Agenda

- Introduction
- Security Concepts
- “BadAlloc” Pattern
- BF
- Bugs Models
- Weakness Taxonomy
- Vulnerability Model
- Formal Language
- Generation Tools
- BF Hands On:
  - BF Specifications of Vulnerabilities
  - NLP, ML, AI Applications
- Potential Impacts
Introduction
Software Security Vulnerabilities

- Lead to failures – used to attack:
  - Cyberspace
  - Critical infrastructure

- Loosely used Notions:
  - Bug
  - Fault
  - Error
  - Weakness
  - Vulnerability
  - Failure

- AI chatbots generate conflating explanations

---

A software security bug is an error, flaw, failure, or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways and creates a security vulnerability. These bugs may arise due to mistakes made by software developers, incorrect or incomplete specifications, or unexpected interactions between different components of a system.

A software security weakness, on the other hand, is a defect or problem in the design or architecture of a piece of software that could be exploited by an attacker to gain unauthorized access or perform some other malicious action. These weaknesses could include things like poor input validation, lack of proper authentication and authorization, or insecure use of system resources. It’s a security issue that is inherent to the system and that could potentially be exploited.
Definitions

- **Software Security Vulnerability**
  - A chain of weaknesses linked by causality
  - Starts with a bug
  - Ends with a final error, which if exploited leads to a security failure

- **Software Security Weakness**
  - A (bug, operation, error) or (fault, operation, error) *triple*.
  - An instance of a *weakness type* that relates to a distinct phase of software execution, the *operations* specific for that phase and the *operands* required as input to those operations.

- **Software Security Bug**
  - A code or specification defect (operation defect)

- **Software Fault**
  - A name, data, type, address, or size error (operand error)

- **Software Error**
  - The result from an operation with a bug or a faulty operand
  - Becomes a next fault or is a final error

- **Software Final Error**
  - An exploitable or undefined system behavior
  - Leads to a security failure

- **Security Failure**
  - A violation of a system security requirement

Bugs Framework (BF)
https://samate.nist.gov/BF
“BadAlloc” Pattern – 25 CVEs

Data Verification Weakness

Type Computation Weakness

Memory Allocation Weakness

Memory Addressing Weakness

Memory Use Weakness

Failure

DoS / RCE

1. EXECUTIVE SUMMARY
   - CVE-2023-30636
   - MediaTek Linkit SDK versions prior to 4.6.1 are vulnerable to integer overflow in memory all memory corruption on the target device.

2. UPDATE INFO
   - Original release date: April

3. RISK EVALUATION
   - Successful exploitation

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

   - ARM mbedtls-alloc 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.

   - ARM mbedtls product Version 6.3.0 is vulnerable to integer wrap-around in malloc_reaper behavior such as a crash or a remote code injection execution.
BF Defined
BF Definition

BF is a classification system of software security bugs, faults, and weaknesses that allows unambiguous formal descriptions of software security vulnerabilities that exploit them.

BF comprises:

- Bugs models
- A weakness taxonomy
- A vulnerability model
- An LL(1) formal language
- A relational database
- Generation tools.
BF Bugs Models
Identify Secure Code Principles:
- Input/Output Safety
- Data Type Safety
- Memory Safety
Data Type Bugs Model

• Four phases, corresponding to the BF_DAT 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_MEM classes: MAD, MMN (MAL & MDL), and MUS
- Memory operations flow
BF Weakness Taxonomy
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
BF Classes

- Structured
- Complete
- Orthogonal
- Language and domain independent

https://samate.nist.gov/BF/
Memory Use Bugs (MUS)

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

https://samate.nist.gov/BF/ > BF Weakness Taxonomy
BF Vulnerability Model
High Level Vulnerability Model

Bugs Framework (BF) Vulnerability Model

- **Cause**: Bug, Fault
- **Weakness Type**: \( T_1, T_2, \ldots, T_N \)
- **Operation**: \( O_1, O_2, \ldots, O_N \)
- **Consequence**: Error, Final Error

Flow diagram:

