1. Министерство науки и высшего образования Российской Федерации
2. Санкт-Петербургский политехнический университет Петра Великого
3. —
4. Институт прикладной математики и механики
5. Высшая школа кибербезопасности и защиты информации

**ЛАБОРАТОРНАЯ РАБОТА №2**

**«Learn&Fuzz машинное обучение для фаззинга»**

1. по дисциплине «Анализ безопасности протоколов»
2. Выполнил
3. студент гр. 3651003/50801 Корольков А.А

<*подпись*>

1. Проверил
2. Жуковский Е.В.

<*подпись*>

1. Санкт-Петербург
2. 2019

**Содержание**

[1. Введение 3](#_Toc38131738)

[2. Структура PDF документов 4](#_Toc38131739)

[3. Изучение статистики содержания объекта 7](#_Toc38131740)

[4. Seq2seq нейросетевой модели 8](#_Toc38131741)

[5. Создание новых объектов PDF 9](#_Toc38131742)

[6. SampleFuzz: выборка с фаззингом 11](#_Toc38131743)

[7. Обучение модели 12](#_Toc38131744)

[8. Экспериментальная оценка метода 12](#_Toc38131745)

[9. Обучающие данные 13](#_Toc38131746)

[10. Уровень покрытия кода 14](#_Toc38131747)

[11. Изучение объектов PDF 16](#_Toc38131748)

[12. Покрытие изученными объектами PDF 17](#_Toc38131749)

[13. Сравнение результатов тестирования покрытия кода 18](#_Toc38131750)

[14. Сочетание обучения модели и фаззинга 19](#_Toc38131751)

[15. Зависимость между охватом и успешностью прохождения 20](#_Toc38131752)

[16. Ошибки 21](#_Toc38131753)

[17. Заключение 22](#_Toc38131754)

1. Введение

Fuzzing — это процесс поиска уязвимостей в коде для анализа входных данных путем многократного тестирования синтаксического анализатора с измененными входных данных. Существует три основных метода, используемых сегодня: черный ящик (blackbox) (случайный метод), белый ящик(whitebox) с ограничениями фаззинга, и на основе грамматик и ограничений, который можно рассматривать как вариант на основе тестирования модели. Blackbox и whitebox фаззинг полностью автоматизированы и уже доказали свою эффективность в поиске уязвимостей в парсерах двоичных файлов. В отличие от них, фаззинг основанный на грамматике не является полностью автоматическим. Такой фаззинг требует ввода грамматики, определяющей входной формат тестируемого приложения. Эта грамматика обычно пишется от руки, и этот процесс трудоемкий и подверженный ошибкам. Тем не менее, основанный на грамматике фаззинг является наиболее эффективным методом фаззинга, известным сегодня для приложений со сложными структурированными входными форматами, такими как веб-браузеры, которые должны принимать в качестве входных ненадежные данные веб-страницы, включая сложные HTML-документы и JavaScript-код.

В рассматриваемой работе рассмотрели проблему автоматического генерирования входных грамматик для фаззинга с использованием методов машинного обучения и входные образцы. В уже существующих решениях и исследованиях использовались варианты традиционных автоматов и алгоритмы обучения контекстно-свободной грамматике. В отличие от предыдущих работ, в данной работе представлена первая попытка использования нейросетевых методов статистического обучения для решения этой задачи. В частности, используются рекуррентные нейронные сети для обучения статистической входной модели, которая также является генеративной: ее можно использовать для генерации новых входных данных на основе распределения вероятностей изученной модели. В решении используется бесконтрольное обучение, и этот подход полностью автоматизирован и не требует какой-либо настройки под конкретный формат.

Рассмотренное исследование представляет углубленное исследование для сложного формата входных данных: PDF. В работе рассматривается большой, сложный и критически важный для безопасности синтаксический анализатор для этого формата: PDF парсер, встроенный в браузер Edge от Microsoft. В ходе серии детальных экспериментов обсуждается проблема learn&fuzz: как научиться, а затем генерировать разнообразные хорошо сформированные входные данные, чтобы максимизировать охват кода синтаксического анализатора, в то же время вводя достаточно плохо сформированных входных данных, чтобы использовать неожиданные ветки кода и код обработки ошибок.

В работе также представлен новый алгоритм learn&fuzz, который использует изученное распределение вероятностей входных данных, чтобы разумно оценить, где именно стоит применять фаззинг на входных данных. Мы покажем, что этот новый алгоритм может превзойти другие алгоритмы обучения и случайного фаззинга, рассмотренные в этой работе.

1. Структура PDF документов

Полная спецификация формата PDF составляет более 1 300 страниц . Большая часть этой спецификации посвящена описанию объектов данных и их взаимосвязей между частями PDF-документа.

