Lec 15: Type Confusion

IS561: Binary Code Analysis and Secure Software Systems

Sang Kil Cha

Type Confusion

Memory Corruption So Far

- Buffer overflows.
- Format string bugs.
- ... (other ways to corrupt memory?)

Type

A classification of data which tells the compiler or interpreter how the programmer intends to use the data[1].

[1] Excerpt from Wikipedia

Type Safety

Types prevent unintended errors.

```
 1 + "1";;
  ----^^^
```

```
error FS0001: The type ’string’ does not match the type ’int’
```

Type Confusion

Type confusion happens when the type-safety is violated.

Two main causes:

1. Unsafe casting.
2. Memory-safety bugs.

Similar, but different from ***weak typing***.

Weak vs. Strong Type System

- Weakly-typed language: PHP, JavaScript, etc.
- Strongly-typed language: F#, Haskell, OCaml, etc.

Type confusion happens a lot with weakly typed languages.

Weak (and Weird) Types in JavaScript

```
> 1 + "1"
’11’
```

```
> !!"false" == !!"true"
true
```

```
> "b" + "a" + +"a" + "a"
’baNaNa’
```

See more @ `https://github.com/denysdovhan/wtfjs`

Unsafe Casting

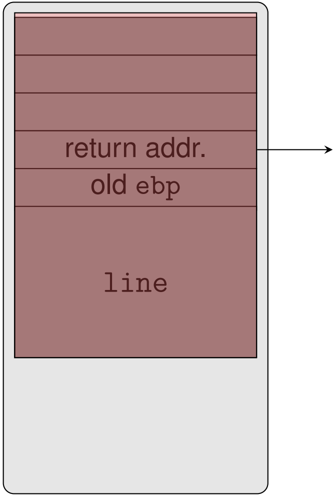
```c
char x = 0x42;
float* ptr = (float*) &x;
printf("%f\n", *ptr); // prints out a weird number
```

Memory Safety and Type Confusion

Any language supports ***type safety*** to some extent, but not every language is ***memory-safe***.

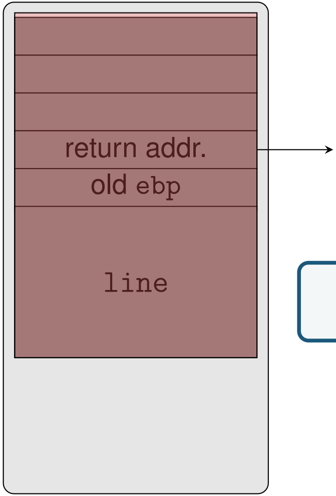
Can memory-safety break type-safety?

Example: Buffer Overflow



What was the intended type for this data?

Example: Buffer Overflow

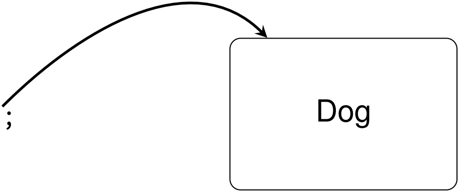


What was the intended type for this data?

C’s type-safety can be broken due to buffer overflows.

Type Confusion Example

```
Dog *d = (Dog*) ptr;
d->bark();
```

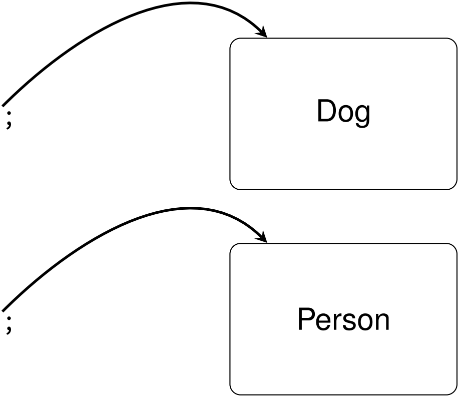


Dog

Type Confusion Example

```
Dog *d = (Dog*) ptr;
d->bark();
```

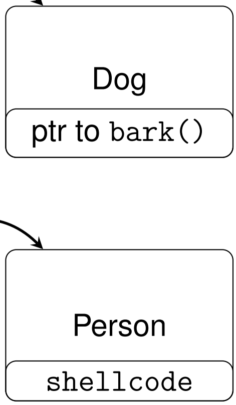
```
Dog *d = (Dog*) ptr;
d->bark(); // ???
```



The Implication

```
Dog *d = (Dog*) ptr;
d->bark();
```

```
person->name = "shellcode";
...
Dog *d = (Dog*) ptr;
d->bark(); // ???
```



Dog

ptr to bark()

