EXT3 File Recovery via Indirect Blocks

Hal Pomeranz
Deer Run Associates
Agenda

• EXT File System Review
• Recovering Deleted Data
  – Issues with Traditional File Carving Tools
  – How Indirect Blocks Can Be Leveraged
  – Tools to Recover Data
• Wrap Up
  – Looking Ahead to EXT4
EXT File System Review
Metadata Layer

• Inodes store typical file metadata:
  – File permissions
  – Ownership info
  – File size, number of links, etc
  – MAC timestamps

• Inode also has fixed number of pointers to the file content (data blocks)…
Direct Blocks (12) [File sizes up to 48K]

Indirect Blocks (1024) [4M storage]

Double Indirect (1M) [4GB storage]

Trible Indirect Pointer [Up to 1G blocks, 4TB storage]
A Word About Data Blocks

- Data blocks (and inodes) organized into logical "Block Groups" (typically 32K blocks/group)
- Contents of a directory will be allocated to the same block group
- Blocks in a file will be allocated consecutively, if possible, using "first-available" algorithm
- Slack space is null-filled
File Deletion in EXT
Data vs. Metadata

• Data blocks are simply marked as unallocated
  – Content remains on disk until blocks re-used

• Treatment of metadata varies by EXT version
  – EXT2: Simply mark inode as unallocated
    \((File\ recovery\ is\ trivial)\)
  – EXT3: Zeroes block pointers, marks as unallocated
    \((File\ recovery?\ Ummmm....)\)
Quick Example: EXT2 Recovery

1. Examine unallocated inodes with ils

   $ ils ext2-simple.img
   st_ino|st_alloc|...|st_size|st_block0|st_block1
   1713|f|...|0|10753|0
   1714|f|...|2300|8705|8706

   File size is non-zero

2. Use icat to recover original file content

   $ icat ext2-simple.img 1714
   This is a deleted file
   This is a deleted file
   This is a deleted file
   ...
So What About EXT3?

- Traditional techniques rely on "file carving":
  - Determine a "signature" for start of file
  - Start grabbing blocks until end of file "signature"
    or until size limit is reached
Problems w/ Carving EXT3

• Many Unix file types have no viable signatures
• Indirect blocks (metadata) in data runs
• File fragmentation, particularly on larger files
Leveraging Indirect Blocks
Pertinent Questions

• Why are we just ignoring indirect block data?
• Couldn't we use it to recover file content?
• Can we rebuild the entire original file?
## Looking At an Indirect Block

```
# blkcat -h ext3-example.img 252611

<table>
<thead>
<tr>
<th>Offset</th>
<th>Block Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>c4da0300 c5da0300 c6da0300 c7da0300 .... .... .... .... ....</td>
</tr>
<tr>
<td>16</td>
<td>c8da0300 c9da0300 cada0300 cbda0300 .... .... .... .... ....</td>
</tr>
<tr>
<td>32</td>
<td>ccda0300 cdda0300 ceda0300 cfda0300 .... .... .... .... ....</td>
</tr>
<tr>
<td>48</td>
<td>d0da0300 d1da0300 d2da0300 d3da0300 .... .... .... .... ....</td>
</tr>
<tr>
<td>64</td>
<td>d4da0300 d5da0300 d6da0300 d7da0300 .... .... .... .... ....</td>
</tr>
<tr>
<td>80</td>
<td>d8da0300 d9da0300 dada0300 dbda0300 .... .... .... .... ....</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>4048</td>
<td>b8de0300 b9de0300 bade0300 bbde0300 .... .... .... .... ....</td>
</tr>
<tr>
<td>4064</td>
<td>bcde0300 bdde0300 bede0300 bfde0300 .... .... .... .... ....</td>
</tr>
<tr>
<td>4080</td>
<td>c0de0300 c1de0300 c2de0300 c3de0300 .... .... .... .... ....</td>
</tr>
</tbody>
</table>
```

![Block Diagram](diagram.png)
Simple File Recovery Strategy

• Find beginning of file via signature
• Does the 13\textsuperscript{th} block look like an indirect block?
• If so, dump associated data blocks
• If last block address is not null, keep going

\textit{We can do this manually...}
There's an App for That...

# sigfind -b 4096 1F8B0800 ext3-example.img
Block size: 4096  Offset: 0  Signature: 1F8B0800
Block: 251904  (-)
Block: 252096  (+192)
Block: 252293  (+197)
Block: 252599  (+306)
...
# frib ext3-example.img 252599 >recovered.gz
# tar ztf recovered.gz
...
perl-5.10.1/patchlevel.h
perl-5.10.1/Configure
#
Don't Have a File Signature?

• Indirect blocks have a signature:
  – Any block N whose first 4-bytes == N+1
• Use relative location of indirect blocks to put file contents back together
• Beginning of file data will (hopefully) be the 12 blocks before the first indirect block
There's an App for That Too...

```bash
# fib ext3-example.img
...
252611   3436
...
# frib -I dbbks ext3-example.img 252611>indblks
# ls -lh *blks
-rw-r--r-- 1 hal hal 48K 2011-01-16 08:09 dbbks
-rw-r--r-- 1 hal hal 14M 2011-01-16 08:09 indblks
# cat dbbks indblks >recovered2.gz
# ls -l recovered*.gz
-rw-r--r-- 1 hal hal 14118912 2011-01... recovered2.gz
-rw-r--r-- 1 hal hal 14118912 2011-01... recovered.gz
# diff recovered*.gz
```
Fragmentation

• Not a problem if fragmentation occurs within data runs from indirect blocks

• Real problem if fragmented in first 13 blocks:
  – Start with signature, can't find first indirect block
  – Start from indirect block, can't find true file start
All is Not (Necessarily) Lost

• Use fib/frib to recover the majority of the file using indirect blocks strategy
• May be able to use file content signatures to piece together the first 12 blocks
• Reduce your search space:
  – Data blocks will tend to be in same block group
  – "First available" algorithm means start of file will usually be found in lower block numbers
Wrapping Up
Looking Ahead: EXT4

• These techniques will not work with EXT4
• EXT4 uses extents (start block + run len) rather than inefficient pointer strategy of EXT2/3
• Extents are zeroed when inode is deallocated—back to file carving again
• Good news:
  – "Delayed allocation" == less fragmentation
  – No more indirect block meta-data in data runs

EXT3 will be with us for a long time...
That's It!

• Any final questions?
• Thanks for listening!

Hal Pomeranz  hal@deer-run.com
Consultant, Mandiant hal.pomeranz@mandiant.com
Faculty Fellow, SANS Institute hal@sans.org

Tools and further detail at blog.mandiant.com