EDITOR: Irena Bojanova, irena.bojanova@computer.org

DEPARTMENT: CYBERSECURITY

Bug, Fault, Error, or Weakness: Demystifying Software Security Vulnerabilities

Irena Bojanova, *NIST, Gaithersburg, MD, 20899, USA*

Carlos Eduardo C. Galhardo, *INMETRO, Duque de Caxias, RJ, 25250-020, Brazil*

Abstract—In this work, we define the notions of software bug, weakness, and vulnerability in the context of cybersecurity and elucidate their causal relations.

Security vulnerabilities lead to failures that are commonly used to attack cyberspace and the critical infrastructure. Communicating about them, however, even security experts use loosely the notions of bug, fault, weakness, vulnerability, and failure. For example, artificial intelligence (AI) chatbots are trained on input from all over the Internet including misunderstandings. Subsequently, conflating explanations about these concepts resurface, providing a glimpse on how software security concepts are used in publications, security advisories, and testing tool reports. Figure 1 shows one particular interaction with ChatGPT on defining software security bug and weakness. Skimming through the definitions, a reader may find them sound, unaware the AI algorithm has been forced to mix apples and oranges.

Building a common language that avoids conflation, synonymy, and polysemy is a critical challenge in systems engineering [1]. This is also true for cybersecurity. For example, misunderstanding the root cause is a critical factor for reopening fixed bugs [2], and the lack of clearly defined concepts hinders the automatic processing of security-related information [3].

In this paper, we clearly define the software security terms bug, fault, error, weakness, vulnerability, and failure, as part of our work on the NIST Bugs Framework (BF) ([4]). Starting from vulnerability, we go towards weakness, bug, and fault, and then towards error, final error, and failure. We model a software security vulnerability as a chain of weaknesses that leads to a security failure. A security bug causes the first weakness, leading to an error. This error becomes the cause (i.e., the fault) for a next weakness and propagates through subsequent weaknesses until a final error is reached, causing the security failure.

Digital Object Identifier 10.1109/XXX.0000.0000000

DEFINITIONS

A broad spectrum of software security researchers and practitioners would benefit from clear definitions of the terms bug, fault, error, weakness, vulnerability, and failure in the context of software security.

We can observe that a security vulnerability leads to a security failure. However, what are the building blocks of a vulnerability? What is the defect in software that triggers a vulnerability? How does it propagate through errors until a final, exploitable error is reached? How do the underlying weaknesses of a vulnerability relate to these propagating errors?

A *software security vulnerability* is a chain of weaknesses linked by causality. It starts with a bug and ends with a final error, which if exploited leads to a security failure.

A *software security weakness* is a `(bug, operation, error)` or `(fault, operation, error)` triple. It is an instance of a weakness type that relates to a distinct phase of software execution, the operations specific for that phase and the operands required as input to those operations.

A *software security bug* is a code or specification defect – an operation defect.

A *software fault* is a name, data, type, address, or size error – an operand error. Name is in reference to a resolved or bound object, function, data type, or namespace. The others are in reference to an object.

A *software error* is the result from an operation with a bug or a faulty operand. It becomes a next fault or is a final error.

Disclaimer: Certain trade names and company products are mentioned in the text or identified. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology (NIST), nor that they are necessarily the best available for the purpose.

