ПОИСК УЯЗВИМОСТЕЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ НА ОСНОВЕ СТАНДАРНЫХ СРЕДСТВ ФАЗЗИНГА

Влацкая И.В., Гаврилов Н.С., Надточий Н.С.
Оренбургский государственный университет, г. Оренбург

Современные информационные системы (ИС) являются важной составляющей основы построения бизнес-процессов компаний и предприятий различных форм и назначений. Присущая для всех ИС уязвимость заставляет уделять особое внимание их защите от несанкционированных воздействий, способных привести к нарушению конфиденциальности, целостности или доступности всей системы.

Фаззинг – это технология автоматизированного тестирования программного обеспечения с целью выявления потенциальных уязвимостей, которая охватывает большое количество граничных случаев путем порождения некорректных входных данных. В качестве входных данных при этом могут выступать обрабатываемые приложением файлы, информация, передающаяся по сетевым протоколам, функции прикладного интерфейса и т.д. Термин возник в эпоху аналоговых телефонных сетей, когда обнаружилось, что случайный шум (fuzz) на линии может вызвать сбой в управляющем модемом программном обеспечении [1,2, 3].

Фаззер – приложение или фреймворк, дающие возможность проводить автоматизированный фаззинг приложений, кода, файлов и т.п. В настоящее время фаззеров не много. В большинстве своём фаззеры написаны для UNIX, но есть также и для Windows.

Все фаззеры относятся к одной из двух категорий: мутационные, которые изменяют существующие образцы данных и создают условия для тестирования, и порождающие, которые создают условия для тестирования с чистого листа, моделируя необходимый протокол или формат файла.

Метод с использованием заранее подготовленных ситуаций для тестирования применяется в системе PROTOS. Разработка ситуации для тестирования начинается с изучения частного примера, чтобы понять, какие структуры данных поддерживаются и каковы приемлемые значения для каждой из этих структур. Позже создаются пакеты с жестким кодом или файлы, с помощью которых исследуются граничные условия или вносятся ошибки в программу. Затем эти случаи могут быть использованы для проверки того, насколько точно спецификация была применена к тестируемым системам. Создание ситуации для тестирования может потребовать серьезной предварительной подготовки, но имеет то преимущество, что ее можно использовать позднее для того, чтобы унифицировано тестировать различные варианты применения одного и того же протокола или файлового формата.

Недостаток заключается в том, что этот метод имеет неизбежные ограничения. Поскольку случайный компонент при этом отсутствует, то тестирование завершается с завершением списка случаев для тестирования.

Метод на основе случайных данных наименее эффективен, однако может быть использован как самый быстрый способ определения того, не содержит ли объект совершенно неверный код. Данный подход заключается в простом вбрасывании псевдослучайных данных в объект в надежде на лучшее или худшее, все зависит от точки зрения.

Сложность при случайностном фаззинге заключается в том, чтобы понять, что же вызвало падение сервера. Разобраться какая именно часть теста вызвала сбой тяжелая задача. Для этого, возможно, понадобится сниффер. Также придется потратить много времени на работу с отладчиком и дизассемблированным кодом. Отладка после того, как стек будет поврежден таким образом. Может оказаться болезненным делом, в особенности потому что список вызовов окажется поврежденным.

Часто используется мутационное тестирование вручную. При ручном тестировании автоматические фаззеры не применяются. Фактически, исследователь сам является фаззером. Загрузив тестируемое приложение, тестер вручную вводит неподходящие данные в попытке обрушить сервер или вызвать его нежелательное поведение. Его преимущество заключается в том, что аналитик может руководствоваться своим прошлым опытом. Чаще всего такой фаззинг применяется для веб-приложений.

Тестирование методом грубой силы. Фаззер в этом случае, начинается с действующего образца протокола или формата данных и искажает каждые байт, слово, двойное слово или строку в пакете данных или файле. Это один из самых ранних подходов – он почти не требует предварительных исследований, и пользоваться им сравнительно просто. Все, что требуется здесь от фаззера, - это изменение данных и их передача.

Однако этот подход малоэффективен, поскольку в течение многих циклов будут получаться данные, которые сперва нельзя будет интерпретировать. Тем не менее этот процесс можно полностью автоматизировать. Охват кода при подходе грубой силы зависит от того, сколько пакетов или файлов тестируется.

