Randomness Classes in Bugs Framework (BF): True-Random Number Bugs (TRN) and Pseudo-Random Number Bugs (PRN)

Irena Bojanova
SSD, ITL
NIST
Gaithersburg, MD, USA
irena.bojanova@nist.gov

Yaacov Yesha
SSD, ITL; CSEE
NIST; UMBC
Gaithersburg; Baltimore MD, USA
yaacov.yesha@nist.gov

Paul E. Black
SSD, ITL
NIST
Gaithersburg, MD, USA
paul.black@nist.gov

***Abstract*—Random number generators may have weaknesses (bugs) and the applications using them may become vulnerable to attacks. Formalization of randomness bugs would help researchers and practitioners identify them and avoid security failures. The Bugs Framework (BF) comprises rigorous definitions and (static) attributes of bug classes, along with their related dynamic properties, such as proximate and secondary causes, consequences and sites. This paper presents two new BF classes: True-Random Number Bugs (TRN) and Pseudo-Random Number Bugs (PRN). We analyze particular vulnerabilities and use these classes to provide clear BF descriptions. Finally, we discuss the lessons learned towards creating new BF classes.**

Keywords—randomness, random numbers, random number generators, pseudo-random number generators, software weaknesses, bug taxonomy, attacks.

I. Introduction

Randomness has application in many fields, including cryptography, simulation, statistics, politics, science, and gaming. Any specific use has its own requirements for randomness – e.g., random bit generation for cryptography or security purposes has stronger requirements than generation for other purposes. For cryptography or security purposes, the National Institute of Standards and Technology (NIST) recommends use of cryptographically secure Pseudo-Random Bit Generators (PRBGs). They are subject to the requirements in 🔒NIST SP 800-90A [8], NIST SP 800-90B [9] and NIST SP 800-90C [10]. Satisfying the requirements for a particular use can be surprisingly difficult [1] *.

Weaknesses (bugs) in random number generators (RNGs) may lead to wrong results from the algorithms that use the generated numbers or allow attackers to recover secret values, such as passwords and cryptographic keys. Formalization of randomness bugs would help researchers and practitioners identify them and avoid security failures. For that we have developed a general descriptive model of randomness and two randomness classes as part of the Bugs Framework (BF) [2, 3].

In this paper, we discuss randomness bugs, present the BF randomness bugs model, and detail our newly-developed randomness classes: True-Random Number Bugs (TRN) and Pseudo-Random Number Bugs (PRN). The details include definitions and taxonomy of these classes, examples of vulnerabilities from the Common Vulnerabilities and Exposures (CVE) [4], and corresponding Common Weakness Enumeration (CWE) [5] or Software Fault Patterns (SFP) [6]. In the concluding section we discuss the lessons learned.

II. The Bugs Framework (BF)

The Bugs Framework (BF) provides accurate, precise, and unambiguous definitions of software weaknesses (bugs) and a language-independent taxonomy that allows clear descriptions of software vulnerabilities [2, 3]. It organizes bugs into distinct classes. The taxonomy of each BF class comprises: level, causes, attributes, consequences, and sites of bugs. Level (high or low) identifies the fault as language-related or semantic. Causes bring about the fault. At least one attribute (denoted as underlined) identifies the software fault, while the rest may be simply descriptive. It is useful to catalog possible consequences of faults. Sites are locations in code (identifiable mainly for low level classes) where the bug might occur under circumstances indicated by the causes.

Previously developed BF classes are: Buffer Overflow (BOF), Injection (INJ), Control of Interaction Frequency Bugs (CIF) [2], Encryption Bugs (ENC), Verification Bugs (VRF), Key Management Bugs (KMN) [3], and Faulty Result (FRS). Here we only give their definitions. Details and application examples of use are available at [7] .

BOF: *The software accesses through an array a memory location that is outside the boundaries of that array.*

INJ: *Due to input with language-specific special elements, the software assembles a command string that is parsed into an invalid construct.*

CIF: *The software does not properly limit the number of repeating interactions per specified unit.*

ENC: *The software does not properly transform sensitive data (plaintext) into unintelligible form (ciphertext) using a cryptographic algorithm and key(s).*

VRF: *The software does not properly sign data, check and prove source, or assure data is not altered.*

KMN: *The software does not properly generate, store, distribute, use, or destroy cryptographic keys and other keying material.*

FRS: *The software produces a faulty result due to conversions between primitive types, range violations, or domain violations.*

* The 🔒 icon is used through the paper where we note the NIST SP 800-90 recommendations for construction of RBGs.

III. Randomness Bugs

A. Randomness Generation

We separate randomness generation in two distinct processes: true-random number generation and pseudo-random number generation. The former is nondeterministic true-randomness generation (full entropy), while the latter is deterministic pseudo-randomness generation.

