

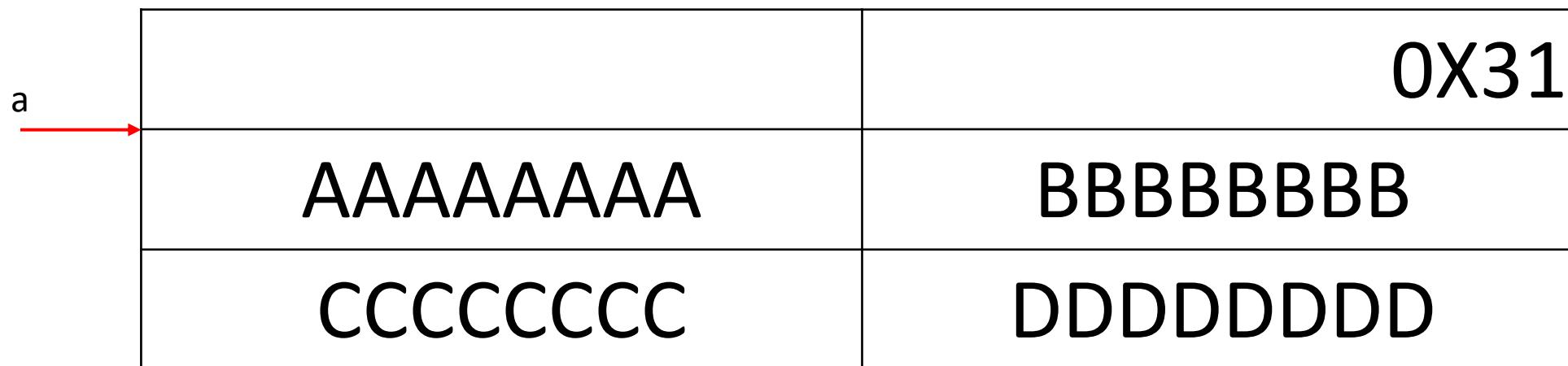
UAF

CyKor 정수환

<https://github.com/1nTEGER-c/uaf>

UAF (Use - After - Free)

`a = malloc(0x20)`



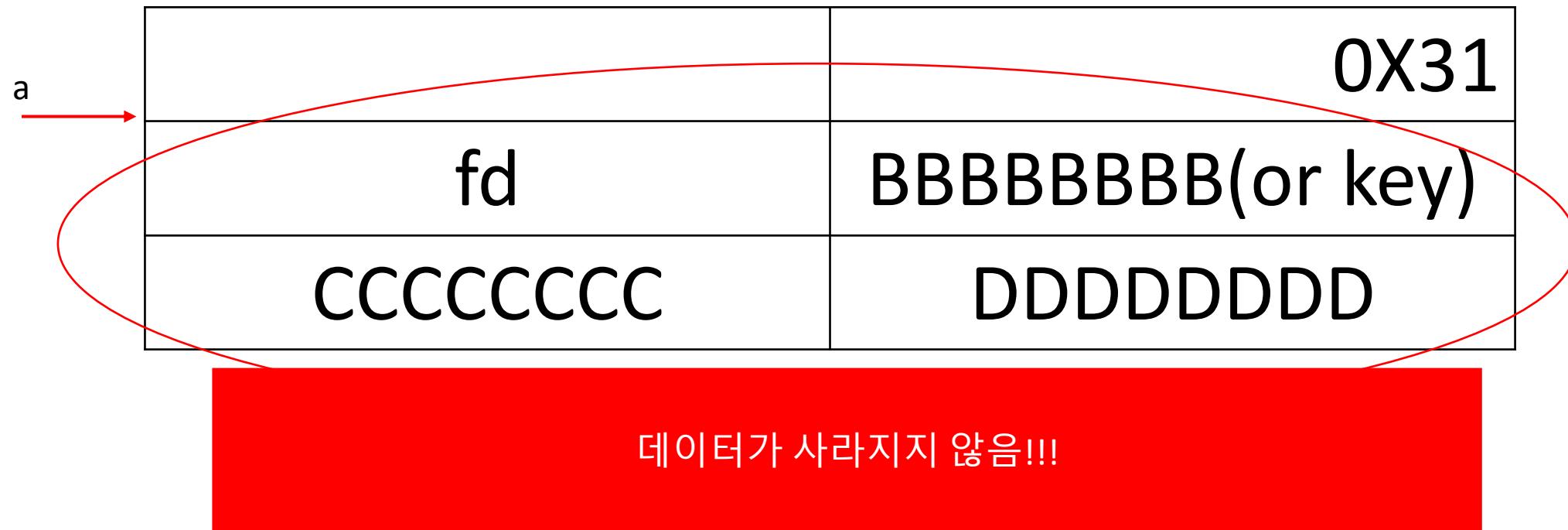
UAF (Use - After - Free)

