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

DEPARTMENT: CYBERSECURITY

Heartbleed Revisited: Is it just a Buffer Over-Read?

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

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

Abstract—In this work, we analyse in detail the weaknesses underlying the Heartbleed vulnerability, CVE-2014-0160, and show why and how highly sensitive information could be exposed via buffer over-read.

Heartbleed was a serious vulnerability in the popular OpenSSL cryptographic software library [1]. The fatal bug was in the Heartbeat Extension of the TLS (Transport Layer Security) protocol implementation. The vulnerability was disclosed in April 2014 with the following Common Vulnerabilities and Exposures (CVE) [2] entry.

CVE-2014-0160: "The (1) TLS and (2) DTLS implementations in OpenSSL 1.0.1 before 1.0.1g do not properly handle Heartbeat Extension packets, which allows remote attackers to obtain sensitive information from process memory via crafted packets that trigger a buffer over-read, as demonstrated by reading private keys, related to d1_both.c and t1_lib.c, aka the Heartbleed bug." [3]

Attacks exploiting Heartbleed, can reveal highly sensitive information – such as private keys, user login credentials, and business or personal information – via reads over the buffer bounds (i.e., buffer over-reads). Heartbleed's name literally means a server (or a client) with a vulnerable Heartbeat Extension can "bleed" data via heartbeat response messages. A small heartbeat message with a large requested length can reveal up to 64KB raw memory, and multiple requests can accumulate huge amounts of data. [4]

For the lack of a more precise Common Weakness Enumeration (CWE) [5] entry, the National Vulnerability Database (NVD) [6]) assigns to CVE-2014-0160 [7] CWE-119 "Improper Restriction of Operations within the Bounds of a Memory Buffer" [8], which covers both underbounds and overbounds, and both reads and writes to or from a buffer. However, we can describe it precisely utilizing the National Institute of Standards and Technology (NIST) Bugs Framework (BF) [9]. We will use the BF Data Verification (DVR), Memory Addressing (MAD), and Memory Use (MUS) classes [9] to describe Heartbleed at a level of detail allowing clear understanding of its underlying bugs, chains of weaknesses, and exploitable errors.

In this article, we analyze the Heartbleed code (see Figure 1a) and clearly describe how Heartbleed leads to buffer over-read. We also examine what else is needed for such buffer over-reads to cause exposure of sensitive information.

THE BUGS FRAMEWORK (BF)

We are developing the NIST Bugs Framework (BF) [9] as a structured, complete, orthogonal classification system of software security bugs and weaknesses, which is also language and domain independent. BF consists of a set of bug/weakness class taxonomies and a formal language for describing software security vulnerabilities.

A BF class is a taxonomic category of a weakness type, defined by a set of operations – where such bugs could happen, a set of causes – the possible improper operations (bugs) and improper operands (faults), a set of consequences – the possible errors (that become faults for next weaknesses) and the possible final errors (that become exploits), a matrix of valid cause→operation→consequence relations, and a set of attributes for the operations and the operands.

The BF formal language adheres to our BF vulnera-

Digital Object Identifier 10.1109/XXX.0000.0000000
Date of current version xx March 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.

```
1448 dtls1_process_heartbeat(SSL *s)
1449   {
1450   unsigned char *p = &s->s3->rrec.data[0], *pl;
1451   unsigned short hbtype;
1452   unsigned int payload;
1453   unsigned int padding = 16; /* Use minimum padding */
1454
1455   /* Read type and payload length first */
1456   hbtype = *p++;
1457   n2s(p, payload);
1458   pl = p;
...
1465   if (hbtype == TLS1_HB_REQUEST)
1466     {
1467     unsigned char *buffer, *bp;
...
1470     /* Allocate memory for the response, size is 1 byte
1471     * message type, plus 2 bytes payload, plus
1472     * payload, plus padding
1473     */
1474     buffer = OPENSSL_malloc(1 + 2 + payload + padding);
1475     bp = buffer;
1476
1477     /* Enter response type, length and copy payload */
1478     *bp++ = TLS1_HB_RESPONSE;
1479     s2n(payload, bp);
1480     memcpy(bp, pl, payload);
```

(a) The Heartbeat buggy C code in `ssl\d1_both.c` [10].

```
/* Naive implementation of memcpy
void *memcpy (void *dst, const void *src, size_t n)
{
    size_t i;          payload
    for (i=0; i<n; i++)
        *(char *) dst++ = *(char *) src++;
    return dst;
}                      bp              pl
```

(b) A naive C implementation of the `memcpy()` function.