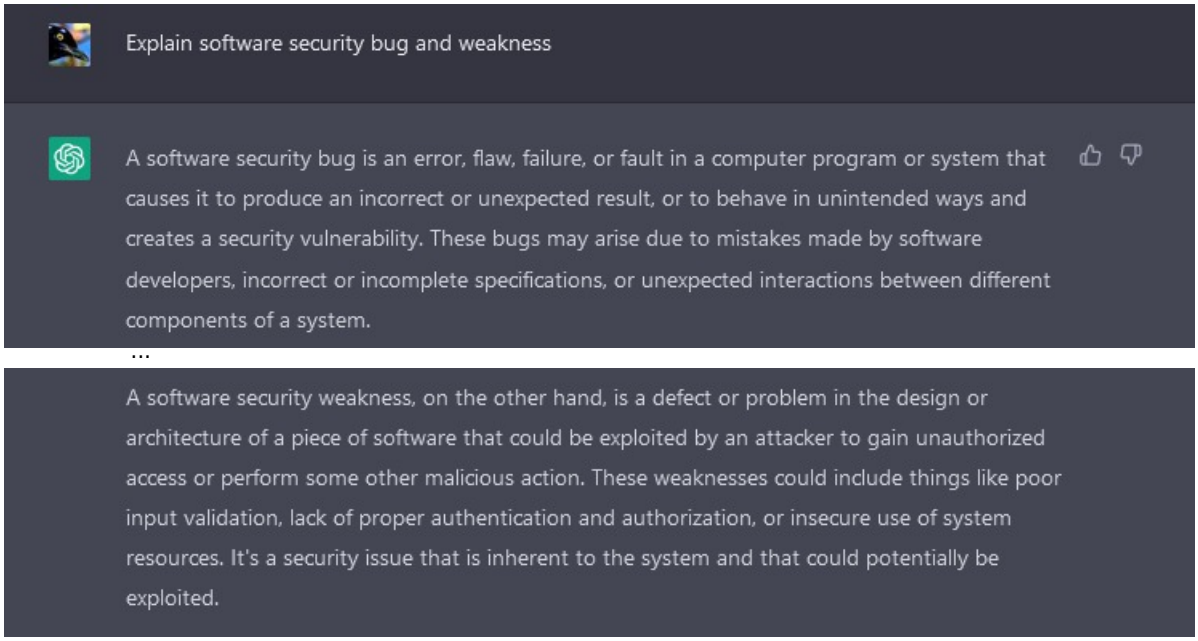

FIGURE 1. Software security concepts conflation, demonstrated by querying the Internet via ChatGPT.

A *software security final error* is an exploitable or undefined system behavior that leads to a security failure.

A *security failure* is a violation of a system security requirement.

A *chain of weaknesses* starts with a bug, propagates through errors that become faults, and ends with a final error. The final error is the one exploited by attackers towards a security failure. For example, missing input validation may propagate to integer overflow, followed by buffer overflow, which if exploited may lead to a remote code execution failure.

The bug must be fixed to resolve the vulnerability; while, in most cases, fixing a fault would only mitigate the vulnerability. To fix a bug (code or specification defect), lines of code or configuration files, etc., must be changed. The bug is a concrete error; it is a wrong sequence of bits that must be changed. Fixing a specification is also code related, as it requires fixing its implementation.

A security failure may be caused by the converging final errors of several vulnerabilities. The bug in at least one of the chains must be fixed to avoid the failure.

Using our definitions, we formalize at a high-level a vulnerability description with the rules in Listing 1 (the complete current BF LL1 grammar is available at [4]).

Listing 1. A high-level grammar of a vulnerability description.

```
START := Vulnerability Converge END
Vulnerability := Bug Operation Error
Error := Fault Operation Error
       | FinalError
Converge := Vulnerability Converge
          | Failure
```

VULNERABILITY MODEL

Figure 2 presents our BF software security vulnerability model. Following the definitions of weakness, bug, operation, error, and final error and our formal grammar (Listing 1), a vulnerability description uses causal relations to form a chain of underlying weaknesses, leading to a failure.

Each weakness is an instance of a weakness type with a particular bug or fault as a cause and an error as a consequence. The error establishes a transition to another weakness or a failure.

A bug always causes the first weakness in a chain of weaknesses[1]; it is a coding or specification defect, which, if fixed, will resolve the vulnerability. A fault causes each intermediate state. The last weakness always ends with a final error (undefined or exploitable

[1]Focus of this work are weaknesses within software.