1. Cause: Bug → Weakness Type \( T_1 \) → Operation \( O_1 \) → Error \( E_1 \)
2. Cause: Fault \( F_1 \) → Weakness Type \( T_2 \) → Operation \( O_2 \) → Error \( E_2 \)
3. \( \ldots \)
4. Cause: Fault \( F_{N-1} \) → Weakness Type \( T_N \) → Operation \( O_N \) → Final Error

Result: Failure
A formal BF specification of a chain of underlying BF weakness types, leading to instances of failure types

Operation Defect Types:
Code, Specification

Operand Fault Types:
Name, Data, Type, Address, Size

Bug = improper operation
Fault = improper operand.
The bug in at least one of the chains must be fixed to avoid the failure.
BF Formal Language
BF LL(1) Formal Grammar for
Class Types _INP, _DAT, _MEM, _CDS

____________________Variables (Nonterminals)_________________________

\[ V = \{ \text{Vulnerability, Convergence\_Failure, Bug, Operation, OprAttrs\_Fault\_FinalError, Operation\_Attr, Fault, OprdAttr\_Operation, Final\_Error, Oprd\_Attr, Failure, Code\ Defect, Specification\ Defect, DVL\_Operation, DVR\_Operation, DCL\_Operation, NRS\_Operation, TCV\_Operation, TCM\_Operation, MAD\_Operation, MMN\_Operation, MUS\_Operation, KMN\_Operation, CPH\_Operation, DSV\_Operation, Mechanism, Source\ Code, Execution\ Space, Data\ Fault, Type\ Fault, Name\ Fault, Address\ Fault, Size\ Fault, Injection, Access, Type Compute, Memory\ Corruption\/Disclosure, Data\ Security, Data\_Kind, Name\_Kind, Type\_Kind, Address\_Kind, Size\_Kind, Data\_State, Name\_State, Type\_State, Address\_State, Size\_State \} \]

____________________Tokens (Terminals)_________________________

\[ \Sigma = \{ \text{Missing Code\_\ Code, Erroneous Code\_\ Code, Under\-Restrictive\ Policy, Over\-Restrictive\ Policy, 'Wrong Code, Missing Modifier, Wrong Modifier, Anonymous\ Scope, Wrong\ Scope, Missing\ Qualifier, Wrong\ Qualifier, Mismatched\ Operation, Added\ Code, Wrong\ Algorithm, Hardcoded\ Key, Weak\ Protocol, Validate, Sanitize, Verify, Correct, Declare, Define, Refer, Call, Cast, Coerce, Calculate,\ Evaluate, Initialize\ Pointer, Reposition, Reassign, Allocate, Extend, Reallocate\-Extend, Deallocate, Reduce, Reallocate\-Reduce, Initialize\ Object, Dereference, Read, Write, Clear, Generate\Select\ Store,\ Distribute, Use, Destroy, Encrypt, Decrypt, Crypto\ Authenticate, Crypto\ Verify, Safelist, Denylist, Format, Length, Codebase, Third\-Party, Standard\ Library, Compiler\/Interpreter, Local, Admin, Bare\-Metal, Value, Quantity, Range, Data\ Type, Other\ Rules, Simple, Generics, Overriding, Overloading, Resolve, Bind, Early\ Bind, Late\ Bind, Ad\-hoc\ Bind, Pass\ In, Pass\ Out, Function, Operator, Method, Lambda\ Expression, Procedure, Direct, Sequential, Userland, Kernel, Implicit, Explicit, Hash + RND, MAC, Digital\ Signature, Symmetric\ Algorithm, Asymmetric\ Algorithm, Corrupted\ Data, Tampered\ Data, Corrupted\ Policy\ Data, Tampered\ Policy\ Data, Invalid\ Data, Wrong\ Type\ Resolved, Missing\ Overridden\ Function, Missing\ Overloaded\ Function, Incomplete\ Type, Wrong\ Generic\ Type, Confused\ Subtype, Wrong\ Argument\ Type, Wrong\ Object\ Resolved, Wrong\ Object\ Type\ Resolved, Under\ Range, Over\ Range, Flipped\ Sign, Wrong\ Type, Mismatched\ Argument, Wrong\ Function\ Resolved, Wrong\ Generic\ Function\ Bound, Wrong\ Overridden\ Function\ Bound, Wrong\ Overloaded\ Function\ Bound, Wrong\ Argument, Reference\ vs.\ Object, Hardcoded\ Address, Single\ Owned\ Address, Wrong\ Index, Wrong\ Size, Wrong\ Index\ Type, Casted\ Pointer, NULL\ Pointer, Wild\ Pointer, Dangling\ Pointer, Untrusted\ Pointer, Under\ Bounds\ Pointer, Wrong\ Position\ Pointer, Not\ Enough\ Memory, Forbidden\ Address, Weak\ Keying\ Material, Weak\ Ciphertext, Unverified\ Data, Weak\ Key, Weak\ Random\ Bits, Repeated\ IV, Weak\ Shared\ Secrets, Revealed\ Key, Entered, Stored, In\ Use, Transferred, Object, Function,\ Data\ Type, Namespace, Primitive, Structure, Resolved, Bound, Numeric, Text, Pointer, Boolean, Stack, Heap, Other, Actual, Used, Huge, Moderate, Little, Hashes, Keying\ Material, Digital\ Certificate, Credentials, System\ Data, State\ Data, Cryptographic, Digital\ Document, Secret, Private, Public, Query\ Injection, Command\ Injection, Source\ Code\ Injection, Parameter\ Injection, File\ Injection, Wrong\ Access\ Object, Wrong\ Access\ Type, Wrong\ Access\ Function, Undefined, Memory\ Leak, Memory\ Overflow, Double\ Free, Object\ Corruption, Not\ Cleared\ Object, NULL\ Pointer\ Dereference, Untrusted\ Pointer\ Dereference, Type\ Confusion, Use\ After\ Free, Buffer\ Overflow, Buffer\ Underflow, Buffer\ Over\-Read, Buffer\ Under\-Read, Uninitialized\ Pointer\ Dereference, Revealed\ IV, Revealed\ Shared\ Secrets, Revealed\ Domain\ Parameter, Revealed\ Random\ Bits, Revealed\ Plaintext, Revealed\ Key, Forged\ Signature, Spoofed\ Identity, ⊕ \} \]
BF LL(1) Formal Grammar

