УДК 004.94

ПОИСК ПРОГРАММНЫХ ОШИБОК В АЛГОРИТМАХ ОБРАБОТКИ СЛОЖНО-СТРУКТУРИРОВАННЫХ ДАННЫХ

А. Н. Макаров

При разработке программных комплексов часто решается задача по интеграции в них программных модулей, для которых отсутствуют исходные коды и сопроводительная техническая документация. Для обеспечения надежности функционирования комплекса в целом может потребоваться выполнить анализ бинарного кода используемых модулей.

В данном случае под анализом бинарного кода подразумевается проверка корректности работы программного обеспечения (ПО) и отсутствия программных ошибок. Существуют различные классификации ошибок, встречающихся в ПО [1, 2]. Далее считаем, что *программная ошибка* — это ошибка реализации ПО, допущенная разработчиками на этапе кодирования. Проявлением программной ошибки является аварийное завершение процесса, связанного с соответствующим ПО.

Сократим анализ бинарного кода. Исследоваться будут только те программные модули (или их части), которые отвечают за обработку входных данных. Этому есть объяснение. Разрабатываемая и используемая методика тестирования бинарного кода [3] хорошо себя зарекомендовала для тестирования ПО, которое обрабатывает сложно-структурированные входные данные. Именно при создании программного кода, обрабатывающего сложно-структурированные данные, со многими внутренними связями (явными и неявными), на качестве кода сказывается человеческий фактор.

Применяют несколько основных подходов, которым отдается предпочтение при решении задачи тестирования бинарного кода [4, 5]:

— обратная инженерия (*reverse engineering*) — применяется с целью получения ПО на языке ассемблера или на языке высокого уровня;
— анализ двоичного кода — предполагает наличие анализирующего приложения, которое читает собранное ПО и просматривает его с применением некоторых эвристических правил;
— тестирование нагрузкой, или стрессовое тестирование — используется набор файлов сценариев, которые посылают ПО разнообразные входные данные различного размера и структуры.

В последнее время получил распространение один из видов стрессового тестирования — фазинг (*fuzzing*) [1]. Его преимущество — возможность быстрого получения результатов.

Тестирование алгоритмов обработки сложно-структурированных данных, основанное на стрессовом тестировании, можно представить в виде последовательности тестов. Каждый тест выполняется за четыре шага.

Шаг 1. С помощью процедур формирования входных данных подготавливаются входные данные, которые будут переданы исследуемому процессу.

Шаг 2. Запускается исследуемый процесс и ему передаются сформированные на первом шаге входные данные.

Шаг 3. Регистрируется состояние исследуемого процесса. Если исследуемый процесс на переданных ему входных данных завершает работу аварийно, то собранная информация о работе процесса поможет разобраться в причинах ошибки.

Шаг 4. На последнем шаге теста, в случае аварийного завершения исследуемого процесса, восстанавливается корректность его последующих запусков для независимого выполнения очередного теста.

Применение стрессового тестирования позволяет обнаружить программные ошибки за короткое время.

Первоначально входные данные для тестирования формировались в статичном режиме, на основе различных эвристических правил и без восстановления алгоритмов работы исследуемого ПО.

Для повышения эффективности тестирования было решено сочетать стрессовое тестирование и динамический анализ на основе трассировки исследуемого процесса. Трассировка помогает сгенерировать входные данные.

При этом возникают две взаимосвязанные задачи. Первая — сопоставление входных данных и результатов трассировки, для чего строится граф потока данных.

Вторая задача — анализ собранной трассы, при этом можно выделить различные подзадачи: обнаружение функций, «схлопывание» циклов, минимизация объемов хранимых данных (поскольку объемы трассы измеряются гигабайтами) и другие. Для решения данной задачи предполагается использовать возможности среды *Ida Pro*. Интеграция со средой *Ida Pro* позволит минимизировать объем трассы и решить ряд подзадач с помощью штатных средств дизассемблера.

Таким образом, выявление программных ошибок в ПО без исходных текстов путем дополнительного тестирования механизмов обработки входных данных позволяет повысить надежность разрабатываемых программных комплексов.

ЛИТЕРАТУРА

1. *Козиол Д., Личфилд Д., Эйтэл Д., и др.* Искусство взлома и защиты системы. СПб.: Питер, 2006. 416 с.
2. *Ховард М., Лебланк Д.* Защищенный код. 2-е изд. М.: Издательско-торговый дом «Русская редакция», 2005. 704 с.
3. *Макаров А. Н.* Метод автоматизированного поиска программных ошибок // Безопасность информационных технологий. Вып. 2. М.: МИФИ, 2008. С. 101–104.
4. *Хогланд Г., Мак-Гроу Г.* Взлом программного обеспечения: анализ и использование кода. М.: Издательский дом «Вильямс», 2005. 400 с.
5. *Eilam E.* Reversing: Secrets of Reverse Engineering. Wiley Publishing, 2005. 589 p.

УДК 004.738

МАРШРУТИЗИРУЕМЫЙ СЕРВИС ПЕРЕДАЧИ ДАННЫХ

В. И. Никонов

Настоящая работа продолжает исследование [1], посвященное разработке алгоритмов разделения данных в распределенных сетях. Этот метод выступает в качестве альтернативы снижению вычислительных затрат при использовании шифрования.

Одним из видов активных сетевых атак является класс атак, основанных на снифинге [2]. Приведем пример, в котором злоумышленник, обладая знаниями, что некоторая организация регулярно передает данные из A в G, может довольно точно определить маршрут от A до G в момент времени Δt и осуществить перехват на каком-нибудь из участков следования трафика (см. рис. 1,a).