

Labeling Software Security Vulnerabilities

Irena Bojanova, <https://orcid.org/0000-0002-3198-7026>, NIST, Gaithersburg, MD, 20899, USA

John J. Guerrero, <https://orcid.org/0000-0001-6547-9137>, Dartmouth College, Hanover, NH, 03755, USA

Abstract—Labeling software security vulnerabilities would greatly benefit modern artificial intelligence cybersecurity research. The National Vulnerability Database (NVD) partially achieves this via assignment of Common Weakness Enumeration (CWE) entries to Common Vulnerabilities and Exposures (CVE) entries. In this work, we explore utilization of the Bugs Framework (BF) formalism for systematic and comprehensive CVE labeling. We specify all memory-related CWEs via BF weaknesses and examine the suitability of these formalisms to describe the corresponding CVEs mapped by NVD. We also identify the similarities and overlaps in CWEs that introduce ambiguities in NVD assignments.

Keywords: Weakness, Vulnerability, Security Failure, Vulnerability Dataset, Cybersecurity Attack, Cybersecurity Mitigation Techniques.

There are more than 228 000 (as of August 2023) publicly disclosed cybersecurity vulnerabilities in the Common Vulnerabilities and Exposures (CVE) [1] repository – over 25 000 documented in 2022 alone. Systematic labeling of this huge set of software security vulnerabilities would enable advances in modern artificial intelligence (AI) cybersecurity research (e.g., [2]). The current state-of-the-art is the National Vulnerability Database (NVD) [3] mappings of CVEs to Common Weakness Enumeration (CWE) entries and assignment of CVE severity scores according to the Common Vulnerability Scoring System (CVSS) [4]. However, deeper analysis of the CWE entries shows many are overly specific, ambiguous, or overlapping, complicating the CWE-CVE assignment.

The Bugs Framework (BF) [5] formalism assures precise descriptions of software security vulnerabilities

[6] and can be instrumental in gaining a deeper understanding of the CVEs. The existing NVD assignments, although flawed, offer an opportunity to gain high-level insights into the CVEs through the CWEs. Mapping the CWEs to BF and then using the NVD CWE-CVE assignments allows us to take advantage of BF's formal model in the context of the CVEs. This application of BF can capture the primary concepts (e.g., recurrent operations, consequences) expressed by the overwhelming number of CVEs and inform a labeling approach based on machine learning (ML). We can learn for example that thousands of CVEs relate to erroneous read while there are none relating to erroneous reallocate. The CWE-BF mappings also provide formal BF descriptions that partially fit many CVEs, which can aid in their annotation.

BF's formalism allows us to specify each CWE as a (cause, operation, consequence) BF weakness or a chain of BF weaknesses [6]. These specifications reveal the underlying tacit model of the CWEs and allows us to identify similarities and overlaps in the CWE. The latter are part of the issues that introduce ambiguities into the CWEs to CVEs assignments.

This work is our first exploration of utilizing BF-CWE-CVE mappings for specifying/labeling CVEs via BF. We focus on the memory-related CWEs, as there are a vast number of memory-related CVEs corresponding to a relatively small number of memory-

XXXX-XXX © 2023 IEEE

Digital Object Identifier 10.1109/XXX.0000.00000000

Date of current version xx July 2023.

Disclaimer: Certain equipment, instruments, software, or materials, commercial or non-commercial, are identified in this paper in order to specify the experimental procedure adequately. Such identification is not intended to imply recommendation or endorsement of any product or service by NIST, nor is it intended to imply that the materials or equipment identified are necessarily the best available for the purpose.

related CWEs. We specify them via BF weaknesses and examine the suitability of these formalisms to describe corresponding CVEs. We also discuss identified similarities and overlaps among the CWEs, elucidating the NVD's and the security community's struggles with assigning CWEs to CVEs.

