Bugs Framework (BF)
Formalizing Cybersecurity Weaknesses and Vulnerabilities

National Defense Industrial Association (NDIA) Trust & Assurance Committee (T&AC)
March 14 → 28, 2024
Agenda

- Introduction
  - CWE, CVE, NVD
  - BF Approach
  - BF Security Concepts

- BF
  - Bugs Models
  - Weakness Taxonomies
  - Vulnerability Models
  - Formal Language

- BF Datasets
  - BFCWE
  - BCVE

- BF Vulnerability Classification Model

- Potential Impacts
Introduction
Current State of the Art

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

- Vulnerabilities
  - **CVE** – Common Vulnerabilities and Exposures [https://cve.mitre.org/](https://cve.mitre.org/)

- Assigning weaknesses to vulnerabilities – CWEs to CVEs
  - **NVD** – National Vulnerabilities Database [https://nvd.nist.gov/](https://nvd.nist.gov/)
Repository Challenges

- Imprecise descriptions
- Unclear causality
- Gaps in coverage
- Overlaps in coverage
- Wrong NVD assignments
- No tracking methodology
- No tools

<table>
<thead>
<tr>
<th>Challenges</th>
<th>Imprecise Descriptions</th>
<th>Unclear Causality</th>
<th>Gaps in Coverage</th>
<th>Overlaps in Coverage</th>
<th>Wrong CVE to CWE mapping</th>
<th>No Tracking Methodology</th>
<th>No Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWE</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CVE</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NVD</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Focus is on descriptions
BF Approach

BF is a classification of security bugs and related faults, featuring a formal language for unambiguous specification of weaknesses and underlined by them vulnerabilities.

- Bugs and faults – as weakness causes
- Errors and final errors – as weakness consequences

- BF formal language – based on:
  - Weakness taxonomies
  - Bugs models
  - Vulnerability models
BF Weakness

Weakness with an improper operation
Weakness with an improper operand
Weakness resulting in a final error
Failure

Improper State₁
(operation₁₁, operand₁₁, ..., operand₁i, ...)

Improper State₂
(operation₂₁, ..., operand₂₁, ...)

Failure

Improper Stateᵢ
(operationᵢ₁, ..., operandᵢ₁, ..., ...)

Results in Error – Improper Operandᵢ₁

Results in Final Error
BF Weakness States

Improper State caused by a Bug – the operation is improper
Improper State caused by a Fault – an operand is improper
BF Vulnerability

Improper State: an \((\text{operation}, \text{operand}_1, \ldots, \text{operand}_j, \ldots)\) tuple with at least one improper element

- Chaining

- Initial State – caused by a Bug
- Propagation State – caused by a Fault
- Final State – supplies an Exploit Vector
- Failure – result of the exploit of the vector supplied by the Final Error

SW/FM Bug in Operation_1

Error → Fault of Operand_{j1}

Error → Fault of Operand_{j2}

... Error → Fault of Operand_{j_{np}}

Final Error → Vector

Failure (exploit, vector)
BF Bugs Detection

BF Bug Identification

Identify the Bug → Improper Operation\(_1\) → Improper Operand\(_2\) → Improper Operand\(_3\) → Improper Operand\(_n\) → Final Error → Failure

Improper State: an \((operation, operand, ..., operand)\) tuple with at least one improper element

- Chaining
- Backtrack to previous State

Initial State – caused by a Bug
Propagation State – caused by a Fault
Final State – results in an Exploitable Error
Failure – caused by exploitation of the Final Error

BF, I. Bojanova, 2014-2024
BF Security Concepts

Bug/Fault – relates to **Execution Phase:**
- Operations
- Input Operands
- Output Results

- **Security Bug**
  - Code or specification defect
  - May result from a hardware defect
  - May resurface by configuration/environment

- **Fault**
  - Name, data, type, address, or size error
  - Could be from a Bug or induced by a hardware defect

- **Error**
  - From bug or fault
  - Propagates to another fault

- **Security Final Error**
  - From bug or fault
  - Undefined system behavior

- **Security Weakness**
  - (bug, operation, error)
  - (fault, operation, error)
  - (bug, operation, final error)
  - (fault, operation, final error)

- **Security Vulnerability**
  - Chain of weaknesses
  - Bug → Error/Fault → ... → Final Error

- **Security Failure**
  - Violation of system security requirement
    - Information Exposure (IEX)
    - Data Tempering (TPR)
    - Denial of Service (DoS)
    - Arbitrary Code Execution (ACE)
Operation₁ has a Bug and results in Error 1, which becomes Fault₁ for Operation₂, leading to Error₂.

