Explainable Vulnerabilities Descriptions with NIST BF

Keynote – ISSRE, SHIFT & IWSF: Software Hardware Interaction Faults & International Workshop on Software Faults

Charlotte, NC, USA (Remote Attendance) – Oct. 31, 2022
Introduction:
  o Terminology:
    ✓ Bug
    ✓ Weakness
    ✓ Vulnerability
    ✓ Failure
  o “Bad Alloc” Pattern

Existing Repositories:
  o CWE
  o CVE
  o NVD
  o KEV

The Bugs Framework (BF)
  o Goals
  o Features

BF Taxonomy

Validation towards CWE

BF Hands On:
  o BF Descriptions of CVEs
  o ML, AI on Failures and Risks

Potential Impacts
Introduction
Terminology

- **Software Bug:**
  - A coding error
  - Needs to be fixed

- **Software Weakness:**
  - Caused by a bug or ill-formed data
  - Weakness Type – a meaningful notion!

- **Software Vulnerability:**
  - An instance of a weakness type that leads to a security failure
  - May have several underlying weaknesses

- **Security failure:**
  - A violation of a system security requirement
“BadAlloc” Pattern – 25 CVEs

4.2 VULNERABILITY OVERVIEW

4.2.1 INTEGER OVERFLOW OR WRAPAROUND CWE-190
Media Tek LinkIt SDK versions prior to 4.6.1 is vulnerable to integer overflow in memory all memory corruption on the target device.
CVE-2021-30636 has been assigned to this vulnerability. A CVSS v3 base score of 7.3 has been assigned to this vulnerability.

4.2.2 INTEGER OVERFLOW OR WRAPAROUND CWE-190
ARM CMSIS RTOS2 versions prior to 2.1.3 are vulnerable to integer wrap-around in nosRbxMe allocation, resulting in unexpected behavior such as a crash or injected code execution.
CVE-2021-27431 has been assigned to this vulnerability. A CVSS v3 base score of 7.3 has been assigned to this vulnerability.

4.2.3 INTEGER OVERFLOW OR WRAPAROUND CWE-190
ARM mbed-allocator memory library Version 1.3.0 is vulnerable to integer wrap-around in fun unexpected behavior such as a crash or a remote code injection/execution.
CVE-2021-27433 has been assigned to this vulnerability. A CVSS v3 base score of 7.3 has been assigned to this vulnerability.

MUS

MAD

MAL

TCM

DVR

Data Verification

Bug

Type Computation

Weakness

Memory Allocation

Weakness

Memory Addressing

Weakness

Memory Use

Weakness

Failure

DoS / RCE
Existing Repositories
Commonly Used Repositories

- Weaknesses:
  - **CWE** – Common Weakness Enumeration  
    [https://cwe.mitre.org/](https://cwe.mitre.org/)

- Vulnerabilities:
  - **CVE** – Common Vulnerabilities and Exposures  
    → over 18,000 documented in 2020  
    [https://cve.mitre.org/](https://cve.mitre.org/)

- Vulnerabilities by priority for remediation – CVEs:
  - **KEV** – Known Exploited Vulnerabilities Catalog  
    [https://www.cisa.gov/known-exploited-vulnerabilities-catalog](https://www.cisa.gov/known-exploited-vulnerabilities-catalog)

- Linking weaknesses to vulnerabilities – CWEs to CVEs
  - **NVD** – National Vulnerabilities Database  
    → links also to KEV  
    [https://nvd.nist.gov/](https://nvd.nist.gov/)


Repository Problems

1. Imprecise Descriptions – CWE & CVE
2. Unclear Causality – CWE & CVE
3. No Tracking Methodology – CVE
4. Gaps in Coverage – CWE
5. Overlaps in Coverage – CWE
6. No Tools – CWE & CVE
Problem #1: Imprecise Descriptions

- Example:

CWE-502: Deserialization of Untrusted Data:
The application deserializes untrusted data without *sufficiently verifying that the resulting data will be valid*.

  - Unclear what “*sufficiently*” means,
  - “verifying that data is valid” is also confusing
Example:

**CVE-2018-5907**
Possible buffer overflow in `msm_adsp_stream_callback_put` due to lack of input validation of user-provided data that leads to integer overflow in all Android releases (Android for MSM, Firefox OS for MSM, QRD Android) from CAF using the Linux kernel.

