Cryptography Classes in Bugs Framework (BF): Encryption Bugs (ENC), Verification Bugs (VRF), and Key Management Bugs (KMN)

Irena Bojanova; Paul E. Black
NIST, Gaithersburg, USA
irena.bojanova@nist.gov, paul.black@nist.gov

Yaacov Yesha
NIST, Gaithersburg, USA; UMBC, Baltimore, USA
yaacov.yesha@nist.gov

***Abstract*—Accurate, precise, and unambiguous definitions of software weaknesses (bugs) and clear descriptions of software vulnerabilities are vital for building the foundations of cybersecurity. The Bugs Framework (BF) comprises rigorous definitions and (static) attributes of bug classes, along with their related dynamic properties, such as proximate, secondary and tertiary causes, consequences, and sites. This paper presents an overview of previously developed BF classes and the new cryptography related classes: Encryption Bugs (ENC), Verification Bugs (VRF), and Key Management Bugs (KMN). We analyze corresponding vulnerabilities and provide their clear descriptions by applying the BF taxonomy. We also discuss the lessons learned and share our plans for expanding BF.**

Keywords—software weaknesses; bug taxonomy; attacks.

I. Introduction

Advances in scientific foundations of cybersecurity rely on the availability of accurate, precise, and unambiguous definitions of software weaknesses (bugs) and clear descriptions of software vulnerabilities. The myriad unprecedented attacks and security exposures, including on Internet of Things (IoT) applications, calls for serious efforts towards such formalization.

To provide a foundation, we are developing the Bugs Framework (BF) [1], which organizes bugs into distinct classes, such as buffer overflow (BOF), injection (INJ), faulty operation (FOP), and control of interaction frequency bugs (CIF). Each BF class has an accurate and precise definition and comprises: level (added after [1]), causes, attributes, consequences, and sites of bugs. Closely related classes may be grouped in clusters. *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. The goal of BF is to help researchers and practitioners more accurately and quickly diagnose, describe, and measure security vulnerabilities.

In this paper, we summarize the BF classes we previously developed, then detail our newly-developed cryptography-related classes: Encryption Bugs (ENC), Verification Bugs (VRF), and Key Management Bugs (KMN). The details include definitions and taxonomy of these classes, examples of vulnerabilities from the Common Vulnerabilities and Exposures (CVE) [2], and corresponding Common Weakness Enumerations (CWE) [3] or Software Fault Patterns (SFP) [4]. The final section summarizes our work, discusses lessons we learned, and presents our plans for expanding BF further.

II. Previously Developed BF Classes

Our first developed BF classes were: Buffer Overflow (BOF), Injection (INJ), and Control of Interaction Frequency Bugs (CIF) [1]. Here we only give their definitions and attributes. Details and examples of use are available at https://samate.nist.gov/BF/.

BOF: *The software accesses through an array a memory location that is outside the boundaries of that array.* Attributes: Access, Boundary, Location, Magnitude, Data Size, Reach.

INJ: *Due to input with language-specific special elements, the software assembles a command string that is parsed into an invalid construct.* Attributes: Invalid Construct, Language, Special Element, Entry Point.

CIF: *The software does not properly limit the number of repeating interactions per specified unit.* Attributes: Interaction, Number, Unit, Actor.

III. Cryptographic Store or Transfer Bugs

A. *Cryptography*

Cryptography is a broad, complex, and subtle area. It incorporates many clearly separate processes, such as encryption/decryption, verification of data or source, and key management. There are bugs if the software does not properly transform data into unintelligible form, verify authenticity or correctness, manage keys, or perform other related operations. Some transformations require keys, for example encryption and decryption, while others do not, for example secret sharing. Authenticity covers integrity of data, identity of data source, origin for non-repudiation, and content of secret sharing. Correctness is verified for uses such as zero-knowledge proofs. Cryptographic processes use particular algorithms to achieve particular security services [5].

Examples of attacks are spoofing messages, brute force attack, replaying instructions, timing attack, chosen plaintext attack, chosen ciphertext attack, and exploiting use of weak or insecure keys.

In this paper, we use cryptographic store or transfer to illustrate our ENC, VRF, and KMN classes of bugs. Note that these classes may appear in many other situations such as self-sovereign identities [6], block ciphers, and threshold cryptography. We focus on transfer (or store) because it is well known and it is what most people think of when "cryptography" is mentioned. We define bugs in cryptographic store or transfer as: The software does not properly encrypt/decrypt, verify, or manage keys for data to be securely stored or transferred.

