

Безопасный компилятор: надёжная оптимизация и улучшение защищённости кода

Дмитрий Мельник

dm@ispras.ru

24 мая 2023 г.

ИСП **РАН**

- Примеры неопределённого поведения (UB) в C и C++. Кодогенерация. Безопасный компилятор.
- Снижение уровня критичности уязвимостей. Диверсификация.
- Классы безопасности. Развёртка опций безопасного компилятора.
- Производительность безопасного кода.

Неопределённое поведение

Программы на языках C и C++, согласно стандартам языков, могут содержать неопределённое поведение, например:

- выход за границы массивов
- знаковое переполнение чисел
- целочисленное деление на 0

Если в программе есть неопределённое поведение:

- нет никаких гарантий относительно поведения программы
- диагностика от компилятора не требуется (и часто невозможна)
- при выполнении программа может продолжить работу с непредсказуемыми результатами

- Компилятор при оптимизации кода исходит из того, что входная программа корректна, и ни для каких входных данных в ней не возникает неопределённого поведения
- Оптимизации гарантируют сохранение семантики только для корректных программ
- Компилятор может использовать любую возможность для оптимизации в рамках указанных ограничений

«Обратная» дедукция на основе наличия UB

		1	f:		
		2	test	edi, edi	
1	int f(int c)	3	jne	.L4	
2	{	4	ret		
3	if (c)	5	.L4:		1 f(int):
4	return 42;	6	mov	eax, 42	2 mov eax, 42
5	}	7	ret		3 ret
	Программа с UB?		gcc -xc		gcc -xc++

<https://godbolt.org/z/611Pehjs9>

Разные сценарии проявления UB

```
1  int foo(int c)
2  {
3      if (!c)
4          return 42;           0ops
5      fputs("Oops\n", stderr); 0ops
6  }                             0ops
7                               0ops
8  int main(int argc, char **argv) 0ops
9  {                             0ops
10     foo(argc);                0ops
11 }                             0ops
```

Программа с UB??

gcc -O -fpic

g++ -O -fpic

[...]

<https://godbolt.org/z/69G671rG9>

Оптимизация -flifetime-dse

```
1 struct PolyBase {
2     double coef[4];
3 };
4 struct Poly : PolyBase {
5     Poly() {}
6     void* operator new(size_t size)
7     {
8         void *r = ::operator new(size);
9         memset(r, -1, size);
10        return r;
11    }
12 };
13 int main()
14 {
15     Poly *p = new Poly();
16     printf("%a\n", p->coef[0]);
17 }
```

0x0p+0

Результат с gcc -O2
(-flifetime-dse=2)

Жертвы:

- Firefox
- LLVM при сборке с LTO

<https://godbolt.org/z/b6znPKbWz>

Удаление «избыточной» записи в память

```
1 void func(void) {  
2     char password[1024] = {0};  
3     get_password(password);  
4     use_password(password);  
5     memset(password, 0, 1024); // delete password from memory  
6 }
```


Удаление «избыточной» записи в память

```
1 void func(void) {  
2     char password[1024] = {0};  
3     get_password(password);  
4     use_password(password);  
5     memset(password, 0, 1024); // delete password from memory  
6 }
```

Записи в память не влияют на результат вычислений!

Удаление «избыточной» записи в память

```
1 void func(void) {  
2     char password[1024] = {0};  
3     get_password(password);  
4     use_password(password);  
5     memset(password, 0, 1024); // delete password from memory  
6 }
```

Записи в память не влияют на результат вычислений!

Решение: `-fpreserve-memory-writesБК` (новая опция безопасного компилятора).

Переполнение знакового целого

```
1  int vfs_fallocate(..., loff_t offset, loff_t len)
2  {
3      struct inode *inode = [...];
4      if (offset < 0 || len <= 0)
5          return -EINVAL;
6      /* Check for wrap through zero too */
7      if ((offset + len > inode->i_sb->s_maxbytes)
8          || (offset + len < 0))
9          return -EFBIG;
10     [...]
11 }
```

Код из fs/open.c в ядре Linux 5.14

Переполнение знакового целого

```
1  int vfs_fallocate(..., loff_t offset, loff_t len)
2  {
3      struct inode *inode = [...];
4      if (offset < 0 || len <= 0)
5          return -EINVAL;
6      /* Check for wrap through zero too */
7      if ((offset + len > inode->i_sb->s_maxbytes)
8          || (offset + len < 0))
9          return -EFBIG;
10     [...]
11 }
```