The chain goes on, until the last operation results in a Final Error, leading to a Failure.
BF Bugs Models
BF Memory (_MEM) Bugs Models

- Identify Secure Code Principles:
  - Memory Safety
Identify Secure Code Principles:

- Input/Output Safety
BF Data Type (DAT) Bugs Model

- Identify Secure Code Principles:
  - Data Type Safety
BF Weakness Taxonomies
BF Memory Use (MUS) Class

An object is initialized, read, written, or cleared improperly.
I. Bojanova, 2024

BF Data Validation (DVL) Class

Example from _INP Class Type

Data are validated (syntax check) or sanitized (escape, filter, repair) improperly.

(Bug, Operation₁, Error₁) ← lookup_weakness_triple()
(Fault₁, Operation₂, Error₂) ← lookup_weakness_triple()
(Faultₙ₋₁, Operationₙ, Final Error) ← lookup_weakness_triple()

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

Taxonomy

BF Data Validation (DVL) Class
An arithmetic expression (over numbers, strings, or pointers) is calculated improperly, or a boolean condition is evaluated improperly.

(Bug, Operation₁, Error₁) ← lookup_weakness_triple()
(Fault₁, Operation₂, Error₂) ← lookup_weakness_triple()
(Faultᵣ, Operationᵣ, Final Error) ← lookup_weakness_triple()
BF Weakness Taxonomies

- **Structured**
  - (bug/fault, operation, error/final error)

- **Complete**
  - no gaps in coverage

- **Orthogonal**
  - no overlaps

- **Language and domain independent**
  - context-free

- **Causation rules**
  - cause-consequence transition by operation
<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"/>
</Operations>

<Operands>
  <Operand Name="Type"/>
</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>

<WeaknessConsequenceType Name="Improper Type (_DAT)">
  <Consequence Name="Wrong Type"/>
  <Consequence Name="Incorrect Type"/>
  <Consequence Name="Inconsistent Type"/>
</WeaknessConsequenceType>

<Values>
</Values>

</Class>

</BF>

<!-- Clusters-->

<Definition Name="_INP" Type="Weakness">Input/Output Check Bugs</Definition>

<Definition Name="_DAT" Type="Weakness">Data Type Bugs - lead to memory bugs</Definition>

<Definition Name="_MEM" Type="Weakness">Memory Bugs - lead to memory bugs</Definition>

<Definition Name="_CRY" Type="Weakness">Cryptographic Store or Transmittal Bugs</Definition>

<Definition Name="_RND" Type="Weakness">Random Number Generation Bugs</Definition>

<Definition Name="_ACC" Type="Weakness">Access Control Bugs - lead to memory bugs</Definition>

<Definitions>

<!-- Classes - xxx update the definitions on BF web-site-->

<Definition Name="DVL">Data are validated (syntax check) or sanitized (semantics check) or contextualized (contextual validation)</Definition>

<Definition Name="DVR">Data are verified (syntax check) or sanitized (semantics check) or contextualized (contextual validation)</Definition>

<Definition Name="DCL">An object, a function, a type, or a name</Definition>

<Definition Name="NRS">The name of an object, a function, or a type</Definition>

<Definition Name="TCV">Data are converted or coerced into other data</Definition>

<Definition Name="TCM">A numeric, pointer, or string value is cast</Definition>

<Definition Name="MAD">The pointer to an object is initialized</Definition>

<Definition Name="MAL">An object is allocated, extended, or reallocated</Definition>

<Definition Name="MUS">An object is initialized, read, written, or deleted</Definition>

<Definition Name="MDL">An object is deallocated, reduced, or reallocated</Definition>

