Introduction:
  - Terminology:
    - Bug
    - Weakness
    - Vulnerability
    - Failure
  - “Bad Alloc” Pattern

Existing Repositories:
  - CWE
  - CVE
  - NVD
  - KEV

The Bugs Framework (BF)
  - Goals
  - Features

BF Taxonomy

Validation towards CWE

BF Hands On:
  - BF Descriptions of CVEs
  - NLP, ML, AI Applications

Potential Impacts
Introduction
Terminology

- **Software Bug:**
  - A coding error or a specification error
  - The first error in a chain of weaknesses
  - Needs to be fixed

- **Software Weakness:**
  - Caused by a bug or a previous weakness
  - A chain of weaknesses ends with a final error
  - **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
  - Caused by the final error

The Bugs Framework (BF)
https://samate.nist.gov/BF
“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.

CWE-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 nosRtxMe allocation, resulting in unexpected behavior such as a crash or injected code execution.

CWE-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.

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

4.2.4 INTEGER OVERFLOW OR WRAPAROUND CWE-190

ARM mbed product Version 6.3.0 is vulnerable to integer wrap-around in malloc_wrapper behavior such as a crash or a remote code injection/execution.
Existing Repositories
Commonly Used Repositories

- Weaknesses:
  - CWE – Common Weakness Enumeration
    - https://cwe.mitre.org/

- Vulnerabilities:
  - CVE – Common Vulnerabilities and Exposures
    - over 18 000 documented in 2020
    - https://cve.mitre.org/

- Vulnerabilities by priority for remediation – CVEs:
  - KEV – 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/
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</td>
<td>CWE-787</td>
<td>CWE-121</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 Features – Clear Causal Descriptions

- BF describes a weakness as:
  - An improper state and
  - It’s 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 – Chaining Weaknesses

- BF describes a vulnerability as:
  - A chain of improper states and their transitions
  - States change until a failure is reached

Initial State – caused by the Bug – the operation is improper
Intermediate State – caused by at least one operand is improper
Final State – ends with a final error
Failure – caused by a final error
BF Features – Backtracking

- How to find the Bug?
- Go backwards by operand until an operation is a cause
BF Features – Converging Vulnerabilities

- Improper State 1 (operation 1 operand 1 ... operand 1 ...)
- Improper operand 2,
- Improper operand n
- Improper State n (operation n operand n ... operand n ...)
- Improper operand q'
- Improper State q' (operation q' ... operand q' ...)
- Improper operand q'
- Improper State q' (operation q' ... operand q' ...)
- Improper State 1'
- Improper operand 1',
- Improper operand q'
- Improper State q' (operation q' ... operand q' ...)
- Improper operand q'

Failure

Initial State – caused by the Bug – the operation is improper
Final State – ends with a final error
Failure – caused by final errors

Initial State – caused by the Bug

Final State – ends with a final error

Failure – caused by final errors
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 Features** – Classification

- **BF 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

- Four phases, corresponding to the BF memory bugs classes: MAD, MAL, MUS, MDL

- Memory operations flow

```
<table>
<thead>
<tr>
<th>Object Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAD (Owner)</td>
</tr>
<tr>
<td>Initialize</td>
</tr>
<tr>
<td>Reassign</td>
</tr>
<tr>
<td>Reposition</td>
</tr>
</tbody>
</table>

MAL (Object)
- Allocate
- Extend
- Reallocate

MUS (Object)
- Dereference
- Read
- Initialize
- Write
- Clear

MDL (Object)
- Reduce
- Realocate

Other Object / NULL
- All Objects

Create Object
- Destroy Object

Object Lifetime

Object Size
- Upper Bound
- Lower Bound

Bojanova, 2022
```
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
- Control Flow Bugs
- Concurrency Bugs
- ...

