Метод фаззинга протоколов с использованием модифицированного клиента

Авторы:
Акользин Виталий Владимирович
Курмангалеев Шамиль Фаимович

{vva1994, kursh}@ispras.ru

23.06.2023

ИСП РАН

Дефекты в современном ПО

Тысячи ошибок выявляются **ежегодно**, в том числе в сетевых приложениях.

Факторы, усугубляющие наличие дефектов в сетевом ПО:

- Удалённое использование дефектов
- Миллионы устройств с одинаковой версией ПО
- Задержки на обновление ПО в сетевой инфраструктуре
- Открытость ПО → легче проводить анализ

Методы анализа ПО

- Формальная верификация
- Статический анализ
- Динамический анализ:
  - **фаззинг**
  - динамическое символьное выполнение

Фаззинг

- **Фаззинг** - техника тестирования программного обеспечения, часто автоматическая или полуавтоматическая, заключающаяся в передаче приложению на вход неправильных, неожиданных или случайных данных.
- **Фаззинг с обратной связью (по покрытию) –** разновидность фаззинга, использующая информацию о покрытии кода для повышения эффективности.
- **Очередь входных данных –** набор образцов входных данных для исследуемой программы, элементы которого мутируются для дальнейшей отправки в исследуемое приложение.

  В фаззинге с обратной связью образец данных, приводящий к росту покрытия программы, добавляется в очередь входных данных.

Сетевой протокол

Сетевой протокол - это набор правил, управляющих тем, как передаются данные между программами.

Stateless-протокол, или протокол без сохранения состояния - протокол общения между устройствами, который рассматривает каждый отдельный запрос как независимую транзакцию, не связанную с предыдущими запросами. Таким образом, коммуникация с сервером представляет собой независимые пары запросов-ответов. Примеры stateless-протоколов: HTTP, IP.

Stateful-протокол, или протокол с сохранением состояния – протокол, учитывающий внутреннее состояние сервера. Примеры stateful-протоколов: SSH, FTP, SMTP.

Сложности фаззинга протоколов

1. Недетерминированное выполнение ПО;

2. Низкая скорость сетевых приложений;

3. Отсутствие спецификации на многие протоколы;

4. Достижение глубоких состояний протокола:

- необходимость выполнения длинных валидных сетевых сессий.

5. Сложные зависимости между данными в сетевой сессии:

- согласованность модели доступа к памяти с состояниями протокола;
- TLV (тип, длина, значение);
- сжатие;
- контрольная сумма.

Сложности фаззинга протоколов

Как преодолеть указанные сложности:

1) Полноценный сетевой клиент → **трудоёмко**.
2) Описание спецификации протокола для фаззера → **трудоёмко**.

   Например, описание автомата состояний протокола и форматов сетевых сообщений в файле типа Peach Pit для фаззера Peach.
3) Снимки памяти в любом состоянии сервера.

   Не учитывая зависимости между данными, обработка пакетов в глубоких состояниях завершится на раннем этапе парсинга, не позволяя продвинуться дальше по протоколу.

   Учёт зависимостей → **трудоёмко**.

