# The Perils of Strict Aliasing

Don’t Break the §3.10.10 (Rules)

Andy Webber
andy@aligature.com
andy.webber@sig.com

# The Rules

§3.10.10 If a program attempts to *access the stored value* of an object through a glvalue of other than one of the following types *the behavior is undefined*:52

— the *dynamic type* of the object,

— a cv-qualified version of the dynamic type of the object,

— a type similar (as defined in 4.4) to the dynamic type of the object,

— a type that is the signed or unsigned type corresponding to the dynamic type of the object,

— a type that is the signed or unsigned type corresponding to a cv-qualified version of the dynamic type of the object,

— an aggregate or union type that includes one of the aforementioned types among its elements or non- static data members (including, recursively, an element or non-static data member of a subaggregate or contained union),

— a type that is a (possibly cv-qualified) base class type of the dynamic type of the object,

— *a char or unsigned char type*.

52) The intent of this list is to specify those circumstances in which an object may or may not be aliased.

# Does this work?

```
uint32_t swaphalves(uint32_t a)
{
   auto ptr = reinterpret_cast<uint16_t*>(&a);
   std::swap(ptr[0], ptr[1]);
   return a;
}
```

(adapted from http://dbp-consulting.com/StrictAliasing.pdf)

## Does this work?

❌

```
uint32_t swaphalves(uint32_t a)
{
   auto ptr = reinterpret_cast<uint16_t*>(&a);
   std::swap(ptr[0], ptr[1]);
   return a;
}
```

(adapted from http://dbp-consulting.com/StrictAliasing.pdf)

# union*

```
uint32_t swaphalves_union_ptr(uint32_t a)
{
   union u
   {
      uint32_t dw;
      int16_t w[2];
   };

   auto u_ptr = reinterpret_cast<u*>(&a);
   std::swap(u_ptr->w[0], u_ptr->w[1]);
   return u_ptr->dw;
}
```

# union*

```
uint32_t swaphalves_union_ptr(uint32_t a)
{
   union u
   {
      uint32_t dw;
      int16_t w[2];
   };

   auto u_ptr = reinterpret_cast<u*>(&a);
   std::swap(u_ptr->w[0], u_ptr->w[1]);
   return u_ptr->dw;
}
```

## union value

```
uint32_t swaphalves_union(uint32_t a)
{
   union
   {
      uint32_t dw;
      int16_t w[2];
   };

   dw = a;
   std::swap(w[0], w[1]);
   return dw;
}
```

# union value

```
uint32_t swaphalves_union(uint32_t a)
{
    union
    {
        uint32_t dw;
        int16_t w[2];
    };

    dw = a;
    std::swap(w[0], w[1]);
    return dw;
}
```

GCC extension
(not C99 or C++14)

# __may_alias__

```
uint32_t swaphalves_mayalias(uint32_t a)
{
   using uint16_alias =
      uint16_t __attribute__((__may_alias__));
   auto ptr = reinterpret_cast<uint16_alias*>(&a);
   std::swap(ptr[0], ptr[1]);
   return a;
}
```

# __may_alias__

GCC extension
(not C99 or C++14)

```
uint32_t swaphalves_mayalias(uint32_t a)
{
   using uint16_alias =
      uint16_t __attribute__((__may_alias__));
   auto ptr = reinterpret_cast<uint16_alias*>(&a);
   std::swap(ptr[0], ptr[1]);
   return a;
}
```

# punt!

```
g++ -std=c++11 -O3 -fno-strict-aliasing alias.cpp -o alias

uint32_t swaphalves(uint32_t a)
{
   auto ptr = reinterpret_cast<uint16_t*>(&a);
   std::swap(ptr[0], ptr[1]);
   return a;
}
```

## punt!

✓

```
g++ -std=c++11 -O3 -fno-strict-aliasing alias.cpp -o alias

uint32_t swaphalves(uint32_t a)
{
   auto ptr = reinterpret_cast<uint16_t*>(&a);
   std::swap(ptr[0], ptr[1]);
   return a;
}
```

# memcpy

```
uint32_t swaphalves_memcpy(uint32_t a)
{
    uint32_t swapped;
    auto swapped_char = reinterpret_cast<char*>(&swapped);
    auto a_char = reinterpret_cast<char const*>(&a);

    std::memcpy(swapped_char, a_char + sizeof(uint16_t),
          sizeof(uint16_t));
    std::memcpy(swapped_char + sizeof(uint16_t), a_char,
          sizeof(uint16_t));
    return swapped;
}
```

## memcpy

```
uint32_t swaphalves_memcpy(uint32_t a)
{
    uint32_t swapped;
    auto a_char = reinterpret_cast<char const*>(&a);
    auto swapped_char = reinterpret_cast<char*>(&swapped);

    std::memcpy(swapped_char, a_char + sizeof(uint16_t),
          sizeof(uint16_t));
    std::memcpy(swapped_char + sizeof(uint16_t), a_char,
          sizeof(uint16_t));
    return swapped;
}
```

# Conclusions?

- Seen all over low-level code and network (de)serialization.
- -fno-strict-aliasing
    - linux kernel, libevent, others?
- gcc-help inconclusive on placement new
- alias_cast?