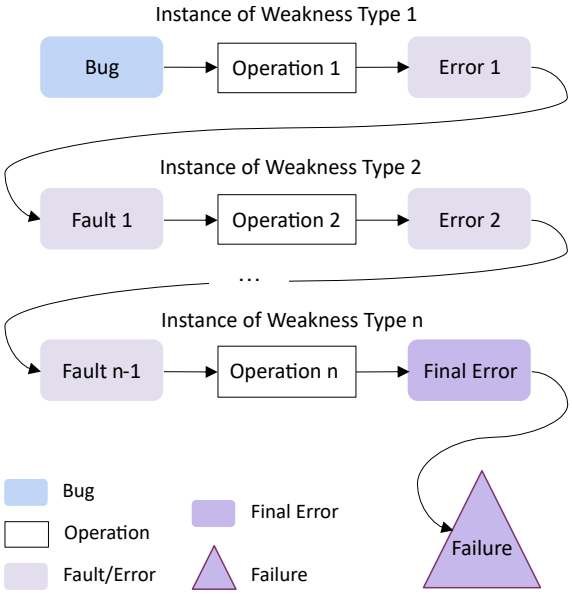

FIGURE 2. The BF software security vulnerability model. A chain of underlying weaknesses, leading to a security failure.

system behavior) that causes the failure (a violation of a system security requirement).

A transition is the result of the operation over the operands. For example, in Figure 2, Operation 1 from the first weakness has a Bug and results in Error 1, which becomes the fault for operation 2, leading to Error 2. The chain goes on, until the last operation results in a Final Error, leading to a failure.

Therefore, a vulnerability can be described precisely as a chain of weaknesses and their transitions. This chain is a sequence of improper states in the vulnerable software.

Each improper state is an instance of a weakness type, corresponding to a Bugs Framework (BF) class [4]. The transition from the initial state is by improper operation (an operation that has a bug) over proper operands. The transitions from intermediate states are by proper operations with at least one improper operand (the operand is at fault).

In some cases, several vulnerabilities must be present for an exploit to be harmful. The final errors resulting from different chains converge to cause a failure (see Figure 3). The bug in at least one of the chains must be fixed to avoid that failure.

EXAMPLE

Let's look at BadAlloc, a pattern discovered by Microsoft researchers [5] and reported by the Cybersecurity and Infrastructure Security Agency (CISA) [6] with

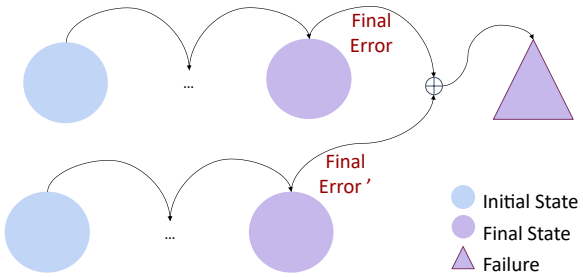

FIGURE 3. Converging software security vulnerabilities, leading to a security failure.

25 similar vulnerabilities found in multiple IoT devices.

The BadAlloc vulnerability pattern comprises five consecutive weaknesses (see Figure 4). The first weakness occurs at the data verification phase of software execution. There is a bug, such as missing code for checking data towards allowed numerical values, creating a data verification weakness. This allows input of an unusually large number [2], which causes a wrap-around error when performing arithmetic calculations (a type computation weakness). This error results in a smaller number being used for memory allocation, leading to not enough memory reserved for a buffer (a

[2]E.g., greater than the maximum allowed integer – $2^{32} - 1$ for 32-bit systems.