Rules (Productions)

\( S \rightarrow \text{Vulnerability Convergence\_Failure} \)

\( \text{Vulnerability} \rightarrow \text{Bug Operation OprAttr\_Fault\_FinalError} \)

\( \text{OprAttr\_Fault\_FinalError} \rightarrow \text{Operation\_Attr OprAttr\_Fault\_FinalError} | \text{Fault OprdAttr\_Operation} | \text{Final\_Error} \)

\( \text{OprdAttr\_Operation} \rightarrow \text{Oprand\_Attr OprdAttr\_Operation} | \text{Operation OprdAttr\_Operation\_Fault\_FinalError} \)

\( \text{Convergence\_Failure} \rightarrow \oplus \hspace{1pt} \text{Vulnerability Convergence\_Failure} | \text{Failure} \epsilon \)

\( \text{Bug} \rightarrow \text{Code\_Defect} | \text{Specification\_Defect} \)

\( \text{Code\_Defect} \rightarrow \text{‘Missing Code’} | \text{‘Erroneous Code’} | \text{‘Wrong Code’} | \text{‘Mismatched Operation’} | \text{‘Added Code’} \)

\( \text{Specification\_Defect} \rightarrow \text{‘Under-Restrictive Policy’} | \text{‘Over-Restrictive Policy’} | \text{‘Missing Modifier’} | \text{‘Wrong Modifier’} | \text{‘Anonymous Scope’} | \text{‘Wrong Scope’} | \text{‘Missing Qualifier’} | \text{‘Wrong Qualifier’} | \text{‘Wrong Algorithm’} | \text{‘Hardcoded Key’} | \text{‘Weak Protocol’} \)

\( \text{Operation} \rightarrow \text{DVL\_Operation} | \text{DVR\_Operation} | \text{DCL\_Operation} | \text{NRS\_Operation} | \text{TCV\_Operation} | \text{TCM\_Operation} | \text{MAD\_Operation} | \text{MMN\_Operation} | \text{MUS\_Operation} | \text{KMN\_Operation} | \text{CPH\_Operation} | \text{DSV\_Operation} \)

