Forensic Timeline Analysis of ZFS Filesystems

Dylan Leigh

Supervisor: AProf. Hao Shi
Coordinator: Prof. Xun Yi

Centre for Applied Informatics, College of Engineering and Science
Victoria University, Melbourne, Australia

BSDCan 2014 - 16 May 2014
http://www.bsdcan.org/2014/schedule/events/464.en.html

Outline

1. Introduction to Timeline Forensics and ZFS
   - Digital Crime Scene Investigation
   - The Zettabyte File System
   - Existing ZFS Forensic Research
   - Our Research Project
2. ZFS Internal Structures
   - Uberblocks
   - Datasets and File Objects
   - Block Pointers
   - Spacemaps
3. Discussion & Demonstration
   - Observations & Issues
   - Demonstration
   - Future Work

Digital Crime Scene Investigation
AKA Computer Forensics

Detective: "I need to know what files were created, modified, deleted and accessed over time".

Timeline Analysis

- Timestamps from Filesystem
- Internal File Metadata
- Registry
- Logs
- "Super Timeline" from many sources

Timestamp Falsification

Criminal: "I should change the modification times on these files ..."

Detecting Forged Timestamps

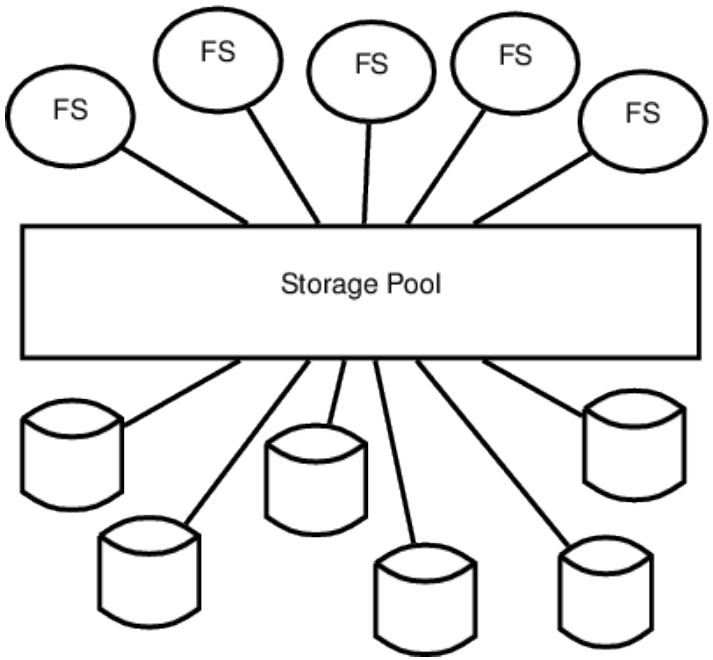
Cross-check all sources of timestamps

- Automated by "Super-timeline" tools
- Disadvantages
  - Time consuming
  - Other sources can be forged as well
  - Not possible to corroborate logs, registry etc with all files
- Filesystem internal structures available for all files
  - Existing forensic tools do not handle ZFS disks

Zettabyte File System

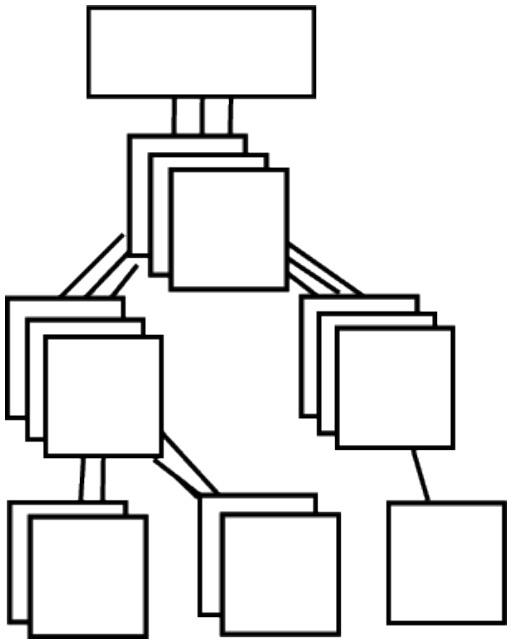
ZFS

- All disks added to a "pool" of storage
- Filesystems created as required from the pool
- Many advantages over traditional volume-based disk management
- Need for new forensic tools



