УДК 004.056

Е.Л. Евневич, Р.Р. Фаткиева

ИЗМЕРЕНИЕ БЕЗОПАСНОСТИ ПРОГРАММНОГО КОДА

Евневич Е.Л., Фаткиева Р.Р. **Измерение безопасности программного кода.**
Аннотация. Динамический анализ особенно востребован для программного кода большого объема и сложности, когда построение абстрактной модели не представляется возможным. В данной статье предлагается метод применения процедуры динамического анализа к поиску уязвимостей кода. Для каждой обнаруженной уязвимости определяются значения метрик безопасности и принимается решение о целесообразности и срочности устранения проблемы.
Ключевые слова: фаззинг, метрики безопасности, тестирование программного кода.

Evnevich E.L., Fatkieva R.R. **Measurement of program code security.**
Abstract. Program code being of large amount and complexity, development of abstract model being impossible, dynamic analysis becomes of special importance. In this paper a technique is proposed for code vulnerabilities detection by means of dynamic analysis procedure. For each detected vulnerability risk assessment is carried out, some security metrics are applied to and decision is made concerning purposefulness and urgency of vulnerability problem solution.
Keywords: fuzzing, security metrics, program code testing.

1. Введение. Проблема измерения безопасности информационно-вычислительных систем (ИС) обусловлена огромным многообразием подсистем программного и аппаратного обеспечения, входящего в ее структуру. Практически любая информационно-вычислительная система существует как часть более крупной системы, информационные потоки которой находятся в постоянном взаимодействии и оказывают влияние на внутреннее состояние системы. Нелинейность поведения ИС связана как с нелинейностью программного кода, так и с наличием системы прерываний, что затрудняет анализ и измерение безопасности. Трудность оценивания и анализа внутреннего состояния системы обусловлена огромным многообразием вероятностных состояний. Проблема оценивания и прогнозирования поведения усугубляется дефектами программного обеспечения, связанными со сложностью программного кода. Наличие циклов и переходов, большое количество подпрограмм затрудняют поиск дефектов. Ошибки проектирования, связанные с недостатком взаимопонимания между заказчиком и исполнителем, отсутствием формализованной модели программного обеспечения порождают программные дефекты. Разработка программных продуктов в условиях конкуренции ставит производителей в жесткие временные рамки, что заставляет выпускать продукцию без тщательного тестирования. Ошибки, пропущенные в ходе тестирования, например, переполнение буфера или ввод «неправильных» дан-

ных, в некоторых ситуациях приводят к сбою программы, проявлению уязвимостей программного обеспечения. Невозможность анализа отдельно взятых фрагментов программы (обычно отдельных функций или процедур) дает нелинейный рост сложности анализа и поиска уязвимостей. Все вышеперечисленное приводит к трудности моделирования поведения программного обеспечения, поэтому возникает необходимость в разработке методов тестирования и поиска уязвимостей программного кода.

Поиск уязвимостей программного обеспечения может производиться методами статического и динамического анализа. Зачастую при построении абстрактных моделей методами статического анализа такие модели не являются исчерпывающими вследствие проблемы проверки истинности обнаруженного дефекта и воспроизведения найденного дефекта при запуске программы на определенных входных данных. В данной статье речь пойдет о динамических методах. Динамический анализ обнаруживает дефекты, определяемые конкретными входными данными в процессе работы программного обеспечения, однако дефекты, находящиеся в других частях программного кода, могут быть при этом не обнаружены. Актуальность динамического анализа обусловлена постоянным увеличением объема и сложности программного кода, в этом случае любые автоматические средства обнаружения ошибок и контроля качества являются полезными и востребованными.

