An Efficient Technique for Enhancing Forensic Capabilities of Ext2 File System

By

Mridul Sankar Barik, Gaurav Gupta, Shubhro Sinha, Alok Mishra, and Chandan Mazumdara

Presented At

The Digital Forensic Research Conference

DFRWS 2007 USA Pittsburgh, PA (Aug 13th - 15th)

DFRWS is dedicated to the sharing of knowledge and ideas about digital forensics research. Ever since it organized the first open workshop devoted to digital forensics in 2001, DFRWS continues to bring academics and practitioners together in an informal environment. As a non-profit, volunteer organization, DFRWS sponsors technical working groups, annual conferences and challenges to help drive the direction of research and development.

http:/dfrws.org

DFRWS 2007

An Efficient Technique For Enhancing Forensic Capabilities Of Ext2 File System

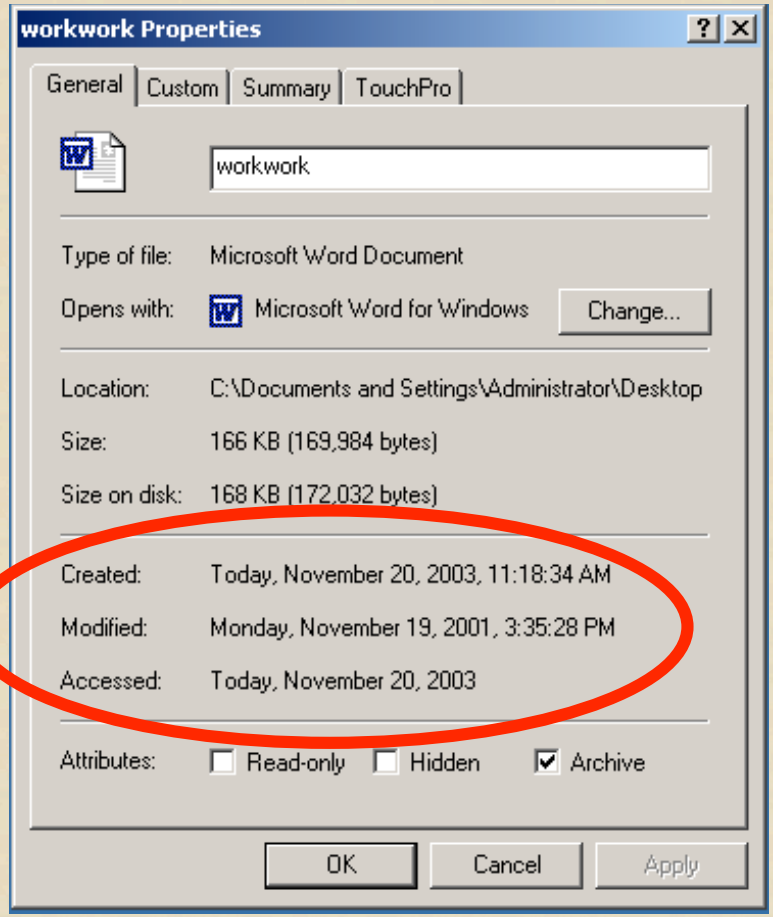
Gaurav Gupta

M.S.Barik, S.Sinha, A. Mishra, Chandan Mazumdar,

India

Definition

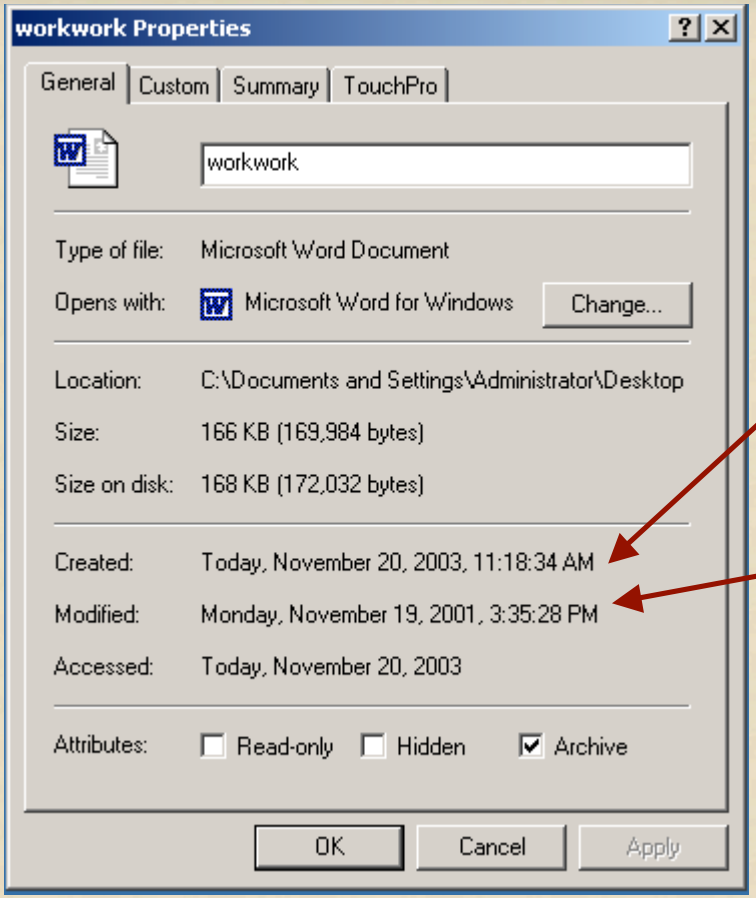
- Modification,
- Access and
- Creation
- Date and Time Stamps
- MAC DTS of Digital Data



Motivation and Goals

- Investigating cases using existing forensic tools
- Questions that baffled me were?
  - How the creation date and time of a file could be later than modified time?

Examples



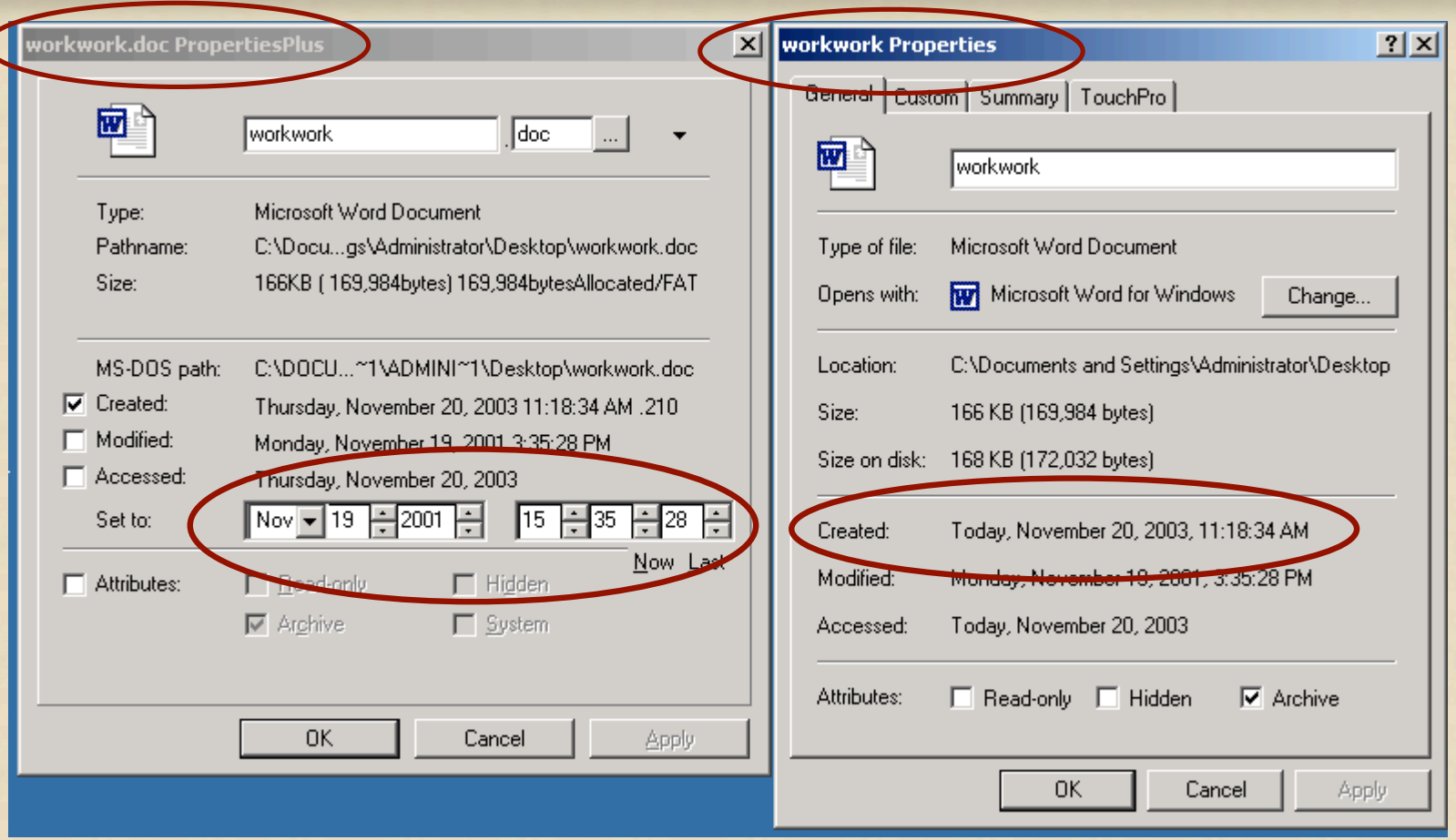
How creation date and time of a file could be