FIGURE 1: Analysis of Heartbleed.

bility model, according to which a vulnerability description is a chain of underlying weaknesses leading to a security failure. [11]

In this work, we describe the Heartbleed vulnerability utilizing the taxonomies of the BF DVR [12], MAD [13], and MUS [13] classes[1]. DVR relates to how input data semantics is checked or corrected. MAD relates to how an object pointer is initialized, repositioned, or reassigned. MUS relates to how an object is initialized, read, written, or cleared.

[1]The most current BF taxonomies are accessible from BF website [9].

HEARTBLEED ANALYSIS

A vulnerability can be described precisely as a sequence of improper states of software execution phases. An improper state is represented by an $(\texttt{operation}, \texttt{operand}_1, \cdots, \texttt{operand}_n)$ tuple, where at least one element is improper. The error from the last state is the exploitable one. In some cases, more than one vulnerability may need to be present for an exploit to be harmful. [11]

The buggy OpenSSL Heartbeat [10] C code that caused Heartbleed and wreaked havoc across the world was both in the `d1_both.c` and `t1_lib.c` files. Figure 1a presents its first occurrence. Let's examine in detail and determine the improper states.

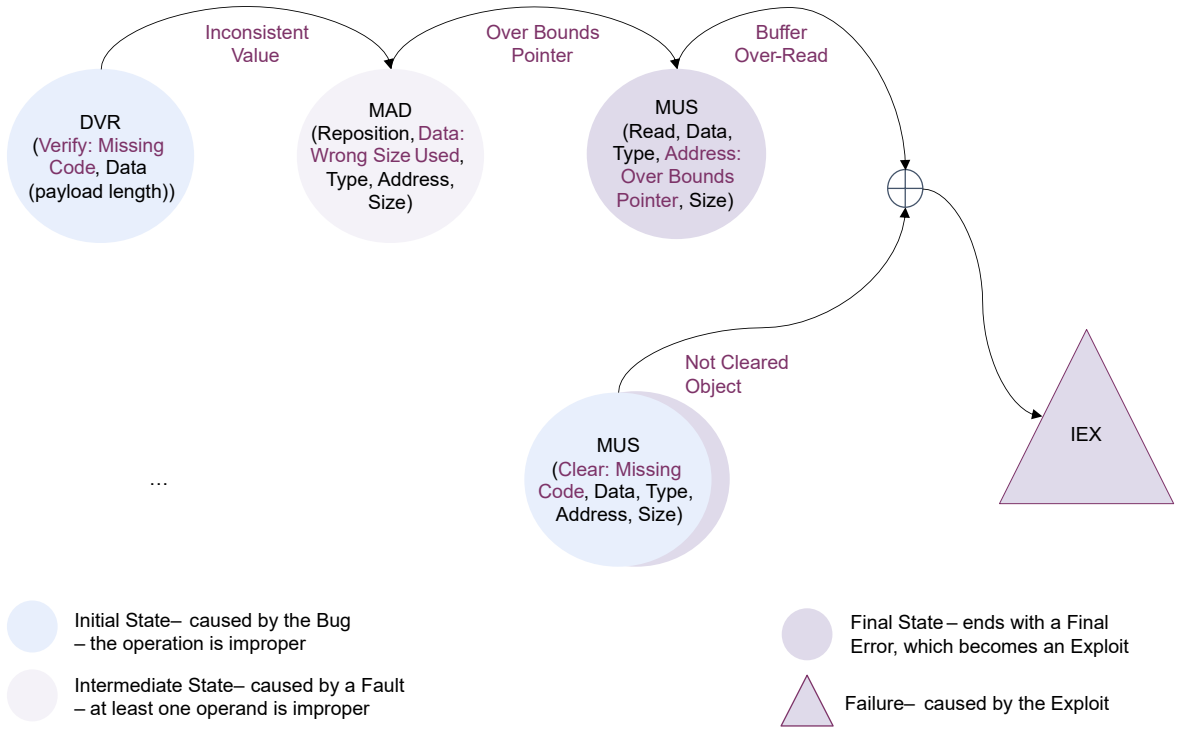


FIGURE 2: Heartbleed's improper states. Two software security vulnerabilities converge at their Buffer Over-Read and Not Cleared Object final errors to cause an Information Exposure (IEX) security failure.

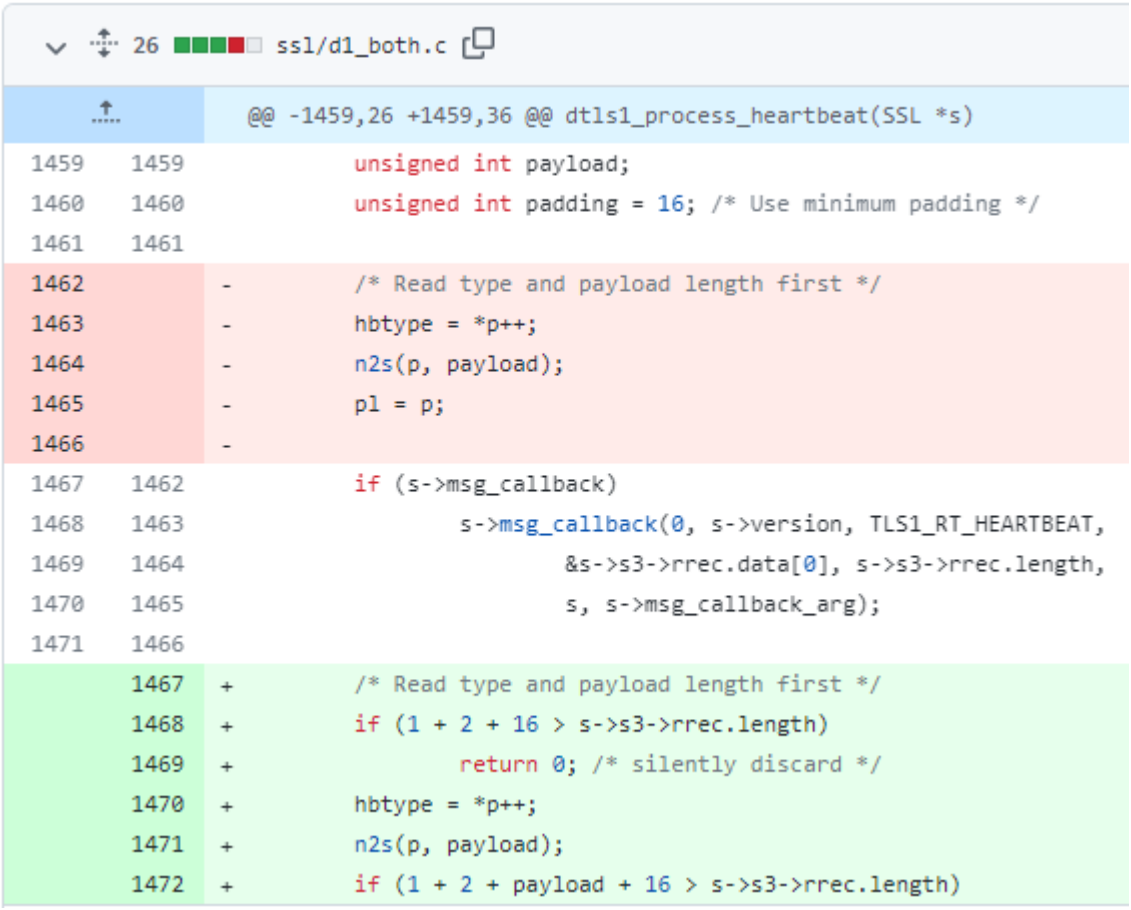
The very first problem is in the data verification phase, where the semantics of the input should be checked and corrected. The variable `payload` is declared as an unsigned integer and can be a huge number (see line `1452`, Figure 1a). It is input data, that hold the payload length, but that length is not checked towards an upper limit (see line `1457`, Figure 1a). Its value is not verified! This improper state is an instance of the BF Data Verification (DVR) class [12], represented by the (`Verify`, `Data`) tuple, where the first element is improper. The operation Verify is missing (see the first state in Figure 2).

