Министерство науки и высшего образования Российской Федерации
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)
Институт прикладной математики и компьютерных наук
Кафедра компьютерной безопасности

ДОПУСТИТЬ К ЗАЩИТЕ В ГЭК
Руководитель ООП
канд. техн. наук, доцент
_____ В. Н. Тренькаев
“ 17 ” января 20 22 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА СПЕЦИАЛИСТА (ДИПЛОМНАЯ РАБОТА)

РАЗРАБОТКА И РЕАЛИЗАЦИЯ ЛЕГКОВЕСНЫХ ФОРМАЛЬНЫХ МЕТОДОВ ТЕСТИРОВАНИЯ СВОЙСТВ БЕЗОПАСНОСТИ МЕХАНИЗМОВ ЗАЩИТЫ ОТ БОТОВ

по специальности 10.05.01 Компьютерная безопасность,
специализация (профиль) «Анализ безопасности компьютерных систем»

Станкеев Павел Эдуардович

Руководитель ВКР
канд. тех. наук, доцент
_____ Колегов Д. Н.
“ 17 ” января 20 22 г.

Автор работы
студент группы № 1165
_____ Станкеев П. Э.
“ 17 ” января 20 22 г.

Томск–2022

Министерство науки и высшего образования Российской Федерации
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)
Институт прикладной математики и компьютерных наук
Кафедра компьютерной безопасности

УТВЕРЖДАЮ
Руководитель ООП
канд. техн. наук, доцент
_____ В. Н. Тренькаев
“ 07 ” октября 20 21 г.

ЗАДАНИЕ
по выполнению выпускной квалификационной работы специалиста обучающемуся
Станкееву Павлу Эдуардовичу
Фамилия Имя Отчество обучающегося
по специальности 10.05.01 Компьютерная безопасность, специализация (профиль) «Анализ безопасности компьютерных систем»

1 Тема выпускной квалификационной работы
Разработка и реализация легковесных формальных методов тестирования свойств безопасности механизмов защиты от ботов

2 Срок сдачи обучающимся выполненной выпускной квалификационной работы:
а) в учебный офис / деканат — 17.01.2022 б) в ГЭК — 28.01.2022

3 Исходные данные к работе:

Объект исследования —	программные реализации систем защиты от ботов для современных веб-приложений.
Предмет исследования —	методы тестирования корректности систем защиты веб-приложений
Цель исследования —	обеспечение свойств безопасности и корректности реализации механизмов защиты от ботов в межсетевых экранах уровня приложения

Задачи:
— разработать исполняемые эталонные спецификации механизмов защиты от ботов
— описать интерфейсы механизмов защиты от ботов
— сформулировать свойства безопасности разработанных моделей
— интегрировать методы тестирования в жизненный цикл разработки реальной системы защиты от ботов

Методы исследования:

прототипирование, проведение экспериментов, методы тестирования, методы обеспечения качественного программного обеспечения, методы компьютерной безопасности, методы моделирования.

Организация или отрасль, по тематике которой выполняется работа, —

05.13.19

4 Краткое содержание работы

В рамках данной работы необходимо разработать исполняемые эталонные спецификации механизмов обнаружения нежелательной автоматизации, сформулировать для них свойства безопасности и описать интерфейсы взаимодействия, а также проверить реализацию этих механизмов на наличие свойств безопасности модели. Найденные отклонения реализации от модели описать и исправить. Интегрировать подход легковесных формальных методов тестирования в разрабатываемую на данный момент систему защиты от ботов компании Wallarm. Описать особенности данного подхода и результаты его применения.

Научный руководитель выпускной
квалификационной работы,
доцент кафедры КБ _____ / Колегов Д. Н./

Задание принял к исполнению,
студент группы № 1165 _____ / Станкеев П. Э./

АННОТАЦИЯ

Актуальность: автоматизированные с помощью программ методы использования веб-приложений называются ботами. Существует огромное количество методов разработки ботов, и они постоянно совершенствуются. Как следствие, существует угроза нежелательной автоматизации работы с системами. Для противодействия этой угрозе некоторые компании, такие как AWS, CloudFlare и Яндекс, вынуждены активно разрабатывать и поддерживать средства защиты от ботов. Другие компании, не обладающие такими же ресурсами, вынуждены прибегать к покупке готовых программных продуктов.

Помимо всего прочего, механизмы защиты от ботов являются сложными, распределенными и постоянно изменяющимися. В связи с вышесказанным, необходимо на постоянной основе осуществлять тестирование данных систем.

Объект исследования: программные реализации механизмов защиты от ботов.

Предмет исследования: методы тестирования корректности систем защиты веб-приложений.

Цель работы: обеспечение свойств безопасности и корректности реализации механизмов защиты от ботов с применением легковесных формальных методов тестирования.

Задачи работы:

— разработать исполняемые эталонные спецификации механизмов защиты от ботов;

— описать интерфейсы механизмов защиты от ботов;

— сформулировать свойства безопасности разработанных моделей;

— интегрировать методы тестирования в жизненный цикл разработки реальной системы защиты от ботов.

Результат: разработаны эталонные реализации механизмов защиты от ботов, описаны программные интерфейсы компонентов системы, сформулированы свойства безопасности моделей, реализации механизмов протестированы с помощью легковесных формальных методов тестирования свойств, найденные ошибки описаны и исправлены.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В настоящее время существует большое количество веб-приложений, и их число постоянно растет. Вместе с этим растет и количество автоматизированных услуг для поддержки приложений, сбора статистики, индексации и прочего, реализованных с помощью компьютерных программ. Подобные автоматизированные методы использования веб-приложений крайне активно развиваются. Такие программы называются ботами.

Боты – это программы, выполняющие повторяющиеся задания автоматически или по расписанию. Как и любые программы, боты могут использоваться как во благо, так и во вред. Поисковые системы используют их для поиска и методичной каталогизации информации с веб-сайтов. Некоторые сайты и службы используют ботов для агрегирования информации. Существуют также и вредоносные. Злоумышленники активно используют в своей деятельности таких ботов, которые совершают атаки: захват учетных записей, переборы карточных данных, отказ в обслуживании, краулинг.

Таким образом, боты несут в себе угрозу в виде нежелательной автоматизации. Для противодействия этой угрозе многие компании такие как AWS, CloudFlare или Яндекс вынуждены активно разрабатывать и поддерживать средства защиты от ботов. Другие компании, не обладающие такими же ресурсами, приобретают сторонние решения.

Для общения с сервером приложения боты используют такие же интерфейсы взаимодействия, как и обычные пользователи. Для их защиты используются межсетевые экраны уровня приложений. Механизмы защиты от нежелательной автоматизации могут являться одним из многих компонентов межсетевого экрана. Они способны с некоторой вероятностью определить нежелательную автоматизацию на основе свойственных ей признаков.

Современные межсетевые экраны являются веб-приложениями. Их разработка ввиду развития современных технологий очень сложна как с точки зрения проектирования, так и реализации. Веб-приложения содержат в себе огромное количество сервисов, библиотек, платформ в зависимости от целей, компании и технологического стека. Кроме того, они часто являются распределенными, что добавляет множество фундаментально теоретически неразрешимых проблем. В силу появления новых способов атак и методов защиты от них межсетевые экраны уровня приложения постоянно находятся в разработке. В связи с этим необходимо на постоянной основе тщательно проверять корректность работы системы.

В 2021 г. была опубликована статья [1], в которой были предложены разработанные в рамках совместных усилий ученых из университетов и практических специалистов из Amazon Web Services (AWS) легковесные формальные методы тестирования, а также описано их применение к собственному распределенному продукту ShardStore

– это хранилище типа ключ-значение, которое в настоящее время развертывается в облачной службе хранилища объектов Amazon S3.

Ключевыми особенностями легковесных формальных методов являются отсутствие математической модели, разработка исполняемой эталонной спецификации системы на безопасном языке программирования и высокая степень автоматизации. Далее исполняемые эталонные спецификации будем также называть моделями. Было решено адаптировать указанный метод для разрабатываемой в настоящее время системы защиты от ботов в компании Wallarm.

В рамках данной работы были разработаны исполняемые эталонные спецификации механизмов обнаружения ботов, описаны интерфейсы и заданы свойства корректной работы системы, было проверено, что реализация этих механизмов удовлетворяет заданным свойствам. Найденные отклонения реализации от модели были исправлены. Легковесные формальные методы тестирования внедрены в жизненный цикл разработки.

Более формально цель и задачи данной работы могут быть описаны следующим образом.

Целью работы является обеспечение свойств безопасности и корректности реализации механизмов защиты от ботов с применением легковесных формальных методов тестирования.