B. *Our Model*

A modern, secure, flexible cryptographic storage or transfer protocol likely involves subtle interaction between encryption, verification, and key management processes. It may involve multiple stages of agreeing on encryption algorithms, establishing public and private keys, creating session keys, and digitally signing texts for verification. Thus, encryption may use key management, which

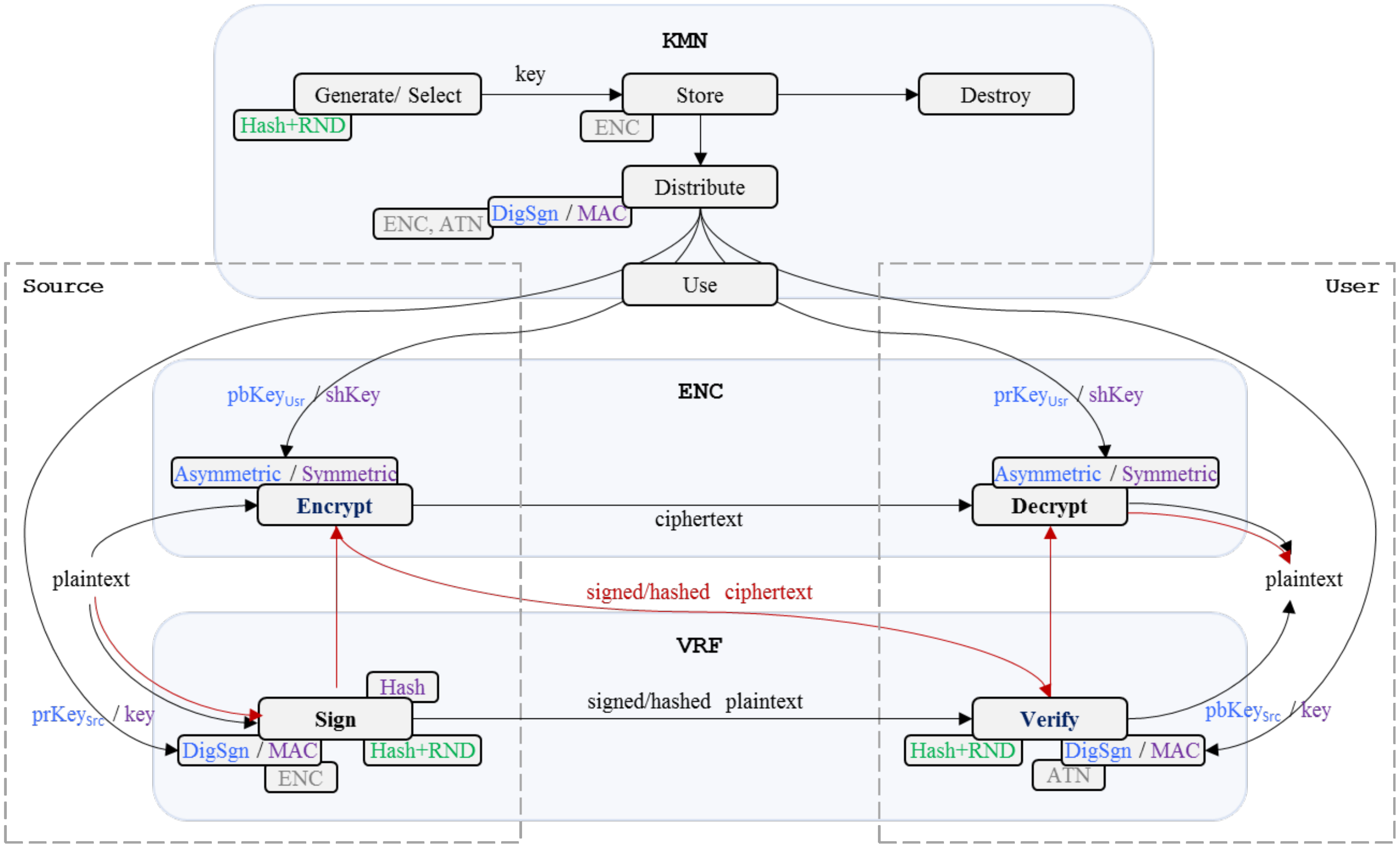

Fig. 1. Our Model of Cryptographic Store or Transfer Bugs. Encryption may occur in tandem with Verification or it may precede Verification serially, if the cipertext is signed or hashed. Encryption uses Key Management, and Key Management likely uses Encryption and Verification to handle keys.

itself uses encryption and verification. Fig. 1 presents a model of these recursive interactions and where potentially the corresponding ENC, VRF, KMN, and other BF bugs could happen. The rounded rectangles indicate the boundaries of the classes. The dashed ones show sending and receiving entities.

