

CWE 구조

• CWE의 표현

- CWE를 View, Weakness, Category, Compound Element 4가지로 구분

CWE VIEW: Research Concepts

View ID: 1000
Type: Graph

CWE CATEGORY: 7PK - Security Features

Category ID: 254

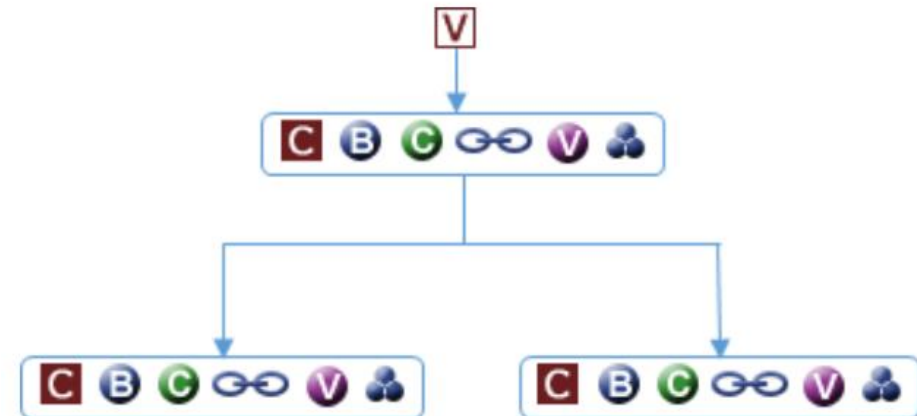
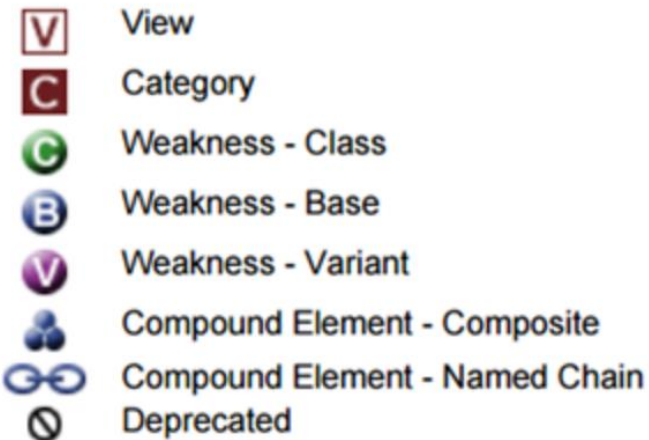
CWE-416: Use After Free

Weakness ID: 416
Abstraction: Variant

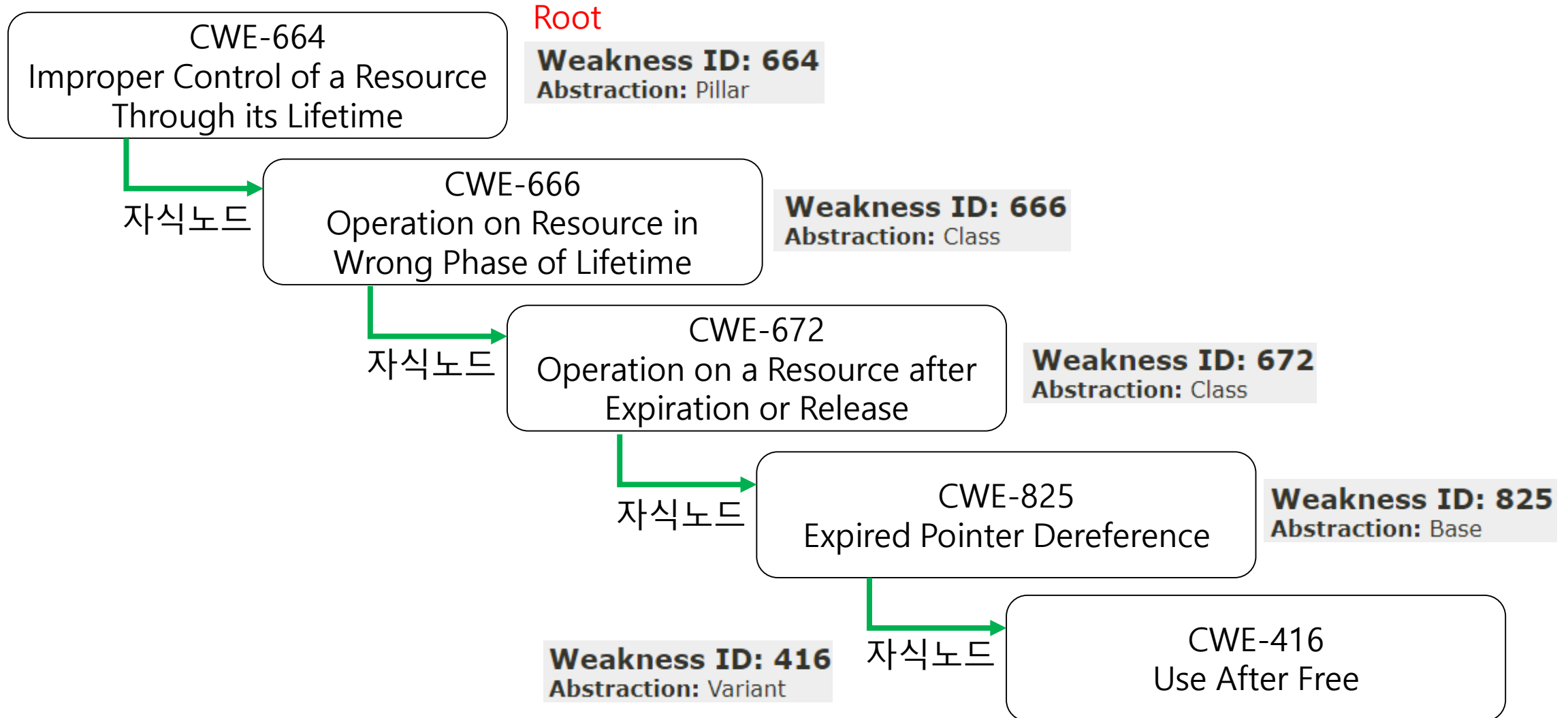
CWE-61: UNIX Symbolic Link (Symlink) Following