2. Этапы процесса обнаружения уязвимости. Вопрос выбора входных данных и их количества достаточно важен при поиске дефектов, от успешности которого зависит эффективность динамического анализа [1]. В работах [2, 3] предлагается подход наибольшего покрытия исполняемого кода, что приводит к более глубокому его анализу. Также существует проблема определения и выбора этапа, на котором целесообразно закончить тестирование для заданного критерия эффективности [4, 5]. В зависимости от разных условий и задач этапы анализа могут варьироваться очень широко, однако можно выделить некоторые наиболее общие фазы:

- *Определение цели тестирования.* В зависимости от исследуемого объекта (программа, база данных, web-сервер и т.п.) возможно различие в выбранных инструментах и тактиках.
- *Определение вводимых значений*. Как правило, все уязвимости, которые можно обнаружить, связаны с некорректной обработкой входного потока данных. Невозможность определить потенциальные источники ввода или ожидаемые значения ввода может серьезно ограничить возможности тестирования.

- *Выбор инструмента тестирования.* Исследование публикаций, посвященных выявлению уязвимостей с помощью динамического анализа [1–7], показало наличие целого ряда специализированных программных продуктов, работающих со многими известными и задокументированными сетевыми протоколами и файловыми форматами. Применение готовых программных продуктов вполне эффективно при работе с целым рядом популярных приложений, однако затруднено для ранее не тестированных или редких протоколов. В этих случаях оправданным является использование так называемых интегрированных сред или фреймворков.
- *Выбор метрик.* Метрики предоставляют средство описания сложных типов данных и кодирования (например, метрики сложности кода), а также средство оценки эффективности процесса тестирования.
- *Тестирование программного кода.* Исследование объекта путем отсылки в объект пакета сгенерированных данных и мониторинг состояния системы. Данный шаг заключается в регистрации всех передаваемых пакетов, а также реакций системы на некорректные данные.
- *Изучение и анализ всех обнаруженных сбоев.* В случаях аварийного завершения работы при помощи монитора процесса создается отчет, который может содержать следующую информацию: инструкции, вызвавшие нарушения прав доступа, а также их месторасположение; дамп всех регистров; дизассемблерные инструкции; список обработчиков SEH, которые были зарегистрированы во время неисправности и др.
- *Оценка эффективности применяемого метода*. С помощью выбранных на более раннем этапе метрик оценивается степень выявления основных возможных видов уязвимостей, обнаруженных при тестировании.

3. Обнаружение уязвимостей на примере тестирования Web-сервера Simple Web Server.

Определение цели тестирования. Целью являлось исследование программного кода для обнаружения уязвимостей динамическим методом с помощью инструментальных средств анализатора Sulley framework.

Определение вводимых значений. Для нахождения уязвимостей в качестве входных данных возможно использование: статических и случайных примитивов; целых чисел; строк и разделителей; блоков и

групп. Использование блочного подхода для генерации отдельных запросов, которые соединяются воедино для формирования сессии с последующим добавлением к запросу примитивов, блоков и вложенных блоков, позволяет осуществить цикл всех возможных мутаций для каждого значения внутри группы. Группирование осуществляет привязку блока к групповому примитиву для указания того, что блок должен пройти цикл всех возможных мутаций каждого значения внутри группы.

Выбор инструмента тестирования. Среди существующих программных сред тестирования можно выделить Sulley fuzzing framework как наиболее мощное средство для разработки собственных анализаторов, обладающее следующими преимуществами: автоматизация процесса мониторинга, обнаружение, отслеживание и классификация ошибок, модульная структура и открытый исходный код и др.

Выбор метрик. Оценку эффективности применяемого метода предполагается выполнить с помощью метрики покрытия кода.