В рамках данной цели необходимо решить следующие **задачи**:

1) разработать исполняемые эталонные спецификации механизмов защиты от ботов;

2) описать интерфейсы механизмов защиты от ботов;

3) сформулировать свойства безопасности разработанных моделей;

4) интегрировать методы тестирования в жизненный цикл разработки реальной системы защиты от ботов.

В первой главе работы обозначается представленная проблема, излагаются возможные пути её решения и адаптируемые в работе методы.

Во второй главе работы приводится информация о возможных механизмах защиты от ботов, их архитектуре и представляются примеры ботов, описываются адаптируемые методы тестирования, способы их применения и причины их использования.

В третьей главе подробнее рассматривается применение методов тестирования в системе защиты от ботов, расписываются особенности и формулируются свойства реализации механизмов, приводятся разборы примеров функций тестирования на основе свойств, а также демонстрируются найденные в системе ошибки.

1 Краткая характеристика предметной области

Межсетевые экраны уровня приложений являются веб-приложениями и средствами защиты. Механизмы защиты от ботов являются их составной частью. Как следствие, одними из требований к ним являются отказоустойчивость и способность работать корректно. Вне зависимости от архитектуры системы отказ в обслуживании механизмов может вызвать как прекращение работы защищаемой системы, так и наплыв ботов, который также приведет к неблагоприятным последствиям.

Кроме того, ввиду развития современных технологий разработки программного обеспечения, такие системы являются распределенными и часто зависимыми от внешних компонентов. Помимо этого, множество механизмов защиты и число способов их обходов постоянно растет. Следовательно, системы непрерывно обновляются, и в процессе их улучшения и разработки важно не допустить поломок разработанных ранее механизмов.

Тестирование распределенных систем обладает рядом особенностей, которые должны учитываться для обнаружения сложных ошибок системы. Необходимо учитывать, что каждый из компонентов системы может быть запущен, выключен или вовсе работать так, как мы того не ожидаем.

Одним из главным подходом тестирования распределенных систем является формальная верификация. Существует множество исследований [1—4], в которых описан опыт использования формальной верификации при тестировании распределенных систем. Так, в исследовании [2] предложена расширенная аварийными состояниями логика Хоара, реализуемая с помощью средства доказательства теорем Coq[5]. С её помощью инженеры способны описать спецификацию для аварийно-безопасной системы хранения и доказать, что она работает корректно. В статье представлен пример такой системы с доказательством её корректности, что является хорошим результатом.

Однако такой подход требует значительных затрат. Предложенная ими расширенная логика Хоара применима в узкой предметной области, в силу чего необходимо её новое расширение. Описание спецификации также потребляет большой объем ресурсов, а с учетом архитектуры системы защиты от ботов и её постоянного обновления это, вероятно, будет нецелесообразно.

В [3] предоставлен инструмент для верификации файловых систем одним нажатием, принимающий на вход тройку параметров: спецификацию специального вида, реализацию и инварианты согласованности, указывающие, находится ли образ файловой системы в согласованном состоянии. Особенностью подхода является использование одного языка программирования при описании спецификации и реализации системы, использование SMT-решателя Z3[6] и предоставление контрпримера в случае обнаружения ошибки в реализации. Данный подход отличается от прошлого иным подходом

к разработке модели, и использованием языка реализации для её описания. Описанный в статье подход обладает теми же недостатками, что и прошлый.

В работе [4] подняты проблемы сложности проведения верификации большой системы и её продолжительности. В исследовании предлагается использовать языки программирования с безопасностью типов и памяти, в данном случае, расширяют язык Dafny[7], делая его похожим на Rust[8]. Помимо этого, они автоматизировали процесс и показали способность подхода к масштабированию к большей кодовой базе.

Наконец, в [1] описан опыт тестирования распределенных систем, свидетельствующий о том, что достичь корректной работы системы можно с помощью легковесных формальных методов.

Одной из ключевых особенностей легковесных методов является отсутствие математической модели и соответствующей ей промежуточных представлений, специфичных для систем с формальной верификацией. Данная особенность позволяет инженерам затрачивать значительно меньше ресурсов при тестировании системы.

В данном подходе моделями являются специально разработанные на безопасных языках программирования исполняемые эталонные спецификации, которые определяют допустимое последовательное безаварийное поведение системы. Важным является то, что при разработке модели необходимо выбирать язык программирования так, чтобы он обладал не меньшим количеством свойств безопасности, чем язык программирования реализации системы. При выборе языка с большими свойствами усиливаются свойства реализованной системы.

Модели, разработанные на языке программирования реализации, позволяют всем разработчикам создавать и поддерживать их, не затрачивая время на изучение неизвестных им языков верификации.

Модели представляют собой облегченные реализации. Примером упрощения может являться использование структуры языка программирования вместо системы управления базами данных, отсутствие фоновых процессов или независимость от сторонних систем. Такой подход значительно уменьшит количество написанного кода, использованных внешних библиотек и возможных обработок ошибок, что позволит проще анализировать код и логику работы разрабатываемой системы.

Высокая степень автоматизации является еще одной особенностью данного подхода. Она подразумевает внедрение тестов на основе свойств [9] в жизненный цикл разработки, использование моделей при функциональном и интеграционном тестировании.

Тестирования на основе свойств используется в качестве одного из основных инструментов, использованных в подходе. Такое тестирование помогает удостовериться в функциональной корректности системы путем проверки соответствия поведения реализации и модели. В таком случае модель и реализация должны иметь одинаковые интерфейсы. Подход подразумевает множественное исполнение функции тестирования

с подачей случайных параметров. В процессе тестирования некоторые из компонентов выходят из строя, перезагружаются или ведут себя не так, как это описано в спецификации. Это позволяет проверить способность работы распределенной системы вне зависимости от поведения её компонентов. Каждая функция тестирования в описанном подходе должна проверять некоторое заданное свойство системы, например, независимость доступности системы от работоспособности внешних компонентов. Условия выполнения свойства описываются в функции тестирования.

Таким образом, формальные методы позволяют проверить наличие заданных свойств реализации, функциональную правильность вызовов уровня API, согласованность структур данных и состояний системы, корректность параллельных вызовов API и задач фонового обслуживания, учитывая инварианты состояний компонентов системы. Но в силу сложности распределенных систем и количества затрат ресурсов на формальную верификацию для тестирования таких систем предлагается использовать легковесные формальные методы, что, однако, даёт меньшую уверенность в наличии выявленных свойств у данной системы, в отличие от формальных методов.

Однако с учётом перечисленных выше особенностей легковесных методов считаем, что они предоставляют достаточную степень уверенности в корректной работе системы, при этом не используя огромного количества ресурсов, в отличие от формальной верификации, при которой процесс может занять продолжительное время, в течение которого система устаревает, в силу постоянного обновления и улучшения механизмов защиты.

2 Механизмы защиты от ботов и методы тестирования

2.1 Общее описание систем защиты от ботов

В виду развития современных технологий разработки программного обеспечения современные веб-приложения, в том числе межсетевые экраны уровня приложений, являются распределенными системами. Для корректной работы межсетевого экрана необходимо соблюдать все требования защищаемой системы. Так, если защищаемое приложение обладает свойством высокой производительности, то и защищаемая его система также должна обладать этим свойством. Поскольку механизмы защиты от ботов являются составной частью межсетевых экранов уровня приложения, они также являются распределенными системами.

Вместе с серией преимуществ, распределенные системы обладают рядом проблем с обработкой сбоев: некоторые компоненты системы могут выходить из строя, пока другие продолжают работу. Подобные ситуации не должны сказываться на доступности всей системы. Помимо этого, некоторые компоненты системы могут обладать доступом к общему ресурсу, поэтому необходимо убеждаться в безопасности ресурса в параллельной среде.

На рисунке 1 изображена типовая схема архитектуры системы защиты от ботов, где прямоугольники с закругленными углами — это сторонние компоненты.

Примером объекта **Web Client** является браузер или мобильное приложение пользователя со встроенными в них агентами. Агенты ответственны за:

— сбор данных об окружении и поведении пользователя;

— генерацию уникального идентификатора, основанного на данных об окружении, аппаратном обеспечении и характеристиках Web Client;

— коммуникацию с компонентом Security mechanisms API, получение токена доступа;

— решение поддерживаемых задач, отправку результатов Security mechanisms API;

— поддержку механизма CAPTCHA.

