MattockFS

Computer-Forensics File-System

A family tree

- 2002: OCFA Anycast
- 2006: CarvFS
- 2006: Sealed Digital Evidence Bag
- 2008: MinorFS

The OCFA Anycast

- Message bus solution
- Based on the **workers** concept
- Part of Open Computer forensics Architecture
- Persistent Priority Queues
- Virtually infinite size queues

Issues with AnyCast

- Infinite size queues
- Multiple tools in tool-chain access same data
- Result: Many **page-cache misses**

Spurious reads in OCFA

- Page-cache misses
- Early <u>hashing</u> and Content Addressed Storage
- No <u>locality of data</u> design
- Server-side data entry

CarvFS

- Storage requirements traditional file carving
- CarvFS allows for **zero-storage** carving
- Carved files not copied our but designated
- CarvPath designations
  - /mnt/carvfs/mp3/18400+4096_S4096_47912+975.crv

Issues with CarvFS

- Read-only access to forensic disk image
- In large cases hundreds of mounted image files
- OCFA hacks
  - Bypass CarvFS to write to underlying growing archive
  - Inefficient hybrid CarvFS/CAS storage

Shared mutable data

- Meta-data
- Evidence data
- Derived evidence data
- Provenance logs

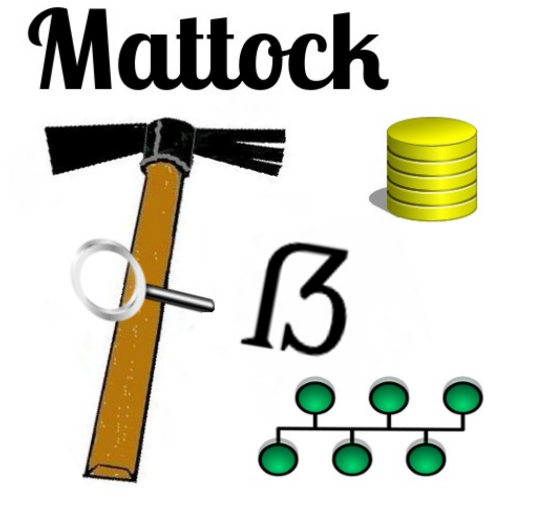
Shared mutable data

- Meta-data
- Evidence data
- Derived evidence data
- Provenance logs

- Anti forensics ?
- Forensic process integrity ?

Integrity measures

- Trusted immutable data and meta-data
  - Lab side sealed digital evidence bags.
- Trusted provenance logging
- Capability based access control
  - Sparse capabilities
  - Mandatory Access Controls



Spurious reads measures

- **Opportunistic hashing**
- Linux **fadvise** API
- **Integration** of message-bus and data-archive
- Priority queues → Sortable **sets**
- Provide **pressure** data for throttling purposes

MattockFS

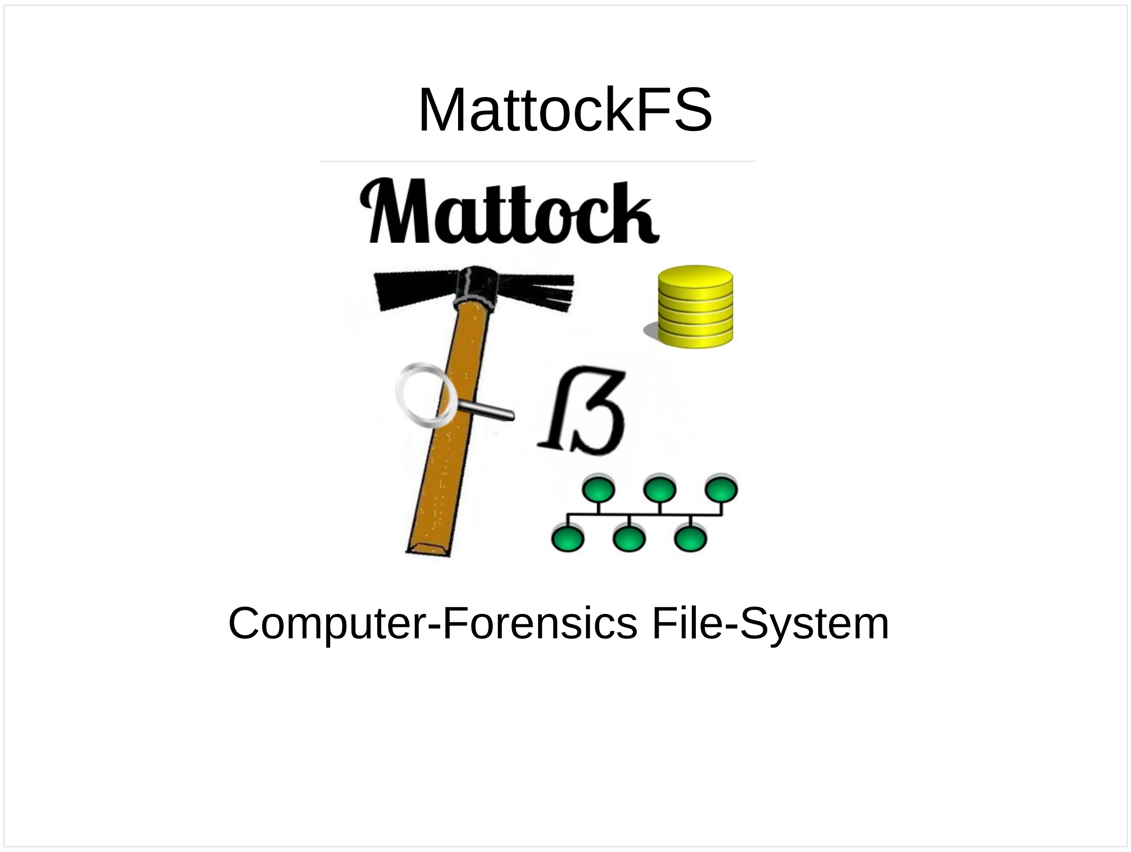
- User space file-system for forensic frameworks
- Zero-storage carving
- Hybrid data-archive/message-bus solution
- Helps minimize spurious read induced performance degradation.
- Provides write-once (meta) data
- Provides safe capability based API
- Provides trusted provenance logs

MattockFS: Hooks

- Hooks for forensic frameworks
  - Framework based routing functionality
  - Pressure data for throttling new-data input
  - Queue size for throttling
- Hooks for multi-server setup
  - Geared to locality of data
  - Migrating CPU intensive jobs

MattockFS

Combining a zero-storage carving-capable high-integrity data-storage archive with trusted forensic logging, a capability secure API and a local page-cache friendly message bus solution.