Далее $offset \geq 0$ и $len > 0$

Код из fs/open.c в ядре Linux 5.14

Переполнение знакового целого

```
1  int vfs_fallocate(..., loff_t offset, loff_t len)
2  {
3      struct inode *inode = [...];
4      if (offset < 0 || len <= 0)
5          return -EINVAL;
6      /* Check for wrap through zero too */
7      if ((offset + len > inode->i_sb->s_maxbytes)
8          || (offset + len < 0))
9          return -EFBIG;
10     [...]
11 }
```

Далее $offset \geq 0$ и $len > 0$

Тождественно ложная проверка

Код из fs/open.c в ядре Linux 5.14

Переполнение знакового целого

```
1  int vfs_fallocate(..., loff_t offset, loff_t len)
2  {
3      struct inode *inode = [...];
4      if (offset < 0 || len <= 0)
5          return -EINVAL;
6      /* Check for wrap through zero too */
7      if ((offset + len > inode->i_sb->s_maxbytes)
8          || (offset + len < 0))
9          return -EFBIG;
10     [...]
11 }
```

Далее $offset \geq 0$ и $len > 0$

Тождественно ложная проверка

Код из fs/open.c в ядре Linux 5.14

Переполнение знакового целого — UB; оно не встречается в корректных (с точки зрения Стандарта) программах → проверку можно удалить.

Переполнение знакового целого

```
1  int vfs_fallocate(..., loff_t offset, loff_t len)
2  {
3      struct inode *inode = [...];
4      if (offset < 0 || len <= 0)
5          return -EINVAL;
6      /* Check for wrap through zero too */
7      if ((offset + len > inode->i_sb->s_maxbytes)
8          || (offset + len < 0))
9          return -EFBIG;
10     [...]
11 }
```

Далее $offset \geq 0$ и $len > 0$

Тождественно ложная проверка

Код из fs/open.c в ядре Linux 5.14

Переполнение знакового целого — UB; оно не встречается в корректных (с точки зрения Стандарта) программах → проверку можно удалить.

-fno-strict-overflow защищает проверку от удаления (стандартная опция GCC).

Необходимость безопасной компиляции

- Уязвимости в программе могут появляться не только из-за ошибок в ее коде, но и в результате оптимизаций, выполняемых компилятором
 - Проблемы могут возникнуть в т.ч. в «старом», проверенном коде, при обновлении версии компилятора
- Компилятор может реализовывать дополнительные методы защиты, затрудняющие реализацию угроз безопасности

Разработка безопасного ПО: многоуровневый подход

При разработке безопасного ПО должен применяться комплексный подход, позволяющий обнаруживать и исправлять ошибки, а также предотвращать угрозы безопасности на разных уровнях жизненного цикла ПО:

- Исходный код
 - Использование стандартов безопасного программирования (MISRA, SEI CERT и др.)
 - Статический анализ
 - Формальная верификация

- Бинарный код
 - Фаззеры
 - Динамический анализ
 - Тестирование на проникновение

Разработка безопасного ПО: многоуровневый подход

При разработке безопасного ПО должен применяться комплексный подход, позволяющий обнаруживать и исправлять ошибки, а также предотвращать угрозы безопасности на разных уровнях жизненного цикла ПО:

- Исходный код
 - Использование стандартов безопасного программирования (MISRA, SEI CERT и др.)
 - Статический анализ
 - Формальная верификация
- **Средства разработки (сборки)**
 - **Безопасный компилятор**
- Бинарный код
 - Фаззеры
 - Динамический анализ
 - Тестирование на проникновение

- В ИСП РАН разрабатывается безопасный компилятор (SAFEC), предлагающий решения для рассматриваемых проблем на этапе компиляции и оптимизации кода
- Безопасный компилятор основан на коде GCC (изначально 9.4, но портирован и на другие версии)