→ the NVD label is **CWE-190**

While the CWEs chain is:
CWE-20 → CWE-190 → CWE-119
Problems #4, #5: Gaps/Overlaps in Coverage

- Example:

  CWEs coverage of buffer overflow by:
  - ✓ Read/ Write
  - ✓ Over/ Under
  - ✓ Stack/ Heap

<table>
<thead>
<tr>
<th></th>
<th>Over</th>
<th>Under</th>
<th>Either End</th>
<th>Stack</th>
<th>Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>CWE-127</td>
<td>CWE-126</td>
<td>CWE-125</td>
<td>⚫</td>
<td>⚫</td>
</tr>
<tr>
<td>Write</td>
<td>CWE-124</td>
<td>CWE-120</td>
<td>CWE-123, CWE-787</td>
<td>CWE-121</td>
<td>CWE-122</td>
</tr>
<tr>
<td>Read/ Write</td>
<td>CWE-786</td>
<td>CWE-788</td>
<td>⚫</td>
<td>⚫</td>
<td>⚫</td>
</tr>
</tbody>
</table>
The Bugs Framework (BF)
1. Solve the problems of imprecise descriptions and unclear causality

2. Solve the problems of gaps and overlaps in coverage
BF describes a bug/weakness as:
- An improper state
- Its transition

- Improper State – a tuple \((operation, operand_1, \ldots, operand_n)\), where at least one element is improper
- Transition – the result of the operation over the operands

BF Features – Clear Causal Descriptions

- Initial State – caused by the Bug
  - the operation is improper
- Intermediate State – caused by ill-formed data
  - at least one operand is improper
- Final State – the Failure
  - caused by a final error
BF describes a vulnerability as:
- A chain of improper states and their transitions
- States change until a failure is reached
BF Features – Backtracking

- **How to find the Bug?**
- **Go backwards by operand until an operation is a cause**

Diagram:

- **Improper operand 1** (operation 1, operand 1... operand 1...
- **Improper operand 2** (operation 2, operand 2...
- **Improper operand 3**...
- **Improper operand n**...
- **Final Error**
- **Failure**

Legend:

- **Initial State** – caused by the Bug
- **Intermediate State** – caused by ill-formed data
- **Final State** – the Failure
- **Final Error** – caused by a final error
BF Features – Converging Vulnerabilities

Initial State – caused by the Bug
- the operation is improper

Intermediate State – caused by ill-formed data
- at least one operand is improper

Final State – the Failure
- caused by a final error
BF Features – Classification

- BF Class – a taxonomic category of a weakness type, defined by:
  - A set of operations
  - All valid cause → consequence relations
  - A set of attributes

- BF bug/weakness description – instance of a BF class with:
  - one cause
  - one operation
  - one consequence
  - and their attributes

- BF vulnerability description –
  - chain of BF classes instances
  - consequence–cause transitions.
BF Taxonomy
- Identify Secure Code Principles:
  - Input/Output Safety
  - Data Type Safety
  - Memory Safety
BF Data Type Bugs Model

- Four phases, corresponding to the BF Data Type Bugs classes: DCL, NRS, TCV, and TCM

- Data Type operations flow

- **Entity:**
  - Object
  - Function
  - Data Type
  - Namespace
BF Memory Bugs Model

- The BF Memory Bugs Model:
  - Four phases, corresponding to the BF memory bugs classes: MAD, MAL, MUS, MDL
  - Memory operations flow
BF – Clusters of Bugs Classes

- Input/Output Bugs:
  - DVL, DVR
- Data Type Bugs:
  - DCL, NRS, TVC, TCM
- Memory Bugs:
  - MAD, MAL, MUS, MD
- Cryptography Bugs:
  - ENC, VRF, KMN
- Random Numbers Generation Bugs:
  - RND, PRN
- Access Control Bugs:
  - ATN, AUT
- Control Flow Bugs: ...
- Concurrency Bugs: ...

