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/

- Vulnerabilities
  - CVE – Common Vulnerabilities and Exposures
    https://cve.mitre.org/

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

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

**Focus is on descriptions**

<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>
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

Results in Error – Improper Operand

Improper State_1
(operation_3, operand_{11}, ..., operand_{i1}, ...)

Improper State_2
(operation_2, ..., operand_{2j}, ...)

Results in Final Error

Improper State_n
(operation_n, ..., operand_{nj}, ...)

Failure

Weakness with an improper operation
Weakness with an improper operand
Weakness resulting in a final error
Failure
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 \((operation, operand_j, \ldots, operand_{j1}, \ldots)\) 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\(_2\)

Error → Fault of Operand\(_3\)

...  

Error → Fault of Operand\(_n\)

Final Error → Vector

Failure (exploit, vector)
BF Bugs Detection

BF Bug Identification

- Identify the Bug
- Improper Operation
- Improper Operand
- Improper State
- 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)
Operation_1 has a Bug and results in Error_1, which becomes Fault_1 for Operation_2, leading to Error_2.

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

Example from _MEM Class Type

An object is initialized, read, written, or cleared improperly.

(Bug, Operation₁, Error₁) ← lookup_weakness_triple()
(Fault₁, Operation₂, Error₂) ← lookup_weakness_triple()
(Faultₙ₋₁, Operationₙ, Final Error) ← lookup_weakness_triple()
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, Operationₙ, Final Error) ← lookup_weakness_triple()
An arithmetic expression (over numbers, strings, or pointers) is calculated improperly, or a boolean condition is evaluated improperly.
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 Taxonomy – BF.xml

<!-- @author Irena Bojanova(ivb) -->
<!-- @date 2/9/2022 -->

<!-- BF Name="Bugs Framework" -->

<!-- Cluster Name="INP" Type="Weakness" -->

<!-- Cluster Name="DAT" Type="Weakness" -->

<!-- Cluster Name="DAT" Title="Declaration Bugs" -->

<!-- Operations -->

<!-- Operation Name="Declare" -->

<!-- Operation Name="Define" -->

<!-- AttributeType Name="Mechanism" -->

<!-- AttributeType Name="Source Code" -->

<!-- AttributeType Name="Entity" -->

<!-- Operators -->

<!-- Operand Name="Type" -->

<!-- AttributeType Name="Type Kind" -->

<!-- Operands -->

<!-- Causes -->

<!-- BugCauseType Name="The Bug" -->

<!-- Cause Name="Missing Code" -->

<!-- Cause Name="Wrong Code" -->

<!-- Cause Name="Erroneious Code" -->

<!-- Cause Name="Missing Modifier" -->

<!-- Cause Name="Wrong Modifier" -->

<!-- Cause Name="Anonymous Scope" -->

<!-- Cause Name="Wrong Scope" -->

<!-- BugCauseType -->

<!-- Consequences -->

<!-- WeaknessConsequenceType Name="Improper Type (.DAT)" -->

<!-- Consequence Name="Wrong Type" -->

<!-- Values -->

<!-- Definition Name="INP" Type="Weakness" -->

<!-- Definition Name="DAT" Type="Weakness" -->

<!-- Definition Name="MEM" Type="Weakness" -->

<!-- Definition Name="CRY" Type="Weakness" -->

<!-- Definition Name="RND" Type="Weakness" -->

<!-- Definition Name="ACC" Type="Weakness" -->

<!-- Clusters -->

<!-- Definition Name="DVL" -->

<!-- Definition Name="DVR" -->

<!-- Definition Name="NRS" -->

<!-- Definition Name="TCV" -->

<!-- Definition Name="TCM" -->

<!-- MEM -->

<!-- MAD -->

<!-- MAL -->

<!-- MUS -->

<!-- MDL -->
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
BF Vulnerability Specification Model

- 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

- 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 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)

\[ \Sigma = \{ \alpha | \alpha \in \Sigma_{Taxon} \cup \Sigma_{Symbol} \} \]