\( \text{DVL\_Operation} \rightarrow \text{‘Validate’} | \text{‘Sanitize’} \)

\( \text{DVR\_Operation} \rightarrow \text{‘Verify’} | \text{‘Correct’} \)

\( \text{DCL\_Operation} \rightarrow \text{‘Declare’} | \text{‘Define’} \)

\( \text{NRS\_Operation} \rightarrow \text{‘Refer’} | \text{‘Call’} \)

\( \text{TCV\_Operation} \rightarrow \text{‘Cast’} | \text{‘Coerce’} \)

\( \text{TCM\_Operation} \rightarrow \text{‘Calculate’} | \text{‘Evaluate’} \)

\( \text{MAD\_Operation} \rightarrow \text{‘Initialize Pointer’} | \text{‘Reposition’} | \text{‘Reassign’} \)

\( \text{MMN\_Operation} \rightarrow \text{‘Allocate’} | \text{‘Extend’} | \text{‘Reallocate-Extend’} | \text{‘Deallocate’} | \text{‘Reduce’} | \text{‘Reallocate-Reduce’} \)

\( \text{MUS\_Operation} \rightarrow \text{‘Initialize Object’} | \text{‘Dereference’} | \text{‘Read’} | \text{‘Write’} | \text{‘Clear’} \)

\( \text{KMN\_Operation} \rightarrow \text{‘Generate/Select’} | \text{‘Store’} | \text{‘Distribute’} | \text{‘Use’} | \text{‘Destroy’} \)

\( \text{CPH\_Operation} \rightarrow \text{‘Encrypt’} | \text{‘Decrypt’} \)

\( \text{DSV\_Operation} \rightarrow \text{‘Crypto Authenticate’} | \text{‘Crypto Verify’} \)

\( \text{Operand} \rightarrow \text{‘Data’} | \text{‘Name’} | \text{‘Type’} | \text{‘Address’} | \text{‘Size’} \)
BF LL(1) Formal Grammar

**Fault** → Data_Fault | Type_Fault | Name_Fault | Address_Fault | Size_Fault
Data_Fault → 'Corrupted Data' | 'Tampered Data' | 'Corrupted Policy Data' | 'Tampered Policy Data' | 'Invalid Data' | 'Under Range' | 'Over Range' | 'Flipped Sign' | 'Wrong Argument' | 'Reference vs. Object' | 'Hardcoded Address' | 'Single Owned Address' | 'Wrong Index' | 'Wrong Size' | 'Forbidden Address' | 'Weak Keying Material' | 'Weak Cipher' | 'Unverified Data' | 'Weak Key' | 'Weak Random Bits' | 'Repeated IV' | 'Weak Shared Secrets' | 'Revealed Key'
Type_Fault → 'Wrong Type Resolved' | 'Incomplete Type' | 'Wrong Generic Type' | 'Confused Subtype' | 'Wrong Argument Type' | 'Wrong Type' | 'Mismatched Argument' | 'Wrong Object Type Resolved' | 'Wrong Index Type' | 'Casted Pointer'
Name_Fault → 'Missing Overridden Function' | 'Missing Overloaded Function' | 'Wrong Object Resolved' | 'Wrong Object Type Resolved' | 'Wrong Function Resolved' | 'Wrong Generic Function Bound' | 'Wrong Overridden Function Bound' | 'Wrong Overloaded Function Bound'
Address_Fault → 'NULL Pointer' | 'Wild Pointer' | 'Dangling Pointer' | 'Untrusted Pointer' | 'Over Bounds Pointer' | 'Under Bounds Pointer' | 'Wrong Position Pointer'
Size_Fault → 'Not Enough Memory'