Тестирование программного кода производилось на Web-сервере Simple Web Server с заданным по умолчанию портом TCP — 80. Рабочая станция имела следующую конфигурацию: процессор Intel® Core™ 2 Duo с тактовой частотой 2.13 Ггц; 2 Гбайт оперативной памяти; 250 Гбайт жесткий диск; операционная система Windows Vista SP2; виртуальная машина VMware Oracle, установленная на хостовой операционной системе. Программное обеспечение анализатора Sulley framework установлено на хостовой и гостевой системах. На гостевой системе запущены мониторы: process_monitor.py (монитор процесса), отвечающий за обнаружение ошибок, которые могут произойти в объектном процессе во время выполнения проверки, и network_monitor.py (сетевой монитор), производящий мониторинг сетевых коммуникаций и запись результатов на диск. Агент монитора процесса принимает соединения от сессии для передачи каждого отдельного контрольного примера с целью обнаружения ошибки, при которой высокоуровневая информация, касающаяся характера ошибки, передается для отображения сессии с помощью внутреннего веб-сервера. Инициализированные ошибки также записываются в последовательно преобразованную «корзину сбоев» для проведения постпрограммного анализа.

Управляющий агент VMWare принимает соединения от сессии для взаимодействия с образом виртуальной машины и позволяет управлять этим образом (приостанавливать или перезагружать образ, выполнять, удалять и восстанавливать моментальные снимки). В случае обнаружения ошибки или если объект не может быть достигнут,

анализатор может установить соединение с данным агентом и вернуть виртуальную машину к заведомо исправному состоянию. Инструмент точного определения последовательности проверки широко использует этого агента для выполнения задания по определению точной последовательности контрольных примеров, приводящих к инициализации той или иной комплексной ошибки.

Изучение и анализ всех обнаруженных сбоев. Для просмотра отчета об ошибках и дальнейшего анализа в Sulley используется утилита crashbin_explorer.py, позволяющая просмотреть место, где была обнаружена ошибка и составить список номеров отдельных контрольных примеров, приведших к инициализации ошибки в этом адресе. В результате работы фаззера было сгенерировано 29 760 тестов. Первое приложение, как и ожидалось, аварийно завершило работу при отправке пакета размером больше 500 бит. Для Web-Server было зафиксировано 21 аварийное завершение, каждое из которых описывается двумя файлами: pcap файл с запросом и файл, содержащий информацию обо всех сбоях с данными об адресах и трассировке стека. Среди запросов, сгенерированных анализатором и вызвавших падение сервера, встречаются такие (рис. 1):

```
Повторение много раз запроса:
…
GET /index.htm HTTP/1.1
Host: 192.168.56.102
…
Повторение строки статуса:
GET /index.htm HTTP/1.1
……
```

Рис. 1. Содержимое запроса на получение данных.

Другие сообщения возникли после данных запросов, следовательно, являются следствием возникших ранее сбоев. Сбой в работе тестируемой программы является уязвимостью отказа в обслуживании. Для углубленного изучения состояния центрального процессора в момент обнаружения сбоя рассмотрим сгенерированное сообщение об ошибках (рис. 2).

```
(1)[INVALID]:41414141 Unable to disassemble at 41414141 from thread 252
caused access violation
        when attempting to read from 0x41414141
(2)CONTEXT DUMP
        EIP: 41414141 Unable to disassemble at 41414141
        EAX: 00000000 (         0) -> N/A
        EBX: 008d3b60 (   9255776) -> 0x@0^C N/A
```

```
        EDX: 0043fe00 (   4455936) -> SC:\ Simple Web Server\ERROR.TXT
(Simple Web Server.exe.data)
        EDI: 00000000 (         0) -> N/A
        ESI: 008d3b60 (   9255776) -> 0x@0^C N/A
        ESP: 00c3ea60 (12839520) -> ;;
        GET/AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
(stack)
        +00: 00c3fe00 (  12844544) -> >@d8e@d8e>`; (stack)
        +04: 00c3eab8 (  12839608) -> GET (stack)
        +08: 00000000 (         0) -> N/A
        +0c: 008d3b60 (   9255776) -> 0x@0^C 0x@0^C N/A
(3)disasm around:
        0x41414141 Unable to disassemble
(4)SEH unwind:
        00c3ffdc -> Simple Web Server.exe:00418468 push ebp
        ffffffff -> kernel32.dll:7c839aa8 push ebp
```

Рис. 2. Отчет об ошибках.