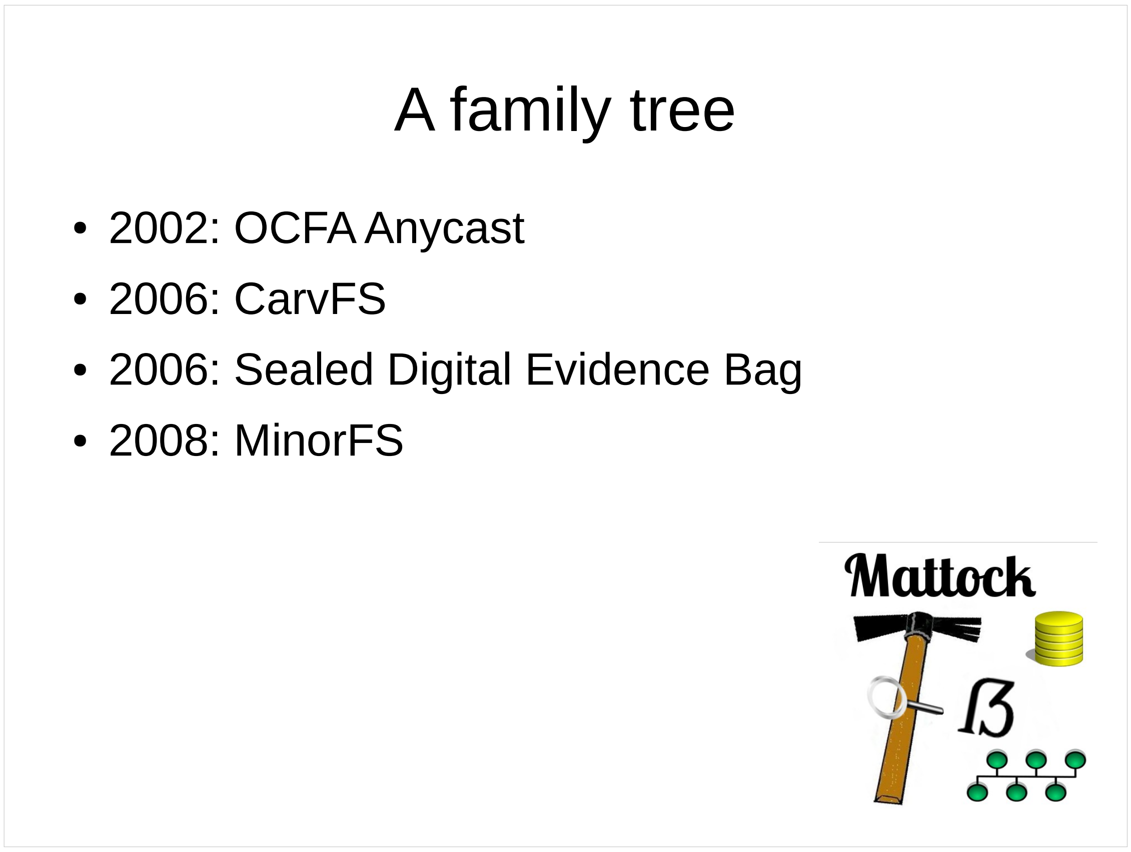


MattockFS

Mattock

Computer-Forensics File-System

This is a short talk about the MattockFS computer-forensic file-system. MattockFS is a user space file-system that runs on the Linux platform and that is geared towards the specific needs for computer-forensic frameworks with respect to data storage and access and with respect to message-bus facilities.



