



A Series of Unfortunate Bugs

Satabdi Das



CODE RECKONS

Science to the CORE

A Series of Unfortunate Bugs

Satabdi Das
CPPP 2021

**Developers spend half their
programming time debugging**

Reversible Debugging Software, University of Cambridge

How do we learn or get better at debugging?

How am I going
to solve this!

Don't know what
the code does!

Nobody knows
what the code
does!

This doesn't
make sense!



I have never done
this before!

Is the compiler
wrong?

What happens if I
change this?!

What if I can't
solve this bug?!



I love debugging! ❤️

How do we get **better** at
debugging unfortunate **bugs?**



**Unfortunate
Bugs**

Difficult to debug,
Critical, Stressful,
Avoidable,
Often **Simple** Fixes

Simple Bugs
Complex Behaviors

4 Bugs





1

**Crash in ancient
code!**

```
void Format::Decode(DBEntry* entry) {  
    assert(mSomeNumber > 0);  
    ...
```

- Assertion failure
- *bohrbug*

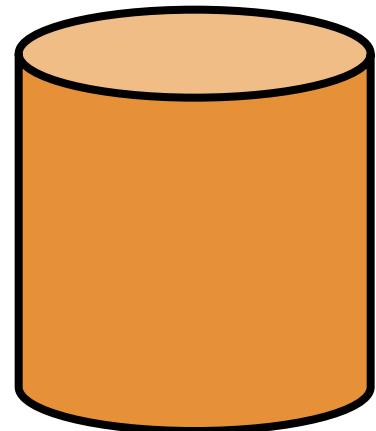
● Assertion failure

```
void Format::Decode(DBEntry* entry) {  
    assert(mSomeNumber > 0);  
    ...  
}
```

- Nobody knew about the code
- Ran for hours
- Nothing obviously wrong

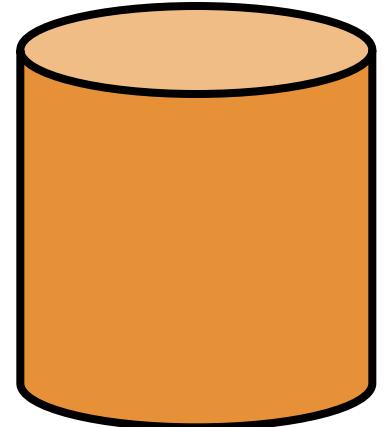
What did the module do?

- A database storing key-value pairs
- Fast
- Compact
- Many bitwise operations



What did the module do?

- Believed to be working fine
- Until one day it didn't!
- Nobody's code



Understand somebody else's code!

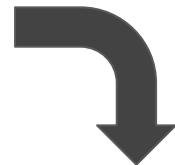


Encode

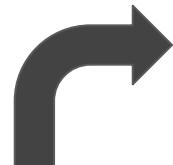
Decode

Encode

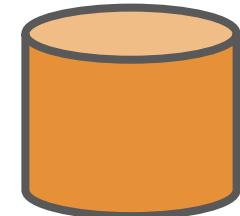
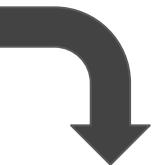
Write

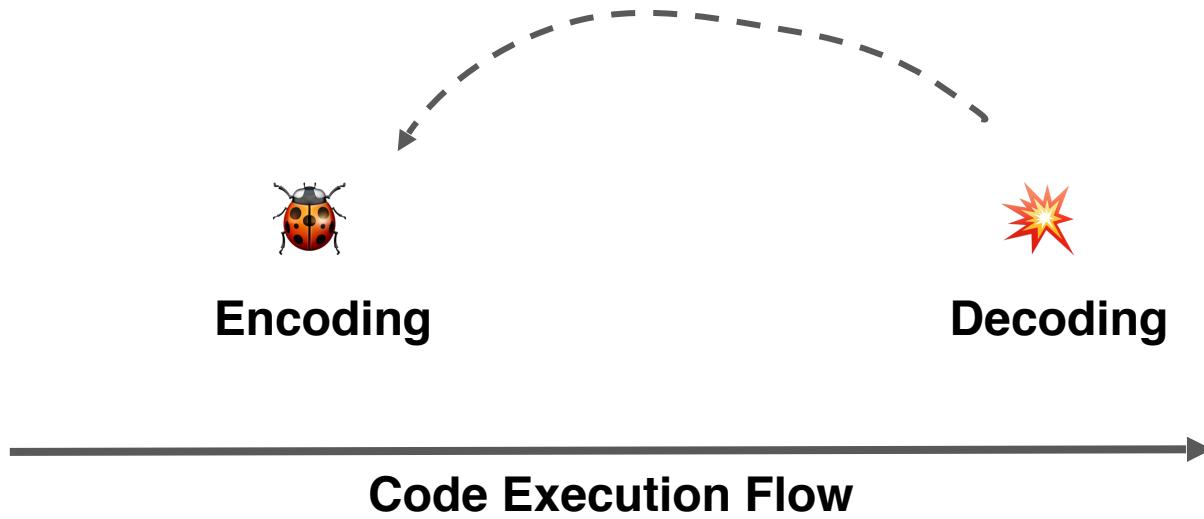


Read



Write





```
unsigned int putBits(unsigned int val, int posn) {  
    int right = 32 - posn;  
    val = val >> right;  
    return val;  
}
```