</Definitions>
BF Vulnerability Models
• The bug in at least one of the chains must be fixed to resolve the vulnerability
• Fixing a fault may only mitigate the vulnerability
- Chain of (cause, operation, consequence) weakness triples
- Bug = improper operation
- Fault = improper operand
- Bug Types: Code, Specification
- Fault Types: Name, Data, Type, Address, Size
- Causation within a weakness
- Causation between weaknesses
- Causation between vulnerabilities
- Propagation between weaknesses
- Propagation between vulnerabilities

**BF Vulnerability Specification Model**

- **Bug Type**: Bug
- **Fault Type**: Fault
- **Operation**: Operation
- **Error Type**: Error
- **Source Code Value**: Source Code Value
- **Execution Space Value**: Execution Space Value
- **Bug**: Improper operation
- **Fault**: Improper operand
- **Bug Types**: Code, Specification
- **Fault Types**: Name, Data, Type, Address, Size
- **Causation within a weakness**: (Bug, Operation, Error)
- **Causation between weaknesses**: (Bug, Operation, Error)
- **Causation between vulnerabilities**: (Bug, Operation, Error)
- **Propagation between weaknesses**: (Bug, Operation, Error)
- **Propagation between vulnerabilities**: (Bug, Operation, Error)
BF Formal Language
BF Context Free Grammar (CFG)

\[ G = (V, \Sigma, R, S) \]  \hspace{1cm} (1)

, where:

- \( \Sigma \) defines the BF lexis (the alphabet of the CFG) as a finite set of tokens (terminals) comprised by the sets of BF taxons and BF symbols (see Listing 3)

\[ \Sigma = \{ \alpha | \alpha \in \SigmaTaxon \cup \SigmaSymbol \} \]

- \( V \) and \( R \) define the BF syntax as
  
  o a finite set of variables (nonterminals)

\[ V = \{ S, V_1, \ldots, V_n \} \]

and

  o a finite set of syntactic rules (productions) in the form

\[ R = \{ A \rightarrow \omega | A \in V \land \omega \in (V \cup \Sigma)^* \} \]

, where:

\((V \cup \Sigma)^*\) is a string of tokens and/or variables

\(A \rightarrow \omega\) means any variable \(A\) occurrence may be replaced by \(\omega\).

- \( S \in V \) is the predefined start variable from which all BF specifications derive.
The formal language is defined as the set of all strings of tokens $\omega$ derivable from the start variable $S$.

$$L(G) = \{ \omega \in \Sigma^* : S \xrightarrow{*} \omega \}$$  \hspace{1cm} (2)

where:

- $\Sigma^*$ is the set of all possible strings that can be generated from $\Sigma$ tokens
- $S$ is the start variable
- $\alpha \xrightarrow{*} \beta$ means string $\alpha$ derives string $\beta$

Note that $\omega$ must be in $\Sigma^*$, the set of strings made from terminals. Strings involving non-terminals are not part of the language.
\[ \Sigma = \{ \Sigma_{\text{Taxon}}, \Sigma_{\text{Symbol}} \} \]

where

\[
\Sigma_{\text{Taxon}} = \{ \Sigma_{\text{Category}}, \Sigma_{\text{ClassType}}, \Sigma_{\text{Class}}, \Sigma_{\text{BugType}}, \Sigma_{\text{Bug}}, \\
\Sigma_{\text{Operation}}, \Sigma_{\text{OperationAttributeType}}, \\
\Sigma_{\text{FaultType}}, \Sigma_{\text{Fault}}, \Sigma_{\text{OperandAttributeType}}, \Sigma_{\text{OperandAttribute}}, \\
\Sigma_{\text{FinalErrorType}}, \Sigma_{\text{FinalError}} \}
\]

\[
\Sigma_{\text{Symbol}} = \{ \rightarrow, \leftarrow, \oplus \}
\]

\[
\Sigma_{\text{Category}} = \{ \text{Weakness}, \text{Failure} \}
\]

\[
\Sigma_{\text{ClassType}} = \{ \_INP', \_DAT', \_MEM', \ldots \}
\]

\[
\Sigma_{\text{Class}} = \{ \_DVL', \_DVR', \_DCL', \_NRS', \_TCV', \_TCM', \_MAD', \_MMN', \_MUS', \ldots \}
\]

