The Secrets of RNG Design
iGaming Business Magazine: March / April – 2009

You've probably heard about one or more online gaming websites falling victim to scandals relating to their Random Number Generator (RNG).

In some cases, external parties found a way to mathematically predict the outcomes of a poorly designed RNG, or they discovered patterns or biases in the game outcomes that could be exploited for gain. In other cases, insiders with intimate knowledge of the RNG found weaknesses that allowed them to cheat the system and keep all the winnings for themselves.

Either way, you can't afford to fall victim to these scandals, no matter how they're perpetrated.

So how do you protect yourself from this hidden threat? Naturally, you need honest and reliable staff, and rock-solid procedures for operating and securing your software environment. But all of these controls will be futile without a properly designed and implemented RNG.

So let's investigate some of the key design aspects for RNGs, and work out solutions for typical exposures that leave online gaming websites vulnerable.

There are two basic types of RNGs: hardware and software. Hardware RNGs are interesting devices that typically come in the form of an electronic card that plugs into a computer; but today we're going to focus instead on software RNGs.

At the heart of every software RNG is an algorithm. Software RNGs are also known as pseudo-RNGs because these algorithms generate game outcomes that only *appear* to be random.

But what exactly is an algorithm, and how does this pseudo-random operation really work?

An algorithm is a complex mathematical equation, which when given any particular input, or 'seed', will in turn produce a specific output. Under the basic operation of a software RNG, each sequential output is sent to the game to create a game outcome, and simultaneously fed back into the algorithm, thus seeding the next output. This repeating process results in a continual string of game outcomes.

This brings us to the first critical element of software RNG design: 'seeding'

Since each outcome is systematically fed back into the algorithm to be used as the seed to create the next outcome, how do you create the first outcome to set the whole process in motion? It's just like the age-old question: which came first, the chicken or the egg? The process of initializing a software RNG upon start-up, which is something like *inventing* the egg, is called 'seeding'.

In order to get things going, the software RNG must first look to a specified location in the software program to find its initial seed value. This value must be sufficiently random and secure to be effectively hidden from attackers.

If an attacker can learn the initial seed value for your RNG, they may be able to predict how the resultant string of outcomes will start. That in mind, a good source of seeding is your first layer of defence against prediction attacks. Your software developer will have to work out the best location for your software RNG to safely and securely find its seed.

The next critical element of software RNG design is the 'period'

Eventually, after a gargantuan number of outcomes have been generated, the algorithm will begin to generate exactly the same string of outcomes all over again.

The number of outcomes generated by the algorithm before it starts to repeat itself is called the 'period'. The period of stronger algorithms is typically far greater than that of weaker algorithms. Since more complex games characteristically demand a greater period from the RNG, the types of games you have should directly impact your choice of algorithm.

Again, your software developer will have to research and select the most appropriate algorithm for your software RNG, fully considering the nature of your games.

Now we'll have to learn about the 'range' of the RNG

Most software RNGs produce extremely large numbers. These are sometimes referred to as the 'raw' numbers generated by the algorithm. The size of these raw numbers tells us the 'range' of the software RNG. Many software RNGs produce 32-bit binary outcomes, which translates to a range of 4,294,967,296 different possible outcomes.

Now I don't know about you, but I haven't played many poker games with four million cards! Naturally, these titanic numbers must be 'scaled' down to more useable values, such as 52 for a standard deck of cards.

'Scaling' is a vital process for any RNG, and a common area for mistakes!

Scaling is achieved by dividing the raw outcomes from the algorithm into a much smaller range of values to be used by the game. This is often easier said than done. Given our example above, consider that 4,294,967,296 is <u>not</u> divisible by 52. Your software developer will have to overcome hurdles such as this through complex mathematical formulas in the software program.

Scaling often wreaks havoc with the quality of the RNG's output, causing biases to particular outcomes. Scaling must be carefully researched, and properly designed, in order to ensure that the software RNG operates as intended.

RNG outcomes need to be 'mapped' before they can be used by the game

Subsequent to the scaling operation, each number must be mapped to a symbol used in a game. For example the number 52 could be mapped to the Ace of Spades in a deck of cards; the number 51 could be mapped to the King of Spades, and so on.