ZFS Internals

- Everything is an object in a tree
- Redundant copies of objects across different devices
- Checksums in pointers allow self-healing
- ZFS manages inter-object dependencies



ZFS Transactions

- All writes to a ZFS pool are COW, atomic transactions
- Writes batched in a Transaction Group
  - Identified by Transaction Group ID (TXG)
  - Written to disk every 5 seconds (typically)
- New objects are written from the bottom up
  - Linked by "Block Pointer" structures, which include 1-3 references
- Finally a new "Uberblock" is written at the top of the tree
  - 4 redundant sets of 128 uberblocks, 2 at start & end of each device

Digital Forensic Implications of ZFS

Nicole Lang Beebe, Sonia D. Stacy, and Dane Stuckey

Digital Investigation, Elsevier, 2009

- Overview of ZFS forensics
- Advantages (COW copies; temporal state awareness...)
- Disadvantages (compression; dynamic sizes...)
- Based on examination of the documentation and source code
- No empirical analysis of actual file systems
- No specific techniques or guidelines for technicians

ZFS Forensics Research

Data Recovery - Max Bruning

- "ZFS On-Disk Data Walk (Or: Where's My Data)." *OpenSolaris Developer Conference*, 2008
- "Recovering removed file on zfs disk." *(website)*, 2008.

"Zettabyte File System Autopsy"

Andrew Li, *Macquarie University*, 2009

- ZDB Enhancement for locating known data within a disk

"Analysis and Implementation of Anti-Forensics Techniques on ZFS"

Cifuentes, J. and Cano, J. *Revista IEEE America Latina*, 2012

Aim

- Determine what timeline information can be obtained from ZFS internal structures
- Conduct empirical studies based on real filesystems written to disk
- Determine techniques for verifying file timestamps and detecting forged timestamps
- Provide a research basis for developing new forensic timeline tools for ZFS

Experiments

- Create ZFS pools with 1-9 disks
  - flat, mirror, mirror pair and RAIDZ configurations where applicable
- Simulated file activity based on typical corporate network storage statistics
  - Data from "A five-year study of file-system metadata.", Agrawal, Nitin, et al., *ACM Transactions on Storage*, 2007
  - Also examined own systems as case studies
- For each pool configuration:
  - Control (No tampering)
  - System clock reverted while a file saved
  - File timestamp modified with touch
- Data saved using ZDB (ZFS Debugger) every 30 minutes

Uberblocks

- Written every 5 seconds under steady load
  - More frequent under heavy load or for synchronized writes
- Extracted per-device, each device contains 4 identical sets of 128 uberblocks
  - `zdb -P -uuu -l /dev/<device>`
  - Sets may differ between top-level virtual devices
- Each Uberblock includes a Block Pointer to the Meta Object Set, Transaction Group ID and the time it was written

Uberblocks: Example

```
Uberblock[73]
    magic = 0000000000bab10c
    version = 28
    txg = 1737
    guid_sum = 44206048787235682O1
    timestamp = 1382428544 UTC = Tue Oct 22 18:55:44
2013
    rootbp = DVA[0]=<0:3e0c000:200>
DVA[1]=<1:3f57200:200> DVA[2]=<2:3593a00:200> [L0 DMU
objset] fletcher4 lzjb LE contiguous unique triple
size=800L/200P birth=1737L/1737P fill=42
cksum=15ffed59a7:7e9c9c5...
Uberblock[74]
    ...
```