- BF class:
  - Set of Operations
  - Set of Causes
  - Set of Consequences

https://samate.nist.gov/BF/
## BF Classes – MAD & MUS

### Memory Addressing Bugs (MAD)
- **The pointer to an object is initialized, repositioned, or reassigned to an improper memory address.**

#### Causes
- Improper Operation:
  - Missing
  - Mismatched
  - Erroneous

#### MAD Operations
- Improper Data Value:
  - Hardcoded Address
  - Wrong Index
  - Wrong Size Used

- Improper Data Type:
  - Wrong Index Type
  - Casted Pointer

- Improper Object Address:
  - NULL Pointer
  - Wild Pointer
  - Dangling Pointer
  - Over Bounds Pointer
  - Under Bounds Pointer
  - Wrong Position Pointer

- Improper Object Size:
  - Not Enough Memory Allocated

![Diagram showing MAD operations and consequences](https://samate.nist.gov/BF/Classes/_MEM/MAD.html)

### Memory Use Bugs (MUS)
- **An object is initialized, read, written, or cleared improperly.**

#### Causes
- Improper Operation:
  - Missing
  - Mismatched
  - Erroneous

#### MUS Operations
- Improper Data Value:
  - Forbidden Address
  - Wrong Size Used

- Improper Data Type:
  - Casted Pointer

- Improper Object Address:
  - NULL Pointer
  - Wild Pointer
  - Dangling Pointer
  - Untrusted Pointer
  - Over Bounds Pointer
  - Under Bounds Pointer
  - Wrong Position Pointer

- Improper Object Size:
  - Not Enough Memory Allocated

![Diagram showing MUS operations and consequences](https://samate.nist.gov/BF/Classes/_MEM/MUS.html)

### Attributes
- **Mechanism:** Direct, Sequential
- **Source Code:** Codebase, Third Party, Standard Library, Compiler/Interpreter
- **Execution Space:** Userland, Kernel, Bare-Metal
- **Object Location:** Stack, Heap
- **Memory Error:** Uninitialized Object, Not Cleared Object, NULL Pointer Dereference, Untrusted Pointer Dereference, Object Corruption, Type Confusion, Use After Free, Buffer Overflow, Buffer Underflow, Uninitialized Pointer Dereference

[Links to more information](https://samate.nist.gov/BF/Classes/_MEM/MAD.html) & [https://samate.nist.gov/BF/Classes/_MEM/MUS.html]
BF Classes – DVL & DVR

Data Validation Bugs (DVL) – *Data are validated (syntax check) or sanitized (escape, filter, repair) improperly.*

**Causes**
- Improper Operation:
  - Missing
  - Erroneous
- Improper Policy:
  - Under-Restrictive Policy
  - Over-Restrictive Policy
- Improper Data:
  - Corrupted Data
  - Tampered Data
- Improper Policy Data:
  - Corrupted Policy
  - Tampered Policy

**DVL Operations**
- Validate
- Sanitize

**Consequences**
- Improper Data Type:
  - Invalid Data

**Injection Error**
- Query Injection
- Command Injection
- Source Code Injection
- Parameter Injection
- File Injection

**Attributes**
- Mechanism:
  - Safelist
  - Denylist
  - Format
  - Length
- Source Code:
  - Codebase
  - Third Party
  - Standard Library
  - Compiler/Interpreter
- Execution Space:
  - Local
  - Admin
  - Bare-Metal
- Data State:
  - Entered
  - Stored
  - In Use
  - Transferred

Data Verification Bugs (DVR) – *Data are verified (semantics check) or corrected (assign value, remove) improperly.*

**Causes**
- Improper Operation:
  - Missing
  - Erroneous
- Improper Policy:
  - Under-Restrictive Policy
  - Over-Restrictive Policy
- Improper Data Type:
  - Invalid Data

**DVR Operations**
- Verify
- Correct

**Consequences**
- Improper Data Value:
  - Wrong Value
  - Inconsistent Value
  - Wrong Type

**Attributes**
- Mechanism:
  - Value
  - Quantity
  - Range
  - Type
  - Other Rules
- Source Code:
  - Codebase
  - Third Party
  - Standard Library
  - Compiler/Interpreter
- Execution Space:
  - Local
  - Admin
  - Bare-Metal
- Data State:
  - Entered
  - Stored
  - In Use
  - Transferred

https://samate.nist.gov/BF/Classes/_INP/DVL.html

https://samate.nist.gov/BF/Classes/_INP/DVR.html
BF Classes – NRS, TCV, TCM

Name Resolution Bugs (NRS) – The name of an object, a function, or a data type is resolved improperly or bound to an improper data type or implementation.

Type Conversion Bugs (TCV) – A data value is cast or coerced into another data type improperly.

Type Computation Bugs (TCM) – An arithmetic expression (over numbers, strings, or pointers) is calculated improperly, or a boolean condition is evaluated improperly.
BF Early Work – Buffer Overflow

Table 2. Buffer Overflow CWEs Attributes.