Undefined behavior



```
int main() {
    int val = 127;

    val = shiftBits(val, 0);
    std::cout << "Value "
                  << val
                  << '\n';

    val = shiftBits(val, 1);
    std::cout << "Value "
                  << val
                  << '\n';

    return 0;
}
```

```
Value 127
Value 0
```

gcc,
x86_64

A simple bug
A simple fix!

```
unsigned int putBits(unsigned int val, int posn) {  
    int right = 32 - posn;  
    val = (unsigned long long) val >> right;  
    return val;  
}
```

**Set yourself up for success,
use the right tools!**

```
g++ -fsanitize=undefined UBshift.cpp
```

```
UBshift.cpp:6:13: runtime error: shift  
exponent 32 is too large for 32-bit type  
'unsigned int'
```

```
Value 127
```

```
Value 0
```

Have helpful logs!

What did I learn?

- Positive mindset/Growth mindset
- Start with hypothesis
- Throw away unhelpful assumptions
- Make the test case smaller
- Understand the system



2

**Crash before the
release!**



32 bit
release
exec



64 bit
release/
debug, 32 bit
debug exec

No memory error reported by **Valgrind!**

Compiler optimizations: -O3, Symbols: Stripped

Debugging an optimized executable is...

Difficult!

```
SomeValue theCrashyFunction(SomeParam& param) {  
    ...  
    AList* pList = nullptr;  
    initializeList(&pList);  
    ...  
    aStruct->mList = pList;  
    if (cond) {  
        callAFunction(aStruct);  
    } else {  
        callAnotherFunction(aStruct);  
    }  
  
    unsigned int size = pList->size();
```



No obvious write to pList!

```
aStruct->mList = pList;  
if (cond) {  
    callAFunction(aStruct);  
} else {  
    callAnotherFunction(aStruct);  
}  
  
size = pList->size(); 
```

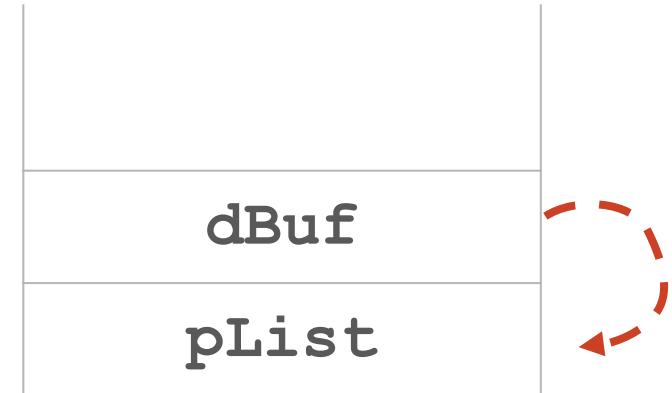


A static buffer overflow!

```
char dBuf[4 * sizeof(long unsigned)];  
sprintf(dBuf, "\.,,/(^ ^)\.,,/-%lx",  
(long unsigned) dpt);
```

Compiler inlined both the functions!

```
SomeValue theCrashyFunction(SomeParam&
param) {
    ...
    AList* pList = nullptr;
    initializeList(&pList);
    ...
    aStruct->mList = pList;
    if (cond) {
        char dBuf[4 *
            sizeof(long unsigned)];
        sprintf(dBuf, "\.,/(^ ^)\.,,%lx",
            (long unsigned) dpt);
        ...
    } else {
        ...
    }
}
```



Right tools & safer coding!

No shotgun debugging!

No back-seat driving!

One of the social rules at Recurse Center,
A programmer's retreat in New York City



3

Deadlock in code!