BF cluster:
- Bugs Model
- Set of Classes

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

https://samate.nist.gov/BF/
BF Classes – DVL & DVR

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

**Causes**
- The Bug:
  - Missing Code
  - Erroneous Code
  - Under-Restrictive Policy
  - Over-Restrictive Policy

**DVL Operations**
- Validate
- Sanitize

**Consequences**
- Improper Type: Invalid Data

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

**Improper Policy Data**
- Corrupted Policy Data
- Tampered Policy Data

**Improper Data**
- Corrupted Data
- Tampered Data

**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**
- The Bug:
  - Missing Code
  - Erroneous Code
  - Under-Restrictive Policy
  - Over-Restrictive Policy

**DVR Operations**
- Verify
- Correct

**Consequences**
- Improper Data: Invalid Data

**Improper Data**
- Corrupted Data
- Ampered Data

**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.*

**Causes**
- The Bug:
  - Errorenous Code
  - Missing Qualifier
  - Wrong Qualifier

**Improper Data:**
- Missing Code
- Wrong Code

**Improper Type:**
- Under Range
- Over Range
- Flipped Sign
- Wrong Object Resolved

**Improper Function:**
- Wrong Function Resolved
- Wrong Generic Function Bound
- Wrong Overridden Function Bound
- Wrong Overloaded Function Bound

**NRS Operations**
- Refer
- Call

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

**Causes**
- The Bug:
  - Missing Code
  - Wrong Code

**Improper Data:**
- Under Range
- Over Range
- Flipped Sign
- Wrong Object Resolved

**Improper Type:**
- Wrong Type
- Wrong Object Resolved Type
- Mismatched Argument Type

**TCV Operations**
- Cast
- Coerce

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

**Causes**
- The Bug:
  - Reference vs. Object
  - Wrong Object Resolved Type

**Improper Data:**
- Wrong Code
- Errorenous Code

**Improper Type:**
- Wrong Type
- Wrong Overloaded Function Bound

**Improper Function:**
- Wrong Function Resolved
- Wrong Generic Function Bound
- Wrong Overridden Function Bound

**TCM Operations**
- Calculate
- Evaluate

**Type Computation Error:**
- Undefined

**Attributes**
- Mechanism:
  - Resolve
  - Bind
  - Early Bind
  - Late Bind
  - Ad-hoc Bind
- Source Code:
  - Codebase
  - Third Party
  - Standard Library
  - Compiler/Interpreter
- Type Kind:
  - Primitive
  - Structured
- Data Kind:
  - Numeric
  - Text
  - Pointer
  - Boolean

https://samate.nist.gov BF/Classes/_DTC/NRS.html

https://samate.nist.gov BF/Classes/_DTC/TCV.html

https://samate.nist.gov BF/Classes/_DTC/TCM.html

Bojanova, 2022
vulnerability ::= bug operation
      \{improperOperand or finalError\}

() – zero or more
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

Formalizing Software Bugs

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

```plaintext
channel network 2;
enum (payloadLength, payload, validPayload, invalidPayload);

BAD_INT = MIN_INT + MAX_INT;
BAD_INT < MIN_INT + MAX_INT < BAD_INT;

attack() = network!payloadLength -> network!payload ->
(network!payloadResponse -> attacker);

CWE_128() = network!payloadLength -> network!payload ->
(payloadLength!=EqualTopayloadSize->network!validPayload->CWE_126() |
payloadLength!=NotEqualTopayloadSize->network!invalidPayload->
CWE_126());

System = attack() || CWE_126();
```
Validation towards CWE
BF Class Related CWEs

- 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

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

BF: https://samate.nist.gov/BF/
CWE: https://cwe.mitre.org/
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
- Class
- Base
- 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
  - Technology and Language Independent

Classification System of software bugs and weaknesses.
BF Hands On: BIG-IP TMUI RCE
**BIG-IP TMUI RCE (CVE-2020-5902)**

**CVE-2020-5902** 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**
- The Bug: Missing Code

**Attributes**
- Mechanism: Format (e.g., via ".*\..*\..*\..*" regular expression)
- Source Code: Codebase (login.jsp)
- Execution Space: Admin
- Data State: Transferred (via network)

**DVL Weakness**
- Operation: Validate

**Consequence**
- Injection Error: File Injection (Relative Path Traversal)

**Remote Code Execution**

**The Bug**
- Failure

**Final Error**
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 been assigned.

4.2.2 INTEGER OVERFLOW OR WRAPAROUND CWE-190
ARM CMSIS RTOS2 versions prior to 2.1.3 are vulnerable to integer wrap-around incosRx0Me 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.

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 been assigned.

1. EXECUTIVE SUMMARY
   - CVSS v3.1
   - ATTENTION: Exploit: A
   - Vendors: Multiple
   - Equipment: Multiple
   - Vulnerabilities: Integer

2. UPDATE INFO
   - This updated advisory is www.cisa.gov/uscert.

3. RISK EVALUATION
   - ARM mbed products Version 6.3.0 is vulnerable to integer wrap-around in malloc_wrapper behavior such as a crash or a remote code injection execution.