Security mechanisms API реализует основные механизмы по обнаружению ботов по данным, полученных от агентов. Помимо этого, Security mechanisms API ответственен за:

— получение и применение настроек для каждого клиента системы от Configuration API через базу данных;

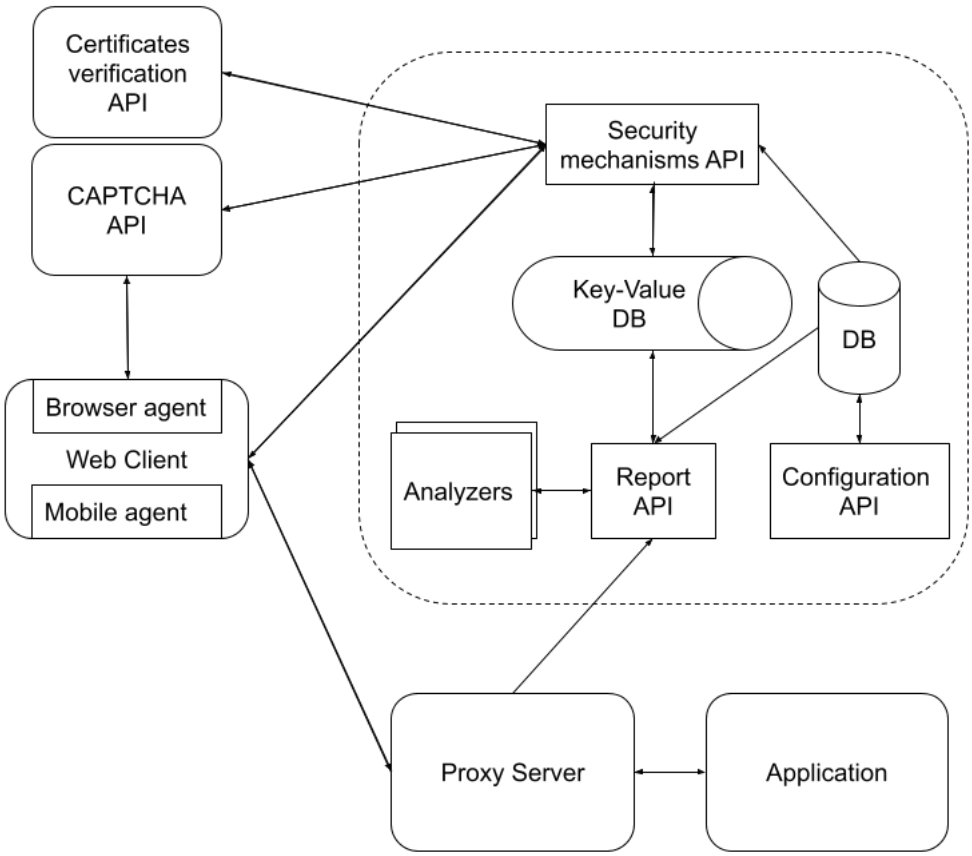


Рисунок 1 — Архитектура системы защиты от ботов

— отслеживание и обновление состояния пользователей в Key-Value DB;

— коммуникацию со сторонним сервисом Certificates Verification API для проверки статусов отзыва сертификатов при работе механизма Key Attestation;

— проверку решения CAPTCHA пользователем с помощью CAPTCHA API;

— обновление состояния пользователей на основе данных в Key-Value DB, полученных от Report API.

Proxy Server следит за соблюдением указанной политики управления ботами. Этим компонентом может являться любой сервис, который обладает способностью фильтрации, проксирования и редактирования запросов. Примером может служить любой межсетевой экран уровня веб-приложения.

Через **Report API** сторонние сервисы (Proxy Server, Analyzers), пройдя авторизацию, могут запросить CAPTCHA для конкретного пользователя, если по каким-либо причинам выявили в нём бота. Таким образом, Proxy Server, обладающий способностью обнаружения ботов или блокировки пользователей, может сделать запрос на Report API, заблокировав этим пользователя до тех пор, пока он не решит CAPTCHA.

Компонент **Configuration API** позволяет авторизованным администраторам защищаемого компонента Application конфигурировать профиль своего приложения в Security mechanisms API. В конфигурации можно указать механизмы защиты, длительность жизни токенов доступа и другие опции.

Analyzers является множеством микросервисов, каждый из которых способен определять нежелательную автоматизацию и докладывать об этом через Report API.

2.2 Типы ботов и способы их атак

Вредоносные боты могут быть запрограммированы на взлом учетных записей пользователей, поиск контактной информации в интернете, рассылку спама и выполнение других вредоносных действий. Боты могут объединяться в ботнет, множество устройств с различными IP-адресами, что затрудняет блокирование трафика ботов при атаке.

Спам-боты размещают рекламу в социальных сетях, в комментариях различных сервисов для привлечения внимания к определенным интернет-ресурсам, собирают e-mail адреса с веб-сайтов социальных сетей, предприятий и организаций и пр. После сбора большого количества адресов электронной почты возможно использовать их:

— для взлома учетных данных путем перебора популярных паролей;

— отправки спама с рекламой или фишингом.

Помимо хищения злоумышленниками личных e-mail адресов пользователей, компании сталкиваются с увеличением затрат на содержание используемых сервисов из-за роста трафика.

Боты социальных сетей используются для генерации сообщений, накрутки подписчиков и др. Такие боты часто используются для трансляции, распространения необходимых идей и мнений на определенную социальную тему в интернете. Отличить их от настоящих пользователей очень сложно, потому что они могут имитировать поведение реальных людей.

DoS и DDoS-боты используются для создания чрезмерной нагрузки на ресурсы сервера и остановки работы сервисов путем DDoS-атак, эксплуатации известных уязвимостей или пользуясь логикой приложения. Например, боты, атакующие интернет-магазины, могут покупать или бронировать ограниченный популярный товар с целью его перепродажи (билеты на концерты, видеокарты). В результате такой атаки реальные пользователи не имеют возможности сделать покупку.

Сканеры уязвимостей часто используются для поиска уязвимостей на веб-сайтах с целью их продажи или взлома, что несет непосредственный вред владельцам интернет-ресурсов.

Боты-скраперы используют для сбора больших объемов данных с веб-сайта с целью повторного использования или продажи этих данных в другом месте.

Таким образом, вредоносные боты используются для:

— кражи конфиденциальных данных;

— атак на веб-сайты;

— получения денег путем вымогательства, продажи собранных данных и самих ботов.

2.3 Общее описание механизмов защиты и их примеры

Так как зачастую действия ботов несут реальную угрозу, присутствует потребность в защите от них. При этом необходимо учитывать, что не все боты являются вредоносными – есть и те, которые приносят пользу и способствуют развитию веб-сайтов.

Таким образом, механизмы защиты должны учитывать данный факт и уметь отличить вредоносного бота по наличию некоторого набора признаков.

Далее будут описаны примеры механизмов обнаружения ботов.

Android's Key Attestation

Данный механизм работает для устройств на базе Android и дает нам больше уверенности в том, что криптографические ключи, которые используются в приложении, генерируются и хранятся в хранилище ключей устройства с аппаратной поддержкой[10].

На рисунке 2 продемонстрирована схема работы механизма, его можно описать интерфейсами клиента и сервера. Клиент должен уметь генерировать рандомные байты и проверять цепочку сертификатов – такой интерфейс назовем Key Attestation Client. Поддерживающие этот механизм устройства предоставляют программный интерфейс – назовем его Key Attestation Server. С помощью него можно получить цепочку сертификатов, в одном из которых будет находится подписанная строка в поле расширения сертификата[11].

Проверку Key Attestation Cleint'ом цепочки сертификата можно разбить на несколько этапов:

— проверка подлинности сертификатов (каждый сертификат в цепочке должен быть подписан парой ключей его родителя);

— проверка корневого сертификата;

— проверка статуса отзыва сертификата;

— проверка подписанной строки.