<table>
<thead>
<tr>
<th></th>
<th>before</th>
<th>after</th>
<th>either end</th>
<th>stack</th>
<th>heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>read</td>
<td>127</td>
<td>126</td>
<td>125</td>
<td></td>
<td></td>
</tr>
<tr>
<td>write</td>
<td>124</td>
<td>120</td>
<td>123, 787</td>
<td>121</td>
<td>122</td>
</tr>
<tr>
<td>either r/w</td>
<td>786</td>
<td>788</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Where:
- access = either read/write
- outside = either before/below start or after/above

Towards a “Periodic Table” of Bugs

Irena Bojanova, Paul E. Black, Yaakov Yehsa, Yan Wu

April 9, 2015

NIST, BGSU

Formalizing Software Bugs

Irena Bojanova
UMUC, NIST

CWE-128 in Z notation

CWE-128: Wrap-around Error: “Wrap around errors occur whenever a value is incremented past the maximum value for its type and therefore “wraps around” to a very small, negative, or undefined value.”

12/08/2014

CWE-2014-160/CAPEC-540 in CSP

1.1 History of CWE

There have been several community efforts to leverage the existing large number of diverse real-world vulnerabilities. For example, an important step towards creating the needed collection of software weaknesses types was the establishment of the CWE (Common Vulnerabilities and Exposures) list [2] in 1999 by MITRE. Another important step from MITRE was creating...
Validation towards CWE
BF Class Related CWEs

- **Identify CWEs:**
  1. CWE Filtering
  2. Automated Extraction
  3. Manual Review

BF: https://samate.nist.gov/BF/
CWE: https://cwe.mitre.org/

- **BF Input/Output Bugs Classes – 161 CWEs:**
  - 80.7% – Input Validation Operation
    - 68.3% – Injection Error

- **BF Data Type Bugs Classes – 78 CWEs:**
  - 50% Declaration/Definition Operation
  - 33.3% Cast/Coerce Operation
    - 16% Access Error
    - 0.6% Type Compute Error

- **BF Memory Bugs Classes 52 CWEs:**
  - 61.5% Initialize, Dereference, Read, Write, Clear Operations
    - 67.3% Memory Error
CWEs by BF Operation

- Data Type CWEs
  (incl. Integer Overflow, Juggling, and Pointer Arithmetics) – mapped by BF DCL, RNS, TCV, TCM operation

CWEs by DTC, NRS, TCV, and TCM operation:

- DCL Declare
- DCL Define
- NRS Refer
- NRS Call
- TCV Cast
- TCV Coerce
- TCM Calculate
- TCM Evaluate

CWEs by Abstraction:

- Pillar
- Base
- Class
- Variant
CWEs by BF Consequence

- Input/Output CWEs (incl. Injection) – mapped by BF DVL and BF DVR consequences

CWE by DVL Injection Error:
- Query Injection
- Command Injection
- Source Code Injection
- Parameter Injection
- File Injection

CWE by DVL or DVR Wrong Data for Next Operation Consequence:
- DVL Invalid Data
- DVR Wrong Value, Inconsistent Value, and Wrong Type
- No consequence (only cause listed)

CWEs by Abstraction:
- Pillar
- Base
- Class
- Variant
BF – Defined

- BF is a ...
  - Structured
  - Complete
  - Orthogonal
  - Language Independent

**Classification System** of software bugs and weaknesses.
BF Hands On: BIG-IP TMUI RCE
In BIG-IP versions 15.0.0-15.1.0.3, 14.1.0-14.1.2.5, 13.1.0-13.1.3.3, 12.1.0-12.1.5.1, and 11.6.1-11.6.5.1, the Traffic Management User Interface (TMUI), also referred to as the Configuration utility, has a Remote Code Execution (RCE) vulnerability in undisclosed pages.

