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
« 23 » января 2023 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА СПЕЦИАЛИСТА (ДИПЛОМНАЯ РАБОТА)

АНАЛИЗ МЕХАНИЗМОВ ЗАЩИТЫ ЛИЦЕНЗИРОВАНИЯ ТЕЛЕКОММУНИКАЦИОННЫХ УСТРОЙСТВ

по основной образовательной программе подготовки специалистов
«Анализ безопасности компьютерных систем»
специальности 10.05.01 Компьютерная безопасность

Раимбекову Азиму Руслановичу

Руководитель
канд. физ. мат. наук, доцент

_____ Д.Н.Колегов
« 23 » января 2023 г.

Автор работы
студент группы № 931724

_____ А.Р.Раимбеков
« 23 » января 2023 г.

Томск – 2023

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
« 27 » июня 2022 г.

ЗАДАНИЕ
по выполнению выпускной квалификационной работы специалиста обучающемуся
Раимбекову Азиму Руслановичу
по специальности 10.05.01 Компьютерная безопасность, специализация (профиль) «Анализ безопасности компьютерных систем»

1 Тема дипломной работы

Анализ механизмов защиты лицензирования телекоммуникационных устройств

2 Срок сдачи студентом выполненной дипломной работы:
а) в учебный офис / деканат – 16.01.2023 б) в ГЭК – 27.01.2023

3 Исходные данные к работе:
Объект исследования – Механизмы лицензирования
Предмет исследования – Механизмы лицензирования Cisco PLR, OPLR
Цель исследования – Обеспечение безопасности механизмов лицензирования телекоммуникационных устройств

Задачи:
Выбрать современные механизмы лицензирования, определить недостатки и уязвимости в безопасности выбранных механизмах лицензирования. Разработать прототипы выбранных механизмов лицензирования. Предложить методы защиты от найденных недостатков и уязвимостей в выбранных механизмах лицензирования.

Методы исследования:
Анализ уязвимостей механизмов лицензирования телекоммуникационных устройств

Организация или отрасль, по тематике которой выполняется работа, –
Лаборатория компьютерной криптографии ТГУ.

4 Краткое содержание работы
Анализ механизмов защиты лицензирования телекоммуникационных устройств. Уязвимости и недостатки механизмов лицензирования Cisco PLR, OPLR.

Руководитель ВКР
канд. физ. мат. наук, доцент _____ / Д.Н.Колегов/
Задание принял к исполнению
студент группы № 931724 _____ / А.Р.Раимбеков/

АННОТАЦИЯ

Дипломная работа содержит 35 страниц, 7 литературных источников, 14 рисунков.

МЕХАНИЗМЫ ЛИЦЕНЗИРОВАНИЯ, МЕТОДЫ ЗАЩИТЫ МЕХАНИЗМОВ ЛИЦЕНЗИРОВАНИЯ

Объекты исследования: механизмы лицензирования телекоммуникационных устройств Cisco PLR, OPLR.

Цель работы: Обеспечение безопасности механизмов лицензирования телекоммуникационных устройств.

Метод исследования: анализ уязвимостей механизмов лицензирования телекоммуникационных устройств.

Результат: выявлены уязвимости механизмов лицензирования телекоммуникационных устройств и предложен метод защиты.

ОГЛАВЛЕНИЕ

ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ 5

ВВЕДЕНИЕ 6

1 Постановка задачи 7

2 Обзор современных систем лицензирования 8

2.1 Недостатки защищенности систем лицензирования 8

3 Описание уязвимостей современных систем лицензирования 11

3.1 Схема анализа систем лицензирования 12

3.2 Анализ системы лицензирования Cisco PLR 14

3.3 Анализ системы лицензирования Cisco OPLR 21

4 Методы защиты систем лицензирования Cisco PLR/OPLR 29

4.1 Общее описание методов защиты 29

ЗАКЛЮЧЕНИЕ 34

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ 35

ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ

Компиляция (compilation) — это трансляция программы, написанной на исходном языке высокого уровня, в эквивалентную программу на низкоуровневом языке, близком машинному коду.