PDF-файлы кодируются в текстовом формате, который может содержать двоичные информационные потоки (например, изображения, зашифрованные данные). PDF-документа — это последовательность, по меньшей мере, одного PDF тела. Тело PDF состоит из трех разделов: объекты, таблица перекрестных ссылок и трейлер.

**Объекты**. Данные и метаданные в PDF-документе организованы в базовые единицы, называемые объектами. Все объекты имеют одинаковый формат, как показано на Рис. 1 (А), и имеют общую внешнюю структуру. Первая строка объекта — это его идентификатор, для косвенных ссылок-его номер поколения, который увеличивается, если объект переопределен более новой версией, «obj» который указывает на начало объекта, а индикатор «endendobj» закрывает объект.

Объект на Рис. 1 (А) содержит структуру словаря, которая разделена символами «<<» и «>>» и содержит ключи, начинающиеся с «/», за которыми следуют их значения.[ 3 0 R ] - это межобъектная ссылка на объект в том же документе с идентификатором 3 и номером генерации 0. Поскольку документ может быть очень большим, доступ к объекту ссылки осуществляется с помощью произвольного доступа через таблицу перекрестных ссылок.

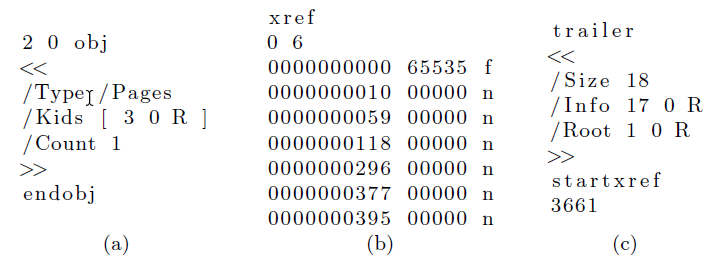


Рисунок 1. Пример структуры PDF файлов

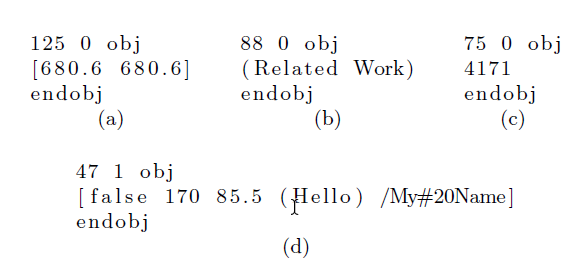


Рисунок 2. Допольнительный пример структуры PDF файлов

Другие примеры объектов показаны на Рис. 2. Объект на Рис. 2 (а)имеет содержимое [680,6 680,6], которое является объектом массива. Его назначение-удерживать координаты, на которые ссылается другой объект. Рис. 2 (б) представляет собой строковый литерал, содержащий текст закладки для раздела PDF-документа. Рисунок 2 (c) представляет собой числовой объект. На рис. 2 (d)представлен объект, содержащий многотипный массив. Все это примеры типов объектов, которые используются как сами по себе, так и в качестве базовых блоков, из которых состоят другие объекты (например, объект справочника на Рис.1(А) содержит массив). Правила определения и составления объектов составляют большую часть спецификации формата PDF.

**Таблица перекрестных ссылок.** Таблицы перекрестных ссылок тела PDF содержат адрес в байтах объектов, на которые ссылается документ. На рис. 1 (б) показана таблица перекрестных ссылок с подразделом, содержащим адреса пяти объектов с идентификаторами 1-5 и заглушкой для идентификатора 0, который никогда не ссылается на объект. Объект, на который указывают, определяется строкой таблицы (подраздел будет включать 6 объектов, начиная с идентификатора 0), где n - индикатор используемого объекта, для него первый столбец это адрес объекта в файле,а второй столбец относится к идентификатору родительского объекта, или в случае объекта 0 к объекту 65535( это последний доступный идентификатор объекта).

**Трейлер**. Трейлер PDF-файла содержит словарь (опять же содержащийся в “<<” и “>>”) информации о теле и startxref, который является адресом таблицы перекрестных ссылок. Это позволяет анализировать тело с конца, читая startxref, затем возвращаясь к таблице перекрестных ссылок и анализируя ее, и только анализируя объекты по мере их необходимости.

**Обновление документа**. PDF-документы можно обновлять постепенно. Это означает, что если программа хочет обновить данные в объекте 12, он начнет новое тело PDF, в нем запишет новый объект с идентификатором 12 и номером поколения, большим, чем тот, который появился раньше. Затем он создаст новую таблицу перекрестных ссылок, указывающую на новый объект, и добавит это тело к предыдущему документу. Старый объект будет удален, создав новую таблицу перекрестных ссылок и отметив ее как свободную. Исследователи используют этот метод для добавления новых объектов в PDF-файл.