KMN is a class of bugs related to key management. Key management comprises key generation, selection, storage, retrieval and distribution, and determining and signaling when keys should be abandoned or replaced. A particular protocol may use any or all of these operations. Key management could be by a third party, the source, or the user – thus the KMN area intersects the Source and User areas. A third-party certificate authority (CA) distributes public keys in signed certificates. Key management often uses a recursive round of encryption and decryption, and verification to establish a shared secret key or session key before the actual plaintext is handled.

ENC is a class of bugs related to encryption and decryption. Encryption is by the source, decryption is by the user. The encryption/decryption algorithm may be symmetric, that is uses the same key for both, or asymmetric, which uses a pair of keys, one to encrypt and the other to decrypt. Public key cryptosystems are asymmetric. Ciphertext may be sent directly to the user, and verification accompanies it separately. The red line is a case where plaintext is signed or hashed and then encrypted.

VRF is a class of bugs related to verification. Verification takes a key and either plaintext or ciphertext, signs or hashes it, then passes the result to the user. The user uses the same key or the other key from the key-pair to verify data integrity or source. Note that hash alone without any other mechanism cannot be used to verify source or to protect data integrity against attackers. However, it can be used to protect data integrity against channel errors [5].

In the cases of symmetric encryption, one secretly shared key (shKey) is used. The source encrypts with shKey, and the user decrypts also with shKey. In the cases of asymmetric encryption, pairs of mathematically related keys are used. The source pair is $\text{pbKey}_{\text{Src}}$ and $\text{prKey}_{\text{Src}}$; the user pair is $\text{pbKey}_{\text{Usr}}$ and $\text{prKey}_{\text{Usr}}$. The source encrypts with $\text{pbKey}_{\text{Usr}}$ and signs with $\text{prKey}_{\text{Src}}$. The user decrypts with $\text{prKey}_{\text{Usr}}$ and verifies with $\text{pbKey}_{\text{Src}}$.

IV. Encryption/Decryption Bugs Class – ENC

A. *Definition*

We define Encryption Bugs (ENC) as:

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

We define also Decryption Bugs as:

The software does not properly transform ciphertext into plaintext using cryptographic algorithm and key(s).

Note that “transform” is for confidentiality.

ENC is related to KMN, Randomization (RND), and Information Exposure (IEX).

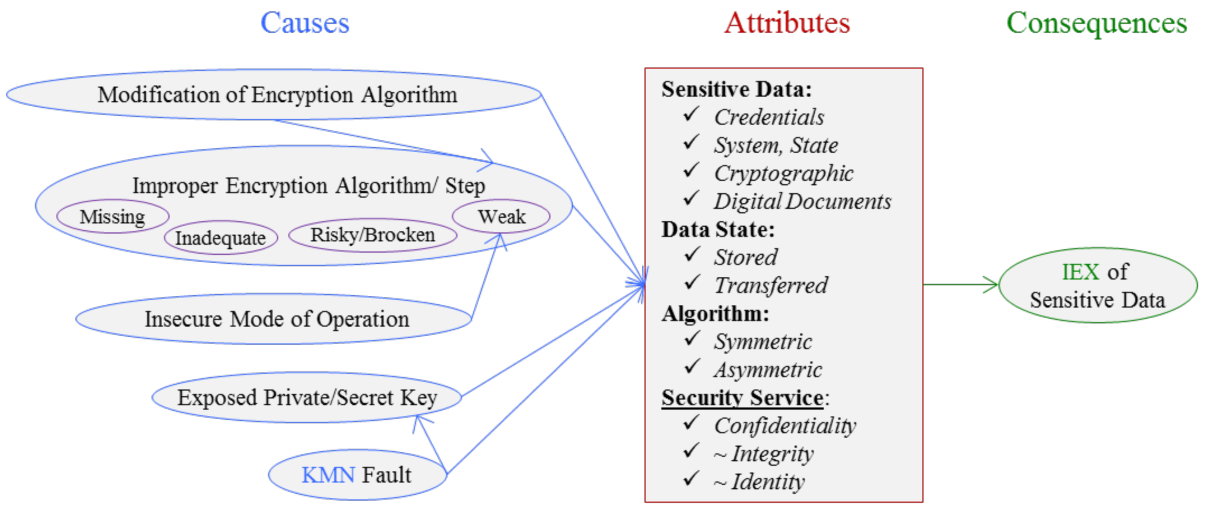

Fig. 2. The Encryption Bugs (ENC) class represented as causes, attributes and consequences.