Data Validation Weakness
Type Computation Weakness
Memory Allocation Weakness
Memory Addressing Weakness
Memory Use Weakness
Failure
DoS / RCE

MAL
MAD
MUS
TCM
DVR

Bojanova, 2022
**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 \&#8220;co64\&#8221; 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.

```
41 GF_Err co64_box_read(GF_Box* s, GF_BitStream* bs) {
42    u32 entries;
43    GF_ChunkLargeOffsetBox* ptr = (GF_ChunkLargeOffsetBox*)s;
44    ptr->nb_entries = gf_bs_read_u32(bs);
45
46    ISOM_DECREASE_SIZE(ptr, 4)
47
48    if (ptr->nb_entries > ptr->size / 8) {
49        GF_LOG(GF_LOG_ERROR, GF_LOG_CONTAINER,
50            ("[iso file] Invalid number of entries %d in co64\n",
51                ptr->nb_entries));
52        return GF_ISOM_INVALID_FILE;
53    }

54    ptr->offsets = (u64*)gf_malloc(ptr->nb_entries * sizeof(u64));
55    if (ptr->offsets == NULL) return GF_OUT_OF_MEM;
56
57    ptr->alloc_size = ptr->nb_entries;
58    for (entries = 0; entries < ptr->nb_entries; entries++) {
59        ptr->offsets[entries] = gf_bs_read_u64(bs);
60    }
61    return GF_OK;
```
**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 &amp;#8220;co64&amp;#8221; 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.
BF Description of “BadAlloc”

**Cause**

- **The Bug:** Missing Code
- **Improper Data:** Wrong Argument Value
- **Improper Size:** Not Enough Memory Allocated

**Consequence**

- **DVR Weakness**
  - Operation: Verify
  - Attributes:
    - Mechanism: Range
    - Source Code: Third Party (Library box_code_base.c)
    - Execution Space: Local
    - Data State: Stored ("number of entries" read from file)
  - Data Kind: Numeric
  - Improper Data: Inconsistent Value (> max 64-bit int)

- **TCM Weakness**
  - Operation: Calculate
  - Attributes:
    - Mechanism: Operator
    - Source Code: Third Party (Library box_code_base.c)
    - Data Kind: Numeric
  - Type Kind: Structured
  - Improper Data: Wrap Around

- **MAL Weakness**
  - Operation: Allocate
  - Attributes:
    - Mechanism: Explicit
    - Source Code: Third Party (Library box_code_base.c)
    - Execution Space: Userland
    - Ownership: Single
    - Location: Heap
  - Improper Size: Not Enough Memory Allocated

- **MAD Weakness**
  - Operation: Reposition
  - Attributes:
    - Mechanism: Sequential
    - Source Code: Third Party (Library box_code_base.c)
    - Execution Space: Userland
    - Location: Heap
  - Improper Data: Over Bounds Pointer

- **MUS Weakness**
  - Operation: Write
  - Attributes:
    - Mechanism: Sequential
    - Source Code: Third Party (Library box_code_base.c)
    - Execution Space: Userland
    - Span: Huge
    - Location: Heap
  - Memory Error: Buffer Overflow

**Denial of Service / Remote Code Execution**

**Final Error**

**Failure**
BF Hands On: Incorrect Pointer Scaling
**Incorrect Pointer Scaling (CWE-468, Ex. 1)**

**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);
```

Diagram:
- **TCV** (Cast: Wrong Code, Data, Type, Function)
- **NRS** (Call, Type: Wrong Argument Type, Function)
- **MUS** (Read, Data, Type, Address: Over Bounds, Size)
- **TCM** (Calculate, Data, Type, Function: Wrong Overloaded Function)
- **MAD** (Reposition, Data, Type, Address: Wrong Index, Size)
- **Wrong Type**
- **Wrong Overloaded Function**
- **Wrong Result**
- **Over Bounds Pointer**
- **Buffer Overflow**

Caused by the Bug
Caused by an improper operand

**Byte Layout:**
- Byte 1: p
- Byte 2: p + 1
- Byte 3: second_char
- Byte 4: Moving 4 bytes

**Note:**
- Incorrect Pointer Scaling (CWE-468, Ex. 1) caused by an improper operand.
**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);
```

- **Wrong Type**
  - TCV (Cast: Wrong Code, Data, Type, Function)

