Undefined Behavior

[未定义行为]

未定义行为是什么?

特定场景下，语言标准明确声明对该处的语义不做任何要求。

未定义行为是什么?

从实际用途/危害来看，它可以是

- 语言设计者手头的强力工具
- 生成奇异bug的反应炉
- 程序优化的有力假设
- 语言本身潜在的致病基因

大衍之数五十，其用四十有九

概览

C99中的经典UB(但是一共有191个)

- 有符号整型溢出
- 除以0
- 修改const修饰类型的对象
- 违反类型规则(如将一个`int*`转换为`float*`)
- 解引用空指针
- 解引用悬垂指针/数组访问越界
- 使用未初始化的变量
- 超出大小的移位运算

C11定义了Memory Model后新增的一些UB

- 程序执行时的数据竞争
- 访问Atomic结构/联合的成员

例子：bzip2中的子串比较函数

此例子出于CppCon2016的talk“Garbage in, Garbage Out：Arguing about Undefined Behavior With Nasal Demons”

出自bzip2的源码(blocksort.c, 名字叫mainGtU)

```
bool compare(uint32_t i1, uint32_t i2, uint8_t *block) {
  uint8_t c1, c2

  // 在一个循环中
  c1 = block[i1]; c2 = block[i2];
  if(c1 != c2) return (c1 > c2);
  i1++; i2++;

}
```

也许你会想，数组下标就应该是非负的，所以uint32_t没问题

但是此处用int32_t在X86-64平台上可避免因处理潜在溢出而加入的截断

生成的汇编只有原本的二分之一

(不过bizp2还在用UInt32)

例子：GCC的鬼故事

涉及strict aliasing

```
// linux/include/net/iw_handler.h

static inline iwe_stream_add_event(char * stream, // Stream of events
                                   char * ends, // End of Stream
                                   struct iw_event * iwe, // Payload
                                   int event_len) // Real size of Payload
{
  if((stream + event_len) < ends) {
    iwe->len = event_len;
    memcpy(stream, (char *) iwe, event_len); // A1
    stream += event_len; // A2
  }
}
```

GCC看起来毫无道理地调换了A1和A2的顺序

实情：Linux内核的memcpy是个宏，将大块数据cast为`long *`进行复制

例子：Linux内核net/tun实现

https://isc.sans.edu/diary/A+new+fascinating+Linux+kernel+vulnerability/6820

发现者：Bojan Zdrnja

日期：2009 - 07 17

```
struct sock* sk = tun->sk;
// tun有可能为NULL

// ......

if(!tun)
  return POLLRR
```

GCC在编译之后将if块整个删去了

另一例: https://www.spinics.net/linux/fedora/redhat-crash-utility/msg08514.html

与未定义行为共处

- 静态分析工具
- 启用编译器警告
- 使用编译器支持的动态检查

例：Clang和gcc都支持`-ftrapv`这个选项，可开启对有符号整型的溢出检查

说多点，这个检查让溢出的行为固定，但是对`INT_MIN / -1`没用

Clang的`-fcatch-undefined-behavior`可提供一些简单的动态检查

- 使用Valgrind等提供额外动态检查的工具
- 使用推理

https://segmentfault.com/a/1190000004250754

POSIX.1-2001标准规定的dlsym函数依赖于void指针与函数指针的转换

虽然这是个UB，但是在符合POSIX.1-2001标准的系统上，转换是安全的

结束