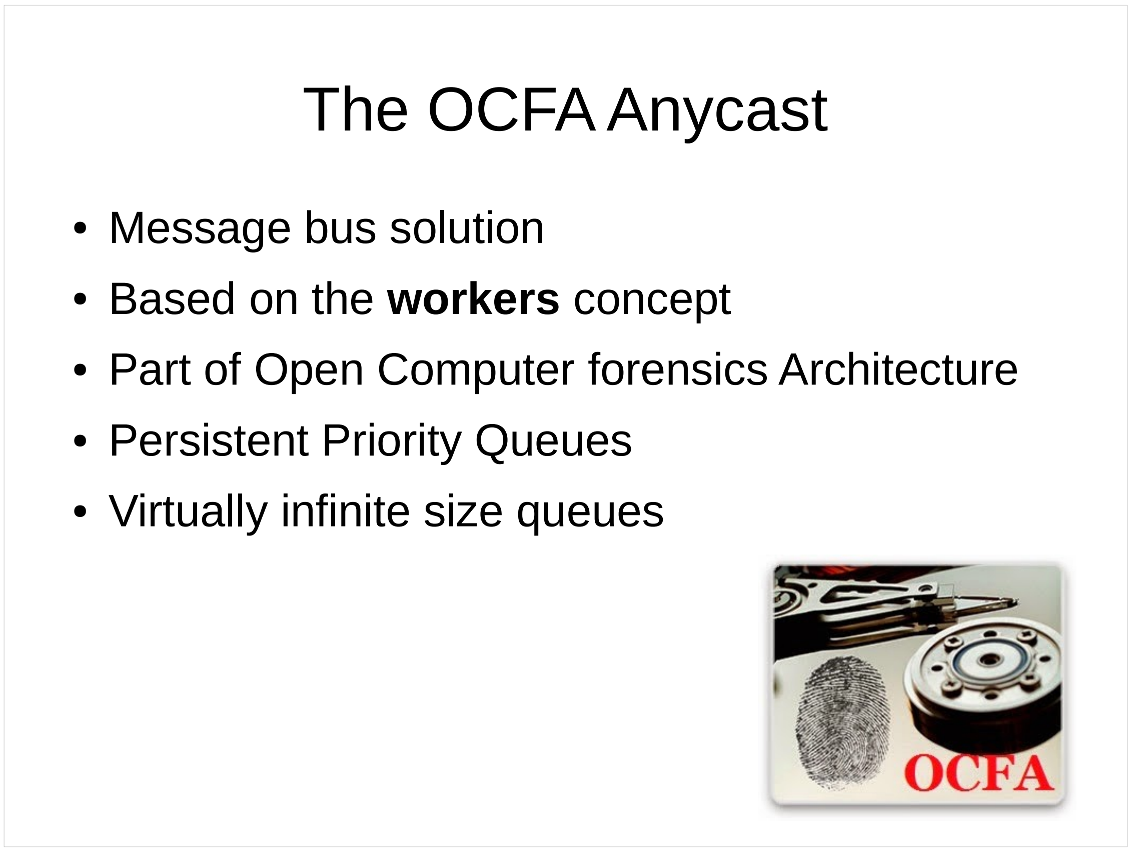
A family tree

- 2002: OCFA Anycast
- 2006: CarvFS
- 2006: Sealed Digital Evidence Bag
- 2008: MinorFS

Before we get into MattockFS itself, you could say MattockFS has four important ancestors with respect to parts of its features.

The Open Computer Forensics Architecture had a core component named the AnyCast. This was basically what today would be called a message broker or a message bus.

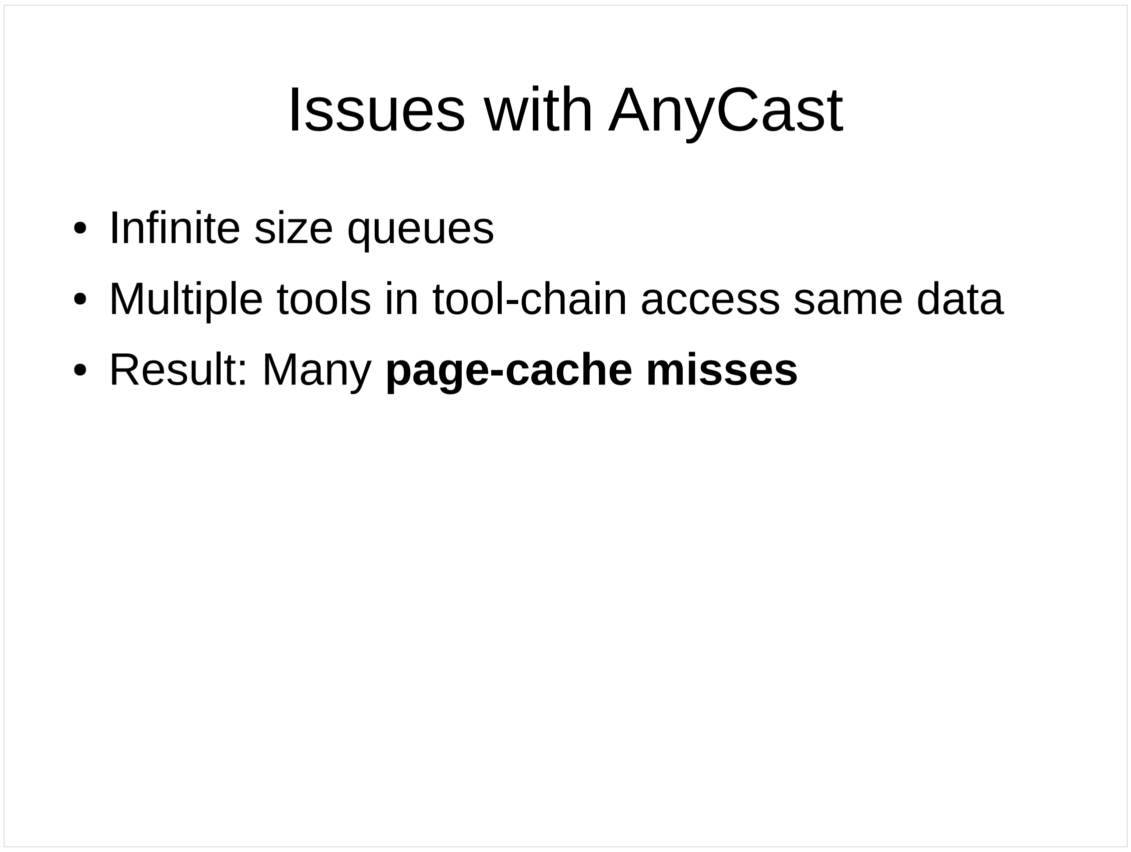
CarvFS introduced the concept of CarvPath designations as a way to use pseudo file-paths to implement a feature known as in-line carving or zero-storage carving where you could extract files from a disk image without actually copying them out. Than the Sealed Digital Evidence Bag concept and MinorFS introduced system integrity properties and facilities that I will show are important with respect to the threath of anti-forensics.



