

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 продуктивном 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

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
 - Поставки (strip and hide symbols)
 - Выполнения (компилятор, библиотеки, ядро)

Правила безопасной разработки

- Безопасные аналоги библиотечных функций
 - Annex K (memset_s et al.) и другие расширения
 - Могут легко сфальсифицироваться при отсутствии контроля!
- Ограничение на использование небезопасных функций
 - rand, strcpy, etc.
- Статический анализ
 - Обязательно -Wall -Wextra -Werror
 - Желательно -Wformat=2 -Wconversion=2
 - Тулы: clang-static-analyzer, clang-tidy, etc.
- DbC
 - Asserts/Контракты

Чем является Hardening

- Hardening – интегральный подход!
- 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

- [1. Choosing the right Linux distribution](#)
- ▶ [2. Kernel hardening](#)
- [3. Mandatory access control](#)
- ▶ [4. Sandboxing](#)
- [5. Hardened memory allocator](#)
- [6. Hardened compilation flags](#)
- [7. Memory safe languages](#)
- ▶ [8. The root account](#)
- [9. Firewalls](#)
- ▶ [10. Identifiers](#)
- ▶ [11. File permissions](#)
- ▶ [12. Core dumps](#)
13. Swap

Требования к Hardening

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

Слайды Юрия

Анатомия фортификации

__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*/)
```

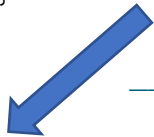
- Аналогичен обычному __builtin_object_size, но размер может быть вычислен динамически
 - нужен для malloc, alloca, VLA

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`

```
extern char a[];
```

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



`__builtin_object_size(...) ≥ src_len`

```
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 выбросить `nullptr`-ветку
- [LLVM #60389](#)

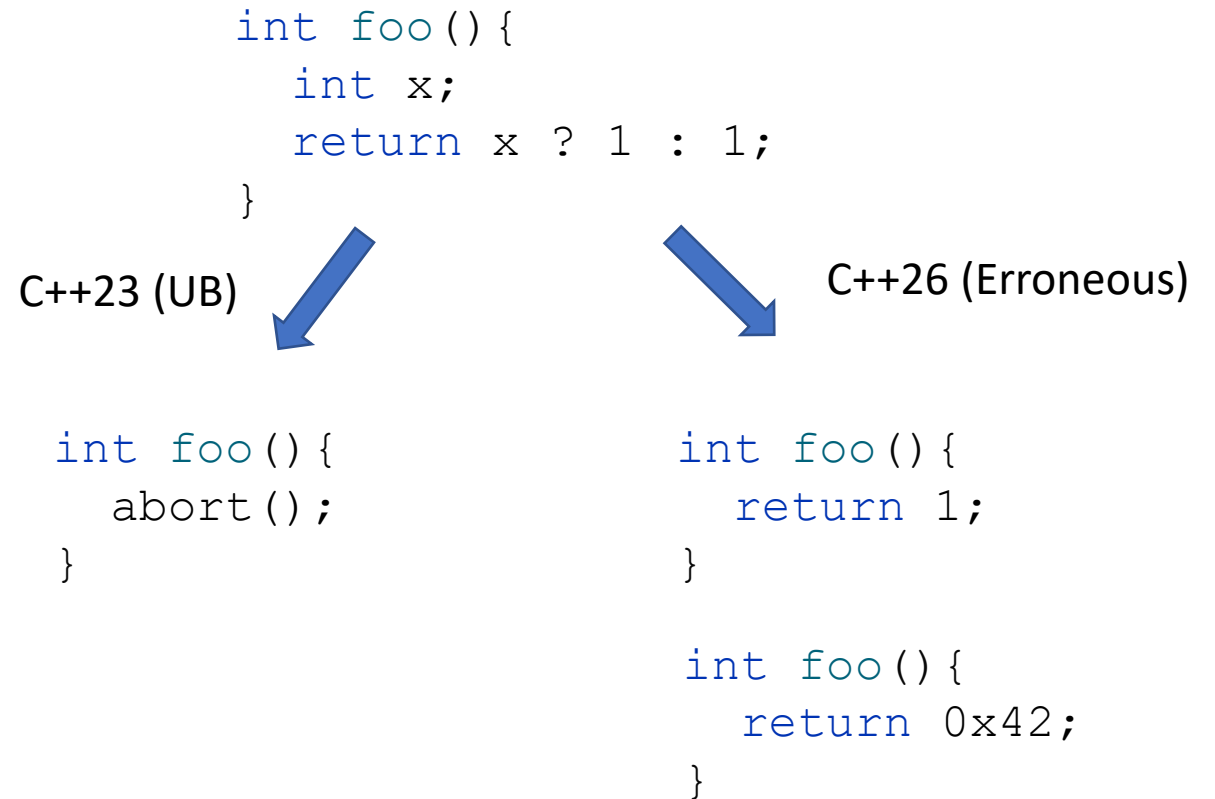
Поддержка Hardening в языке

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
 - Возможно для некоторых missing return (P2973)
- Вероятно будет реализовано на базе *-ftrivial-auto-var-init=*



[[indeterminate]] атрибут

- Отмена 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` для `vector::operator[]`
 - Будут реализованы на контрактах (precondition assertion)
- 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



Заключение

Джентельменский набор

- 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)