later than modification date and time????

Motivation and Goals

- Investigating Cases using existing forensic tools
- Questions that baffled me were?
  - How the creation time of a file could be later than modified time?
  - How easy, it is to change MAC DTS associated with a file?

Examples



Tools With Great User Interface

Motivation and Goals

- Investigating Cases using existing forensic tools
- Questions that baffled me were?
  - How the creation time of a file could be later than modified time?
  - How easy, it is to change MAC DTS associated with a file?
  - Can MAC DTS be a fundamental digital evidence?

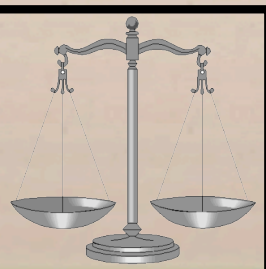
Observation

- Many speakers during DFRWS 2007 used MAC DTS and assumed them to be trustworthy
- Ensuring authenticity of MAC DTS will help these research gain solid backing to their work.
- Goals and objectives are derived out to practical necessities of investigators.

Problem

- Digital Data is editable, perfectly replicable or mutable.
- Change in MAC DTS of Digital Data with ease pose very serious challenge in front of Digital Forensic Community.
- Criminals take malicious and unlawful advantage by tampering MAC DTS.
- Authenticity, trustworthiness and tamperproof ness are key essentials for admissible Digital Evidence.

Forensic Science



Forensic Science is the application of science to law.

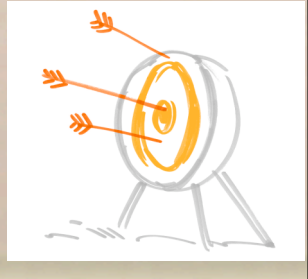
The Challenging aspect of Forensic Science is to link Crime with the Criminal based on Forensic Principles.

This will help in Preventing the further commencement of crime.

Constraints in Current File System

- Data in inodes and FAT is overwritten.
- Inodes only keep MAC information of last access.
- Convenience vs Forensic Readiness
- How can we get information about access trail?
  - Solution
    - Modify system calls that updates MAC times to capture MAC trail

Basic Objective



Since MAC DTS could form the fundamental digital evidence in almost all cases of computer frauds and cyber crimes. The basic objective of this study are:

- Pre-installed tool (LKM/ Enhanced file system).
- Establishing Authenticity by finding policy voilation/ discrepancies in existing MAC DTS.
- This will help in in Reliable Reconstruction of Sequence of Events.