The OCFA Anycast

- Message bus solution
- Based on the **workers** concept
- Part of Open Computer forensics Architecture
- Persistent Priority Queues
- Virtually infinite size queues

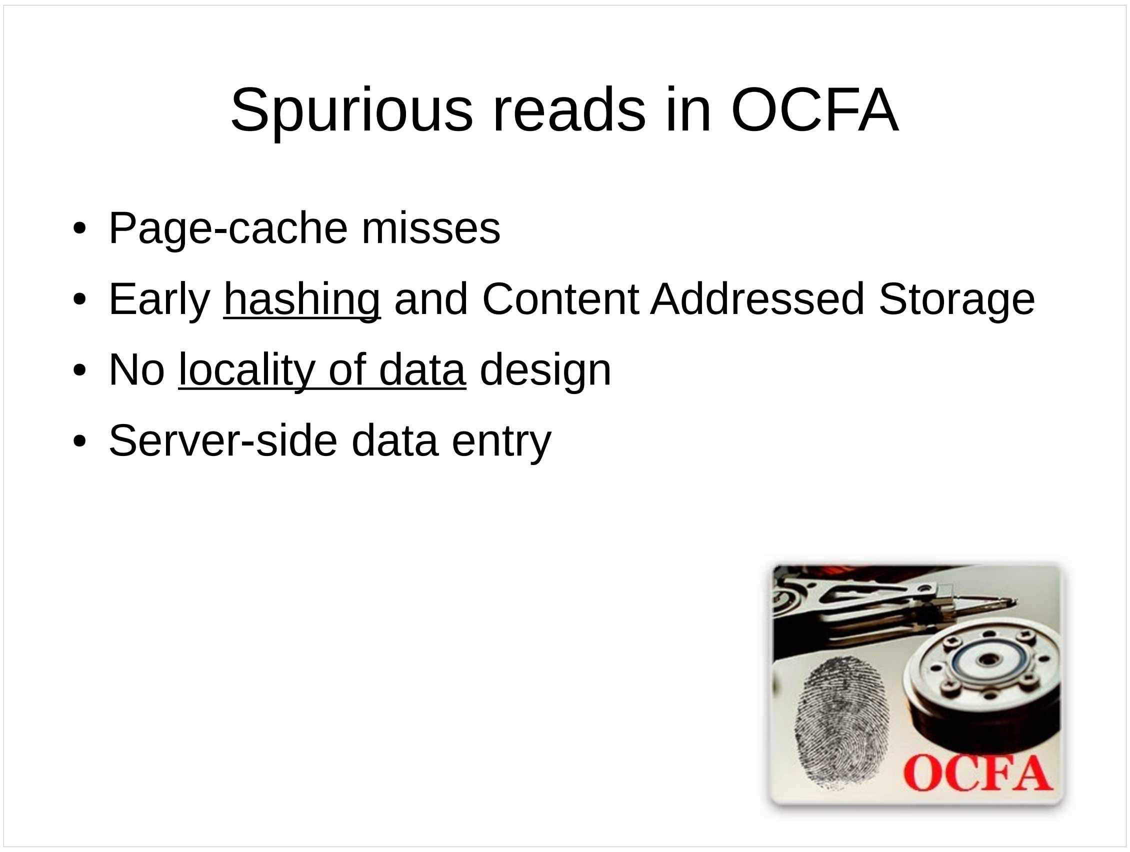
Lets start by looking at one of these MattockFS ancestors. The OCFA Anycast. The Anycast was a message bus solution, kind of like RabitMQ, but geared soly on the worker concept. It was part of a computer forensic framework named the Open Computer Forensics Architecture or OCFA. It used the concept of persistent priority queues that for all intent and purpose were infinite in size, a property that we later found out had some drawbacks.



Issues with AnyCast

- Infinite size queues
- Multiple tools in tool-chain access same data
- Result: Many **page-cache misses**

So what issues were there with the Anycast? The problem lies in the very nature of the computer forensic process. In this process, multiple tools in a tool chain are used consecutively on a single piece of data. The infinite size queues however combined with the lack of throttling in the OCFA framework, resulted in the fact that much data would long have been moved out of the operating systems page-cache at the moment the next tool would access the data. This would result in many page-cache misses and thus many spurious reads from the actual hard disks.



Spurious reads in OCFA

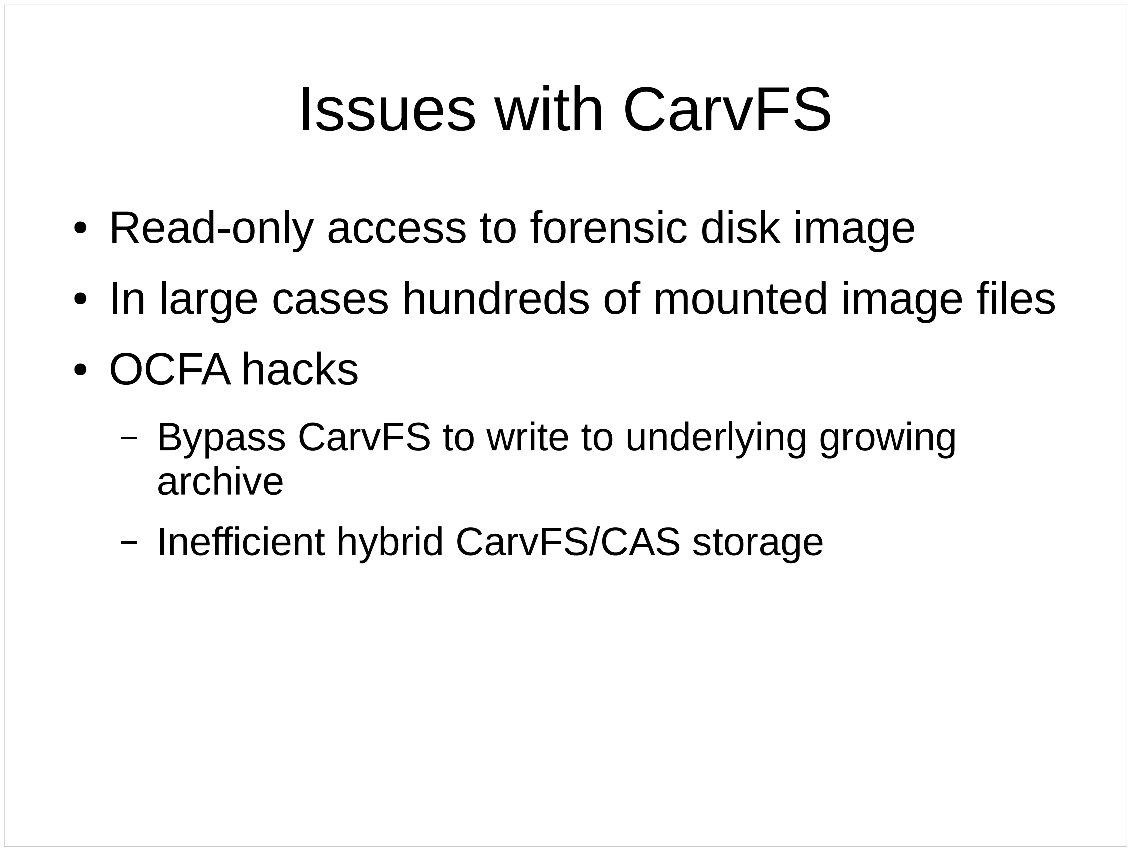
- Page-cache misses
- Early hashing and Content Addressed Storage
- No locality of data design
- Server-side data entry