- \( V \) and \( R \) define the BF syntax as
  - a finite set of variables (nonterminals)
    \[ V = \{S, V_1, \ldots, V_n\} \]

and

- 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

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

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

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

\[\Sigma_{\text{ClassType}} = \{ \text{`INP'}, \text{`DAT'}, \text{`MEM'}, \ldots \} \]

\[\Sigma_{\text{Class}} = \{ \text{`DVL'}, \text{`DVR'}, \text{`DCL'}, \text{`NRS'}, \text{`TCV'}, \text{`TCM'}, \text{`MAD'}, \text{`MMN'}, \text{`MUS'}, \ldots \} \]

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

\[\Sigma_{\text{BugType}} = \{ \text{`CodeDefect'}, \text{`SpecificationDefect'} \} \]

\[\Sigma_{\text{Bug}} = \{ \text{`MissingCode'}, \text{`ErroneousCode'}, \text{`Under -- RestrictivePolicy'}, \text{`Over -- RestrictivePolicy'}, \text{`WrongCode'}, \text{`MissingModifier'}, \text{`WrongModifier'}, \text{`AnonymousScope'}, \text{`WrongScope'}, \text{`MissingQualifier'}, \text{`WrongQualifier'}, \text{`MismatchedOperation'} \ldots \} \]

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

\[\Sigma_{\text{FinalError}} = \{ \text{`QueryInjection'}, \text{`CommandInjection'}, \text{`SourceCodeInjestion'}, \text{`ParameterInjection'}, \text{`FileInjection'}, \text{`WrongAccessObject'}, \text{`WrongAccessType'}, \text{`WrongAccessFunction'}, \text{`Undefined'}, \text{`MemoryLeak'}, \text{`MemoryOverflow'}, \text{`DoubleDeallocation'}, \text{`ObjectCorruption'}, \text{`NullClearedObject'}, \text{`NULLPointerDereference'}, \text{`UntrustedPointerDereference'}, \text{`TypeConfusion'}, \text{`UseAfterDeallocation'}, \text{`BufferOverflow'}, \text{`BufferUnderflow'}, \text{`BufferOverflow -- Read'}, \text{`BufferUnder -- Read'} \ldots \} \]
BF CFG Syntax

\[ S ::= (\text{Vulnerability} (\oplus \text{Vulnerability})? \text{Failure}^+) + \epsilon \]  

(4)

Vulnerability ::= \text{+Weakness}

Weakness ::= \text{Cause Operation Consequence}

Cause ::= \text{Bug} | \text{Fault}

Consequence ::= \text{Error} | \text{FinalError}

\[ S ::= (\text{Vulnerability} (\oplus \text{Vulnerability})? \text{Failure}^+) + \epsilon \]  

(5)

Vulnerability ::= \text{SingleWeakness}

| \text{FirstWeakness (Weakness}^+) \text{LastWeakness}

SingleWeakness ::= \text{Bug Operation FinalError}

FirstWeakness ::= \text{Bug Operation (Error} | \text{FinalError})

Weakness ::= \text{Fault Operation Error}

LastWeakness ::= \text{Error Operation FinalError}
BF Syntax – LL(1) Grammar

\[ S ::= \text{Vulnerability} \text{Converge_FAILURE} \quad (6) \]

\[ \text{Vulnerability} ::= \text{Bug_FAULT} \text{Operation} \text{OperAttrs_Error_FinalError} \]
\[ \text{Bug_FAULT} ::= \text{Bug} \]
\[ \quad | \text{Fault} \]
\[ \text{OperAttrs_Error_FinalError} ::= \text{OperationAttribute} \text{OperAttrs_Error_FinalError} \]
\[ \quad | \text{Error Fault OprndAttrs_Operation} \]
\[ \quad | \text{FinalError} \]
\[ \text{OprndAttrs_Operation} ::= \text{OperandAttribute} \text{OprndAttrs_Operation} \]
\[ \quad | \text{Operation OperAttrs_Error_FinalError} \]
\[ \text{Converge_FAILURE} ::= \ominus \text{Vulnerability Converge_FAILURE} \]
\[ \quad | \text{Vector Exploit NextVulner_FAILURE} \]
\[ \text{NextVulner_FAILURE} ::= \text{Fault OprndAttrs_Operation} \]
\[ \quad | \text{Failure} \epsilon \]
BF Semantics – Attribute CGF