Then, `memcpy` reads `payload` number of bytes from the object pointed by `pl` and copies them to the object pointed by `bp` (see line `1480`, Figure 1a). Following the naive C implementation of `memcpy` in Figure 1b, `bp` and `pl` are passed by reference via `dst` and `src`, and the huge payload length is passed via the argument `n`. First, one byte is read from `pl` and copied to `bp`; then, until the huge payload length is reached, both pointers move one byte up, and the newly pointed by `pl` byte is read and copied. However, while `bp` is allocated large enough (see line `1475`, Figure 1a), `pl` points to an array with reasonable size (see line `1458`, Figure 1a). As the content of this array is read and copied to `bp`, so, too, is a huge amount of data from over its bounds.

There are two improper states here: when `pl` gets repositioned over its upper bound, and when data are read from there. The former is an instance of the BF Memory Addressing (MAD) class [13], represented by the (`Reposition`, `Data`,`Type`, `Address`, `Size`) tuple, where the second element is improper (see the second state in Figure 2). There is no bug in the repositioning itself, however, a value that is inconsistent with the size of the `pl` object is used. The latter is an instance of the BF Memory Use (MUS) class [13], represented by the (`Read`, `Data`, `Type`, `Address`, `Size`) tuple, where the fourth element is improper (see the third state in Figure 2). Again, there is no bug in the Read operation itself, but, because `pl` points over bounds, the software reads data that should not be read – aka buffer over-read.

This chain of three BF states (see the upper row in Figure 2) shows there is read over the buffer bounds, however, it does not show how an exploit could reach sensitive data, such as private keys. The vulnerability triggered by the missing size verification bug is not enough to get access to sensitive information.

To describe the bug in this parallel vulnerability, we use again the BF MUS class [13]. The improper state



26 ssl/d1_both.c

@@ -1459,26 +1459,36 @@ dtls1_process_heartbeat(SSL *s)

```
1459  1459          unsigned int payload;
1460  1460          unsigned int padding = 16; /* Use minimum padding */
1461  1461
1462        -       /* Read type and payload length first */
1463        -       hbtype = *p++;
1464        -       n2s(p, payload);
1465        -       pl = p;
1466        -
1467  1462          if (s->msg_callback)
1468  1463                  s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
1469  1464                          &s->s3->rrec.data[0], s->s3->rrec.length,
1470  1465                          s, s->msg_callback_arg);
1471  1466
      1467  +       /* Read type and payload length first */
      1468  +       if (1 + 2 + 16 > s->s3->rrec.length)
      1469  +               return 0; /* silently discard */
      1470  +       hbtype = *p++;
      1471  +       n2s(p, payload);
      1472  +       if (1 + 2 + payload + 16 > s->s3->rrec.length)
```

FIGURE 3: The Heartbleed fix in Heartbeat – see the C code in lines `1468` and `1469` in the green area [14].

is represented by the (`Clear`, `Data`, `Type`, `Address`, `Size`) tuple, where the first element is improper, but this time the operation Clear is missing (see the second chain in Figure 2). Converging the final errors from both chains – buffer over-read and not cleared object – the buggy software can now reach and expose sensitive information.

The fix of the bug in the main Heartbleed chain is as trivial as the bug itself (see the C code in Figure 3). [14]. An input data value check is added, so that, if the requested and the actual lengths (sizes) are different (see line `1468`, Figure 3), the heartbeat message is silently ignored (see line `1469`, Figure 3).