\[
\Sigma_{\text{Operation}} = \{ \text{Validate}, \text{Sanitize}, \text{Verify}, \text{Correct}, \text{Declare}, \text{Define}, \text{Refer}, \\
\text{Call}, \text{Casi}, \text{Coerce}, \text{Calculate}, \text{Evaluate}, \text{InitializePointer}, \\
\text{Reposition}, \text{Reassign}, \text{Allocate}, \text{Extend}, \text{Reallocate – Extend}, \\
\text{Deallocate}, \text{Reduce}, \text{Reallocate – Reduce}, \text{InitializeObject}, \\
\text{Dereference}, \text{Read}, \text{Write}, \text{Clear}, \text{Generate/Select}, \text{Store}, \\
\text{Distribute}, \text{Use} \ldots \}
\]

\[\text{type} = \{ \text{CodeDefect}, \text{SpecificationDefect} \}
\]

\[\text{Bug} = \{ \text{MissingCode}, \text{ErroreousCode}, \text{Under – RestrictivePolicy}, \]
\[\text{Over – RestrictivePolicy}, \text{WrongCode}, \text{MissingModifier}, \]
\[\text{WrongModifier}, \text{AnonymousScope}, \text{WrongScope}, \]
\[\text{MissingQualifier}, \text{WrongQualifier}, \text{MismatchedOperation}, \ldots \}
\]

\[\Sigma_{\text{FinalErrorType}} = \{ \text{Injection}, \text{Access}, \text{TypeCompute}, \]
\[\text{MemoryCorruption/Disclosure}, \ldots \}
\]

\[\Sigma_{\text{FinalError}} = \{ \text{QueryInjection}, \text{CommandInjection}, \text{SourceCodeInjection}, \]
\[\text{ParameterInjection}, \text{FileInjection}, \text{WrongAccessObject}, \]
\[\text{WrongAccessType}, \text{WrongAccessFunction}, \text{Undefined}, \]
\[\text{MemoryLeak}, \text{MemoryOverflow}, \text{DoubleDeallocate}, \]
\[\text{ObjectCorruption}, \text{NotClearedObject}, \]
\[\text{NULLPointerDereference}, \text{UntrustedPointerDereference}, \]
\[\text{TypeConfusion}, \text{UseAfterDeallocate}, \text{BufferOverflow}, \]
\[\text{BufferUnderflow}, \text{BufferOver – Read}, \text{BufferUnder – Read}, \ldots \} \]
BF CFG Syntax

\[ S ::= (Vulnerability (⊕ Vulnerability)? Failure^+) + \varepsilon \]  \hspace{1cm} (4)

\[
\begin{align*}
\text{Vulnerability} & ::= +\text{Weakness} \\
\text{Weakness} & ::= \text{Cause Operation Consequence} \\
\text{Cause} & ::= \text{Bug} | \text{Fault} \\
\text{Consequence} & ::= \text{Error} | \text{FinalError}
\end{align*}
\]

\[ S ::= (Vulnerability (⊕ Vulnerability)? Failure^+) + \varepsilon \]  \hspace{1cm} (5)

\[
\begin{align*}
\text{Vulnerability} & ::= \text{SingleWeakness} \\
& \quad | \text{FirstWeakness} (\text{Weakness}^+) \text{LastWeakness} \\
\text{SingleWeakness} & ::= \text{Bug Operation FinalError} \\
\text{FirstWeakness} & ::= \text{Bug Operation} (\text{Error} | \text{FinalError}) \\
\text{Weakness} & ::= \text{Fault Operation Error} \\
\text{LastWeakness} & ::= \text{Error Operation FinalError}
\end{align*}
\]
BF Syntax – LL(1) Grammar

\[ S ::= \text{Vulnerability Converge\_Failure} \]

\[ \text{Vulnerability ::= Bug\_Fault Operation OperAtts\_Error\_FinalError} \]
\[ \text{Bug\_Fault ::= Bug} \]
\[ \quad | \text{Fault} \]
\[ \text{OperAtts\_Error\_FinalError ::= OperationAttribute OperAtts\_Error\_FinalError} \]
\[ \quad | \text{Error Fault OprndAtts\_Operation} \]
\[ \quad | \text{FinalError} \]
\[ \text{OprndAtts\_Operation ::= OperandAttribute OprndAtts\_Operation} \]
\[ \quad | \text{Operation OperAtts\_Error\_FinalError} \]
\[ \text{Converge\_Failure ::= } \oplus \text{Vulnerability Converge\_Failure} \]
\[ \quad | \text{Vector Exploit NextVuln\_Failure} \]
\[ \text{NextVuln\_Failure ::= Fault OprndAtts\_Operation} \]
\[ \quad | \text{Failure } \epsilon \]
BF Semantics – Attribute CGF