- **Надёжная оптимизация кода**
 - Не должны вноситься дополнительные уязвимости на этапе компиляции ПО (в т.ч. оптимизация кода с неопределённым поведением)
- **Диагностика**
 - Выдача предупреждений о потенциально небезопасном коде (напр., `-Wextra-safety`)
- **Улучшение защищённости кода**
 - Включение динамических методов защиты на этапе компиляции (`hardening`, `sanitizers`) — снижение степени потенциальной угрозы безопасности

Проверка указателя на NULL после разыменования

```
1 struct dev {
2     int encoded_id;
3 };
4
5 int get_fallback_id(struct dev *);
6
7 int decode_id(struct dev *dev)
8 {
9     int raw_id = dev->encoded_id;
10
11     if (!dev)
12         return get_fallback_id(dev);
13
14     return raw_id ^ 0x31415926;
15 }
```

<https://godbolt.org/z/qzW5MWP1j>

```
1 decode_id:
2     mov     eax, DWORD PTR [rdi]
3     xor     eax, 0x31415926
4     ret
```

Взгляд компилятора:

1. Разыменование `dev->encoded_id` означает, что `dev != NULL` (программа не может содержать UB).
2. Условие `!dev` ложно → код может быть удалён.

Удаление проверки может привести к угрозам безопасности вплоть до появления эксплуатируемой уязвимости [1].

Проверка указателя на NULL после разыменования

```
1 struct dev {
2     int encoded_id;
3 };
4
5 int get_fallback_id(struct dev *);
6
7 int decode_id(struct dev *dev)
8 {
9     int raw_id = dev->encoded_id;
10
11     if (!dev)
12         return get_fallback_id(dev);
13
14     return raw_id ^ 0x31415926;
15 }
```

<https://godbolt.org/z/qzW5MWP1j>

```
1 decode_id:
2     mov     eax, DWORD PTR [rdi]
3     xor     eax, 0x31415926
4     ret
```

Взгляд компилятора:

1. Разыменование `dev->encoded_id` означает, что `dev != NULL` (программа не может содержать UB).
2. Условие `!dev` ложно → код может быть удалён.

Удаление проверки может привести к угрозам безопасности вплоть до появления эксплуатируемой уязвимости [1].

`-fno-delete-null-pointer-checks` защищает проверку от удаления.

БИТОВЫЙ СДВИГ НА НЕДОПУСТИМУЮ ВЕЛИЧИНУ

1	<code>void g(int);</code>	1	<code>f:</code>	
2		2	<code>mov ecx, esi</code>	
3	<code>void f(int val, int shift)</code>	3	<code>mov eax, 1</code>	
4	<code>{</code>	4	<code>sal eax, cl</code>	
5	<code>int x = 1u << shift;</code>	5	<code>mov ecx, eax</code>	
6	<code>if (x)</code>	6	<code>mov eax, edi</code>	
7	<code>g(val / x);</code>	7	<code>cdq</code>	
8	<code>}</code>	8	<code>idiv ecx</code>	<i>Безусловное</i>
		9	<code>mov edi, eax</code>	<i>деление</i>
		10	<code>jmp g</code>	<i>и вызов g()</i>

В отсутствие неопределённого поведения значение `x` строго положительно → проверку можно удалить.

<https://godbolt.org/z/bfdrKexK9>

Неинициализированная переменная как источник энтропии?

```
1  int random()
2  {
3      int uninit;
4      return getpid() ^ uninit;
5  }
6  int test()
7  {
8      return random() & 1;
9  }
```

По мотивам `srandomdev()` в `libc`
FreeBSD 8.0

```
1  random:
2      jmp     getpid@PLT # TAILCALL
3  test:
4      push   rax
5      call  getpid@PLT
6      xor   eax, eax
7      pop   rcx
8      ret
```

<https://godbolt.org/z/vKf3ch6aK>

Неинициализированная переменная как источник энтропии?

```
1  int random()
2  {
3      int uninit;
4      return getpid() ^ uninit;
5  }
6  int test()
7  {
8      return random() & 1;
9  }
```

```
1  random:
2      jmp     getpid@PLT # TAILCALL
3  test:
4      push   rax
5      call  getpid@PLT
6      xor   eax, eax
7      pop   rcx
8      ret
```

По мотивам `srandomdev()` в `libc`
FreeBSD 8.0

<https://godbolt.org/z/vKf3ch6aK>

`-finit-local-vars` принудительно инициализирует переменные нулями.