Given the IO intensive nature of the computer forensic process, spurious reads can be quite a performance issue for the overall system. Page cache misses as induced by the Anycast design are one source of such spurious reads, but in OCFA there were more design issues also leading to even more spurious reads. Either by aggravating the page-cache miss problems, or for other reasons. OCFA used early hashing of all data, partialy resulting from its primary storage model that was based on Content Addressed Storage or CAS. This added extra time between the first read and the last read, thus resulting in increased page cache issues. Two other spurious read issues resulted from the lack of a “locality of data” based design and the server-side data entry.

CarvFS

- Storage requirements traditional file carving
- CarvFS allows for **zero-storage** carving
- Carved files not copied our but designated
- CarvPath designations
  - /mnt/carvfs/mp3/18400+4096_S4096_47912+975.crv

A second ancestor of MattockFS is CarvFS. CarvFS is a user space file-system that allows mounting a single computer forensic disk image, and allows to then carve files from unallocated space without actually having to copy the data out. CatvFS introduced the concept of CarvPath designations, like the one you see at the bottom of this slide. These designations are made up primarily of a combination of offset and size that together describe where in the mounted entity the actual data resides.



Issues with CarvFS

- Read-only access to forensic disk image
- In large cases hundreds of mounted image files
- OCFA hacks
  - Bypass CarvFS to write to underlying growing archive
  - Inefficient hybrid CarvFS/CAS storage

While CarvFS was a big step forward for carving out files, in the light of usage within a computer forensic framework there were a number of issues. CarvFS was designed to mount a single disk image, In a large case, hundreds of disk images will be entered. In the case of OCFA this meant that hundreds of disk image files would have to be mounted as user-space file-system simultaneously, resulting in massive use of system resources that turned out to be prohibitive. An other issue was that for adding data in any other way, CAS storage was still needed. A hack was used to use CarvFS with OCFA that basically had modules bypass CarvFS altogether in order to create one single big archive file. The integration of CarvFS into OCFA never became fully optimal because CAS storage remained such an integral part of its core design.

Shared mutable data

- Meta-data
- Evidence data
- Derived evidence data
- Provenance logs

Modern insights with respect to system integrity force us to look at the use of shared mutable state within computer forensic frameworks. Basically a forensic framework weaves together a number of tools into suitable tool-chains for different types of data. Each module in such a framework accessing shared mutable data has the theoretic ability to change the shared mutable data. As non of the modules is expected to be malicious, the historic approach to shared mutable data also used in OCFA was that this would not be a serious issue.

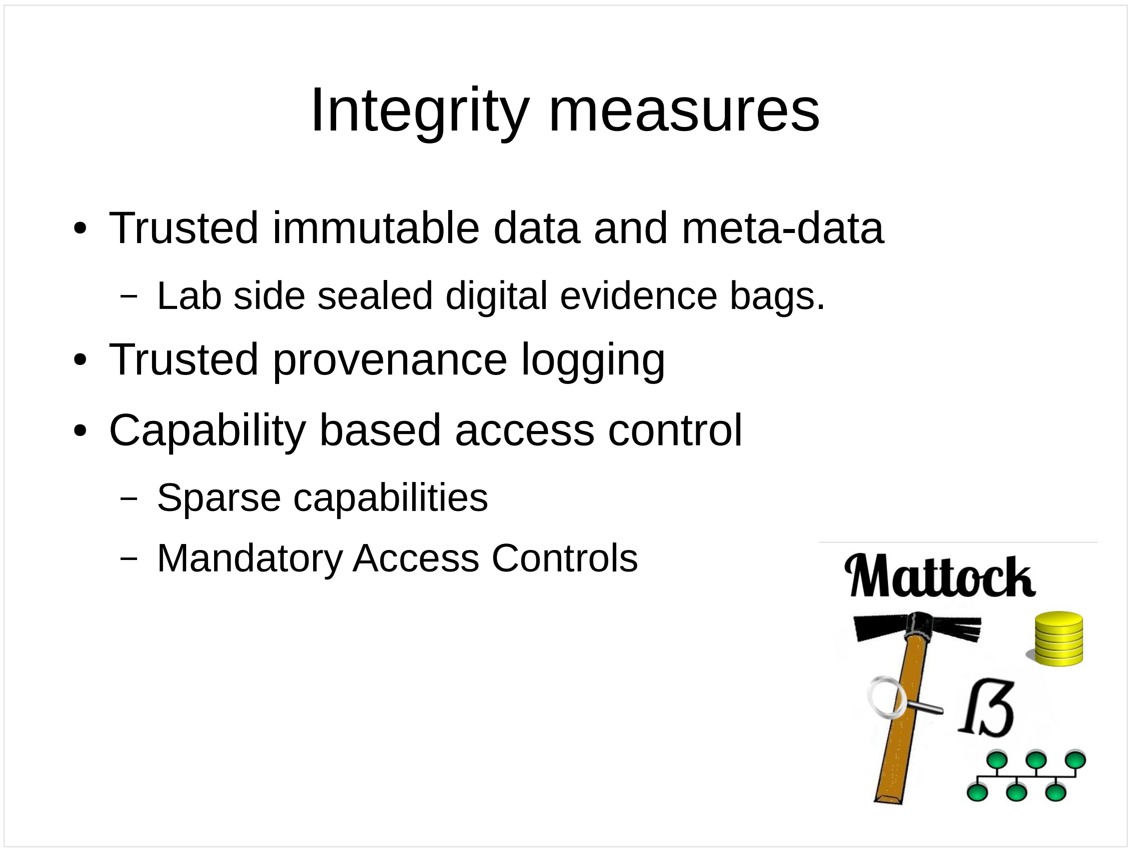
Shared mutable data

- Meta-data
- Evidence data
- Derived evidence data
- Provenance logs

- Anti forensics ?
- Forensic process integrity ?