Сообщение можно разбить на несколько частей для последующего анализа: часть (1) показывает инструкцию, вызвавшую нарушение прав доступа, и ее местонахождение. Вторая часть (2) является дампом всех регистров; в данном случае видно, что регистр EIP перезаписан шестнадцатеричным значением прописной буквы A, и вероятно, находится под контролем, что делает возможным инъекции байта кода. Третья часть (3) содержит дизассемблерные инструкции до и после вызывающей ошибку инструкции. И последняя часть (4) содержит список обработчиков SEH, которые были зарегистрированы во время неисправности.

Оценка эффективности применяемого метода была выполнена с помощью метрики покрытия кода. Данная метрика основана на количестве исполненных базовых блоков. За базовый блок принимается совокупность инструкций, причем каждая инструкция в блоке заведомо выполняется по порядку, если выполнена первая инструкция блока. В ходе исследования были получены следующие данные: покрытие базовых блоков — 19, общее количество блоков — 32.

4. Заключение. Динамический анализ позволяет успешно обнаружить явные уязвимости и необходим при разработке и тестировании программного обеспечения с большим объемом кода и/или с повышенными требованиями к безопасности. С другой стороны, динамический анализ плохо справляется с обнаружением нетипичных маловероятных уязвимостей, а также целого класса уязвимостей, связанных с

ошибками проектирования. Исследования применения динамического анализа показали его эффективность на этапе разработки, при тестировании отдельных модулей программного кода. Использование динамических методов анализа с утилитами, предназначенными для контроля покрытия кода, трассировки программных ветвей, позволят достигать более глубоких слоев приложения, увеличивая тем самым количество обнаруженных уязвимостей.

Литература

1. *Тихонов А. Ю., Аветисян А. И.* Комбинированый (статический и динамический) анализ бинарного кода// Труды Института системного программирования РАН, том 22, 2012 г. С. 131–152.
2. *Котляров В.П.* Основы тестирования программного обеспечения: учеб. пособие / В.П.Котляров, Т.В.Коликова. М.: БИНОМ. Лаб. знаний: Интернет-Ун-т информ. технологий, 2006. 285 с.
3. *Благодаренко А.В.* Метод обеспечения и оценки покрытия кода тестовыми данными. //VI Ежегодная научная конференция студентов и аспирантов базовых кафедр Южного научного центра РАН: Тезисы докладов (19-30 апреля 2010 г., г. Ростов-на-дону). Ростов н/Д: Изд-во ЮНЦ РАН, 2010. С. 93–94.
4. *Воробьев В.И., Фаткиева Р.Р.* Динамический метод обнаружения уязвимостей // Информационно-измерительные и управляющие системы. 2009. Т. 7. № 11. С. 28–31.
5. *Фаткиева Р.Р., Помецко В.В.* Метод идентификации узвимостей программного кода. Информационно-измерительные и управляющие системы. 2010. Т. 8. № 7. С. 55–59
6. Искусство взлома и защиты систем / Козиол Дж., Личфилд Д., Эйтэл Д., Энли К. [и др.]. СПб/ Питер, 2006. 416 с.
7. *Афонин А.Ю.* Тестирование программного обеспечения посредством фаззинга // Новые информационные технологии и системы: труды IX Международной научно-технической конференции ч. 2. Пенза: Изд-во ПГУ, 2010. С. 113–118.

Фаткиева Роза Равильевна — канд. техн. наук; старший научный сотрудник лаборатории информационно-вычислительных систем, СПИИРАН. Область научных интересов: моделирование информационных систем. Число научных публикаций — 26. rikki2@yandex.ru; СПИИРАН, 14-я линия В.О., д. 39, г. Санкт-Петербург, 199178, РФ; р.т. +7(812)328-4369, факс +7(812)328-4450.