В работе используется автоматическое изучение структуры (правил) для определения таблиц перекрестных ссылок и трейлеров, которые включают ограничения на списки, адреса, указатели и счетчики, выглядит слишком сложным и менее перспективным для обучения с помощью нейронных сетей. Мы также не рассматриваем двоичные объекты данных, которые кодируются в двоичных субформатах (например, изображение) и для которых уже эффективно полностью автоматическое размытие blackbox и whitebox.

1. Изучение статистики содержания объекта

Основная идея подхода заключается в том, чтобы обучить генеративную языковую модель на множестве символов объектов PDF, заданных большим набором объектов. Используется модель sequence-to-sequence (seq2seq) , которая дает хорошие результаты для многих различных задач обучения, таких как машинный перевод и распознавание речи. Модель seq2seq позволяет изучать контексты произвольной длины для предсказания следующей последовательности символов по сравнению с традиционными подходами, основанными на n-граммах, которые ограничены контекстами конечной длины. Учитывая разновидность объектов PDF, модель seq2seq можно обучить неконтролируемым образом, чтобы изучить генеративную модель для создания новых объектов PDF с использованием набора входных и выходных последовательностей. Входные последовательности соответствуют последовательностям символов в объектах PDF, а соответствующие выходные последовательности получаются путем сдвига входных последовательностей на одну позицию. Изученная модель затем может быть использована для генерации новых последовательностей (объектов PDF) путем выборки распределения, заданного начальным префиксом (например, “obj").

1. Seq2seq нейросетевой модели

Рекуррентная нейронная сеть (RNN) - это нейронная сеть, которая работает на входной последовательности переменной длины (x1, x2,···, xT ) и состоит из скрытого состояния h и выходного y. RNN обрабатывает входную последовательность в серии временных меток (по одной для каждого элемента в последовательности). Для заданного времени t, скрытое состояние обозначается ht,а выход уt. Рассчитываются они так:

ht = f(ht−1, xt)

yt = φ(ht)

где f-нелинейная функция активации, такая как сигмоид, тангенс и т. д. и φ-это функция, такая как softmax, которая вычисляет распределение вероятности выхода по заданному словарю на текущем скрытом состояниии. RNN может узнать распределение вероятностей над символьной последовательности (Х1, · · · , ХТ−1) путем обучения, чтобы предсказать следующий символ XT в последовательности, т. е., он может выучить условное распределение р(хt|(Х1, · · · , ХТ−1)).

Используется модель seq2seq, которая состоит из двух рекуррентных нейронных сетей, кодера RNN, который обрабатывает входную последовательность переменного размера в фиксированное размерное представление, и декодера RNN, который принимает фиксированное размерное входное представление последовательности и генерирует переменную размерную выходную последовательность. Сеть декодера генерирует выходные последовательности, используя предсказанный выходной символ, сгенерированный на временном шаге t, в качестве входного символа для временного шага t + 1. Иллюстрация архитектуры seq2seq показана на рис. 3. Эта архитектура позволяет нам изучать условное распределение по последовательности следующих выходов, то есть p((y1, · · · , yT1 )|(x1, · · · , xT2 )).

Seq2seq модель обучается с использованием совокупности объектов в формате PDF рассматривая каждый из них как последовательность символов. Во время обучения мы сначала объединяем все объектные файлы si в один файл, в результате чего получается большая последовательность символов S = s1 + · · · + sn. Затем мы разбиваем последовательность на несколько обучающих последовательностей фиксированного размера d, так что i-й обучающий экземпляр ti = S[i ∗ d : (i + 1) ∗ d], где S[k : l] обозначает подпоследовательность S между индексами k и l. Выходная последовательность для каждой обучающей последовательности-это входная последовательность, сдвинутая на 1 позицию, т. е. Ot = S[i ∗ d+1 : (i + 1) ∗ d+ 1]. Модель seq2seq затем обучается от начала до конца, чтобы изучить генеративную модель по набору всех обучающих экземпляров.

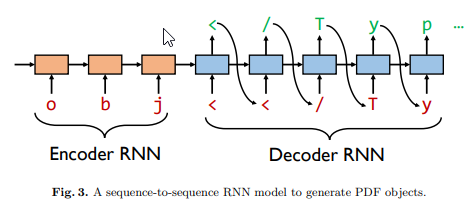


Рисунок 3. Модель для генерации PDF объектов.

1. Создание новых объектов PDF

Существует много различных стратегий генерации объектов в зависимости от стратегии выборки, используемой для изученния распределения. Объект всегда начинается с префикса последовательности "obj“ (обозначающего начало экземпляра объекта), а затем запрашиваем модель для генерации последовательности выходных символов, пока она не выдаст” endobj", соответствующий концу экземпляра объекта. Теперь мы опишем три различных стратегии выборки, которые мы используем для генерации новых экземпляров объектов.