Formalizing Memory-Related CWEs via BF Weaknesses or Chains of BF Weaknesses

The CWE and the BF both classify weaknesses; however, as a formal model, BF has the ability to make the tacit CWE model of memory-related software weaknesses explicit. We employ the BF formal representation of weaknesses as (cause, operation, consequence) triples, the BF vulnerability model [6], and the BF Memory Bugs taxonomy [7]. Specifically, we use the Memory Corruption/Disclosure (_MEM) BF class type, consisting of three BF classes: the Memory Addressing (MAD) class deals with initialization, repositioning, or reassignment of pointers; the Memory Management (MMN) class – erroneous allocation, deallocation, resizing, or reallocation of an object; and the Memory Use (MUS) class deals with the initialization, reading, writing, and clearing an object.

The CWEs' web pages mostly consist of paragraph descriptions (i.e., the description, extended description, explanations of code snippets, etc.). Many distinct phrases across CWEs have nearly identical meanings, the paragraphs themselves could be interpreted differently, and there is no guarantee a description has all the necessary information to fully understand the corresponding weakness. As an LL1 grammar [6], BF inherently lacks ambiguity, and its cause-operation-consequence structure means that it is guaranteed to fully describe a software weakness or a vulnerability. Thus, BF has the potential to clarify and supplement the CWE entries.

We identified 60 memory-related CWEs used for labeling CVEs, starting from the CWE to BF class mappings introduced in [7]. We updated these mappings by BF class operation from [7] and created a (cause, operation, consequence)-based BF description for each of these CWEs by deeply analyzing each CWE description, extended description, common consequences, demonstrative code examples, and observed CVE examples.

In total, the memory-related CWEs map to 48 distinct BF chains formed by the identified BF (bug, operation, error) and/or (fault, operation, error) weakness triples [6]. 37 of these BF chains map to a single CWE, and 11 map

to more than one CWE. The 60 BF-CWE mappings are depicted across Table 1, Table 2, and Table 3. Table 1 shows CWEs with entirely different BF descriptions. Table 2 and Table 3 show CWEs that have similarities. More specifically, Table 2 shows groups of CWEs that have the same causing chain and Table 3 shows groups of CWEs that have identical BF descriptions. Note that we omit the BF Class names when describing the weakness triples, as BF classes are orthogonal by operation. For example, for CWE-562 (Erroneous Code, Reassign, Wild Pointer) is the same as (Erroneous Code, MAD Reassign, Wild Pointer), where MAD stands for BF Memory Addressing Bugs Class.

For CWE-823, CWE-400, and CWE-1325 we were able to identify multiple distinct BF chains (shown bulleted in Table 1). Each CWE entry is supposed to describe only a single weakness; the existence of CWEs that describe multiple distinct weaknesses highlights overlaps within the CWE.

Many CWEs provide information about preceding weaknesses that we formalize into BF chains of weaknesses. As a result, some BF specifications contain causing weakness operations from non-memory-related BF classes – see for example Verify and Calculate for CWE-823, and Coerce for CWE-843 in Table 1. The first operation is an operation from the Input/Output Check BF Class Type (_INP) and the second two form the Data Type BF Class Type (_DAT) [5]. For example, CWE-126 ostensibly describes a buffer over-read weakness. However, it also describes an erroneous calculation leading to an incorrect pointer reposition. These three weaknesses form the (Erroneous Code, Calculate, Wrong Result) → (Wrong Index, Reposition, Over Bounds Pointer) → (Over Bounds Pointer, Read, Buffer Over-Read) BF chain.

Memory-Related CWEs Similarity and Overlaps via BF Specifications

There are three groups of memory-related CWEs (see Table 2) that share one or more of the same BF weakness triples causing a different BF weakness triple. For example, CWE-127, CWE-786, and CWE-124 all begin with (Erroneous Code, Calculate, Wrong Result), which causes (Wrong Index, Reposition, Under Bounds Pointer). This same causal chain leads to three different main BF weakness triples for CWE-127, CWE-786, and CWE-124, which distinguish the three entries.

There are also 11 groups of memory-related CWEs (see Table 3) with the same BF weakness or chain of

TABLE 1: Memory-Related CWEs with Different BF Weaknesses/Chains of Weaknesses (order is by BF Memory Bugs Model operation flow [7]); causing weaknesses are in *italics*; arrows (→) depict chaining; asterisks (*) mark CWEs not assigned to any CVEs).