Weakness ID: 61
Abstraction: Compound

- CWE 항목 사이의 관계를 그래프구조로 표현함



CWE 구조



CWE의 상세 설명

- 각 CWE에 이름과 번호(ID)를 부여

- ID: 416
- Abstraction: Variant
- Status: Stable

CWE-416: Use After Free

Weakness ID: 416
Abstraction: Variant
Structure: Simple

Status: Stable

- **Weakness (Abstraction)**

- Weakness Pillar: 더 이상 추상적으로 만들 수 없는 가장 높은 수준의 결함
- Weakness Class: 특정 언어 또는 기술에 연관되지 않은 범위가 넓고 일반화된 결함
- Weakness Base: Variant보다 포괄적이고 Class보다 구체적인 정보 제공
- Weakness Variant: 특정 언어 또는 기술에 연관된 세밀한 정보를 제공함

CWE의 상세설명

- Compound Element





- 두 개 이상으로 구성된 CWE의 묶음, 복합 요소를 의미함

- Category

- 결함의 공통된 특성에 따라서 CWE 항목들의 집합으로 구성

Weakness ID: 61

Abstraction: Compound

Nature	Type	ID
Requires		362
Requires		340
Requires		386
Requires		732










CWE CATEGORY: 7PK - Security Features

Category ID: 254

▼ Summary

Software security is not security software. Here we're concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management.

▼ Membership

Nature	Type	ID	Name
MemberOf		700	Seven Pernicious Kingdoms
HasMember		256	Unprotected Storage of Credentials
HasMember		258	Empty Password in Configuration File
HasMember		259	Use of Hard-coded Password
HasMember		260	Password in Configuration File
HasMember		261	Weak Encoding for Password
HasMember		272	Least Privilege Violation
HasMember		284	Improper Access Control
HasMember		285	Improper Authorization
HasMember		330	Use of Insufficiently Random Values
HasMember		359	Exposure of Private Personal Information to an Unauthorized Actor
HasMember		798	Use of Hard-coded Credentials

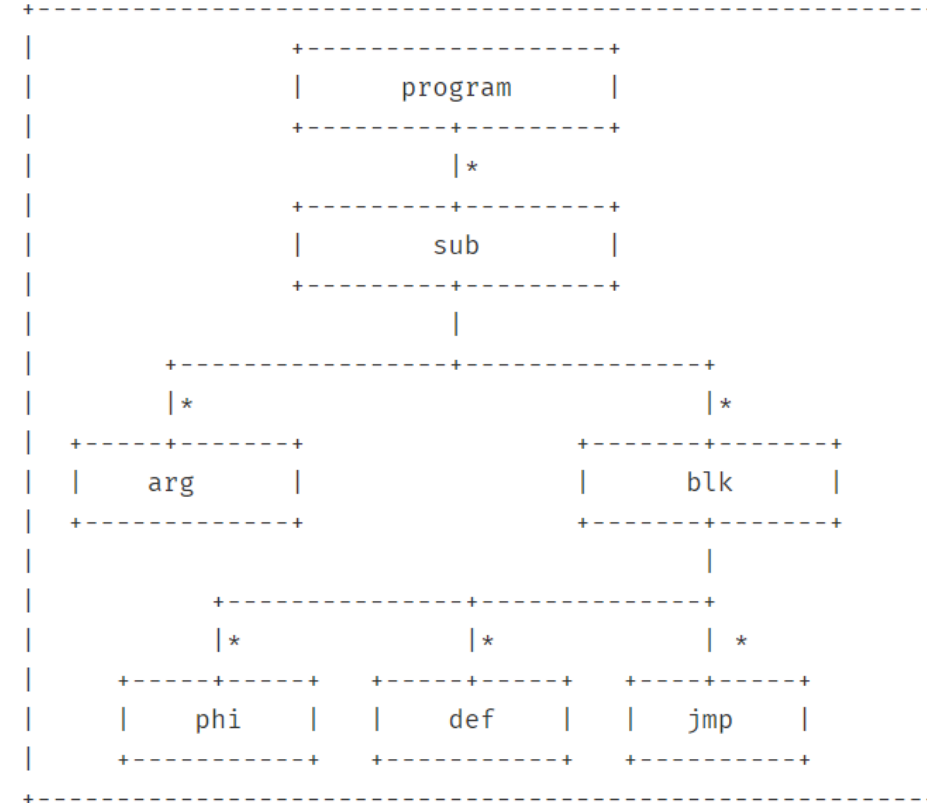
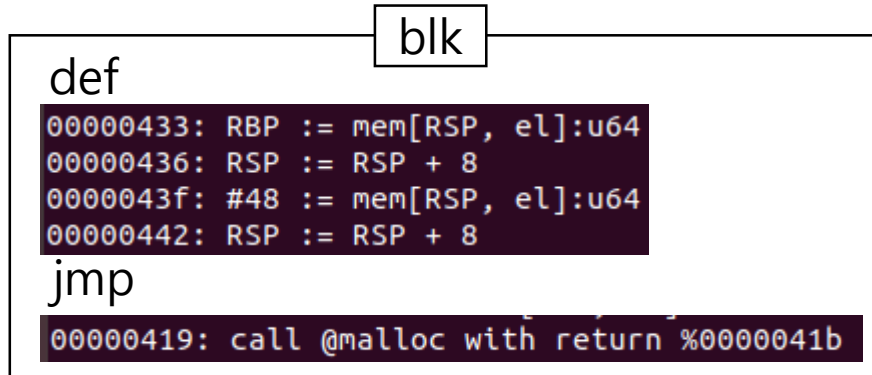
CWE 리스트