**NoSample**: в этой стратегии генерации мы используем изученное распределение, чтобы точно предсказать лучший символ, заданный префиксом. Эта стратегия приводит к созданию PDF-объектов, которые, скорее всего, будут хорошо сформированы и действительны, но она также ограничивает число объектов, которые могут быть созданы. Учитывая префикс типа "obj", лучшая последовательность следующих символов определяется однозначно, и поэтому эта стратегия приводит к одному и тому же объекту PDF. Это ограничение не позволяет использовать данную стратегию для фаззинга.

**Sample**: использует изученное распределение для выборки следующих символов (вместо выбора самого вероятного символа) в последовательности, заданной префиксной последовательностью. Эта стратегия выборки способна генерировать разнообразный набор новых объектов PDF путем объединения различных паттернов, которые модель изучила из различных наборов объектов на этапе обучения. Из-за использования выборки символов генерируемые объекты PDF не всегда гарантированно корректно сформированы, что полезно с точки зрения фаззинга.

**SampleSpace**: эта стратегия выборки представляет собой комбинацию стратегий Sampleи NoSample. Он производит выборку распределения следующих символов только тогда, когда текущая последовательность префиксов заканчивается пробелом, и использует наиболее вероятный символ из выбранного распределения в середине последовательностей, аналогично стратегии NoSample. Ожидается, что эта стратегия.

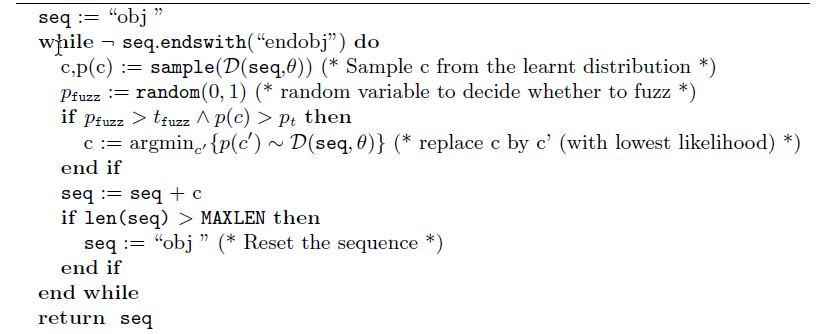


Рисунок 4. Псевдокод алгоритма SampleFuzz

1. SampleFuzz: выборка с фаззингом

Цель изучения генеративной модели объектов PDF в конечном счете состоит в том, чтобы выполнить фаззинг. Совершенная техника обучения всегда будет генерировать хорошо сформированные объекты, которые не будут выполнять никакого кода, содержащего ошибки, в то время как плохая техника обучения приведет к плохо сформированным объектам, которые будут быстро отвергнуты синтаксическим анализатором заранее. Чтобы исследовать этот компромисс, мы представляем новый алгоритм, получивший название SampleFuzz, для выполнения некоторого фаззинга при выборке новых объектов. Мы используем изученную модель для создания новых экземпляров объектов PDF, но в то же время задействуем выполнения кода обработки ошибок.

Алгоритм SampleFuzz показан в рисуноке 4. Он принимает в качестве входных данных изученное распределение D (x, θ), вероятность фаззинга символа *t\_fuzz* и пороговую вероятность *pt,* которая используется для принятия решения об изменении предсказанного символа. При генерации выходной последовательности *seq* алгоритм производит выборку из изученной модели для получения некоторого следующего символа *c* и его вероятности *p(c)* в определенный момент времени *t*. Если вероятность *p(c)* выше заданного пользователем порога *pt*, то есть если модель уверена, что *c*, вероятно, является следующим символом в последовательности, алгоритм выбирает вместо этого другой символ из выборки *c’* на его месте, при этом *с’* имеет минимальную вероятность *p(с’)* в изученном распределении. Эта модификация (фаззинг) имеет место только в том случае, если результат случайный параметр *p\_fuzz* окажется выше, чем входной параметр *t\_fuzz*, что позволяет пользователю дополнительно контролировать вероятность фаззинга символов. Ключевой смысл алгоритма SampleFuzz заключается в том, чтобы вводить неожиданные символы в объекты только в тех местах, где модель очень уверена в распределении, чтобы обмануть анализатор PDF. Алгоритм также гарантирует, что длина объекта ограничена MAXLEN.

1. Обучение модели

Поскольку мы обучаем модель seq2seq в условиях самообучения, у нас нет тестовых меток, чтобы явно определить, насколько хорошо работают обученные модели. Вместо этого мы обучаем несколько моделей, параметризованных числом проходов, называемых эпохами, которые алгоритм обучения выполняет над обучающим набором данных. Таким образом, эпоха определяется как итерация алгоритма обучения для прохождения через полный набор обучающих данных. Мы оцениваем модели seq2seq, обученные для пяти различных чисел эпох: 10, 20, 30, 40 и 50. Одна эпоха занимает около 12 минут для обучения модели seq2seq, а модель с 50 эпохами занимает около 10 часов для обучения. Мы используем модель LSTM (вариант RNN) с 2 скрытыми слоями, где каждый слой состоит из 128 скрытых состояний.