SyntaxRules:

\[ S ::= \text{Vulnerability Converge.Failure} \]
\[ \text{Vulnerability ::= Bug.Fault Operation OperAttr.Error.FinalError} \]
\[ \text{Bug.Fault ::= Bug} \]
\[ \text{Fault} \]
\[ \text{OperAttr.Error.FinalError ::= OperationAttribute OperAttr.Error.FinalError} \]
\[ \text{OperndAttr.Operation ::= OperandAttribute OperndAttr.Operation} \]
\[ \text{Converge.Failure ::= \oplus Vulnerability Converge.Failure} \]
\[ \text{NextVulner.Failure ::= Fault2 OperndAttr.Operation} \]
\[ \text{Failure} \epsilon \]

SemanticRules:

\[ (\text{Bug, Operation}_1, \text{Error}) \leftarrow \text{lookup\_weakness\_triple()} \]
\[ (\text{Bug, Operation}_1, \text{FinalError}) \leftarrow \text{lookup\_weakness\_triple()} \]
\[ (\text{Fault}_1, \text{Operation}_k, \text{Error}), k > 1 \leftarrow \text{lookup\_weakness\_triple()} \]
\[ (\text{Fault}_1, \text{Operation}_k, \text{FinalError}), k > 1 \leftarrow \text{lookup\_operation\_flow()} \]
\[ \text{Fault}_1 \leftarrow \text{if (Fault}_1.\text{ClassType} == \text{Error.ClassType}) \text{then Error} \]

Predicates:

\[ \text{Fault}_1.\text{Type} == \text{Error.Type} \]
\[ \text{Vector.Type} == \text{FinalError.Type} \]
\[ \text{Fault}_2.\text{Type} == \text{ExploitResult.Type} \]
BF Specifications of CWEs
BFCWE Dataset
CWE-125 – Two BF Specifications

BF Specifications of CWE-125

- **Address Fault**: Over Bounds Pointer → **Operation**: Read (Improper Operand: Address) → **Final Error**: Buffer Over-Read
- **Address Fault**: Under Bounds Pointer → **Operation**: Read (Improper Operand: Address) → **Final Error**: Buffer Under-Read

<table>
<thead>
<tr>
<th>Class Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Corruption/Disclosure (MEM)</td>
<td>Bugs/weaknesses allowing a memory corruption/disclosure exploit.</td>
</tr>
<tr>
<td>Class</td>
<td>Definition</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>Definition</td>
</tr>
<tr>
<td>Read</td>
<td>Use the value of an object’s data.</td>
</tr>
<tr>
<td>Cause</td>
<td>Definition</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>Consequence</td>
<td>Definition</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
Analyzing HW CWEs
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.