- **Надёжная оптимизация кода**
 - Не должны вноситься дополнительные уязвимости на этапе компиляции ПО (в т.ч. оптимизация кода с неопределённым поведением)
- **Диагностика**
 - Выдача предупреждений о потенциально небезопасном коде (напр., `-Wextra-safety`)
- **Улучшение защищённости кода**
 - Включение динамических методов защиты на этапе компиляции (`hardening`, `sanitizers`) — снижение степени потенциальной угрозы безопасности

Разбор примера setjmp/longjmp

```
1     jmp_buf jb;
2     /*volatile*/ int ret = 1;
3
4     if (!setjmp(jb)) {
5
6         ret = 0;
7
8         longjmp(jb, 1);
9     }
10
11     return ret;
```

<https://godbolt.org/z/fjsY3hbeM>

Разбор примера setjmp/longjmp

```
1     jmp_buf jb;
2     /*volatile*/ int ret = 1;
3
4     if (!setjmp(jb)) {           обычный возврат (0)
5
6         ret = 0;
7
8         longjmp(jb, 1);
9     }
10
11     return ret;
```

<https://godbolt.org/z/fjsY3hbeM>

Разбор примера setjmp/longjmp

```
1     jmp_buf jb;
2     /*volatile*/ int ret = 1;
3
4     if (!setjmp(jb)) {           обычный возврат (0)
5
6         ret = 0;                переопределение
7
8         longjmp(jb, 1);
9     }
10
11     return ret;
```

<https://godbolt.org/z/fjsY3hbeM>

Разбор примера setjmp/longjmp

```
1     jmp_buf jb;
2     /*volatile*/ int ret = 1;
3
4     if (!setjmp(jb)) {           обычный возврат (0)
5
6         ret = 0;                переопределение
7
8         longjmp(jb, 1);         переход на строку 4
9     }                           из вложенного кадра
10                                стека
11     return ret;
```

<https://godbolt.org/z/fjsY3hbeM>

Разбор примера setjmp/longjmp

```
1  jmp_buf jb;
2  /*volatile*/ int ret = 1;
3
4  if (!setjmp(jb)) {           обычный возврат (0)           возврат через longjmp (1)
5
6      ret = 0;                переопределение
7
8      longjmp(jb, 1);         переход на строку 4
9  }                           из вложенного кадра
10                             стека
11  return ret;
```

<https://godbolt.org/z/fjsY3hbeM>

Разбор примера `setjmp/longjmp`

```
1  jmp_buf jb;
2  /*volatile*/ int ret = 1;
3
4  if (!setjmp(jb)) {           обычный возврат (0)           возврат через longjmp (1)
5
6      ret = 0;                переопределение
7
8      longjmp(jb, 1);         переход на строку 4
9  }                           из вложенного кадра
10                             стека
11  return ret;                 возвращаем значение
                               (какое?)
```

<https://godbolt.org/z/fjsY3hbeM>

Разбор примера setjmp/longjmp

```
1  jmp_buf jb;
2  /*volatile*/ int ret = 1;
3
4  if (!setjmp(jb)) {           обычный возврат (0)           возврат через longjmp (1)
5
6      ret = 0;                переопределение
7
8      longjmp(jb, 1);         переход на строку 4
9  }                           из вложенного кадра
10                             стека
11  return ret;                 возвращаем значение
                               (какое?)
```

<https://godbolt.org/z/fjsY3hbeM>

Program returned: 1

Улучшение диагностики -Wclobbered

```
1     jmp_buf jb;
2     /*volatile*/ int ret = 1;
3
4     if (!setjmp(jb)) {
5
6         ret = 0;
7
8         longjmp(jb, 1);
9     }
10
11     return ret;
```

Улучшение диагностики -Wclobbered

```
1  jmp_buf jb;
2  /*volatile*/ int ret = 1;
3
4  if (!setjmp(jb)) {
5
6      ret = 0;
7
8      longjmp(jb, 1);
9  }
10
11 return ret;
```

- Clang не предоставляет опций для диагностики

Улучшение диагностики -Wclobbered

```
1  jmp_buf jb;
2  /*volatile*/ int ret = 1;
3
4  if (!setjmp(jb)) {
5
6      ret = 0;
7
8      longjmp(jb, 1);
9  }
10
11 return ret;
```