True-random number generation uses entropy sources, while pseudo-random number generation uses true-random numbers as seeds. It is possible for a PRBG to use non-random seeds (e.g., for generating random numbers for simulation or game algorithms). PRBGs are used to extend the true-random seeds, produced from a True-Random Bit Generator (TRBG) output – if the seed has length `n`, the output of the PRBG can have length `m`, where `m>n`. However, a PRBG cannot increase the entropy of its seed.

Examples of randomness related attacks are direct RSA common factor attack [11, 12] cryptanalytic attack, input based attack, state compromise attack. [13]

B. The BF Randomness Model

Fig. 1 presents our BF randomness bugs model, showing in which software components of TRNG and Pseudo-Random Number Generator (PRNG) bugs can occur. It is a descriptive and not as a prescriptive model. It should not be used as a model for construction of Random Bit Generators (RBGs). (🔒 NIST SP 800-90C specifies construction of RBGs using the mechanisms and entropy sources described in SP 800-90A and SP 800-90B, respectively.) TRN is the name of our BF class of True-Random Number Bugs. PRN is the name of our BF class of Pseudo-Random Number Bugs. The BF randomness model helps identify where in the corresponding bugs could occur. TRN covers bugs related to entropy sources, TRBG, and TRNG. PRN covers bugs related to entropy pools, PRBG, and PRNG. Although, output from the former process may be used as input to the latter (see the red arrow in Fig. 1), they are distinct from the point of view that bugs related to each have different causes, attributes, and consequences. The random bits are optionally converted in a pseudo-random number based on the range that applications provide as an argument.