**Final_Error** → Injection | Access | Type_Compute | Memory_Corruption/Disclosure | Data_Security
Injection → 'Query Injection' | 'Command Injection' | 'Source Code Injection' | 'Parameter Injection' | 'File Injection'
Access → 'Wrong Access Object' | 'Wrong Access Type' | 'Wrong Access Function'
Type_Compute → 'Undefined'
Memory_Corruption/Disclosure → 'Memory Leak' | 'Memory Overflow' | 'Double Free' | 'Object Corruption' | 'Not Cleared Object' | 'NULL Pointer Dereference' | 'Untrusted Pointer Dereference' | 'Type Confusion' | 'Use After Free' | 'Buffer Overflow' | 'Buffer Underflow' | 'Buffer Over-Read' | 'Buffer Under-Read' | 'Uninitialized Pointer Dereference'
Data_Security → 'Revealed IV' | 'Revealed Shared Secrets' | 'Revealed Domain Parameter' | 'Revealed Random Bits' | 'Revealed Plaintext' | 'Revealed Key' | 'Forged Signature' | 'Spoofed Identity'
BF LL(1) Formal Grammar

**Operation_Attr** → Mechanism | Source_Code | Execution_Space
Mechanism → 'Safelist' | 'Denylist' | 'Format' | 'Length' | 'Value' | 'Quantity' | 'Range' | 'Data Type' | 'Other Rules' | 'Simple' | 'Generics' | 'Overriding' | 'Overloading' | 'Resolve' | 'Bind' | 'Early Bind' | 'Late Bind' | 'Ad-hoc Bind' | 'Pass In' | 'Pass Out' | 'Function' | 'Operator' | 'Method' | 'Lambda Expression' | 'Procedure' | 'Direct' | 'Sequential' | 'Implicit' | 'Explicit' | 'Hash + RND' | 'MAC' | 'Digital Signature' | 'Symmetric Algorithm' | 'Asymmetric Algorithm'
Source_Code → 'Codebase' | 'Third-Party' | 'Standard Library' | 'Compiler/Interpreter'
Execution_Space → 'Local' | 'Admin' | 'Bare-Metal' | 'Userland' | 'Kernel'

**Oprand_Attr** → Data_Kind | Name_Kind | Type_Kind | Address_Kind | Size_Kind | Data_State | Name_State | Type_State | Address_State | Size_State
Data_State → 'Entered' | 'Stored' | 'In Use' | 'Transferred'
Data_Kind → 'Numeric' | 'Text' | 'Pointer' | 'Boolean' | 'Hashes' | 'Keying Material' | 'Digital Certificate' | 'Credentials' | 'System Data' | 'State Data' | 'Cryptographic' | 'Digital Document' | 'Secret' | 'Private' | 'Public'
Name_Kind → 'Object' | 'Function' | 'Data Type' | 'Namespace'
Name_State → 'Resolved' | 'Bound'
Type_Kind → 'Primitive' | 'Structure'
Address_State → 'Stack' | 'Heap' | '/other/
Address_Kind → 'Huge' | 'Moderate' | 'Little'
Size_Kind → 'Actual' | 'Used
BF Specification of 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.
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 {
1450     unsigned char *p = &s->s3->rrec.data[0], *pl;
1451     unsigned short hbtype;
1452     unsigned int payload;
1453     unsigned int padding = 16; /* Use minimum padding */
1454     /* Read type and payload length first */
1455     hbtype = *p++;
1456     n2s(p, payload);
1457     pl = p;