Our code analysis reveals the causal relationships between the weaknesses underlying Heartbleed. Missing input data verification leads to use of inconsistent size for a buffer, allowing a pointer reposition over bounds, which converging with missing clear, allows remote reads of sensitive information and its exposure.

BF DESCRIPTION OF HEARTBLEED

A BF vulnerability description [11] 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. It is represented by a `(bug, operation, error)` or `(fault, operation, error)` triple, which elements adhere to the BF class taxonomies [9]. The error is the result of an operation over its operands. It establishes the transition to another weakness or a failure. A BF vulnerability description also uses attributes to explain what, how, and where went wrong. They help us understand the severity of the bug or the fault causing the weakness.

Our precise BF description of Heartbleed is depicted in Figure 4 (machine readable versions are accessible from the BF website [9]). Using the BF taxonomy of the involved weaknesses, first is the Data Verification (DVR) weakness, represented

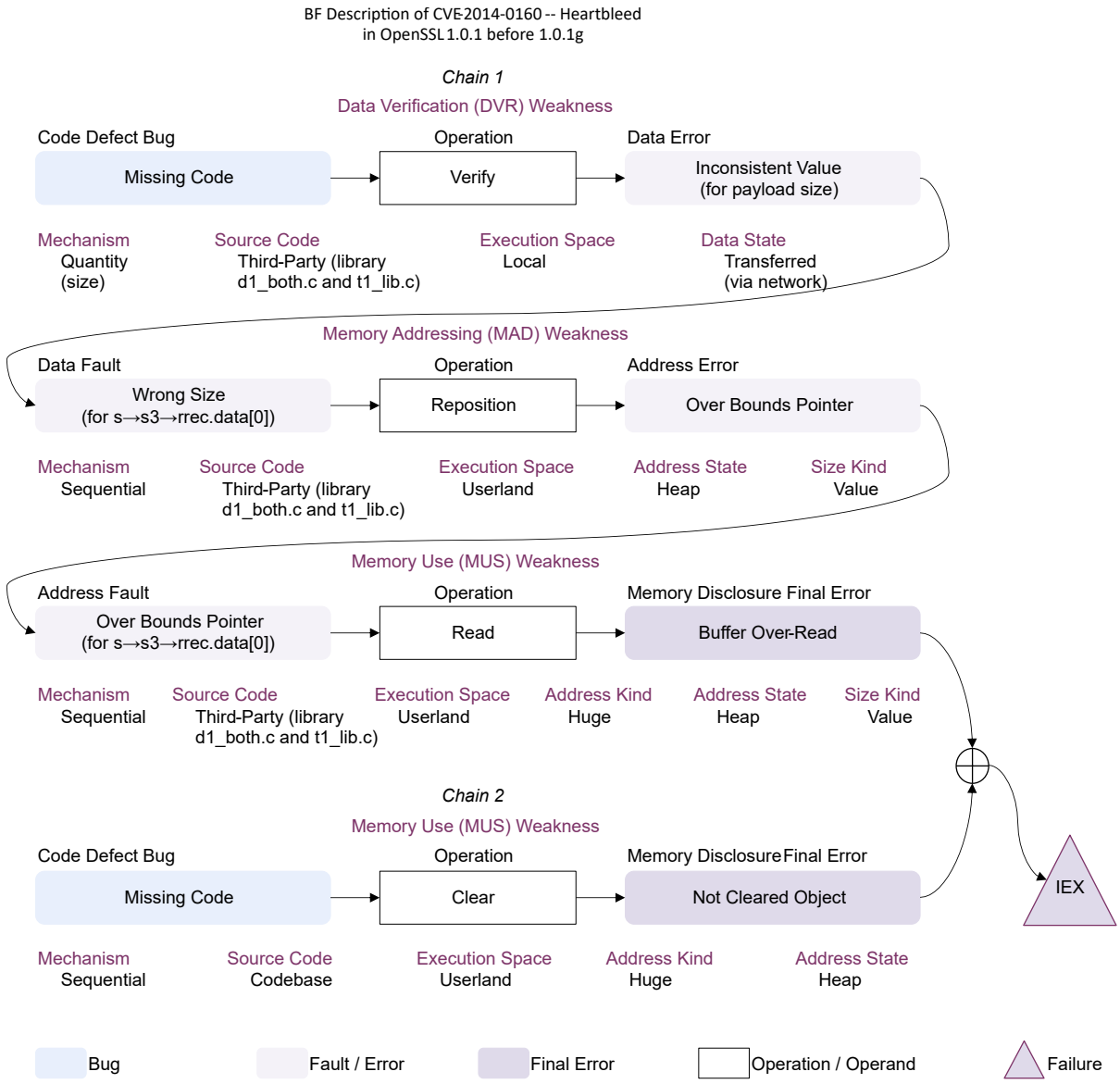


FIGURE 4: The BF description of Heartbleed, CVE-2014-0160. Comprises two vulnerability chains of underlying weaknesses, which converge to cause information exposure (IEX) security failure. Machine readable versions of this description are accessible from the BF website [9].

by the `(Missing Code, Verify, Inconsistent Value)` cause-operation-consequence triple (see the first group in *Chain 1*, Figure 4). The missing input data verification (semantics check) code defect bug[2] leads to a data error – a data value that is inconsistent with (not corresponding to) the size of the buffer. The Mechanism, Source Code, and Execution Space attributes are about the Verify operation. The Data State attribute is about the Data operand. Mechanism shows the missing verification should have been check against size (length). Source Code shows the buggy code is in codebase – the `d1_both.c` and `t1_lib.c` source files. Execution Space shows with what privilege level the buggy code is running – it is in an environment with local user (limited) permissions. Data State shows

[2]A bug always causes the first weakness in a chain of weaknesses; it is the code or specification defect, that has to be fixed to resolve the vulnerability [11]

where the data come from – they are transferred via a network.