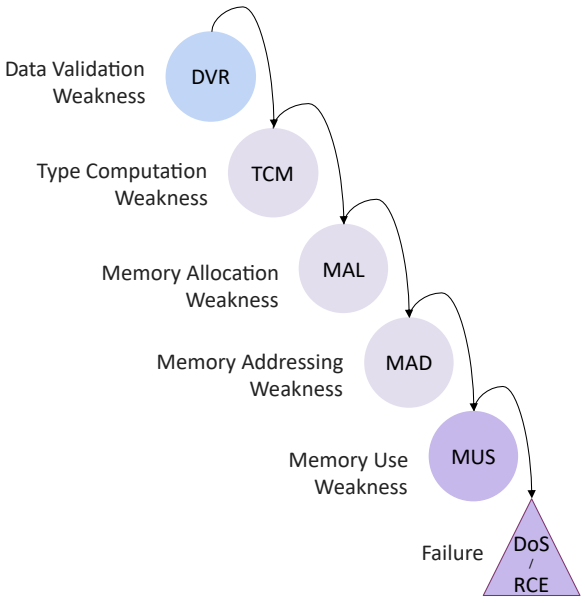

FIGURE 4. The BadAlloc vulnerability pattern, described using the BF classification [4].

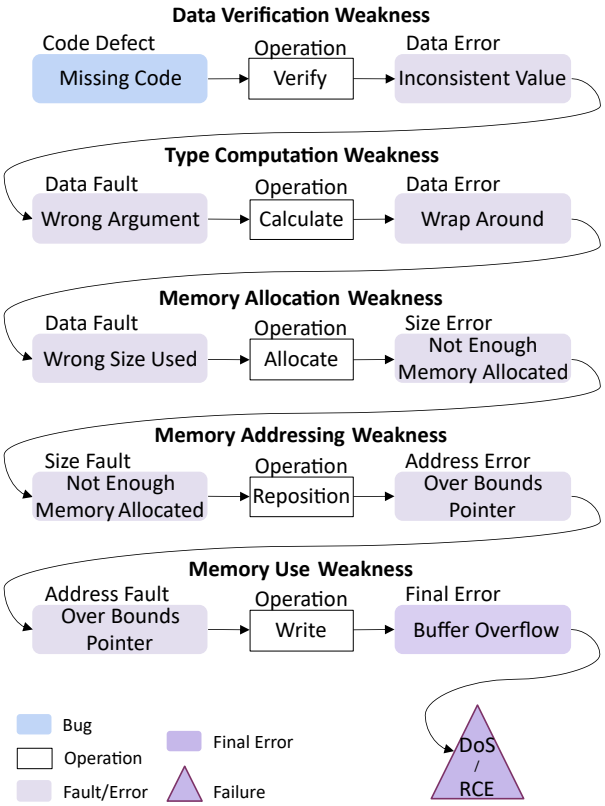

FIGURE 5. The BF [4] BadAlloc chain as in CVE-2021-21834.

memory allocation weakness). This allows a pointer to move outside the buffer boundaries (a memory addressing weakness) and cause a buffer overflow final error while writing data there (a memory use weakness). The final error then can lead to a failure, such as denial of service or remote code execution.

Figure 5 presents the chain of weaknesses, underlying the particular BadAlloc vulnerability CVE-2021-21834. To examine its fully detailed BF description, please refer to [7].

DISCUSSION

Explanations generated by ChatGPT about particular software security notions depend on the dialog, but it is astonishing how wrong they could be to start with. For example, the result from the query on Figure 1 states a security bug is "an error, a flaw, a failure, or a fault" that "causes incorrect or unexpected results" and "creates a security vulnerability". It then plays with the common understanding of vulnerability [8] instead of explaining weakness, except it adds a "weakness is a defect or problem in the design of a piece of software". While a bug is not a failure, a weakness is not a vulnerability, defect relates to bug, and fault relates to error. The results support our own research conclusion that clear definitions of all these notions are greatly needed.

Our vulnerability model (Figure 2) and definitions can be used on a high enough abstract level to describe the weakness pattern for several vulnerabilities – e.g., the BadAlloc pattern (Figure 4). However, they are also concrete enough to provide details on the weaknesses underlying a particular vulnerability – e.g., the CVE-2021-21834 chain (Figure 5). Our approach allows to reveal how same types of weakness chain to form different vulnerabilities and how a particular bug in a piece of code leads to a failure.