Пример - CVE-2021-3449

- TLS-рукопожатие + дальнейший обмен данными

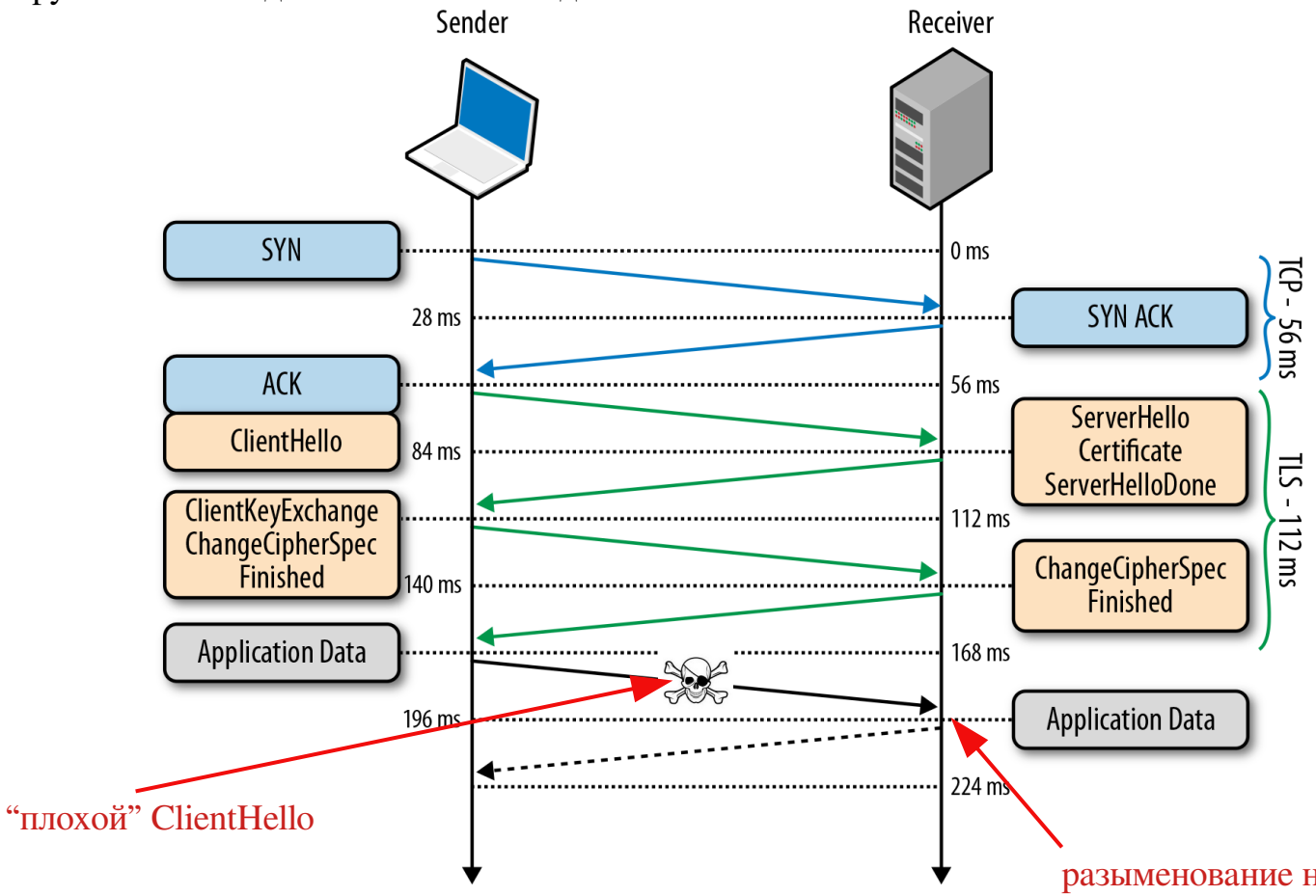


Рисунок позаимствован из https://habr.com/ru/articles/258285/

ИСП РАН

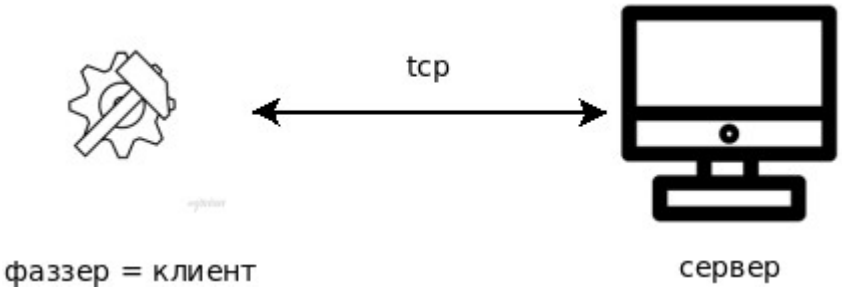
Цели

Разработать метод фаззинга, позволяющий:

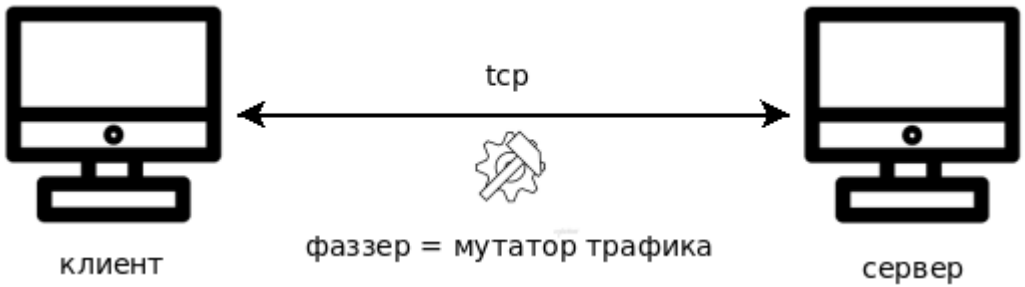
1) Автоматически учитывать сложные связи между данными и состояниями в протоколе для достижения глубоких состояний и повышения эффективности фаззинга.
2) Гибко настраивать мутации входных данных.