- CWE-125: Out-of-bounds Read → **Weakness Base**
- CWE-190: Integer Overflow or Wraparound → **Weakness Base**
- CWE-269: Improper Privilege Management → **Weakness Class**
- CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition') → **Weakness Class**
- CWE-264: Permissions, Privileges, and Access Controls → **Category (Obsolete)**
- CWE-284: Improper Access Control → **Weakness Pillar**
- CWE-254: 7PK - Security Features → **Category**

CWE 탐지 방법

• BAP IR코드를 통한 탐지

- BAP을 이용해서 바이너리를 IR로 변화시킬 수 있음
- BAP 모듈을 이용해서 IR에서 세부정보를 추출할 수 있음
- Ocaml언어로 구현해야 함



CWE 탐지 방법

```
000001a4: sub main(main_argc, main_argv, main_result)
000001c6: main_argc :: in u32 = RDI
000001c7: main_argv :: in out u64 = RSI
000001c8: main_result :: out u32 = RAX
0000016c:
0000016d: v175 := RBP
0000016e: RSP := RSP - 8
0000016f: mem := mem with [RSP, el]:u64 <- v175
00000170: RBP := RSP
00000172: RSP := RSP - 0x30
00000179: mem := mem with [RBP + 0xFFFFFFFFFFFFDC, el]:u32 <- low:32[RDI]
0000017a: mem := mem with [RBP + 0xFFFFFFFFFFFFD0, el]:u64 <- RSI
0000017b: RAX := mem[FS_BASE + 0x28, el]:u64
0000017c: mem := mem with [RBP + 0xFFFFFFFFFFFFF8, el]:u64 <- RAX
0000017d: RAX := 0
00000184: mem := mem with [RBP + 0xFFFFFFFFFFFFEC, el]:u32 <- 0x7B
00000185: mem := mem with [RBP + 0xFFFFFFFFFFFFF0, el]:u32 <- 0xEA
00000186: mem := mem with [RBP + 0xFFFFFFFFFFFFF4, el]:u32 <- 0x159
00000187: RAX := pad:64[mem[RBP + 0xFFFFFFFFFFFFF8, el]:u32]
00000188: mem := mem with [RBP + 0xFFFFFFFFFFFFEC, el]:u32 <- low:32[RAX]
00000189: RAX := 0
0000018a: RDX := mem[RBP + 0xFFFFFFFFFFFFF8, el]:u64
0000018b: RDX := RDX ^ mem[FS_BASE + 0x28, el]:u64
00000191: ZF := 0 = RDX
00000197: when ZF goto %00000192
00000196: goto %00000193
```

Target IR

CWE-416: Use After Free

Free이후 메모리접근을 확인하여
0값을 집어 넣는지 확인

```
open Core_kernel
open Bap.Std

let name = "CWE125"
let version = "0.1"

let collect_int_value = Exp.fold ~init:[] (object
  inherit [word list] Exp.visitor
  method! enter_int x addr = x :: addr
end)

let check_cwe program _ _ _ =
  let sub_t = Term.enum sub_t program in
  Seq.iter sub_t ~f:(fun sub ->
    if Bap.Std.Sub.name sub = "main" then
      let blk_t = Term.enum blk_t sub in
      Seq.iter blk_t ~f:(fun blk ->
        let def_t = Term.enum def_t blk in
        Seq.iter def_t ~f:(fun def ->
          let rhs = Def.rhs def in
          let int_values = collect_int_value rhs in
          List.iter int_values ~f:(fun x -> print_endline(Bitvector.to_string(x)))
        )
      )
    )
  )
```

Parsing Code

```
dongmin@ubuntu:~/BAP/cwe_checker$ cwe_checker test_a
INFO: Using standard configuration...
mem[RBP + 0xFFFFFFFFFFFFE0, el]:u64
Dangling Pointer
mem with [RBP + 0xFFFFFFFFFFFFF0, el]:u64 <- 0
No Dangling Pointer
```

Result

CWE 탐지 방법

• BAP Primus를 통한 탐지

- Primus는 프로그램을 Emulate할 수 있음
- 프로그램 실행 중 다양한 **Observation**을 제공하여 프로그램 실행 중에 발생하는 모든 이벤트를 추적할 수 있는 기능을 제공함
- **Observation**이 발생할 때마다 신호를 수신하고 언어를 이용해서 어떠한 규칙을 적용시킬 수 있음
- Lisp 언어를 이용해야 함

```
val clock: clock이 값을 변경할 때 발생
val pc_change: 주소의 코드가 실행될 때 발생
val loading: 메모리에서 주소의 값을 로드하기 전 발생
val loaded: 메모리주소가 로드된 후 발생
val storing: 값이 주소에 저장되기 전에 발생
val stored: 값이 주소에 저장된 후 발생
val reading: environment에서 값을 읽기전 발생
val read: 변수가 다른 값으로 평가될 때 발생
val writing: 변수에 값이 쓰여지기 전 발생
val written: 변수에 값이 쓰여진 후 발생
val jumping: Jump하기 직전에 발생
...
...
```