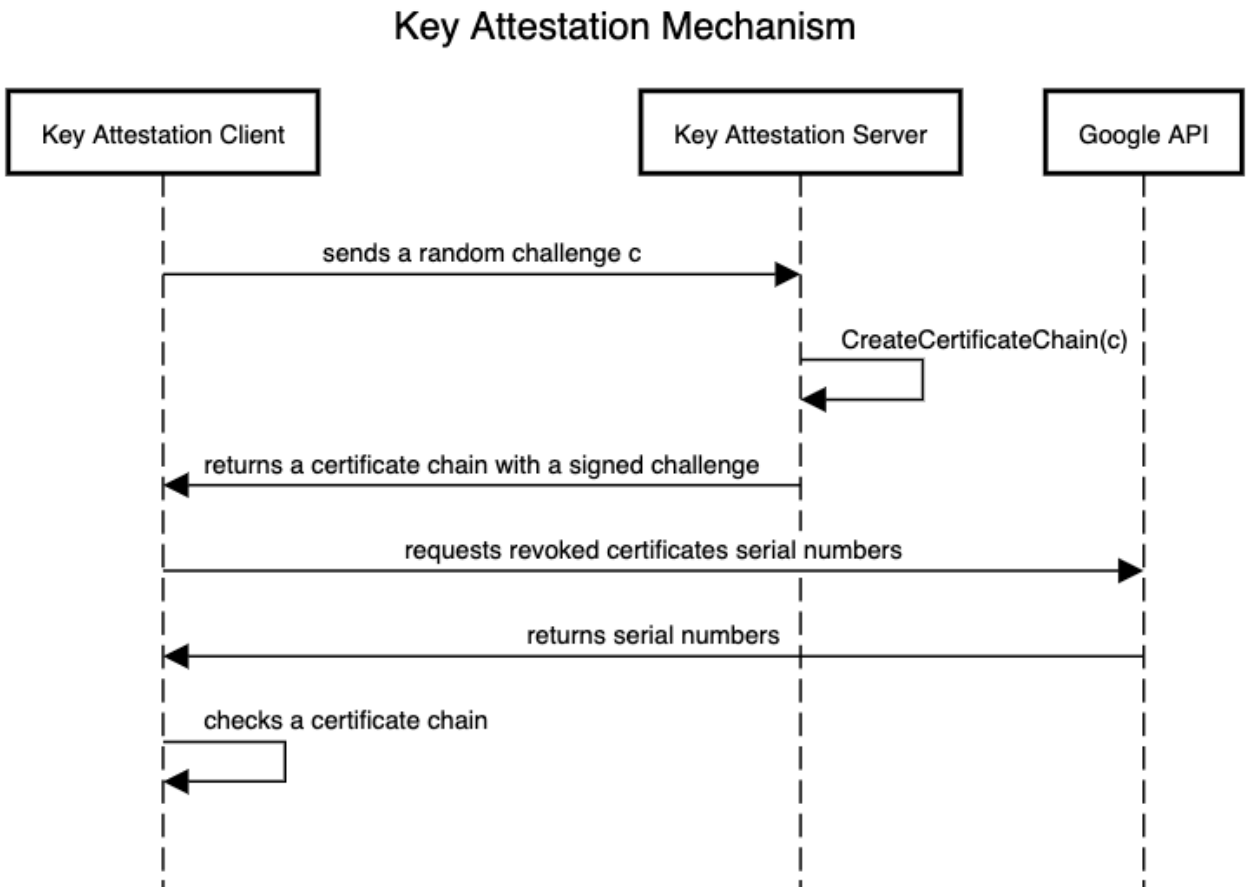


Рисунок 2 — Механизм аттестации ключа в общем виде

Для проверки статуса отзыва сертификата используется стороннее API, предоставляемое серверами Google.

Данный механизм защищает нас от скомпрометированных и виртуальных устройств на базе Android, которыми могут пользоваться злоумышленники.

Например, при корректной работе Key Attestation приложения с магазинами, банковские приложения не будут работать на эмуляторах или скомпрометированных устройствах, что мешает злоумышленникам исследовать и автоматизировать атаки на систему с этих устройств (перебор учетных данных, автоматическая покупка товаров и т. д.).

Данные о компрометации устройства или его криптографического модуля предоставляет Google. Факт компрометации также фиксирует представленная компания, при этом дополняя список отозванных сертификатов.

К недостаткам механизма можно причислить тот факт, что его работа зависит от компаний-производителей смартфонов, некоторые из которых не поддерживают описываемый механизм. Следовательно, Key Attestation не может быть единственным механизмом обнаружения автоматизации на устройствах на базе Android.

Механизм CAPTCHA

Описываемый механизм представляет из себя тест, который достаточно легко решить человеку, но трудоёмко программным образом. Представленный механизм используют в качестве некоторой проверки на предмет нежелательной автоматизации и выдают тест при подозрительной активности. Примерный алгоритм работы механизма продемонстрирован на рисунке 3.

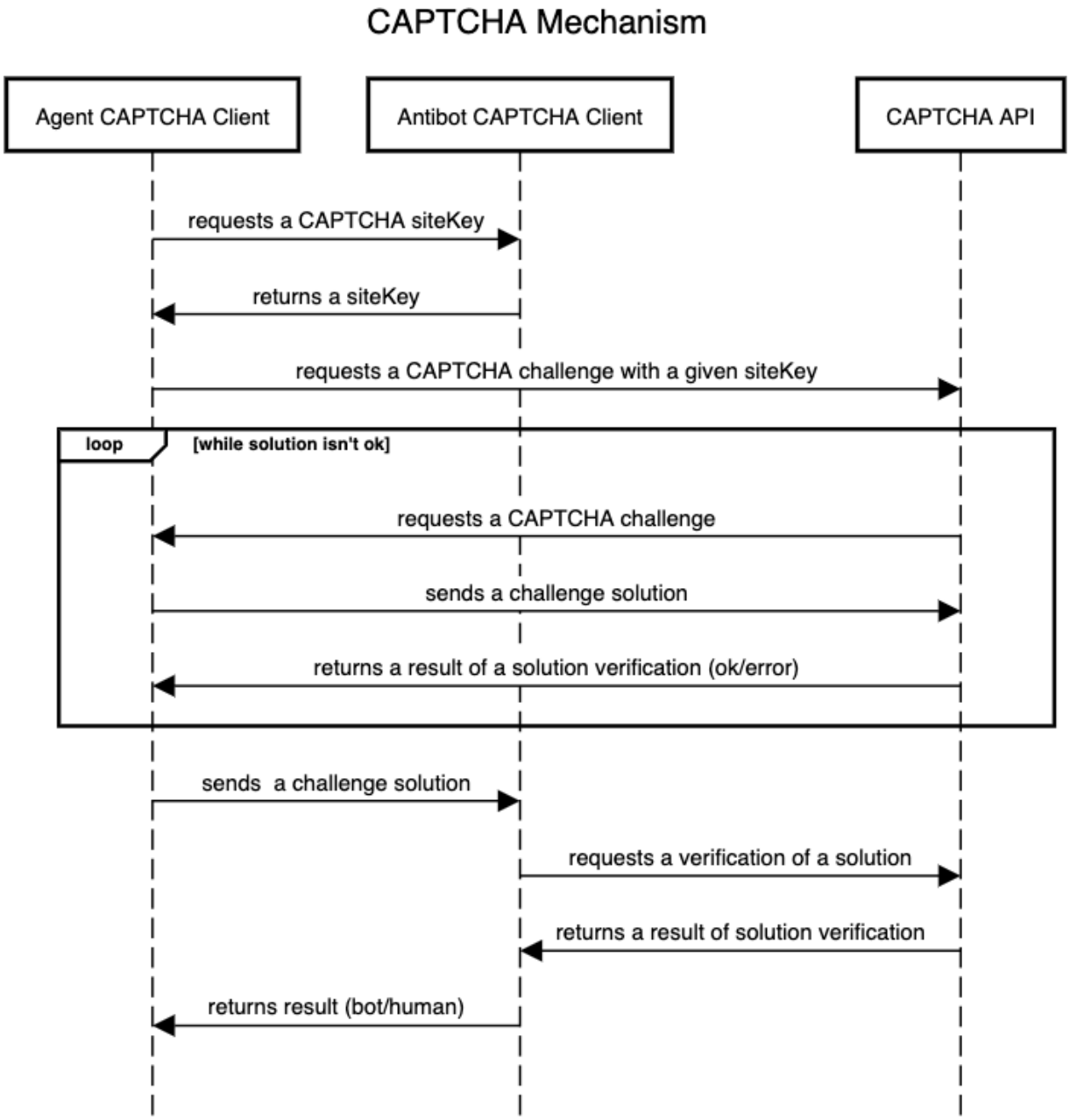


Рисунок 3 — Механизм CAPTCHA в общем виде

Механизм описывается тремя компонентами:

— CAPTCHA API – сторонним сервисом, генерирующим задачи и проверяющим решения;

— CAPTCHA Client, установленным на стороне сервиса защиты от ботов;

— Agent CAPTCHA Client, установленным на стороне пользовательского клиента.

В случае работы механизма CAPTCHA на веб-сайте, атакованном вредоносным ботом, перебирающим учетные данные или сканирующим на наличие уязвимостей, бот не сможет продолжать атаку без помощи человека после того, как механизм защиты отправит запрос пользователю на прохождение CAPTCHA.

Таким образом, данный механизм делает невозможным работу бота без участия человека или компрометации провайдера CAPTCHA API.

Заметим, что важным моментом в нем является двойная проверка решения, одна из которых происходит непосредственно во время процесса решения, а вторая — со стороны CAPTCHA Client во время верификации присланного агентом решения, что расширяет границы возможных атак.