1465     if (hbtype == TLS1_HB_REQUEST)
1466         {
1467             unsigned char *buffer, *bp;
1468             /* Allocate memory for the response, size is 1 byte * message type, plus 2 bytes payload, plus * payload, plus padding */
1469             buffer = OPENSSL_malloc(1 + 2 + payload + padding);
1470             bp = buffer;
1475             /* Enter response type, length and copy payload * /
1476             pl =ERTLS1_HB_RESPONSE;
1477             *bp++ = TLS1_HB_RESPONSE;
1478             for (i=0; i<n; i++)
1479                 *(char *) dst++ = *(char *) src++;
1480             return dst;
1481         }
1482     }
1483 }
```

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)

Caused by the Bug

Caused by an improper operand

Ends with a final error
Heartbleed (CVE-2014-0160)

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

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

- **MUS** (Read, Data, Type, Address: Over Bounds Pointer, Size)
  - Buffer Overflow
  - Buffer Overflow
  - Not Cleared Object

- **IEX**
  - Intermediate State – caused by at least one operand is improper
  - Initial State – caused by the Bug – the operation is improper
  - Final State – ends with a final error
  - Failure – caused by a final error

I. Bojanova, 2022
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
- Memory Use Weakness
  - Address Error
    - Over Bounds Pointer
    - Operation
      - Reposition
    - Source Code Third-Party
      - d1_both.c, tl_lib.c
    - Execution Space
      - Userland
    - Location
      - Heap
- Memory Use Weakness
  - Address Error
    - Over Bounds Pointer
    - Operation
      - Read
    - Source Code Third-Party
      - d1_both.c, tl_lib.c
    - Execution Space
      - Userland
    - Span
      - Huge
    - Location
      - Heap

Code Defect Bug
- Operation
  - Clear
- Memory Use Weakness
  - Not Cleared Object
- Final Error
  - IEX
  - Source Code Codebase
    - Execution Space
      - Userland
    - Span
      - Huge
    - Location
      - Heap

Mechanism
- Range
  - Sequential
- Source Code
  - Third-Party
    - d1_both.c, tl_lib.c
  - Codebase
  - Execution Space
    - Userland
  - Span
    - Huge
  - Location
    - Heap

Execution Space
- Admin
  - Transferred via network

Data Error
- Wrong Size Used for s-a3-rec.data[0]
  - Operation
    - Reposition
  - Source Code Third-Party
    - d1_both.c, tl_lib.c
  - Execution Space
    - Userland
  - Location
    - Heap

Final Error
- Bug
  - Operation
    - Read
  - Source Code Third-Party
    - d1_both.c, tl_lib.c
  - Execution Space
    - Userland
  - Span
    - Huge
  - Location
    - Heap
  - Final Error
BF Generation Tools
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">
    <Class Name="DCL" Title="Declaration Bugs">
      <Operation Name="Declare"/>
      <Operation Name="Define"/>
    
      <AttributeType Name="Mechanism">...
      <AttributeType Name="Source Code">...
      <AttributeType Name="Entity">...
    
    </Operations>
  
    <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>

</Cluster>

</BF>

<!-- Clusters -->

<Definition Name="_INP" Type="Weakness">Input/Output Check Bugs</Definition>
<Definition Name="_DAT" Type="Weakness">Data Type Bugs - lead to memory leak</Definition>
<Definition Name="_MEM" Type="Weakness">Memory Bugs - lead to memory leak</Definition>
<Definition Name="_CRY" Type="Weakness">Cryptographic Store or Use Bugs</Definition>
<Definition Name="_RND" Type="Weakness">Random Number Generation Bugs</Definition>
<Definition Name="_ACC" Type="Weakness">Access Control Bugs - lead to memory leak</Definition>

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

<!-- _INP -->

<Definition Name="DVL">Data are validated (syntax check) or sanitized (content check) before use</Definition>
<Definition Name="DVR">Data are verified (semantics check) or compared against a data source</Definition>

<!-- _DAT -->

<Definition Name="DCL">An object, a function, a type, or a name are declared</Definition>
<Definition Name="NRS">The name of an object, a function, or a variable is used</Definition>
<Definition Name="TCV">Data are converted or coerced into other data types</Definition>
<Definition Name="TCM">A numeric, pointer, or string value is cast</Definition>

<!-- _MEM -->

<Definition Name="MAD">The pointer to an object is initialized, the value is invalid</Definition>
<Definition Name="MAL">An object is allocated, extended, or reallocated</Definition>
<Definition Name="MUS">An object is initialized, read, written, or deallocated</Definition>
<Definition Name="MDL">An object is deallocated, reduced, or released</Definition>

...
CVE-2014-0160 - Heartbleed.bfcve
CVE-2014-0160 - Heartbleed.bfcve