Next is the Memory Addressing (MAD) weakness, represented by the `(Wrong Size, Reposition, Over Bounds Pointer)` cause-operation-consequence triple (see the second group in *Chain 1*, Figure 4). The wrong size data fault[3] at repositioning leads to a pointer pointing overbounds address error. The Mechanism attribute here shows how the repositioning is done – it is sequential, iterating over the buffer elements. The Address State attribute shows where the buffer is laid out – it is dynamically allocated in the heap. The Size Kind attribute shows what the limit for iteration over object's elements is – it is a requested value.

Last in this chain is the Memory Use (MUS) weakness, represented by the `(Over Bounds Pointer, Read, Buffer Over-Read)` cause-operation-consequence triple (see the third group in *Chain 1*, Figure 4). The over bounds pointer address fault results in a buffer over-read memory disclosure final error[4]. The Address Kind attribute shows what the accessed outside object's bounds memory is – it is huge, up to 64KB of memory.

Coming from another chain is again a MUS weakness, represented by the `(Missing Code, Clear, Not Cleared Object)` cause-operation-consequence triple (see *Chain 2*, Figure 4). The missing clear (change to non-meaningful value – e.g., via zeroization) code defect bug leads to an object with not cleared data memory disclosure final error (see the one-group chain in Figure 4). The attributes are the same as for the MUS weakness in the first chain, however, this is a different vulnerability and the source code is in different software.

The memory disclosure final errors buffer over-read and not cleared object, combined, cause information exposure (IEX) security failure. Either the missing verification bug or the missing clear bug has to be fixed to avoid this security failure.

DISCUSSION

The OpenSSL cryptographic software library is used by Internet servers – including Hypertext Transfer Protocol Secure (HTTPS) websites – to secure network communications against eavesdropping or to identify communicating parties. It is widely popular and deployed on servers worldwide. However, it is still software and may have bugs, leading to security vulnerabilities.

Heartbleed was introduced in the OpenSSL TLS Heartbeat Extension in 2012 [1] and when discovered in 2014 was already on hundreds of thousands of web servers. It was so severe that for increased awareness became the first vulnerability with its own logo and name [1], and catalyzed formal methods research towards security vulnerabilities [15].

