

# Stack-use-after-scope detector in AddressSanitizer

Vitaly Buka  
Google

# Trivial example

```
void save(int64_t* p);
```

```
void use();
```

```
void uas() {
```

```
    {
```

```
        int64_t v;
```

```
        save(&v);
```

```
    }
```

```
    use();
```

```
}
```

# Instrumentation

```
void uas() {  
    {  
        int64_t v;  
        save(&v);  
    }  
    use();  
}
```

Diagram illustrating the instrumentation of the `uas()` function. Arrows indicate the mapping between code blocks and instrumentation calls:

- The opening curly brace `{` of the function `uas()` is mapped to `__asan_set_shadow_f1(..., 4)`, `__asan_set_shadow_f8(..., 1)`, and `__asan_set_shadow_f3(..., 3)`.
- The opening curly brace `{` of the inner block is mapped to `__asan_set_shadow_00(..., 1)`.
- The variable declaration `int64_t v;` is mapped to `llvm.lifetime.start()`.
- The `save(&v);` statement is mapped to `save()`.
- The closing curly brace `}` of the inner block is mapped to `__asan_set_shadow_f8(..., 1)`.
- The `use();` statement is mapped to `llvm.lifetime.end()`.
- The closing curly brace `}` of the function `uas()` is mapped to `use()`.
- The final closing curly brace `}` of the function `uas()` is mapped to `__asan_set_shadow_00(..., 1)`.

# Usage

```
clang++ -g -fsanitize=address \  
-fsanitize-address-use-after-scope t.cpp -o test  
./test  
==ERROR: AddressSanitizer:stack-use-after-scope on...  
WRITE of size 8 at 0x7ffc02d05b00 thread T0  
    #0 0x515e40 in use() /tmp/test.cpp:8:17  
...  
Address 0x7ffc02d05b00 is located in stack of thread  
T0 at offset 32 in frame  
    #0 0x515e6f in use_after_scope() /tmp/test.cpp:10  
This frame has 1 object(s):  
    [32, 40) 'v:12' <== ... inside this variable
```

# Destructors

```
struct A {  
    void Init(const int* v) { p = v; }  
    ~A() { std::cout << *p; }  
    const int* p;  
};
```

```
// 50% of all bugs  
void uas_in_destructor() {  
    A a;  
    int v = 5;  
    a.Init(&v);  
}
```

# Temporaries

```
struct A {  
    A(const int& v) {  
        p = &v;  
    }  
    void print() {  
        std::cout << *p;  
    }  
    const int* p;  
};
```

// 20% of all bugs

```
void explicit_temp() {  
    A a(5);  
    a.print();  
}
```

// 5% of bugs

```
void temp_from_conversion() {  
    double v = 5;  
    A a(v);  
    a.print();  
}
```

# Temporary lifetime extension

```
const int& fn(const int& arg) {  
    return arg;  
}
```

```
// 5% of bugs  
void extend() {  
    const int& v = fn(5);  
    std::cout << v;  
}
```

# Simple one

```
// 15% of all bugs
void out(bool c) {
    int* p;
    if (c) {
        int v = 3;
        p = &v;
    }
    std::cout << *p;
}
```