(Observation)

CWE 탐지 방법

```
000001a4: sub main(main_argc, main_argv, main_result)
000001c6: main_argc :: in u32 = RDI
000001c7: main_argv :: in out u64 = RSI
000001c8: main_result :: out u32 = RAX
0000016c:
0000016d: v175 := RBP
0000016e: RSP := RSP - 8
0000016f: mem := mem with [RSP, el]:u64 <- v175
00000170: RBP := RSP
00000172: RSP := RSP - 0x30
00000179: mem := mem with [RBP + 0xFFFFFFFFFFFFDC, el]:u32 <- low:32[RDI]
0000017a: mem := mem with [RBP + 0xFFFFFFFFFFFFD0, el]:u64 <- RSI
0000017b: RAX := mem[FS_BASE + 0x28, el]:u64
0000017c: mem := mem with [RBP + 0xFFFFFFFFFFFFF8, el]:u64 <- RAX
0000017d: RAX := 0
00000184: mem := mem with [RBP + 0xFFFFFFFFFFFFEC, el]:u32 <- 0x7B
00000185: mem := mem with [RBP + 0xFFFFFFFFFFFFF0, el]:u32 <- 0xEA
00000186: mem := mem with [RBP + 0xFFFFFFFFFFFFF4, el]:u32 <- 0x159
00000187: RAX := pad:64[mem[RBP + 0xFFFFFFFFFFFFF8, el]:u32]
00000188: mem := mem with [RBP + 0xFFFFFFFFFFFFEC, el]:u32 <- low:32[RAX]
00000189: RAX := 0
0000018a: RDX := mem[RBP + 0xFFFFFFFFFFFFF8, el]:u64
0000018b: RDX := RDX ^ mem[FS_BASE + 0x28, el]:u64
00000191: ZF := 0 = RDX
00000197: when ZF goto %00000192
00000196: goto %00000193
```

Primus Machine
(Emulate)

Run

```
(machine-fork (1 2))
(machine-switch (2 1))
(leave-blk %00000371)
(leave-term %00000371)
(leave-pos (blk %00000371))
(clock 719)
(jumping (1:1u#1604 0x520:64u#1605))
(linker-exec %%0000081d)
(enter-exp RDI)
(reading RDI)
(read (RDI 0x370:64u#1560))
(leave-exp RDI)
(call (malloc 0x370))
(lisp-call (malloc 0x370))
(visited-blocks (3 33))
(const 0:64u#1607)
(const 0:1u#1608)
(clock 720)
(binop ((= 0x370:64u#1560 0:64u#1607) 0:1u#1609))
(clock 721)
(lisp-primitive (= 0x370 0 0))
(clock 722)
(binop ((= 0:1u#1609 0:1u#1610) 1:1u#1611))
```

Interpreter
(Observation)

Signal

```
(defmethod call (name arg)
  (when (= name 'malloc)
    (incident-report 'malloc_was_called)
  )
)
```

Lisp(Rule)

execute

```
(call (malloc 0x370))
(incident (malloc_was_called))
```








Result

CWE-125: Out-of-bounds Read

• Description

- Abstraction: Weakness Base
- 설명: 정해진 버퍼 이외의 공간에서 데이터를 읽을 경우 해당 취약점 발생
- 공격자가 다른 메모리 위치에 있는 중요한 정보를 읽거나 충돌을 일으킬 수 있음

• Relationships (하위 취약점 2개)

<i>Nature</i>	<i>Type</i>	<i>ID</i>	<i>Name</i>
ChildOf		119	Improper Restriction of Operations within the Bounds of a Memory Buffer
ParentOf		126	Buffer Over-read
ParentOf		127	Buffer Under-read
CanFollow		822	Untrusted Pointer Dereference
CanFollow		823	Use of Out-of-range Pointer Offset
CanFollow		824	Access of Uninitialized Pointer
CanFollow		825	Expired Pointer Dereference

CWE-125: Out-of-bounds Read

• Sample Code

```
int main(){  
  
    int *arr;  
    int size = 4;  
    int ret = 0;  
  
    arr = (int *)malloc(sizeof(int) * size);  
    arr[0] = 0xAA;  
    arr[1] = 0xBB;  
    arr[2] = 0xCC;  
    arr[3] = 0xDD;  
    arr[3] = arr[-1];  
  
    printf("PRINT: %d \n", arr[3]);  
}
```

int(4byte) * 4
만큼 메모리 할당

arr[-1]에 접근

결과: `dongmin@ubuntu:`
`PRINT: 0`

Primus
Observation

```
(call (malloc 0x10))  
(incident (malloc was called))  
(call-return (malloc 0x10 0x201028))
```

Ptr: 0x201028
Size: 0x10

```
(read (RAX 0x201028:64u#2044))  
(loaded (0x201027 64u#2046 0:8u#1783))  
(read (RAX 0x201028:64u#2044))  
(loaded (0x201026:64u#2048 0:8u#1724))  
(read (RAX 0x201028:64u#2044))  
(loaded (0x201025:64u#2051 0:8u#1665))  
(read (RAX 0x201028:64u#2044))  
(loaded (0x201024:64u#2054 0x10:8u#1606))
```