This is another common area for problems in RNGs, as many developers will overlook the importance of mapping. Certain types of bonus features in games can be especially problematic, as many developers will mistakenly introduce biases in the final game outcomes by trying to map multiple scaled numbers to the same mapped value.

Last but not least, the best defence against prediction attacks is 'background cycling'

The final and most critical layer of defence against those who might try to predict the game outcomes is called 'background cycling'. Unfortunately, this is also the most common area for costly mistakes in software RNG design.

You will remember above when we talked about how the algorithm will eventually begin to generate exactly the same string of outcomes all over again. I bet that made you awfully nervous to read that! But don't worry; background cycling actually prevents an algorithm from repeating the same string of outcomes.

Background cycling works in two different ways.

First, with background cycling implemented properly, a software RNG will generate outcomes, at a highly accelerated and variable rate, whether or not outcomes are actually required by a game at any given point in time. That means that an unknown number of outcomes are actually being skipped in the background.

Since an attacker would have no idea how many outcomes have been skipped, it would not be possible for them to observe and record game outcomes over time to reverse-engineer exactly where the RNG is in its period. How could they figure out what the next number will be, when they don't even know what the last number was!

When designed correctly, background cycling works in another way too. It will actually operate, or cycle, during a game that is currently in progress.

Let's take a game that uses multiple separate outcomes, and displays them to the player all at once, like a poker game displaying all five cards dealt face-up to the player. If the player was attempting a prediction attack on the algorithm, we would be giving them too much information if we were to display all five outcomes directly and sequentially from the algorithm. Sometimes, seeing just five numbers in a row is enough to crack the algorithm.

Background cycling should operate in between each of those five cards being drawn, so that they are not five sequential cards in a row. Instead of the attacker seeing five sequential outcomes from the algorithm, they see five separate independent cards. Now the attacker would only be led astray by trying to reverse-engineer the algorithm with those numbers.

Naturally, this all happens in the background, so the player's enjoyment of the game isn't impacted in the slightest. When designed correctly, background cycling provides for fair and safe game play for everyone.

Unfortunately, many developers will either implement their background cycling incorrectly, or leave it out altogether. A lack of background cycling is the number one cause of successful prediction attacks in the online gaming industry.

Whether it was a lack of background cycling, improper scaling that led to a bias, or any number of others things that can go horribly wrong with an RNG, unfair game outcomes have put companies out of business.

These exposures may not be easy to see with the naked eye, but without a doubt there are experts out there looking to find them. Some of those experts are there to help you, like reputable software developers that build safe and reliable games, and Accredited Testing Facilities (ATFs) that evaluate and certify RNGs.

Unfortunately, many of the remaining experts have entirely different plans in mind: prediction attacks on your online gaming website. Their only goal is to help themselves to your profits.

So protect your website, protect your reputation, and protect your RNG! Design and implement your RNG carefully, because it could make all the difference in the world.

Bio

Mr. Noah Turner is the Chief Technical Officer (CTO) of Technical Systems Testing (TST), an internationally recognized Accredited Testing Facility (ATF) offering evaluation and consultation services for both the land-based (traditional / terrestrial) and Interactive gaming, lottery and Information Technology (IT) industries.

Office: +1 (604) 873-5833
Email: nturner@tstglobal.com

OFFICES:
Vancouver *– Suite #420, 1367 West Broadway, Vancouver, British Columbia, Canada, V6H 4A7 //* ***O:*** *+1 (604) 873-5833 //* ***F:*** *+1 (604) 873-1075*
London *– Swan Centre, Fishers Lane, Chiswick, London, England, United Kingdom, W4 1RX //* ***O:*** *+44 (0)2087 474 956 //* ***F:*** *+44 (0)2087 427 967*
Sydney *– Suite #305 / 306, 30 – 40 Harcourt Parade, Rosebery, New South Wales, Australia, 2018 //* ***O:*** *+61(2) 9700 7023 //* ***F:*** *+61(2) 9700 7024*
Melbourne *– Level 28, 303 Collins Street, Melbourne, Victoria, Australia, 3000 //* ***O:*** *+61 (3) 9678 9095 //* ***F:*** *+61 (2) 9700 7024*
Macau *– Macau Number 39, 17F Central Plaza, 61 Avenida de Almeida Ribeiro, Macau, China //* ***O:*** *+853 8291 3992 //* ***F:*** *+853 8291 3889*