SyntaxRules:
\[
S ::= \text{Vulnerability Converge\_Failure}
\]
\[
\text{Vulnerability} ::= \text{Bug\_Fault Operation Oprnd\_Attrs\_Error\_Final\_Error}
\]
\[
\text{Bug\_Fault} ::= \text{Bug}
\]
\[
\text{Fault}
\]
\[
\text{Oprnd\_Attrs\_Error\_Final\_Error} ::= \text{Operation\_Attribute Oprnd\_Attrs\_Error\_Final\_Error}
\]
\[
\text{Error Fault}_1 \text{ Oprnd\_Attrs\_Operation}
\]
\[
\text{Final\_Error}
\]
\[
\text{Oprnd\_Attrs\_Operation} ::= \text{Operand\_Attribute Oprnd\_Attrs\_Operation}
\]
\[
\text{Operation}_k \text{ Oprnd\_Attrs\_Error\_Final\_Error}
\]
\[
\text{Converge\_Failure ::= }\oplus \text{ Vulnerability Converge\_Failure}
\]
\[
\text{Vector Exploit Next\_Vulner\_Failure}
\]
\[
\text{Next\_Vulner\_Failure ::= Fault}_2 \text{ Oprnd\_Attrs\_Operation}
\]
\[
\text{Failure} \varepsilon
\]

SemanticRules:
\[
\text{(Bug, Operation}_1, \text{ Error)} \leftarrow \text{lookup\_weakness\_triple}()
\]
\[
\text{(Bug, Operation}_1, \text{ Final\_Error)} \leftarrow \text{lookup\_weakness\_triple}()
\]
\[
\text{(Fault}_1, \text{ Operation}_k, \text{ Error})), k > 1 \leftarrow \text{lookup\_weakness\_triple}()
\]
\[
\text{(Fault}_1, \text{ Operation}_k, \text{ Final\_Error})), k > 1 \leftarrow \text{lookup\_operation\_flow}()
\]
\[
\text{Fault}_1 \leftarrow \text{if} \text{ (Fault}_1\text{\_ClassType == Error\_ClassType) then Error}
\]

Predicates:
\[
\text{Fault}_1\text{\_Type == Error\_Type}
\]
\[
\text{Vector\_Type == Final\_Error\_Type}
\]
\[
\text{Fault}_2\text{\_Type == Exploit\_Result\_Type}
\]
BF Specifications of CWEs
BFCWE Dataset