1. Экспериментальная оценка метода

Исследователи использовали три основных стандартных показателя для измерения эффективности фаззинга:

**Покрытие**. Для каждого выполнения теста мы измеряем покрытие инструкций, то есть набор всех уникальных инструкций, выполненных во время этого теста. Каждая инструкция однозначно идентифицируется парой значений DLL-name и DLL-offset. Покрытие для набора тестов — это просто объединение наборов покрытия каждого отдельного теста.

**Процент успешного прохождения**. Для каждого выполнения теста программно проверяется наличие сообщений об ошибках синтаксического анализа в журнале выполнения PDF-анализатора. Если нет сообщений об ошибках, мы называем этот тест пройденным. Пройдённые тесты соответствует PDF-файлам, которые анализатором Edge PDF считаются хорошо сформированными. Эта метрика менее важна для целей фаззинга, но она поможет нам оценить качество обучения.

**Ошибки**. Каждое выполнение теста выполняется под контролем инструмента AppVerifier, бесплатного инструмента runtime мониторинга, который может ловить ошибки повреждения памяти (такие как переполнение буфера) с низкими затратами времени выполнения (обычно несколько процентов).

1. Обучающие данные

Исследователи извлекли около 63 000 не бинарных PDF-объектов из разнообразного набора из 534 PDF-файлов. Эти 534 файла были предоставлены нам командой Windows fuzzing и использовались для внутреннего расширенного fuzzing тестирования анализатора Edge PDF. Этот набор из 534 файлов сам по себе был результатом минимизации ключевого (seed) содержимого, то есть процесса вычисления подмножества большего набора входных файлов, который обеспечивает тот же охват инструкций, что и больший набор. Минимизация является стандартным первым шагом, применяемым перед фаззингом файла. Больший набор PDF-файлов был получен из различных источников, таких как предыдущие PDF-файлы, используемые для фаззинга, а также другие PDF-файлы, собранные из сети.

Эти 63 000 не бинарных объектов являются обучающим набором для RNN, который использовался в этой работе. Бинарные объекты, встроенные в PDF-файлы (обычно представляющие изображения в различных графических форматах), в работе не рассматривались.

Мы изучаем, генерируем и проводим фаззинг над PDF-объектами, но анализатор Edge PDF обрабатывает полные PDF-файлы, а не отдельные объекты. Поэтому мы написали простую программу для корректного добавления нового PDF-объекта к существующему (хорошо сформированному) PDF-файлу, который мы называем хостом. В частности, эта программа сначала идентифицирует последний трейлер в хост файле. В нем содержатся сведения о файле, такие как адреса объектов и таблица перекрестных ссылок, а также идентификатор последнего используемого объекта. Затем в файл добавляется новый раздел тела. В нем новый объект включается с идентификатором объекта, который переопределяет последний объект в файле хоста. Добавляется новая таблица перекрестных ссылок, которая увеличивает показатель поколений переопределенного объекта. Наконец, добавляется новый трейлер объекта.

1. Уровень покрытия кода

Чтобы обеспечить интерпретацию результатов покрытия кода, мы произвольно выбрали 1000 объектов PDF из наших 63 000 обучающих объектов и измерили покрытие кода анализатора Edge PDF, которое будет точкой отсчета(базовым покрытием кода-baseline) для дальнейших измерений

Первый вопрос заключается в том, какой файл PDF хоста мы должны использовать в наших экспериментах: поскольку любой файл PDF будет содержать некоторые объекты в нем.

Чтобы изучить этот вопрос, мы выбрали три самых маленьких PDF-файла в нашем наборе из 534 файлов и использовали их в качестве хостов. Эти три хоста имеют размер 26кб, 33кб и 16кб соответственно.

На рис. 4 показано покрытие инструкций, полученное при запуске анализатора Edge PDF на трех хостах, обозначенных host1, host2 и host3. Он также показывает покрытие, полученное при вычислении объединения этих трех наборов, обозначенных host123. Охват колеблется от 353 327 (host1) до 457 464 (host2) уникальных инструкций, в то время как объединение (host123) составляет 494 652 и больше, но общее для всех случаем – каждый хост охватывает некоторые уникальные инструкции, не охваченные двумя другими. Обратите внимание, что наименьший файл host3 не приводит к наименьшему покрытию.

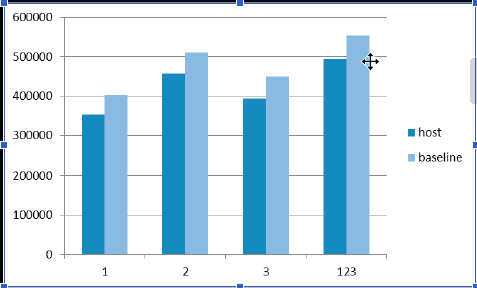


Рисунок 5. Покрытие для хостов PDF и базовых линий.