Ptr: 0x201027~0x201024
해당 메모리주소 로드

CWE-125: Out-of-bounds Read

• 탐지전략

- Malloc에서 pointer와 Size를 알 수 있음
- 배열에 접근할 때 RAX로 malloc ptr을 가져온 후 메모리주소를 load하는 패턴

```
arr[3] = arr[-1];
```

```
(read (RAX 0x201028:64u#2124))  
(binop ((+ 0x201028:64u#2124 0xFFFFFFFFFFFFFFFF:64u#2125) 0x201027:64u#2126))  
(loaded (0x201027:64u#2126 0:8u#1788))  
(read (RAX 0x201028:64u#2124))  
(binop ((+ 0x201028:64u#2124 0xFFFFFFFFFFFFFFFFE:64u#2127) 0x201026:64u#2128))  
(loaded (0x201026:64u#2128 0:8u#1729))  
(read (RAX 0x201028:64u#2124))  
(binop ((+ 0x201028:64u#2124 0xFFFFFFFFFFFFFFFFD:64u#2130) 0x201025:64u#2131))  
(loaded (0x201025:64u#2131 0:8u#1670))  
(read (RAX 0x201028:64u#2124))  
(binop ((+ 0x201028:64u#2124 0xFFFFFFFFFFFFFFFFC:64u#2133) 0x201024:64u#2134))  
(loaded (0x201024:64u#2134 0x10:8u#1611))
```

- 1) RAX를 arr pointer로 변경
- 2) Pointer값을 접근할 메모리 영역으로 변경한 뒤
- 3) 메모리 Load

CWE-125: Out-of-bounds Read

• 탐지전략

malloc 정보 식별

```
(call (malloc 0x10))  
(call-return (malloc 0x10 0x201028))
```

- 1) Malloc 함수 식별
- 2) Pointer부터 Size만큼
메모리영역 정보 저장
0x201028 ~ 0x201038

Malloc pointer 식별

```
(read (RAX 0x201028:64u#5551))  
(incident (malloc_pointer))
```

- 1) read 동작에서 RAX를 식별
- 2) RAX데이터와 저장된 malloc
메모리영역에서 가장 낮은
값(Pointer)과 비교
- 3) 일치하다면 readflag를 킴

loaded 메모리 영역확인

```
(loaded (0x201027:64u#5569 0:8u#3176))  
(read (*readflag* 1:64u#5567))S  
(read (*malloc-arena-end* 0x241024:64u#2808))  
(read (*malloc-arena-start* 0x201024:64u#2038))  
(incident (CWE_125))
```

- 1) Loaded 식별
- 2) Readflag가 1이면 저장된
메모리영역 바운더리를 비교
- 3) 저장한 영역을 벗어난
주소라면 Detect

CWE-125: Out-of-bounds Read

- CWE에서 제공하는 Sample Code

```
int getValueFromArray(int *array, int len, int index){  
    int value;  
    if(index < len){  
        value = array[index];  
    }  
    else{  
        printf("Value is: %d\n", array[index]);  
        value = -1;  
    }  
    return value;  
}
```

Index값을 배열의 크기보다 작은 지 확인하지만
음수 값은 확인하지 않음

CWE-125: Out-of-bounds Read

```
int getValueFromArray(int *array, int len, int index){
    int value;
    if(index < len){
        value = array[index];
    }
    else{
        printf("Value is: %d\n", array[index]);
        value = -1;
    }
    return value;
}

int main(){
    int *arr;
    int size = 4;
    int ret = 0;

    arr = (int *)malloc(sizeof(int) * size);
    arr[0] = 0xAA;
    arr[1] = 0xBB;
    arr[2] = 0xCC;
    arr[3] = 0xDD;
    ret = getValueFromArray(arr, size, -1);
    printf("PRINT: %d \n", ret);
}
```

(call (malloc 0x10))
(call-return (malloc 0x10 0x201028))



(read (RAX 0x201028:64u#8145))
(incident (malloc_pointer))



(loaded (0x201027:64u#12979 0:8u#3276))
(incident (CWE 125))
(loaded (0x201026:64u#13047 0:8u#3217))
(loaded (0x201025:64u#13079 0:8u#3158))
(loaded (0x201024:64u#12769 0x10:8u#3099))

탐지

CWE-125: Out-of-bounds Read

```
int getValueFromArray(int *array, int len, int index){
    int value;
    if(index < len){
        value = array[index];
    }
    else{
        printf("Value is: %d\n", array[index]);
        value = -1;
    }
    return value;
}

int main(){

    int *arr;
    int size = 4;
    int ret = 0;

    arr = (int *)malloc(sizeof(int) * size);
    arr[0] = 0xAA;
    arr[1] = 0xBB;
    arr[2] = 0xCC;
    arr[3] = 0xDD;
    ret = getValueFromArray(arr, size, -1);
    printf("PRINT: %d \n", ret);
}
```

```
(call (main 0 0x40000008))
(call (malloc 0x10))
(call-return (malloc 0x10 0x201028))
(incident (TAINT))
(call (getValueFromArray))
(incident (CWE_125))
(call-return (getValueFromArray 0x10))
(call (printf 0x822))
(call-return (printf 0x822 0xC7C760E3))
(call (deregister_tm_clones))
(call-return (deregister_tm_clones 0x201010))
(call-return (main 0 0x40000008 0x201010))
(call-return (deregister_tm_clones 0))
(call-return (main 0 0x40000008 0))
(call-return (deregister_tm_clones 0))
(call-return (main 0 0x40000008 0))
```

