УДК 00 – 519.6

Влацкая И.В., к.т.н.
доцент кафедры «компьютерной безопасности и математического обеспечения информационных систем»
Заельская Н.А.
старший преподаватель кафедры «компьютерной безопасности и математического обеспечения информационных систем»
Надточий Н.С.
старший преподаватель кафедры «компьютерной безопасности и математического обеспечения информационных систем»
Оренбургский государственный университет
Россия, Оренбург

ИДЕНТИФИКАЦИЯ УЯЗВИМОСТЕЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДИНАМИЧЕСКИМИ СРЕДСТВАМИ ФАЗЗИНГА

Аннотация: 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.

Ключевые слова: фаззинг, информационные системы, фаззер, тестирование.

Vlatskaya I.V., Ph.D.
Associate Professor of the Department of "Computer Security and Mathematical Support of Information Systems"
Zaelskaya N.A.
Senior lecturer of the Department of "Computer Security and Mathematical Support of Information Systems"

Nadtochiy N.S.
senior lecturer of the Department of "Computer Security and Mathematical Support of Information Systems"
Orenburg State University
Russia, Orenburg

IDENTIFICATION OF SOFTWARE VULNERABILITIES BY DYNAMIC FUZZING TOOLS

Abstract: the tendency to use fuzzing at all stages of the software lifecycle proves the uniqueness of this methodology for the community of security researchers. First of all, alternative technologies such as binary reengineering and in-depth source code analysis require special skills, the mastery of which is simply unrealistic for developers and quality controllers.

Keywords: fuzzing, information systems, fuzzer, testing.

Существуют множество подходов выявления уязвимостей, и у каждого свои достоинства и недостатки. Ни один из них не является единственно правильным, и ни один не способен раскрыть все возможные уязвимости.

Определенных технологий фаззинга не существует. Результат данной технологии зависит исключительно от результатов теста. Фаззинг это процесс предсказания того, какие типы программных ошибок могут обнаружиться в продукте и какие именно введённые значения вызовут ошибки. Фаззинг как методика тестирования в основном относится к области серого и черного ящиков.

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.

Цель для фаззинга (fuzz target) — это функция, которая принимает на вход данные и обрабатывает их с использованием тестируемого API. Иными словами, это то, что нам необходимо фаззить.

Данный этап заключается в тщательном анализе каждой цели для фаззинга из attack surface. Вот что необходимо узнать:

- Аргументы функции, через которые передаются данные для обработки. Нужен сам буфер для данных и его длина, если её возможно определить.

- Тип передаваемых данных. Например, документ html, картинка png, zip-архив. От этого зависит то, как будут генерироваться и мутироваться поступающие на вход данные.

- Список ресурсов (память, объекты, глобальные переменные), которые должны быть инициализированы перед вызовом целевой функции.

- Если проводят фаззинг внутренних функций компонентов, а не API, то понадобится составить список ограничений, которые накладываются на данные кодом, выполненным ранее. Бывает так, что проверка данных происходит в несколько этапов — это нам тоже следует учитывать.

Этот этап является самым кропотливым, ведь целей для фаззинга может быть очень много: сотни или даже тысячи. Времени и сил на анализ может быть потрачено примерно столько же, сколько уходит на реверс приличного бинарного файла.

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.

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. 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.

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.

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.

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.

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.

Использованные источники:

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

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

3. Fuzzing [Электронный ресурс]. – Режим доступа: https://www. owasp.org/index.php/Fuzzing . – Загл. с экрана.