- **Vulnerability in BIG-IP TMUI login interface**
  
  https://[F5 Host]/tmui/login.jsp/

- **Proof-Of-Concept: TMSH command execution**
  
  https://[F5 Host]/tmui/login.jsp/..;/tmui/locallb/workspace/tmshCmd.jsp
BF Description of BIG-IP TMUI RCE

Cause
Improper Operation: Missing

DVL Operation
Validate

Consequence
Injection Error:
File Injection
(Relative Path Traversal)

Attributes
Mechanism:
Format (e.g., via ".*\..*\..*" regular expression)

Source Code:
Codebase (login.jsp)

Execution Space:
Admin

Data State:
Transferred (via network)

Remote Code Execution

The Bug
The Failure
BF Hands On: Bad Alloc
“BadAlloc” Pattern – 25 CVEs

4.2 VULNERABILITY OVERVIEW

4.2.1 INTEGER OVERFLOW OR WRAPAROUND CWE-190

MediaTek LinkIt SDK versions prior to 4.6.1 is vulnerable to integer overflow in memory all memory corruption on the target device.

CVE-2021-30636 has been assigned to this vulnerability. A CVSS v3 base score of 7.3 has be

4.2.2 INTEGER OVERFLOW OR WRAPAROUND CWE-190

ARM CMSIS RTOS2 versions prior to 2.1.3 are vulnerable to integer wrap-around in nosRbxMe allocation, resulting in unexpected behavior such as a crash or injected code execution.

CVE-2021-27431 has been assigned to this vulnerability. A CVSS v3 base score of 7.3 has be

4.2.3 INTEGER OVERFLOW OR WRAPAROUND CWE-190

ARM mbed-ualloc memory library Version 1.3.0 is vulnerable to integer wrap-around in fun unexpected behavior such as a crash or a remote code injection/execution.

CVE-2021-27433 has been assigned to this vulnerability. A CVSS v3 base score of 7.3 has be

2. UPDATE INFO

This updated advisory is www.cisa.gov/usert.

3. RISK EVALUAT

Successful exploitation:

- DVR: Data Verification Bug
- TCM: Type Computation Weakness
- MAL: Memory Allocation Weakness
- MAD: Memory Addressing Weakness
- MUS: Memory Use Weakness
- DoS/RCE: Failure
"BadAlloc" (CVE-2021-21834)

An exploitable integer overflow vulnerability exists within the MPEG-4 decoding functionality of the GPAC Project on Advanced Content library v1.0.1. A specially crafted MPEG-4 input when decoding the atom for the "co64" FOURCC can cause an integer overflow due to unchecked arithmetic resulting in a heap-based buffer overflow that causes memory corruption. An attacker can convince a user to open a video to trigger this vulnerability.

```c
GF_Err co64_box_read(GF_Box* s, GF_BitStream* bs) {
    u32 entries;
    GF_ChunkLargeOffsetBox* ptr = (GF_ChunkLargeOffsetBox*)s;
    ptr->nb_entries = gf_bs_read_u32(bs);
    ISOM_DECREASE_SIZE(ptr, 4)
    if (ptr->nb_entries > ptr->size / 8) {
        GF_LOG(GF_LOG_ERROR, GF_LOG_CONTAINER,
            ("[iso file] Invalid number of entries %d in co64\n", 
            ptr->nb_entries));
        return GF_ISOM_INVALID_FILE;
    }
    ptr->offsets = (u64*)gf_malloc(ptr->nb_entries * sizeof(u64));
    if (ptr->offsets == NULL) return GF_OUT_OF_MEM;
    ptr->alloc_size = ptr->nb_entries;
    for (entries = 0; entries < ptr->nb_entries; entries++) {
        ptr->offsets[entries] = gf_bs_read_u64(bs);
    }
    return GF_OK;
}
```

DVR (Missing Verify, Data Value (number of entries), Policy)

TCM (Calculate, Data Value: Wrong Argument Value, Data Type, Function)

MAL (Allocate, Object Address: Over Bounds, Object Size)

MAD (Reposition, Object Address, Object Size: Not Enough Allocated)

MUS (Write, Object Address: Over Bounds, Object Size)