Enter anti-forensics. Tools used in the tool-chain, just like programs such as web browsers and PDF readers do have vulnerabilities. Remember that the suspect controls what data resided on his disk. There could be exploits targeting specific tools or libraries in the tool-chain, and these exploits could be used to target shared mutable data in our framework. Either corrupting the data or possibly even rendering the whole system useless.

With respect to system integrity, experience with OCFA has shown some modules can just crash benignly if exposed to certain wrongly identified data files. Given that a random crash can have undefined behavior, there remains a serious risk that even non anti-forensic crashes might corrupt the system through shared mutable state.

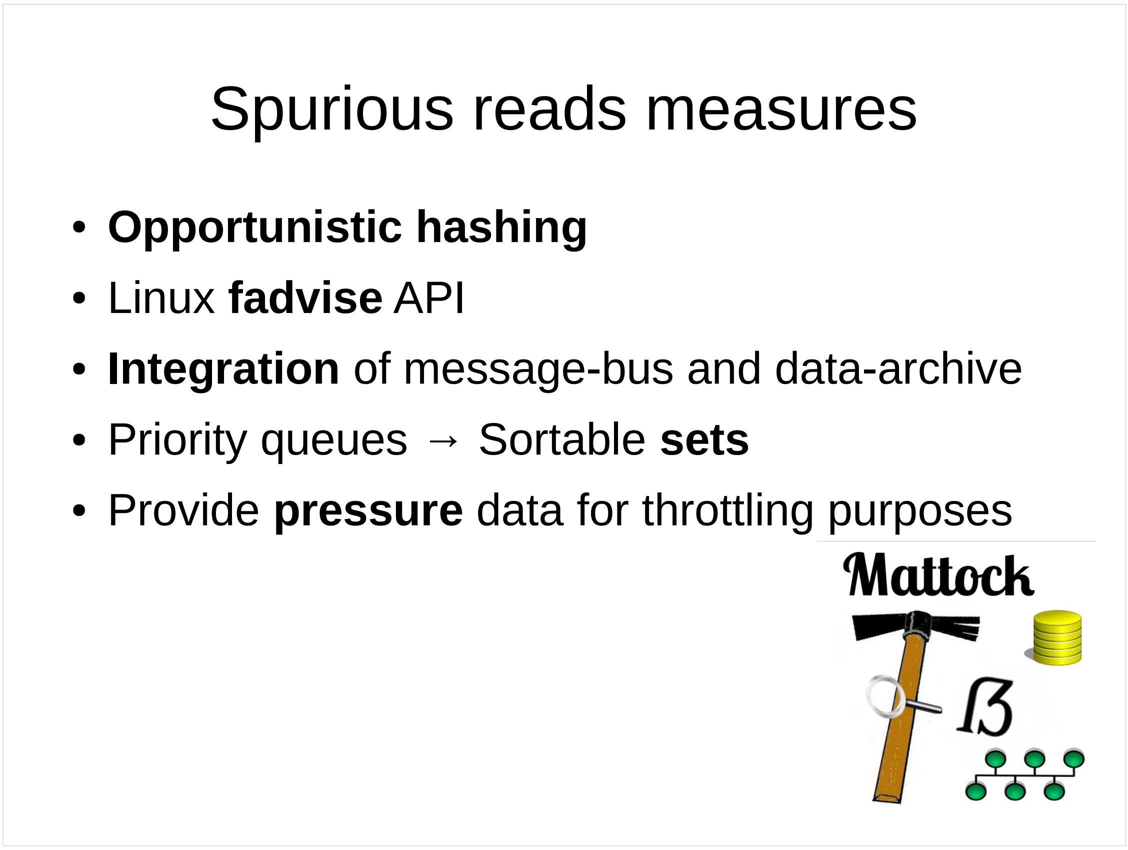


Integrity measures

- Trusted immutable data and meta-data
  - Lab side sealed digital evidence bags.
- Trusted provenance logging
- Capability based access control
  - Sparse capabilities
  - Mandatory Access Controls

