4.4-Security (CIA)

4.4.1-Data integrity

Reliable access to data is a prerequisite for most computer systems and applications. Several factors cause unexpected or unauthorized modifications to stored data. Data can get corrupted due to hardware or software malfunctions. Disks errors are common today[3] and storage software that exists in typically not designed to handle a large class of these errors. A minor integrity violation, when not detected by the higher-level software on time, could cause further loss of data. For example, a bit-flip while reading a file system inode bitmap could cause the file system to overwrite an important file. So, prompt detection of integrity violations is vital for the reliability and safety of the stored data.

Integrity violations could also be caused by malicious intrusions. Large classes of attacks are caused by malicious modifications of disk data. An attacker that has gained administrator privileges could potentially make changes to the system, like modifying system utilities (e.g. */bin* files or daemon processes), adding back-doors or Trojans, changing file contents and attributes, accessing unauthorized files, etc. Such file system inconsistencies and intrusions can be detected using utilities like *Tripwire*[4,5,6].

There are different notations of integrity in storage. File system consistency is one of the common ones. Most file systems today come with integrity checking utilities such as the *UNIX* ***fsck*** that perform a scan through the storage device to fix logical inconsistencies between data and meta-data. (Tools such as ***fsck*** are often said to be performing "sanity" checking.).

This reduces the like-hood of file corruption and wasted disk space in the event of system crash. Advanced methods like journaling [7] and transactional file systems [8] ensure file system consistency even in the event of unexpected systems faults.

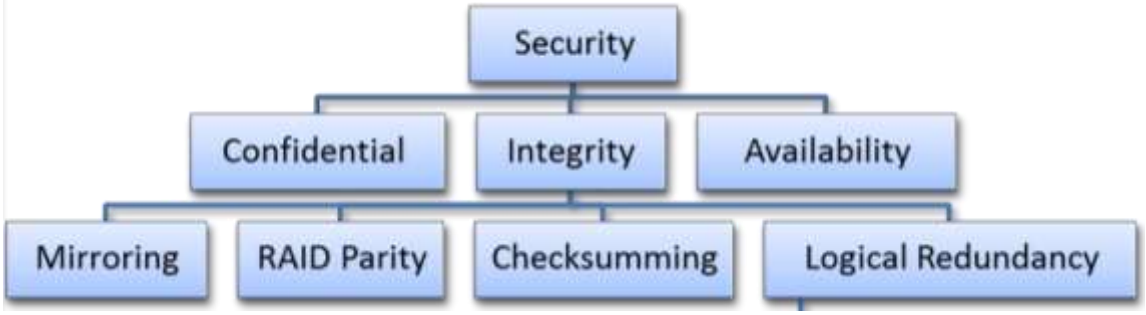
Why Data Integrity is Important ?

Data corruption, although rare, can have catastrophic effects on data-intensive transactional applications, often crippling operations during the many hours it can take to detect, troubleshoot, and repair the problem. On occasion, data corruption can go undetected for months invalidating backups and increasing the difficulty of recovery.

To overcome data corruption we add information (metadata) to the data. Examples of such protection metadata include error correcting code (ECC) and cyclical redundancy checks (CRC). Historically, the most expensive hosts, storage and communication systems have internally used protection metadata. With advances in technology, such approaches to reducing corruption have begun to be incorporated into less expensive commodity hardware[12].

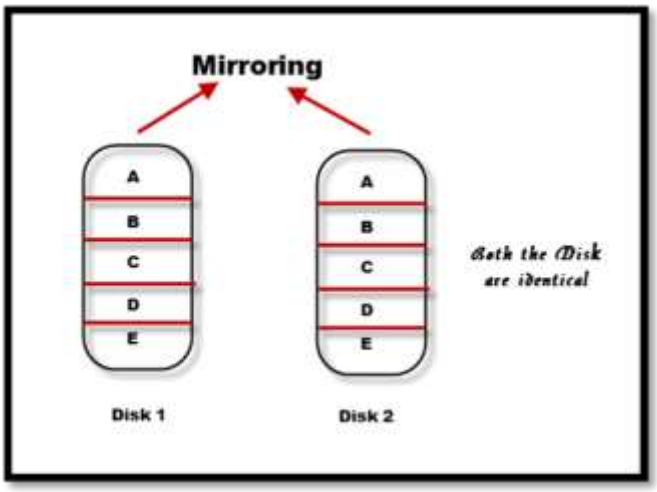
Techniques

Techniques like mirroring, parity, or checksumming can be used to detect data integrity violations at the file or block level. Cryptographic hash functions could even detect malicious forging of checksums.