Understanding the role of faults as propagating errors in a chain of weaknesses, makes it easier to see that the final error of a vulnerability is the one that gets exploited. Recall the BadAlloc pattern and the CVE-2021-21834 description adherent to our vulnerability model: an unverified large input to a calculation at buffer allocation causes use of a wrapped-around value as size, leading to a smaller buffer and allowing overbound writes. First and more importantly, we learn about the vulnerability severity: a write buffer overflow may crash the system or, even worse, allow remote code execution. Next, we learn what should be fixed to resolve the vulnerability: the missing input verification bug. Last, we can reason about in-depth defense measures to mitigate the vulnerability. For example, use of address space layout randomization to mitigate buffer overflow on dynamically allocated memory or safe integer libraries to mitigate wrap-around errors. Understanding the chain of weaknesses allows better development, defense, and mitigation decisions. It would be misleading to say CVE-2021-21834 has a buffer overflow bug or is a buffer overflow failure. Much would be missing also if we only say it has a buffer overflow weakness.

Note that the BF vulnerability model focuses on weaknesses within software. Embedding it in attack specific models (e.g., NIST Vulntology [9]), would allow external causes, such as hardware failures, system misconfigurations, interactions with other software, or human interactions.

RELATED WORK

In this section, we compare and contrast our BF vulnerability model with related works.

Chillarege et al. introduce the idea of causal, orthogonal classification of defects [10]. Our definition of bug parallels their definition of defect. However, we delve deeper and differentiate the initial defect (the bug) from the propagated errors (the faults). We define

all the concepts on a level of abstraction that would help clear explanation of a causal chain from the bug through faults to the eventual security failure. Facilitating clear communication about security vulnerabilities is our main goal. Our approach, however, by its nature, may also allow automated backtracking to the bug [11].

The reliability community has also struggled to define the concepts of software defect, fault, error, and failure. Several papers from the 90's discuss these concepts. Some found the hardware analogy tempting [12], but it had limits and it was found confusing [13]. Instead, we build our definitions from the notion of software security vulnerability as the cause of security failure (i.e., loss of a security property). The failure is triggered by a software security bug unintended functionality that breaks basic security principles.

Avizienis et al. [14] define security faults, errors, and failures using causal relationships. They explain that errors propagate inside components from an initial fault until a failure is reached. Their definitions of fault as "the adjudged or hypothesized cause of an error" and error as "a part of a system's total state that may lead to a failure" reflects our understanding of bug and fault. We also reason a fault is a cause for an error, but in addition we deem recurrence essential to explain better how errors propagate in software. For example, an erroneous result of an operation could be a faulty cause for a next operation. We define the bug as the cause of error from the initial improper state, propagating through errors from intermediate states, towards the error from the final state, which leads to a failure. They state that error propagation is through the computation process, however, they do not delve as deep as we do. We bring up the concept of operation (and its operands) to explain how an error, resulting from a bug or fault, transitions into the fault, causing another error. We state a vulnerability is underlined by a chain of weaknesses, each corresponding to a particular bug or fault and a particular operation that results in an error. The notion of transition is important as the error resulting from a weakness can be more abstract than the concrete fault of a next weakness.

CONCLUSION

In this paper, we define the fundamental notions of security failure and software security vulnerability, weakness, bug, and final error; and detail the definitions of software fault and error. We have developed them iteratively, while creating the NIST Bugs Framework (BF) [4] software security vulnerability model. They help us reason about and create weakness taxonomies, allowing precise descriptions of existing vulnerabilities.

A broad spectrum of information technology (IT) managers, software developers, and security researchers would benefit from a clear understanding of these terms in the context of software security. Accurate understanding of underlying weaknesses would ensure proper bug identification, which could improve fixing times and decrease chances of introducing new bugs via patches. Formalized definitions would assist in machine processing of security-related information and in generating software testing reports.

