ИСП РАН

Вопросы безопасной компиляции программного обеспечения

Андрей Белеванцев

Институт системного программирования
им. В.П. Иванникова РАН

13 февраля 2019 г.

Разработка безопасного ПО

ИСП РАН

Исходный код	Сборка кода	Бинарный код
• Статический анализ • Архитектура с учетом безопасности	• Компилятор • Ассемблер • Компоновщик • ???????	• Фаззеры • Динамический анализ (символьное выполнение) • Тестирование на проникновение

- Уязвимости в программе могут появляться не только из-за ошибок в ее коде, но и в результате оптимизаций, выполняемых компилятором
- В набор средств для обеспечения безопасности ПО должны быть также включены средства разработки (компилятор)

Пример небезопасной оптимизации

ИСП РАН

```
char *buf = ...;
char *buf_end = ...;
unsigned int len = ...;
if (buf + len >= buf_end)
  return;
/* len too large */
if (buf + len < buf)
  return;
/* overflow, buf+len wrapped around */
/* write to buf[0..len-1] */
```

Проверка на принадлежность *buf+len* границам области [*buf, buf_end*]

- При выполнении оптимизаций компилятор в соответствии со стандартом языка может полагаться на отсутствие в программе неопределенного поведения
- Выделенный участок кода полагается на ***переполнение*** при выполнении арифметических операций с указателями, что является ***неопределенным поведением*** в соответствии со стандартом языка Си
- Указанная проверка может быть удалена при выполнении оптимизаций компилятором, как избыточная
- Удаление проверки границ перед записью в память может привести к появлению эксплуатируемой уязвимости

ИСП РАН

Пример небезопасной оптимизации (2)

```
unsigned int
tun_chr_poll(struct file *file, poll_table *wait)
{
  struct tun_file *tfile = file->private_data;
  struct tun_struct *tun = __tun_get(tfile);
  struct sock *sk = tun->sk;
  if (!tun)
    return POLLERR;
  ...
}
```

Разыменование предшествует проверке указателя на NULL

- Разыменование нулевого указателя в соответствии со стандартом языка Си является неопределенным поведением, и компилятор может считать, что если разыменование уже произошло, то далее указатель ненулевой
- Выделенный код с проверкой может быть удален компилятором, как избыточный
- Удаление проверки указателя может привести к появлению эксплуатирумых уязвимостей

Пример небезопасной оптимизации (3)

```
char * password = malloc(PASSWORD_SIZE);
// ... read and check password
memset(password, 0, PASSWORD_SIZE);
free(password);
```

- С точки зрения компилятора, запись нулей в массив с паролем является избыточной, т.к. далее в программе нет обращения к этому массиву, поэтому вызов функции *memset()* может быть удален в ходе оптимизации
- Удаление операций очистки массива может привести к утечке конфиденциальных данных

Задачи безопасного компилятора (C/C++)

- Безопасная оптимизация кода
  - Отсутствие внесения дополнительных уязвимостей на этапе компиляции ПО (в т.ч. оптимизация кода с *неопределенным поведением*)
- Диагностика
  - Выдача предупреждений о потенциально небезопасном коде (напр., -Wextra-safety)
- Снижение степени потенциальной угрозы безопасности
  - Применение методов защиты на этапе компиляции (hardening, sanitizers)
- Предоставление готовых профилей оптимизации
  - Профили для сборки ПО, выполняющие безопасные оптимизации, в которых можно варьировать степень защищенности

Предметная область

ИСП РАН

- Анализ предыдущих исследований
- Анализ случаев неопределенного поведения, перечисленных в стандарте языка Си
- Анализ уязвимостей из базы CVE, связанных с оптимизацией программ на этапе компиляции
- Экспериментальные исследования современных версий компилятора GCC и Clang

Безопасная оптимизация

ИСП РАН

- Отсутствие внесения в исполняемый код дополнительных уязвимостей по сравнению с исходным кодом
  - Безопасная оптимизация кода, содержащего неопределенное поведение
  - Сохранение побочных эффектов операций записи в память (гарантия сохранения в ходе оптимизаций кода, очищающего конфиденциальные данные)
  - Доопределение неинициализированных переменных нулевыми значениями
- Минимизация потерь в производительности из-за отключения небезопасных оптимизаций
  - Стандартного уровня оптимизации “-O0” компиляторов gcc/clang недостаточно

Требования к оптимизациям (примеры)