Uberblocks: Forensic Uses

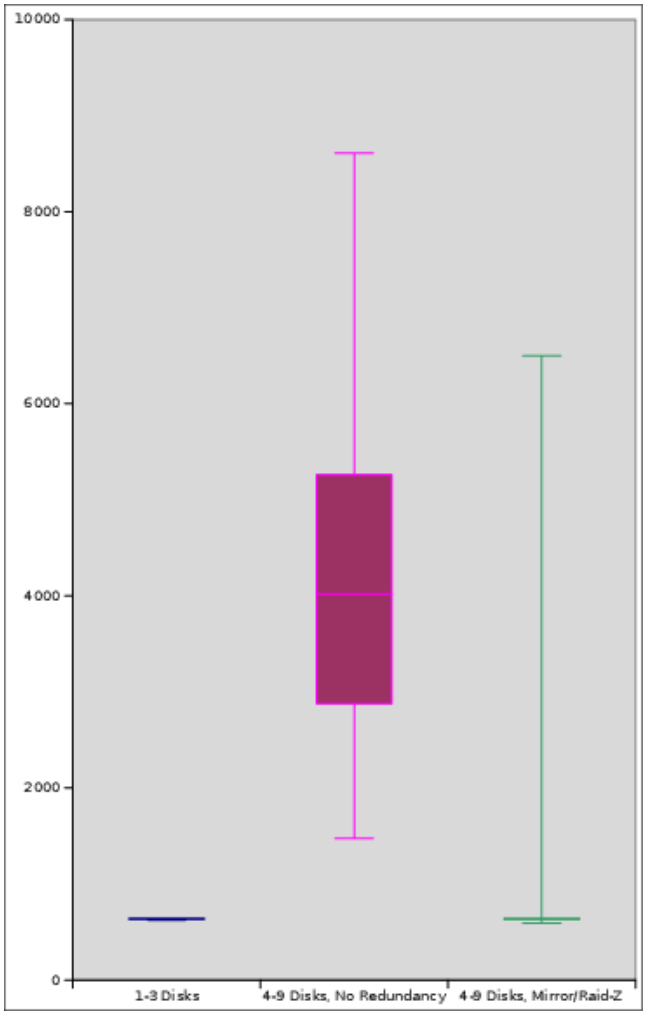
- Link a TXG to the time it was written
  - Detect forged timestamps if the timestamp on the file and uberblock do not match (within 5 sec) for the same TXG
- Consecutive Uberblocks have increasing TXG and timestamp
  - System clock alterations will be visible if they last longer than 5 seconds

Disadvantages

- Uberblocks are relatively easy for attacker to tamper with
  - At the top of the ZFS hash tree
- Typically last only 10.5 minutes

Uberblocks: Gone in 640 Seconds

- Uberblocks only last 635-640 seconds in most cases ($5sec \times 128$)
- Pools with 4 or more top-level vdevs retain uberblocks for longer
  - A TXG may not affect all vdevs
  - At most only 2.4 hours in experiments
- Thus uberblocks are most useful in a "Dawn Raid" scenario
- Detection in 50% of experiments with forged timestamps (despite collecting data every 30 minutes)



ZFS Datasets

- Each filesystem stored in a Dataset (object set)
  - Datasets may also be Snapshots, Clones, ZVOLs
  - Each ZFS Pool may have up to 2^{64} datasets
- Metadata extracted per dataset
  - `zdb -P -dddddd -bbbbbb <poolname>/<dataset-name>`
  - List of all objects and their block pointers in the dataset
  - Includes dataset-specific objects e.g. Delete Queue
  - Directory objects include a list of filenames and the ID of the corresponding File object
  - File objects include attributes and BPs pointing to the file data

