REDACTED]	[REDACTED]	[REDACTED]	01 01	00 0c	01 01	46 d3
001d7500	ec 16	08 00	00 03	00 00	00 20	00 81	00 04	00 08
001d7510	00 00	00 00	00 69	00 73	00 74	00 6f	00 20	00 65
001d7520	00 20	00 75	00 6d	00 20	00 74	00 65	00 73	00 74
001d7530	00 65	00 00	00 00	00 00	00 00	00 00	08 01	00 00

At offset 0x6 we find the current record count 0xe14 (being this the last valid record in the chunk this value is equal to the chunk's number of records minus one).

At offset 0xe and 0x86 the timestamp in file format. Corresponds to the event viewer's "Logged" time field.

At offset 0x7a the Event Id.

At offset 0x7e the Keywords field.

At offset 0x96 the record Id.

At offset 0x9e the SID.

At offset 0xda we've got the log message.

Figure 7: Event Viewer log entry

teste	
<u>Log Name:</u> Application	
<u>Source:</u> EventCreate	<u>Logged:</u> 15-03-2011 20:48:38
<u>Event ID:</u> 999	<u>Task Category:</u> None
<u>Level:</u> Error	<u>Keywords:</u> Classic

Hope you enjoyed