Person

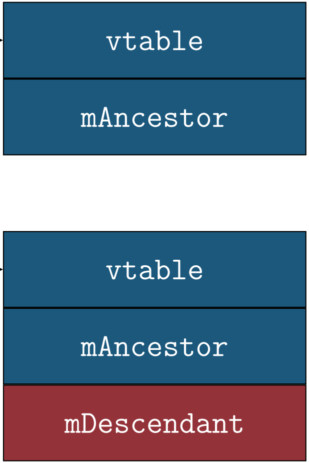
shellcode

Downcasting Problem

```
class Ancestor {
  public:
    int mAncestor;
  ...
};

class Descendant: public Ancestor {
  public:
    int mDescendant;
  ...
};
```

Downcasting Problem

```
class Ancestor {
  public:
    int mAncestor;
  ...
};

class Descendant: public Ancestor {
  public:
    int mDescendant;
  ...
};

Ancestor* a = new Ancestor();
Descendant* d = static_cast<Descendant*>(a);
d->mDescendant = 42;
```



vtable

mAncestor

vtable

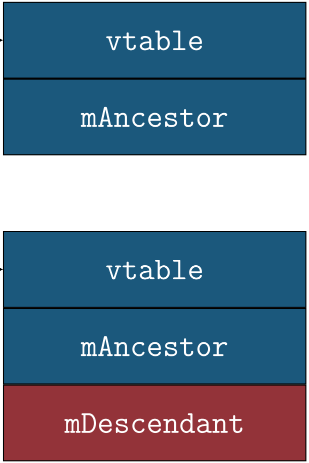
mAncestor

mDescendant

Downcasting Problem

***Memory Corruption*:**
The pointer after casting can now access a region beyond the boundary of the object.

```
class Descendant: public Ancestor {
  public:
    int mDescendant;
  ...
};

Ancestor* a = new Ancestor();
Descendant* d = static_cast<Descendant*>(a);
d->mDescendant = 42;
```



vtable

mAncestor

vtable

mAncestor

mDescendant

Question: But, Why Get Confused?

```
Ancestor* a = new Ancestor();
Descendant* d = static_cast<Descendant*>(a);
d->mDescendant = 42;
```

Question: But, Why Get Confused?

```
Ancestor* a = new Ancestor();
Descendant* d = static_cast<Descendant*>(a);
d->mDescendant = 42;
```

Suppose there is a huge gap between these lines,
e.g., separated in two different libraries.

Attacker’s Perspective

Type confusion, unlike other attack vectors, allows an attacker to ***reliably*** corrupt a certain memory area that is located relative to the victim object.

For example, we don’t need to know the actual address of `mDescendant`.

Example: Webkit Type Confusion

- CVE-2013-0912
- Confused `HTMLUnknownElement` with `SVGElement`.
- Used in Pwn2Own 2013.

Example: Webkit Type Confusion (cont'd)

```
<svg xmlns="http://www.w3.org/2000/svg">
  <foreignobject x="42" y="42" width="42" height="42">
    <body xmlns="http://www.w3.org/1999/xhtml'>
      <feColorMatrix id="viewTarget"></feColorMatrix>
    </body>
  </foreignobject>
</svg>
```

1. `feColorMatrix` becomes an `HTMLUnknownElement`.
2. `foreignObject` allows inclusion of a foreign XML namespace which has its graphical content drawn by a different user agent[2].

[2] https://developer.mozilla.org/en/docs/Web/SVG/Element/foreignObject

Example: Webkit Type Confusion (cont’d)

```
SVGElement* SVGViewSpec::viewTarget() const
{
    if (!m_contextElement)
        return 0;
    return static_cast<SVGElement*>(
        m_contextElement->treeScope()
                         ->getElementById(m_viewTargetString)
    );
}
```

Always (down)-cast to `SVGElement`

This function can return `HTMLUnknownElement`

Fix

```
SVGElement* SVGViewSpec::viewTarget() const
{
    if (!m_contextElement)
        return 0;
-    return static_cast<SVGElement*>(
+    return dynamic_cast<SVGElement*>(
        m_contextElement->treeScope()
                        ->getElementById(m_viewTargetString)
    );
}
```

Why Not Use `dynamic_cast` All the Time?

- `dynamic_cast` is expensive.
- `dynamic_cast` is not always available.
  - Compiler options such as `-fno-rtti` can disable it.

Union Type Confusion

```
struct Message {
  int msgType; // NAME or ID
  union {
    char * name;
    int    id;
  };
};
```

Both `name` and `id` are located at the same memory location.

Example

```
struct Message {
  int msgType; // NAME or ID
  union {
    char * name;
    int    id;
  };
};

void printMessage(Message *msg) {
  if (msg->msgType == NAME)
    printf("name: %s\n", msg->name);
  else
    printf("id: %d\n", msg->id);
}
```

Example

```
struct Message {
  int msgType; // NAME or ID
  union {
    char * name;
    int    id;
  };
};

void printMessage(Message *msg) {
  if (msg->msgType == NAME)
    printf("name: %s\n", msg->name);
  else
    printf("id: %d\n", msg->id);
}
```

Suppose there is a malformed message that is originally constructed as an ID message, but has a NAME type by mistake.