Caused by ill-formed data

Caused by the Bug

The Failure – caused by final error(s)
“BadAlloc” – the Fix

**CVE-2021-21834** An exploitable integer overflow vulnerability exists within the MPEG-4 decoding functionality of the GPAC Project on Advanced Content library v1.0.1. A specially crafted MPEG-4 input when decoding the atom for the “co64” FOURCC can cause an integer overflow due to unchecked arithmetic resulting in a heap-based buffer overflow that causes memory corruption. An attacker can convince a user to open a video to trigger this vulnerability.

```c
GF_Err co64_box_read(GF_Box* s, GF_BitStream* bs)
{
    u32 entries;
    GF_ChunkLargeOffsetBox* ptr = (GF_ChunkLargeOffsetBox*)s;
    ptr->nb_entries = gf_bs_read_u32(bs);
    ISOM_DECREASE_SIZE(ptr, 4)
    if ((u64)ptr->nb_entries > ptr->size / 8 ||
        (u64)ptr->nb_entries > (u64)SIZE_MAX/sizeof(u64))
        GF_LOG(GF_LOG_ERROR, GF_LOG_CONTAINER,
            ("[iso file] Invalid number of entries %d in co64\n",
                ptr->nb_entries));
        return GF_ISOM_INVALID_FILE;
    }

    ptr->offsets = (u64*)gf_malloc(ptr->nb_entries * sizeof(u64));
    if (ptr->offsets == NULL) return GF_OUT_OF_MEM;
    ptr->alloc_size = ptr->nb_entries;
    for (entries = 0; entries < ptr->nb_entries; entries++) {
        ptr->offsets[entries] = gf_bs_read_u64(bs);
    }
    return GF_OK;
}
```

---

**Caused by the Bug**
- DVR (Missing Verify, Data Value (number of entries), Policy)
- TCM (Calculate, Data Value: Wrong Argument Value, Data Type, Function)
- MAL (Allocate, Object Address: Over Bounds, Object Size)
- MAD (Reposition, Object Address, Object Size: Not Enough Allocated)
- MUS (Write, Object Address: Over Bounds, Object Size)

**Caused by ill-formed data**
- Wrap Around
- Not Enough Allocated
- Over Bounds Pointer
- Buffer Overflow

**The Failure – caused by final error(s)**
- DoS / RCE

---

If 
(u64)ptr->nb_entries > ptr->size / 8
||
(u64)ptr->nb_entries > (u64)SIZE_MAX/sizeof(u64))
{
CVE-2021-21834
An exploitable integer overflow vulnerability exists within the MPEG-4 decoding functionality of the GPAC Project on Advanced Content library v1.0.1. A specially crafted MPEG-4 input when decoding the atom for the "co64" FOURCC can cause an integer overflow due to unchecked arithmetic resulting in a heap-based buffer overflow that causes memory corruption. An attacker can convince a user to open a video to trigger this vulnerability.

---

**Caused by the Bug**
- DVR (Missing Verify, Data Value (number of entries), Policy)
- TCM (Calculate, Data Value: Wrong Argument Value, Data Type, Function)
- MAL (Allocate, Object Address: Over Bounds, Object Size)
- MAD (Reposition, Object Address, Object Size: Not Enough Allocated)
- MUS (Write, Object Address: Over Bounds, Object Size)

**Caused by ill-formed data**
- Wrap Around
- Not Enough Allocated
- Over Bounds Pointer
- Buffer Overflow

**The Failure – caused by final error(s)**
- DoS / RCE
BF Hands On: Incorrect Pointer Scaling
CWE-468, Example 1: This example attempts to calculate the position of the second byte of a pointer.

Example Language: C

```c
int *p = x;
char * second_char = (char *)(p + 1);
```

---

**Incorrect Pointer Scaling (CWE-468, Ex. 1)**

- **CWE-468**: Indicates the vulnerability is related to CWE-468, which is likely a specific issue or condition.
- **Example Language**: Indicates the code example is written in C.
- **Example Code**: Shows an attempt to calculate the position of the second byte of a pointer.
- **Diagram**: Illustrates the process and potential issues related to pointer scaling, highlighting incorrect pointer calculation and potential overflow.
- **Comments**: Notes that the incorrect scaling is caused by ill-formed data and that the bug is related to the pointer calculation.
- **Analysis**: Breaks down the areas of concern, such as wrong type, wrong overloaded function, and over bounds, leading to buffer overflow.
**Incorrect Pointer Scaling – the Fix**

**CWE-468 Example 1**

This example attempts to calculate the position of the second byte of a pointer.