1. Mirroring

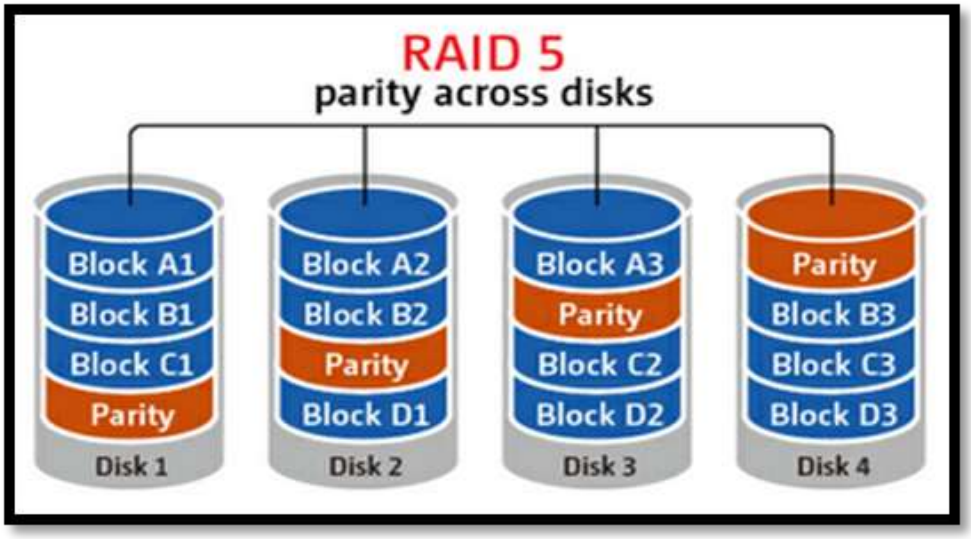
One simple way to implement integrity verification is data replication or *mirroring*. By maintaining two or more copies of the same data in storage device, integrity checks can be made by comparing copies. An integrity violation in one of the copies can be easily detected using this method. While implementing this is easy, this method is inefficient both in terms of storage space and in terms of time. Mirroring can detect integrity violations caused by data corruption due to hardware errors, but cannot help in recovering from the damage, as a discrepancy during comparison does not provide information about which of copies is legitimate. Recovery is possible using majority rules if the mirroring is 3-way or more. Mirroring can be used to detect integrity violations, which are caused due to data corruption and generally not malicious modification of data. A malicious user who wants to modify data can easily modify all copies of the data, unless the location of the copies is maintained in a confidential manner. Mirroring also cannot detect integrity violations caused by user errors, because in most cases user modification are carried out in all mirrors. RAID-1 uses mirroring to improve storage reliability, but does not perform online integrity checks using the redundant data.