- **A multi-threaded program**
 - Sometimes hangs on startup
 - Stack traces of hang at different places
 - No inconsistencies in locking

● Stack traces - 2 different low level locks

```
Thread 2 (Thread 0xf7763b70 (LWP 28839)):  
#0 0x00c25430 in __kernel_vsyscall ()  
#1 0x00d181b9 in __l11_lock_wait () from /lib/libpthread.so.0  
#2 0x00d1355e in __L_lock_731 () from /lib/libpthread.so.0
```

```
Thread 1 (Thread 0xf77646c0 (LWP 28720)):  
#0 0x00c25430 in __kernel_vsyscall ()  
#1 0x00d181b9 in __l11_lock_wait () from /lib/libpthread.so.0  
#2 0x00d13550 in __L_lock_677 () from /lib/libpthread.so.0  
#3 0x00d13421 in pthread_mutex_lock () from /lib/libpthread.so.0  
#4 0x080488f4 in SleepLock::SleepLock (this=0x8476098) at thread.cpp:10
```

- Heavy use of singleton pattern

```
static Thing& getInstance() {  
    static Thing instance  
    return instance;  
}
```

- Is this thread-safe?

```
static Thing& getInstance() {  
    static Thing instance;  
    return instance;  
}
```

C++ 11: Yes, Older C++: ?

G++ 4.2



G++ 4.3

Thread 1

Acquires user space lock

Thread 2

Initializes a local static variable

Initializes a local static variable

Acquires user space lock



```
class SleepLock {
public:
    SleepLock() {
        usleep(5000);
        pthread_mutex_lock(&mutex);
        pthread_mutex_unlock(&mutex);
    }
};

int main(int, char*[]) {
    pthread_t tid;
    pthread_create(&tid, 0,
                  &thread_1, 0);
    static SleepLock obj;
    void* result;
    pthread_join(tid, &result);
    return 0;
}
```

```
void* thread_1(void*) {
    pthread_mutex_lock(&mutex);
    usleep(5000);
    static Noop* obj = new Noop();
    pthread_mutex_unlock(&mutex);
    return 0;
}
```

Difficult to reason
Code review or tools **not helpful**

**The environment is important
too!**

Talk about your bugs!

Perils of forgetting after fixing it!

Memory helps you find creative solution to a problem

Smart Thinking

Art Markman, Cognitive Scientist

Bug reports and Pareto Principle



4

Refactoring bug!

Cargo Cult Software Engineering

Steve McConnell
Author of Code Complete

```
bool someLongFunc(Type1 var1, Type2 var2,  
                  Type3 var3, Type4 var4,  
                  ...) {  
    ...  
    // Do some things  
    ...  
    jmp_buf env;  
    int status = setjmp(env);  
    ...  
    // Do some things  
    ...  
    longjmp(env, someValue);  
    ...  
}
```

```
bool someLongFunc(Type1 var1, Type2 var2,  
                  Type3 var3, Type4 var4,  
                  ...) {  
    ...  
    // Do some things  
    ...  
    jmp_buf env;  
    int status = setjmp(env);  
  
    -----  
    // Do some things  
    ...  
    longjmp(env, someValue);  
    ...  
}
```

```
bool notLongFunc1(Type1 var1,  
                  Type2 var2,  
                  Type3 var3,  
                  Type4 var4,  
                  ...) {  
  
    ...  
    // Do some things  
    ...  
    jmp_buf env;  
    int status = setjmp(env);  
    ...  
    // Do some things  
}
```

```
bool notLongFunc(Type1 var1,  
                  Type2 var2,  
                  Type3 var3,  
                  Type4 var4,  
                  ...) {  
  
    ...  
    // Do some things  
    ...  
    longjmp(env, someValue);  
    ...  
}
```

```
bool ok = someLongFunc(var1,  
                      var2,  
                      var3,  
                      ...);
```

```
bool ok1 = notLongFunc1(var1,  
                        var2,  
                        ...);  
bool ok2 = notLongFunc2(var1,  
                        var2,  
                        ...);
```

Crash
in the optimized executable!

Undefined behavior



```
bool someLongFunc(Type1 var1, Type2 var2,  
                  Type3 var3, Type4 var4,  
                  ...) {  
    ...  
    // Do some things  
    ...  
    jmp_buf env;  
    int status = setjmp(env);  
  
    -----  
    // Do some things  
    ...  
    longjmp(env, someValue);  
    ...  
}
```

Perseverance + Methodical approach

Debugging Principles



- Understand the System
- Make it Fail
- Quit Thinking and Look
- Divide and Conquer
- Change One Thing at a Time
- Keep an Audit Trail
- Check the Plug
- Get a Fresh View
- If You Didn't Fix it, It Ain't Fixed

... by David Agans

Delta Debugging

**Methodology developed by Andreas Zeller
Author of DDD**

What to focus on?

Pair Debugging

Want to get great at something, get a coach

Atul Gawande

Surgeon, Public Health Professor, Writer

Deliberate Practice

K Anders Ericsson,
**Psychologist, Author of “Peak: Secrets from the New Science of
Expertise”**

Bugs are great learning opportunities!

Start talking
Stop the culture of blaming
Believe that you can fix the bug

**Best debugging
is
No debugging**

See an unfortunate bug?

@i_satabdi