B. *Taxonomy*

Fig. 2 depicts ENC causes, attributes and consequences. In the graph of causes, modification of algorithm is remove/change or add a cryptographic step. Improper algorithm or step could be missing, inadequate, weak, risky/broken. Insecure mode of operation leads to weak encryption algorithm.

The attributes of ENC are:

Sensitive Data – Credentials, System Data, State Data, Cryptographic Data, Digital Documents. This is secret (confidential) data. Credentials include password, token, smart card, digital certificate, biometrics (fingerprint, hand configuration, retina, iris, voice.) System Data could be configurations, logs, Web usage. Cryptographic Data is hashes, keys, and other keying material.

Data State – Stored, Transferred. This reflects if data is in rest or use, or if data is in transit. Secure store is needed for data that is in rest or use from files (e.g. ini, temp, configuration, log server, debug, cleanup, email attachment, login buffer, executable, backup, core dump, access control list, private data index), directories (Web root, FTP root, CVS repository), registry, cookies, source code & comments, GUI, environmental variables. Secure transfer is needed also for data in transit between processes or over a network.

Algorithm – Symmetric, Asymmetric. This is the key encryption scheme used to securely store/transfer sensitive data. Symmetric (secret) key algorithms (e.g. Serpent, Blowfish) use one shared key. Asymmetric (public) key algorithms (e.g. Diffie-Hellman, RSA) use a pair of keys: public and private.

Security Service(s) – Confidentiality (and Integrity and Identity Authentication). This is the security service that was failed by the encryption process. Confidentiality is the main security service provided by encryption. Those marked with '~' are only for some specific modes of encryption.

ENC is a high level class, so sites do not apply.

C. *Examples*

1) CVE-2002-1946

This vulnerability is listed in [7] and discussed in [8, 9, 10]. Our BF description is:

ENC: *Use of weak symmetric encryption algorithm (one-to-one mapping) allows confidentiality failure of stored (in registry) sensitive data (passwords), which may be exploited for IEX of that sensitive data (passwords).*

Analysis (based on [8, 9, 10]): The one-to-one mapping uses two fixed arrays of characters. There was no remedy as of 09/01/2014!

2) CVE-2002-1697

This vulnerability is listed in [11] and discussed in [12, 13, 14]. Our BF description is:

ENC: *Use of insecure mode of operation (ECB) leads to weak symmetric encryption algorithm (for same shared key produces same ciphertext from same plaintext) and allows confidentiality failure of transferred sensitive data, which may be exploited for IEX of that sensitive data.*

Analysis (based on [12, 13, 14]): Using electronic codebook (ECB) results in weak encryption, that produces the same ciphertext from the same plaintext blocks. This is a case of deterministic encryption, where patterns in plaintext become evident in the ciphertext.

D. *Related CWEs and SFP*

CWEs related to ENC are: CWE-256, 257, 261, 311-318, 325, 326, 327, 329, 780 [3].

The related SFP clusters are SPF 17.1 Broken Cryptography and SFP 17.2 Weak Cryptography under Primary Cluster: Cryptography [4]. Note that some of the CWEs listed there are not ENC.

V. VERIFICATION BUGS CLASS – VRF

A. *Definition*

We define Verification Bugs (VRF) as:

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

Note that "check" is for identity authentication, "prove" is for origin (signer) non-repudiation, and "not altered" is for integrity authentication.

VRF is related to KMN, RND, ENC, Authentication (ATN), IEX.

B. *Taxonomy*

Fig. 3 depicts VRF causes, attributes and consequences. In the graph of causes, modification of algorithm and improper algorithm or step have the same meaning as in ENC.

The attributes of VRF are:

Verified Data – Secret, Public. This is the data that needs verification. It may be confidential or public. Secret (confidential) data could be cryptographic hashes, secret keys, or keying material. Public data could be signed contract, documents, or public keys.

