Maximilian Eibl, Martin Gaedke. (Hrsg.): INFORMATIK 2017,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 543

Separated Random Number Generators for Virtual Machines

Clemens Fritzsch[1], Jörn Hoffmann[1], Martin Bogdan[1]

Abstract: High quality random numbers are essential for secure servers, alas virtual machines suffer from low entropy. The generation of hard to predict random numbers is a crucial problem. Moreover, existing solutions can pose severe security risks, e.g. by providing covert channels. This work illustrates basic ways to provide high entropy random numbers to virtual machines. Furthermore the security risks and effectiveness of the random number distributions are compared. Finally a new approach is presented. It improves the security of virtualized server environments by assigning each virtual machine a separate TRNG.

Keywords: Security Virtualization Random Numbers TRNG

1 Introduction

The security of server infrastructure is a mayor issue. Especially due to the fact that the average cost of a data breach is estimated to be about $ 4 million [Po16]. As approximately 80 % of the workload on x86 servers are processed in virtualized server environments [BDW16], the security of virtual machines (VMs) attracted much attention.

A vital aspect of ensuring a high level of security is the generation of high entropy random numbers because they are an essential element of cryptography. Since server applications obtain their random numbers from interfaces provided by the server operating system (e.g. /dev/urandom under Linux), much research is focused on the internal random number generation of operating systems. Gutterman et al. showed that random numbers generated at boot time in low entropy environments can be predictable [GPR06]. That a virtual machine is a low entropy environment was proven by Fernandes et al. [Fe13].

Further, reset vulnerabilities are another typical problem of virtualized environments. Reset vulnerabilities arise when a VM is started multiple times from the same disk image or is reverted to a snapshot. As a result the VM has exactly the same state it had once before. Consequently, predicting the random numbers generated by the operating system is possible [Ev14]. Low entropy environments increase the possibility that predictable random numbers are generated. The common element of the problems mentioned above is the scarcity of sources for high entropy random numbers.

A common approach to provide high entropy random numbers is the use of a true random number generator (TRNG). A TRNG is a hardware device that generates random numbers

[1] Universität Leipzig, Abteilung Technische Informatik, Augustusplatz 10, 04109 Leipzig, Germany {fritzsch, jhoffmann,bogdan}@informatik.uni-leipzig.de

doi:10.18420/in2017_49

based on random physical processes like thermal noise or quantum phenomena. Nevertheless the integration of TRNGs in virtualized server environments proves to be challenging. Evtyushkin and Ponomarev demonstrated that sharing of TRNGs can open covert channels and weaken the isolation between VMs [EP16].

This paper presents an overview of the basic ways to provide high entropy random numbers to virtual machines. Furthermore it presents a new approach that improves the security of virtualized server environments by assigning each virtual machine an own, dedicated TRNG.

2 State of the Art in Random Number Distribution

The basic use case of random number distribution in virtualized server environments consists of multiple VMs, one hypervisor and at most one source of high entropy random numbers in form of a TRNG. Figure 1a shows an example without a TRNG; Figure 1b another including a TRNG. The task is to provide a sufficient quantity of suitable random numbers to each virtual machine. The hypervisors need for random numbers was not explicitly included as it is not different from the need of an additional VM. Practical applications can be modeled by combining multiple instances of the basic use case, for example to include multiple TRNGs or to access one TRNG in multiple ways.

2.1 Non-virtualized Behavior

Without any considerations regarding the generation of random numbers, the operating system inside a VM behaves as it had direct access to the hardware. It gathers random values from user interactions (e.g. via mouse or keyboard) or from the hardware (e.g. disk I/O or specific interrupts), in order to feed a pseudo-random number generator (PRNG) [GPR06; Sp17]. A PRNG is a deterministic algorithm that generates a large amount of seemingly random numbers from a given seed. A PRNG can't increase the entropy. But the generated numbers are hard to tell apart from true random numbers [La90, p. 124]. That way a small amount of random values with high entropy can be used to generate a large amount of sufficiently random numbers.

This approach crucially depends on the high entropy and unpredictability of the random values fed into the PRNG. Both attributes are hard to ensure in virtualized server environments. Since servers in general have fewer direct user interactions, they miss an important source of random values. This problem is exacerbated for virtual machines by the additional abstraction layer provided by the hypervisor. The quantity of the collected random values is deficient. While the quality of the generated random numbers remains high, the generation rate is very low inside a VM [Fe13]. Additionally, reset vulnerabilities can lead to highly predictable random numbers [Ev14].