Автоматическое порождающее тестирование – это более продвинутый метод тестирования грубой силой. Для его реализации требуется предварительное исследование: вначале нужно понять и оценить спецификации протокола или определение файла. Однако тестер, вместо того, чтобы создать образец с жестко заданным кодом, создает грамматику, которая описывает работу спецификации протокола. Таким образом, он определяет те порции пакета или файла, которые должны остаться неизменными. И те, которые служат переменными для фаззинга. Затем фаззер проводит динамический анализ этих шаблонов, создает фаззинговые данные и направляет на объект получившийся пакет или файл. Успех при таком подходе зависит от способности тестера определить при анализе те куски протокола, которые, скорее всего, вызовут ошибки в изучаемом объекте. Недостатком этого подхода является то, что необходимо затратить время на создание грамматики или определения.

Рассмотрим несколько примеров прикладных программ, реализующих различные виды фаззинга [4,5].

MiniFuzz – это небольшая утилита-исказитель (fuzzer), созданная командой SDL в Microsoft с целью демонстрации концепций файлового фаззинга и помощи разработчикам в выявлении уязвимостей безопасности и возможных отказов обслуживания (DoS) программного обеспечения, прежде чем приложения будут переданы заказчикам.

MiniFuzz может функционировать как самостоятельное приложение или интегрированный инструмент VisualStudio.

