электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

УДК 004.415.532.2 004.052.42

Использование фаззинга для поиска уязвимостей в программном обеспечении

Бучнев Я.А., *студент*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Защита информации»

Вахрушев И.А., *студент*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Защита информации»

Научный руководитель: Астрахов А.В., к. т. н., доцент
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана
zi@bmstu.ru

Фаззинг

Тестирование методом фаззинга, или коротко *фаззинг*, это метод динамического тестирования программ, который позволяет эффективно находить уязвимости в программном обеспечении путем передачи неправильных или неожиданных данных как входных данных в программу. Фаззинг нацелен на обнаружение случаев, в которых программист при тестировании программы упустил крайние или бессмысленные значения. Фаззинг добивается этого путем передачи сгенерированных или измененных входных данных в программу и отслеживая эффекты, вызванные такими входными данными в целевой программе, особенно ее падения.

Существует множество типов фаззинга, основанных на механизмах генерации входных данных. В соответствии с [2] классифицируем техники фаззинга следующим образом:

По входным данным:

Основанный на изменении (*мутационный фаззинг*): Либо детерминированно либо вероятностно измененные части входного файла или аргументов как средство генерации тестовых случаев. Это требует знания правильных входных файлов, особенно тех, что заставляют программу выполнить код. Многие доступные фаззеры используют эту

технику по умолчанию, так как это наиболее понятный и простой способ быстрого создания больших тестов.

Основанный на генерации (*порождающий фаззинг*): Вместо изменения доступного корректного входного файла, входные файлы генерируются автоматически, основываясь на грамматическом шаблоне, который описывает синтаксис файлов. Этот тип создания входных данных требует хорошее моделирование входных данных, что требует досконального понимания спецификации для эффективности.

По знанию контекста:

Фаззинг черного ящика: Фаззер в этом типе фаззинга не знает внутренностей целевой программы. Созданные файлы подаются в программу, и фаззер отслеживает выходные данные (состояние) программы. В основном, этот метод используется по умолчанию в большинстве доступных фаззеров благодаря простоте и производительности. Однако, фаззинг черного ящика страдает от проблем покрытия кода, поскольку не имеет понятия о процессе выполнения программы – таким образом фаззер не понимает, что все время подает схожие порции кода.

Фаззинг белого ящика: Фаззер такого типа имеет преимущество тем, что знает внутренние структуры и проектирование целевой программы. Обычно фаззер отслеживает состояние, и каждый запуск имеет в качестве цели проверку правильности определенной части кода итеративно. Цель может быть в максимальном покрытие кода или тестирование крайних значений определенных функций. Фаззинг белого ящика обычно сопровождается другими техниками, такими как символический анализ или таинт-анализ.

Выделим фазы фаззинга в соответствии с [1]:

1. Определение цели. Инструмент и технологию фаззинга необходимо выбирать с учетом особенностей целевой программы. Помимо программы, также можно выбрать для фаззинга библиотеку, используемую этой программой.
2. Определение вводимых значений. Большинство ошибок вызываются приложениями, в которых пользователям необходимо вводить данные и работать с ними без предварительной обработки.
3. Порождение некорректных данных. Выбирается способ создания некорректных данных.
4. Использование некорректных данных. Самая суть процесса фаззинга.
5. Мониторинг исключений. Процесс поиска ошибок.
6. Определение работоспособности. Анализ выявленных ошибок.

Ограничения и исключения фаззинга:

1. Ошибки контроля доступа
2. Ошибка в логике устройства
3. Тайные ходы
4. Повреждение памяти
5. Многоступенчатые уязвимости

Постановка задачи

Разработать мутационный фаззер для тестирование файлов формата docx, содержащих элементы ActiveX Windows Media Player. Фаззер должен изменять заданные поля содержимого файла, отвечающие за настройки ActiveX элемента, а также отслеживать результат работы при открытии измененных файлов.

Создание прикладной программы

При написании мутационного фаззера использовался интерпретируемый язык программирования Python 2.7. Для тестирования была выбрана программа Microsoft Word 2010, входящая в пакет Microsoft Office 2010. Формат docx, обрабатываемый текстовым процессором Word 2010 представляет собой zip-архив, содержащий текст в виде XML, графику и другие данные. Например, ActiveX1.xml задает параметры элемента ActiveX документа.