Затем исследователим рекомбинировали каждый из 1000 базовых объектов с каждым из наших трех хостов, чтобы получить три набора из 1000 новых PDF-файлов, обозначенных baseline1, baseline2 и baseline3 соответственно. На рис. 4 показано покрытие каждого набора, а также их объединение baseline123. Мы наблюдаем следующее.

- Базовое покрытие кода варьируется в зависимости от хоста, но оно всегда больше, чем имел хост до рекомбинации. Наибольшее различие между хостом и базовым покрытием составляет 59 221 инструкция для host123 из 553 873 инструкций для baseline123. Другими словами, 90% всех инструкций покрываются хостом независимо от того, какие новые объекты добавляются.

- Каждый тест обычно охватывает порядка полумиллиона уникальных инструкций.

- Обработка 1000 PDF-файлов занимает около 90 минут.

Так же был измерен процент успешного прохождения для каждого эксперимента, он составляет 100% для всех 3 хостов.

Основной вывод: несмотря на то, что покрытие кода варьируется между хостами, поскольку объекты могут взаимодействовать по-разному с каждым хостом, повторно объединенный PDF-файл всегда успешно обрабатывается программой Edge PDF.

1. Изучение объектов PDF

При обучении RNN важным параметром является количество используемых эпох. На рисунке ниже приведены результаты экспериментов, полученных после тренировки RNN в течение 10, 20, 30, 40 и 50 эпох соответственно. После обучения исследователи использовали каждую изученную модель RNN для создания 1000 уникальных объектов PDF. Исследователи также сравнили сгенерированные объекты с 63 000 объектов, используемых для обучения модели, и не нашли точных соответствий.

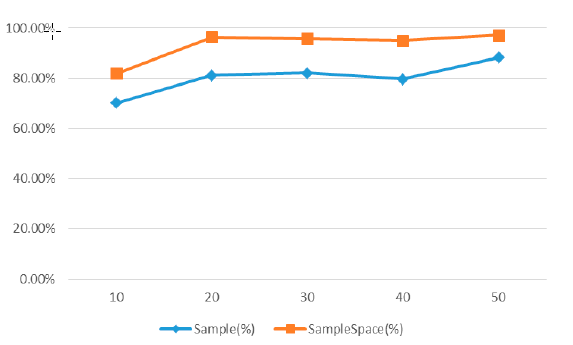


Рисунок 6. Уровень успешности прохождения

Частота успешной обработки PDF объектов для обоих методов выборки при обучении с 10 до 50 эпох представлена на Рис. 6. Наблюдается следующее:

- успешность обработки для SampleSpace лучше, чем для Sample.

- для 10 эпох успешность обработки образца уже превышает 70%. Это означает, что обучение имеет хорошее качество.

– По мере увеличения числа эпох скорость прохождения увеличивается, как и ожидалось, так как обученные модели становятся более точными, но они также требуют больше времени (см. Раздел 3).

- Лучшая успешность обработки -97%, полученная с SampleSpace и 50 эпохами.

Интересно, что успешность обработки практически одинакова независимо от используемого хост-файла PDF: она варьируется не более чем на 0,1% между хостами.

Основной вынос успешность обработки колеблется между 70% и 97% и показывает, что обучение модели прошло успешно.

1. Покрытие изученными объектами PDF

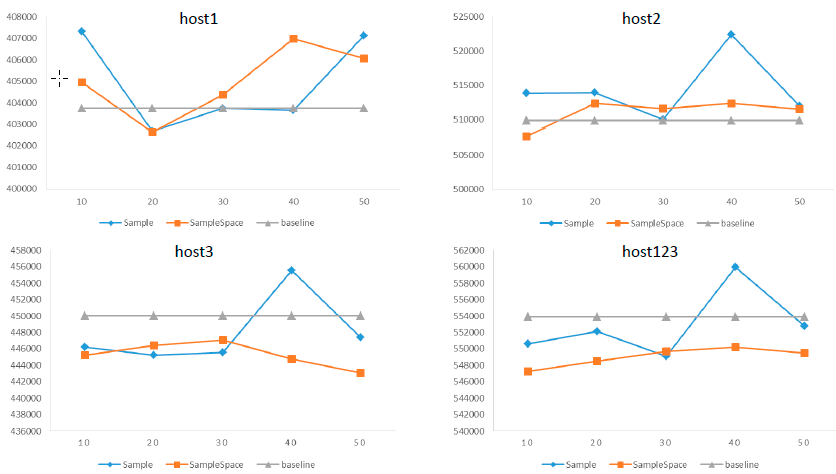


Рисунок 7. Сравнение покрытия кода.