Фаззеры сетевых протоколов

- Фаззер в роли клиента



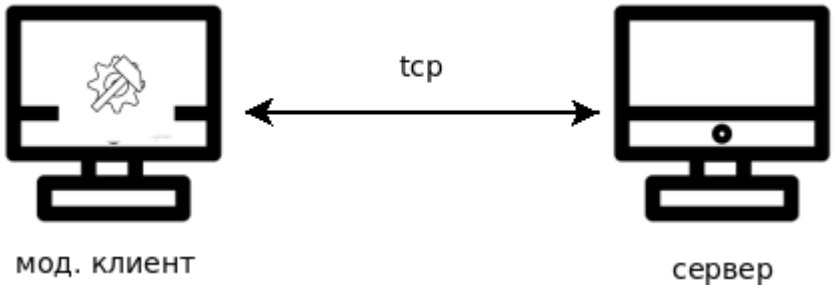
- Фаззер мутирует трафик между сервером и клиентом



Метод фаззинга с использованием модифицированного клиента

Возможности:

- Мутации памяти клиента
- API для гибкой настройки клиента



Требования:

- Наличие исходного кода клиента

Мутации клиента

- **Что мутируется –** целочисленные переменные в клиенте.

  Аналитик самостоятельно расставляет вызовы мутирующих функций, используя API мод. клиента. Пример:

  uint32_t c = a + b; // оригинальный код

  c = mutate_int(c, 4); // мутирующая вставка

- **Как мутируется –** применяются алгоритмы мутаций: bflip, arith (из фаззера AFL), выбор случайного значения байта.
- **Когда мутируется** - в процессе взаимодействия с сервером (объектом фаззинга). Это позволяет автоматически учитывать ответы сервера.

Мутации клиента

Возможный выбор переменных для мутаций:

1) Переменные – идентификаторы состояний;
2) Значения опций целевого приложения;
3) Значения полей сетевых пакетов до следующих преобразований:
   - добавление служебных заголовков;
   - вычисление: контрольных сумм, длины полезной нагрузки;
   - сжатие, шифрование.

Оптимизация клиента

- Для ускорения фаззинга во многих фаззерах (например, в AFL) используется технология fork-сервера, позволяющая не выполнять каждый раз тяжеловесный вызов exec при запуске процесса исследуемого приложения. Вместо этого каждый новый процесс порождается с некоторой точки выполнения с помощью вызова fork.
- **В модифицированном клиенте** также **используется fork-сервер**.

  Для установки точка вызова fork в клиенте в API мод. клиента добавлена функция *custom_fork_server*.

API мод. клиента

- *uint64_t mutate_int(uint64_t value, size_t len)*

  Выполняет мутации целочисленных значений в клиенте.

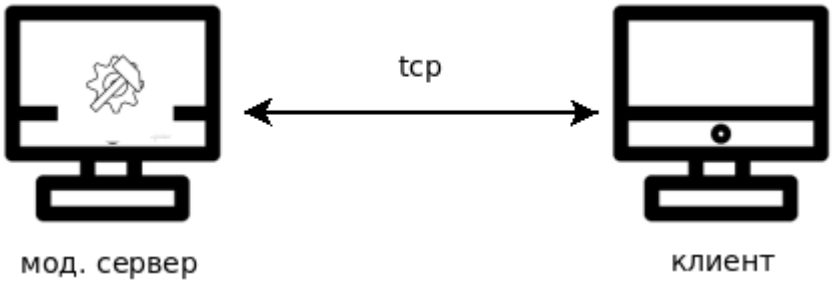
- *custom_fork_server()*

  Позволяет ускорить порождение новых процессов клиента.

Тестируется автоматизация вставки API вызовов с помощью инструментирующего компилятора.

Симметричная схема фаззинга

- Фаззинг клиента с помощью модифицированного сервера



Результаты (OpenSSL)

Эксперимент по фаззингу OpenSSL версии 1.1.1j:

- OpenSSL 1.1.1j содержит ряд дефектов
- Объект фаззинга — TLS-сервер:

  - работает по протоколу TLS 1.2

  - использует rsa-ключ и x509-сертификат

  - завершается после 1 соединения
- Начальный сценарий: рукопожатие (handshake) + переустановка параметров TLS-сессии (renegotiation) + выход.
- Запуск: метод модифицированного клиента, 20 ядер, 1 час.
- В результате сгенерированы, данные приводящие к **известной CVE-2021-3449** (рассмотрена ранее).

Заключение

В данной работе представлен метод фаззинга сетевых протоколов с использованием модифицированного клиента, который позволяет:

- Достигать глубоких состояний в протоколах со сложными связями между данными без необходимости значительных временных затрат на подготовку к фаззингу, что включает глубокое изучение протокола. Это повышает шанс обнаружить дефекты в исследуемом ПО.
- Гибко настраивать работу модифицированного клиента за счёт разработанного API.

Данный метод был апробирован на протоколе TLS (OpenSSL) и показал свою способность находить ошибки.

ИСП РАН

www.ispras.ru