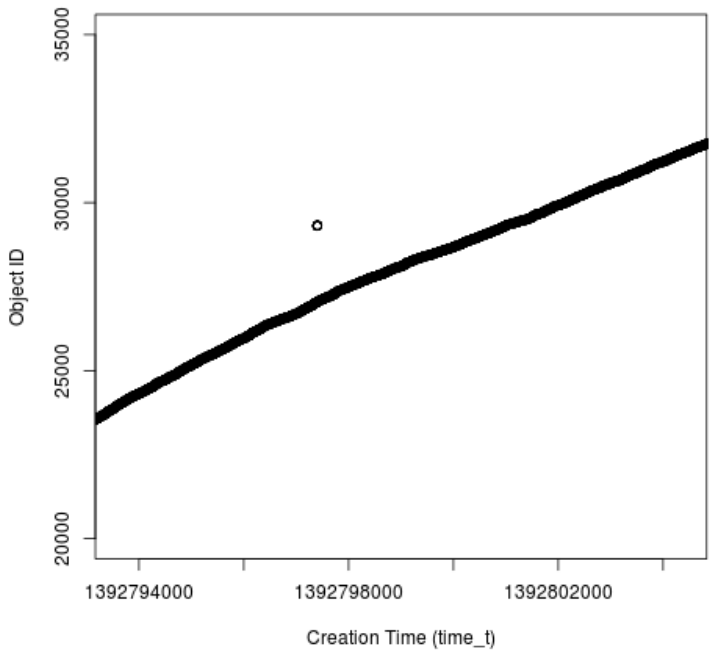
ZFS Plain File (simplified)

```
        Object  lvl   iblk  lsize   %full  type
         15417    1  16384  67072  100.00  ZFS plain file
            path    /tampered-file
            uid     0
            gid     0
            atime   Tue Oct 22 17:55:39 2013
            mtime   Tue Oct 22 17:55:40 2013
            ctime   Tue Oct 22 18:58:00 2013
            crtime  Tue Oct 22 18:55:39 2013
            gen     1737
            mode    100644
            size    66566
            parent  4
    Indirect blocks:

 0 L0 DVA[0]=<2:353c200:10600> [L0 ZFS plain file] single
size=10600L/10600P birth=1737L/1737P ...
```

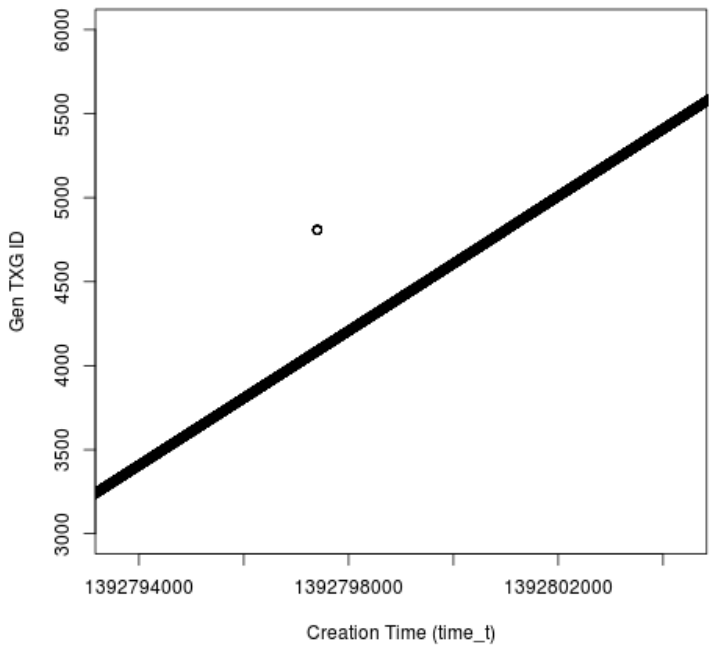
Object Number

- All dataset objects have increasing ID numbers
- If the creation time of a file is falsified, the object numbers will show the true order
  - Successful in all experiments involving file creation time
- Cannot be used to detect a falsified modification time



Generation TXG

- File and directory objects store the Transaction Group ID when they were created ("Gen")
- Can be used like Object Number to detect an out-of-order creation time
  - Successful in all experiments involving file creation time
- Cannot be used to detect a falsified modification time