Algorithm – Hash Function + RND, message authentication code (MAC), Digital Signature. Hash functions are used for integrity authentication. They may use RND. MAC are symmetric key algorithms (one secret key per source/user), used for integrity authentication, identity authentication. It needs authentication code generation, source signs data, user gets tag for key and data, and verifies data by tag and key. Digital Signature is an asymmetric key algorithm (two keys), used for integrity and identity authentication, and origin (signer) non-repudiation. It needs key generation, signature generation, and signature verification. MAC and Digital Signature use KMN and recursively VRF.

Security Service – Data Integrity Authentication, Identity Authentication, Origin (Signer) Non-Repudiation. This is the security service the verification process failed. Integrity Authentication is for data and keys. Identity Authentication and Origin Non-Repudiation are for source authentication.

VRF is a high level class, so sites do not apply.

C. *Examples*

1) CVE 2001-1585

This vulnerability is listed in [15] and discussed in [16, 17]. Our BF description is:

VRF: *Missing verification step (challenge-response of private key using digital signature) in public key authentication allows identity authentication failure, which may be exploited for IEX.*

Analysis (based on [16, 17]): The step that should be included is challenge-response authentication: The client is required by the server to sign a message using the client's private key. Successful verification of that signature by the server, using the public key, confirms that the client owns the private key that is paired with that public key, and therefore that client should be allowed to login. That challenge-response authentication step is missing.

2) CVE-2015-2141

This vulnerability is listed in [18] and discussed in [19, 20, 21]. Our BF description is:

VRF: *Modification of digital signature verification algorithm (Rabin-Williams) by adding a step (blinding) leads to recovery of private key, that allows identity authentication failure, which may be exploited for IEX.*

Analysis (based on [19, 20, 21]): Having the private key allows an attacker to be authenticated as the owner of that key.

The software intends to use blinding to defend against a timing attack, as follows: Instead of signing the data directly, the data is first transformed using a secret random value (blinding) and then is digitally signed using a private key. At the end, the effect is removed (unblinding), so that there is signed data as if no transformation took place. See [20, 21] for blinding used for RSA.

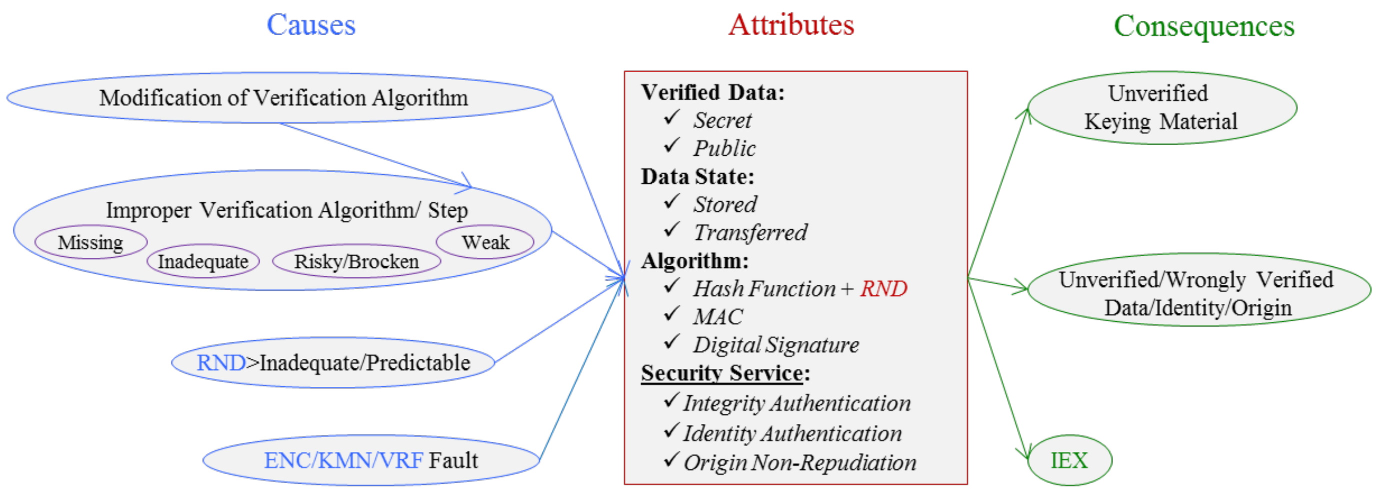

Fig. 3. The Verification Bugs (VRF) class represented as causes, attributes and consequences.

The flaw in this CVE is in doing blinding/ unblinding incorrectly, so that in some cases the effect of the transformation is not removed from the data. This enables the attacker to use the transformed data to recover the private key using a mathematical calculation as described in [20]. In [20] it is observed that if the secret random integer used to transform the message is a quadratic residue modulo an appropriate integer, then the unblinding step correctly undoes the transformation. The fix in [20] assures that the integer is such a quadratic residue.