2.2 Distribution by Hypervisor

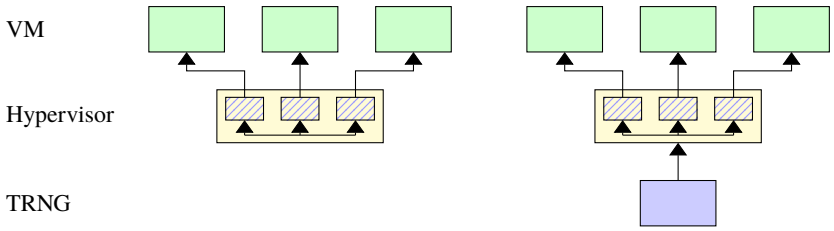

Fig. 1: The hypervisor provides random numbers for the VMs via virtual TRNGs. Either as it is (a) or supported by an actual TRNG (b).

As the hypervisor directly interacts with hardware, it can gather random values faster than a virtual machine [Fe13]. Furthermore, it can't be reset like the VMs. Therefore, the hypervisor is not affected by reset vulnerabilities. Hence, an approach to mitigate the slow gathering of suitable random values is to dispense the random numbers generated by the hypervisor to the VMs (Fig. 1a). To accomplish this, each VM is presented with a virtual random number generator (virtual RNG). A virtual RNG is a virtual device that injects the random numbers generated by the hypervisor into the associated VM. In addition, the hypervisor can limit the throughput of random numbers per virtual RNG in order to prevent covert channels. An example of this distribution model is the virtual RNG from the VirtIO framework [Sh13].

However, typically one hypervisor manages multiple VMs, each with the need for random numbers equivalent to a standalone server. Furthermore the hypervisor still lacks user interactions as source for random values. Hence, current best practice is to connect the hypervisor to a TRNG [Sh15] like shown in Figure 1b.

2.3 Shared TRNG

The need for random numbers is so ubiquitous that x86 processors implement two special instructions: RDRAND and RDSEED. The instructions provide access to a hardware TRNG[2] inside the CPU [Me14]. Due to the fact that these instructions are not restricted, each VM has access to the TRNG. This design constitutes a shared TRNG (Fig. 2a).

This way severe security problems are created. A single malicious program can deplete the supply of random numbers and either facilitate a denial of service or force the use of possibly predictable random numbers. Furthermore, the shared TRNG can be used as a covert channel for unrestricted communication between virtual machines. This subverts the isolation between them [EP16]. Evetyushkin and Ponomarev proposed to trap the execution

[2] RDSEED directly accesses the TRNG while RDRAND accesses a PRNG that is seeded by the TRNG.

of RDSEED and RDRAND into the hypervisor to regulate the access to the TRNG. However, this procedure constitutes distribution by hypervisor with TRNG (cf. 2.2).

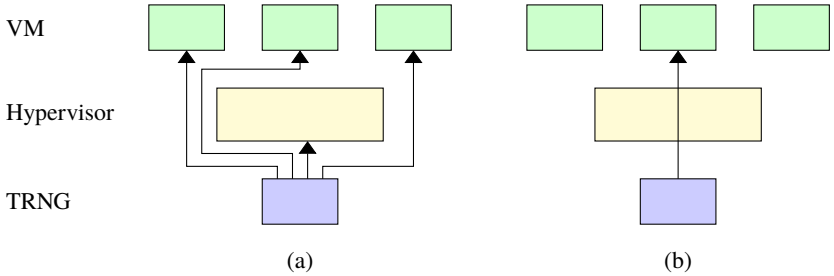

Fig. 2: Depicted are two possible integrations of a TRNG. (a) Shared TRNG: The hypervisor and the VMs obtain random numbers directly from a TRNG. (b) Dedicated TRNG: A TRNG is exclusively and directly accessed by one VM.

2.4 Dedicated TRNG

Modern virtualization hardware allows to dedicate a hardware device to a single virtual machine [AM16]. Data transfers from or to the device are handled by the VM via direct memory access (DMA). Therefore, high transfer rates can be achieved while minimizing the risk of accidentally leaking the data outside of the VM. Yet the application to TRNGs (Fig. 2b) is rare due to the fact that only a single VM is provided with enough high entropy random numbers while all other VMs remain depleted.

3 Proposition: Separated TRNGs

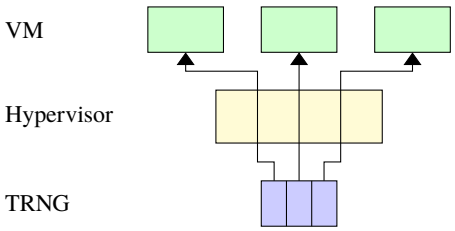

Fig. 3: Every VM has exclusive access to its own separated part of the TRNG.

In order to overcome the disadvantages of the discussed approaches, this work proposes the concept of separated TRNGs. Here a single hardware device is partitioned into multiple separated TRNGs, such that every VM gets its own TRNG (Fig. 3).