2 RAID Parity

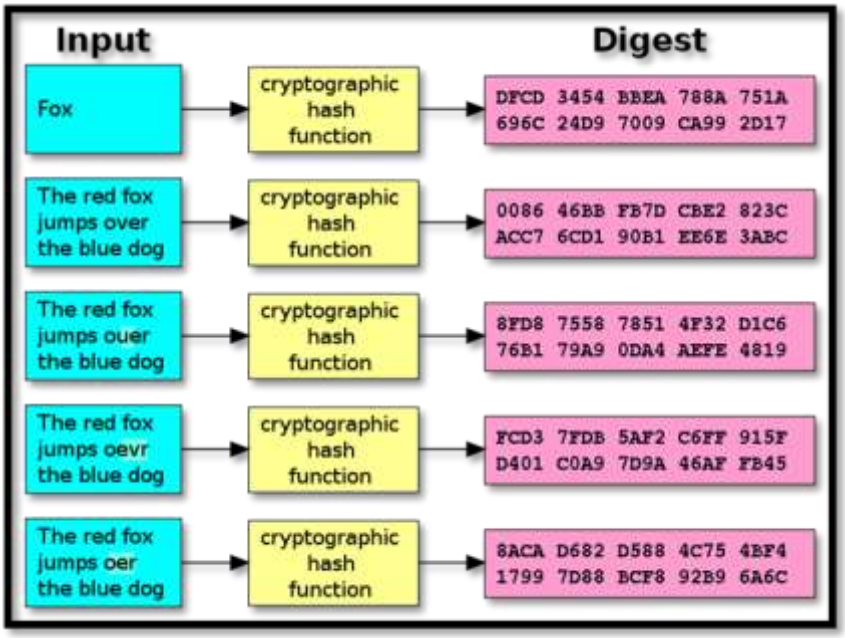
Parity is used in RAID-3, RAID-4 and RAID-5 [9] levels to validate the data written to the RAID array. Parity across the array is computed using the XOR (Exclusive OR) logical operation. XOR parity is a special kind of erasure code. The basic principle behind erasure codes is to transform N blocks of data, into N + M blocks such that upon loss of any M blocks of data, they can be recovered from the remaining N blocks, irrespective of which blocks are lost. The parity information in RAID can rather be stored on a separate, dedicated drive, or be mixed with the data across all the drives in the array. Most RAID schemes are designed to operate on fail-

stop disks. Any single disk failure in RAID (including the parity disk) can be recovered from the remaining disks by just performing an XOR on their data. This recovery process is offline in natural. Although the parity scheme in RAID does not perform online integrity checks, it is used for recovering from a single disk failure in the array. The organization of RAID-5 parity is shown in Figure next page.



3 Checksumming

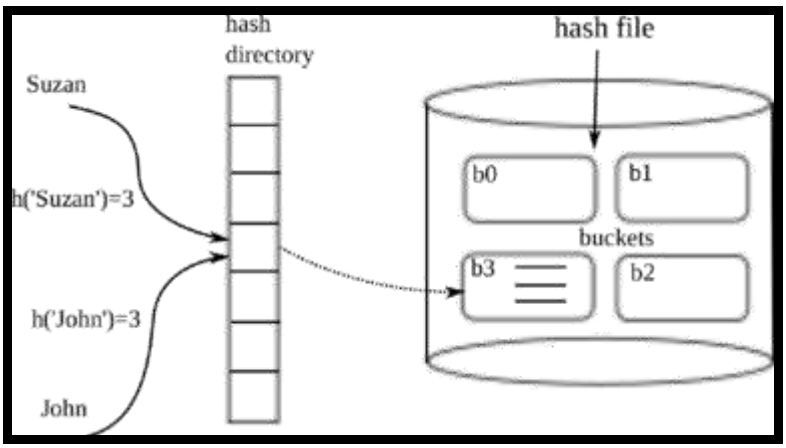
Checksumming is a well know method for performing integrity checks. Checksums can be computed for disk data and can be stored persistently. Data integrity can be verified by comparing the stored and the newly computed values on every data read. Checksums are generated using a hash function. The use of cryptographic hash function has become a standard in internet applications and protocols. Cryptographic hash functions map string of different lengths to short fixed size results. These functions are generally designed to be collision resistant, which means that finding two strings that have the same hash result should be infeasible. In addition to basic collision resistant, functions like MD5 and SHA 1 also have some properties like randomness. HMAC is specific type of a hashing function where the hash generated is cryptographically protected. It works by using an underlying hash function over a message and a key, thereby detecting unauthorized tampering of checksum values. It is currently one of the predominant means of ensuring that secure data is not corrupted in transit over insecure channels (like the internet). This can also be used in the context of storage systems to ensure integrity during read.