```xml
<?xml version="1.0" encoding="utf-8"?>
<CVE Name="I CVE-2014-0160">
  <BugWeakness Type="_INP" Class="DVR">
    <Cause Type="The Bug">Missing Code</Cause>
    <Operation>Verify</Operation>
    <Consequence Comment="" Type="Improper Data">Inconsistent Value</Consequence>
    <Attributes>...</Attributes>
  </BugWeakness>
  <Weakness Type="_MEM" Class="MAD">
    <Cause Comment="(for s=s3+rec.data[0])" Type="Improper Data">Wrong Size Used</Cause>
    <Operation>Reposition</Operation>
    <Consequence Type="Improper Address">Over Bounds Pointer</Consequence>
    <Attributes>
      <Attribute Type="Mechanism">Sequential</Attribute>
      <Attribute Comment="d1_both.c and tl_lib.c" Type="Source Code">Codebase</Attribute>
      <Attribute Type="Execution Space">Userland</Attribute>
    </Attributes>
    <Operand Name="Object Address">
      <Attribute Type="Location">Heap</Attribute>
    </Operand>
    </Attributes>
  </Weakness>
  <Weakness Type="_MEM" Class="MUS">
    <Cause Comment="(for s=s3+rec.data[0])" Type="Improper 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>
```
BF CVE Challenge

Irene Bojanova, Primary Investigator and lead, Bugs Framework (BF)

Let's start creating a labeled dataset of memory related software security vulnerability specifications using BF's memory bugs formalism (taxonomy and LL(1) formal grammar).

There are 60 426 memory related CVEs (as of August 2023). To start with, we query the CVE for entries with CWEs assigned by NVD, where the CWEs also map by operation to BF Memory Corruption and Disclosure classes. We then order them by their severity scores according to the Common Vulnerability Scoring System (CVSS) and select maximum ten CVEs per operation — thus reducing the count to 91 most severe CVEs per _MEM BF operation.

First set of steps:

1. Explore the 91 CVEs listed below. Each one has a memory related underlying weakness identified via our CWE2BF mappings and the NVD CWE to CVE assignments.
2. Identify a CVE for which you can find the Bug Report, the Code with Bug, and the Code with Fix (locate the specific GitHub repository with the Diffs). See how these are listed for the examples in BFCVE on the left.

Second set of steps:

3. Get to know the BF Memory Bugs Model.
4. Get to know the taxonomies of the BF Memory Corruption/ Disclosure Classes.
5. Get to know the BF Tool.
6. Collaborate on creating a BF specification of your CVE.

Important note: Use the “NVD CWE” and “BF Chain(s) Identifiable from NVD CWE” columns only as possibly useful guidance. In some cases, a listed CWE may be a wrongly assigned one by NVD, so please notify us if you encounter such. In some cases, the listed chains may be wrong or not the only possible, as the CWE information may be wrong or limited.

Third set of steps:

7. Open in a text editor the .bfcve file where you saved the BF CVE description using the BF Tool.
8. Copy the entire content of the .bfcve file. This is your BF CVE specification in XML format.
9. Submit the the copied .bfcve content and the links to the Bug Report, the Code with Bug, and the Code with Fix here:

<table>
<thead>
<tr>
<th>_MEM CVEs</th>
<th>CVSS</th>
<th>BF Class</th>
<th>BF Operation</th>
<th>NVD CWE</th>
<th>BF Chain(s) Identifiable from NVD CWE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE2022-1699</td>
<td>9.9</td>
<td>MMN</td>
<td>Allocate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVE2022-2259</td>
<td>9.8</td>
<td>MMN</td>
<td>Allocate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVE2022-14259</td>
<td>9.8</td>
<td>MMN</td>
<td>Allocate</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CWE mapped to BF – BFCWE.xml
Data Type 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
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.
BF in ML & AI

- 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
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/
BF Model of Security Vulnerability

vulnerability ::= bug operation
   {improperOperand operation
     finalError
   } - zero or more

The Bug

Attributes
Operand Attribute 1, Attribute 2
...
Operand Attribute 1, Attribute 2

Consequence
Improper Operand 1, Improper Operand 2
...
Improper Operand 1, Improper Operand 2

Cause
Class 1

Weakness

The Bug

Attributes
Operand Attribute 1
...
Operand Attribute 1

Consequence
Improper Operand 1
...
Improper Operand 1

Cause
Class 2

Weakness

Improper Operand 1

Attributes
Operand Attribute 2, Attribute 2
...
Operand Attribute 2, Attribute 2

Consequence
Improper Operand 2
...
Improper Operand 2

.............

The Bug

Attributes
Operand Attribute n, Attribute n