CWE ID	BF (cause, operation, consequence) Triple(s)
655	(Missing Code/Erroneous Code, <i>Initialize Object</i> , Uninitialized Object)
466	(n/a, <i>Initialize Pointer/Reposition</i> , Over Bounds Pointer/Under Bounds Pointer)
587	(Hard Coded Address, <i>Initialize/Reassign</i> , Wild Pointer)
823	<ul style="list-style-type: none"> • (<i>Missing Code</i>, <i>Verify</i>, <i>Wrong Value</i>) → → (Wrong Index, <i>Reposition</i>, Over Bounds Pointer) • (<i>Erroneous Code</i>, <i>Calculate</i>, <i>Wrong Result</i>) → → (Wrong Index, <i>Reposition</i>, Over Bounds Pointer)
562	(Erroneous Code, <i>Reassign</i> , Wild Pointer)
1325	<ul style="list-style-type: none"> • (<i>Erroneous Code</i>, <i>Allocate</i>, <i>Memory Overflow</i>) • (<i>Erroneous Code</i>, ..., <i>Not Enough Memory</i>) → → (Wrong Size, <i>Allocate</i>, Memory Overflow)
400	<ul style="list-style-type: none"> • (<i>Missing Code</i>, <i>Verify</i>, <i>Wrong Value</i>) → → (Wrong Size, <i>Allocate</i>, Memory Overflow) • (<i>SingleOwnedAddress</i>, <i>Reassign</i>, <i>Memory Leak</i>) • (<i>Missing Code</i>, <i>Deallocate</i>, <i>Memory Leak</i>)
822	(Untrusted Pointer, <i>Dereference</i> , Untrusted Pointer Dereference)
690	<ul style="list-style-type: none"> • (<i>Missing Code</i>, <i>Verify</i>, <i>Wrong Value</i>) → → (Forbidden Address, <i>Dereference</i>, NULL Pointer Dereference)
476	(NULL Pointer, <i>Dereference</i> , NULL Pointer Dereference)
824	<ul style="list-style-type: none"> • (<i>Missing Code</i>, <i>Initialize Pointer</i>, <i>Wild Pointer</i>) → → (Wild Pointer, <i>Read/Write/Dereference</i>, Uninitialized Pointer Dereference)
588	<ul style="list-style-type: none"> • (<i>Erroneous Code</i>, <i>Cast</i>, <i>Wrong Type</i>) → → (Casted Pointer, <i>Read/Write/Dereference</i>, Type Confusion)
672	(Erroneous Code, <i>Read/Write/Dereference</i> , Use After Free)
843	<ul style="list-style-type: none"> • (<i>Wrong Object Type Resolved</i>, <i>Coerce</i>, <i>Wrong Type</i>) → → (Casted Pointer, <i>Read/Write/Dereference</i>, Type Confusion)
119	<ul style="list-style-type: none"> • (<i>Missing Code</i>, <i>Verify</i>, <i>Wrong Value</i>) → → (Wrong Index, <i>Reassign</i>, Over Bounds Pointer/Under Bounds Pointer) → → (Over Bounds Pointer/Under Bounds Pointer, <i>Read/Write</i>, Buffer Overflow/Buffer Underflow/Buffer Over-Read/Buffer Under-Read)
118	<ul style="list-style-type: none"> • (<i>Missing Code/Erroneous Code</i>, <i>Verify</i>, <i>Wrong Value</i>) → → (Wrong Index, <i>Reassign</i>, Over Bounds Pointer/Under Bounds Pointer) → → (Over Bounds Pointer/Under Bounds Pointer, <i>Read/Write</i>, Buffer Overflow/Buffer Underflow/Buffer Over-Read/Buffer Under-Read)
120	<ul style="list-style-type: none"> • (<i>Missing Code</i>, <i>Verify</i>, <i>Wrong Value</i>) → → (Wrong Size, <i>Allocate</i>, <i>Not Enough Memory Allocated</i>) → → (Not Enough Memory Allocated, <i>Write</i>, Buffer Overflow)
680	<ul style="list-style-type: none"> • (<i>Erroneous Code</i>, <i>Calculate</i>, <i>Wrap Around</i>) → → (Wrong Size, <i>Allocate</i>, <i>Not Enough Memory</i>) → → (Not Enough Memory, <i>Write</i>, Buffer Overflow)
459	(Erroneous Code, <i>Clear</i> , Not Cleared Object)
404	(Missing Code/Erroneous Code, <i>Deallocate</i> , Memory Leak/Object Corruption)
761	<ul style="list-style-type: none"> • (<i>Wrong Index</i>, <i>Reposition</i>, <i>Wrong Position Pointer</i>) → → (Wrong Position Pointer, <i>Deallocate</i>, Object Corruption)
772	(Missing Code, <i>Deallocate</i> , Memory Overflow)
1091	(Missing Code, <i>Deallocate</i> , None)
460	(Missing Code/Erroneous Code, <i>Deallocate</i> , Memory Leak)
586*	(Erroneous Code, <i>Deallocate</i> , None)
568*	(Missing Code, <i>Deallocate</i> , None)

