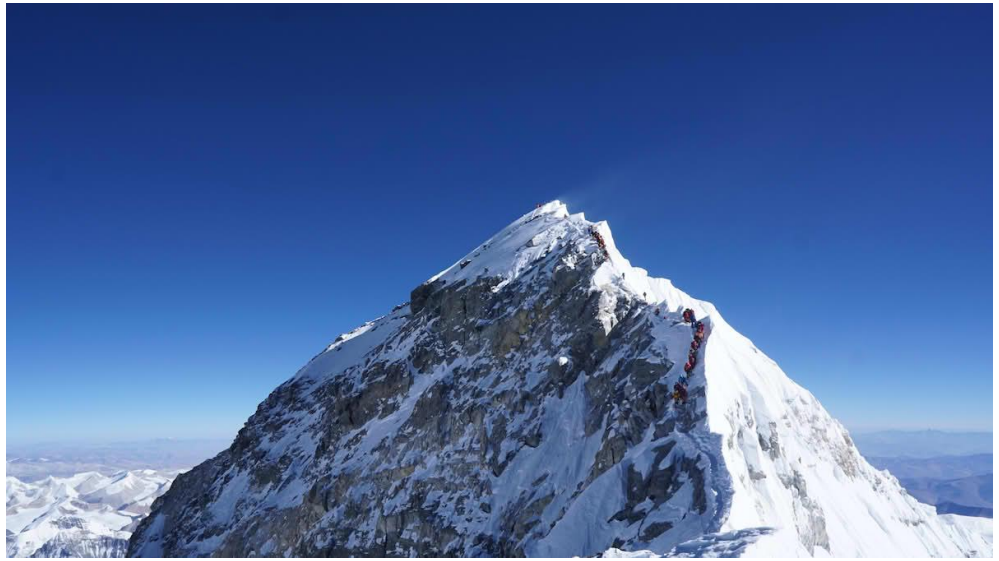
Hardening: текущий статус и перспективы развития

Что такое hardening

- Средства отлова и/или предотвращения различных видов уязвимостей
  - Выход за границу буфера, обращение по невалидному адресу, использование неинициализированных переменных и т.п
- Можно использовать в продуктовом коде релизной версии



https://snl.no/Mount_Everest

Введение

- Почему безопасность и стабильность программного обеспечения так актуальна для C/C++:
  - 70% ошибок в Microsoft и Google Chrome – ошибки памяти
  - 70% ошибок памяти – 0-day уязвимости
  - Появление новых более безопасных языков
  - Гос. заказчики различных стран рекомендуют использовать hardening или безопасных языков

Цель

- Детальный обзор Hardening
  - Какие средства имеются
  - Как использовать в своих приложениях
  - Какие накладные расходы
- Дальнейшее развитие в языке

Суть Hardening

Чем является Hardening

- Правила безопасных
  - Разработки (secure development process)
  - Поставки (удаление и сокрытие символов библиотек)
  - Выполнения (защиты в компиляторе, библиотеке, ядре)

Правила безопасной разработки

- Безопасные аналоги библиотечных функций
  - Annex K (memset_s et al.) и другие расширения
  - Могут легко фальсифицироваться при отсутствии контроля!
- Ограничения на использование небезопасных функций
  - rand, strcpy, etc.
- Статический анализ
  - Обязательно -Wall -Wextra -Werror
  - Желательно -Wformat=2 -Wconversion -Wsign-conversion
  - Тулы: Clang Static Analyzer, Clang-Tidy, etc.
- Проверки спецификаций (Design-by-Contract)
  - Asserts/контракты

Чем является Hardening

- Hardening – интегральный подход!
- В *этом* докладе рассматриваем только рантайм-проверки
  - Компилятор и библиотека

Linux Hardening Guide

Last edited: March 19th, 2022

Linux is not a secure operating system. However, there are steps you can take to improve it. This guide aims to explain how to harden Linux as much as possible for security and privacy. This guide attempts to be distribution-agnostic and is not tied to any specific one.

Contents

Требования к Hardening

- Минимальные накладные расходы (не более 2-3%)
- Высокая точность – отсутствие false positives
- Легкая интеграция
  - Совместимость ABI
  - Простота использования

Слайды Юрия

Анатомия фортификации

Как работает фортификация?