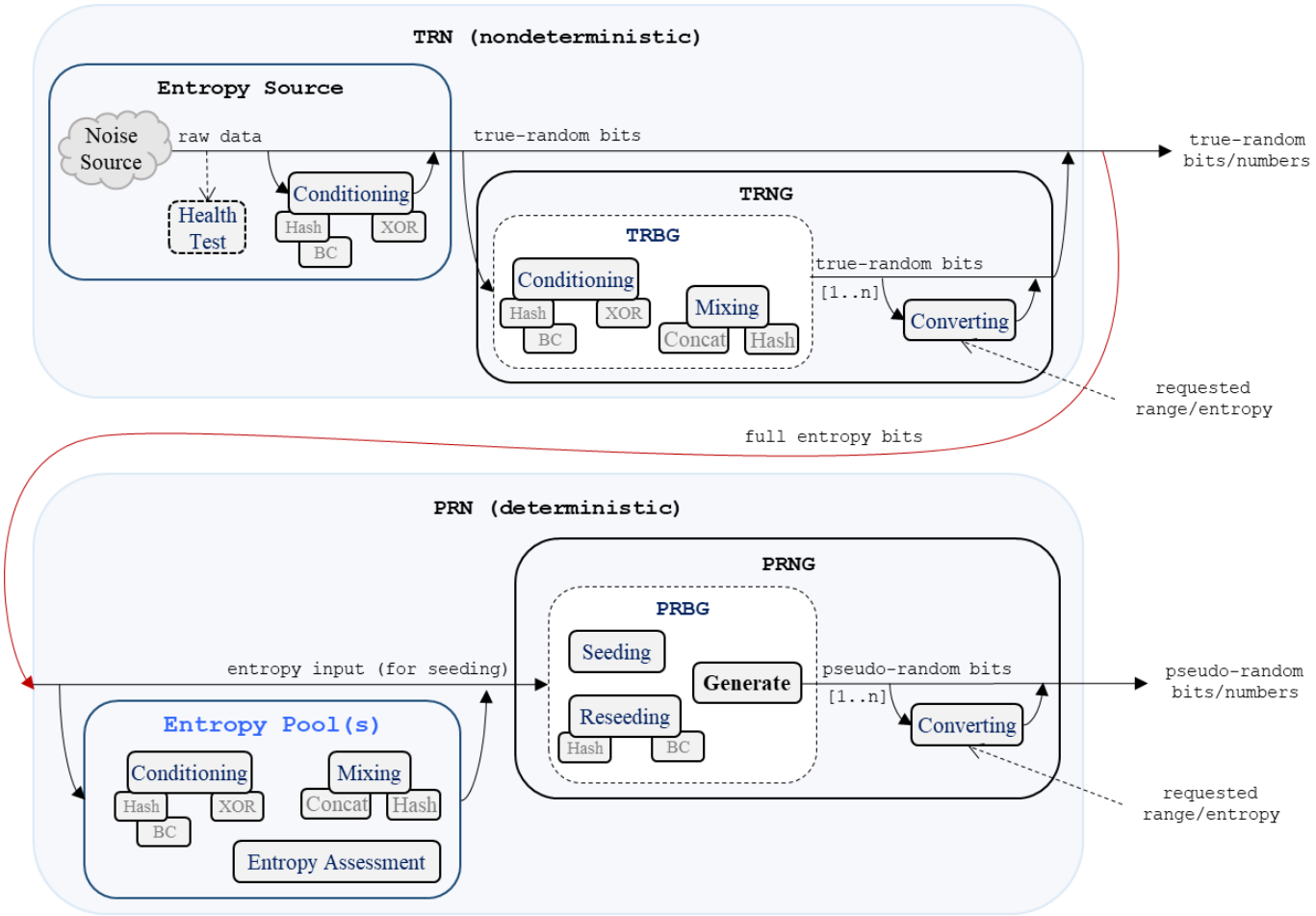

Fig. 1. The BF Model of Randomness Bugs. It is a descriptive and not as a prescriptive model. It should not be used as a model for construction of RBGs.
🔒 NIST SP 800-90A/B/C give specific requirements and architectures for appovoed RBGs.

(TRN – True-Random Number Bugs
PRN – Pseudo-Random Number Bugs
TRBG: True-Random Bit Generator
TRNG: – True-Random Number Generator
PRBG – Pseudo-Random Bit Generator
PRNG – Pseudo-Random Number Generator
BC – Block Cipher)

If live entropy source is used, the PRBG is said to support prediction resistance. A PRBG without prediction resistance can still be used where an on-demand entropy source and immediate resetting are not required. [8]

PRNGs are algorithmic and can have bugs. Most PRNGs are not cryptographically secure.

IV. True-Random Number Bugs Class - TRN

A. Definition

We define True-Random Number Bugs (TRN) as:

The software generated output does not satisfy all use-specific true-randomness requirements.

Note that the output sequence is of random bits, where values are obtained from one or more sources of entropy.

TRN is related to: PRN, ENC, VRF, KMN, IEX (Information Exposure Bugs).

B. Taxonomy

Fig. 2 depicts TRN causes, attributes and consequences.

The attributes of TRN are:

Function – Health Test, Conditioning, Mixing, Output, Converting.

Algorithm – Hash Function, Block Cipher, XOR, etc.

Used For – Seeding, Generation.

This is what the output sequence is used for. It could be used as a seed for a PRNG or for generation of passwords or cryptographic keying material (keys, nonces) [15].

Randomness Requirement – Sufficient Entropy, Sufficient Space Size, Non-Inferable. This is the failed requirement.

The importance of sufficient entropy is discussed in (CWE-333 [5]). The notion of entropy used in this paper is min-entropy, as the negative logarithm of the probability of the most likely outcome. Let `X` be a random variable such that the set of possible values that it can have is finite. Let `P` be the set of probabilities of `X` having those values. The min-entropy of `X` is defined as $-\log_2 \max(P)$, where `max()` finds the largest value in a set. The min-entropy is a measure of how difficult it is to guess the most likely entropy source output. [9, 16]

Space size is the number of elements of the space of possible outputs (CWE-334 [5]). If the number of different outputs is not sufficiently large, there is a vulnerability to a brute force attack. Non-inferable means one cannot recover from known (guessed) information anything about the TRBG output (CVE-2008-0141 [4]). TRBGs used for cryptography/security must satisfy the Non-Inferable randomness requirement.

In the graph of causes: Incorrect Entropy Assessment may result in TRN output having insufficient entropy. For example, a certain number of bits with full entropy may be needed, and because of Incorrect Entropy Assessment, the output may have insufficient entropy.

In the graph of consequences: Inadequate Input to PRNG could be repeating, weak, insufficiently random, predictable, small space seed or other input. "A program may crash or block if it runs out of random numbers" (CWE-333 [5]. Denial of Service (DoS) can be a direct consequence (CWE-333 [5]).

KMN could be a consequence as in finding private keys from public keys using a common factor attack [11, 12]. VRF could be a consequence of using a predictable random number with a signature algorithm, such as DSA (Digital Signature Algorithm). The information leaked (IEX) could be of the exact value of a generated random number (CWE-342 [5]) or a small range of values (CWE-343 [5]) from which the generated random number is easy to figure out.