- При выполнении различных видов анализа и оптимизаций компилятор не должен полагаться на отсутствие в программе заданных видов неопределенного поведения
  - Значения указателей различающихся типов могут совпадать при операциях загрузки и сохранения данных через эти указатели
  - Указатель, по которому происходит запись/чтение данных может содержать нулевое значение во всех случаях, если только с помощью анализа условий выполнения разыменования указателя не может быть доказано обратное
  - Вызовы функций стандартных библиотек не могут быть заменены на эквивалентные им последовательности машинных инструкций
  - При выполнении операций над целыми числами может возникнуть целочисленное переполнение, приводящее к исключительной ситуации, во всех случаях, когда не доказано обратное

Диагностика

ИСП РАН

- Предупреждения о выполнении потенциально небезопасных преобразований на этапе компиляции
- Предупреждения о наличии в коде конструкций с неопределённым поведением (не всегда возможно на этапе компиляции)

Требования к диагностике (примеры)

- Обнаружены пути выполнения программы, на которых происходят операции разыменования указателя, имеющего нулевое значение
- Обнаружена загрузка значения переменной автоматического класса памяти, которая не была ранее инициализирована
- Обнаружена переменная автоматического класса памяти, которая хранится на машинном регистре перед вызовом функции longjmp, и ее значение используется после вызова этой функции путем загрузки из соответствующего регистра
- Обнаружена операция загрузки или записи в массив, адрес которой находится за пределами выделенной памяти для этого массива

Снижение критичности уязвимости

- Данные методы позволяют снизить уровень угрозы безопасности с очень высокого (исполнение произвольного кода) до более низкого (отказ в обслуживании)
  - Легковесные методы защиты, затрудняющие реализацию угроз безопасности
  - Генерация динамических проверок в исполняемом коде для обнаружения неопределенного поведения на этапе выполнения (sanitizers)
    - Динамические проверки существенно влияют на производительность, поэтому их использование оправдано только в системах высокого класса защищенности
    - Не всегда приемлемо для критически важных систем

Снижение критичности уязвимости (2)

- Легковесные методы защиты, затрудняющие реализацию угроз безопасности
  - Защита от переполнения стека (-fstack-protect)
  - 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 неопределенного 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.

Требования к динамическому контролю (примеры)

ИСП РАН

- операция загрузки значения в переменную булевого типа, которое отлично от лжи и истины
- операция преобразования между значениями переменных вещественного типа с плавающей запятой, приводящая к вещественному переполнению
- операция деления на переменную вещественного или целого типа, значение которой равно нулю
- операция целочисленного сдвига, в которой второй аргумент операции сдвига (величина сдвига) отрицателен либо не меньше ширины типа (в битах) первого аргумента операции (сдвигаемого значения)
- операция с целочисленными переменными, результат которой не может быть представлен в результирующей переменной
- операция загрузки или записи по указателю, значение которого не кратно размеру типа хранящегося по этому адресу объекта
- операция загрузки или записи по указателю, значение которого равно нулю
- операция загрузки или записи в массив по адресу, находящемуся за пределами выделенной для массива памяти, если размер памяти может быть определен в процессе трансляции
- ...

Квалификационные тесты для безопасных компиляторов

- Тесты времени выполнения и ассемблерные тесты проверяют, что компилятор в ходе оптимизаций не выполняет преобразований, основанных на предположении об отсутствии в программе случаев неопределенного поведения
  - В тестах времени выполнения проверяется результат выполнения тестовой программы, содержащей неопределенное поведение
  - В ассемблерных тестах проверяется отсутствие некорректных преобразований, подходящих под требование, **при этом программа не обязательно содержит неопределенное поведение**

Квалификационные тесты для безопасных компиляторов (2)

- Тесты на производительность проверяют замедление реальных программ, скомпилированных в безопасном режиме
- Тесты наличия предупреждений проверяют, что компилятор выдает предупреждение на этапе компиляции в случае нарушения требований безопасности
- Тесты, выполняемые с динамическим анализатором, проверяют способность компилятора выдать ошибку при обнаружении условий неопределенного поведения во время работы программы (теста)

Выводы

- Проблема возникновения уязвимостей на этапе сборки актуальна и может быть закрыта для языков C/C++ безопасными средствами разработки (компилятором)
- Ведутся работы по созданию требований к компилятору, опытного образца на базе GCC, квалификационных тестов
- Для других языков должны быть выполнены схожие работы, решаемые задачи в средствах разработки могут быть иного плана (нет неопределенного поведения)