`a = malloc(0x20) ->free()`

	0X31
fd	BBBBBBBB(or key)
CCCCCCCC	DDDDDDDD

UAF (Use - After - Free)

`a = malloc(0x20) ->free()`



UAF (Use - After - Free)

`a = malloc(0x20) ->free()`

여기서 `malloc(0x20)`을 하면??



cccccccccccc DDDDDDDDDDD

The diagram shows a memory layout with two adjacent blocks. The first block is 8 bytes long and contains the hex value 'cccccccc'. The second block is 8 bytes long and contains the hex value 'dd'. A red line starts from the left edge of the first block and points to a red box at the bottom. Another red line starts from the right edge of the second block and points to the same red box at the bottom. This visualizes that the freed memory is still at its original address, and a new allocation has been made immediately after it.

데이터가 사라지지 않음!!!

UAF (Use - After - Free)

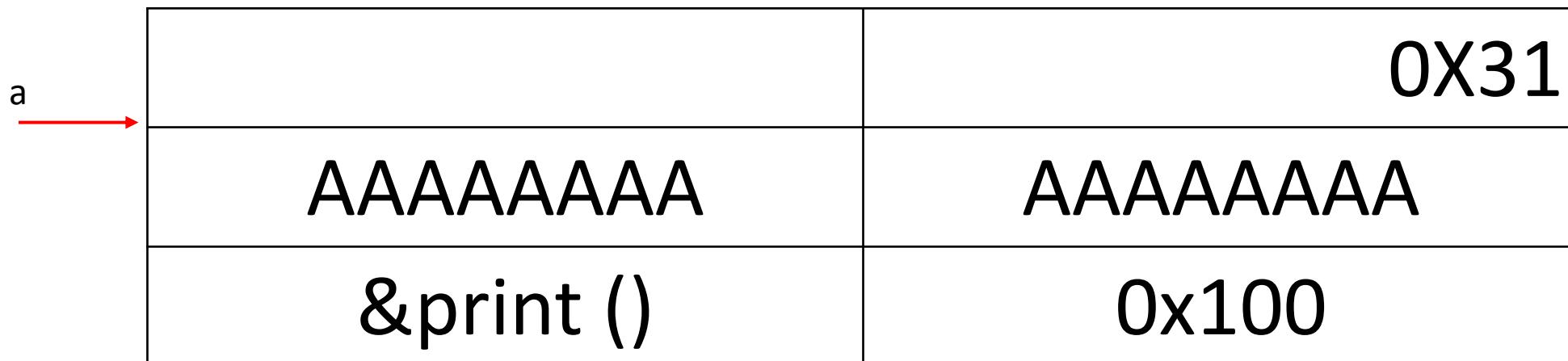
a = malloc(0x20) ->free() -> b = malloc(0x20) : 아래와 동일한 곳에 할당..!

a , b	0X31
fd	BBBBBBBB(or key)
CCCCCCCC	DDDDDDDD

UAF (Use - After - Free)

```
struct Name{  
    char name[0x10];  
    void(*print_name)(void*);  
    score();  
}
```

```
Name * a = malloc(0x20)
```



a	0X31
AAAAAA	AAAAAA
&print ()	0x100

UAF (Use - After - Free)

`a = malloc(0x20) ->free() -> b = malloc(0x20)`

<code>a , b</code>	
<code>AAAAAA</code>	<code>0X31</code>
<code>&system</code>	<code>0x100</code>

UAF (Use - After - Free)

`a = malloc(0x20) ->free() -> b = malloc(0x20)`

<code>a, b</code>	
<code>AAAAAA</code>	<code>0X31</code>
<code>&system</code>	<code>AAAAAA</code>

만약, 여기서 `a`를 이용해서 `print_name`함수를 호출한다면??