TABLE 2: Memory-Related CWEs with the Same Causing BF Chains (order is by BF Memory Bugs Model operation flow [7]); causing weaknesses are in italics; arrows (→) depict chaining).

CWE ID	Same Causing Chain	Different Main Weakness
126	<i>(Erroneous Code, Calculate, Wrong Result) → → (Wrong Index, Reposition, Over Bounds Pointer)</i>	(Over Bounds Pointer, Read , Buffer Over-Read)
788		(Over Bounds Pointer, Read/Write , Buffer Overflow/Buffer Over-Read)
127	<i>(Erroneous Code, Calculate, Wrong Result) → → (Wrong Index, Reposition, Under Bounds Pointer)</i>	(Under Bounds Pointer, Read , Buffer Under-Read)
786		(Under Bounds Pointer, Read/Write , Buffer Underflow/Buffer Under-Read)
124		(Under Bounds Pointer, Write , Buffer Underflow)
125	<i>(Erroneous Code, Calculate, Wrong Result) → → (Wrong Index, Reposition, Over Bounds Pointer/Under Bounds Pointer)</i>	(Over Bounds Pointer/Under Bounds Pointer, Read , Buffer Over-Read/Buffer Under-Read)
787		(Over Bounds Pointer/Under Bounds Pointer, Write , Buffer Overflow/Buffer Underflow)

weaknesses, suggesting areas of repetition or overlap within the CWE.

For example, CWE-401 describes a memory leak due to an allocated object with no references, while CWE-771 describes erroneous removal of all references to an allocated object. However, these weaknesses are quite similar, as the erroneous removal of all references to an allocated object in CWE-771 causes the memory leak described by CWE-401, and the state described by CWE-401 is created by the re-assign operations described by CWE-771. In BF terms, these CWE entries are both specified as *(Single Owned Address, Reassign, Memory Leak)*.

Other CWEs have slight differences only by BF attributes, which inform about the severity of the weakness. For example, CWE-121 describes buffer overflow on the stack and CWE-122 describes buffer overflow on the heap. Once the stack vs. heap difference is accounted for by a BF attribute, these two entries are specified by the same instance of a BF weakness type: *(Over Bounds Pointer/Under Bounds Pointer, Write, Buffer Overflow/Buffer Underflow)*.

The CWE hierarchical relationships between CWEs with the same BF chain (see third column in Table 3), reveals that most of them have either ChildOf or PeerOf relations. Some of them, such as CWEs: 121-122-123 and CWEs: 170-463-464, are also siblings with common parent. However, there are also instances of CWEs without direct relationships that have same chains, such as CWE-401 and CWE-771. An inter-

esting observation is that while CWE-123, CWE-415, CWE-416 are specified with very different BF weakness triples, they are listed as peers in the CWE.