На рис. 7 показано покрытие инструкций парсера, полученное с помощью Sample и SampleSpace от 10 до 50 эпох и использующее host1 (вверху слева), host2 (вверху справа), host3 (внизу слева) и общее покрытие для всех хостов host123 (внизу справа). Мы наблюдаем следующее:

- В отличие от успешности обработки, выбор хоста значительно влияет на покрытие, как уже указывалось ранее. Кроме того, формы каждой линии варьируются в зависимости от хостов.

- Для host1 и host2 покрытие для выборки и SampleSpace выше базового покрытия для большинства результатов epoch, в то время как они в основном ниже базового покрытия для host3 и host123.

- Наилучшее покрытие кода для общего случая получено методом Sample с сетью 40 эпох

- Охват baseline123 в целом является вторым лучшим после метода Sample с сетью 40 эпох

- Лучшее покрытие, полученное с *SampleSpace*, также с 40-эпохами.

Основной вывод: лучшее покрытие кода получается с образцом созданным после 40 эпох методом *Sample*.

1. Сравнение результатов тестирования покрытия кода

До сих пор мы просто подсчитывали количество уникальных инструкций, которые были охвачены. Теперь мы углубляемся в общие данные покрытия host123 на рис. 7 и вычисляем перекрытие между общими наборами покрытия, полученными с помощью Sample-40e и SampleSpace-40e, а также покрытия baseline123 и покрытия host123. Результаты представлены на рис. 8. Наблюдаем следующее:

– Все наборы являются включают себя набор функций host123, как и ожидалось (см. строку host123), за исключением нескольких сотен инструкций каждая.

- Sample-40e-включает в себя почти все инструкции всех других наборов, за исключением 1680 инструкций по сравнению с SampleSpace-40e и нескольких сотен инструкций по сравнению с baseline123 и host123 (см. столбец Sample-40e).

- Sample-40e и SampleSpace-40e имеют гораздо больше общих инструкций чем различных (10 799 и 1680), причем Sample-40e имеет лучшее покрытие, чем SampleSpace-40e.

-SampleSpace-40e несравним с baseline123: в нем есть еще 3393 инструкции, но также 6 514 отсутствующих инструкций.

Вывод: Sample-40e покрывает больше всего функций кода парсера PDF.

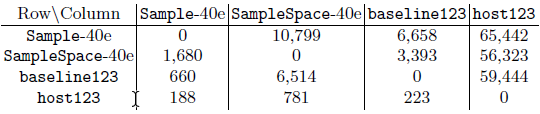


Рисунок 8. Перекрёстное сравнение покрытия кода

1. Сочетание обучения модели и фаззинга

Рассмотрим широко используемый простой алгоритм случайного фаззинга методом blackbox, обозначаемый Random, который случайным образом выбирает позицию в файле, а затем заменяет значение байта случайным значением между 0 и 255. Алгоритм использует коэффициент размытости 100: длина файла, деленная на 100, является средним числом байтов, которые фаззированы в этом файле.

Мы используем случайные числа для генерации 10 вариантов каждого PDF-объекта, генерируемого нейронными сетями с 40 эпохами методами Sample-40e, SampleSpace-40e и baseline (базовые pdf файлы). Полученные фаззированные объекты повторно объединяются с нашими тремя хост-файлами, чтобы получить три набора из 30 000 новых PDF-файлов, обозначенных соответственно Sample+Random, SampleSpace+Random и baseline+Random.

Для целей сравнения мы также включаем результаты запуска Sample-40e для генерации 10000 объектов, обозначенных Sample-10K.

Наконец, мы рассмотрим наш новый алгоритм SampleFuzz, описанный в разделе 3, который решает, где следует фаззировать значения на основе изученного распределения. Мы применили этот алгоритм с изученным распределением модели RNN 40-эпох, tfuzz = 0.9 и порогом pt = 0.9.

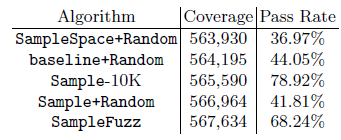


Рисунок 9. Сравнение результатов фаззинга

На рис. 9 показаны общее покрытие кода и скорость прохождения для каждого набора. Обработка каждого набора из 30 000 PDF-файлов занимает около 45 часов. Строки сортируются путем увеличения покрытия. Мы наблюдаем следующее:

– После применения Random к объектам, генерируемым с помощью Sample, SampleSpace и baseline, покрытие кода увеличивается, а успешность прохождения снижается: она постоянно ниже 50%.

- После анализа пересечения между наборами покрытия (данные здесь не показаны), все фаззированные наборы являются почти супернаборами (включают себя все остальные) своих исходных не размытых наборов.

- Покрытие кода для Sample-10К также увеличивается на 6173 инструкции по сравнению с Sample, в то время как успешность прохождения остается около 80% (как и ожидалось).

- Возможно, удивительно, но лучший охват кода получается с SampleFuzz.Частота успешного прохождения для него составляет 68,24%.