Представленный механизм работает как для браузеров, так и для всех мобильных устройств, что делает его универсальным способом проверки пользователя, однако требует взаимодействия с пользователем.

2.4 Общее описание адаптируемых методов

Реализация сервиса защиты от ботов несет в себе некоторые трудности: сложные структуры в базе данных, одновременный доступ к ним и их изменения, а также необходимость поддерживать согласованность при сбоях. Помимо этого, постоянное обнаружение новых атак, способов обхода защит влечет за собой постоянное обновление системы. В силу этих условий и опыта исследований, описанных в разделе 1, было предложено использовать легковесные формальные методы тестирования, которые позволяют проверять корректность системы достаточно быстро и на постоянной основе.

Далее будут сформулированы свойства корректности системы, описаны адаптируемые легковесные формальные методы тестирования, их особенности и преимущества.

2.4.1 Формулировка свойств корректности системы

Система защиты от ботов должна быть надежной, согласованной при сбоях и параллелизме, высокодоступной во время нормальной работы и соответствовать требованиям по производительности. Свойства производительности могут быть протестированы на этапах нагрузочного и интеграционного тестирования, когда в свойствах

надежности и консистентности трудно убедиться, используя только традиционные методы тестирования.

Данный метод подразумевает разработку исполняемой спецификации системы, которая описывает переходы из одного состояния системы в другое после конкретных, описанных в спецификации, операций API. Исполняемую спецификацию будем называть эталонной реализацией или моделью.

Будем считать, что свойства надежности соблюдены, если модель и реализация остаются в одинаковых состояниях после каждого вызова API. Иными словами, мы убеждаемся, что реализация изменяет данные в своих структурах так же, как это делает модель. В случае механизма защиты от ботов эквивалентным состоянием системы будем считать то, где структуры состояния каждого пользователя системы будут одинаковы.

Поскольку система является распределенной, необходимо учитывать, что какой-либо компонент системы может выйти из строя во время работы или начать работать неожиданным для всей системы образом, что может влиять на состояние системы. Проверить подобные ситуация вышеописанный способ с участием модели не позволяет, как и ситуации с параллельными операциями.

Следовательно, проверку свойств надежности удобно декомпозировать таким образом:

— для последовательных вызовов API без выхода из строя компонентов системы;

— для последовательных вызовов API с учетом выхода из строя компонентов системы;

— для параллельных вызовов API системы.

Проблемы с параллельными вызовами API на момент написания работы не были учтены, поскольку не удалось найти способ, который бы позволил проверять эти свойства в автоматическом режиме с невысокими затратами ресурсов.

Декомпозиция позволяет сохранять эталонную реализацию облегченной, что упрощает её аудит. Более сложные свойства проверяются путем отдельного расширения модели для заданных задач.

Дополнительные свойства. При проверке наличия вышеописанных свойств мы убеждаемся, что основной функционал системы работает корректно. Однако для того, чтобы получить больше уверенности в правильной работе системы защиты, необходимо также проверять и локализованные свойства некоторых механизмов с помощью соответствующих инструментов.

Например, обработку параметров, согласованность типов параметров между системами и проверка прав доступа относятся к дополнительным свойствам. Для их тестирования может использоваться, например, проверка граничных значений параметров.

2.4.2 Использование эталонной реализации

Для каждого компонента системы разработана эталонная реализация, которая является исполняемой спецификацией на языке Go[12], предоставляющая тот же функционал, обладающая точно такими же интерфейсами, как и сам компонент, но с облегченной реализацией. Использование структуры типа хэш-таблица за место базы данных типа ключ-значение, будет хорошим примером облегчения реализации.

Поскольку модель обладает таким же интерфейсом, как и реализация, одинаковые операции над моделью и реализацией должны приводить к одинаковым переходам состояния системы. Таким образом, для проверки корректности реализации необходимо сравнивать её состояние с состоянием модели после осуществления каждой операции.

Однако при использовании моделей существуют некоторые сложности в проведении тестировании, такие как внеплановое отключение базы данных, потеря интернет-соединения или исчерпание ресурсов. Следовательно, такие случаи необходимо проверять другим образом. Одним из вариантов является расширение модели и эмулирование соответствующего поведения.

Разработка модели на языке программирования реализации, в отличие от специфичных языков формальной верификации (Promela[13], Alloy[14]), имеет ряд преимуществ:

— инженерам легче поддерживать модель в актуальном состоянии;

— код модели можно использовать при реализации, что уменьшает затраты ресурсов инженеров;

— модели компонентов можно использовать во всех видах тестирования.

Кроме того, упрощенная реализация модели предполагает уменьшение сложности анализа кода и логики работы системы.

2.4.3 Использование тестирования на основе свойств

Классическое модульное тестирование обладает рядом недостатков. Во-первых, трудно охватить множество различных входных данных с помощью ручного написания тестов, где не исключено допущение разработчиком ошибок. Во-вторых, для каждого теста необходимо записать некоторое утверждение, за которым нужно следить и обновлять в случае изменения API. Нечасто можно увидеть и тестирование граничных случаев для внешних зависимостей, поскольку обычно таким библиотекам доверяют безусловно, несмотря на то, что разработчики несут ответственность за их использование.

Тестирование на основе свойств применяется для проверки соответствия тестируемой функции заданному свойству. При использовании данного подхода нет необхо-

димости в том, чтобы задавать все тестовые примеры. Его задача — сопоставлять характеристики на выходе с заданным свойством. Свойство в данном контексте является некоторым утверждением. Для каждого утверждения необходимо реализовывать программный код в функции тестирования, который будет проверять выполнение свойства. При таком виде тестирования входные параметры генерируются случайным образом. При наличии нетривиальных входных параметров генераторы необходимо реализовывать инженерам самостоятельно. В остальных случаях библиотеки, помогающие реализовывать данный подход, предлагают свои генераторы для всех тривиальных типов данных, которые, к тому же, рассматривают для них граничные ситуации.

Таким образом, модульное тестирование проверяет работоспособность функции на конкретных примерах, когда тестирование, основанное на свойствах, проверяет функции на случайных данных множества раз за один запуск функций тестирования.

В нашем случае мы используем библиотеку gopter[15]. Она обладает описанным выше функционалом, а также имеет дополнительную особенность, которая позволяет в случае невыполнения утверждения подобрать минимальное значение параметра, при подаче которого в функцию тестирования свойство не будет выполнено.

2.5 Выводы

В данной главе была представлена основная актуальная информация по типовой архитектуре механизмов защиты от ботов, типам ботов и примерам их атак на момент написания работы.

Было дано общее описание работы механизмов защиты от ботов. В качестве примеров рассмотрены общий механизм CAPTCHA и механизм Key Attestation, применяющийся для мобильных устройств на платформе Android. Отмечены их особенности и недостатки.

Описаны адаптируемые методы тестирования, сформулированы и декомпозированы свойства корректной работы системы, представлена информация по особенностям применения легковесных формальных методов тестирования, а именно об использовании эталонной реализации и тестирования на основе свойств.

3 Применение легковесных формальных методов тестирования

При наличии у нас модели, определяющей ожидаемое поведение системы, необходимо проверить, соответствует ли реализация модели, опираясь на указанные выше свойства корректности.

3.1 Тестирование свойств корректности без выхода из строя компонентов

Для проверки того, что реализация соответствует свойствам модели, используется тестирование на основе свойств, в процессе которого для каждого запуска проверки системы генерируется множество входных значений, которые подаются поочередно в указанные функции тестирования. При этом заданное разработчиком свойство должно выполняться при любых сгенерированных параметрах при каждом запуске.

Тестирование на основе свойств можно рассматривать как расширенное фаззинг-тестирование с предоставленными пользователем свойствами и структурированными входными данными, которые дают возможность провести проверку поведения системы более углубленно. Входные данные, при которых свойство не выполняется, предоставляются инженеру для того, чтобы воспроизвести поведение системы и упростить процесс тестирования внесенных исправлений.

Использование тестирования на основе свойств позволяет убедиться, что любое наблюдаемое поведение реализации разрешено моделью. Таким образом, в качестве входных параметров подается произвольная последовательность случайных операций API, взятых из определенного нами алфавита. Такую последовательность будем называть сессией. Каждая операция применяется как к модели, так и к реализации. После выполнения каждой операции выполняется проверка на соответствие состояний модели и реализации. Другими словами, свойство будет выполнено тогда, когда и модель, и реализация будут иметь одинаковое состояние после каждой операции.