Example

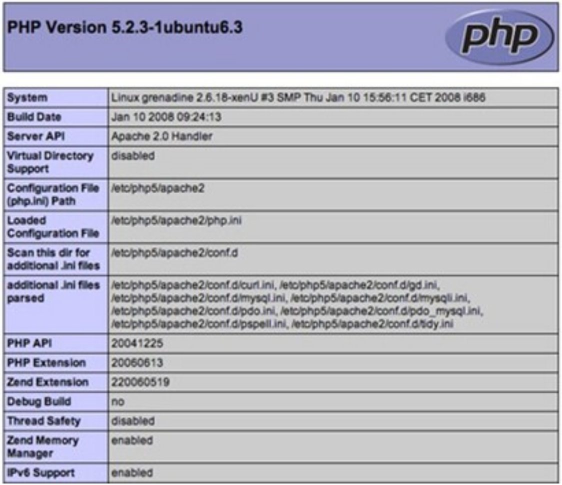
```
struct Message {
  int msgType; // NAME or ID
  union {
    char * name;
    int    id;
  };
};
```

```
{
  msgType = NAME;
  id = 0x42424242;
}
```

```
void printMessage(Message *msg) {
  if (msg->msgType == NAME)
    printf("name: %s\n", msg->name);
  else
    printf("id: %d\n", msg->id);
}
```

Example: PHP Type Confusion

```
phpinfo();
```



PHP Version 5.2.3-1ubuntu6.3

System	Linux grenadine 2.6.18-xenU #3 SMP Thu Jan 10 15:56:11 CET 2008 i686
Build Date	Jan 10 2008 09:24:13
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/apache2
Loaded Configuration File	/etc/php5/apache2/php.ini
Scan this dir for additional .ini files	/etc/php5/apache2/conf.d
additional .ini files parsed	/etc/php5/apache2/conf.d/curl.ini, /etc/php5/apache2/conf.d/gd.ini, /etc/php5/apache2/conf.d/mysql.ini, /etc/php5/apache2/conf.d/mysqli.ini, /etc/php5/apache2/conf.d/pdo.ini, /etc/php5/apache2/conf.d/pdo_mysql.ini, /etc/php5/apache2/conf.d/pspell.ini, /etc/php5/apache2/conf.d/tidy.ini
PHP API	20041225
PHP Extension	20060613
Zend Extension	220060519
Debug Build	no
Thread Safety	disabled
Zend Memory Manager	enabled
IPv6 Support	enabled

Example: PHP Type Confusion

```
/* ext/standard/info.c */

void php_print_info(int flag)
{
  ...
  if (zend_hash_find(&EG(symbol_table),
                     "PHP_SELF", sizeof("PHP_SELF"), (void**) &data)
      != FAILURE)
  {
    php_info_print_table_row(2, "PHP_SELF", Z_STRVAL_PP(data));
  }
  ...
}
```

Example: PHP Type Confusion

```
/* ext/standard/info.c */

void php_print_info(int flag)
{
  ...
  if (zend_hash_find(&EG(symbol_table),
                   "PHP_SELF", sizeof("PHP_SELF"), (void**) &data)
      != FAILURE)
  {
    php_info_print_table_row(2, "PHP_SELF", Z_STRVAL_PP(data));
  }
  ...
}
```

This is a union type.

But is always considered as a string.

PHP Union Value and Exploit

```
typedef union _zvalue_value {
  long lval;
  double dval;
  struct {
    char *val;
    int len;
  } str;
  ...
} zvalue_value;
```

Exploit:

```
<?php
$PHP_SELF = 0x42424242;
phpinfo(INFO_VARIABLES);
?>
```

PHP Union Value and Exploit

```
typedef union _zvalue_value {
  long lval;
  double dval;
  struct {
    char *val;
    int len;
  } str;
  ...
} zvalue_value;
```

Save a value as an integer first.

Retrieve a value using the string pointer.

Attacker can leak the SSL private key from the memory!

Exploit:

```
<?php
$PHP_SELF = 0x42424242;
phpinfo(INFO_VARIABLES);
?>
```

Pointer Casting in C

C Pointer Casting Problem

C standard says:

> A pointer to an object or incomplete type may be converted to a pointer to a different object or incomplete type. If the resulting pointer is not correctly aligned for the referenced type, the behavior is ***undefined***.

Unaligned Access

```c
int main(void)
{
  char a[8] = { 0, };
  int *i = (int*)(a + 1); // 0x7ffda4152631
  return printf("%p, %x\n", i, *i);
}
```

Can Unaligned Access Be A Problem on Intel?

- Intel x86 and x86-64 processors allow unaligned access.
- But, there are several instructions that require aligned access, such as `movdqa`.

Question?

Exercise

Write in assembly a function that moves 16 bytes from `src` to `dst` using `movdqa`, and observe the behavior of both aligned and unaligned memory accesses.