- Разница в абсолютном охвате между SampleFuzz и следующим лучшим Sample + Random составляет всего 670 инструкций. Кроме того, после анализа перекрытия набора SampleFuzz охватывает на 2622 инструкции больше, чем Sample+Random, но также пропускает 1,952 инструкции, охватываемые Sample+Random. Таким образом, ни один из этих двух вариантов покрытия полностью не дублирует эффекты другого.

Основной вывод: все алгоритмы, основанные на обучении, рассматриваемые здесь, конкурентоспособны по сравнению с baseline+Random, и три из них превосходят это базовое покрытие.

1. Зависимость между охватом и успешностью прохождения

Главным выводом из всех тестов является зависимость, которую мы наблюдаем между покрытием кода и успешностью прохождения парсинга.

Эту зависимость можно наблюдать на рис. 9. Но это также видно в более ранних результатах: если мы соотнесем результаты покрытия рисунка 7 с результатами успешность прохождения рисунка 6, мы можем ясно увидеть, что SampleSpace имеет лучший % прохождения, чем Sample, но Sample имеет лучшее покрытие, чем SampleSpace (см. host123 в правом нижнем углу рисунка 7).

Эту зависимость можно объяснить следующим образом. Чистый алгоритм обучения с почти идеальным % прохождения (например, SampleSpace) генерирует почти только хорошо сформированные объекты и выполняет мало кода обработки ошибок. Напротив, алгоритм обучения с помехами (например, Sample) с более низкой вероятностью проходит парсинг и может не только генерировать много хорошо сформированных объектов, но и генерировать некоторые плохо сформированные объекты, которые выполняют код обработки ошибок.

Применение алгоритма случайного фаззинга к ранее сгенерированным (почти) хорошо сформированным объектам оказывает еще более сильное влияние на снижение % прохождения (см. рис. 8) при одновременном увеличении покрытия, опять же, вероятно, из-за увеличения покрытия кода обработки ошибок.

Новый алгоритм SampleFuzz, похоже, попал в золотую середину между % прохождения и охватом. В экспериментах идеальный баланс для % прохождения, по − видимому, составляет около 65%-70%: эта скорость прохождения достаточно высока, чтобы генерировать разнообразные хорошо сформированные объекты, которые охватывают много кода в синтаксическом анализаторе PDF, но достаточно низка, чтобы также осуществлять обработку ошибок кода во многих частях этого синтаксического анализатора.

Обратите внимание, что охват инструкциями в конечном счете является лучшим показателем эффективности фаззинга, чем % прохождения, который вместо этого является показателем качества обучения.

1. Ошибки

В дополнение к охвату и % прохождения, третий показатель - количество найденных ошибок. В ходе экспериментов, описанных ранее в этом разделе, ошибок обнаружено не было. Обратите внимание, что анализатор Edge PDF был тщательно обработан в течение нескольких месяцев с другими анализаторами (включая SAGE), прежде чем мы выполнили это исследование, и что все ошибки, обнаруженные во время этого предварительного анализа, были исправлены в версии анализатора PDF, которую мы использовали для этого исследования.

Однако во время более длительного эксперимента с Sample+Random, 100 000 объектов и 300 000 PDF-файлов (который занял почти 5 дней), в Анализаторе Edge PDF была обнаружена ошибка переполнения стека: генерируется PDF-файл обычного размера (его размер составляет 33 КБ), но он запускает неожиданную рекурсию в анализаторе, что в конечном итоге приводит к переполнению стека. Позже эта ошибка была подтверждена и исправлена командой разработчиков Microsoft Edge. В ближайшем будущем мы планируем провести и другие более длительные эксперименты.

1. Заключение

Фаззинг на основе грамматики эффективен для приложений со сложными структурированными входными данными при условии, что имеется полная входная грамматика. В данной статье описана первая попытка использования методов нейросетевого статистического обучения для автоматической генерации входных грамматик из выборочных входных данных. Исслендователи представили и оценили алгоритмы, которые используют последние достижения в генерации последовательностей с помощью нейронных сетей, а именно рекуррентные нейронные сети seq2seq, для автоматического обучения генеративной модели объектов PDF. Мы разработали несколько методов выборки для создания новых объектов PDF из изученного распределения. Мы показываем, что обученные модели не только способны генерировать большой набор новых хорошо сформированных объектов, но и приводят к увеличению охвата парсера PDF, используемого в экспериментах, по сравнению с различными формами случайного фаззинга.

Хотя результаты могут отличаться для других приложений и областей применения, общие наблюдения о зависимотси между конфликтующими целями обучения и фаззига останутся в силе: обучение хочет захватить структуру хорошо сформированных входных данных, в то время как фаззинг хочет разрушить эту структуру, чтобы покрыть неожиданные пути кода и найти ошибки. Исследователи считают, что неотъемлемая статистическая природа обучения с помощью нейронных сетей является мощным инструментом для решения этой проблемы обучения и фаззинга.