Weakness Enumeration

<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>
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
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;
}
```

Ref: [OpenSSL Heartbleed Bug](https://www.openssl.org/news/vulnerabilities/20140402a.html)

---

**Diagram:**
- **DVR:** (Data Repositioning) Missing Code, Data
- **MAD:** (Memory Access Violation), Wrong Size Used, Type, Address
- **MUS:** (Memory Usage Error), Read, Data, Type, Address, Over Bounds Pointer, Size
- **Buffer Overflow:** Caused by an improper operand
- **Caused by the Bug:** Ends with a final error

---

I. Bojanova, 2024
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)

Inconsistent Value

Over Bounds Pointer

Buffer Overflow

Not Cleared Object

Intermediate State – caused by at least one improper operand

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

Failure – ends with a final error

Final State – ends with a final error

IEX
BF Specification of CVE-2014-0160 (Heartbleed)

Data Verification Weakness
- Code Defect Bug: Missing Code to check payload length
- Operation: Verify
- Data Error: Inconsistent Value for payload size

Memory Addressing Weakness
- Operation: Reposition
- Address Error: Over Bounds Pointer

Memory Use Weakness
- Operation: Read
- Final Error: Buffer Over-Read
BF Tool – BF Specification of Heartbleed
**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 ‘ds1_process_heartbeat(SSL *)’) to Range Verify length (1+2+16 <= s->s+r->rec.length 1+2+payload+16 <= s->s+r->rec.length) Transferred (via network) in Third-Party (ssl /d1_both.crl1462 s/l1_lib.crl2591) Local leads to Inconsistent Value (‘payload’)

which propagates to Wrong Size (in ‘memcpy(byt, pl, payload)’) Sequential Reposition (pointer) Heap Used (for s+3+r->rec.data[0]) Third-Party (ssl /d1_both.crl1487 s/l1_lib.crl2620) in Userland resulting in Over Bounds Pointer (‘pl’) which propagates to Over Bounds Pointer (in ‘memcpy(byt, pl, payload)’) Sequential Read (object) Huge (up to 64kb per exploit) Heap Used Third-Party (ssl /d1_both.crl1487 s/l1_lib.crl2620) in Userland resulting in Buffer Over-Read (‘pl’)

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

---

**BF Welcome!**

Models ▼ ▼ ▼ Formalism ▼ ▼ ▼ Tools ▼ ▼ ▼ APIs ▼ ▼ ▼ Approach ▼ ▼ ▼ Background ▼ ▼ ▼ Publications ▼ ▼ ▼ Contact

---

**TAXONOMY**

**BF CVE**

Overview
CVE-2004-1287
CVE-2004-2362
CVE-2007-1220
CVE-2007-6429
CVE-2008-4539
CVE-2013-4930
CVE-2013-4931
CVE-2014-0160
CVE-2015-0235
CVE-2015-5221
CVE-2017-17833
CVE-2018-14557
CVE-2019-14814
CVE-2021-19342
CVE-2022-34835
CVE-2023-1283
CVE-2023-2356
CVE-2023-2356
CVE-2023-2364
CVE-2023-3765
...

---

**BFCV Dataset**

---

**Class**

**Definition**

DVR

---

...
BF Vulnerability Classification

```
with cweClass as (
    select distinct c.Type, class = c.Name, wo.cwe
    from bf.class c
    inner join bf.operation a on c.Name = a.Class
    inner join cweBF.operation wo on c.Name = wo.operation
)
select m.cve [CVE], m.cwe [CWE], n.score [CVSS], ci.url [CodeWithFix], c.Type [BFClassType],
    c.class [BFClass], v.cause [Cause], v.operation [Operation], v.consequence [Consequence]
from cweBFClass c
inner join nvd.mapCveCwe m on m.cve = c.cve
inner join nvd.cve n on m.cwe = n.cwe
inner join gitHubVul.cve u on u.cve = n.cve
inner join gitHubVul.commitId ci on ci.id = u.commitId
inner join cwe.cwe w on w.id = m.cwe
inner join cweBF.specification s on s.cwe = m.cwe
inner join cweBF.mainWeakness m1 on m1.mainWeakness = s.mainWeakness
inner join cbf.validWeakness v on v.id = m1.mainWeakness
left outer join cweBF.otherWeakness cw on cw.cwe = m.cwe and cw.mainWeakness = s.mainWeakness
left outer join cbf.validWeaknessvv on vv.id = cw.weakness
left outer join cbf.operation oo on oo.Name = vv.operation
left outer join cbf.class cc on cc.class = cc.Name
where (c.Type = 'MEM')
order by n.score desc, m.cve, s.cwe, cw.chainId
```
BF Data in NVD

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
Irena Bojanova, BF PI & Lead

irena.bojanova@nist.gov