- Clang не предоставляет опций для диагностики
- В GCC оптимизации скрывают проблему от -Wclobbered

Улучшение диагностики -Wclobbered

```
1  jmp_buf jb;
2  /*volatile*/ int ret = 1;
3
4  if (!setjmp(jb)) {
5
6      ret = 0;
7
8      longjmp(jb, 1);
9  }
10
11  return ret;
```

- Clang не предоставляет опций для диагностики
- В GCC оптимизации скрывают проблему от -Wclobbered
- В SAFEC есть специализированный анализ:

warning: value assigned to 'ret' might be clobbered when execution resumes [-Wclobbered]

```
6 |         ret = 0;
  |         ~~~~^~~
```

In file included from <source>:1:

note: after call to function '_setjmp' that may return twice:

```
4 |     if (!setjmp(jb)) {
  |         ^~~~~~
```

- **Надёжная оптимизация кода**
 - Не должны вноситься дополнительные уязвимости на этапе компиляции ПО (в т.ч. оптимизация кода с неопределённым поведением)
- **Диагностика**
 - Выдача предупреждений о потенциально небезопасном коде (напр., `-Wextra-safety`)
- **Улучшение защищённости кода**
 - Включение динамических методов защиты на этапе компиляции (`hardening`, `sanitizers`) — снижение степени потенциальной угрозы безопасности

Снижение критичности уязвимости

- Легковесные методы защиты, затрудняющие реализацию угроз безопасности
 - Защита от переполнения стека (-fstack-protector)
 - Защита от переполнения буфера при работе со стандартной библиотекой (FORTIFY_SOURCE)
 - Поддержка рандомизации распределения адресного пространства в компиляторе (-fPIE)
- Генерация динамических проверок в исполняемом коде для обнаружения неопределённого поведения на этапе выполнения (sanitizers)
 - Проверки на целочисленное переполнение, выход за границы массива, переполнение при преобразовании типов, деление на ноль, корректность выравнивания указателей, адресная арифметика и др.

Защита переполнения стека - fstack-protector-strong

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int checkpass(char *input)
5 {
6     char buf[16];
7     sscanf(input, "password:_%s", buf);
8
9     return !strcmp(buf, "Alohomora!");
10 }
```

<https://godbolt.org/z/hYX8Zc>

Защита переполнения стека - fstack-protector-strong

checkpass:

```
    sub    rsp, 40
    mov    esi, OFFSET FLAT:.LC0
    mov    rax, QWORD PTR fs:40
    mov    QWORD PTR [rsp+24], rax
    xor    eax, eax
    lea   rdx, [rsp+8]
    call  __isoc99_sscanf
    mov   esi, OFFSET FLAT:.LC1
    lea   rdi, [rsp+8]
    call  strcmp
    test  eax, eax
    sete  al
    mov   rcx, QWORD PTR [rsp+24]
    xor   rcx, QWORD PTR fs:40
    je    .L2
    call  __stack_chk_fail
```

.L2:

```
    movzx eax, al
    add   rsp, 40
    ret
```

Запись и проверка
признака изменения
стека (stack canary)

0x7ffe8630: ...
0x7ffe8628: return address
0x7ffe8620: (не исп.)
0x7ffe8618: stack canary
0x7ffe8610: buf[8..15]
0x7ffe8608: buf[0..7]
0x7ffe8600: (alignment)

Защита от переполнений `_FORTIFY_SOURCE`

```
1     char buf[16];
2     sprintf(buf, "username:_%s\n", input);
3
4     puts(buf);
5 }

```

Вместо вызова `sprintf` подставляется