D. *Related CWEs and SFP*

CWEs related to VRF are: CWE-295-296, 347 [3].

The related SFP cluster is 17.2 Weak Cryptography under Primary Cluster: Cryptography [4]. Note that some of the listed CWEs are not VRF.

VI. Key Management Bugs Class – KMN

A. *Definition*

We define Key Management Bugs (KMN) as:

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

KMN is related to ENC, RND, VRF, IEX.

B. *Taxonomy*

Fig. 4 depicts KMN causes, attributes and consequences.

The attributes of KMN are:

Cryptographic Data – Hashes, Keying Material, Digital Certificate.

Algorithm – Hash Function + RND, MAC, Digital Signature. Different cryptosystem have their own key generation algorithm(s).

Operation – Generate or Select, Store, Distribute, Use, Destroy. This is the failed operation. Generate uses RND. Store includes update and recover. Distribute includes key establishment, transport, agreement, wrapping, encapsulation, derivation, confirmation, shared secret creation; uses ENC and KMN (reclusively).

KMN is a high level class, so sites do not apply.

C. *Examples*

1) *CVE-2016-1919*

This vulnerability is listed in [22] and discussed in [23]. Our BF description is:

A KMN leads to an ENC.

KMN: *Use of weak algorithm (eCryptFS-key from password and stored TIMA key) allows generation of keying material (secret key) that can be obtained through brute force attack, which may be exploited for IEX of keying material (the secret key).*

ENC: *KMN fault leads to exposed secret key that allows confidentiality failure of stored sensitive data, which may be exploited for IEX of that sensitive data.*

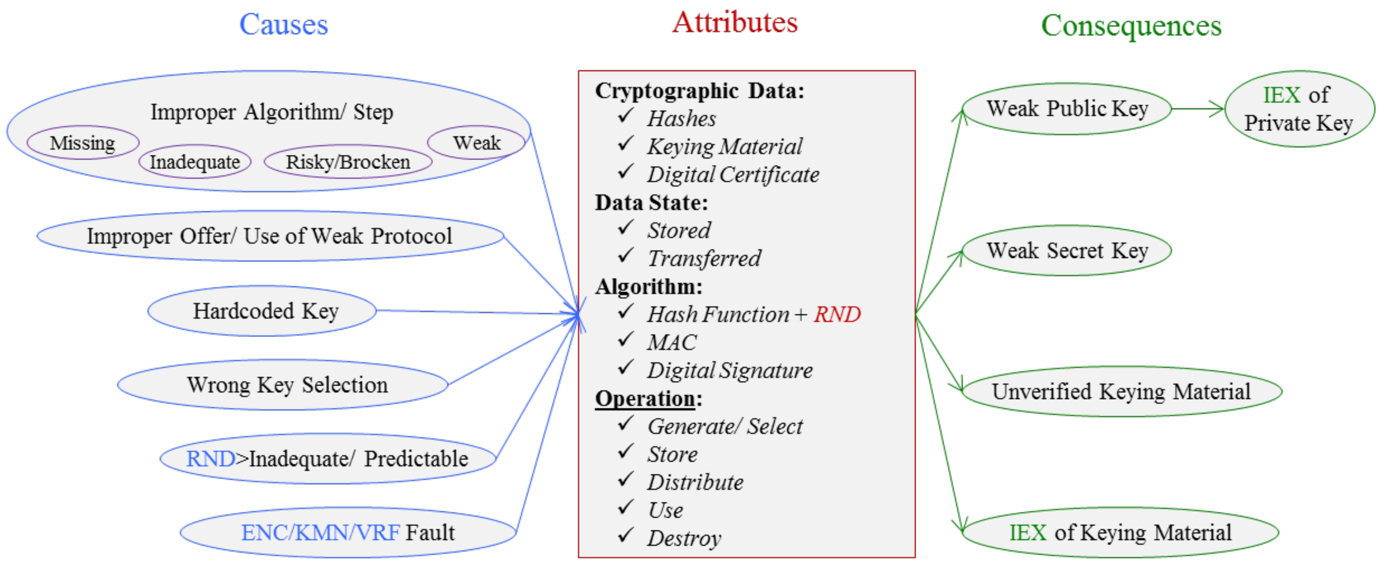

Fig. 4. The Key Management Bugs (KMN) class represented as causes, attributes and consequences

Analysis (based on [23]): The set of possible keys is a known small set.