The results from ChatGPT queries show we must rely on more than just AI to discern concepts. Under the ChatGPT's hood lays a model that learns from all over the Internet, including misunderstandings. In parallel to the ancient Oracle of Delphi, the caller should be well prepared to provide the right questions and context; otherwise, the reasoning may be misleading and the result disastrous. Using our software security expertise to pose more and more tuned questions, thus providing more context, eventually we got ChatGPT to at least partially discern our own reasoning. The collective knowledge seems to approve the direction we are delving in via our BF research ([4]).

Irena Bojanova, is a computer scientist at NIST, USA. She is the primary investigator and lead of the NIST Bugs Framework (BF) project. Her current research interests include cybersecurity and formal methods. She is a Senior member of the IEEE Computer Society. Contact her at irena.bojanova@nist.gov.

Carlos E. C. Galhardo, is a researcher at Inmetro, Brazil. His research interests include information science, cybersecurity, and mathematical modeling in interdisciplinary applications. Contact him at cegalhardo@inmetro.gov.br.

References

[1] D. A. Broniatowski, "Building the tower without climbing it: Progress in engineering systems," *Systems Engineering*, vol. 21, no. 3, pp. 259–281, 2018.

[2] T. Zimmermann, N. Nagappan, P. J. Guo, and B. Murphy, "Characterizing and predicting which bugs get reopened," in *2012 34th International Conference on Software Engineering (ICSE)*, 2012, pp. 1074–1083. DOI: 10.1109/ICSE.2012.6227112.

[3] D. Malzahn, Z. Birnbaum, and C. Wright-Hamor, "Automated vulnerability testing via executable attack graphs," in *2020 International Conference*

on Cyber Security and Protection of Digital Services (Cyber Security), IEEE, 2020, pp. 1–10.

[4] NIST, *The Bugs Framework*, Accessed: 2023-01-06, 2023. [Online]. Available: https://samate.nist.gov/BF/.

[5] T. A. Omri Ben-Bassat, "ERROR: BadAlloc! - Broken Memory Allocators Led to Millions of Vulnerable IoT and Embedded Devices," Slideshow presented at Blackhat USA 2021, 2021.

[6] CISA, *ICS Advisory (ICSA-21-119-04), Multiple RTOS (Update E)*, Accessed: 2022-09-08, 2022. [Online]. Available: https://www.cisa.gov/uscert/ics/advisories/icsa-21-119-04.

[7] I. Bojanova, C. E. Galhardo, and S. Moshtari, "Data type bugs taxonomy: Integer overflow, juggling, and pointer arithmetics in spotlight," in *2022 IEEE 29th Annual Software Technology Conference (STC)*, 2022, pp. 192–205. DOI: 10.1109/STC55697.2022.00035.

[8] Joint Task Force Transformation Initiative, "NIST Special Publication 800-30 revision 1: Guide for Conducting Risk Assessments," *US Dept. of Commerce*, 2012.

[9] NIST, *NIST Vulntology*, Accessed: 2023-01-18, 2023. [Online]. Available: https://github.com/usnistgov/vulntology.

[10] R. Chillarege, I. S. Bhandari, J. K. Chaar, *et al.*, "Orthogonal defect classification-a concept for in-process measurements," *IEEE Transactions on software Engineering*, vol. 18, no. 11, pp. 943–956, 1992.

[11] I. Bojanova, C. E. Galhardo, and S. Moshtari, "Input/output check bugs taxonomy: Injection errors in spotlight," in *2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 2021, pp. 111–120. DOI: 10.1109/ISSREW53611.2021.00052.

[12] B. Parhami, "Defect, fault, error,..., or failure?" *IEEE Transactions on Reliability*, vol. 46, no. 4, pp. 450–451, 1997.

[13] T. Yellman, "Failures and related topics," *IEEE Transactions on Reliability*, vol. 48, no. 1, pp. 6–8, 1999. DOI: 10.1109/TR.1999.765921.

[14] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE transactions on dependable and secure computing*, vol. 1, no. 1, pp. 11–33, 2004.