Уязвимость — недостаток программного средства или информационной системы в целом, который может быть использован для реализации угроз безопасности информации

Угроза безопасности информации — совокупность условий и факторов, создающих потенциальную или реальную существенную опасность нарушения безопасности информации.

Уязвимость кода — уязвимость, появившаяся в процессе разработки программного обеспечения.

Объектный файл (object file) — это созданный ассемблером промежуточный файл, хранящий часть машинного кода.

Ассемблерный код (assembly code) — это доступное для понимания человеком представление машинного кода.

Обфускация (obfuscation) — процесс запутывания программы, позволяющий видоизменить исходный и байт-код таким образом, при котором пользовательский интерфейс и поведение программы останутся неизменными.

Гипервизор (hypervisor) – это программа, которая управляет физическими ресурсами вычислительной машины и распределяет эти ресурсы между несколькими различными операционными системами, позволяя запускать их одновременно.

Маршрутизатор (router) – телекоммуникационное устройство, которое соединяет вычислительные устройства и сети с другими сетями.

Коммутатор (switch) – телекоммуникационное устройство, предназначенное для соединения нескольких узлов компьютерной сети в пределах одного сегмента сети.

Межсетевой экран (firewall) – программный или программно-аппаратный элемент компьютерной сети, осуществляющий контроль и фильтрацию проходящего через него сетевого трафика в соответствии с заданными правилами.

ВВЕДЕНИЕ

В настоящее время существует множество способов лицензирования программного обеспечения телекоммуникационного оборудования и каждая компания делает это по-разному, но все преследуют одну цель - контроль проприетарного программного обеспечения. Поставленная в данной работе цель актуальна, поскольку одной из целей любого проприетарного проекта является лицензировать и контролировать распространение программного обеспечения телекоммуникационного устройства. На практике мы встречаемся с тем, что большинство программных продуктов, требующих наличие лицензии подвергаются компрометации и в последующем используются нелегитимно.

Данная работа заключает в себе методы защиты механизма лицензирования телекоммуникационного устройства и анализ безопасности механизмов лицензирования PLR, OPLR компании Cisco. Следует понимать, что любая программа может быть в конечном итоге скомпрометирована, а абсолютной защиты нет. Способом защиты является усложнение метода анализа программного обеспечения, чтобы злоумышленник потратил больше времени, чем время между обновлениями программы.

Телекоммуникационным устройством в данной работе будут выступать маршрутизаторы и коммутаторы компании Cisco.

Результат данной работы - проделан анализ систем лицензирования и найдены уязвимости методов защиты систем лицензирования, а также предложены методы защиты.

В первой главе данной работы описывается предметная область механизмов лицензирования

Вторая глава рассматривает практическую составляющую, в которой показаны как и где уязвимы современные системы лицензирования.

Третья глава описывает методы защиты механизмов лицензирования телекоммуникационного устройства

1 Постановка задачи

На данный момент существует множество способов защиты систем лицензирования программного обеспечения телекоммуникационного устройства и каждая по-своему уникальна. Наша задача будет состоять в анализе безопасности систем лицензирования и предложении способов защиты от найденных недостатков безопасности механизма лицензирования телекоммуникационного оборудования. Для анализа мы выберем несколько механизмов лицензирования и предложим методы защиты, которые позволят защитить механизм лицензирования от анализа.

Целью работы является обеспечение безопасности механизмов лицензирования телекоммуникационных устройств

Для достижения заданной цели необходимо решить следующие **задачи**:

— сделать выбор современных механизмов лицензирования для дальнейшего анализа;

— провести исследование выбранных систем лицензирования;

— предложить эффективные методы защиты от анализа злоумышленником.

2 Обзор современных систем лицензирования

Лицензия на программное обеспечение представляет собой правовой инструмент, с помощью которого определяются способы его использования и распространения. Программно аппаратное лицензирование состоит из двух критериев временного и функционального ограничения:

— временное ограничение включает в себя сроки эксплуатации лицензированного телекоммуникационного устройства;

— телекоммуникационное устройство может содержать как программно так и аппаратно модули улучшающие характеристики устройства, такого рода ограничения называют функциональным.