Two Scenarios

1. With pre-installed plugin/ modifying kernel/ installing a hardware tool.

2. When we don't have any pre-installed software/plugin/hardware to support authenticity of MAC DTS

Desired features

- Tool should be able to store the MAC trail however long the trail may be.
- MAC trails should be kept in such a manner that normal users do not even get an idea about existence of such a trail logging mechanism in their file system.
- The trail should be easily accessible to the administrator.
- The MAC trail logging mechanism should not degrade the system performance considerably.

The Basic Question?

? ? ? ? ? ?

Where to put MAC trail?

- Obviously not in a file

- Solution
  - In disk random disk blocks
  - Disk blocks connected as a doubly linked list
  - Doubly linked list – to lower list traversal time

- Problem: Where to keep the list head pointer?
- Inode of a file does not have free room for keeping that pointer

The Basic Question?

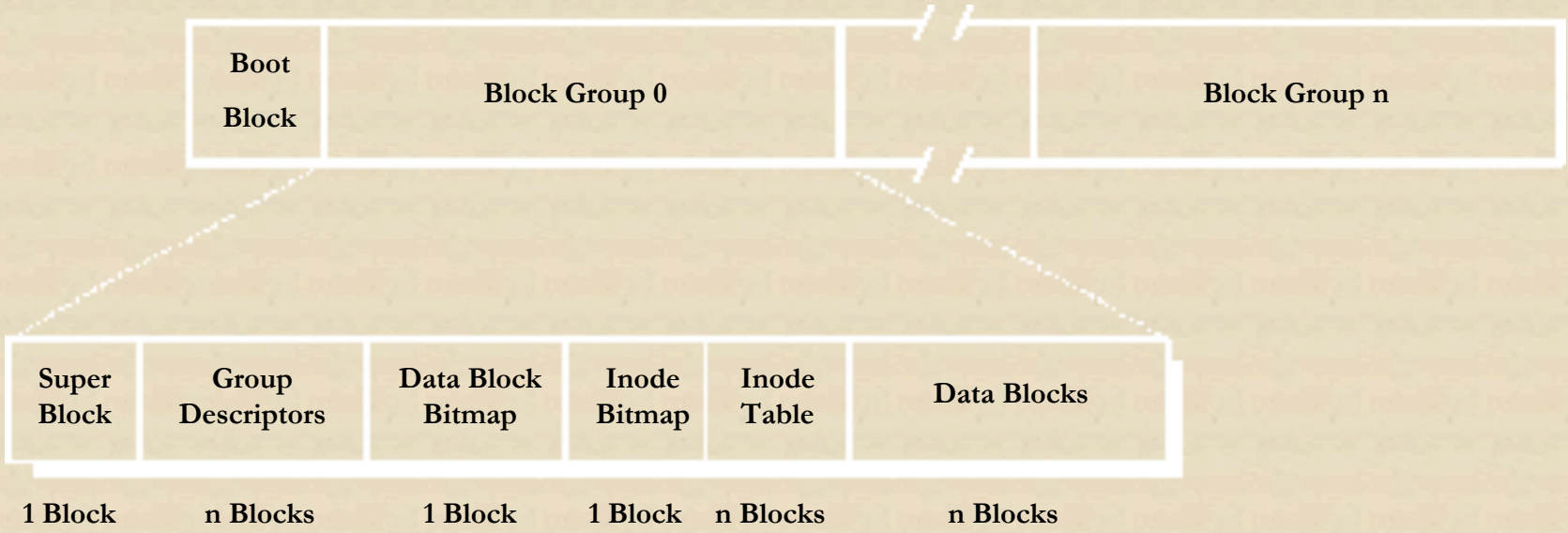
? ? ? ? ? ?