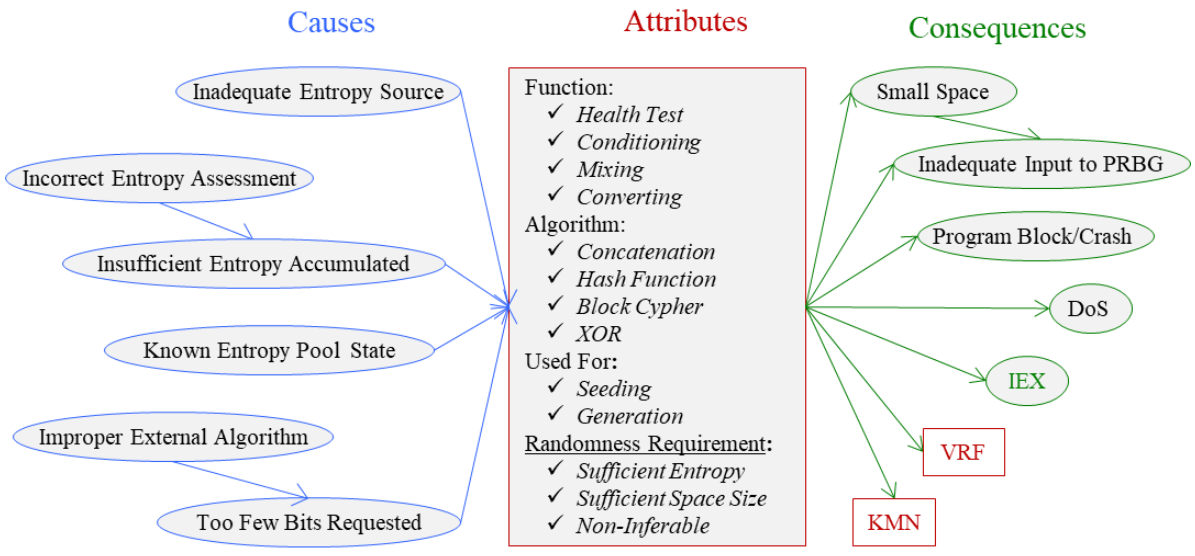

Fig. 2. The True-Random Number Bugs (TRN) represented as causes, attributes and consequences.

C. An Example

CVE-2008-0141

This vulnerability is listed in CVE-2008-0141 [4] and discussed in [17].

The BF TRN taxonomy for this vulnerability is:

Cause: Inadequate Entropy Sources (current date/time and user name)

Attributes:

Function: Mixing

Algorithm: Concatenation

Used for: Generation (of password)

Randomness Requirement: Non-Inferable (time known from password reset time, name - from user register)

Consequences: IEX (of password), leading to ATN (Authentication Fault)

Our BF description is:

Inadequate entropy sources (date/time and user name) mixing using concatenation allow generation of passwords that do not satisfy the non-inferable randomness requirement (time known from password reset time, name - from user register), which may be exploited for IEX (of password), leading to ATN.

Analysis (based on CVE-2008-0141 [4] and [17]): [17] includes "… The new password is simply the date (minute+seconds) and the var `$user` taken through `register_globals` (we can let it [be] empty) …". An attack exploiting that is described there.

D. Related CWEs and SFP

The related SFP cluster is SFP Primary Cluster: Predictability [6], which is CWE-905 with members CWE 330 to 344 [5].

Among them, the TRN CWEs are: 330, 331, 332, 333, 334, 337, 339, 340, 341, 342, 343 [5].

V. Pseudo-Random Number Bugs Class -- PRN

A. Definition

We define Pseudo-Random Number Bugs (PRN) as:

The software generated output does not satisfy all use-specific pseudo-randomness requirements.

Note that the output sequence is of random bits or numbers from a PRNG.

PRN is related to: TRN, ENC, VRF, KMN, IEX.

B. Taxonomy

Fig. 3 depicts PRN causes, attributes and consequences.

The attributes of PRN are:

Function – Conditioning, Mixing, Entropy Assessment, Seeding, Reseeding, Generate, Converting.

Algorithm – Concatenation, Hash Function, Block Cipher, XOR. Concatenation is the usual mixing of output from IID sources.

Used For – ASLR (Address Space Layout Randomization), Generation, Initialization, Input to Algorithm. This is what the output sequence is used for. It could be used for ASLR, generation of passwords or cryptographic keying material (keys, nonces) [15], initialization of cryptographic primitives [18] (e.g., an initialization vector for cipher block chaining mode of encryption; or a salt for hashing), or input to simulation, statistics, mathematics (e.g., Monte Carlo integration), or general algorithms.

Pseudo-Randomness Requirement – Unpredictability/ Indistinguishability, Prediction/Backtracking Resistance, Sufficient Space Size, Use Specific Statistical Tests. This is the failed requirement.

The pseudo-random output sequence should be statistically independent and unbiased [10]. It should pass the use-specific statistical tests for randomness. Unpredictability means that it should not be possible to predict next generated output from previous output [15]. Prediction Resistance means it is not possible to predict future output bits even if past or present state is known. Prediction resistance is not possible without a live entropy source. Backtracking Resistance means it is not possible to recover (backtrack) past output bits based on knowledge of the state at a given point in time [8, 10].

For Space Size see section IV.B. above. Indistinguishability for a PRNG means that its output is computationally indistinguishable (i.e., by any probabilistic polynomial time algorithm) from a truly random sequence [15].

PRNGs used for cryptography/security must satisfy the Unpredictability/Indistinguishability, Backtracking Resistance, and Sufficient Space Size requirements. Prediction Resistance however is not always required – e.g. for PIV cards.

C. Examples

1) CVE-2001-1141

This vulnerability is listed in (CVE-2001-1141 [4]) and discussed in [19, 20].