- **Wrong Overloaded Function**
  - TCM (Calculate, Data, Type, Function: Wrong Overloaded Function)

- **Wrong Result**
  - MUS (Read, Data, Type, Address: Over Bounds, Size)

- **Over Bounds Pointer**
  - MAD (Reposition, Data, Type, Address: Wrong Index, Size)

**Buffer Overflow**

- **Caused by the Bug**
- **Caused by an improper operand**

```
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>

Bojanova, 2022
ACME Description of CWE-468, Example 1

**Mechanism:**
- Pass In

**Source Code:**
- Codebase

**Data Kind:**
- Pointer

**Type Kind:**
- Primitive

---

**Attributes**

**Consequence**

**Improper Type:** Wrong Type

**Improper Function:** Wrong Overloaded Function

**Operation:**

\[
\text{(char *)} (p + 1) \quad \text{instead of} \quad \text{(char *)} p + 1
\]

**Cause**

**Imprecise Type:**
- Wrong Argument Type

**Operation:**

\[
\text{Call} \quad ( + \text{operator})
\]

**Attributes**

- Ad-hoc Bind
- Codebase

---

**TCM Weakness**

**Operation:**

**Calculate**

**Consequence**

**Improper Data:** Wrong Result (Pointer Position)

**Attributes**

- Operator
- Codebase

---

**MAD Weakness**

**Operation:**

**Reposition**

**Attributes**

- Direct
- Codebase

---

**MUS Weakness**

**Operation:**

**Read**

**Attributes**

- Direct
- Codebase

---

**The Bug:**

Wrong Code

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

```c
1448 dtls1_process_heartbeat(SSL *s) {
1449     unsigned char *p = &s->s3->rrec.data[0], *pl;
1450     unsigned short hbtype;
1451     unsigned int payload;
1452     unsigned int padding = 16; /* Use minimum padding */
1453     /* Read type and payload length first */
1454     hbtype = *p++;
1455     n2s(p, payload);
1456     pl = p;
1457 ...
1458     if (hbtype == TLS1_HB_REQUEST) {
1459         unsigned char *buffer, *bp;
1460         /* Allocate memory for the response, size is 1 byte
1461          * message type, plus 2 bytes payload, plus
1462          * payload, plus padding */
1463         buffer = OPENSSL_malloc(1 + 2 + payload + padding);
1464         bp = buffer;
1465 ...
1466         /* Enter response type, length and copy payload */
1467         *bp++ = TLS1_HB_RESPONSE;
1468         s2n(payload, bp);
1469         memcpy(bp, pl, payload);
1470     }
1471 }
```
Heartbleed (CVE-2014-0160)

- DVR (Verify: Missing Code, Data (payload length), Policy)
- MAD (Reposition, Data: Wrong Size Used, Type, Address, Size)
- MUS (Read, Data, Type, Address, Over Bounds Pointer, Size)

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

- Initial State – caused by the Bug – the operation is improper
- Intermediate State – caused by at least one operand is improper
- Failure – caused by a final error
- Final State – ends with a final error
BF Description of Heartbleed

The Bug: Missing Code

Mechanism: Sequential
Source Code: Codebase (d1_both.c and tl_lib.c)
Execution Space: Admin
State: Transferred (via network)

Improper Data: Inconsistent Value (for payload size)

MAD Weakness
Operation: Reposition

Improper Address: Over Bounds Pointer

Attributes

Memory Error: Not Cleared Object

MUS Weakness
Operation: Clear

Consequence: Not Cleared Object

Attributes

The Bug: Missing Code

Mechanism: Sequential
Source Code: Codebase (d1_both.c and tl_lib.c)
Execution Space: Userland
Location: Huge

Buffer Overflow

Attributes

Information Exposure

The Bug
Final Error
Failure
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

*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.*

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

April 9, 2015  NIST, BGSU
BF Hands On: NLP/ML/AI Applications for Security Vulnerabilities Research
BF Taxonomy – BF.xml

<!-- @author Irena Bojanova(ivb)-->
<!-- @date 2/9/2022-->
<!-- BF Name="Bugs Framework"> -->
  <Cluster Name="_INP" Type="Weakness">...
  </Cluster>
  <Cluster Name="_DAT" Type="Weakness">...
  </Cluster>
  <Class Name="DCL" Title="Declaration Bugs">...
    <Operations>
      <Operation Name="Declare"/>
      <Operation Name="Define"/>
      <AttributeType Name="Mechanism">...
      <AttributeType Name="Source Code">...
      <AttributeType Name="Entity">...
    </Operations>
    <Operands>
      <Operand Name="Type">...
        <AttributeType Name="Type Kind">...
      </Operand>
    </Operands>
    <Causes>
      <BugCauseType Name="The Bug">
        <Cause Name="Missing Code"/>
        <Cause Name="Wrong Code"/>
        <Cause Name="Erroneous Code"/>
        <Cause Name="Missing Modifier"/>
        <Cause Name="Wrong Modifier"/>
        <Cause Name="Anonymous Scope"/>
        <Cause Name="Wrong Scope"/>
      </BugCauseType>
    </Causes>
    <Consequences>
      <WeaknessConsequenceType Name="Improper Type (_DAT)"
        <Consequence Name="Wrong Type"/>
      </WeaknessConsequenceType>
    </Consequences>
  </Class>
</BF>
CVE-2014-0160 - Heartbleed.bfcve
CVE-2014-0160 - Heartbleed.bfcve
CVE-2021-21834 - Bad Alloc.bfcve

<?xml version="1.0" encoding="utf-8"?>

<CVE Name="CVE-2021-21834">
  <BugWeakness Type="_INP" Class="DVR">
    <Cause Type="The Bug">Missing Code</Cause>
    <Operation Comment="(u64)ptr &gt; nb_entries &gt; (u64)SIZE_MAX/sizeof(u64)">Verify</Operation>
    <Consequence Comment="max 64-bit int" Type="Improper Data">Inconsistent Value</Consequence>
    <Attributes>...</Attributes>
  </BugWeakness>

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

  <Weakness Type="_MEM" 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>
      <Operand Name="Object Address">
        <Attribute Type="Span">Huge</Attribute>
        <Attribute Type="Location">Heap</Attribute>
      </Operand>
      <Attributes>...</Attributes>
    </Attributes>
  </Weakness>

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

  <Weakness Type="_MEM" Class="MAD">
    <Cause Type="Improper">...</Cause>
  </Weakness>

  <Failure Type="_FLR" Class="DOS">
    <Cause Type="Memory Error">Buffer Overflow</Cause>
  </Failure>
</CVE>
CWE mapped to BF – BFCWE.xml