Once identified, the bug in the TLS implementation of the Heartbeat Extension [10] was easy to fix [14] and a patch for the OpenSSL cryptographic software library was distributed. Nevertheless, in 2019 – five years later – the Heartbleed vulnerability was still present on 77K devices worldwide, including over 18K Apache httpd servers [4]. In 2020, attacks against Heartbleed were still ongoing [16] and there might still be thousands of unpatched, and thus vulnerable to Heartbleed systems, that we are accessing unaware of the risks.

The most frequent memory corruption exploits relate to buffer overflow (i.e., buffer over-write), which is also second to injection in severity [17]. However, as demonstrated by Heartbleed, buffer over-read although not so frequent can also be very dangerous, as exposure of sensitive information is at stake. It is key to acknowledge though that there would be no information exposure failure if sensitive data were always protected or at least always cleared from memory after being used.

Developers of applications could prevent the damage from buffer over-reads if they understood clearly how the failure emerges. As we point, there are two contributing chains of weaknesses and the information exposure failure can be prevented by fixing the bug in only one of these chains (see Figures 2 and 4). The Heartbleed vulnerability bug from the first chain would not cause any harm if the missing clear bug of the second chain is not present or fixed. If unused sensitive data are always promptly removed from memory, even a severe vulnerability such as Heartbleed would be much harder to exploit.

One may argue that the Heartbleed vulnerability could be used to expose sensitive data while they are in use. For example, an ingenious attacker could hit the SSL/TLS certificate private key when it is in the system memory and being used. Even though this attack is extremely hard [18], clarifying that BF descriptions are context-sensitive is crucial.

The context of other software or hardware, interacting with the vulnerable software, may inflict modifications in the detailed BF description of a particular vulnerability. The provided in this article description of Heartbleed considers only the OpenSSL Heartbeat

[3]A fault, corresponding to the error from a previous weakness, causes each intermediate weakness. [11]

[4]The last weakness always ends with a final error (undefined or exploitable system behavior). [11]

Extension and the sensitive data left in memory by other software. For example, no trusted execution environment (TEE) [19] is taken into consideration.

In the context of strong memory access control (such as hardware-based protection), the second chain of the BF Heartbleed description would be different. We would use the BF Authentication (ATN) class to describe Heartbleed for a server where private keys could be protected by a TEE but they are not, allowing an attacker to use Heartbleed to reach these keys outside the secure memory.

All this brings us to the realization that more context aware efforts and research are needed towards understandable and formalizable vulnerability descriptions. The former – to assure clear communication about bugs, weaknesses, and vulnerabilities among Information Technology (IT) executives and professionals. The latter – to empower development of tools for vulnerability detection, prediction, and prevention. The precise detailed description of Heartbleed in this article is a demonstration of our efforts towards these goals.

CONCLUSION

Software security vulnerabilities open the door for attacks towards Cyberspace and the critical infrastructure. Heartbleed was one of the most severe and damaging vulnerabilities in the Web history. Because the OpenSSL cryptographic software library is used for secure communication over SSL/TLS and Datagram Transport Layer Security (DTLS), the easy Heartbleed exploits can reveal highly sensitive information, such as private keys, login credentials, and business or personal information from all over the Internet.

In this article, we identify and clearly describe the two chains of weaknesses that underlie Heartbleed and that converge to cause the information exposure failure regularly blamed only on buffer overread. We explain the causal relations between the improper states in the buggy C code. Then we detail the `(bug, operation, error)` or `(fault, operation, error)` triples for each involved weakness. Finally, we provide the full BF description of Heartbleed, clearly showing the bugs starting the two chains, of which at least one must be fixed to avoid the information exposure security failure.

The OpenSSL team fixed the main bug and distributed the patch almost immediately after Heartbleed was disclosed in April 2014. However, even now – nine years later – we have no guarantee sensitive information would not be exposed from the system memory of a server (or a client) running a vulnerable version of OpenSSL.