Компании распространяющие лицензионное программное обеспечение также могут предоставлять бессрочное и полностью снимающее функциональное ограничение лицензию. Механизм лицензирования PLR, описанные далее, как раз относиться к данной категории лицензий раскрывающий весь функционал телекоммуникационного устройства.

Лицензирование позволяет добиться максимальной окупаемости инвестиций компаний и зачастую цена лицензий может быть дороже цены программного продукта. Контроль лицензий программного продукта осуществляется механизмом лицензирования. Каждая компания выделяет ресурсы на разработку стойкого от анализа злоумышленника механизма лицензирования. Следовательно, компрометация механизма лицензирования должна быть невозможной или сложно решаемой задачей, что опровергается в рамках данной работы.

2.1 Недостатки защищенности систем лицензирования

Лицензирование телекоммуникационного устройства предполагает подключение в общедоступную сеть интернет для выдачи ключа лицензирования от сервера вендора, что усложняет анализ механизма, так как проверка ключа происходит на стороне сервера вендора. Но с развитием телекоммуникационных сетей крупные компании содержат сегменты сети, которые не подключены в общедоступную сеть интернет, что требует

другого подхода лицензирования. Вендором была поставлена задача разработать механизмы лицензирования, позволяющие пролицензировать без подключения программного устройства в интернет. Такой подход подразумевает два исхода событий:

— механизм лицензирования будет в самом устройстве, что является потенциальной угрозой;

— механизм лицензирования будет на стороне сервера компании, которую можно будет получить администратору сетевой инфраструктуры после авторизации.

В данной работе будут рассмотрены первое описание, когда механизм лицензирования в телекоммуникационном устройстве.

В большинстве случаев при проектировании программного обеспечения, содержащего в себе механизм лицензирования, используют общепринятые методы защиты. Зачастую такой защиты не хватает и требуется дополнительные или модифицированные методы защиты. Но не стоит забывать, что каждая компания заинтересована быстро и качественно разработать программное обеспечение и заработать на этом. Элемент быстроты присутствует, но качественность, увы, всегда дает о себе знать. В данном контексте качество означает качество написанного кода программного продукта, а также такие составляющие как механизм лицензирования. Для хорошего качества программного продукта требуется больше времени и ресурсов, на которые компании не готовы идти. Стоит определить что входит в множество недостатков защищенности систем лицензирования:

— пренебрежение общепринятыми методами защиты программного обеспечения;

— использование, в программном обеспечении, не стойкого или не правильно использованного метода защиты программного обеспечения;

— использование, в программном обеспечении, не стойкого или не правильно использованного криптографического протокола, шифра.

Недостаток защищенности систем лицензирования зачастую упирается не в стойкости криптографических механизмов, а в неправильном использовании или разработки защиты систем лицензирования.

Механизмы лицензирования, описанные в данной работе являются хорошим примером неправильной разработки безопасности систем лицензирования телекоммуникационного устройства. Основные ошибки, допускаемые при разработке и проектирова-

нии механизма лицензирования, описаны в следующей главе.

Вывод. В данной главе описаны то, как работают современные методы лицензирования и краткое описание недостатков методов лицензирования.

3 Описание уязвимостей современных систем лицензирования

Выбор систем лицензирования осуществлялся по критериям общедоступности и актуальности. В данной работе будут исследованы современные механизмы лицензирования телекоммуникационных устройств компании Cisco, так как наткнулся на статью « Cracking Cisco's Sourcefire licensing system » [1], в которой показаны уязвимость системы лицензирования программного средства *IDS/IPS*.

Стоит отметить, что данный механизм лицензирования устарел и выбор состоял из современных механизмов лицензирования, таких как Cisco PLR и OPLR. Объектом исследования будет файл прошивки программного продукта, содержащий в себе механизм лицензирования. Целью компании является защита от анализа исходного кода программы механизма лицензирования.