The TIMA key is a random stored byte string. The secret key used is obtained by XOR of the TIMA key and the password characters, where the minimum password length is 7. However, if the password length is no more than 8, a base 64 expansion results in a key that does not depend on the password. The TIMA key is stored, and for a known TIMA key, the key is known, or, if the password length slightly exceeds 8, there is a small set of possible keys. The TIMA key can be obtained using a preliminary step.

2) CVE-2015-0204, 1637, 1067 (FREAK - Factoring attack on RSA-ExportKeys)

This vulnerability is listed in [24, 25, 26] and discussed in [27, 28, 29, 30]. Our BF description is:

An inner KMN leads to an inner ENC, which leads to an outer ENC.

Inner KMN: *Client-accepted improper offer of weak protocol (SSL with Export RSA) from MITM-tricked server allows use of an algorithm (Export RSA) that generates 512-bit keying material (pair of keys), for which private key may be obtained through factorization of public key, which may be exploited for IEX of keying material (the private key).*

Inner ENC: *KMN fault leads to exposed private key for asymmetric encryption (RSA) that allows confidentiality failure of transferred sensitive data (Pre-Master Secret), which may be exploited for IEX of sensitive data (Master Secret).*

Outer ENC: *KMN fault leads to exposed secret key (Master Secret) for symmetric encryption allows confidentiality failure of transferred sensitive data (passwords, credit cards, etc.), which may be exploited for IEX of that sensitive data (passwords, credit cards, etc.).*

Inner KMN and inner ENC only set up the secret key. Outer ENC is the actual general data transfer.

Interestingly in this example the consequence from the first bug (inner KMN) causes the second bug (inner ENC), whose consequences cause the third bug (outer ENC). The inner KMN is a server bug, sending a weak key, (that the client did not ask for), intended for KMN use by client (encrypting Pre-Master Secret). It is also a client bug, as the client accepted the offer of using the insecure method, and therefore the server proceeded. The client could have refused that offer. The inner ENC is a client bug, using that weak key to encrypt the Pre-Master Secret, and then transmitting that weakly encrypted Pre-Master Secret over a network that is not secure.

Analysis (based on [27, 28, 29, 30]): The server offers a weak protocol (Export RSA) while the client requested strong protocol (RSA).

Communication is encrypted by symmetric encryption. The key for that encryption (Master Secret) is created by both client and server from a Pre-Master Secret and nonces sent by client and server. The Pre-Master Secret is sent encrypted by RSA cryptosystem. The client requests RSA protocol, but man in the middle (MITM) intercepts and requests Export RSA that uses a 512 bit key. Factoring a 512 bit RSA key is feasible.

Because of a bug, the client agrees to Export RSA. MITM factors the public 512 bit public RSA key, uses this factoring to recover the private RSA key, and then uses that private key to decrypt the Pre-Master Secret. Then it uses the Pre-Master Secret and the nonces to generate the Master Secret. The Master Secret enables MITM to decrypt the encrypted communication from that point on.

D. *Related CWEs and SFP*

CWEs related to KMN are: CWE-321, 322, 323, 324 [3].

The related SFP clusters are SFP 17.2 Weak Cryptography under Primary Cluster: Cryptography and SFP 4.13 Digital Certificate under Primary Cluster: Authentication [4]. Note that, some of the CWEs listed in 17.2 are not KMN.

VII. Concluding Remarks

A. *Summary*

We presented three new BF classes: Encryption/Decryption Bugs (ENC), Verification Bugs (VRF), and Key Management Bugs (KMN). They join other rigorously-defined classes: Injection (INJ), Control of Interaction Frequency Bugs (CIF), and Buffer Overflow (BOF). 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 description of FREAK, using KMN and ENC, is quite concise and still far clearer than unstructured explanations that we have found.

B. *Lessons Learned*

At first, we tried to define a Cryptography class with attributes, causes, and consequences. However, we realized that subsidiary processes, like randomization and verification, are used in completely different contexts. As we developed a model of cryptographic store and transfer bugs, we learned how rich and subtle the relations between processes were. We ended up defining separate ENC, VRF, and KMN classes of bugs.

We found that a model of the cryptographic store or transfer processes helps: it illustrated the relation between the classes and helped us determine what operations should be included and what should not. We learned that the structure of BF is *not* settled. It may need to be refined further.