As demonstrated by Heartbleed, information exposure failure could be reached by exploiting a vulnerability that enables remote access to uncleared objects with sensitive data. Given the intricacy of software stacks in real-world applications, it is improbable such vulnerabilities to be entirely eradicated. Therefore, we urge software developers to promptly clean memory from unused – and especially unencrypted – sensitive data. [20]

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] Wikipedia, *Heartbleed*, Accessed: 2023-02-20. [Online]. Available: https://en.wikipedia.org/wiki/Heartbleed.

[2] MITRE, *Common Vulnerabilities and Exposures (CVE)*, Accessed: 2021-08-28, 2023. [Online]. Available: https://cve.mitre.org/.

[3] MITRE, *CVE-2014-0160*, Accessed: 2023-02-20. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160.

[4] Invicti, Z. Banach, *The heartbleed bug: How a forgotten bounds check broke the internet*, Accessed: 2023-02-24, 2020. [Online]. Available: https://www.invicti.com/blog/web-security/the-heartbleed-bug/.

[5] MITRE, *Common weakness enumeration (CWE)*, Accessed: 2023-03-02, 2023. [Online]. Available: https://cwe.mitre.org.

[6] NVD, *National Vulnerability Database (NVD)*, Accessed: 2023-03-02, 2023. [Online]. Available: https://nvd.nist.gov.

[7] National Vulnerability Database (NVD), *CVE-2014-0160 detail*, Accessed: 2023-02-20. [Online]. Available: https://nvd.nist.gov/vuln/detail/cve-2014-0160.

[8] MITRE, *CWE-119*, Accessed: 2023-02-20. [Online]. Available: https://cwe.mitre.org/data/definitions/119.html.

[9] NIST, I. Bojanova, *The Bugs Framework (BF)*, Accessed: 2023-02-14, 2023. [Online]. Available: https://samate.nist.gov/BF/.

[10] OpenSSL, *Openssl/ssl/d1_both.c*, Accessed: 2023-02-26, 2005. [Online]. Available: https://git.openssl.org/?p=openssl.git;a=blob;f=ssl/d1_both.c;h=0a84f957118afa9804451add380eca4719a9765e;hb=4817504d069b4c5082161b02a22116ad75f822b1.

[11] 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.

[12] 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.

[13] I. Bojanova and C. E. Galhardo, "Classifying Memory Bugs Using Bugs Framework Approach," in *2021 IEEE 45nd Annu. Computer, Software, and Applications Conf. (COMPSAC)*, 2021, in press.

[14] OpenSSL, *Openssl/openssl, openssl_1_0_1-stable*, Accessed: 2023-03-02, 2014. [Online]. Available: https://github.com/openssl/openssl/commit/96db9023b881d7cd9f379b0c154650d6c108e9a3.

[15] A. Thakkar, *Heartbleed: A formal methods perspective*, Accessed: 2023-02-24, 2023. [Online]. Available: https://aalok-thakkar.github.io/webpage_files/publications/CIS673_report.pdf.

[16] NTT, *2020 global threat intelligence report*, Accessed: 2023-02-24, 2020. [Online]. Available: https://services.global.ntt/en-us/insights/2020-global-threat-intelligence-report.

[17] C. E. Galhardo, P. Mell, I. Bojanova, and A. Gueye, "Measurements of the most significant software security weaknesses," in *2020 Annual Computer Security Applications Conference (ACSAC)*, 2020, pp. 154–164.

[18] Nick Sullivan, *Answering the critical question: Can you get private ssl keys using heartbleed?* Accessed: 2023-03-01, 2014. [Online]. Available: https://blog.cloudflare.com/answering-the-critical-question-can-you-get-private-ssl-keys-using-heartbleed/.

[19] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: What it is, and what it is not," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1, 2015, pp. 57–64. DOI: 10.1109/Trustcom.2015.357.

[20] J. Wang, M. Zhao, Q. Zeng, D. Wu, and P. Liu, "Risk assessment of buffer "heartbleed" overread vulnerabilities," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2015, pp. 555–562. DOI: 10.1109/DSN.2015.59.