The BF PRN taxonomy for this vulnerability is:

Cause: Improper PRNG Algorithm (C `md_rand` – the secret PRNG state is updated with portion, as small as one byte, of the PRNG's previous output, which is not secret)

Attributes:

Function: Mixing (back into entropy pool)

Algorithm: Hash Function (SHA-1 – used for PRNG output and to update its internal secret state)

Used For: Generation (of cryptographic keying material – nonces, cryptographic keys)

Pseudo-Randomness Requirements: Sufficient Space Size and Unpredictability (can be predicted from previous value through brute force)

Consequences: KMN>Generate with IEX of future keying

Our BF description is:

Use of improper PRNG algorithm (C `md_rand` uses SHA-1 for mixing back in the entropy pool portion, as small as one byte, of previous output to update PRNG's state), allows generation of cryptographic keying material (nonces and cryptographic keys) that does not satisfy the sufficient space size and unpredictability (can be predicted from previous values through brute force) pseudo-randomness requirements, which leads to KMN>Generate and IEX of future keying material.

Analysis (based on CVE-2001-1141 [4] and [19 – 22]):

A PRNG used for cryptography does not satisfy the requirement of unpredictability from previous values, because the internal state can be determined from number of output requests. Possible consequences include: IEX of future PRNG output (CVE-2001-1141 [4]) (which is KMN>Generation failure) and weak encryption, confidentiality compromise [21] (which is ENC>Confidentiality failure [3]).

The entropy accumulation implementation (entropy pool and associated mixing function) allows reconstruction of the PRNG internal state [20]. The mixing hash function for `md` (in the C `md_rand`) gets half of the previous value of `md` and bytes from the PRNG internal state. Wrongly, the half used is the one with the PRNG's previous output (failed implementation relative to specification). Also, the number of used state bytes depends on the number of bytes requested as output, which could be as small as one byte. This enables a brute-force attack. The PRNG state could be reconstructed from the output of one large PRNG request (large enough to gain knowledge on `md`) followed by consecutive 1-byte PRNG requests [21, 22].

2) CVE-2008-4107

This vulnerability is listed in (CVE-2008-4107 [4]) and discussed in [23-26].

The BF PRN taxonomy for this vulnerability is

Cause: Improper PRNG Algorithms (not cryptographically strong PHP 5 `rand` and `mt_rand`)

Attributes:

Function: Generate (pseudo-random numbers)

Algorithms: e.g., LCG or LFSR, Mersenne Twister

Used For: Generation (of passwords)

Pseudo-Randomness Requirements: Unpredictability/ Indistinguishability and Prediction Resistance

Consequence: IEX (of password), leading to ATN

Our BF description is:

Improper PRNG algorithms (not cryptographically strong PHP 5 `rand` and `mt_rand`, based on algorithms such as LCG or LFSR, and Mersenne Twister) used to generate pseudo-random numbers, allow generation of passwords that do not satisfy the unpredictability/ indistinguishability and prediction resistance pseudo-randomness requirements and may be exploited for IEX of password, leading to ATN.

Analysis: PHP's `rand()` [23] usually uses LCG or LFSR, which is weak. PHP 5 `mt_rand()` [24] uses Mersenne Twister, which is weak as well. It enables finding the internal state and all future values from 624 values. [25]. This vulnerability can be used for password guessing (CVE-2008-4107 [4]), [26]).