```
1 __sprintf_chk(buf,
2     __builtin_object_size(buf),
3     "username:_%s\n", input);

```

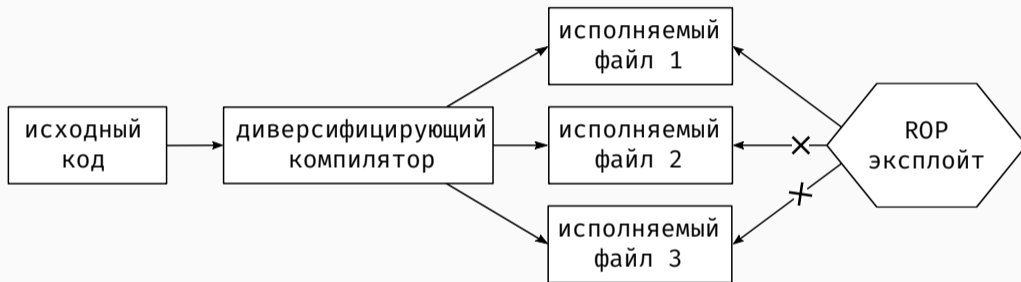
```
1 login_start:
2     sub    rsp, 24
3     mov    r8, rdi
4     mov    ecx, OFFSET FLAT:.LC0
5     xor    eax, eax
6     mov    rdi, rsp
7     mov    edx, 16
8     mov    esi, 1
9     call  __sprintf_chk
10    mov    rdi, rsp
11    call  puts
12    add    rsp, 24
13    ret

```

<https://godbolt.org/z/5WrC7D>

`_FORTIFY_SOURCE` позволяет предотвращать переполнение буфера при использовании некоторых стандартных функций

Диверсификация на этапе компиляции



Модель атаки на диверсифицированную популяцию исполняемых файлов.

Диверсификация на этапе запуска программы

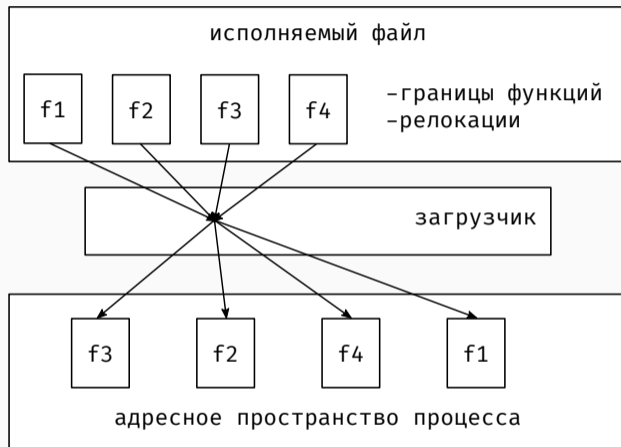


Схема работы мелкозернистой рандомизации.

Мелкозернистая рандомизация адресного пространства

нет ASLR	ASLR	мелкогранулярная ASLR
	_____ <- random part	_____ <- random part
00401000 r-xp .text	55ae5543d000 r-xp .text	55ae5543d000 r-xp .text
00401100 foo	55ae5543d100 foo	55ae5543d100 main
00401200 boo	55ae5543d200 boo	55ae5543d200 boo
00401300 coo	55ae5543d300 coo	55ae5543d300 foo
00401400 main	55ae5543d400 main	55ae5543d400 coo
00404000 rw-p .data	55ae55440000 rw-p .data	55ae55440000 rw-p .data
01794000 rw-p [heap]	55ae55bd4000 rw-p [heap]	55ae55bd4000 rw-p [heap]
7ffc13802000 rw-p [stack]	7ffe4dbe2000 rw-p [stack]	7ffe4dbe2000 rw-p [stack]

- **Предоставление готовых профилей оптимизации**

- Профили (напр., опций вида `-O2` → `-O3Safe`), включающие набор безопасных функций, с помощью которых можно выбирать уровень защиты

- **Совместимость**

- По возможности, не должен требовать изменения команд сборки, а также изменений в исходном коде программы (или такие изменения минимальны)

- **Производительность**

- Исполняемый код не должен существенно замедляться (по крайней мере, без динамических проверок)

Классы безопасности компилятора

Три класса безопасности: (самый слабый) 3 → 2 → 1 (самый сильный)

- Классы безопасности определяют баланс между производительностью и совместимостью с одной стороны, и безопасностью — с другой.
- На уровне 3 могут требоваться минимальные изменения в исходном коде пакетов
 - основной эффект — включаются в обязательном порядке имеющиеся опции GCC, соответствующие безопасной оптимизации
- На уровнях 2 и 1 могут требоваться более существенные изменения в коде
 - некоторые предупреждения становятся ошибками (-Werror), включаются санитайзеры
- В данный момент безопасный компилятор проходит апробацию на сборке популярных дистрибутивов Linux

Класс безопасности 3. Оптимизации и генерация кода

- Защита от переполнения при вызове функций libc;
 - Защита от переполнения буфера на стеке;
 - Определение семантики переполнения как перехода через 0 в дополнительном коде при операциях над знаковыми целыми и указателями;
 - Генерация позиционно-независимого кода;
 - Запрет вывода о ненулевом значении указателя при разыменовании (считать обратное допустимым);
 - Запрет оптимизаций на основании предположения о непересечении в памяти объектов разных типов;
 - Предотвращение замены функций ввода/вывода стандартной библиотеки на эквивалентные машинные инструкции;
 - Предотвращение замены функций работы с памятью стандартной библиотеки на эквивалентные машинные инструкции.
- D_FORTIFY_SOURCE=2
 - fstack-protector-strong
 - fstack-protector-exit-on-error=БК
 - fwrapv
 - fwrapv-pointer
 - fpic -fpie -fPIC
 - fno-delete-null-pointer-checks
 - fno-strict-aliasing
 - fno-builtin-printf
 - fno-builtin-sprintf
 - fno-builtin-fprintf ...
 - fno-builtin-memcpy
 - fno-builtin-memset
 - fno-builtin-strcpy ...

Класс безопасности 3. Предупреждения

- Обнаружение переменных, распределяемых на регистры, значение которых может быть изменено в результате вызова функций `longjmp()/vfork()`; `-Wclobbered`
- Обнаружение операций загрузки и записи в массив за пределами памяти, выделенной для него; `-Warray-bounds`
- Обнаружение некорректного использования битовых сдвигов; `-Wshift-count-negative`
`-Wshift-count-overflow`
- Обнаружение деления на нуль. `-Wdiv-by-zero`

Класс безопасности 2. Функции безопасности сверх класса 3

- Сохранение записей в память при наличии хотя бы одного обращения к этому участку памяти;
- Инициализация нулями неинициализированных переменных на стеке;
- Запрет оптимизаций, основанных на допустимом диапазоне значений правого операнда сдвига;
- Запрет полагаться на наличие выравнивания примитивных типов и объектов в соответствии со стандартом языка;
- Консервативный анализ алиасов: считать, что при выходе за пределы объекта указатель может ссылаться на что угодно;
- Прерывать трансляцию при выдаче предупреждений с третьего класса безопасности;
- Прерывать трансляцию при использовании функции `gets()` в исходном коде.