Mattock

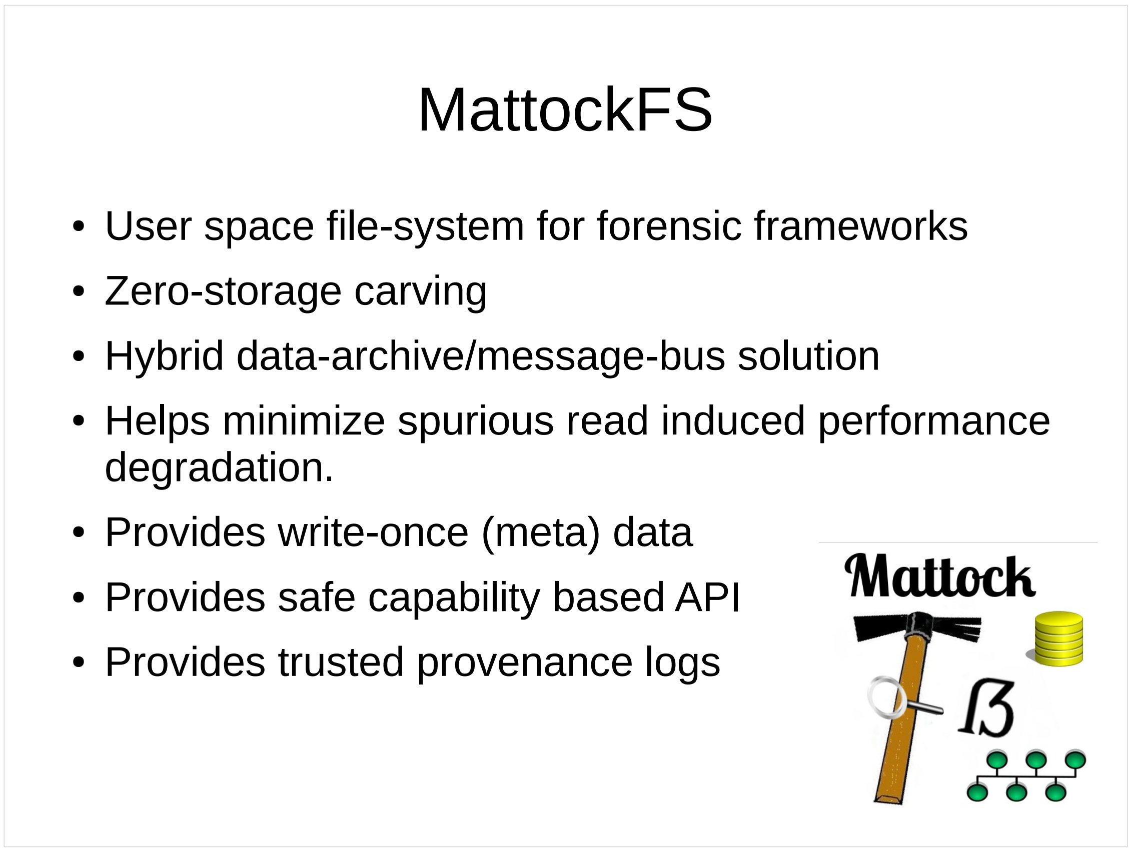
Ok, lets start looking at MattockFS now. First let look at the system integrity measures that MattockFS introduces. The concept of Sealed Digital Evidence Bags introduced us to the idea that data and meta-data should be sealed. MattockFS uses the fact that it runs under a different user id than the rest of the system to provide trusted write-once immutable data. It also uses this fact to provide a trusted provenance log facility. Data, meta-data and provenance log are protected from tempering by an exploit against one of the tools in the tool-chain. Access to the MattockFS file-system with respect to anything that might intervene with this goes through what is called capability based access control. We implement these in a way we borrowed from the MinorFS least authority file-system.



Spurious reads measures

- **Opportunistic hashing**
- Linux **fadvise** API
- **Integration** of message-bus and data-archive
- Priority queues → Sortable **sets**
- Provide **pressure** data for throttling purposes

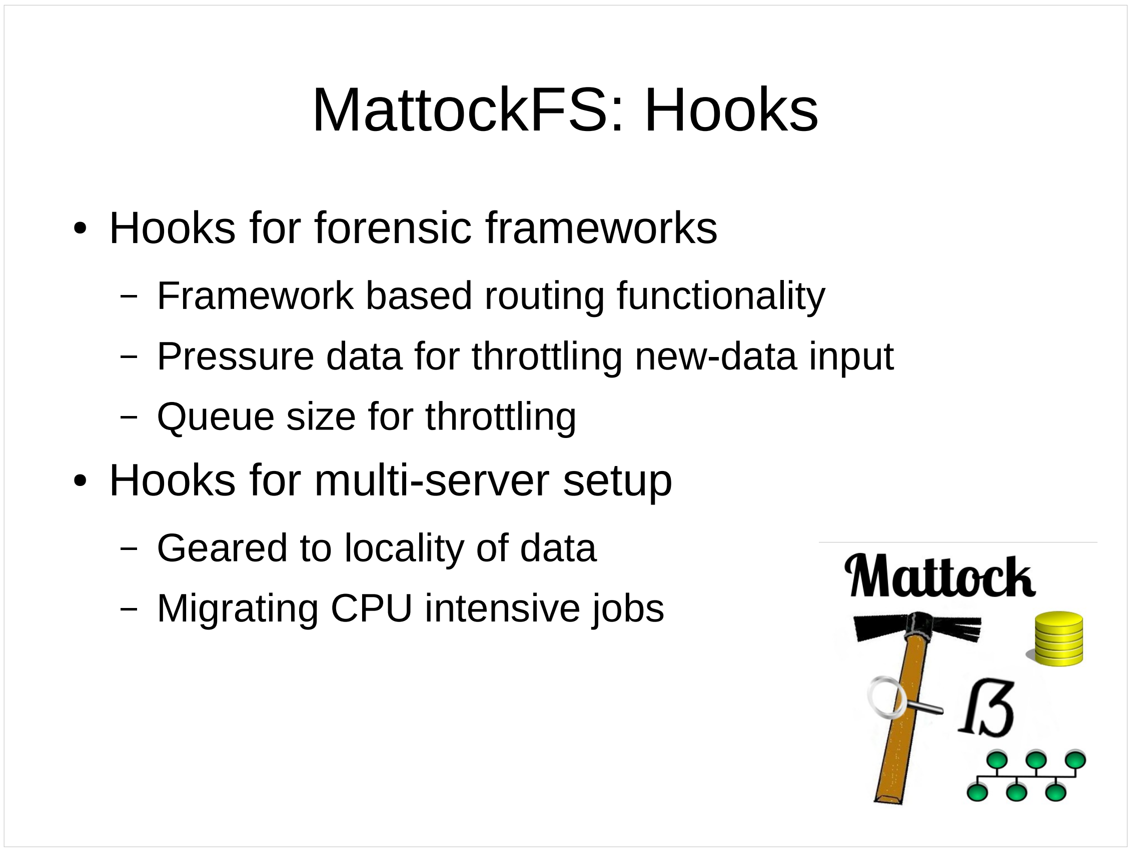
Now that we have tackled the anti-forensics and integrity concerns, lets look at what MattockFS does for attenuating our spurious reads problem. Firs of all MattockFS has built-in hashing functionality. Whenever data is read from or written to in MattockFS, MattockFS tries if it can use this data to further calculating the hash of any CarvPath entity that this data is part of. Secondly, mattockFS communicates with the Linux kernel regarding archive data being accessed. It informs the kernel what data it will soon want to access again and what data it doesn't. This should allow the kernel to be smarter about page-cache handling. In order to allow the message bus and storage system to effectively work together, MattockFS integrates the message-bus into the file-system. If provides multiple policies for picking messages that are most beneficial to opportunistic hashing or page-cache pressure. Finaly MattockFS provides data to the module framework that should allow the framnework to properly implement throttling.



