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

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

Введение

Цель

- Детальный обзор типов hardening в C/C++ приложениях
- Использование средств hardening в своих приложениях
- Накладные расходны на использование hardening
- Развитие языка в направлении включения hardening

Содержание

- Суть hardening (конкретные примеры, особенности реализации, отличие hardening от других средств отладки и т.п)
- Детальный обзор имеющихся средств hardening
- Дальнейшее развитие в Стандарте языка

Суть Hardening

Чем является Hardening

- Правила безопасной разработки
- Безопасная поставка софта
- Проверки в рантайме (компилятор, библиотеки, ядро)

Правила безопасной разработки

- Предпочтение в использовании безопасных аналогов библиотечных функций
  - memset_s, memcpy_s, etc.
- Ограничение на использование небезопасных функций
  - rand, strcpy, etc.
- Статический анализ
  - Обязательно -Wall -Wextra -Werror
  - Дополнительно -Wformat=2 -Wconversion=2
  - Дополнительные инструменты: clang-static-analyzer, clang-tidy, etc.
- Дополнительные проверки
  - asserts
  - Контракты

TODO: упомянуть Annex K

Безопасная поставка софта

- Удалять всю информацию о символах из исполняемого файла (опция линкера -s)
- Скрыть символы из динамической таблицы символов (-fvisibility=hidden)

Проверки в рантайме

- Включение ASLR (-fPIE)
- Включение stack protector (-fstack-protector / -fstack-protector-strong)
- Включение фортификации (-D_FORTIFY_SOURCE=3)
- Использование minimal UBsan (-fsanitize=undefined -fsanitize-minimal-runtime)
- Etc.

Чем является Hardening

- Hardening – интегральный подход!
- Рассматриваем рантайм проверки в тулчейне (компилятор и библиотека)

Contents

Требования к Hardening

- Минимальные накладные расходы (не более 1-2%)
- Высокая точность – отсутствие false positive
- Легкая интеграция
  - Совместимость ABI
  - Простота использования (флаг командной строки)

Слайды Юрия

Поддержка Hardening в языке (компилятор, библиотека)

__builtin_object_size

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

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

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

- Аналогичен обычному builtin_object_size, но размер может быть вычислен динамически

Пример с strcpy

```
char a[4];

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,
__USE_FORTIFY_LEVEL > 1));
  puts(a);
}
```

__builtin_object_size(...) != -1

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

__builtin_object_size(...) == -1 или dst_len <= src_len

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

Артефакты фортификации

```
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");
}
```

C++ Safe Buffers

- TODO (1-3 мин)

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

- Противопоставляется UB
- Определенное поведение
  - Некорректный код (означает не то, что имел в виду программист при на написании)
  - Не приводит к рискам безопасности

TODO: пример который был UB

Erroneous behavior для неинициализированных переменных

- Инициализация произвольным значением
- TBD (вмержить вверх), упомянуть про missed return, и stdlib

Автоинициализация переменных (ftrivial-auto-var-init=*)

```
int main() {
  char buffer[2048];

  size_t n;

  do {
    n = fread(buffer, 1,
sizeof(buffer), stdin);
    fwrite(buffer, 1, n ,
stdout);
  } while(n);
  return 0;
}
```

-O0

```
int main() {
  char buffer[2048];
  memset(buffer, 0,
sizeof(buffer));
  size_t n;
  n = 0;
  do {
    n = fread(buffer, 1,
sizeof(buffer), stdin);
    fwrite(buffer, 1, n ,
stdout);
  } while(n);
  return 0;
}
```

-O1

```
int main() {
  char buffer[2048];
  memset(buffer, 0,
sizeof(buffer));
  size_t n;

  do {
    n = fread(buffer, 1,
sizeof(buffer), stdin);
    fwrite(buffer, 1, n ,
stdout);
  } while(n);
  return 0;
}
```

[[indeterminate]] атрибут

- Отмена erroneous behavior для локальной переменной или параметра функции (не будет инициализации)

```
void f(int);

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

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

Контракты

- Контрактное программирование (DbC)
  - Предусловия
  - Постусловия
  - Инварианты

Контракты

- Предусловия
  - pre(cond)
- Постусловия
  - post(cond)
- contract_assert(cond)

```
int divide(int dividend, int divisor)
pre(divisor != 0) {
  return dividend / divisor;
}
```

```
int absolute_value(int num) post(r :
r >= 0) {
  return std::abs(num);
}
```

Hardening в стандартной библиотеке

- Предусловия для некоторых функций стандартной библиотеки
  - например idx < size для vector::operator[]
  - TODO: упомянуть контракты
  - TODO: упомянуть erroneous
- Как включить (сослаться на Юру, упростить нижние буллеты)
  - тулчейны будут предоставлять флаг –fhardened, который включает проверки (возможно не только стандартной билбилотеки)
  - -D_LIBCPP_HARDENING_MODE=mode

C++ Profiles

- Развитие и стандартизация C++ Core Guidelines
- Запрет небезопасных или непроверяемых конструкций языка (например, std::span вместо raw pointers)
  - Контролируется статическим анализом
- Средства миграции
  - C++ safe buffers, Clang-tidy fix-its
- Кандидаты включения
  - Hardening стандартной библиотеки
  - Заперт raw pointers
  - Явное владение ресурсом (raii)
  - TBD
- Будет ли единым механизмом для унификации hardening?

Что включать у себя?

- ASLR (-fpie)
- Stack Protector (-fstack-protector-strong)
- Фортификация (-D_FORTIFY_SOURCE=3)
- Full RELRO (-Wl,-z,relro -Wl,-z,now)
- Защита от Stack Clash (-fstack-clash-protection)
- Control-flow Integrity (-fcf-protection на X86, -mbranch-protection на AArch64)