- Solution
  - Keep a directory structure where MAC trail list head pointers for individuals are kept
  - Concept of block groups helped in this regard
  - This directory structure is distributed among different block groups
  - Directory structure in a block group contains information regarding all files whose first block resides in that very block group only
  - Starting block number from where the directory structure starts is kept in the block group descriptor itself

Second Extended File System

- Second Extended (ext2) File System inherits all the basic attributes of the traditional UNIX™ file system
  - Files are represented by inodes
  - Directories are simply files containing a list of entries
  - Devices can be accessed by requesting I/O on special files

Second Extended File System



Layout of ext2 file system

Second Extended File System

Block Bitmaps

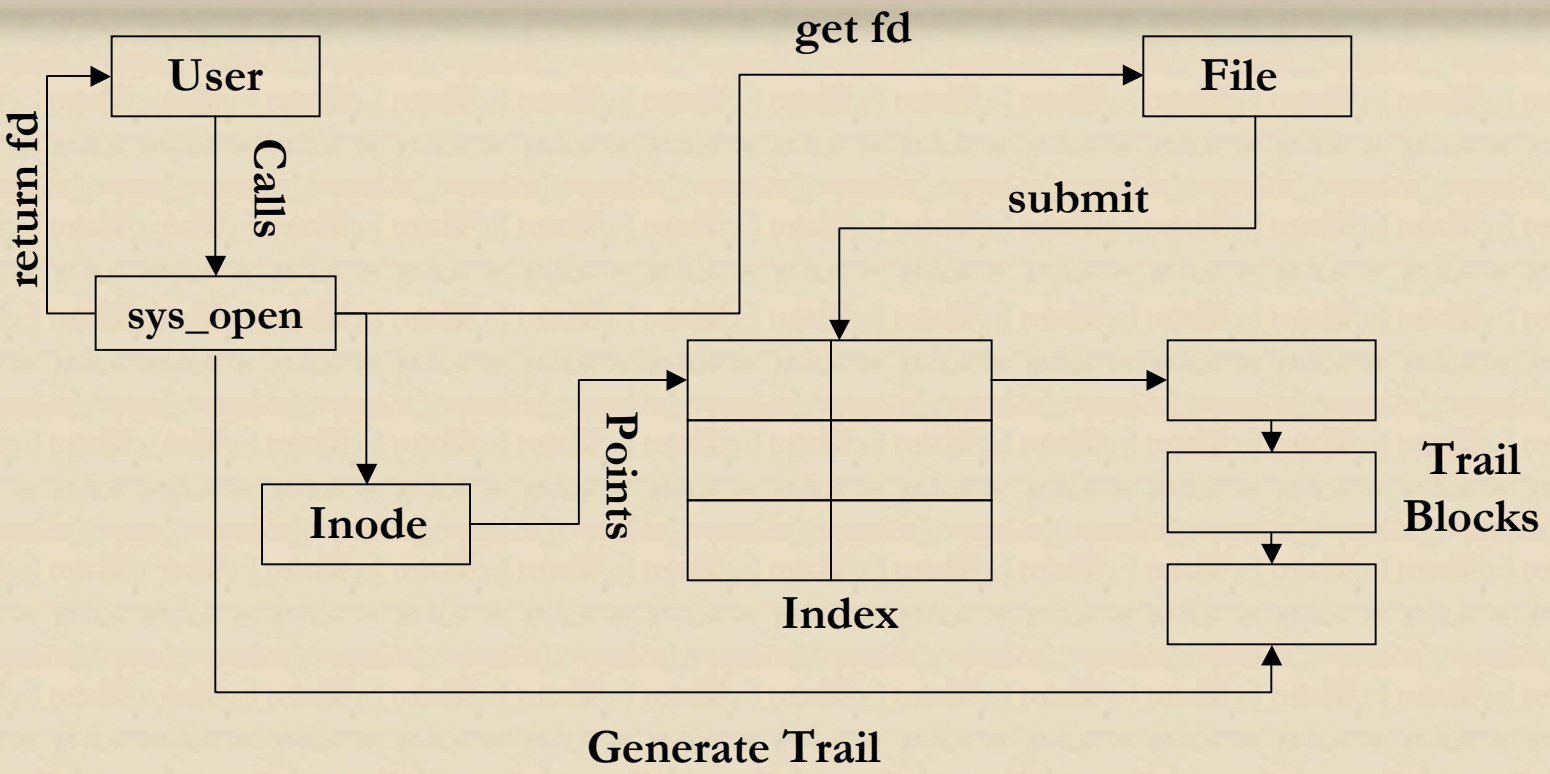
- These are the location map for the file system
- Allocation and de-allocation of blocks depend on this bitmap