<xml version="1.0" encoding="utf-8"?>
<-- Bugs Framework (BF), BFCWE Tool, I. Bojanova, NIST, 2020-2024 -->
<BFCWE-Dataset>
  <CWE ID="20">  
    <BFCWE Cause="Missing Code" Operation="Verify" Consequence="Wrong Value" />
    <BFCWE Cause="Missing Code" Operation="Verify" Consequence="Inconsistent Value" />
    <BFCWE Cause="Errorneous Code" Operation="Verify" Consequence="Wrong Value" />
    <BFCWE Cause="Errorneous Code" Operation="Verify" Consequence="Inconsistent Value" />
    <BFCWE Cause="Missing Code" Operation="Validate" Consequence="Invalid Data" />
    <BFCWE Cause="Errorneous Code" Operation="Validate" Consequence="Invalid Data" />
  </CWE>
  <CWE ID="22">  
    <BFCWE Cause="Under-Restrictive Policy" Operation="Sanitize" Consequence="File Injection" />
  </CWE>
  <CWE ID="23">  
    <BFCWE Cause="Under-Restrictive Policy" Operation="Sanitize" Consequence="File Injection" />
  </CWE>
  <CWE ID="24">  
    <BFCWE Cause="Under-Restrictive Policy" Operation="Sanitize" Consequence="File Injection" />
  </CWE>
  <CWE ID="25">  
    <BFCWE Cause="Under-Restrictive Policy" Operation="Sanitize" Consequence="File Injection" />
  </CWE>
  <CWE ID="26">  
    <BFCWE Cause="Under-Restrictive Policy" Operation="Sanitize" Consequence="File Injection" />
  </CWE>
  <CWE ID="27">  
    <BFCWE Cause="Under-Restrictive Policy" Operation="Sanitize" Consequence="File Injection" />
  </CWE>
  <CWE ID="28">  
    <BFCWE Cause="Under-Restrictive Policy" Operation="Sanitize" Consequence="File Injection" />
  </CWE>
  <CWE ID="29">  
    <BFCWE Cause="Under-Restrictive Policy" Operation="Sanitize" Consequence="File Injection" />
  </CWE>
  <CWE ID="30">  
    <BFCWE Cause="Under-Restrictive Policy" Operation="Sanitize" Consequence="File Injection" />
  </CWE>
  <CWE ID="125">  
    <BFCWE Cause="Over Bounds Pointer" Operation="Read" Consequence="Buffer Over-Read" />
    <BFCWE Cause="Under Bounds Pointer" Operation="Read" Consequence="Buffer Under-Read" />
  </CWE>
  <CWE ID="126">  
    <BFCWE Cause="Over Bounds Pointer" Operation="Read" Consequence="Buffer Over-Read" />
  </CWE>
  <CWE ID="127">  
    <BFCWE Cause="Under Bounds Pointer" Operation="Read" Consequence="Buffer Under-Read" />
  </CWE>
  <CWE ID="128">  
    <BFCWE Operation="Calculate" Consequence="Wrap Around" />
  </CWE>
  <CWE ID="129">  
    <BFCWE Cause="Missing Code" Operation="Verify" Consequence="Wrong Value" />
    <BFCWE Cause="Errorneous Code" Operation="Verify" Consequence="Wrong Value" />
  </CWE>
  <CWE ID="130">  
    <BFCWE Cause="Missing Code" Operation="Verify" Consequence="Inconsistent Value" />
    <BFCWE Cause="Errorneous Code" Operation="Verify" Consequence="Inconsistent Value" />
  </CWE>
  <CWE ID="131">  
    <BFCWE Operation="Calculate" />
  </CWE>
  <CWE ID="132">  
    <BFCWE Cause="Missing Code" Operation="Validate" Consequence="Parameter Injection" />
  </CWE>
  <CWE ID="133">  
    <BFCWE Cause="Missing Code" Operation="Validate" Consequence="Wrong Result" />
  </CWE>
  <CWE ID="134">  
    <BFCWE Cause="Missing Code" Operation="Sanitize" Consequence="Invalid Data" />
    <BFCWE Cause="Errorneous Code" Operation="Sanitize" Consequence="Invalid Data" />
  </CWE>
  <CWE ID="135">  
    <BFCWE Cause="Missing Code" Operation="Sanitize" Consequence="Parameter Injection" />
  </CWE>
  <CWE ID="136">  
    <BFCWE Cause="Missing Code" Operation="Sanitize" Consequence="Parameter Injection" />
  </CWE>
  <CWE ID="137">  
    <BFCWE Cause="Missing Code" Operation="Sanitize" Consequence="Parameter Injection" />
  </CWE>
  <CWE ID="138">  
    <BFCWE Cause="Errorneous Code" Operation="Sanitize" Consequence="Parameter Injection" />
  </CWE>
  <CWE ID="139">  
    <BFCWE Cause="Errorneous Code" Operation="Sanitize" Consequence="Parameter Injection" />
  </CWE>
</BFCWE-Dataset>
CWE-125 – Two BF Specifications

