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—Intelligently surfing all over the Internet, ChatGPT demonstrates there is a high level of confusion about the notions of software security bug, weakness, and vulnerability. In this paper, we define them clearly in the context of cybersecurity and elucidate their causal relations.

Security vulnerabilities lead to failures that are commonly used to attack the cyberspace and the critical infrastructure. Communicating about them, however, even expert security researchers use interchangeably the notions of bug, fault, weakness, vulnerability, and even failure. To get a glimpse on how publications, security advisories, and test tools reports use these notions, we have asked ChatGPT [1] (the latest and grates development in AI) to explain them. The generated responses depend on the dialog, but it is astonishing how wrong they could be to start with. As an example, Figure 1 shows ChatGPT's explanations about software security bug and weakness.

ChatGPT explains 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 plays with the common understanding of vulnerability [2] instead of defining weakness, except it adds that a "weakness is a defect or problem in the design of a piece of software". While the least to point, a bug is not a failure, a weakness is not a vulnerability, defect relates to bug, and fault relates to error.

We must rely on more than just AI to discern concepts. Under the ChatGPT's hood lays is a model that learns from all over the Internet, including misunderstandings. The use of robust AI systems nowadays parallels a call in favor of the Oracle of Delphi in ancient times. The caller must be well prepared to provide the right questions and context; otherwise, the reasoning may be misleading and the result disastrous.

This supports our own research conclusion that clear definitions of all these notions are greatly needed. This is even stronger in the context of cybersecurity, where the focus is on security related software bugs, weaknesses, and vulnerabilities.

In this paper, we present a set of clear definitions starting from failure towards vulnerability, weakness, and bug, and then towards fault, error, and final error. We aim to demystify this terminology in the context of cybersecurity. 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 the subsequent weakness and propagates through intermediate weaknesses until a final error is reached, causing the security failure. We compare and discuss our vulnerability model with related works. In conclusion, we provide insights on how our definitions can benefit future research.

DEFINITIONS

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

A quick query to ChatGPT shows the security failure concept is well understood and easy to define. We can also 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?

Digital Object Identifier 10.1109/XXX.0000.0000000

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.

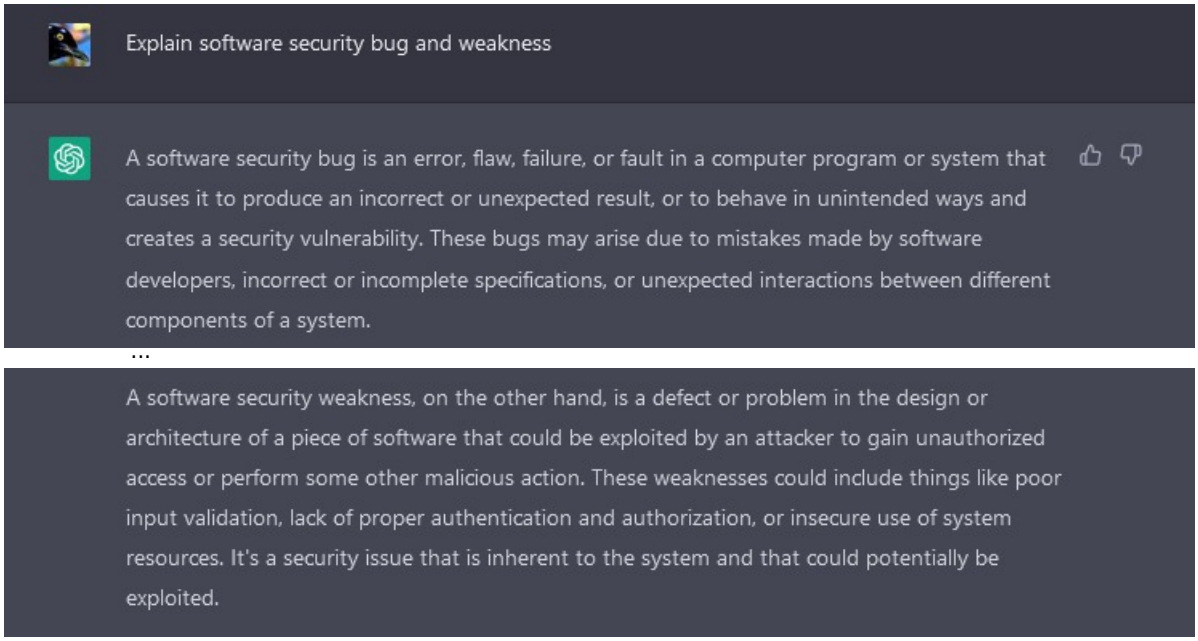

FIGURE 1. Confusing definitions generated by OpenAI ChatGPT.