MattockFS

- User space file-system for forensic frameworks
- Zero-storage carving
- Hybrid data-archive/message-bus solution
- Helps minimize spurious read induced performance degradation.
- Provides write-once (meta) data
- Provides safe capability based API
- Provides trusted provenance logs

A quick sumary of the main features of mattockFS. MattockFS is a user-space file-system for Linux that has facilities for zero storage carving. It is a hybrid data-archive and message-bus sytem geares pecifically at the needs of the computer forensic process. One of its primary goals is the minimization of spurious hard disk reads that hinder overal system performance. Next to performance, MattockFS provides multiple anti-anti-forensics facilities. Write-once access to data and meta-data, trusted provenance logs and a secure capability based form of access control for modules communicating with the file-sytem.

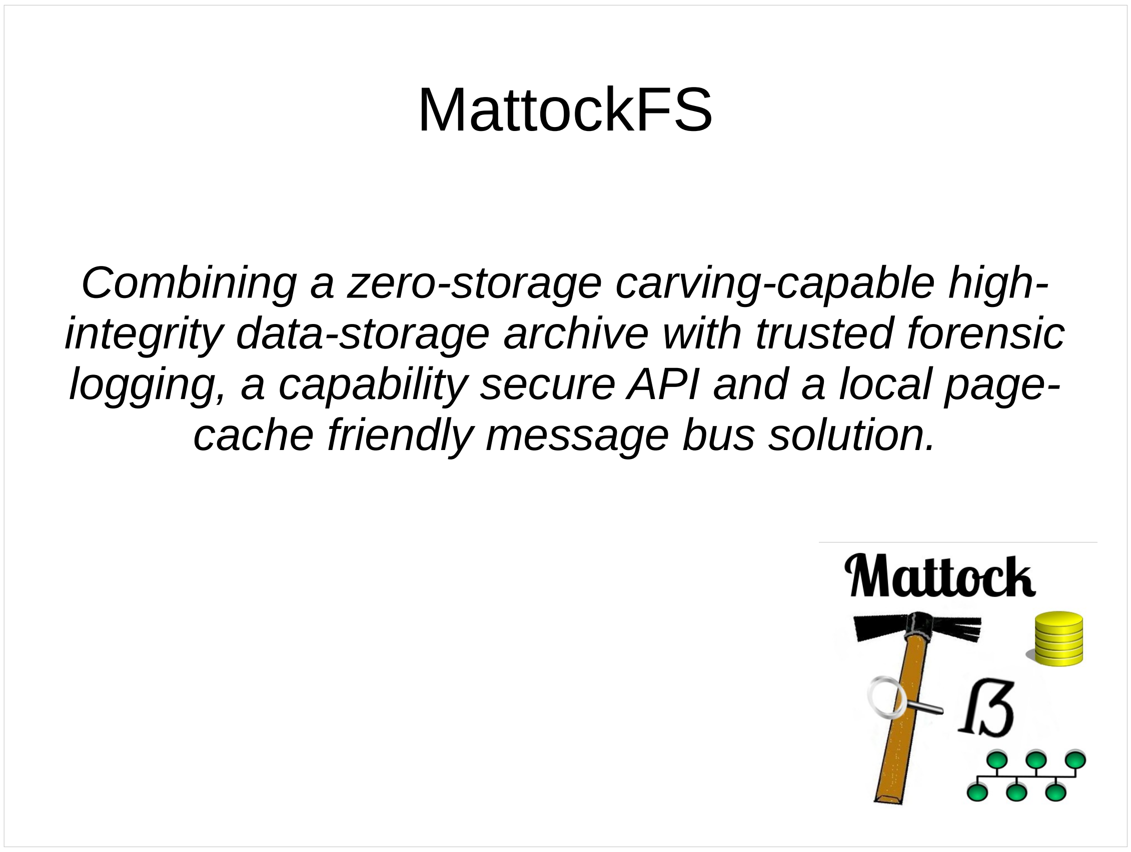


MattockFS: Hooks

- Hooks for forensic frameworks
  - Framework based routing functionality
  - Pressure data for throttling new-data input
  - Queue size for throttling
- Hooks for multi-server setup
  - Geared to locality of data
  - Migrating CPU intensive jobs

There are some things that MattockFS only facilitates in by providing hooks. MattockFS expects the module framework on top of it to implement routing and file-type determination functionality. It provides the routing functionality with a possibility to maintain state. Also the module framework is expected to implement some form of throttling, especially for entering new data into the system, and to a lesser extent, adding new messages into the system. MattockFS provides data about how filled up the page-cache could be about to get, and about how much data and messages are queued for the different modules.

Finally MattockFS provides hooks for creating multi-server setups. These are geared to respecting locality of data concerns and migrating only the most CPU intensive data to other servers.



MattockFS

Combining a zero-storage carving-capable high-integrity data-storage archive with trusted forensic logging, a capability secure API and a local page-cache friendly message bus solution.

To wrap things up, this basically in short describes what MattockFS can provide. If you have any data-intensive problems that require a message-bus solution, MattockFS is an option you might want to explore. MattockFS is geared primarily at computer forensics, but should be suitable also for non-forensic data-intensive problems that favor a worker approach to concurrency