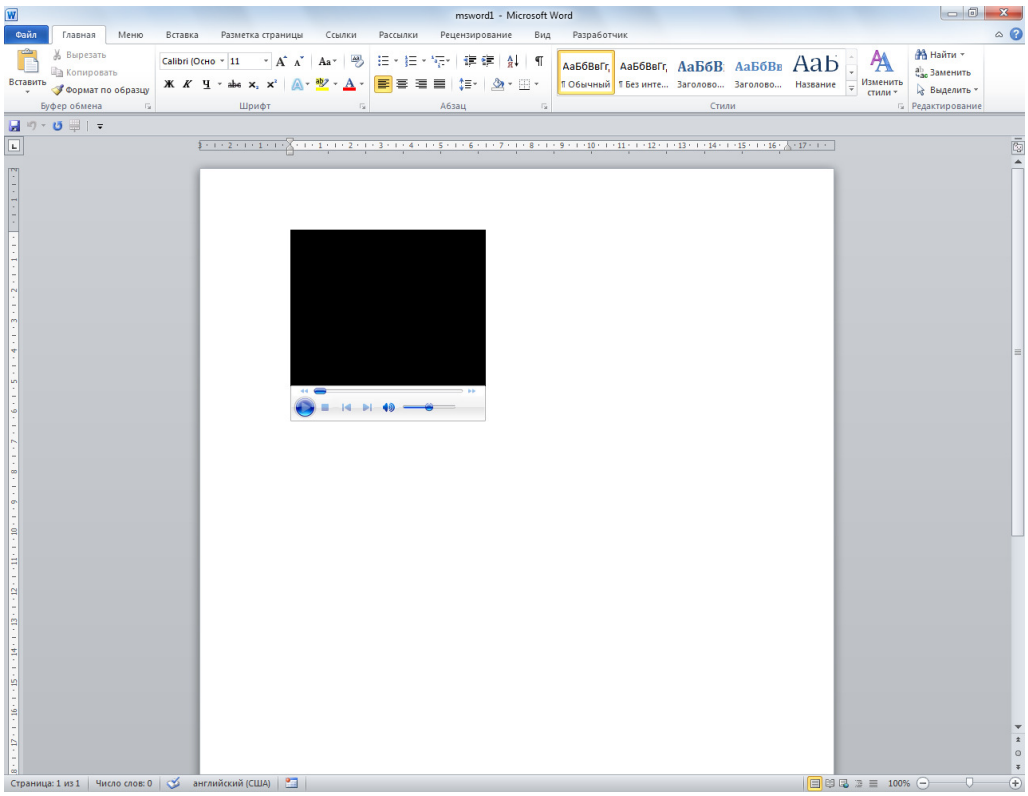

Рис. 1. Базовый документ

Выбирается файл формата *.docx с элементом ActiveX Windows Media Player (Рис. 1). После чего фаззер распаковывает исходный документ. В файле ActiveX1.xml случайным образом производится изменение как отдельных параметров элемента, так и их комбинаций. Затем фаззер собирает документ с измененным ActiveX1.xml и подает его в текстовый процессор Microsoft Word для выяснения результатов обработки мутированного файла.

```
<?xml version="1.0" encoding="UTF-8"?>
- <ax:ocx xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships" xmlns:ax="http://schemas.microsoft.com/office/2006/activeX"
ax:persistence="persistPropertyBag" ax:classid="{DFEAF541-F3E1-4C24-ACAC-99C30715084A}">
    <ax:ocxPr ax:value="5080" ax:name="_cx"/>
    <ax:ocxPr ax:value="5080" ax:name="_cy"/>
    <ax:ocxPr ax:value="" ax:name="Background"/>
    <ax:ocxPr ax:value="0" ax:name="EnableFramerateCounter"/>
    <ax:ocxPr ax:value="0" ax:name="EnableCacheVisualization"/>
    <ax:ocxPr ax:value="0" ax:name="EnableRedrawRegions"/>
    <ax:ocxPr ax:value="" ax:name="Source"/>
    <ax:ocxPr ax:value="60" ax:name="MaxFramerate"/>
    <ax:ocxPr ax:value="0" ax:name="Windowless"/>
    <ax:ocxPr ax:value="" ax:name="OnError"/>
    <ax:ocxPr ax:value="" ax:name="OnFullScreenChanged"/>
    <ax:ocxPr ax:value="" ax:name="OnResize"/>
    <ax:ocxPr ax:value="" ax:name="OnZoom"/>
    <ax:ocxPr ax:value="" ax:name="OnLoad"/>
    <ax:ocxPr ax:value="" ax:name="InitParams"/>
    <ax:ocxPr ax:value="" ax:name="Culture"/>
    <ax:ocxPr ax:value="" ax:name="UICulture"/>
    <ax:ocxPr ax:value="1" ax:name="EnableHtmlAccess"/>
    <ax:ocxPr ax:value="1" ax:name="AllowHtmlPopupWindow"/>
    <ax:ocxPr ax:value="" ax:name="SplashScreenSource"/>
    <ax:ocxPr ax:value="" ax:name="OnSourceDownloadComplete"/>
    <ax:ocxPr ax:value="" ax:name="OnSourceDownloadProgressChanged"/>
    <ax:ocxPr
        ax:value="49352071084971316767366252179561593376355573820520117273468891835776364178721294541879762061459628370161356697130212553
        ax:name="MinRuntimeVersion"/>
    <ax:ocxPr ax:value="" ax:name="AutoUpgrade"/>
    <ax:ocxPr ax:value="0" ax:name="EnableGPUAcceleration"/>
    <ax:ocxPr ax:value="0" ax:name="EnableAutoZoom"/>
    <ax:ocxPr ax:value="" ax:name="EnableNavigation"/>
    <ax:ocxPr ax:value="" ax:name="ProductId"/>
    <ax:ocxPr ax:value="" ax:name="InstanceId"/>
</ax:ocx>
```