Рассмотрим пример с проверкой корректности изменения состояния пользователя в системе. Согласно спецификации, состояния могут изменять как сами пользователи, так и компоненты системы, реализующие механизмы защиты.

Состояние пользователя характеризуется большой структурой, но некоторые поля которой имеют конечное число значений и правила изменений. Например, если пользователю была выдана CAPTCHA, его состояние не может измениться на состояние, эквивалентное состоянию решенной CAPTCHA, без запроса с правильным её решением.

Механизмы обнаружения ботов включают в себя:

— проверка поведения;

— проверка окружения;

— проверки цепочки сертификатов;

— аутентификация агента процедурой типа "запрос–ответ";

— CAPTCHA;

— проверка отпечатков веб-клиента.

Помимо этого, существует специальный механизм для лиц, пользующихся мобильной версией приложения, который позволяет переподключиться и восстановить обмен данными по протоколу без получения блокирующих действий в случае потери соединения.

Таким образом, каждый вышеописанный механизм может каким-либо образом изменить состояние пользователя на другое состояние. Необходимо убедиться, что никаких других переходов, кроме реализованных в модели и описанных в спецификации, быть не может.

Для проверки этого факта требуется модель компонента, функционирующая по вышеописанной логике, и сам реализованный компонент. Для большей уверенности в невозможности возникновения некорректных изменений необходимо выполнить вызовы API в произвольном порядке со случайными параметрами. В силу сложности протокола и его сообщений абсолютно случайные данные во всех параметрах будут неэффективны.

Для большей эффективности вводим шаблоны API-запросов с некоторыми фиксированными параметрами в качестве составляющих входного алфавита. Для каждого из типов запросов реализованы генераторы, а также генератор последовательности запросов или сессии. Генератор сессии создает последовательность запросов для тестирования на основе свойств с учетом задания вероятности появления каждого из них. В теле функции тестирования после выполнения каждой операции API проверяется состояние пользователя в хранилище модели с состоянием, представленным в реализации. В случае несовпадения, функция возвращает соответствующие значение и указывает последовательность запросов для повторения ошибки.

В листинге 1 указан псевдокод теста, проверяющий свойство: в реализации не существует переходов состояния пользователя, не разрешенных в эталонной реализации.

```
1     api = []func() gopter.Gen{
2         MobileVerifyRequestGen,
3         VerifyCaptchaSolutionGen,
4         ...
5         GetCaptchaPageGen,
6         MobileReportClientErrorGen,
7     }
8
```

```
 9 TestCase := struct {
10         implPath  string
11         modelPath string
12         request   interface{}
13     }
14
15 func (st *serviceTest) testStatesAreAlwaysEquals(t
      *testing.T, testCases []TestCase) {
16     mobileBotDetector := detector.NewMobile()
17     referenceMobileBotDetector := refdetector.NewMobile()
18     for _, testCase := range testCases {
19             response, err :=
      st.newAPIRequest(testCase.implPath, testCase.request)
20             ...
21             refResponse, err :=
      st.newAPIRequest(testCase.modelPath, testCase.request)
22             ...
23             _, err = st.isStateEquals()
24             if err != nil {
25                 t.Error(err)
26             }
27         }
28 }
```

Листинг 1. Пример тестирования на основе свойств

На первых строках определяется алфавит операций (строки 1–7). Операции покрывают все возможные API-запросы от пользователя в сторону тестируемого Security Mechanisms API. Все элементы алфавита операций являются функциями, генерирующими и возвращающими данные для запроса к API.

Каждый запуск теста на основе свойств создает случайное множество операций и передает их функции testStatesAreAlwaysEquals (строка 15), где каждая операция выполняется отдельно на реализации (строка 19) и модели (строка 21) и после каждой происходит сравнение состояний (строка 23). В случае различия состояний последовательность операций выводится для повтора. При каждом запуске тесты на основе свойств выполняются указанное множество раз, что позволяет следить за выполнением свойства при каждом обновлении системы.

Дополнительный анализ последовательности переходов. В процессе тестирования на основе свойств функция сравнения состояния также сохраняет и анализирует последовательность на наличие невозможных переходов в контексте одной сессии. Другими словами, описываемая проверка предотвращает возникновение ситуации, при которой модель и реализация одновременно модифицированы так, что ведут

себя одинаково, но реализуют некорректные переходы состояний. Так, при проведении тестирования данный механизм указал на наличие ошибки, при которой заблокированный пользователь мог перейти в состояние до его блокировки. Ее возможно было бы выявить при аудите исполняемой спецификации, но автоматический анализ определил ошибку гораздо раньше – на этапе тестирования функционала.

Уровень покрытия. При тестировании на основе свойств выбираются случайные операции, что может приводить к пропуску ошибок, поскольку некоторые функции просто не будут вызваны в силу невыполнения условий. Мы значительно снижаем этот риск путем расширения генераторов операций.

В системе защиты от ботов реализовано множество механизмов проверки. Один из них – это механизм Key Attestation, подробнее описанный в подразделе 2.3. Данный механизм требует отправки цепочки сертификатов с правильной подписью. Отсюда следует, что при случайной генерации цепочки сертификатов механизм всегда будет выдавать отрицательный результат, посчитав, что запрос был сделан ботом.

Для противодействия подобному поведению многие отдельные от всей системы механизмы конфигурируются таким образом, что появляется возможность успешной проверки. Например, механизм аттестации ключей во время тестирования доверяет отозванным и небезопасным корневым сертификатам. Следовательно, используя такие сертификаты в расширении генераторов, можно сгенерировать подходящую цепочку, при подаче которой проверка будет пройдена. Важно не допускать такого поведения вне тестов.

Таким образом, генератор запросов API необходимо расширять так, чтобы он был способен создать запрос для любого реализованного механизма. Такой запрос сможет как пройти проверку механизма, так и провалить с заданием вероятностей этих событий.

Для механизмов без возможности подобной конфигурации, как, например, для анализа поведения пользователя системы, используется библиотека gomock [16]. По описанным интерфейсам моделей библиотека генерирует код, позволяющий задать аргументы функции, при подаче которых функция вернет ожидаемое значение.

Однако такой способ не позволяет протестировать механизмы, для которых использовалась представленная выше библиотека, поскольку код самой функции не исполняется. В таких случаях эти механизмы тестируются отдельно (подробнее об этом – в параграфе «Атомарное тестирование механизмов защиты» в подразделе 3.2).

Как итог, получаем генераторы запросов, способные сделать это так, что запрос либо пройдет, либо не пройдет каждую из проверок с заданной вероятностью. Это позволяет протестировать больший объем кодовой базы.

3.2 Тестирование свойств корректности с выходом из строя компонентов

Одним из требований к системе является доступность системы защиты, поскольку она напрямую влияет на доступность защищаемого приложения. В работе рассматриваются два следующих случая:

1) компонент системы недоступен с момента начала работы;

2) компонент перестал быть доступен во время работы.

Для этих случаев мы дополняем алфавит операций API, добавив в него операцию остановки и запуска некоторого компонента. Во избежание повторных остановок или повторных запусков одного и того же компонента генератор последовательностей операций был модифицирован таким образом, что операцию остановки компонента он устанавливает в случайную позицию последовательности, а запуск — в случайную позицию после операции остановки. Таким образом покрываются все возможные случаи отключения компонентов.

Написанный тест, продемонстрированный в листинге 1, необходимо дополнить операциями остановки и запуска компонентов, а в тело функции тестирования добавить условные операторы для обработки новых операций. В случае потери доступа к базе данных, содержащей информацию о состояниях пользователей, состояния изменяться не могут. В таком случае необходимо полагаться на спецификацию разрабатываемой системы.

В качестве примера рассмотрим компонент, хранящий в себе данные о возможных состояниях пользователей. В случае его выхода из строя все пользователи системы становятся практически неотличимы друг от друга; невозможно узнать, кто из них ранее считался ботом. В данной ситуации ожидается, что пользователи будут получать корректные токены доступа и не считаться ботами до тех пор, пока компонент не сможет вновь запуститься. Если доступные до остановки его работы данные не будут успешно восстановлены, то процесс проверки каждого пользователя начнется заново.

Другими словами, в спецификации заданы следующие свойства:

1) вне зависимости от работоспособности компонентов системы пользователь способен получить токен доступа к защищаемому приложению;

2) после успешного запуска работы базы данных информация о состояниях пользователей должна быть восстановлена.

Следовательно, функцию тестирования необходимо дополнить специальными условиями на случай отсутствия доступа к базе данных. Данные условия будут работать только в вышеописанной ситуации. Сформулируем их:

1) полученное в ответе состояние является начальным;

2) в ответе находится корректный токен доступа.