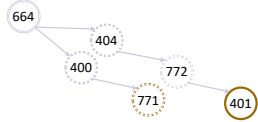
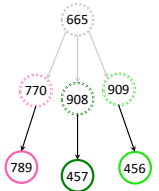
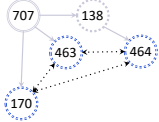
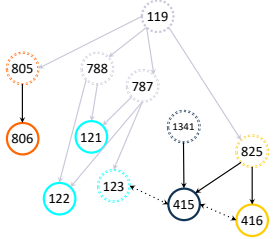
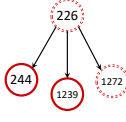
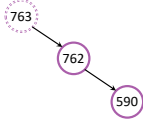
Of special note are parent-child CWE pairs that share a BF chain. Per the CWE, the parent entry is supposed to be more abstract than the child entry. In BF terms, this is expressed by having multiple possible causes, consequences, and/or operations (e.g., *Buffer Overflow/Buffer Underflow* vs. only *Buffer Overflow*). One would expect that parent/child CWEs would have slightly different BF chains, or the chain for the child would be contained within the chain for the parent. For example, *(Over Bounds Pointer, MUS Read, Buffer Over-Read)* is contained within *(Over Bounds Pointer/Under Bounds Pointer, MUS Read, Buffer Over-Read/Buffer Under-Read)*. The fact that a parent and child have the same chain means this difference in ambiguity is missing and highlights an area of overlap within the CWE.

BF also reveals missing relationships within the CWE. CWEs that have differences only by BF attributes (e.g., CWE-121 and CWE-122) should have some relationship (e.g., PeerOf) within the CWE.

Labeling Memory-Related CVEs via BF Specifications

There are 60 426 memory-related CVEs as of August 2023. We queried the CVE repository for entries with CVEs assigned by NVD that map by operation to the

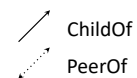
TABLE 3: Memory-Related CWEs with the Same BF Weakness/Chain of BF Weaknesses (order is by operation flow in BF Memory Bugs Model [7]); listing the operation is sufficient, as BF classes are orthogonal by operation; causing weaknesses are in *italics*; arrows (→) depict chaining; asterisks (*) mark CWEs not assigned to any CVEs; ellipsis (...) depicts many possible operations and consequences).

CWE ID	Same BF (cause, operation, consequence) Triple(s)	CWE Relationships
401 771	(Single Owned Address, Reassign , Memory Leak)	
770 789 456 909 457 908	(<i>Missing Code</i> , Verify , <i>Wrong Value</i>) → → (Wrong Size, Allocate , Memory Overflow) (Missing Code, Initialize Object , Uninitialized Object) (<i>Missing Code</i> , Initialize Object , <i>Uninitialized Object</i>) → → (Uninitialized Object, ..., ...)	
170 463* 464*	(Erroneous Code, Write , Object Corruption)	
805 806* 121 122 123 416 825 415 1341*	(Wrong Size, Read/Write , Buffer Overflow/Buffer Underflow/Buffer Over-Read/Buffer Under-Read) (Over Bounds Pointer/Under Bounds Pointer, Write , Buffer Overflow/Buffer Underflow) (Dangling Pointer, Read/Write/Dereference , Use After Free) (Erroneous Code, Deallocate , Double Free)	
226 244* 1239* 1272*	(Missing Code, Clear , Not Cleared Object)	
590 762 763	(Mismatched Operation, Deallocate , Object Corruption)	

CWEs by Abstraction:



CWEs Relation:



BF Memory Corruption/Disclosure (`_MEM`) class type. We ordered them by the NVD-assigned CVSS severity scores and selected a maximum of ten CVEs per operation – thus reducing the count to 91 observable CVEs for this exploratory analysis. Then we analyze the groups of CVEs mapped to CWEs with same causing BF chains and of CVEs mapped to CWEs with identical BF weaknesses or chains of BF weaknesses. From the latter group we also identified the CVEs mapped to CWEs with parent-child relationships.

Analyzing this subset of CVEs, we find that it covers well the BF memory operations Reposition, Reassign, Verify, Initialize Object, Read, Write, Dereference, Clear, and Deallocate. However, although there may be CVEs related to the BF memory operations Initialize Pointer, Extend, Reallocate-Extend, Reduce, and Reallocate-Reduce, they are not identifiable via CWEs in the entire CVE. This indicates gaps in CWEs or issues with the CWE assignments. In any case, we would need different methods to identify and specify CVEs related to these operations.