오탐없이 정확히 탐지

CWE-190: Integer Overflow or Wraparound

• Description

- Abstraction: Weakness Base
- 설명: 정수값이 관련 표현을 저장하기에 너무 큰값으로 증가할 때 발생
- 반복문 제어, 메모리 할당, 메모리 복사 시 보안상 문제를 발생시킬 수 있음

• Relationships (하위 취약점 없음)

<i>Nature</i>	<i>Type</i>	<i>ID</i>	<i>Name</i>
ChildOf	P	682	Incorrect Calculation
PeerOf	B	128	Wrap-around Error
CanPrecede	C	119	Improper Restriction of Operations within the Bounds of a Memory Buffer

CWE-190: Integer Overflow or Wraparound

- CWE에서 제공하는 Sample Code

```
int main(){  
    int num_imgs;  
    num_imgs = 0xFFFFFFFF;  
    num_imgs = num_imgs + 0xDD;  
    int table_ptr = (int *)malloc(sizeof(int) * num_imgs);  
}
```

0xFFFFFFFF + 0xDD의 결과로 Integer Overflow 발생
num_imgs는 **0xDC**가 된다.

Malloc의 결과로 예상보다 작은 값이 할당된다.

```
(call (malloc 0x370)  
(call-return (malloc 0x370 0x201028)))
```

Malloc에 의해 0x370만큼 할당됨
0xDC * int(4byte) = 0x370

CWE-190: Integer Overflow or Wraparound

• 구현된 탐지 모듈

- 함수를 호출할 때마다 호출 직전 Basic Block에서 곱셈 명령이 있는지 확인

```
let collect_multiplications = Exp.fold ~init:0 (object
  inherit [Int.t] Exp.visitor
  method! enter binop op o1 o2 binops = match op with
    | Bil.TIMES | Bil.LSHIFT -> binops + 1
    | _ -> binops
end)
```

• 현재 모듈의 한계점

- 호출직전의 Basic Block만을 검사하여 탐지함
- 실제로 오버플로우가 되는지 확인하지 않음
- 더하기 또는 빼기로 생기는 오버플로우는 탐지하지 못함

CWE-190: Integer Overflow or Wraparound

• 탐지전략

- 현재 Malloc에 인자로 들어가는 변수의 값을 추적해야 함

```
(int *)malloc(sizeof(int) * num_imgs);
```

→ 처음부터 모든 변수를 기록하고 검사하기는 어려움
Malloc함수에 나오는 인자를 역추적해야 함

- 해당 변수가 Integer overflow가 발생하는지 검사해야 함

```
num_imgs = num_imgs + 0xDD;
```

→ 변수의 덧셈과 곱셈을 확인하여, 본래의 값보다 작은 값이 저장되거나 음수 값이 되는지 확인해야 함

CWE-190: Integer Overflow or Wraparound

• 탐지전략 (Primus)

- 포인터를 역추적하여 변수 추적 가능

```
(stored (0x3FFFFFFD8:64u#1346 0xFF:8u#1347))  
(stored (0x3FFFFFFD9:64u#1343 0xFF:8u#1344))  
(stored (0x3FFFFFFDA:64u#1340 0xFF:8u#1341))  
(stored (0x3FFFFFFDB:64u#1337 0xFF:8u#1338))  
(stored (0x3FFFFFFD8:64u#1408 0xDC:8u#1422))
```

0x3FFFFFFD8: **0xDC** → 0x3FFFFFFD8: **0xFFFFFFFF**

- Primus의 binop를 확인하여 Integer overflow 확인 가능

```
(binop ((+ 0xFFFFFFFF:32u#1419 0xDD:32u#1420) 0xDC:32u#1421))  
(stored (0x3FFFFFFD8:64u#1408 0xDC:8u#1422))  
(stored (0x3FFFFFFD9:64u#1392 0:8u#1406))  
(stored (0x3FFFFFFDA:64u#1376 0:8u#1390))  
(stored (0x3FFFFFFDB:64u#1360 0:8u#1374))
```

Overflow
0xFFFFFFFF + 0xDD = 0xDC

- 그러나 Lisp에서 binop 신호를 받지 못해 문제해결 중

```
(defmethod binop (_  
  (incident-report 'print)  
)
```









```
dongmin@ubuntu:~/ .opam/4.09.0/BAP/toolkit$  
|  
> (defmethod binop (_  
>   (incident-report 'print)  
> )  
> ^
```

CWE-269: Improper Privilege Management

• Description

- Abstraction: Weakness Class
- 설명: 사용자에게 대한 권한을 잘못 설정하거나 확인하지 않아 발생
- 관리자 권한은 악용될 소지가 있으므로, 권한을 최소화할 필요가 있음

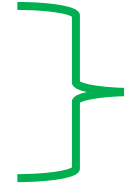
• Relationships (하위 취약점 8개)

<i>Nature</i>	<i>Type</i>	<i>ID</i>	<i>Name</i>
ChildOf	[P]	284	Improper Access Control
ParentOf		250	Execution with Unnecessary Privileges
ParentOf		266	Incorrect Privilege Assignment
ParentOf		267	Privilege Defined With Unsafe Actions
ParentOf		268	Privilege Chaining
ParentOf		270	Privilege Context Switching Error
ParentOf		271	Privilege Dropping / Lowering Errors
ParentOf		274	Improper Handling of Insufficient Privileges
ParentOf		648	Incorrect Use of Privileged APIs