```xml
<Cluster Name="_ALL">
  <showClassCWEs>...</showClassCWEs>
</Cluster>

<Cluster Name="_INP">
  <!--fig 1-->
  <showClassCWEs>
    <ClassOperation name="DVL Validate">
      <CWE>41</CWE>
      <CWE>42</CWE>
      <CWE>43</CWE>
      <CWE>44</CWE>
      ...
    </ClassOperation>
    <ClassOperation name="DVL Sanitize">
      <!--DVL Sanitize operations...-->
    </ClassOperation>
    <ClassOperation name="DVR Check">
      <!--DVR Check operations...-->
    </ClassOperation>
    <ClassOperation name="DVR Verify">
      <!--DVR Verify operations...-->
    </ClassOperation>
    <ClassOperation name="DVL Validate and DVR Verify">
      <!--DVL Validate and DVR Verify operations...-->
    </ClassOperation>
  </showClassCWEs>
</Cluster>

<Consequence name="Query Injection">
  <!--Query Injection consequences...-->
</Consequence>

<Consequence name="Command Injection">
  <!--Command Injection consequences...-->
</Consequence>

<Consequence name="Source Code Injection">
  <!--Source Code Injection consequences...-->
</Consequence>

<Consequence name="Parameter Injection">
  <!--Parameter Injection consequences...-->
</Consequence>

<Consequence name="File Injection">
  <!--File Injection consequences...-->
</Consequence>

<Consequence name="DVL by DVR or DVR Wrong Data">
  <!--DVL by DVR or DVR Wrong Data consequences...-->
</Consequence>

<Consequence name="DVR Wrong Value, Inconsistent Value, and Wrong Behavior">
  <!--DVR Wrong Value, Inconsistent Value, and Wrong Behavior consequences...-->
</Consequence>

<OnlyCause name="No consequence" c="C8C8DA" f="F3F3F3"/>
</classStyles>
```
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:
  o Chain vulnerabilities via logical directed graphs
  o Determine most mitigation “paths” with least changes
  o 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
BF Contact

Irena Bojanova: irena.bojanova@nist.gov

https://samate.nist.gov/BF/