BF Specifications of CWE-125

<table>
<thead>
<tr>
<th>Class Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Corrupion/Disclosure</td>
<td>Bugs/weaknesses allowing a memory corruption/disclosure exploit.</td>
</tr>
<tr>
<td>(MEM)</td>
<td></td>
</tr>
<tr>
<td>Memory Use (MUS)</td>
<td>An object is initialized, read, written, or cleared improperly.</td>
</tr>
<tr>
<td>Operation</td>
<td>Use the value of an object's data.</td>
</tr>
<tr>
<td>Address Fault</td>
<td>The object address in use is wrong.</td>
</tr>
<tr>
<td>Over Bounds Pointer</td>
<td>Holds an address above the upper boundary of its object.</td>
</tr>
<tr>
<td>Under Bounds Pointer</td>
<td>Holds an address above the lower boundary of its object.</td>
</tr>
<tr>
<td>Memory Corruption/Disclosure Final Error</td>
<td>An exploitable or undefined system behavior caused by memory addressing, allocation, use, and deallocation bugs.</td>
</tr>
<tr>
<td>Buffer Over-Read</td>
<td>Reading above the upper bound of an object.</td>
</tr>
<tr>
<td>Buffer Under-Read</td>
<td>Reading below the lower bound of an object.</td>
</tr>
</tbody>
</table>
Data Type CWEs by BF Operation

- Data Type CWEs
  (incl. Integer Overflow, Juggling, and Pointer Arithmetics) – mapped by BF DCL, RNS, TCV, TCM operation
BF Specifications of CVEs
Heartbleed (CVE-2014-0160)

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.
Heartbleed (CVE-2014-0160)