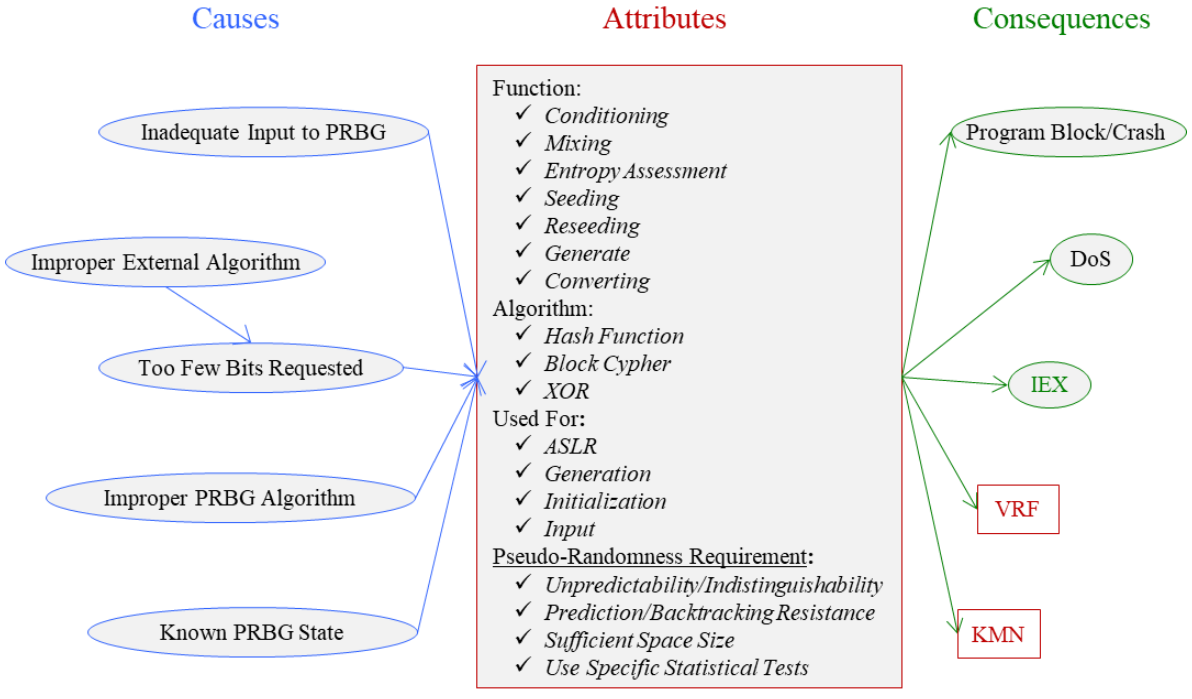

Fig. 3. The Pseudo-Random Number Bugs (PRN) represented as causes, attributes and consequences (ASLR – Address Space Layout Randomization).

CVE-2009-3238

This vulnerability is listed in (CVE-2009-3238 [4]) and discussed in [27-33].

The BF TRN and PRN taxonomy of this vulnerability is:

An TRN leads to a PRN.

TRN

Cause: Improper RNG Algorithm (same area is used for the `hash` array, allowing to repeatedly start from the same seed)

Attributes:

Functions: Mixing (of pid and jiffies), Conditioning

Algorithms: Concatenation, Hash Function (MD4)

Used For: Seeding

Randomness Requirements: Sufficient Entropy

Consequence: Inadequate Input to PRNG (repeating seed)

PRN

Cause: Inadequate Input to PRNG (repeating seed)

Attributes:

Function: Generate (pseudo-random numbers)

Used For: ASLR

Pseudo-Randomness Requirement: Unpredictability

Consequences: IEX of addresses

Our BF description is:

An TRN leads to a PRN.

TRN: *Improper RNG algorithm (same area is used for the `hash` array, allowing to repeatedly start from the same seed) for mixing (concatenation of pid and jiffies) followed by conditioning (using MD4 hash function), allows seeding that does not satisfy the sufficient entropy randomness requirement, leading to Inadequate Input to PRNG (repeating seed).*

PRN: *Inadequate input to PRNG (repeating seed), used to generate pseudo-random numbers, allows use of ASLR that does not satisfy the unpredictability pseudo-randomness requirement and may be exploited for IEX of addresses.*

Analysis (based on CVE-2009-3238 [4] and in [27-33]):

There is always a specific area for the `hash` array, allowing an attacker to repeatedly start from the same seed [32]. The result is predictability of address space randomization over a short time period. This violates the sufficient entropy requirement.

"Address space randomization hinders some types of security attacks by making it more difficult for an attacker to predict target addresses" [27]. CVE-2009-3238 [4] includes as a reference [33] that describes a fix that increases the randomness of ASLR: "Following security issues were fixed: ... CVE-2009-3238: The randomness of the ASLR methods used in the kernel was increased."

The TRN consequence is Inadequate Input to PRNG (repeating seed) [32]. The PRN consequence is IEX of addresses [27-30].

*3) **CVE-2017-15361*** (ROCA – Return Of the Coppersmith Attack)

This vulnerability is listed in CVE-2017-15361 [4] and discussed in [34].

The BF KMN [3] with inner PRN (see RND>Inadequate in [3]) taxonomy for this vulnerability is:

A KMN with inner PRN**.**

KMN:

Cause: Improper Algorithm Step (for generation of primes for RSA keys) leads to inner PRN

Attributes:

Cryptographic Data: Keying Material (keys)

Algorithm: RSA (key generation from two primes)

Operation: Generate (pair of public and private keys)

Consequences: Weak Public Key, which leads to IEX of Private Key.

Inner PRN:

Cause: Improper External Algorithm (generation of primes for RSA keys from random numbers and a constant related to keys size) leads to Too Few Bits Requested

Attributes:

Functions: Converting, Seeding (low entropy requested)

Used for: Generation (of secret prime numbers)

Randomness Requirement: Sufficient Space Size (e.g., one random number is only `37` bits for 512-bit RSA keys)

Consequence: IEX of generated primes (which format allows keys fingerprinting, factorization with Coppersmith algorithm, and finding random numbers and primes).