```
-fno-dse -fno-tree-dse  
-fno-store-merging  
-fpreserve-memory-writesБК
```

```
-finit-local-varsБК
```

```
-fkeep-oversized-shiftsБК
```

```
-fassume-unalignedБК
```

```
-finbounds-aliasingБК
```

```
-Werror=clobbered  
-Werror=shift-count...
```

Класс безопасности 1. Функции безопасности сверх класса 2

- Динамические проверки, аварийный останов при наступлении UB;
 - целочисленное или вещественное переполнение;
 - разыменованние нулевого указателя;
 - разыменованние указателя, указывающего за пределы массива;
 - деление на нуль;
 - и другие проверки из стандартного набора;
 - вызов функции по указателю, формальный тип которого не соответствует фактическому.
 - Предотвращение утечки информации через отладочные сообщения при аварийном останове;
 - Более строгий контроль типов на этапе компоновки.
- fsanitize=alignment,bool,bounds,builtin,float-cast-overflow,float-to-float-cast-overflow^{БК},integer-divide-by-zero,nonnull-attribute,null,pointer-overflow,return,returns-nonnull-attribute,shift,signed-integer-overflow,unreachable,vla-bound
 - fsanitize=function^{БК}
 - fsanitize-undefined-exit-on-error=^{БК}
 - fno-sanitize-function-emit-type-descriptors^{БК}
 - fno-common

Класс безопасности 1. Диверсификация

- Уникальное распределение машинного кода функций в процессе динамической компоновки;
- В отсутствие поддержки со стороны динамического загрузчика — статическая рандомизация.

-fdynamic-func-reorder^{БК}
-fadd-loc-var=<количество>^{БК}

-floc-var-per^{БК}
-frandom-func-reorder^{БК}

-frandom-func-and-globals-reorder^{БК}
-flayout-random-seed=<значение>^{БК}

Производительность безопасного кода

Сценарий	GCC	-Safe3	-Safe3 замедл.	-Safe2	-Safe2 замедл.	-Safe1	-Safe1 замедл.
GNU GO	4,67 с	4,85 с	3,85%	5,23 с	11,99%	9,32 с	99,57%
LAME	3,45 с	3,55 с	2,89%	3,55 с	2,83%	9,89 с	186,31%
fannkuch	2,03 с	2,42 с	19,21%	2,42 с	19,21%	3,51 с	72,91%
x264	7,91 с	7,90 с	-0,23%	8,06 с	1,78%	18,30 с	131,21%
zlib	2,25 с	2,29 с	2,00%	2,28 с	1,33%	2,80 с	24,44%

[2]

Производительность безопасного кода. SPEC CPU® 2017

Режим		intspeed	intrate	fpspeed	fprate
-O2		x	x	x	x
-Safe3	-O2	0,963x	0,978x	x	0,981x
-Safe2	-O2	0,959x	0,978x	x	0,981x
-Safe1		—	—	—	—

Базовые опции компиляции для C:



```
-m64 -std=c99 -O2 -march=native -fPIC  
-fno-strict-aliasing  
-fno-unsafe-math-optimizations  
-fno-finite-math-only -fgnu89-inline
```

Базовые опции компиляции для C++:

```
-m64 -std=c++03 -O2 -march=native -fPIC
```

- -Safe3 intspeed: 0,80x на 623.xalancbmk_s
- -Safe3 intrate: 0,93x на 523.xalancbmk_r
- -Safe3 fprate: 0,86x на 511.povray_r
- Для -Safe1 тесты не удовлетворяют требованиям безопасности (SPEC проверяет целостность исходных кодов)

- Комплексный подход к обеспечению безопасности ПО должен также включать компилятор
- Нами разрабатывается безопасный компилятор (SAFEC) на основе GCC, предлагающий решения для некоторых известных проблем безопасности на этапе компиляции и оптимизации кода
- Работа велась совместно с разработкой ГОСТ по безопасной разработке (статика, динамика, компилятор)
- В данный момент безопасный компилятор проходит апробацию на сборке популярных дистрибутивов Linux

-  Jonathan Corbet.
Fun with null pointers, part 1.
<https://lwn.net/Articles/342330/>.
-  Баев Р.В., Скворцов Л.В., Кудряшов Е.А., Буцацкий Р.А., and Жуйков Р.А.
Предотвращение уязвимостей, возникающих в результате оптимизации кода с неопределенным поведением.
In Труды ИСП РАН, том 33, вып. 4, 2021 г.
-  Xi Wang, Haogang Chen, Alvin Cheung, Zhihao Jia, Nickolai Zeldovich, and M. Frans Kaashoek.
Undefined behavior: What happened to my code?
In Proceedings of the Asia-Pacific Workshop on Systems, 2012.

Вопросы и ответы

Дмитрий Мельник
dm@ispras.ru

Безопасный компилятор SAFEC
<https://www.ispras.ru/technologies/safecomp/>

[Bonus] «Обратная» дедукция на основе наличия UB

```
1 int f(int c, int *failed)
2 {
3     if (c) {
4         *failed = 0;
5         return 42;
6     }
7     *failed = 1;
8 }
```

Программа с UB?

<https://godbolt.org/z/vvYfEo4fs>

```
1 f:
2     xor     eax, eax
3     test   edi, edi
4     sete   al
5     mov   dword ptr [rsi], eax
6     mov   eax, 42
7     ret
```

clang -xc

```
1 f(int, int*):
2     mov   dword ptr [rsi], 0
3     mov   eax, 42
4     ret
```

clang -xc++

[Bonus] Польза UB

```
1 int k;  
2 int *ic, *is;  
3 ...  
4 for (k = 1; k <= M; k++) {  
5     ...  
6     ic[k] += is[k];  
7     ...  
8 }
```

Фрагмент теста 456.hmmeg

- Предположение о том, что k++ не переполняется, позволяет компилятору переписать цикл с использованием 64-битной переменной (на x86-64) в качестве счётчика цикла (вместо использования операции знакового расширения для доступа к массиву).
- M может иметь значение INT_MAX.
- Запрет этого предположения приводил к замедлению всего теста на 7.2% для GCC и на 9.0% для Clang[3].