The following is our list of definitions starting from security failure to software security vulnerability, weakness type, weakness, and bug; to software fault and error; to software security final error.

A *security failure* is a violation of a system security requirement. A *software security vulnerability* is an instance of a weakness type (or a chain of weakness type instances) that starts with a bug and leads to a final error. More than one underlying weaknesses get linked by causality.

A *software security weakness type* is a `(bug, operation, error)` or a `(fault, operation, error)` triple. It relates to a distinct phase of software execution, the operations specific for that phase and the operands required as input to those operations. An *operation* relates to software code or specification. An *operand* relates to data, type, address, or size of object[1]; or resolved/bound name for entity[2]. A *software security weakness* is an instance of a weakness type.

A *software security bug* is an operation (code or specification) defect. A *software fault* is an operand (data, type, address, size, or name) error. An *software error* is the result from an operation with a bug or an operation with a faulty operand. It is a final error or becomes a next fault. A *software security final error* is undefined or exploitable system behavior that leads to a security failure. A *chain of weaknesses* starts with a bug, propagates through errors that become faults, and ends with a final error. For example, missing input validation may propagate to integer overflow and end with buffer overflow. The final error is the one exploited by attackers towards a security failure. For example, the buffer overflow final error 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 it. 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 can formalize on a high level a vulnerability description with the rules shown on Listing 1 (see complete LL1 grammar at [3]).