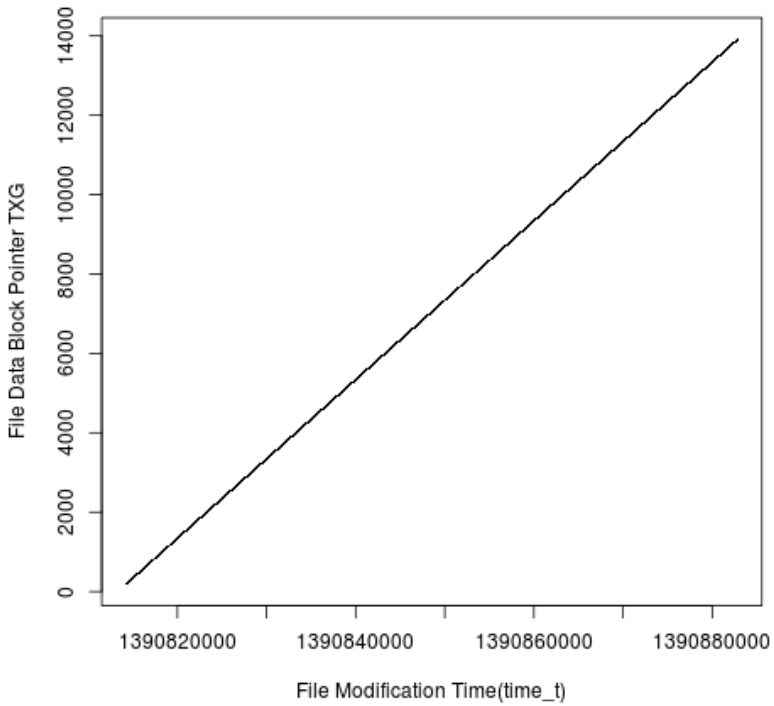
Block Pointers

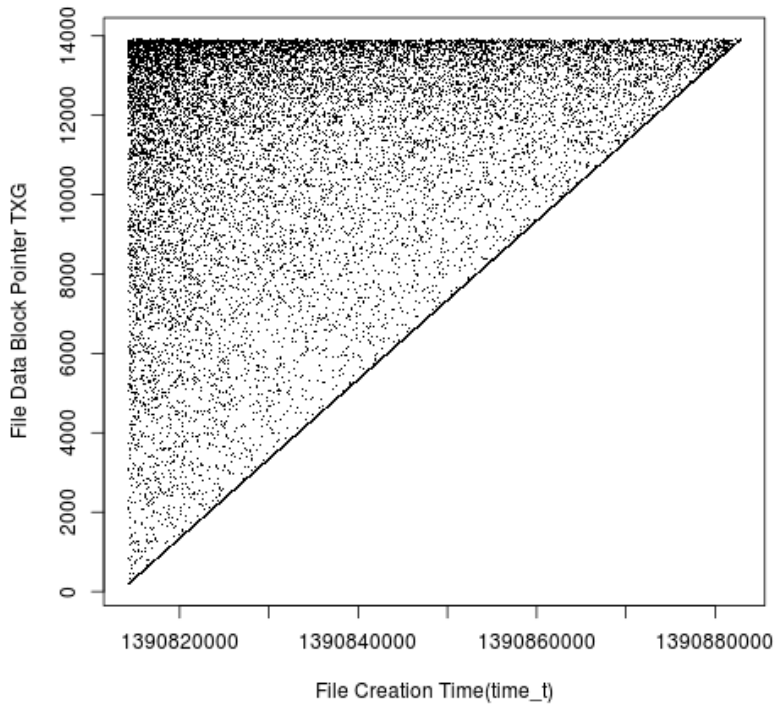
- Block pointers contain the TXG in which they were written
- This can be cross-referenced to a TXG from other files, directories, or uberblocks
  - Need 5 seconds of tolerance for each Transaction Group
  - Can be used to detect falsified modification and creation times
  - Successful at detecting tampering in all experiments
- File data BPs can provide past modification information for larger files
  - BPs to blocks written in prior transactions (and not changed) will contain prior TXG
  - Effective for large files which are updated in small, isolated parts (e.g. VM images)

Large File Example

```
    Object  lvl   iblk  lsize   %full  type
     57296    2  16384  262144  100.00  ZFS plain file
        ...
        size    158599
        ...
Indirect blocks:
    0 L1  DVA[0]=<1:202c7400:400> DVA[1]=<2:2183e000:400>
        [L1 ZFS plain file] double size=4000L/400P
        birth=15853L/15853P fill=2
    0  L0 DVA[0]=<1:24409600:20000> [L0 ZFS plain file]
        single size=20000L/20000P birth=15464L/15464P fill=1
20000  L0 DVA[0]=<1:24a7da00:20000> [L0 ZFS plain file]
        single size=20000L/20000P birth=15853L/15853P fill=1
```