UAF (Use - After - Free)

`a = malloc(0x20) ->free() -> b = malloc(0x20)`

<code>a, b</code>	
<code>AAAAAA</code>	<code>0X31</code>
<code>&system</code>	<code>AAAAAA</code>

만약, 여기서 a를 이용해서 `print_name`함수를 호출한다면??

→ `system` 호출 가능..!

Double Free

- Free -> Free ??

```
a = malloc(0x30);  
free(a);  
free(a);
```

Double Free

- 16.04 (ERROR)
 - `a = malloc(0x30);`
 - `free(a);`
 - `free(a);`
- 18.04 (OK)
 - `a = malloc(0x30);`
 - `free(a);`
 - `free(a);`
- 20.04 (OK)
 - `a = malloc(0x30);`
 - `free(a);`
 - `free(a);`

Double Free

- 16.04 (ERROR)
 - a = malloc(0x30);
 - free(a);
 - **free(b);**
 - free(a);
- 18.04 (OK)
 - a = malloc(0x30);
 - free(a);
 - free(a);
- 20.04 (OK)
 - a = malloc(0x30);
 - free(a);
 - a의 key 자리 overwrite
 - free(a);

Tcache - dup

```
a = malloc(0x20);  
free(a);
```



tcache_bin : 0x1000

Tcache - dup

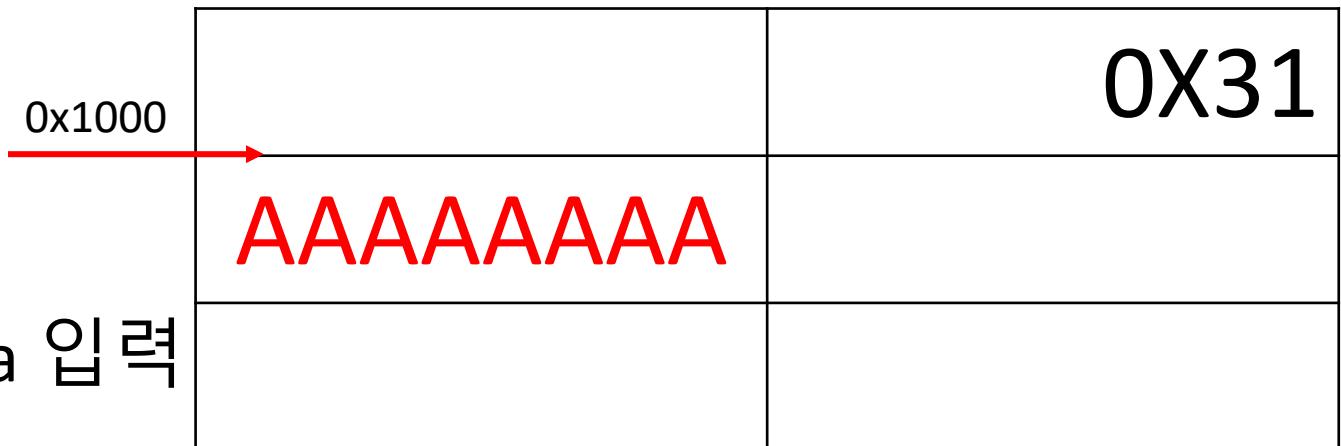
```
a = malloc(0x20);  
free(a);  
free(a);
```