Заметим, что у модели реализации база данных отсутствует (её функционал реализован на уровне языка программирования), отключить её невозможно. Поскольку задано условие совпадения состояний пользователей у модели и реализации и после восстановления работы компонента пользователи и их состояния должны быть восстановлены, запросы к модели не отправляются во время отсутствия доступа к базе данных.

Таким образом, в данном подразделе был описан один из вариантов проверки наличия заданных в спецификации свойств тестируемой системы в случае выхода из строя некоторых её компонентов.

Атомарное тестирование механизмов защиты. Механизмы защиты предлагается тестировать отдельно, предварительно сформулировав их свойства, для большей уверенности в правильной работе.

В качестве примера рассмотрим механизм Key Attestation. Он имеет зависимость от не подконтрольного стороннего сервиса. Как следствие, у нас возникают следующие вопросы:

— что будет, если запрос не дойдет до Certificates Verification API?

— что будет, если Certificates Verification API изменит формат ответа и Security mechanisms API его не поймет?

В обоих случаях поведение системы будет непредсказуемо. Возможно возникновение ошибки, и тогда Security mechanisms API сделает запрос на прохождение CAPTCHA пользователю системы, что противоречит спецификации, поскольку пользователь не получает возможности пройти проверку Key Attestation. Таким образом, система фактически не предоставляет доступ пользователю к защищаемой системе, что влияет на её доступность. В данном случае необходимо считать, что цепочка сертификатов была верна.

Другими словами, пользователь, не являющийся ботом, всегда должен иметь доступ к защищаемой системе вне зависимости от наличия ошибок, связанных с работой сторонних сервисов.

Не менее важными являются свойства корректной работы механизма. Для Key Attestation их можно сформулировать таким образом:

1) если мы получаем ошибку во время запроса статуса отзыва сертификата, мы считаем, что сертификат не был отозван;

2) если мы не можем обработать полученный ответ от Certificates verification API во время запроса статуса отзыва сертификата, мы считаем. что сертификат не был отозван;

3) если запрос клиента Key Attestation содержит строку для подписи, сгенерированную клиентом Key Attestation, сервер Key Attestation способен подписать и отправить её обратно клиенту вместе с цепочкой сертификатов, клиент считает подпись правильной тогда и только тогда, когда подпись строки можно проверить с помощью сертификатов, корневой сертификат является доверенным и ни один сертификат из цепочки не был отозван Google.

На листинге 2 продемонстрирован пример тестирования второго свойства.

```
1  func testCheckRevocationStatusWithErrorOnDo(serialNumbers
       []string, client *reference.MockHTTPClient) bool {
2       ra := reference.NewWithCustomParams(logrus.New(),
       client, []string{TrustedRootCertificate1,
       TrustedRootCertificate2})
3
4       a := attestor{logrus.New(), client,
       []string{TrustedRootCertificate1,
       TrustedRootCertificate2}}
5
6       isRevokedCertificatesRef :=
       ra.CheckCertificatesRevocationStatus(serialNumbers)
7       isRevokedCertificates :=
       a.CheckCertificatesRevocationStatus(serialNumbers)
8
9       if isRevokedCertificatesRef != isRevokedCertificates &&
       isRevokedCertificatesRef {
10          logrus.WithFields(logrus.Fields{
11              "reference-model result":
       isRevokedCertificatesRef,
12              "implementation result":  isRevokedCertificates,
13              "expected":               false,
14          }).Error("the behavior of the reference-model and
       the implementation are different")
15          return false
16      }
17      return true
18 }
```

Листинг 2. Пример тестирования на основе свойств Key Attestation

На первых строках создается объект модели (строка 2) и объект реализации (строка 4). Далее объекты вызывают одни и те же функции(строки 6,7) и сравнивают результаты их вызова между собой (строка 9). В случае несовпадения выводится сообщение об ошибке со всеми необходимыми данными(строки 10–14). При запуске тестирования на основе свойств генерируется множество значений аргументов функции, среди которых – заведомо отозванные серийные номера сертификатов.

В результате тестирования мы убеждаемся, что в случае возникновения ошибки во время получения ответа от Certification verification API зависимость от входных параметров отсутствует, и выводом всегда является следующее утверждение: сертификаты не были отозваны.

Поскольку механизм CAPTCHA также является зависимым от стороннего компонента (рисунок 3), для него можно сформулировать аналогичные свойства, дополнив свойствами, выполнение которых говорит о корректных изменений состояний CAPTCHA пользователей. Свойства, совпадающие со свойствами Key Attestation, тестируются таким же образом. Тестирование корректности изменения состояний описано в подразделе 3.1.

3.3 Тестирование дополнительных свойств

Несмотря на то, что проверка вышеописанных свойств покрывают весь основной функционал системы, мы определили еще несколько классов ошибок, которые проверяем отдельно для большей уверенности в корректности работы системы.

Одним из классов является согласованность типов данных между всеми компонентами системы, особенно если компоненты написаны на разных языках. Например, компонент Configuration API может считать параметром первого типа, база данных – второго типа, а Report API – третьего, что может привести к непредвиденному поведению системы и, вероятно, к трудностям в поиске ошибки.

Некоторые из компонентов системы предполагают авторизацию (напр. Report API). Неавторизованный пользователь не должен получать какую-либо информацию от системы, помимо сообщения об отсутствии необходимых прав доступа. Следовательно, для авторизованных запросов корректным поведением будем считать такое, при котором структурно правильный запрос с согласованными типами данных никогда не повлечет за собой возникновение ошибки от компонента.

Соответствующие свойства можно сформулировать таким образом:

1) любой неавторизованный запрос всегда возвращает сообщение о недостаточных правах доступа;

2) если запрос имеет корректную структуру, данные и клиент авторизованы, клиент всегда получает корректный ответ без ошибок.

Корректным ответом считаются различные данные в зависимости от компонента, следовательно проверка второго свойства будет напрямую зависеть от его выбора.

3.4 Результаты применения

В результате применения адаптируемых методов тестирования были найдены ошибки системы.

Несогласованность типов данных между компонентами системы является самым часто встречающимся видом ошибки, однако не единственным. Например, механизм восстановления общения между мобильным агентов и системой защиты от ботов в случае потери соединения позволял перейти из заблокированного состояния, в которое пользователь приходит в случае неверного решения CAPTCHA, в состояние до блокировки, но с ожиданием верного ответа. Таким образом, пользователь имел бесконечное количество попыток решения CAPTCHA.

Также существовала ошибка, позволяющая пользователю вовсе уйти от блокировки, совершив созданный специальным образом запрос.

В процессе разработки модели и параллельного аудита кода реализации выяснилось, что анализ поведения пользователя проводился вне зависимости от его состояния, что позволяло злоумышленнику, даже будучи заблокированным, нагружать систему специально сгенерированными данными.

Краткое описание других найденных ошибок приведено в приложении в таблице ??.

Код созданных моделей компонентов значительно проще кода их реализации, что облегчает анализ и поддержку моделей. Пример сравнения одного из компонентов представлен в таблице 1.

Таблица 1 — Сравнительная таблица характеристик реализации и модели компонента

Параметр	Реализация	Модель
Строчки кода	2217	1269
Количество внешних зависимостей	6	1
Количество элементов	9	5

Интеграция эталонных моделей в процесс тестирования кода реализации имеет решающее значение для обеспечения того, чтобы проверки корректности системы оставались эффективными по мере изменения кода с течением времени. Кроме того, в силу большого покрытия и высокой автоматизации тестирования ошибки обнаруживаются на более ранних этапах разработки.

Однако в работе не рассматривается поведение системы при параллельных вызовах API, что является важным при тестировании распределенных систем. На момент написания работы не было найдено способа проверки корректности параллельных вызовов в автоматическом режиме и с невысокими затратами ресурсов. Данное направление является перспективным для будущего исследования.

3.5 Выводы

В рамках данной главы были подробнее рассмотрены методы тестирования и их применение в рамках реализуемой в настоящее время системы защиты от ботов.

Подробнее описаны некоторые реализованные в системе механизмы, необходимые для лучшего понимания работы предложенных тестовых функций, сформулированы их свойства. Введены понятия случайных последовательностей запросов API, их генераторы, описан общий алгоритм тестирования и разобраны конкретные примеры ситуаций с выходом из строя компонентов системы и без него, а также дополнительные свойства компонентов, включающие в себя корректную авторизацию и согласованность типов данных между компонентами системы.

Также были предложены дополнительные методы автоматического анализа, включающие в себя проверку последовательности переходов в рамках сессии и атомарное тестирование механизмов защиты, что помогает допускать меньшее количество ошибок при разработке реализации. Помимо этого, были кратко описаны найденные в системе ошибки.