Block Pointer Plots





(5 disk RAIDZ3, No tampering, 24 Hours)

Space Allocation

Virtual Device

- Modified Round Robin algorithm to choose device to write
  - Switches devices every 512k

Metaslab

- Vdevs divided into equal regions called “metaslabs”
- ZFS tries to fill metaslabs before using new ones

Spacemap

- Free space metadata for each metaslab stored in a “spacemap” object

Virtual Devices

- The round robin algorithm means that the order of object writes is reflected in the vdev it is written to
- Theoretically this could be used to verify timestamps
- In practice, transient files prevent this from being useful to verify timestamps or detect tampering
    - Could possibly be useful in write-only workloads to show that there was unknown activity between known writes

Metaslabs

- Extracted per pool:
  - `zdb -P -mmmmmm <pool>`
- One set of metaslabs for each top-level virtual device
  - Each metaslab has a spacemap with many alloc/free segments:

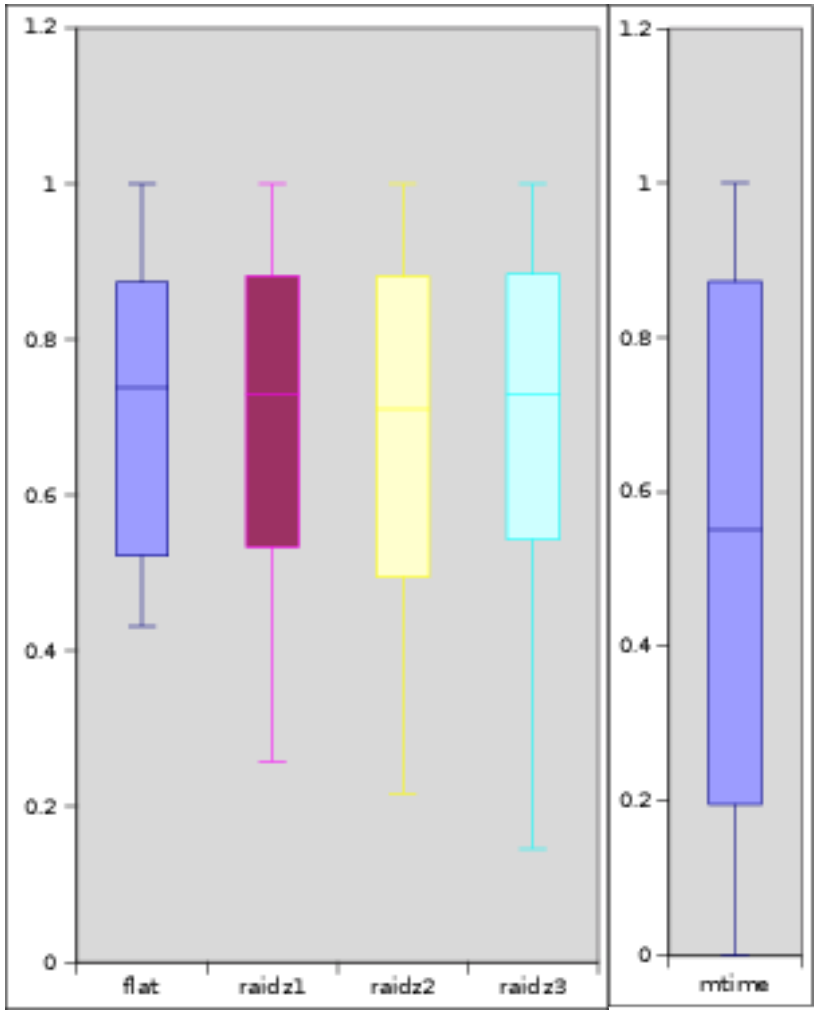
```
metaslab  0   offset    0   spacemap     33  free 107904000
            segments 3259   maxsize  256512  freepct    80%
   [   0] ALLOC: txg 3857, pass 1
   [   1]     A  range: 0000000000-0000000400  size: 000400
   [   2]     A  range: 0000002000-0000002c00  size: 000c00
...
   [4792] FREE: txg 3857, pass 2
   [4793]     F  range: 00000cae00-00000cba00  size: 000c00
   [4794]     F  range: 000025ac00-000025b800  size: 000c00
...
```