Fatkieva Rosa Ravilievna— Ph.D., senior researcher, Laboratory of Computer and Information Systems, SPIIRAS. Research interests: modeling of information systems. Number of publications — 26. rikki2@yandex.ru; SPIIRAS, 39, 14-th Line V.O., St. Petersburg, 199178, Russia; office phone +7(812)328-4369, fax +7(812)328-4450.

Евневич Елена Людвиговна - канд. физ.-мат. наук; старший научный сотрудник лаборатории информационно-вычислительных систем, СПИИРАН. Область научных интересов: автоматизация разработки параллельных и распределенных программ, системы электронного документооборота. Число научных публикаций — 25. eva@iias.spb.su;

СПИИРАН, 14-я линия В.О., д. 39, г. Санкт-Петербург, 199178, РФ; р.т. +7(812)328-3112, факс +7(812)328-4450.

Evnevich Elena Lyudvigovna – Ph.D., senior researcher, Laboratory of Computer and Information Systems, SPIIRAS. Research interests: parallel and distributed developments automation, e-documents systems. Number of publications — 25. eva@iias.spb.su; SPIIRAS, 39, 14-th Line V.O., St. Petersburg, 199178, Russia; office phone +7(812)328-3112, fax +7(812)328-4450.

Рекомендовано лабораторией информационно-вычислительных систем СПИИРАН, заведующий лабораторией Воробьев В.И, д-р техн. наук, проф.
Статья поступила в редакцию 05.03.2012.

РЕФЕРАТ

Евневич Е.Л.,Фаткиева Р.Р. **Измерение безопасности программного кода.**

Поиск уязвимостей программного обеспечения может производиться как методами статического, так и динамического анализа. При построении абстрактных моделей методами статического анализа модели не являются исчерпывающими вследствие проблемы проверки истинности обнаруженного дефекта и воспроизведения найденного дефекта при запуске программы на определенных входных данных. Динамический анализ обнаруживает дефекты, определяемые конкретными входными данными в процессе работы программного обеспечения. Актуальность методов динамического анализа обусловлена постоянным увеличением объема и сложности программного кода.

В зависимости от разных условий и задач этапы анализа могут значительно различаться, однако можно выделить некоторые наиболее общие. К ним относятся определение цели и инструмента тестирования, выбор анализатора и метрик, тестирование программного кода, изучение и анализ всех обнаруженных сбоев, оценка эффективности применяемого метода. Этапы предложенного метода проиллюстрированы на примере тестирования Web-сервера Simple Web Server.

Динамический анализ позволяет успешно обнаружить явные уязвимости и необходим при разработке и тестировании программного обеспечения с большим объемом кода и/или с повышенными требованиями к безопасности. Исследования применения динамического анализа показали его эффективность на этапе разработки, при тестировании отдельных модулей программного кода.

SUMMARY

Evnevich E.L., Fatkieva R.R. **Measurement of program code security**

Software vulnerabilities may be detected by static as well by dynamic analysis techniques. Abstract static models are not comprehensive due to the problem of fixed defect verification and its reproduction during program run with certain input data. Dynamic analysis is capable to detect software defects accounted for specific input data and under implementation. Dynamic analysis is of special importance as far as amount and complexity of program code constantly increase.

Depending on different types of problems and conditions the stages of analysis may vary to a considerable extent, meanwhile there will always be the following ones: determination of testing purpose and tools, choice of analyzer and metrics, code testing, investigation and analysis of all failures, efficiency assessment of proposed technique. The stages are illustrated by an example of testing of web server “Simple Web Server”.

Dynamic analysis makes possible apparent vulnerabilities detection and is of special need under development and testing of software code of large amount and/or with strong requirements to security. All the experience of dynamic analysis technique application proves its efficiency at the stages of software development and separate modules testing.