An Overview of Intrusion Tolerance Techniques

Wenbing Zhao
Department of Electrical and Computer Engineering, Cleveland State University, USA

INTRODUCTION

Web and cloud based systems are expected to be highly available and trustworthy, i.e., they are accessible any time a user wants, they always provide correct services, and they never reveal confidential information to an unauthorized party. To meet such high expectation, the system must be carefully designed and implemented, and rigorously tested (for intrusion prevention). However, considering the intense pressure for short development cycles and the widespread use of commercial-off-the-shelf software components, it is not surprising that software systems are notoriously imperfect. The vulnerabilities due to insufficient design and poor implementation are often exploited by adversaries to cause a variety of damages, e.g., crashing of the system, leaking of confidential information, modifying or deleting of critical data, or injecting of erroneous information into a system.

This observation prompted the research on intrusion tolerance techniques (Castro & Liskov, 2002; Deswarte et al., 1991; Verissimo et al., 2003, Yin et al., 2003). Such techniques can tolerate intrusion attacks in two respects: (1) a system continues providing correct services (may be with reduced performance) and (2) no confidential information is revealed to an adversary. The former can be achieved by using the replication techniques, as long as the adversary can only compromise a small number of replicas. The later is often built on top of secrete sharing and threshold cryptography techniques. Plain replication is often perceived to reduce the confidentiality of a system, because there are more identical copies available for penetration. However, if replication is integrated properly with secrete sharing and threshold cryptography, both availability and confidentiality can be enhanced.

BACKGROUND

In this section, we introduce some basic security and dependability concepts and techniques related to intrusion tolerance. A secure information system is one that exhibits the following properties (Pfleeger & Pfleeger, 2002):

- Confidentiality. Only authorized users have access to the information.
- Integrity. The information can be modified only by authenticated users in authorized ways. Any unauthorized modification can be detected.
- Availability. The information is available whenever a legitimate user wants to access it.

Confidentiality is often achieved by using encryption, authentication, and access control. Encryption is a reversible process that scrambles a piece of plaintext into something uninterpretable. Encryption is often parameterized with a security key. To decrypt, the same or a different security key is needed. Authentication is the procedure to verify the identity of a user that wants to access confidential data. Access control is used to restrict what an authenticated user can access.

Integrity can be protected by using secure hash functions, message authentication code (MAC) and digital signatures. For data stored locally, including the application binary files, a checksum is often used as a way to verify data integrity. The checksum can be generated by applying an oneway secure hash transformation on the data. Before the data is accessed, one can verify its integrity by recomputing the checksum and comparing it with the original one. The integrity of a message transmitted over the network can be guarded by a MAC. A MAC is generated by hashing on both

DOI: 10.4018/978-1-4666-5888-2.ch415

the original message and a shared secret key (and often with a sequence number as well). If it is tampered with, the message can be detected in a way similar to that for checksum. For stronger protection, a message can be signed by the sender. A digital signature is produced by first hashing the message using a secure hash function, and then encrypting the hash using the sender's private key.

High availability is achieved by using replication, checkpointing and recovery techniques. Replication is a technique that relying on running redundant copies of an application so that if one copy fails, the services can be provided by the remaining copies. Checkpointing means to take a snapshot of the state of a replica. The saved state can be used to bring a new or a restarted replica up to date. Checkpointing is also useful to avoid log buildup (when a checkpoint is taken, all previously logs can be garbage collected). Recovery techniques concern the tasks of removing faulty replicas, repairing them, and reintegrating them back to the system.

INTRUSION TOLERANCE TECHNIQUES

Intrusion tolerance is built on top of two fundamental techniques: replication and secret sharing/threshold cryptography (Deswarte et al., 1991). In the context of intrusion tolerance, a very general fault model must be used because a compromised replica might exhibit arbitrary faulty behaviors. Such a fault model is often termed as Byzantine fault (Lamport et al., 1982).

Byzantine Fault Tolerance

An intrusion attack might bring a service down, or compromise the integrity of a service. An effective defense is to introduce redundancy into the system, i.e., to replicate critical components in the system. Assuming that an intrusion attack can only penetrate a small fraction of the replicas, the service availability and integrity can be preserved by the remaining correct replicas. However, achieving this goal is not trivial – we must ensure consistent execution of all correct replicas despite the attacks launched by faulty replicas.

A Byzantine faulty replica may use all kinds of strategies to prevent the normal operations of a replica. In particular, it might propagate conflicting information to other replicas or components that it interacts with. To tolerate f Byzantine faulty replicas in an asynchronous environment, we need to have at least 3f+1 number of replicas (Castro & Liskov, 2002). An asynchronous environment is one that has no bound on processing times, communication delays, and clock skews. Internet applications are often modeled as asynchronous systems. Usually, one replica is designated as the primary and the rest are backups.

There are two different approaches to Byzantine fault tolerance. In a Byzantine quorum system (Malkhi & Reiter, 1997), read and write operations issued by some clients are applied on a set of data items (which consists of the state of a service). It is assumed that the read and write operations are synchronized. A read operation retrieves information from a quorum of correct replicas and a write operation applies the update to a quorum of correct replicas. In a system with 3f+1 replicas, a quorum can be formed by 2f+1 replicas so that any two quorums overlap by at least f+1 replica, among which at least one is not faulty. This guarantees the correct operations of the quorum-based system.

A more general method is the state-machine based approach (Schneider, 1990). In the state-machine based approach, a replica is modeled as a state machine. The state change is triggered by remote invocations on the methods offered by the replica. This approach is applicable to a much wider range of applications. Consider a client server application where the server is replicated using the state-machine based approach (Castro & liskov, 2002). The client first sends its request to the primary replica. The primary then broadcasts the request message to the backups and also determines the execution order of the message. To prevent a faulty primary from intentionally delaying a message, the client starts a timer after it sends out a request. It waits for f+1 identical replies from different replicas. Because at most f replicas are faulty, at least one reply must come from a correct replica. If the timer expires before it receives a correct reply, the client broadcasts the request to all server replicas. This enables the correct replicas to detect the primary failure so that a new primary can be elected (this is often called a view change).

All correct replicas must agree on the same set of input messages with the same execution order. In other words, the request messages must be delivered to the replicas reliably in the same total order. To understand this better, consider the scenario illustrated in Figure