Рис. 2. Мутированный ActiveX1.xml

После проведения серии тестов было установлено, что при завершении сеанса работы с мутированным документом MS Word изменял содержимое ActiveX1.xml на начальное во всех случаях, кроме изменения поля *MinRuntimeVersion* (Рис. 2). Это поле задает минимальную необходимую версию проигрывателя Silverlight для обработки содержимого элемента. Текстовый процессор сравнивал установленную в системе версию проигрывателя с необходимой, и в случае, если номер необходимой версии был больше, предлагал скачать обновление с официального сайта Microsoft Silverlight (Рис. 3).

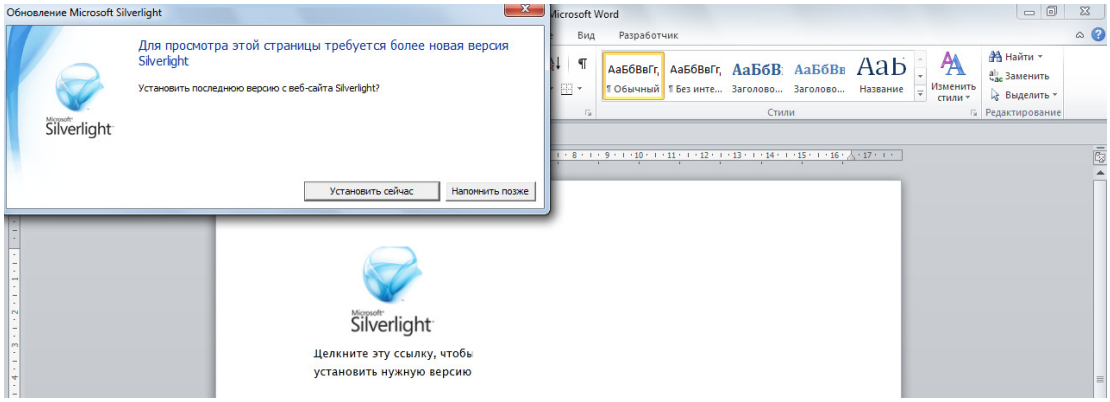

Рис. 3. Сообщение о новой версии

Результаты тестов позволили разработать сценарий вероятной атаки. На атакуемый компьютер присылается электронное письмо, во вложении которого находится файл формата docx, содержащий в себе модифицированный элемент ActiveX Windows Media Player и макрос (в программах пакета MS Office макросы составляются на языке Visual Basic Script, позволяющем осуществлять операции с файловой системой). При открытии документа макрос изменяет файл hosts системы (текстовый файл, содержащий базу данных доменных имен и используемый при их трансляции в сетевые адреса узлов. Запрос к этому файлу имеет приоритет перед обращением к DNS-серверам). Затем, пользователя уведомляют о наличии обновления для Microsoft Silverlight. Пользователь переходит на сайт злоумышленника с тем, чтобы загрузить более опасное вредоносное программное обеспечение или установить средства удаленного управления — в зависимости от содержимого сайта.

Заключение

В ходе исследования был разработан фаззер MS Word. Процесс фаззинга занимает очень много времени, поэтому в качестве дальнейшей работы планируется выбор новых диапазонов значений элементов, изменяемых фаззером. Также планируется применение фаззера для тестирования других элементов ActiveX, включенных в MS Word и остальные компоненты MS Office. Возможен выбор новых способов отслеживания сбоев в работе тестируемой программы.

Список литературы

1. Саттон Майкл Дж.Д., Грин А., Амини П.: Fuzzing. Исследование уязвимостей методом грубой силы. М.: Символ-Плюс, 2009. 560 с.
2. Pak Brian S., Hybrid Fuzz Testing: Discovering Software Bugs via Fuzzing and Symbolic Execution. Режим доступа: http://reports-archive.adm.cs.cmu.edu/anon/2012/CMU-CS-12-116.pdf (дата обращения 20.01.2014).
3. Takanen Ari, DeMott Jared, Miller Charlie: Fuzzing for software security testing and quality assurance. Norwood: Artech House, 2008. 287 pp.
4. Balakrishnan Gogul, Reps Thomas W.: WYSINWYX: What you see is not what you eXecute. Режим доступа: http://research.cs.wisc.edu/wpis/papers/wysinwyx05.pdf (дата обращения 20.01.2014).