C. *Future Work*

One of our next steps is to explain more cryptographic bugs using ENC, VRF, and KMN. We also need to explore the use of ENC, VRF, and KMN in contexts other than cryptographic store and transfer. This will show where BF structures need refinements.

Another step is to develop other BF classes. We are currently working on Randomization Bugs (RND), Authentication Bugs (ATN), Authorization Bugs (AUT), Information Exposure (IEX), Faulty Operation (FOP), and Memory Allocation Bugs (MAL). FOP includes faults during arithmetic operations, such as integer overflow and divide by zero. MAL includes memory allocation faults, like use after free, multiple free, and failure to free.

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

REFERENCES

[1] I. Bojanova, P. E. Black, Y. Yesha, and Y. Wu, "The Bugs Framework (BF): A Structured Approach to Express Bugs," 2016 IEEE International Conference on Software Quality, Reliability, and Security (QRS 2016). Vienna, Austria. August 1-3 2016.

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

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

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

[5] E. Barker, "NIST Special Publication 800-175B Guideline for Using Cryptographic Standards in the Federal Government: Cryptographic Mechanisms," August 2016. http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-175B.pdf.

[6] A. Tobin and D. Reed, "The Inevitable Rise of Self-Sovereign Identity", September 2016, https://sovrin.org/wp-content/uploads/2017/07/The-Inevitable-Rise-of-Self-Sovereign-Identity.pdf.

[7] The MITRE Corporation, CVE-2002-1946, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1946.

[8] The MITRE Corporation, CWE 326, https://cwe.mitre.org/data/definitions/326.html.

[9] SecurityTracker. VSNL Integrated Dialer Weak Encoding Discloses Passwords to Local Users Alert ID: 1005515, http://securitytracker.com/id/1005515.

[10] IBM X-Force Exchange, Integrated Dialer Software stores passwords using weak encryption algorithm: CVE-2002-1946, https://exchange.xforce.ibmcloud.com/vulnerabilities/10517.

[11] The MITRE Corporation, CVE-2002-1697, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-1697.

[12] Wikipedia, RSA (cryptosystem), http://en.wikipedia.org/wiki/RSA_(cryptosystem).

[13] Seclists, Security weaknesses of VTun, http://seclists.org/bugtraq/2002/Jan/119.

[14] Wikipedia, Deterministic encryption, https://en.wikipedia.org/wiki/Deterministic_encryption.

[15] The MITRE Corporation, CVE-2001-1585, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-1585.

[16] OpenBSD Security Advisory, Authentication By-Pass Vulnerability in OpenSSH-2.3.1, http://www.openbsd.org/advisories/ssh_bypass.txt.

[17] S. Tatham, PuTTY User Manual – Chapter 8: "Using public keys for SSH authentication," http://the.earth.li/~sgtatham/putty/0.60/htmldoc/Chapter8.html.

[18] The MITRE Corporation, CVE-2141, http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=2015-2141

[19] Bugzilla – Bug 936435, VUL-0: CVE-2015-2141: libcryptopp: libcrypto++ -- security update, https://bugzilla.suse.com/show_bug.cgi?id=936435.

[20] E. Sidorov, "Breaking the Rabin-Williams digital signature system implementation in the Crypto++ library," 2015, http://eprint.iacr.org/2015/368.pdf.

[21] Wikipedia, "Blinding Cryptography," https://en.wikipedia.org/wiki/Blinding_(cryptography).

[22] The MITRE Corporation, CVE-2016-1919, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-1919.

[23] openwall.net, [CVE-2016-1919] "Weak eCryptFS Key generation from user password," http://lists.openwall.net/bugtraq/2016/01/17/2.

[24] The MITRE Corporation, CVE--2015-0204, https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2015-0204.

[25] The MITRE Corporation, CVE--2015-1637, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-1637.

[26] The MITRE Corporation, CVE--2015-1067, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-1067.

[27] R. Heaton, The SSL FREAK vulnerability explained, http://robertheaton.com/2015/04/06/the-ssl-freak-vulnerability.

[28] Censys, The FREAK Attack. https://censys.io/blog/freak

[29] StackExchange, Protecting phone from the FREAK bug, http://android.stackexchange.com/questions/101929/protecting-phone-from-the-freak-bug/101966.

[30] GitHub, openssl, Only allow ephemeral RSA keys in export ciphersuites, https://github.com/openssl/openssl/commit/ce325c60c74b0fa784f5872404b722e120e5cab0?diff=split.