Our BF description is:

*A KMN with inner PRN**.***

KMN: *Improper algorithm step (for generation of primes for RSA keys) leads to inner PRN>Inadequate and allows generation of keying material (pair of public and private keys) with weak public key, leading to IEX of the private key.*

Inner PRN: *Improper external algorithm (generation of primes for RSA keys, from random numbers and a constant related to keys size) leads to too few bits requested at converting and at seeding, and allows generation of random numbers not satisfying the sufficient space size requirement*, which may be exploited for IEX of primes through fingerprinting of keys and factorization with Coppersmith algorithm.

Analysis (based on CVE-2017-15361 [4] and [34]): The generated RSA primes have the form: `p = k * M + (65537^a mod M)`. Where, `k` and `a` are random numbers; `M`

is a primorial (the product of the first `n` successive primes), related to the key size, which is a multiple of 32 bits. For keys with size in the `[512; 960]` interval, `n = 39` is used (i.e., `M = 2 * 3 * … * 167`); `n = 71; 126; 225` is used for key sizes within intervals `[992; 1952]`, `[1984; 3936]`, `[3968; 4096]`, respectively.

For keys with the same size, the generated RSA primes differ only in the values of `k` and `a`. The size of `M` is large – almost comparable to the size of the prime `p` (e.g., `M` has `219` bits for the 256-bit prime `p` used for 512-bit RSA keys). Since `M` is large, the sizes of `k` and `a` are small (e.g., `k` has `256 - 219 = 37` bits and `a` has `62` bits for 512-bit RSA). Hence, the resulting RSA primes suffer from a significant loss of entropy (e.g., a prime used in 512-bit RSA has only `99` bits of entropy), and the pool from which primes are randomly generated is reduced (e.g., from `2^256` to `2^99` for 512-bit RSA). [34]

The specific format of the primes allows fingerprinting of the keys followed by factorization. The size of `M` is large, but the logarithm $\log_{65537} N \bmod M$ can be computed, as `M` has only small factors. Factoring `N` using the Coppersmith's algorithm reveals `a` and `k` and eventually the prime `p`. [34]

D. Related CWEs and SFP

The related SFP cluster is SFP Primary Cluster: Predictability [6], which is CWE-905 with members CWE 330 to 344 [5].

Among them, the PRN CWEs are: 330, 331, 332, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343 [5].

VI. Conclusion

A. Summary

In this paper, we presented two new BF Classes: True-Random Number Bugs (TRN) and Pseudo-Random Number Bugs (PRN). They join other rigorously-defined BF classes, such as Encryption/Decryption Bugs (ENC), Verification Bugs (VRF), and Key Management Bugs (KMN). We presented the (static) attributes of the classes, along with the classes' causes and consequences. We analyzed particular vulnerabilities related to those classes and provided clear descriptions. We showed that the BF-structured descriptions of randomness bugs are quite concise, while still far clearer than unstructured explanations that we have found.

B. Lessons Learned and Future Work

At first, we tried to define a single Randomness class. However, that meant mixing the causes, attributes, and consequences related to true-random number generation and pseudo-random number generation. While exploring related vulnerabilities, we also realized that some of the consequences from the former may be causes for the latter. For example, if the consequence of a faulty seed generation is weak seed, this becomes the cause for a PRNG fault.

We considered keeping it all in one class by grouping the causes, attributes' values, and consequences related to true- and pseudo-random number generation. However, the model of randomness generation that we developed showed that these are two clearly separated processes and there should be two separate PRN and PRN randomness classes.

Work on explaining more randomness bugs using TRN and PRN will help us determine if our BF taxonomy needs refinement.

Our goal is for BF to become software developers' and testers' "Best Friend."

VII. References

[1] D. Eastlake, J. Schiller, S. Crocker, "Randomness Requirements for Security", 2005, https://tools.ietf.org/html/rfc4086/.

[2] I. Bojanova, P. E. Black, Y. Yesha, and Y. Wu, "The Bugs Framework (BF): A Structured approach to express bugs", Proceedings of IEEE International Conference on Software Quality, Reliability and Security (QRS), 2016, , pp. 175-182.

[3] I. Bojanova, P. E. Black, and Y. Yesha, "Cryptography classes in Bugs Framework (BF): Encryption Bugs (ENC), Verification Bugs (VRF), and Key Management Bugs (KMN)", 28th Annual IEEE Software Technology Conference (STC), 2017.

[4] The MITRE Corporation, Common Vulnerabilities and Exposures (CVE), https://www.cve.mitre.org.

[5] The MITRE Corporation, Common Weakness Enumeration (CWE), https://cwe.mitre.org.

[6] N. Mansourov, "DoD Software Fault Patterns," KDM Analytics, Inc., 2011. http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADB381215/.

[7] The Bugs Framework, https://samate.nist.gov/BF.

[8] E. Barker, J. Kelsey, "NIST Special Publication 800-90A, Revision 1, Recommendation for Random Number Generation Using Deterministic Random Bit Generators", NIST, June 20q5, http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf.