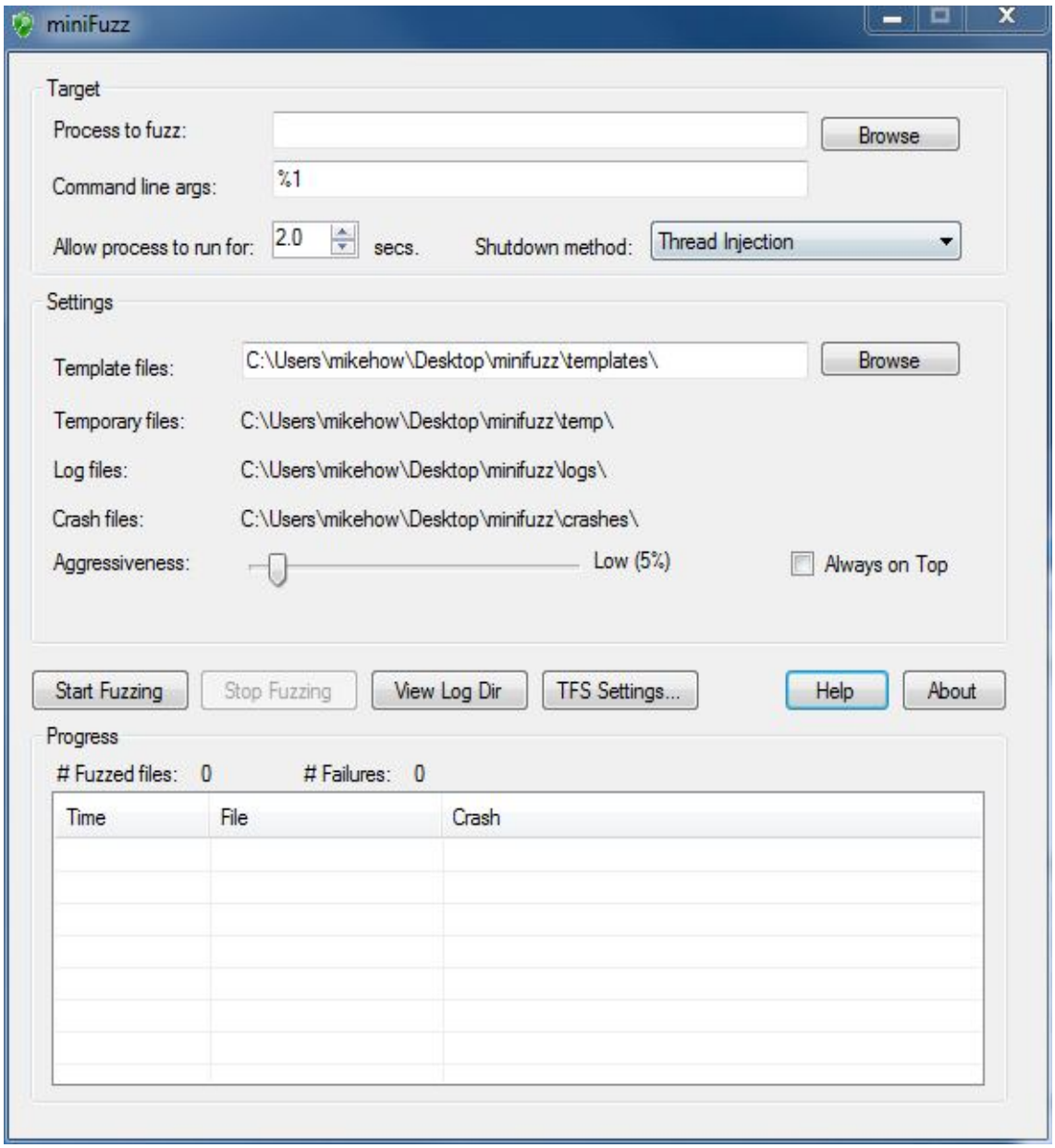

Рисунок 1 – Интерфейс фаззера miniFuzz

Если MiniFuzz — это очень простой (хотя и эффективный) dump-фаззер, то проект Peach (в переводе – персик), разработанный Майком Эддингтоном — это уже мощное решение для smart-фаззинга, поддерживающее как режим мутации, так и генерации [7].

В отличие от Minifuzz, Peach может фаззить не только файлы, но и сетевые сервисы, RPC, COM/DCOM, SQL-хранимые процедуры и многое другое. Однако такая универсальность приводит и к некоторым трудностям в использовании.

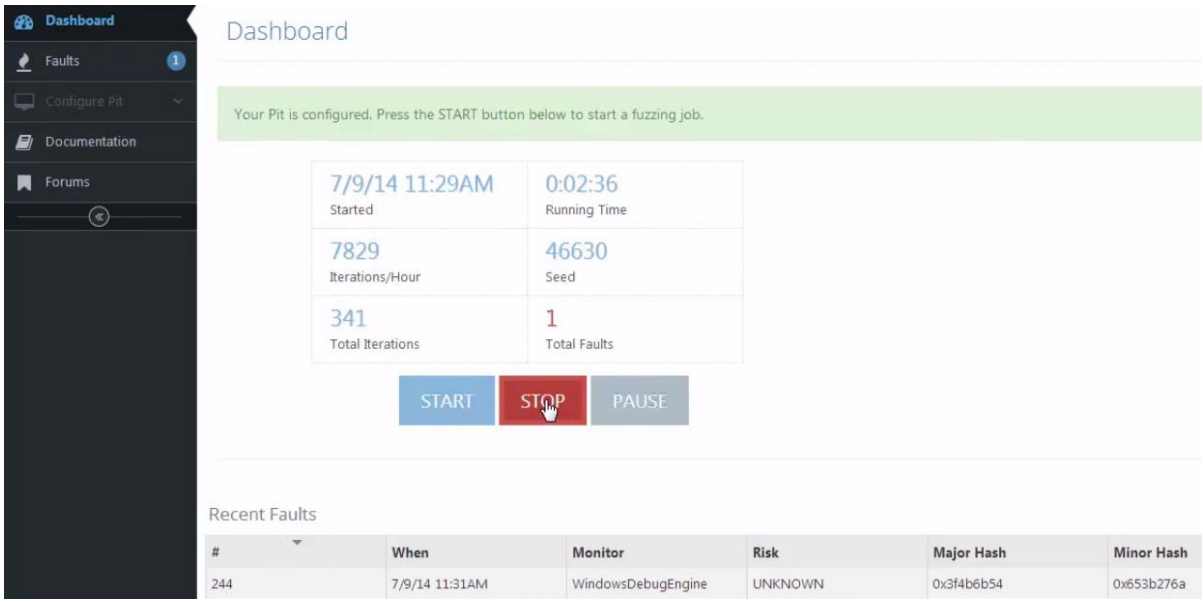

Рисунок 2 – Интерфейс фаззера Peach

Любой фаззинг в Peach'е начинается с создания PeachPit. В этом XML-файле определяется цель фаззинга, описывается структура данных, с которой будет работать фаззер, а также определяются правила манипуляции с ними. Для удобства автор фреймворка предлагает библиотеку для VisualStudio, серьезно упрощающую работу с PeachPit'ами, в том числе с помощью автодополнения кода.