*Example Language: C*

```c
int *p = x;
char * second_char = (char *)p + 1;
```

<table>
<thead>
<tr>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **TCV** (Wrong Cast, Data Value, Data Type)
- **TCM** (Calculate, Data Type, Data Value, Wrong Overloaded Function)
- **NRS** (Call, Data Type, Wrong Argument Type, Function)
- **MAD** (Reposition, Object Address, Wrong Index)
- **MUS** (Read, Object Address: Over Bounds, Object Size)

- **Wrong Type**
- **Wrong Overloaded Function**
- **Wrong Result**
- **Over Bounds Pointer**
- **Buffer Overflow**

Caused by **the Bug**
Caused by **ill-formed data**

- **Wrong**
- **Overloaded**
- **Function**
- **Over Bounds**
- **Pointer**

Caused by **ill-formed data**

**Over Bounds**

Caused by **the Bug**
Mechanism:
• Pass In
Source Code:
• Codebase
Data Value Kind:
• Pointer
Data Type Kind:
• Primitive

Mechanism:
• Ad-hoc Bind
Source Code:
• Codebase
Entity:
• Function
Data Type Kind:
• Primitive

Mechanism:
• Operator
Source Code:
• Codebase
Data Value Kind:
• Pointer
Data Type Kind:
• Primitive

BF Description of CWE-468, Example 1

```c
int *p = x;
char * second_char = (char *)(p + 1);
```
BF Hands On: Heartbleed
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.

<table>
<thead>
<tr>
<th>CWE-ID</th>
<th>CWE Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWE-119</td>
<td>Improper Restriction of Operations within the Bounds of a Memory Buffer</td>
</tr>
</tbody>
</table>
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.

```c
void *memcpy (void *dst, const void *src, size_t n) {
    size_t i;
    for (i=0; i<n; i++)
        *(char *) dst++ = *(char *) src++;
    return dst;
}
```

Inconsistent Value

DVR
(Verify: Missing,
Data Value,
Policy)

MAD
(Replication,
Object Address,
Wrong Size Used)

MUS
(Read,
Object Size)

Buffer Overflow

Caused by the Bug

Caused by ill-formed data
Heartbleed (CVE-2014-0160)

- **Inconsistent Value**
- **Over Bounds**
- **Buffer Overflow**
- **Buffer Overflow**
- **Not Cleared Object**

- **DVR (Verify: Missing, Data Value (payload length), Policy)**
- **DVR (Verify: Missing, Data Value (payload length), Policy)**
- **DVR (Verify: Missing, Data Value (payload length), Policy)**
- **DVR (Verify: Missing, Data Value (payload length), Policy)**
- **DVR (Verify: Missing, Data Value (payload length), Policy)**

- **Caused by the Bug**
- **Caused by ill-formed data**
- **The Failure – caused by final error(s)**
Heartbleed buffer overflow is:
- caused by **Data Too Big**
- because of **User Input not Checked Properly**
- where there was a **Read that was After the End that was Far Outside**
- of a buffer in the **Heap**
- which may be exploited for **Information Exposure**

Towards a “Periodic Table” of Bugs

Irena Bojanova, Paul E. Black, Yaacov Yesha, Yan Wu

April 9, 2015  NIST, BGSU

---

*Input not checked properly leads to too much data, where a huge number of bytes are read from the heap in a continuous reach after the array end, which may be exploited for exposure of information that had not been cleared.*

BF Hands On:
NLP/ML/AI on Failures and Risks
CVE-2014-0160 - Heartbleed.bfcve
CVE-2014-0160 - Heartbleed.bfcve

<?xml version="1.0" encoding="utf-8"?>
< CVE-2014-0160 >
  < Bug Type=" INP " Class=" DVR " >
    < Cause Type=" Improper Operation " > Missing </ Cause >
    < Operation > Verify </ Operation >
    < Consequence Comment=" for payload size " Type=" Improper Data Value " > Inconsistent Value </ Consequence >
  </ Bug >
  < Weakness Type=" MEM " Class=" MAD " >
    < Cause Comment=" (for s=s3+rrec.data[0]) " Type=" Improper Data Value " > Wrong Size Used </ Cause >
    < Operation > Repositioin </ Operation >
    < Consequence Type=" Improper Object Address " > Over Bounds Pointer </ Consequence >
    < Attributes >
      < Operation >
        < Attribute Type=" Mechanism " > Sequential </ Attribute >
        < Attribute Comment=" dl_both.c and tl_lib.c " Type=" Source Code " > Codebase </ Attribute >
        < Attribute Type=" Execution Space " > Userland </ Attribute >
      </ Operation >
      < Operand Name=" Object Address " >
        < Attribute Type=" Location " > Heap </ Attribute >
      </ Operand >
    </ Attributes >
  </ Weakness >
  < Weakness Type=" MEM " Class=" MUS " >
    < Cause Comment=" (for s=s3+rrec.data[0]) " Type=" Improper Object Address " > Over Bounds Pointer </ Cause >
    < Operation > Read </ Operation >
    < Consequence Type=" Memory Error " > Buffer Overflow </ Consequence >
    < Attributes > ... </ Attributes >
  </ Weakness >
  < Failure Type=" FLR " Class=" IEX " >
    < Cause Type=" Memory Error " > Buffer Overflow </ Cause >
  </ Failure >
</ CVE-2014-0160 >
<?xml version="1.0" encoding="utf-8"?>
<CVE Name="CVE-2021-21834">
  <Bug Type="_INP" Class="DVR">
    <Cause Type="Improper Operation">Missing</Cause>
    <Operation Comment="(u64)ptr &gt; nb_entries &gt; (u64)SIZE_MAX/sizeof(u64)">Verify</Operation>
    <Consequence Comment="&gt; max 64-bit int" Type="Improper Data Value">Inconsistent Value</Consequence>
    <Attributes>...</Attributes>
  </Bug>