[9] Meltem Sönmez Turan, Elaine Barker, John Kelsey, Kerry A. McKay, Mary L. Baish, Mike Boyle, NIST Special Publication 800-90B, Recommendation for the Entropy Sources Used for Random Bit Generation, NIST, January, 2018, http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90B.pdf,.

[10] E. Barker, J. Kelsey, (Second Draft) NIST Special Publication 800-90C "Recommendation for Random Bit Generator (RBG) Constructions", https://csrc.nist.gov/csrc/media/publications/sp/800-90c/draft/documents/sp800_90c_second_draft.pdf.

[11] N. Heninger, "New research: There's no need to panic over factorable keys–just mind your Ps and Qs", Feb. 15, 2012, https://freedom-to-tinker.com/2012/02/15/new-research-theres-no-need-panic-over-factorable-keys-just-mind-your-ps-and-qs/.

[12] A. K. Lenstra, J. P. Hughes, M. Augier, J. W. Bos, T. Kleinjung, and C. Wachter, "Ron was wrong, Whit is right", Infscience, https://infoscience.epfl.ch/record/174943.

[13] J. Kelsey, B. Schneier, D. Wagner, C. Hall. "Cryptanalytic attacks on pseudorandom number generators". Fast Software Encryption, Fifth International Workshop Proceedings. Springer-Verlag. pp. 168–188, 1998.

[14] J. Kelsey, B. Schneier, N. Ferguson, "Yarrow-160: Notes on the Design and Analysis of the Yarrow Cryptographic Pseudorandom Number Generator", https://www.schneier.com/academic/paperfiles/paper-yarrow.pdf.

[15] Wikipedia, "Cryptographically secure pseudorandom number generator",https://en.wikipedia.org/wiki/Cryptographically_secure_pseudorandom_number_generator/ .

[16] John Kelsey, Kerry A. McKay, and Meltem Sönmez Turan, "Predictive Models for Min-Entropy Estimation", International Workshop on Cryptographic Hardware and Embedded Systems, CHES 2015, pp 373-392, http://ws680.nist.gov/publication/get_pdf.cfm?pub_id=918415.

[17] WebPortal CMS 0.6-beta - Remote Password Change, Exploit Database, https://www.exploit-db.com/exploits/4835/.

[18] Wikipedia, "Initialization vector", .https://en.wikipedia.org/wiki/Initialization_vector.

[19] VULDB, "OPENSSL/SSLEAY Pseudo-random Number Generator Weak Encryption", https://vuldb.com/?id.16976.

[20] Security Focus, "OpenSSL PRNG Internal State Disclosure Vulnerability", https://www.securityfocus.com/bid/3004.

[21] OpenSSL, "Weakness of the OpenSSL PRNG in versions up to OpenSSL 0.9.6a", "https://www.openssl.org/news/secadv/prng.txt

[22] Security Traker, "OpenSSL Uses Potentially Predictable Pseudo-Random Number Generator", https://securitytracker.com/id/1001961

[23] PHP Manual, rand, http://php.net/manual/en/function.rand.php.

[24] PHP Manual, mt_rand, .http://php.net/manual/en/function.mt-rand.php.

[25] StackExchange, "Information Security, How insecure are PHP's rand functions?", https://security.stackexchange.com/questions/18033/how-insecure-are-phps-rand-functions .

[26] Security Focus, "WordPress Random Password Generation Insufficient Entropy Weakness", http://www.securityfocus.com/bid/31115 .

[27] Wikipedia, "Address space layout randomization", https://en.wikipedia.org/wiki/Address_space_layout_randomization.

[28] GitHub, Inc. [US], "torvalds/linux", https://github.com/torvalds/linux/blob/master/drivers/char/random.c.

[29] J. Salwan, "ASLR implementation in Linux Kernel 3.7", Jan. 19, 2013, http://shell-storm.org/blog/ASLR-implementation-in-Linux-Kernel-3.7.

[30] xorl.wordpress, "Linux kernel ASLR Implementation" with 4 comments, https://xorl.wordpress.com/2011/01/16/linux-kernel-aslr-implementation.

[31] git.kernel, index:: kernel/git/torvalds/linux.git", https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=8a0a9bd4db63bc45e3017bedeafbd88d0eb84d02.

[32] xorl.wordpress, "CVE-2009-3238: Linux kernel get_random_int() Predictable Numbers", https://xorl.wordpress.com/2009/10/26/cve-2009-3238-linux-kernel-get_random_int-predictable-numbers.

[33] openSUSE Mailinglist Archive: opensuse-security-announce (12 mails) https://lists.opensuse.org/opensuse-security-announce/2009-11/msg00005.html.

[34] M. Nemec, M. Sys, P. Svenda, D. Klinec, V Matyas, "The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli", ACM CCS 2017, https://crocs.fi.muni.cz/_media/public/papers/nemec_roca_ccs17_preprint.pdf.