...
Operand Attribute n, Attribute n

Consequence
Improper Operand n-1
...
Improper Operand n-1

Cause
Class n

Weakness

Improper Operand n-1

Attributes
Operand Attribute n, Attribute n
...
Operand Attribute n, Attribute n

Consequence
Final Error
...
Final Error

Failure

The Bug

Final Error

Failure
BF Early Work – Buffer Overflow

Causes
- User Input Not Checked Properly
- Size Calculation Too Small
- Incorrect Calculation
- Writables Pointed Out Of Range

Table 2. Buffer Overflow CWEs Attributes.

<table>
<thead>
<tr>
<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>125</td>
<td></td>
<td></td>
</tr>
<tr>
<td>write</td>
<td>124</td>
<td>120</td>
<td>123</td>
<td>787</td>
</tr>
<tr>
<td>either</td>
<td>786</td>
<td>788</td>
<td>121</td>
<td>122</td>
</tr>
</tbody>
</table>

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

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.”

Towards a “Periodic Table” of Bugs

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

April 9, 2015 NIST, BGSU
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="&gt; max 64-bit int Type="Improper Data"> Inconsistent Value </Consequence>
    < Attributes > ... </Attributes>
  </ BugWeakness >

  < Weakness Type="_DTC" Class="TCM" >
    < Cause Type="Improper Data"> Wrong Argument Value </Cause>
    < Operation Comment="ptr-&gt;nb_entries*sizeof(u64)"> calL 
    < 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"> Th 
        < Attribute Type="Execution Space"> User EylA </Attribute>
      </ Operation >
      < Operand Name="Object Address"> 
        < Attribute Type="Span"> Huge </Attribute>
        < Attribute Type="Location"> Heap </Attribute>
      </ Operand >
    </ Attributes >
  </ Weakness >

  < Failure Type="_FLR" Class="DOS" >
    < Cause Type="Memory Error"> Buffer Overflow </Cause>
  </ Failure >
</CVE>
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

Classifying Memory Bugs Using Bugs Framework Approach

Irena Bojanova
Arizona State University
Irena.bojanova@asu.edu

Carlos Eduardo Galdino
IMEDEF
USP
Carlos.galdino@ime.usp.br

Abstract—Recent years have seen an increasing number of software bugs, particularly those related to memory management. In this paper, we propose a new framework for classifying memory bugs, which is based on the Common Vulnerabilities and Exposures (CVE) format. The framework includes a set of rules for identifying memory bugs, as well as a classification of these bugs into different categories.

1. INTRODUCTION

Memory bugs are a significant problem in software development, leading to a wide range of issues such as crashes, data corruption, and security vulnerabilities. The Common Vulnerabilities and Exposures (CVE) format is a widely used standard for identifying and classifying software vulnerabilities, including memory bugs. In this paper, we propose a new framework for classifying memory bugs, which is based on the CVE format.

2. RULES FOR IDENTIFYING MEMORY BUGS

The rules for identifying memory bugs are based on the following criteria:

- The bug must be related to memory management.
- The bug must have an impact on the program's behavior.
- The bug must be reproducible.

3. CLASSIFICATION OF MEMORY BUGS

The memory bugs are classified into different categories, based on their characteristics. The categories include:

- Use-after-free
- Use-after-lease
- Use-after-return
- Use-after-throw
- Use-after-throw

4. EXAMPLES

Example 1: Use-after-free

When a memory block is freed and later accessed, it may contain invalid data, leading to a crash or security vulnerability.

Example 2: Use-after-lease

When a memory block is leased and later accessed, it may contain invalid data, leading to a crash or security vulnerability.

Example 3: Use-after-return

When a memory block is returned and later accessed, it may contain invalid data, leading to a crash or security vulnerability.

Example 4: Use-after-throw

When a memory block is thrown and later accessed, it may contain invalid data, leading to a crash or security vulnerability.

Example 5: Use-after-throw

When a memory block is thrown and later accessed, it may contain invalid data, leading to a crash or security vulnerability.

Conclusions: The proposed framework for classifying memory bugs is a valuable tool for developers and security professionals. It enables a more systematic approach to identifying and addressing memory bugs, which can help improve the overall quality and security of software applications.
BF Contact

Irena Bojanova: irena.bojanova@nist.gov

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