  <Weakness Type="_DTC" Class="TCM">
    <Cause Type="Improper Data Value">Wrong Argument Value</Cause>
    <Operation Comment="ptr &gt; nb_entries &gt; (u64)">Call</Operation>
    <Consequence Type="Improper Data Value">Wrap Around</Consequence>
    <Attributes>...</Attributes>
  </Weakness>

  <Weakness Type="_MEN" Class="MAD">
    <Cause Type="Improper Object Size">Not Enough Memory Allocated</Cause>
    <Operation>Reposition</Operation>
    <Consequence Type="Improper Object Address">Over Bounds Pointer</Consequence>
    <Attributes>...</Attributes>
  </Weakness>

  <Weakness Type="_MEM" Class="MUS">
    <Cause Type="Improper Object Address">Over Bounds Pointer</Cause>
    <Operation>Write</Operation>
    <Consequence Type="Memory Error">Buffer Overflow</Consequence>
    <Attributes>
      <Operation>
        <Attribute Type="Mechanism">Sequential</Attribute>
        <Attribute Comment="Library_box_code_base.c" Type="Source Code">The</Attribute>
        <Attribute Type="Execution Space">Userland</Attribute>
      </Operation>
    </Attributes>
  </Weakness>

  <Weakness Type="_MEM" Class="MAL">
    <Cause Comment="Size of memory to allocate" Type="Improper">Allocate</Cause>
    <Operation Comment="gf_malloc()">Allocate</Operation>
    <Consequence Type="Improper Object Size">Not Enough Memory</Consequence>
    <Attributes>...</Attributes>
  </Weakness>

  <Failure Type="_FLR" Class="DOS">
    <Cause Type="Memory Error">Buffer Overflow</Cause>
  </Failure>
</CVE>
BF in ML & AI

Machine readable formats of:
- BF taxonomy
- BF vulnerability descriptions
- CWEs to BF mappings

→ Query and analyze sets of BF descriptions
→ NLP, ML, and AI projects related to software bugs/weaknesses, failures and risks.
JHU APL – Automated Vulnerability Testing via Executable Attack Graphs:
- Chain vulnerabilities via logical directed graphs
- Determine most mitigation “paths” with least changes
- Detect user behavior prior to malicious effect

The lack of formal, precise descriptions of known vulnerabilities and software weaknesses in the current National Vulnerability Database (NVD) has become an increasingly limiting factor in vulnerability research, mitigation research, and expression of software systems in low level modeling form.

We were thrilled to hear that a researcher at NIST was undertaking the needed improvement to make such descriptions more formal and machine-readable. Such an endeavor will greatly enhance the ability of cyber researchers to explore more complex attacks via computational methods. This will be a huge boost to the U.S.’s ability to defend its networks, military systems, and critical infrastructure, and will lead the way to better mitigation designs, improved software development practices, and automated cyber testing capabilities.

RIT Secure and Trustworthy Cyberspace (SaTC):

The NIST Bugs Framework (BF) has made significant advances in creating first-of-its-kind classification of software weaknesses that has enabled the community to express vulnerabilities using a precise description, allowing us to obtain a fine-grained understanding of security bugs and their root causes. Additionally, the taxonomies and root causes in each bug class will provide us valuable data to guide and enhance our static program analysis techniques and achieve higher accuracy.
BF – Potential Impact
BF – Potential Impacts

- Allow precise communication about software bugs and weaknesses
- Help identify exploit mitigation techniques
Questions