Second Extended File System

Example

- 0 at bit position 100 implies that block 100 is empty while 1 implies it has been already allocated to a file

MAC Trail logging



Overview of the MAC Trail logging system

Module Based Approach

- In this approach the logging mechanism is designed over the existing file system
- The logging mechanism is invisible to the VFS so it is hidden from the normal users
- This approach is inspired from the directory organisation of the ext2 file system
- The trails are generated and written in random blocks with each block having a pointer to its previous block and the next block

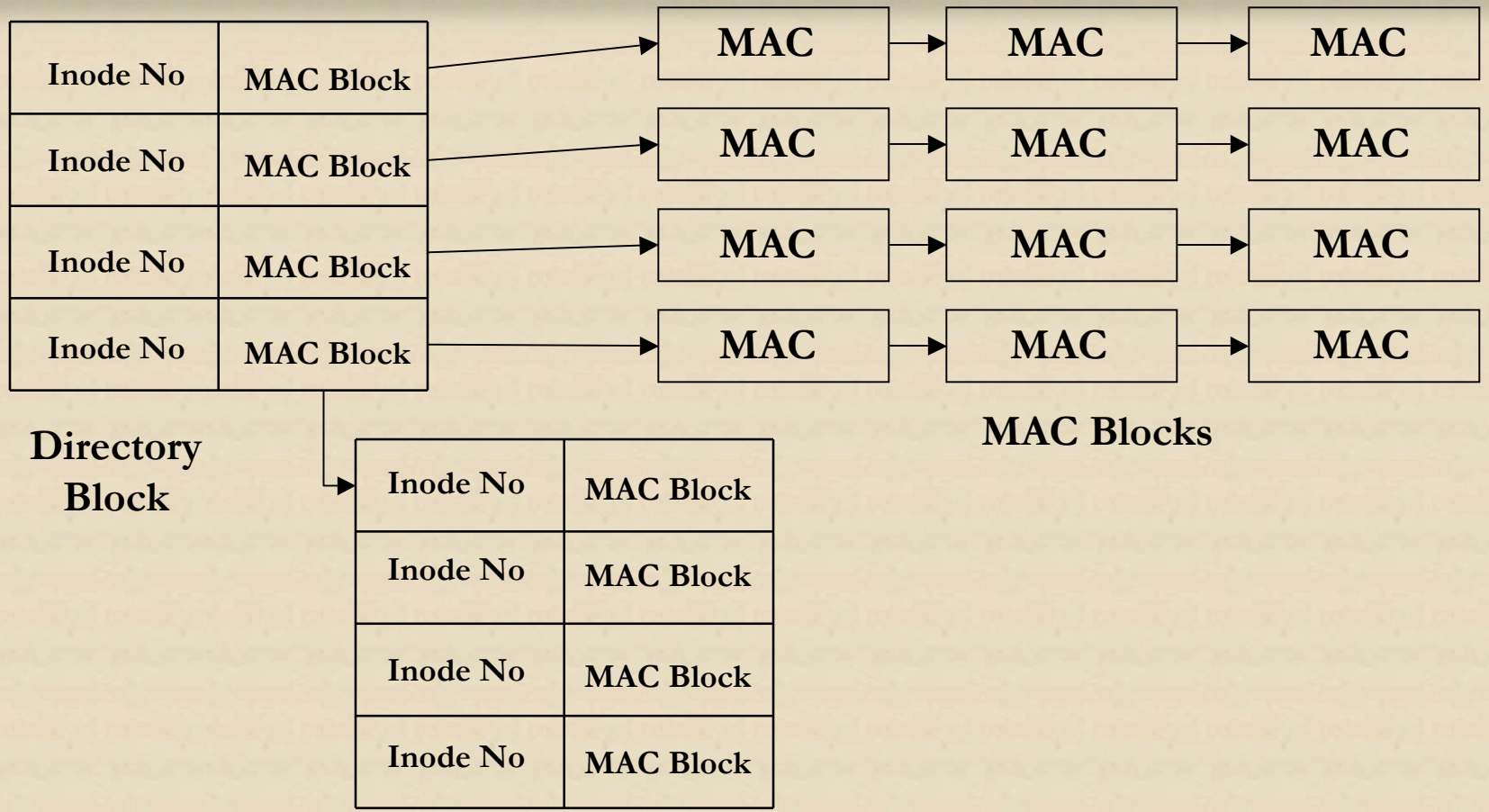
Module Based Approach

- Design
  - In a Linux system every file is accessed by the *sys_open* system call
  - To monitor the file activities the *sys_open* system call needs to be intercepted and modified
  - The file and the MAC trail of that file need to be linked in some way
  - In this approach we create a directory structure that serves as a index for the inode numbers and the starting MAC trail block for every monitored file.

Module Based Approach

- Design (Contd.)
  - Each directory block keeps the <inode no, MAC block no> for files whose first data block lie in the same block group
  - The pointer to the directory block is kept in the block group descriptor of the block group(*bg_reserved[0]*)
  - If the number of entries in the directory block exceeds the block size a new directory block is generated randomly and linked with the first directory block.
  - The MAC blocks are also placed in linked lists with each block keeping pointers to previous and next blocks

Module Based Approach



Overview of the Module Implementation

Module Based Approach

To generate the MAC trail the file first needs to be submitted for monitoring.

Module Based Approach

This is done using a system call

- Activities performed by the system call:
  - Locate the first data block for the file
  - Get the group descriptor for the corresponding block group
  - *bg_reserved[0]* points at the directory block for the corresponding group. If not then allocate a directory block and insert entry corresponding to the file and update group descriptor accordingly (i.e. *bg_reserved[0]*)
  - If an entry for the file doesn't exit then an random empty block is allocated for MAC trail of that file and the corresponding <inode no, MAC_block> is inserted into the directory block

Module Based Approach

- To increase the performance of the file system all data corresponding to MAC trail generation are cached in the VFS inode.
- The fields that are used are:
  - *i_generation* : For keeping the current MAC block
  - *i_sock* : For marking the file as monitored.