tcache_bin : `0x1000 -> 0x1000`

Tcache - dup

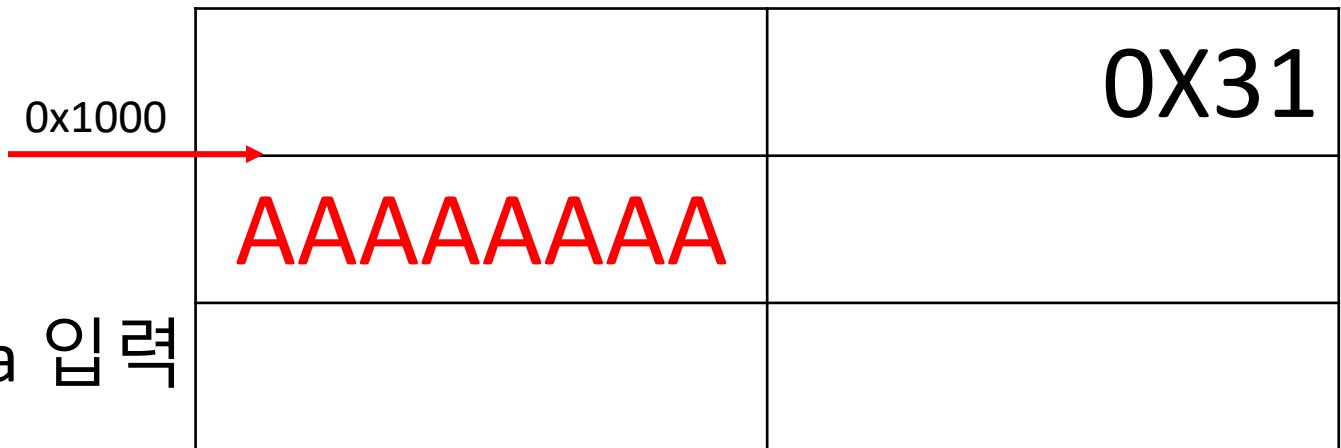
```
a = malloc(0x20);
free(a);
free(a);
b = malloc(0x20); //0aaaaaaaaa 입력
```



tcache_bin : 0x1000 -> 0xAAAAAAA

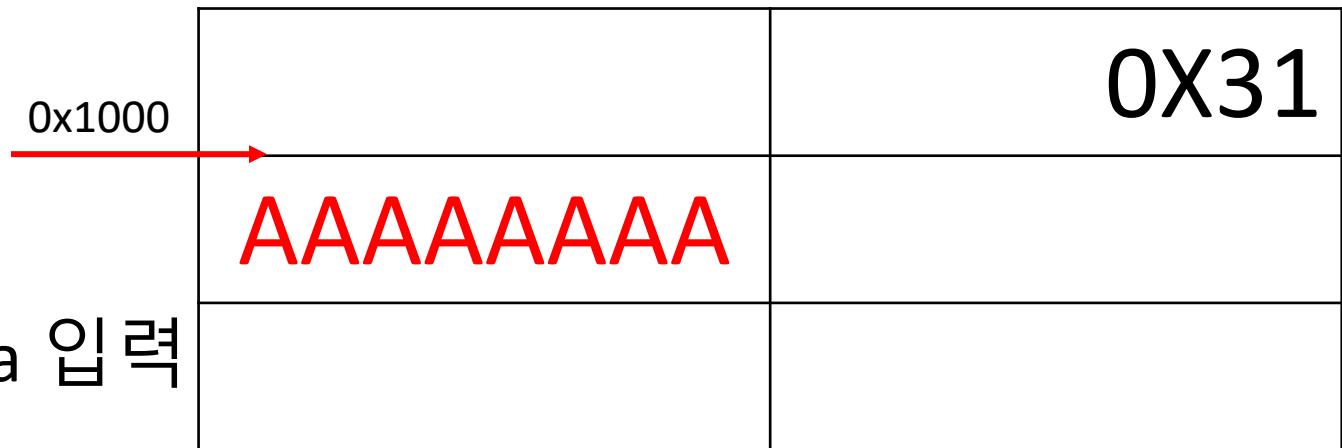
Tcache - dup

```
a = malloc(0x20);
free(a);
free(a);
b = malloc(0x20); //0aaaaaaaaa 입력
c = malloc(0x20);
```



Tcache - dup

```
a = malloc(0x20);
free(a);
free(a);
b = malloc(0x20); //0aaaaaaaaa 입력
c = malloc(0x20);
d = malloc(0x20); // 0xAAAAAAA에 chunk 할당..!
```



tcache_bin :

- 어디에 ? 무엇으로 ? 덮어야 할까요

Hook (__malloc_hook / __free_hook)

oneshot_gadget

과제

- git에 함께 첨부되어 있는 homework/tcache
18.04 / 20.04에서 각각 Exploit 해보기

(18.04가 더 쉬우니 18.04 먼저 하세요)

제출 메일 : pk2861@naver.com 기한 : 12/1 23:59

질문있으면 간통하세요