- При включенном макросе _FORTIFY_SOURCE стандартные функции вызывают встроенные функции компилятора (builtins)
- Builtins раскрываются
  - В обычные функции
    - если не удалось определить размер dst
    - либо известно, что размер dst >= src
  - В *_chk версии, которые проверяют размер объекта в рантайме

__builtin_object_size

```
size_t __builtin_object_size (const void *ptr, /*detail*/)
```

- Возвращает
  - размер объекта, на который указывает ptr
  - -1, если не удалось статически вычислить размер

```
size_t __builtin_dynamic_object_size(const void *ptr, /*detail*/)
```

- То же что __builtin_object_size, но размер может быть вычислен динамически
  - нужен для malloc, alloca, VLA

Пример с strcpy

```
int main(int argc, char *argv[]) {
  strcpy(a, argv[1]);
  puts(a);
}
```

```
int main(int argc, char *argv[]) {
  __builtin___strcpy_chk (a, argv[1], __builtin_object_size(a, ...));
  puts(a);
}
```

__builtin_object_size(...) > src_len

__builtin_object_size(...) ⩾ src_len

```
extern char a[];

int main(int argc, char *argv[]) {
  strcpy(a, argv[1]);
  puts(a);
}
```

```
char a[4];

int main(int argc, char *argv[]) {
  __strcpy_chk(a, argv[1], dst_len);
  puts(a);
}
```

Пример с strcpy

```
int main(int argc, char *argv[]) {
  __strcpy_chk (a, argv[1], dst_size);
  puts(a);
}
```

```
char * __strcpy_chk(char *dest, const char *src, size_t destlen){
  size_t len = strlen(src);

  if (len >= destlen)
    abort();

  return memcpy(dest, src, len + 1);
}
```

Побочные эффекты

```
int main() {
  void *p;
  size_t n;
  if (getenv("TEST")) {
    p = &n;
    n = sizeof n;
    puts("Hello");
  } else {
    p = nullptr;
    n = 0;
  }
  memset(p, 0, n);
}
```

С фортификацией

```
int main() {
  getenv("TEST");
  puts("Hello");
}
```

- При фортификации у параметра memset выставляется атрибут nonnull
- Следствие - компилятор выбросит nullptr-ветку
- Детали в LLVM #60389

Поддержка Hardening в языке

Типы поведения программы в C++

- Типы поведения
  - Undefined
    - оптимизатор может делать все
  - Unspecified
    - зависит от имплементции (недокументированно)
  - Implementation-defined
    - Зависит от имплементации (документированно)
  - Erroneous

Erroneous behavior

- Определенное поведение для некорректного кода
  - Не должно приводить к уязвимостям
- Противопоставляется UB
- В C++26 пока только для неинициализированных переменных (P2795)
  - Возможно для некоторых missing return (P2973)
- Вероятно будет реализовано на базе флага -ftrivial-auto-var-init=

```
int foo(){
  int x;
  return x ? 1 : 1;
}
```

C++23 (UB)

```
int foo(){
  abort();
}
```

```
int foo(){
  while(true);
}
```

C++26 (Erroneous)

```
int foo(){
  return 1;
}
```

```
int foo(){
  return 0x42;
}
```

[[indeterminate]] атрибут

- Отключает Erroneous Behavior для локальной переменной или параметра функции (не будет инициализации, возврат к UB)

```
void f(int);

void g() {
  int x [[indeterminate]]; // indeterminate value
  int y;                   // erroneous value

  f(x); // undefined behavior
  f(y); // erroneous behavior
}
```

Standard library hardening в Стандарте

- P3471
- Стандартизация предусловий для некоторых функций стандартной библиотеки
  - Hardened preconditions
    - Например idx < size в vector::operator[]
  - Будут реализованы на контрактах (precondition assertions)
- Тулчейны будут предоставлять флаги для включения
  - Например -fhardened или -D_LIBCPP_HARDENING_MODE

C++ Profiles

- Развитие и стандартизация C++ Core Guidelines
- Запрет небезопасных или непроверяемых конструкций языка
  - Например замена raw pointers на std::span
  - Контролируется статическим анализом
- Средства миграции
  - C++ Safe Buffers, Clang-Tidy fix-its
    - Уже используется в Chrome
- Кандидаты для включения в Safety Profiles:
  - Hardening стандартной библиотеки
  - Запрет сырых указателей
  - Явное владение ресурсом (RAII)
  - Запрет union
  - Запрет на небезопасные касты
- Будет ли единым механизмом для унификации hardening?

Safety Profiles:
Type-and-resource Safe programming in ISO Standard C++

Bjarne Stroustrup
Columbia University
www.stroustrup.com
Gabriel Dos Reis
Microsoft