- After updating fields the usage counter of the inode are incremented so that they remain in the memory for entire time the system is up

Module Based Approach

- Trail generation
  - To generate and write trail into persistent storage the *sys_open* system call is intercepted and modified as:
    - On call the VFS inode of the file is located and checked whether monitored or not. *(i_sock == 0)*
    - If monitored then the current trail writing block is retrieved from the VFS inode *(i_generation)*
    - The trail is then generated and written onto disk.
    - After completion of the above activities the normal open call is resumed.

Important Functions

- The function locate_mac_dir_list returns a pointer to the head of the MAC directory list of a monitored file.
- The function get_group_desc returns the block group descriptor for a given block number.
- The function get_free_block finds a random block and returns a pointer to it if it is free.
- Modified sys_open system call
- To submit a file for monitoring we have created a system call submit_file which takes the file path as an argument

Advantages of Module Based Approach

- Can be implemented in any Linux kernel with LKM support.
- Works on existing ext2 file system without any modification.
- Logging depends upon will of the implementer. He/She can stop or start logging at will.
- This feature can easily be ported to any ext2 derived file system like ext3 etc. with little modification at all.

Disadvantages of Module Based Approach

- VFS information gets deleted with every boot.
- Have to re-initiate the VFS of monitored file at every boot. This is quite time consuming.
- Requires another LKM that would bring the MAC disk meta-data to in-memory VFS inode.
- System could become considerably slow
- Solution????
  - Editing the kernel source code

Tampering Frauds

- Intentional Modification or change in Data or metadata.
- Intentional Modification and Change in MAC DTS.
- The MAC DTS frauds are difficult to detect and establish due to inherent non supportive design of File System.
- Falsely Implicating by Tampering MAC DTS

Where it is not applicable

- When origin of file in question is not known
- When file is being created for the first time
- When hard disk is formatted
- When Storage media is very small and used very heavily which does not seems like these days.

Challenges Faced

- In Linux kernel version 2.6, this mechanism does not work as of now as it does not support modification of the sys_call_table kernel data structure.
- Utime system call need to be modified.
- Redundantly storing data in free space
- Finding discrepancies as a result of forced changes in MAC DTS.
- Non Availability of source code of FAT etc.