**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 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      {
1460          unsigned char *buffer, *bp;
1461          ...
1462          /* Allocate memory for the response, size is 1 byte message type, plus 2 bytes payload, plus */
1463          /* payload, plus padding */
1464          buffer = OPENSSL_malloc(1 + 2 + payload + padding);
1465          bp = buffer;
1466          /* Enter response type, length and copy payload */
1467          *bp++ = TLS1_HB_RESPONSE;
1468          *bp++ = payload;
1469          memcpy(bp, pl, payload);
1470      }
```

The diagram illustrates the vulnerability and the code snippet shows the problematic memcpy function which causes a buffer over-read. The diagram highlights the inconsistency in the code where the response type is not properly handled. The code snippet includes the problematic memcpy function call and the variable `pl` which is used to copy the payload, leading to a buffer overflow.
BF States of CVE-2014-0160 (Heartbleed)

DVR (Verify: Missing Code, Data (payload length))

MAD (Reposition, Data: Wrong Size Used, Type, Address, Size)

MUS (Read, Data, Type, Address: Over Bounds Pointer, Size)

MUS (Clear: Missing Code, Data, Type, Address, Size)

Buffer Overflow

Not Cleared Object

Inconsistent Value

Over Bounds Pointer

Intermediate State – caused by at least one improper operand

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

Failure

Final State – ends with a final error

IEX

I. Bojanova, 2024
BF Specification of CVE-2014-0160 (Heartbleed)

Data Verification Weakness
- Code Defect Bug: Missing Code to check payload length
  - Mechanism: Range
  - Source Code: Third-Party d1_both.c, tl_lib.c
- Operation: Verify
- Data Error: Inconsistent Value for payload size
  - State: Transferred via network

Memory Addressing Weakness
- Data Error: Wrong Size Used for s\rightarrow s\rightarrow send.data[0]
  - Mechanism: Sequential
  - Source Code: Third-Party d1_both.c, tl_lib.c
- Operation: Reposition
- Address Error: Over Bounds Pointer pl
  - Execution Space: Userland
  - Location: Heap

Memory Use Weakness
- Address Error: Over Bounds Pointer pl
  - Mechanism: Sequential
  - Source Code: Third-Party d1_both.c, tl_lib.c
- Operation: Read
- Final Error: Buffer Over-Read
  - Execution Space: Userland
  - Span: Huge
  - Location: Heap

Code Defect Bug: Missing Code
- Operation: Clear
- Final Error: Not Cleared Object
  - Mechanism: Sequential
  - Source Code: Codebase
  - Execution Space: Userland
  - Span: Huge
  - Location: Heap

BF Specification of CVE-2014-0160 (Heartbleed)
BF Tool – BF Specification of Heartbleed
CVE-2014-0160 - Heartbleed.bfcve
BF Specification of CVE-2014-0160
Heartbleed Heap Buffer Over-Read in OpenSSL v1.0.1 before v1.0.1g

Missing verification of ‘payload’ towards a upper limit leads to use of an inconsistent size for an object, allowing a pointer reposition over its bounds, which, when used in ‘memcpy()’ leads to a heap buffer over-read. If exploited, this can lead to exposure of sensitive information – confidentiality loss.

//generated// Missing Code (in 'dbs1_process heartbeat(SSL *s)') to Range Verify length (1 + 2 + 16 <= s->s3->rec.length 1 + 2 + payload + 16 <= s->s3->rec.length) Transferred (via network) in Third-Party (ssl /d1_both.c#1462 ssl /1/1/lib.c#2591) Local leads to Inconsistent Value (‘payload’)

which propagates to Wrong Size (in ‘memcpy(s3p, pl, payload’) Sequential Reposition (pointer) Heap Used (for s->s3->rec.data[0]) Third-Party (ssl /d1_both.c#1487 ssl /1/1/lib.c#2620) in Userland resulting in Over Bounds Pointer (‘pl’)

which propagates to Over Bounds Pointer (in ‘memcpy(s3p, pl, payload’) Sequential Read (object) Huge (up to 64kb per exploit) Heap Used Third-Party (ssl /d1_both.c#1487 ssl /1/1/lib.c#2620) in Userland resulting in Buffer Over-Read (‘pl’)

If exploited this can lead to IEX (confidentiality loss).

---

### Class Definition

**DVR**
Data Verification (DVR) class — Data are verified (semantics check) or corrected (assign, remove) improperly.
BF Vulnerability Classification

with cweClass as (
    select distinct c.Type, class = c.Name, wo.cwe
    from bf.class c
    inner join bf.operation o on c.Name = o.Class
    inner join cwebf.operation wo on o.Name = wo.operation
)

select m.cve [CVE], m.cwe [CWE], n.score [CVSS], c.url [CodeWithFix], c.Type [BFClassType],
    c.class [BFClass], v.cause [Cause], v.operation [Operation], v.consequence [Consequence]
from cweClass c
inner join nvd.mapCveCwe m on m.cwe = c.cwe
inner join nvd.cve n on n.cve = m.cve
inner join githubVul.cve u on u.cve = n.cve
inner join githubVul.commitId ci on ci.id = u.commitId
inner join cve.cwe w on w.id = m.cwe
inner join cwebf.specification s on s.cwe = m.cwe
inner join cwebf.mainWeakness mw on mw.mainWeakness = s.mainWeakness
inner join bf.validWeakness v on v.id = mw.weakness
left outer join cwebf.weakness cv on cv.cwe = m.cwe and cv.mainWeakness = s.mainWeakness
left outer join bf.validWeakness v on v.id = cv.weakness
left outer join bf.operation oo on oo.Name = vv.operation
left outer join bf.class cc on cc.class = cc.Name
where (c.Type = 'MEM')
order by n.score desc, m.cve, s.cwe, cv.chainId
NVD’s One-to-Five Year Plan

Once the NVD is up and running, Brewer said the program will consider new approaches to improving its processes within the next one to five years, especially around software identification.

Some of the ideas include:

- **Involving more partners:** Being able to have outside parties submit CPE data for the CPE Dictionary in ways that scale to fit the ever-growing number of IT products.

- **Software identification improvements:** Dealing with software identification in the NVD in a way that scales with growing complexities (the adoption of PURLS is considered).

- **New types of data:** Developing capabilities to publish additional kinds of data to the NVD (e.g. from EPSS, NIST Bugs Framework).

- **New use cases:** Developing a way to make NVD data more consumable and more customizable to targeted use cases (e.g. getting email alerts from NVD when CVEs are published).

- **CVE JSON 5.0:** Expanding the NVD’s capabilities to utilize new data points available in CVE JSON 5.0.

- **Automation:** Developing a way to automate at least some CVE analysis activities.

BF in Security Research

Machine readable formats of:
- BF taxonomy
- BFCWE specifications
- BFCVE specifications
- Vulnerability classifications

✔ Projects related to:
  - Vulnerability specification generation
  - Bug detection
  - Vulnerability analysis and remediation
  - Security failures and risks
BF – Potential Impact
BF – Potential Impacts

- Allow precise communication about security bugs, weaknesses, and vulnerabilities
- ML/AI bug finding, vulnerability analysis, and resolution
- Help identify exploit mitigation techniques.
Questions
BF Contact

Irena Bojanova, BF PI & Lead
irena.bojanova@nist.gov

https://samate.nist.gov/BF/
https://usnistgov.github.io/BF/