Examining further the subset of 91 CVEs, we confirm that, overall the CWE to CVE assignments are almost completely correct by BF operation. For example, the CVEs mapped to CWE-126 and CWE-788 (see Table 2) correctly distinguish between the read only and read/write operations, respectively. We only sporadically find examples of wrong CWE to CVE assignments by operation, such as CWE-123 to CVE-2018-12036. The confusion for this CVE must relate to the use of "writes" in its description, while in fact it is a BF Input/Output Check (`_INP`) class type vulnerability: (`Missing Code, Validate, File Injection`). However, when examining the CVEs by the CWEs BF weaknesses of chains of weaknesses, which cover not only operations but also causes and consequences, we find parts of these BF weakness specifications may not fit all of the corresponding CVEs. For example, the BF CWE-126 chain (see Table 2) completely fits the (`Over Bounds Pointer, Read, Buffer Over-Read`) main weakness of CVEs such as CVE-2014-0160 (Heartbleed), as well as their (`Wrong Index, Reposition, Over Bounds Pointer`) direct causing weakness. However, most of the CVEs with CWE-126 assigned, have as an initial weakness `Missing Code` for a `Verify` operation and not `Erroneous Code` in a `Calculate` operation – see the first weakness in [8]. We conclude the causal chains from Table 2 may be helpful for describing some CVEs but are too specific to fit other CVEs. This can be explained with CWE's lack of flexibility to describe all possible security weaknesses

– the entries could be too specific to be reused, some may be missing, and some may be overlapping. This indicates we will have to use entirely different methods to identify and specify the parts that do not fit the too specific CWE variation.

We find also that from the CWEs with different BF specifications (see Table 1) there are no assignments to any CVE at all of CWE-586 and CWE-568. Then, from the CWE groups with identical BF specifications (see Table 3) there are no assignments to any CVE at all of the second CWEs in the 805, 806 pair and in the 415, 1341 pair; the last two CWEs in the 170, 463, 464 triple, and the last three CWEs in the 226, 224, 1239, 1272 quadruple. It is interesting to explore if there are CVEs that are better described by the unused CWEs (marked with an asterisks * in Table 1 and Table 3). One such example is CVE-2023-38434 (although outside of our 91 CVEs set) that describes a double free when closing a web connection. NVD assigns it CWE-415 (Double Free) of a memory resource, instead of CWE-1341 – the more general (Multiple Releases of Same Resource or Handle). These two CWEs share the (`Erroneous Code, Deallocate, Double Free`) BF weakness triple (see Table 3); their similarity and the vague memory vs. resource distinction introduces errors and ambiguities in CWE-CVE assignments that would also affect our efforts for CVE labeling.

The rest of the similar CWEs (see Table 3) lead in many cases to ambiguous CWE-CVE assignments. One such example is CVE-2022-0519, which describes a buffer access with an incorrect length value leading to a buffer over-read. NVD assigns this CVE to CWE-119 (Improper Restriction of Operations within the Bounds of a Memory Buffer) and CWE-805 (Buffer Access with Incorrect Length Value). However, CWE-126 (Buffer Over-Read) also nicely and more accurately describes this CVE than the abstract CWE-119.

Another area of ambiguity among CWEs relates to BF attributes. For example, CVE-2023-40295 describes a buffer overflow on the heap. NVD assigns CWE-787 (Out of Bounds Write) to this CVE, which accurately describes the vulnerability. However, CWE-122 is also a suitable assignment, as it describes buffer overflow specifically on the heap. Apart from this minor difference (captured in BF by an address-related attribute), CWE-122 and CWE-787 share an identical main weakness: (`Over Bounds Pointer/Under Bounds Pointer, Write, Buffer Overflow, Buffer Underflow`). While CWE-122 is the more specific mapping, the similarities create difficulty in deciding which CWE should be assigned to this CVE.

