Category: Software & Systems Design

Building Secure and Dependable Online Gaming Applications

Bo Chen
Cleveland State University, USA

Wenbing Zhao
Cleveland State University, USA

INTRODUCTION

Online gaming has become a multibillion-dollar industry. The security and dependability of such games are critical for both the game providers and honest game players alike. Essential to all such applications is the use of random numbers; for example, random numbers are needed to shuffle cards. For obvious reasons, if the hands can be predicated, players could gain unfair advantages. The nature of this type of applications poses great challenges in increasing their availability while preserving their integrity (Arkin, Hill, Marks, Scjmod, & Walls, 1999; Viega & McGraw, 2002; Young & Yung, 2004).

Byzantine fault tolerance (BFT; Castro & Liskov, 2002) is a well-known technique to tolerate various malicious attacks to online systems and it often involves state machine replication (Schneider, 1990). However, state machine replication assumes that all replicas are deterministic, which is not the case for online gaming applications. In this article, we elaborate how we address this dilemma using an online poker application that uses a pseudorandom number generator (PRNG) to shuffle the cards as an illustrating example. We propose two alternative strategies to cope with the intrinsic application nondeterminism. One depends on a Byzantine consensus algorithm and the other depends on a practical threshold signature scheme. Furthermore, we thoroughly discuss the strength and weaknesses of these two schemes.

BACKGROUND

In this section, we provide a brief introduction of PRNG, the entropy concept, and the methods to collect and enhance entropy.

A PRNG is a computer algorithm used to produce a sequence of pseudorandom numbers. It must be initialized by a seed number and can be reseeded prior to each run. The numbers produced by a PRNG are not truly random because computer programs are in fact deterministic machines. Given the same seed, a PRNG will generate the same sequence of numbers. Consequently, if the seed is known to an adversary, then one can simulate the stream of random numbers (Young & Yung, 2004). Therefore, to make the random numbers unpredictable, it is important that the seeds to the PRNG cannot be guessed or estimated. Ideally, a highly random number that is unpredictable and infeasible to be computed is required to seed the PRNG in order to produce a sequence of random numbers.

The activity of collecting truly random numbers is referred to as collecting entropy by cryptographers (Young & Yung, 2004). Entropy is a measure of how much real randomness is in a piece of data, for example, using the outcome of coin flipping as 1 bit of entropy. If the coin toss is perfectly fair, then the bit should have an equal chance of being a 0 or a 1. In such a case, we have a perfect 1 bit of entropy. If the coin flip is slightly biased toward either head or tail, then we have something less than a bit of entropy. Entropy is what we really want when we talk about generating numbers that cannot be guessed. In general, it is often difficult to figure out how much entropy we have, and it is usually difficult to generate a lot of it in a short amount of time.

It is a common practice to seed a PRNG with the value of the local clock. Unfortunately, this is not a sound approach to preserve the integrity of the system, as described by Arkin et al. (1999) when telling about how they attacked a Texas Hold'em Poker online game. They show that with the knowledge of the first few cards, they can estimate the seed to the PRNG and subsequently predict all the remaining cards.

TOWARD SECURE AND DEPENDABLE ONLINE GAMING APPLICATIONS

In this section, we describe two possible strategies for building secure and dependable online gaming applications. One depends on a Byzantine consensus algorithm and the other depends on a practical threshold signature algorithm. These two algorithms are aimed at ensuring that all replicas use the same value to seed their PRNGs, with each replica taking entropy from its respective entropy source.

Byzantine Fault Tolerance

A Byzantine fault (Lamport, Shostak, & Pease, 1982) is a fault that might bring a service down or compromise the integrity of a service. A Byzantine faulty replica may use all kinds of methods to prevent the normal operation of a replicated service; in particular, it might propagate conflicting information to other replicas. To tolerate f Byzantine faulty replicas in an asynchronous environment, we need to have at least $3f+1$ 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.

We choose to use the well-known Byzantine fault tolerance algorithm developed by Castro and Liskov (2002). The BFT algorithm has three communication phases in normal operation. During the first phase, the *pre-prepare* phase, upon receiving a request from the client, the primary assigns a sequence number and the current view number to a message and multicasts this *pre-prepare* message to all backups. In the second phase, referred to as the *prepare* phase, a backup broadcasts a *prepare* message to the rest of the replicas after it accepts the *pre-prepare* message. Each nonfaulty replica enters into the *commit* phase, that is, the third phase, only if it receives $2f+1$ *prepare* messages (from different replicas) that have the same view number and sequence number as the *pre-prepare* message, then it broadcasts the *commit* message to all replicas including the primary. A replica commits the corresponding request after it receives $2f$ matching *commit* messages from other replicas. To prevent a faulty primary from intentionally delaying a message, the client starts a timer after it sends out a request and waits for $f+1$ consistent responses from different replicas. Due to the assumption that at most f replicas can be faulty, at least one response must have come from a nonfaulty replica. If the timer expires, the client broadcasts the request to all replicas. Each backup replica also maintains a timer for similar purposes. On expiration of their timers, the backups initiate a view change and a new primary is selected. In the BFT algorithm, a digital signature or an authenticator is employed to ensure the integrity of the messages exchanged, and a cryptographic hash function is used to compute message digests.

The above BFT algorithm must be modified to cope with intrinsic replica nondeterminism. The modified algorithm also consists of three phases, as shown in Figure 1. In the beginning of the first phase, the primary invokes the ENTROPY-EXTRACTION operation to extract its entropy and append the entropy to the *pre-prepare* message. It then

Figure 1. The adapted BFT algorithm to handle entropy extraction and combination