FuzzDB — это проект, объединяющий в себе большое количество фаззинг-баз, упорядоченных по своему назначению. В FuzzDB входят:

- распространенные пути файлов и директорий, представляющих ценность для атакующего, например пути логов и конфигурационных файлов;
- шаблоны атак — собственно те строки, которые отправляются приложению, вследствие чего возникают ошибки и исключения;
- шаблоны ответов — строки, с помощью которых можно идентифицировать наличие уязвимости;
- другие полезности, например коллекция web-шеллов под большинство платформ и словари для брутфорса;
- документация.

Основой для FuzzDB являются базы таких известных фаззеров как jBroFuzz (проект OWASP), wapiti, SPIKE. Проект поддерживается лишь одним человеком, но, тем не менее, не стоит на месте.

По сути, FuzzDB — лишь набор текстовых файлов с шаблонами, отсортированных по платформам, типам атак, языкам. Использовать FuzzDB можно где угодно, например в самописномweb-сканнере директорий или в

профессиональном инструменте для проведения пен-тестов. Рассмотрим применение FuzzDB на практике в BurpIntruder, входящий в составBurpSuite.

Независимые исследователи безопасности продолжают расширять горизонты и разрабатывать технологии фаззинга, а производители коммерческих продуктов прикладывают усилия для создания первого удобного в обращении фаззера, который бы хорошо вписывался в разные среды разработки. Среди важнейших требований будет автоматизация поиска и улучшение механизма обнаружения ошибок [9,10]

Технология фаззинга позволяет выявлять определенные типы слабых мест объекта такие как:

- ошибки контроля доступа;
- ошибки в логике устройства;
- направления ввода, требующие идентификации пользователя;
- повреждение памяти;
- многоступенчатые уязвимости.

Подводя итоги, отметим, что тенденция к применению фаззинга на всех этапах жизненного цикла программного обеспечения доказывает уникальность данной методологии для сообщества исследователей безопасности. Прежде всего, такие альтернативные технологии, как бинарный реинжиниринг и углубленный анализ исходного кода, требуют специальных навыков, овладение которыми попросту нереально для разработчиков и контролеров качества. В то же время фаззинг можно автоматизировать, и он в таком виде подойдет обеим категориям специалистов.

Список литературы

1 Michael Sutton, Adam Greene, and Pedram Amini. Fuzzing : brute force vulnerabilty discovery. Upper Saddle River, NJ: Addison-Wesley, 2007. Print.

2 Takanen, Ari, Jared D. Demott, and Charles Miller. Fuzzing for software security testing and quality assurance. Boston: Artech House, 2008. Print.

3 Fuzzing : the Past, the Present and the Future [Электронный ресурс]. – Режим доступа: http://actes.sstic.org/SSTIC09/Fuzzing-the_Past-the_Present_and_the_Future/SSTIC09-article-A-Takanen-Fuzzing-the_Past-the_Present_and_the_Future.pdf

4 Mark Dowd, John McDonald, and Justin Schuh. The art of software security assessment : identifying and preventing software vulnerabilities. Indianapolis, Ind: Addison-Wesley, 2007. Print.

5 Coverity [Электронный ресурс]. – Режим доступа: http://www.idapro.ru/description/. – Загл. с экрана.

6 IDA Pro [Электронный ресурс]. – Режим доступа: http://www.idapro.ru/description/. – Загл. с экрана.

7 Peach Fuzzer [Электронный ресурс]. – Режим доступа: http://peachfuzzer.com/. – Загл. с экрана.

8 Lcamtuf's blog [Электронный ресурс]. – Режим доступа: http://lcamtuf.blogspot.ru/. – Загл. с экрана.

9 Fuzzing [Электронный ресурс]. – Режим доступа: https://www.

owasp.org/index.php/Fuzzing . – Загл. с экрана.
10 Качественные решения в задачах информационной безопасности, (золотая медаль форума) / А. И. Захаров, Э. А. Лидский, У. А. Михалева // Научные труды международной научно-практической конференции «СВЯЗЬ-ПРОМ 2007» в рамках 4-го Евро-Азиатского форума «СВЯЗЬ-ПРОМЭКСПО 2007». – Екатеринбург: ЗАО «Компания Реал-Медиа», 2007. С. 225–229.