ЗАКЛЮЧЕНИЕ

В данной работе описывается внедрение легковесных формальных методов тестирования в реализуемую в настоящее время систему защиты от нежелательной автоматизации. Продемонстрировано общее описание и архитектура систем защиты от ботов, представлены примеры механизмов обнаружения, описаны адаптируемые методы и их преимущества.

В процессе работы были разработаны модели для элементов системы, реализующие механизмы обнаружения ботов, с помощью которых возможна разработка компонентов на других языках программирования или проверка существующих на их корректность. Были описаны интерфейсы для взаимодействия с компонентами и сформулированы свойства безопасности механизмов.

Разработанные модели, совместно с тестированием на основе свойств, были использованы для проверки корректности реализации механизмов защиты от ботов при последовательном выполнении операций API с выходом из строя компонентов и без него.

Наконец, в результате проведения тестирования были найдены и исправлены ошибки системы, а легковесные формальные методы тестирования свойств внедрены в разрабатываемую на данный момент систему защиты от ботов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Using Lightweight Formal Methods to Validate a Key-Value Storage Node in Amazon S3 / J. Bornholt [и др.] // SOSP '21: Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles. — 2021. — с. 836—850.
2. Using Crash Hoare Logic for Certifying the FSCQ File System / H. Chen [и др.] // SOSP '15: Proceedings of the 25th Symposium on Operating Systems Principles. — 2015. — с. 18—37.
3. Push-Button Verification of File Systems via Crash Refinement / H. Sigurbjarnarson [и др.] // Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16). — 2016. — с. 1—16.
4. Storage Systems are Distributed Systems (So Verify Them That Way!) / T. Hance [и др.] // Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation. — 2020. — с. 99—115.
5. Introduction and Contents [Electronic resource] // Coq Reference Manual, Version 8.4pl5. — Electronic data. – 2014. — URL: `https://coq.inria.fr/distrib/current/refman/` (дата обращения: 22.01.2022).
6. *Bjorner N.* The Z3 Theorem Prover [Electronic resource] // GitHub. — Electronic data. – 2021. — URL: `https://github.com/Z3Prover/z3` (дата обращения: 22.01.2022).
7. Dafny: A Language and Program Verifier for Functional Correctness [Electronic resource] / C. Hawblitzel [и др.] // Microsoft : official website. — Electronic data. – 2010. — URL: `https://www.microsoft.com/en-us/research/project/dafny-a-language-and-program-verifier-for-functional-correctness/` (дата обращения: 22.01.2022).
8. *Labnik S. Nichols C.* The Rust Programming Language [Electronic resource]. — Electronic data. — URL: `https://doc.rust-lang.org/book` (дата обращения: 22.01.2022).
9. Property Based Testing [Electronic resource] // Gopher Academy Blog. — Electronic data. – 2017. — URL: `https://blog.gopheracademy.com/advent-2017/property-based-testing/` (дата обращения: 22.01.2022).
10. Key and ID Attestation [Electronic resource] // Android Open Source Project. — Electronic data. – 2021. — URL: `https://source.android.com/security/keystore/attestation?hl=el` (дата обращения: 22.01.2022).
11. IBM Sterling External Authentication Server documentation [Electronic resource] // IBM: official website. — Electronic data. – 2021. — URL: `https://www.ibm.com/docs/en/external-auth-server/2.4.3?topic=securing-x509-extensions` (дата обращения: 22.01.2022).

12. Go [Electronic resource]. — Electronic data. – 2021. — URL: https://go.dev/ (дата обращения: 22.01.2022).

13. *Holzman G. J.* Promela Manual [Electronic resource]. — Electronic data. — URL: https://courses.cs.washington.edu/courses/csep590/03su/Lectures/Promela.html (дата обращения: 22.01.2022).

14. Alloy [Electronic resource]. — Electronic data. — URL: https://alloytools.org/ (дата обращения: 22.01.2022).

15. GOPTER [Electronic resource] // GO. — Electronic data. – 2020. — URL: https://pkg.go.dev/github.com/leanovate/gopter (дата обращения: 22.01.2022).

16. Gomock [Electronic resource] // GO. — Electronic data. – 2021. — URL: https://pkg.go.dev/github.com/golang/mock/gomock (дата обращения: 22.01.2022).

ПРИЛОЖЕНИЕ А

Найденные недостатки

№ п/п	Компонент	Краткое описание
1	Security mechanisms API Mobile	Пользователь имел возможность обойти блокировку.
2	Security mechanisms API	Анализ поведения пользователя работал даже если пользователь заблокирован, что позволяло загружать систему.
3	Security mechanisms API	Некоторые механизмы неверно изменяли состояния пользователей.
4	Security mechanisms API Mobile	Один из механизмов позволял изменять состояние пользователя так, что появлялась возможность перебирать решения CAPTCHA.
5	Security mechanisms API	Работа механизма CAPTCHA была зависима от работоспособности CAPTCHA Server
6	Security mechanisms API	Есть возможность решить CAPTCHA без её запроса
7	Security mechanisms API Mobile	Вызов функции аттестации проверок окружения всегда пропускался в силу добавления нового механизма.
8	Mobile Agent	Неверная реализация одного из механизмом аутентификации.
9	Security mechanisms API	Несколько ошибок с несогласованностью типов между базой данных и сервисом.
10	Configuration API	Несколько ошибок с несогласованностью типов между базой данных и сервисом.

АНТИПЛАГИАТ
ОБНАРУЖЕНИЕ ЗАИМСТВОВАНИЙ

Отчет о проверке на заимствования №1

Автор: Stankeev Pavel
Проверяющий: Stankeev Pavel (nocasisn@gmail.com / ID: 8619796)

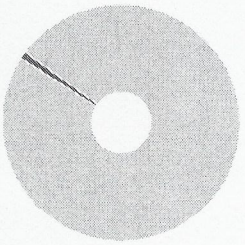
Отчет предоставлен сервисом «Антиплагиат» - users.antiplagiat.ru

ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 11
Начало загрузки: 25.01.2022 07:18:23
Длительность загрузки: 00:00:01
Имя исходного файла: BP.pdf
Название документа: BP
Размер текста: 61 кБ
Символов в тексте: 64037
Слов в тексте: 7578
Число предложений: 443

ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Начало проверки: 25.01.2022 07:18:24
Длительность проверки: 00:00:04
Комментарии: не указано
Модули поиска: Интернет Free



ЗАИМСТВОВАНИЯ	САМОЦИТИРОВАНИЯ	ЦИТИРОВАНИЯ	ОРИГИНАЛЬНОСТЬ
1,47%	0%	0%	98,53%

Заимствования — доля всех найденных текстовых пересечений, за исключением тех, которые система отнесла к цитированиям, по отношению к общему объему документа.
Самоцитирования — доля фрагментов текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника, автором или соавтором которого является автор проверяемого документа, по отношению к общему объему документа.
Цитирования — доля текстовых пересечений, которые не являются авторскими, но система посчитала их использование корректным, по отношению к общему объему документа. Сюда относятся оформленные по ГОСТу цитаты; общеупотребительные выражения; фрагменты текста, найденные в источниках из коллекций нормативно-правовой документации.
Текстовое пересечение — фрагмент текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника.
Источник — документ, проиндексированный в системе и содержащийся в модуле поиска, по которому проводится проверка.
Оригинальность — доля фрагментов текста проверяемого документа, не обнаруженных ни в одном источнике, по которым шла проверка, по отношению к общему объему документа.
Заимствования, самоцитирования, цитирования и оригинальность являются отдельными показателями и в сумме дают 100%, что соответствует всему тексту проверяемого документа.
Обращаем Ваше внимание, что система находит текстовые пересечения проверяемого документа с проиндексированными в системе текстовыми источниками. При этом система является вспомогательным инструментом, определение корректности и правомерности заимствований или цитирований, а также авторства текстовых фрагментов проверяемого документа остается в компетенции проверяющего.

№	Доля в отчете	Источник	Актуален на	Модуль поиска	Комментарии
[01]	0,67%	http://vital.lib.tsu.ru/vital/access/services/Download/vital:6890/SOURCE01 http://vital.lib.tsu.ru	07 Сен 2020	Интернет Free	
[02]	0,27%	http://vital.lib.tsu.ru/vital/access/services/Download/vital:8382/SOURCE01 http://vital.lib.tsu.ru	24 Янв 2020	Интернет Free	
[03]	0%	http://vital.lib.tsu.ru/vital/access/services/Download/vital:11008/SOURCE01 http://vital.lib.tsu.ru	24 Янв 2020	Интернет Free	

Еще источников: 4
Еще заимствований: 0,53%