CWE-269: Improper Privilege Management

• Sample Code

- CWE-266: Incorrect Privilege Assignment
→ 관리자 권한 획득
- CWE-250: Execution with Unnecessary Privileges
- CWE-271: Privilege Dropping / Lowering Errors
→ chroot() 이후 관리자 권한삭제 하지 않음
- CWE-267: Privilege Defined With Unsafe Actions
- CWE-268: Privilege Chaining
- CWE-270: Privilege Context Switching Error
- CWE-274: Improper Handling of Insufficient Privileges



```
seteuid(0);  
/* do some stuff */  
seteuid(getuid());
```



```
chroot(APP_HOME);  
chdir("/");  
FILE* data = fopen(argv[1], "r+");  
...
```



Sample Code 없음

CWE-269: Improper Privilege Management

• CWE-266 탐지전략

- seteuid로 점프하기전 RDI에서 0인 값을 확인
- getuid 호출 이후 곧바로 seteuid 호출

```
seteuid(0);  
/* do some stuff */  
seteuid(getuid());
```

seteuid(0) : 루트권한으로 설정

seteuid(getuid()) : 실제 유저 권한으로 설정

```
000003ac: #41 := RBP  
000003af: RSP := RSP - 8  
000003b2: mem := mem with [RSP, e1]:u64 <- #41  
000003b9: RBP := RSP  
000003c0: RDI := 0  
000003c9: RSP := RSP - 8  
000003cc: mem := mem with [RSP, e1]:u64 <- 0x6E8  
000003cf: call @seteuid with return %000003d1
```

seteuid(0)

```
000003ee:  
000003f5: RSP := RSP - 8  
000003f8: mem := mem with [RSP, e1]:u64 <- 0x6FE  
000003fb: call @getuid with return %000003fd  
000003fd:  
00000402: RDI := pad:64[low:32[RAX]]  
0000040b: RSP := RSP - 8  
0000040e: mem := mem with [RSP, e1]:u64 <- 0x705  
00000411: call @seteuid with return %00000413
```

seteuid(getuid())

CWE-269: Improper Privilege Management

• CWE-250, 271 탐지전략

- Chroot 호출을 탐지
 - Chroot는 관리자권한일때만 실행
- 0이 아닌 setuid() 호출이 있는지 검사

```
chroot(APP_HOME);  
chdir("/");  
FILE* data = fopen(argv[1], "r+");  
...
```

chroot() : root 디렉토리 변경

```
000003ac: #41 := RBP  
000003af: RSP := RSP - 8  
000003b2: mem := mem with [RSP, e1]:u64 <- #41  
000003b9: RBP := RSP  
000003c0: RDI := 0x794  
000003c9: RSP := RSP - 8  
000003cc: mem := mem with [RSP, e1]:u64 <- 0x6EA  
000003cf: call @chroot with return %000003d1
```

chroot

```
0000041e:  
00000423: RDI := 0x3E8  
0000042c: RSP := RSP - 8  
0000042f: mem := mem with [RSP, e1]:u64 <- 0x751  
00000432: call @setuid with return %00000434
```

setuid(1000)
0x3EB = 1000

CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')

• Description

- Abstraction: Weakness Class
- 설명: 해당 취약점은 공유리소스에 대해서 하나의 프로세스만 접근해야 하지만 동시에 공유 리소스를 접근할 수 있을 경우에 발생

• Relationships (하위 취약점 6개)

<i>Nature</i>	<i>Type</i>	<i>ID</i>	<i>Name</i>
ChildOf	P	691	Insufficient Control Flow Management
ParentOf	B	364	Signal Handler Race Condition
ParentOf	B	366	Race Condition within a Thread
ParentOf	B	367	Time-of-check Time-of-use (TOCTOU) Race Condition
ParentOf	B	368	Context Switching Race Condition
ParentOf	B	421	Race Condition During Access to Alternate Channel
ParentOf	B	1223	Race Condition for Write-Once Attributes
CanFollow	C	662	Improper Synchronization

CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')

• CWE-362 Sample Code

- 다음 함수에서는 공유 리소스에 대한 작업을 수행하기 위해 Mutex를 획득하려고 시도 함
- 코드는 pthread_mutex_lock()의 반환 값에서 오류를 확인하지 않음
- Pthread_mutex_lock()이 어떠한 이유로 Mutex를 획득할 수 없는 경우 해당 함수는 race condition에 도입하여 부적절한 동작을 발생시킬 수 있음

```
void f(pthread_mutex_t *mutex) {  
    pthread_mutex_lock(mutex);  
  
    /* access shared resource */  
  
    pthread_mutex_unlock(mutex);  
}
```

CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')

• CWE-364: Signal Handler Race Condition

- SIGHUP을 처리하기 위해 sh()이 호출
- 첫번째 sh() 호출은 global1이 free되는 시점에 도달
- SIGTERM으로 두번째 sh()호출
- 두번째 sh()는 global1을 또다시 free
- 결과적으로 double-free 취약점을 가지게 됨

```
void *global1, *global2;
char *what;
void sh (int dummy) {
    syslog(LOG_NOTICE, "%s\n", what);
    free(global2);
    free(global1);
    /* Sleep statements added to expand timing window for race condition */

    sleep(10);
    exit(0);
}

int main (int argc, char* argv[]) {
    what=argv[1];
    global1=strdup(argv[2]);
    global2=malloc(340);
    signal(SIGHUP, sh);
    signal(SIGTERM, sh);
    /* Sleep statements added to expand timing window for race condition */

    sleep(10);
    exit(0);
}
```

CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')

- **CWE-366: Race Condition within a Thread**

- 다중 스레드 환경에서 동시에 해당 리소스를 사용하는 경우 문제가 발생할 수 있음
- Mutex와 같은 locking 함수를 사용해야 함

```
int foo = 0;
int storenum(int num) {
    static int counter = 0;
    counter++;
    if (num > foo) foo = num;
    return foo;
}
```

CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')

- **CWE-367: Time-of-check Time-of-use Race Condition**

- 해당 코드는 lstat함수로 파일의 정보를 확인한 후 해당 내용을 업데이트 하는 코드
- Printf의 실행시간 때문에 lstat함수와 Check시간 중간에 파일이 업데이트되었다면, 의도하지 않게 데이터가 변경될 수 있음

```
struct stat *sb;  
...  
lstat(".",sb); // it has not been updated since the last time it was read  
printf("stated file\n");  
if (sb->st_mtimespec==...){  
    print("Now updating things\n");  
    updateThings();  
}
```

CWE-264: Permissions, Privileges, and Access Controls

• Description

- 설명: 이 카테고리의 Weakness는 접근 제어를 수행하는데 사용되는 권한과 기타 보안기능의 관리와 관련 되어 있음 → **현재 폐쇄된 취약점**
- Abstraction: Category

CWE CATEGORY: Permissions, Privileges, and Access Controls

Category ID: 264Status: Obsolete

▼ Summary

Weaknesses in this category are related to the management of permissions, privileges, and other security features that are used to perform access control.

▼ Membership

Nature	Type	ID	Name
MemberOf	<input checked="" type="checkbox"/>	635	Weaknesses Originally Used by NVD from 2008 to 2016

▼ Notes

Maintenance

This entry heavily overlaps other categories and has been marked obsolete.

Maintenance

This entry is a Category, but various sources map to it anyway despite CWE guidance that Categories should not be mapped. Future mappings should use an appropriate weakness going forward.

▼ References

[REF-7] Michael Howard and David LeBlanc. "Writing Secure Code". Chapter 7, "How Tokens, Privileges, SIDs, ACLs, and Processes Relate" Page 218. 2nd Edition. Microsoft Press. 2002-12-04. <<https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223>>.

▼ Content History

▼ Submissions

Submission Date	Submitter	Organization
2006-07-19	PLOVER	

▶ Modifications

CWE-284: Improper Access Control

- **Description**

- 설명: 권한이 없는 사용자에게 리소스에 대한 접근을 적절하게 제한하지 않음
- Abstraction: **Pillar**

- **Relationships**

- 해당 취약점은 Weakness중에서 가장 추상적인 CWE 유형이기 때문에 구체적인 설명이나 샘플코드가 없음
- 접근제어와 **관련한 하위 취약점 27**개를 가지고 있음

CWE-284: Improper Access Control













<i>Nature</i>	<i>Type</i>	<i>ID</i>	<i>Name</i>
MemberOf	V	1000	Research Concepts
ParentOf	G	269	Improper Privilege Management
ParentOf	G	282	Improper Ownership Management
ParentOf	G	285	Improper Authorization
ParentOf	G	286	Incorrect User Management
ParentOf	G	287	Improper Authentication
ParentOf	B	346	Origin Validation Error
ParentOf	G	923	Improper Restriction of Communication Channel to Intended Endpoints
ParentOf	V	942	Permissive Cross-domain Policy with Untrusted Domains
ParentOf	B	1191	Exposed Chip Debug and or Test Interface With Insufficient Access Control
ParentOf	B	1220	Insufficient Granularity of Access Control
ParentOf	B	1224	Improper Restriction of Write-Once Bit Fields
ParentOf	B	1231	Improper Implementation of Lock Protection Registers
ParentOf	B	1242	Inclusion of Undocumented Features or Chicken Bits
ParentOf	B	1252	CPU Hardware Not Configured to Support Exclusivity of Write and Execute Operations
ParentOf	B	1256	Hardware Features Enable Physical Attacks from Software
ParentOf	B	1257	Improper Access Control Applied to Mirrored or Aliased Memory Regions
ParentOf	B	1259	Improper Protection of Security Identifiers
ParentOf	B	1260	Improper Handling of Overlap Between Protected Memory Ranges
ParentOf	B	1262	Register Interface Allows Software Access to Sensitive Data or Security Settings
ParentOf	B	1267	Policy Uses Obsolete Encoding
ParentOf	B	1268	Agents Included in Control Policy are not Contained in Less-Privileged Policy
ParentOf	B	1270	Generation of Incorrect Security Identifiers
ParentOf	B	1274	Insufficient Protections on the Volatile Memory Containing Boot Code
ParentOf	V	1275	Sensitive Cookie with Improper SameSite Attribute
ParentOf	B	1276	Hardware Block Incorrectly Connected to Larger System
ParentOf	B	1280	Access Control Check Implemented After Asset is Accessed
ParentOf	B	1283	Mutable Attestation or Measurement Reporting Data

CWE-254: 7PK – Security Features

• Description

- 설명: 이 카테고리의 Weakness는 인증, 접근제어, 기밀성, 암호화, 권한관리와 같은 취약점의 카테고리
- Category: 해당 취약점은 11가지 취약점유형을 포함

• Relationships

Nature	Type	ID	Name
MemberOf		700	Seven Pernicious Kingdoms
HasMember		256	Unprotected Storage of Credentials
HasMember		258	Empty Password in Configuration File
HasMember		259	Use of Hard-coded Password
HasMember		260	Password in Configuration File
HasMember		261	Weak Encoding for Password
HasMember		272	Least Privilege Violation
HasMember		284	Improper Access Control
HasMember		285	Improper Authorization
HasMember		330	Use of Insufficiently Random Values
HasMember		359	Exposure of Private Personal Information to an Unauthorized Actor
HasMember		798	Use of Hard-coded Credentials