4 Logical Redundancy

Techniques that involve logical redundancy exploit the semantics of data to verify integrity. Logical redundancy mechanisms can be used to perform integrity checks by exploiting the inherent semantic redundancy that already exists in the data. An example of a system that employs logical redundancy is the Pilot operating system whose file system meta-data is *advisory* in nature in that the entire meta-data of a file can be regenerated by scanning through the file data. It stores meta-data for faster access to the file, but it can potentially use this information to perform integrity checks on the file when needed by just comparing the regenerated value and the stored value of the meta-data. Similarly, in some size changing stackable file systems index files that store mapping information for sizes are totally constructible from the file data. This can also be used for performing online integrity checks to detect inconsistencies between file sizes. Logical redundancy is a common method used by databases to maintain the semantic integrity of information they store. I have identified the general technique of logical redundancy can be exploited by storage systems at various levels, to implement seamless integrity checking. The main advantage of this method is that there is minimal additional overhead imposed by the integrity checking mechanism, because the storage and perhaps retrieval of redundant information is already performed by the system for other purposes.

Hash File



A Hashed File is a reference table based on key fields which provides fast access for lookups. They are very useful as a temporary or non-volatile program storage area. An advantage of using hashed files is that they can be filled up with remote data locally for better performance. To increase performance, hashed files can be preloaded into memory for fast reads and support write-caching for fast writes.

A hash function is any function that can be used to map digital data of arbitrary size to digital data of fixed size, with slight differences in input data producing very big differences in output data.[11] The values returned by a hash function are called hash values, hash codes, hash sums, or simply hashes. One practical use is a data structure called a hash table, widely used in computer software for rapid data lookup. Hash functions accelerate table or database lookup by detecting duplicated records in a large file. A cryptographic hash function allows one to easily verify that some input data matches a stored hash value, but makes it hard to reconstruct the data from the hash alone. Hash functions are related to (and often confused with) checksums, check digits, fingerprints, randomization functions, error-correcting codes, and ciphers. Although these concepts overlap to some extent, each has its own uses and requirements and is designed and optimized differently. The Hash Keeper database maintained by the American National Drug Intelligence Center, for instance, is more aptly described as a catalog of file fingerprints than of hash values [14].

REFERENCES

[1] Title 44 of United States Code.

[2] Perrin, Chad. "The CIA Triad". Retrieved 31 May 2012.

[3] V. Prabhakaran, N. Agrawal, L. N. Bairavasundaram, H. S. Gunawi, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. IRON File Sysetms. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP '05)*, Brighton, UK, October 2005.

[3] As cited in: Francis, Bob. "SNIA nails down ILM definition." InfoWorld. 1 November 2004: 14.

[4] G. Kim and E. Spafford. Experiences with Tripwire: Using Integrity Checkers for Intrusion Detection. In *Proceedings of the Usenix System Administration, Networking and Security (SANS III)*, 1994.

[5] G. Kim and E. Spafford. The Design and Implementation of Tripwire: A File System Integrity Checker. In *Proceedings of the 2nd ACM Conference on Computer Commuications and Society (CCS)*, November 1994.

[6] Tripwire Inc. Tripwire Software. www.tripwire.com.

[7] R. Hagmann.Reimplementing the cedar file system using logging and group commit. *ACM SIGOPS Operating Systems Review*,21(5):155.162, 1987.

[8] E.Gal and S.Toledo. A transactional fiash file system for microcontrollers. In *Usenix '05: Proceedings of the Usenix Annual Technical Conference*, pages 89.104, Anaheim, CA., USA, 2005. Usenix.

[9] D. Patterson, G. Gibson, and R. Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of the ACM SIGMOD*, pages 109.116, June 1988.

[10] D. D. Redell, Y. K. Dalal, T. R. Horsley, H. C. Lauer, W. C. Lynch, P. R. McJones, H. G. Murray, and S. C. Purcell. Pilot: An operating system for a personal computer. In *Proceedings of the 7th ACM Symposium on Operating Systems Principles (SOSP)*, pages 106.107, 1979.

[11] Ovie Carroll and Mark Krotoski, "Using 'Digital Fingerprints' (or Hash Values) for Investigations and Cases Involving Electronic Evidence," 62 United States Attorneys' Bulletin 44-82 (May 2014) .

[12] Why Data Integrity is important to you , page 3, No. 08-485 11/07.

[13] Wikipedia , B-Tree, http://en.wikipedia.org/wiki/B-tree .

[14] Wikipedia, Hash Function, http://en.wikipedia.org/wiki/Hashfunction .