In many cases, CWE-CVE assignments capture

either the cause of a vulnerability or its consequence but not both. CVE-2022-34399 and CVE 2022-32454 describe a buffer over-read and a buffer over-write vulnerability, respectively. The BF chain for CVE-2022-34399 is (Missing Code, DVR Verify, Inconsistent Value) → (Wrong Index, MAD Reposition, Over Bounds Pointer) → (Over Bounds Pointer, MUS Write, Buffer Overflow) and the BF chain for CVE-2022-32454 is (Missing Code, Verify, Wrong Value) → (Wrong Size, MUS Read, Buffer Over-Read). NVD assigns CWE-119 (Improper Restriction of Operations within the Bounds of a Memory Buffer) and CWE-805 (Buffer Access with Incorrect Length Value) to CVE 2022-34399 and CWE-121 (Stack-based Buffer Overflow) to CVE 2022-32454. However, the CWEs assigned to CVE 2022-34399 only describe the cause of the vulnerability, and the CWE assigned to CVE 2022-32454 only describes its consequence. One must assign CWE-126 (Buffer Over-Read) to CVE 2022-34399 and CWE-112 (Missing XML Validation) to capture the full story of each weakness underlying the vulnerability. This inconsistency in capturing either the cause or the consequence of a vulnerability further complicates the CWE-CVE assignment.

The findings from this work show areas that would require additional analysis to create precise BF CVE descriptions. We would have to examine deeply corresponding vulnerability reports, source code with the bugs, source code with fixes, and other available related sources. Utilizing the BF vulnerability model, the BF cybersecurity concepts definitions and BF classes taxons definitions [6] and [5], as well as any of their synonyms in use, we can employ modern ML and AI approaches towards automatic CVEs analysis and generation of BF CVE specifications.

Conclusion

Labeled vulnerability descriptions are of great demand for advanced AI research related to cybersecurity vulnerabilities, attacks, and mitigation techniques. As a formal bugs/weaknesses model [6], BF has the ability to unambiguously describe the chains of underlying weaknesses for any software security vulnerability.

With this work we begin specifying the detailed information provided for each CWE. We use the CWEs as a bridge to the corresponding CVEs and explore to what extent the BF CVE descriptions may aid manual creation of BF CVE descriptions. We invite you to collaborate with us in this direction by joining the BF-CVE specification challenge at [5].

Our future goal is to employ ML and AI approaches for automated generation of BF CVE descriptions. The result would be a reference dataset of labeled vulnerability specifications for use by AI algorithms. The labels will constitute the exhaustive sets of causes, operations, consequences, and attributes values, precisely defined as BF class taxonomies. The BF CVE reference dataset will be a great source not only for research but also for cybersecurity education and guidance.

Irena Bojanova, is a computer scientist at the National Institute of Standards and Technology (NIST), USA. She is the primary investigator and lead of the 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.

John Guerrero is a sophomore at Dartmouth College and a Summer Undergraduate Research Fellowship (SURF) student at NIST. He is majoring in Computer Science and Mathematical Data Science. Contact him at john.j.guerrero.26@dartmouth.edu.

References

- [1] MITRE, *Metrics*, Accessed: 2023-07-07, 2023. [Online]. Available: <https://www.cve.org/About/Metrics>.
- [2] 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. DOI: 10.1109/CyberSecurity49315.2020.9138852.
- [3] NVD, *National Vulnerability Database (NVD)*, Accessed: 2023-03-02, 2023. [Online]. Available: <https://nvd.nist.gov>.
- [4] FIRST, *Common vulnerability scoring system special interest group*, Accessed: 2023-07-07, 2023. [Online]. Available: <https://www.first.org/cvss>.
- [5] I. Bojanova, NIST, *The Bugs Framework (BF)*, Accessed: 2023-02-14, 2023. [Online]. Available: <https://samate.nist.gov/BF/>.
- [6] I. Bojanova and C. E. Galhardo, "Bug, fault, error, or weakness: Demystifying software security vulnerabilities," *IT Professional*, vol. 25, no. 1, pp. 7–12, Jan. 2023. DOI: 10.1109/MITP.2023.3238631.
- [7] I. Bojanova and C. E. Galhardo, "Classifying Memory Bugs Using Bugs Framework Approach," in *2021 IEEE 45th Annu. Computer, Software, and*

- Applications Conf.*, IEEE, vol. 1, 2021, pp. 1157–1164. DOI: [10.1109/compsac51774.2021.00159](https://doi.org/10.1109/compsac51774.2021.00159).
- [8] I. Bojanova and C. E. Galhardo, “Heartbleed revisited: Is it just a buffer over-read?” *IT Professional*, vol. 25, no. 2, pp. 83–89, Mar. 2023. DOI: [10.1109/MITP.2023.3259119](https://doi.org/10.1109/MITP.2023.3259119).