Separated TRNGs provide high transfer rates and minimal risk of data leakage as dedicated TRNGs while providing every VM with high entropy random numbers. Furthermore, the

separated TRNGs can't be used as a covert channel between VMs, because the VMs can only access their own TRNG.

The exclusive dedication of hardware to one VM limits the amount of VMs that can get a separated TRNG. To manage situations where more VMs are running than separated TRNGs can be provided, the actual use case has to be analyzed and an appropriate strategy has to be applied. E.g. a fall back system that dedicates one separated TRNG to the hypervisor and distributes the random numbers as described in 2.2.

The challenging task for the implementation is to ensure that the TRNGs are kept separated while using the same hardware device. This is due to avoid correlation of the generated random numbers and the rise of a covert channel. One critical aspect is their significant physical proximity which increases the risk of thermal or electrical coupling. Another critical aspect are shared resources like the power supply or the bus that connects the hardware device to the server. For example the capacity of the shared bus has to be managed carefully. The transfer rate of a separated TRNG has to remain independent from the usage of the others in order to avoid a covert channel.

4 Conclusion

This work presented an overview of the basic ways to provide random numbers to virtual machines. A basic usage model was defined. With this model state of the art concepts to distribute random numbers were evaluated regarding security risks and effectiveness. The distribution by the hypervisor with an additional TRNG was identified as a reasonable solution. It provides high entropy random numbers to VMs and avoids security breaches like the covert channel described by Evetyushkin and Ponomarev [EP16].

In oder to overcome further drawbacks, the concept of separated TRNGs was proposed. It compares favorable to the state of the art solutions. The proposed approach can provide better isolation and lower overhead than the distribution by the hypervisor. The separated TRNGs avoid covert channels that are a typical problem of shared TRNGs. While one dedicated TRNG is needed per VM, separated TRNGs can cover multiple VMs using only one hardware device. Therefore only one physical interface is required, e.g. a PCIe slot.

In further work, the proposed concept will be implemented and the achievable data rate as well as the additional workload of the hypervisor will be measured.

5 Acknowledgment

This work was funded by the German Federal Ministry of Education and Research within the projects Competence Center for Scalable Data Services and Solutions (ScaDS) Dresden/Leipzig (BMBF 01IS14014B) and EXPLOIDS (BMBF 16KIS0522K).

References

[AM16] AMD: I/O Virtualization Technology (IOMMU) Specification, AMD, 2016.

[BDW16] Bittman, T. J.; Dawson, P.; Warrilow, M.: Magic Quadrant for x86 Server Virtualization Infrastructure, Gartner, Inc., 2016, URL: https://www.gartner.com/doc/reprints?ct=160707&id=1-3B9FAM0.

[EP16] Evtyushkin, D.; Ponomarev, D.: Covert channels through random number generator: Mechanisms, capacity estimation and mitigations. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. ACM, pp. 843–857, 2016.

[Ev14] Everspaugh, A.; Zhai, Y.; Jellinek, R.; Ristenpart, T.; Swift, M.: Not-so-random numbers in virtualized Linux and the Whirlwind RNG. In: Security and Privacy (SP), 2014 IEEE Symposium on. IEEE, pp. 559–574, 2014.

[Fe13] Fernandes, D. A.; Soares, L. F.; Freire, M. M.; Inácio, P. R.: Randomness in virtual machines. In: Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing. IEEE Computer Society, pp. 282–286, 2013.

[GPR06] Gutterman, Z.; Pinkas, B.; Reinman, T.: Analysis of the linux random number generator. In: Security and Privacy, 2006 IEEE Symposium on. IEEE, 15–pp, 2006.

[La90] Lagarias, J.: Pseudorandom number generators in cryptography and number theory. Cryptology and computational number theory 42/, pp. 115–143, 1990.

[Me14] Mechalas, J.: Intel Digital Random Number Generator (DRNG) Software Implementation Guide, 2.0, 2014.

[Po16] Ponemon Institute LLC: 2016 Cost of Data Breach Study: Global Analysis, Ponemon Institute LLC, 2016, URL: https://www.ibm.com/security/data-breach.

[Sh13] Shah, A.: About Random Numbers and Virtual Machines, 2013, URL: http://log.amitshah.net/2013/01/about-random-numbers-and-virtual-machines/, visited on: 04/23/2017.

[Sh15] Shah, A.: Red Hat Enterprise Linux Virtual Machines: Access to Random Numbers Made Easy, 2015, URL: http://rhelblog.redhat.com/2015/03/09/red-hat-enterprise-linux-virtual-machines-access-to-random-numbers-made-easy/, visited on: 04/23/2017.

[Sp17] Spelvin, G.; Mackall, M.; Georget, L.; Mavrogiannopoulos, N.: random(7), 4.10, 2017, URL: http://man7.org/linux/man-pages/man7/random.7.html.