Jadavpur University
Kolkata

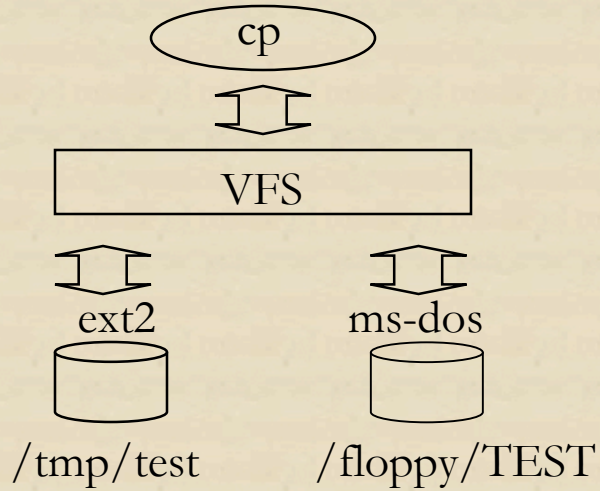
Thank you. . .

… very much.

gaurav1980@gmail.com

Virtual File System

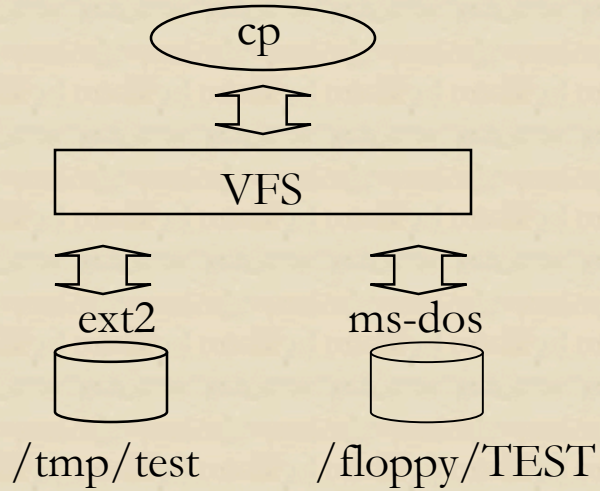
- What is it?
  - It is a kernel software layer that handles all system calls related to a standard Unix filesystem
  - It provides a common interface to several kinds of file systems, viz. ext2, ext3, FAT, NTFS, xenix, ufs, etc.
  - It exists only in the main memory



Virtual File System

Example

- cp /floppy/TEST /tmp/test
- /floppy is ms-dos partition
- /tmp is ext2 partition



Virtual File System

- The VFS data structures are similar to the primitive Linux file system to minimize the response time of Linux file systems

- Components of Virtual File System:
  - The superblock object
    - Stores information concerning a mounted file system. It is created one per file system
  - The inode object
    - Stores general information about a specific file. It is created one per file as and when required

Virtual File System

- The file object
  - Stores information about the interaction between an open file and a process. This information exists only in kernel memory during the period each process accesses a file.
- The dentry object
  - Stores information about the linking of a directory entry with the corresponding file.

Each disk-based file system stores this information in its own particular way on disk.

Virtual File System

- The super block object
  - One per file system
  - Contains useful file system information
  - Stays in memory the entire duration of mounted file system
  - Contains a pointer to in-core disk super block if available
  - For non-Linux file system like FAT, NTFS the entries of super block objects are dynamically calculated and filled based on the corresponding disk structure of the file system
  - Required for speedy access of the file system

Virtual File System

- The Inode objects
  - It is generated whenever a file is accessed
  - It is one per file
  - Contains the inode number of the file in the file system.
  - Also contains the last MAC times of a file
  - Exists in the main memory only when required
  - Present in the main memory as hash queue linked to other VFS inode objects
  - Allocated and de-allocated in LRU fashion
  - Contains some unused fields which would help us in trail generation.

Problems of the ext2 File System

- Doesn’t keep creation time of the file. To get the creation time we need to track the first inode change time. This would obviously be the creation time of the file
- No file activity logging mechanism. The lack of logging mechanism is evident in all the standard file systems