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
```

[1]A memory region used to store data.
[2]Object, function, data type, or namespace.

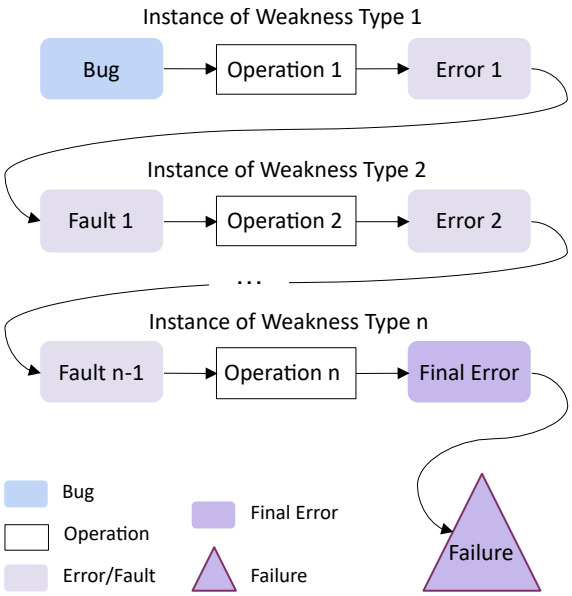

FIGURE 2. A software security vulnerability model. Shows the chain of underlying weaknesses, leading to a security failure.

VULNERABILITY MODEL

Figure 2 presents our software security vulnerability model. Following our definitions of weakness type, 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; 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 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 [3]. 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.

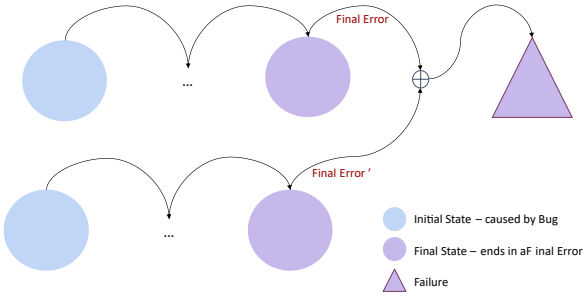

FIGURE 3. Converging software security vulnerabilities, leading to a security failure.

EXAMPLE

Let's look at "BadAlloc", a pattern discovered by Microsoft researchers [4] and reported by CISA [5], listing 25 similar vulnerabilities found in multiple IoT devices.

The BadAlloc vulnerability pattern comprises five

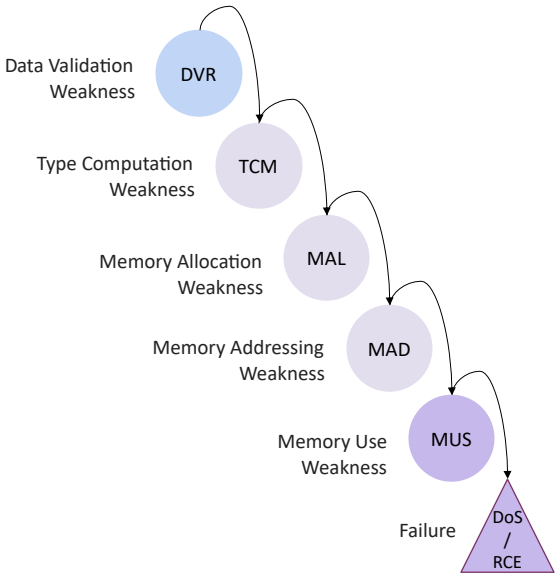

FIGURE 4. The "BadAlloc" pattern.

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 [3], which causes a wrap around error when performing arithmetic calculations (a type calculation weakness).

This error results in a smaller number being used for memory allocation, leading to not enough memory reserved for a buffer (a 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 BadAlloc pattern using the NIST BF classification [3]. To examine the detailed description of the BadAlloc chain and CVE-2021-21834 please refer [6].

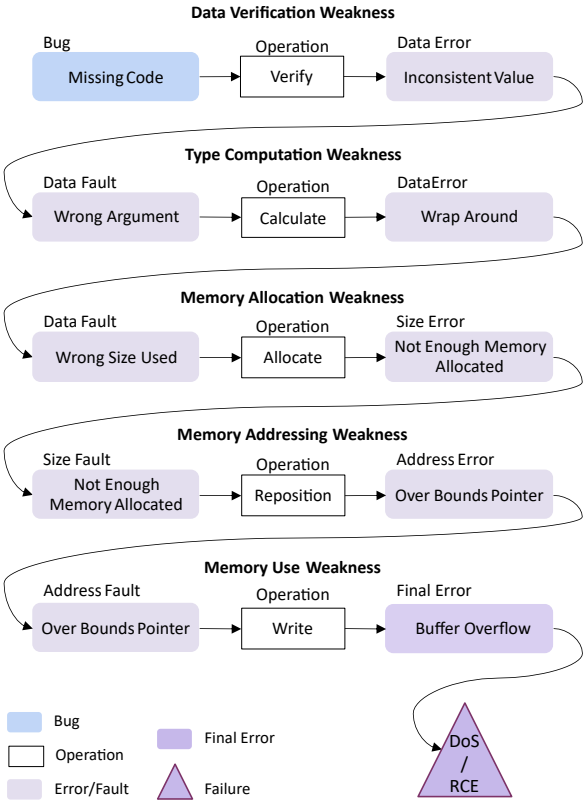

FIGURE 5. The BadAlloc chain, as in CVE-2021-21834.

RELATED WORK

Chillarege et al. introduce the idea of causal, orthogonal classification of defects [7]. 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 [8].

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 [9], but it had limits and found confusing [10]. Instead, we build our definitions from the notion of security failure as loss of a security property. It is a triggered by a software security bug unintended functionality that breaks basic security principles.

Avizienis et al. [11] 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 a the error resulting from a weakness can be more abstract than the concrete fault of a next weakness.

Using our security expertise to pose more and more tuned questions to ChatGPT, thus providing more context to it, we got it to partially validate our own def-

[3]e.g., greater than the maximum allowed integer – $2^{32} - 1$ for a 32-bits system.

initions. Excitingly, the collective knowledge confirms we are delving in the right direction.

CONCLUSION

In this paper, define the fundamental notions of security failure and software security vulnerability, weakness, bug, and final error. We also detail the definitions of software fault and error. We also we present our software security venerability model.

We have developed the presented definitions and model iteratively, while creating the Bugs Framework (BF) and the grammar of its taxonomy. They help us reason about and create software security weaknesses taxonomies, allowing precise descriptions of existing software security vulnerabilities [3].

Understanding the role of faults as propagating errors in the chain of weaknesses of a vulnerability, makes it easier to see that the final error is the one that gets exploited. Recall the BadAlloc pattern and 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 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 that CVE-2021-21834 has a buffer overflow weakness.

Irena Bojanova, is a computer scientist at NIST, USA. Her current research interests include cybersecurity and formal methods. She is a Senior member of the IEEE Computer Society. Contact her at irena.bojanova@computer.org.

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] OpenAI, *ChatGPT: Optimizing Language Models for Dialogue*, Accessed: 2023-01-06, 2022. [Online]. Available: https://openai.com/blog/chatgpt/.

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

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

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

[11] 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.