Spacemaps

- Spacemaps are log-structured lists of free blocks
  - "Condensed" when alloc/free entries which cancel out are detected
  - Condensation leads to many recent segments and a long tail of older ones
- Each segment stores the TXG when it was written/condensed
  - Probably later than the TXG when the space was allocated!
- Transient files and condensation prevent them from being forensically useful, even for recent files
  - Cannot be used to detect tampering
  - Can sometimes be used to verify the minimum age of files

Spacemaps: Longevity

- Median age of a segment was 72.3% of the current TXG
- Pools with Mirror and Raid-Z vdevs have earlier outliers
  - Effect increases with increasing redundancy
- Highly dependent on workload and other factors



Summary of Results

Detection	Structures				
	Uberblock	BP TXG	Gen TXG	Obj. No.	Spacemap
Forged Mtime	Sometimes	Yes	No	No	No
Forged CRTime	Sometimes	Yes	Yes	Yes	No
Past Mtime	No	Sometimes	No	No	No

Key Points

- TXGs from File Object Block Pointers are most useful structure for timeline forensics
  - Can sometimes find previous modification times
- Forged creation time easier to detect than forged modification time
- Uberblocks only effective if collected soon after tampering

False Positives

- Timestamp mismatches can be caused by normal events as well as anti-forensics
- Clock corrections will affect all timestamps
  - Correcting a fast clock will cause out-of-order false positives
    - Occurred in 2/76 experiments
- Innocent userspace changes of timestamps
  - Unpacking archives with timestamps preserved
  - Copying/moving files with timestamps preserved

Falsification of Internal Data

- A determined adversary could tamper with internal metadata as well as the file timestamps
  - Uberblocks are relatively easy to modify
- Tampering of objects/BPs also requires recalculating checksums for all parent BPs
  - Object Number and Generation TXG could be easily changed providing there is a viable false ID/TXG to replace it with
  - Block pointer TXG could also be changed with more care to keep order of TXG consistent
  - Spacemaps may be more difficult to alter, although the alteration would probably be indistinguishable from condensation

Demonstration

```
# zpool create ...
```

Future Work: Next Steps

Survey

- Real data would be better than simulated file activity
- The next phase of this project is to conduct a survey of real ZFS pools from production systems
  - Volunteers submit anonymized ZDB data (paths and names removed)

Automation

- Python scripts used for experiments
- Develop practical timeline utility
- Work in progress

Future Work: More Structures to Examine

- Per-dataset objects (Master Node, Delete Queue, ...)
- ZFS Intent Log
- Snapshots
- Meta Object Set
- Past object trees from previous uberblocks
- Old blocks on disk no longer referenced...

Future Work: More Pools and Workloads

- Longer running times, larger disks, more disks...
- ZFS Features: Compression, Deduplication ...
  - Dedicated Log and Cache devices
- Other workloads
  - Desktop, Home NAS, Webserver, VM host, Databases...
  - Pools with many datasets
- Pools providing ZVOLs for other FS

Summary

- Multiple ZFS Structures can be used to corroborate timestamps and detect falsified timestamps
  - Block Pointers in File Objects are particularly useful and can sometimes be used to determine the time of previous modifications
- Clock corrections and other normal behaviour could appear to be deliberate tampering with timestamps
- More work needs to be done to examine other ZFS structures, configuration options and systems with varied workloads

Thanks and Acknowledgements

Thankyou

BSDCan for funding my travel and providing me with this opportunity to speak to the BSD community.

My supervisor, coordinator and other VU staff for their invaluable guidance and assistance.

My wife for her constant patience, support and understanding.

Contact Details and Questions

- Dylan Leigh
  - Email: research@dylanleigh.net
  - Web: research.dylanleigh.net
- Formal Paper and Quick Reference
  - http://www.bsdcan.org/2014/schedule/events/464.en.html
- Questions?