При исследовании систем лицензирования можно обнаружить общепринятые конструкции программирования, которые потенциально упрощают анализ исходного кода механизма лицензирования. Давайте отметим несколько из них:

— **логирование** это хронология событий и их источников, ошибок и причин, которые в свою очередь помогают разработчикам и конечным пользователям в понимании работоспособности программы. Следующие строки кода на рисунке 1 представлены из прошивки исследуемого телекоммуникационного устройства;

```
public String generateReturnCode(String authCode) {
    log.debug("generateUniversalCode: auth code: " + authCode);
    if (authCode == null) {
        return null;
    }
    boolean validSpecificAuthCode = validateSpecificAuthorizationCode(authCode);
    log.debug("validSpecificAuthCode: " + validSpecificAuthCode);
    if (validSpecificAuthCode) {
        return generateSpecifiReturnCode(authCode);
    }
    boolean validUniversalAuthCode = validateUniversalAuthorizationCode(authCode);
    log.debug("validUniversalAuthCode: " + validUniversalAuthCode);
    if (validUniversalAuthCode) {
        return generateUniversalReturnCode(authCode);
    }
    return null;
}
```

Рисунок 1 — Пример логирования в прошивке исследуемого маршрутизатора на языке Java

Следовательно, исследуя файл прошивки телекоммуникационного устройства можно понять по семантике логирования какая примерно функция находится рядом со строкой вывода лога.

— **комментарии** содержат в себе исчерпывающую информацию о том, что происходит в том или ином участке кода программы, следовательно, упрощая процесс исследования программного обеспечения. Способ определений жестко заданных константных строк в программе тоже может упростить анализ методом обратной разработки. На следующем рисунке 2 изображены строки кода с комментариями исследуемого телекоммуникационного устройства;

```
public enum SyslogCode {
    AGENT_READY(IPlatformIndependent.SysLogSeverity.INFO, "Smart Agent is initialized"),
    AGENT_ENABLED(IPlatformIndependent.SysLogSeverity.INFO, "Smart Agent is enabled "),
    SYSTEM_CLOCK_CHANGED(IPlatformIndependent.SysLogSeverity.NOTICE, "Smart Agent System clock has been changed"),
    AGENT_REG_FAILED(IPlatformIndependent.SysLogSeverity.ERR, "Smart Agent Registration with Cisco licensing cloud failed:
    AGENT_REG_SUCCESS(IPlatformIndependent.SysLogSeverity.INFO, "Smart Agent Registration with Cisco licensing cloud succes
    AGENT_DEREG_SUCCESS(IPlatformIndependent.SysLogSeverity.INFO, "Smart Agent De-registration with Cisco licensing cloud
    AGENT_DEREG_FAILED(IPlatformIndependent.SysLogSeverity.ERR, "Smart Agent DeRegistration with CSSM failed: %s"),
    IN_COMPLIANCE(IPlatformIndependent.SysLogSeverity.NOTICE, "All entitlements are authorized"),
    OUT_OF_COMPLIANCE(IPlatformIndependent.SysLogSeverity.ERR, "One or more entitlements are out of compliance"),
    IN_OVERAGE(IPlatformIndependent.SysLogSeverity.WARNING, "One or more entitlements are in overage"),
    EVAL_START(IPlatformIndependent.SysLogSeverity.NOTICE, "Entering evaluation period"),
    EVAL_EXPIRED(IPlatformIndependent.SysLogSeverity.ERR, "Evaluation period expired"),
    AUTHORIZATION_EXPIRED(IPlatformIndependent.SysLogSeverity.NOTICE, "Authorization expired"),
    DISABLED(IPlatformIndependent.SysLogSeverity.INFO, "Smart Agent disabled"),
    BAD_MODE(IPlatformIndependent.SysLogSeverity.ERR, "An unknown mode was specified: %d"),
    BAD_NOTIF(IPlatformIndependent.SysLogSeverity.ERR, "A bad notification type was specified: %d"),
```

Рисунок 2 — Пример комментирования в прошивке исследуемого маршрутизатора на языке Java

— **общепринятые названия функций и переменных** упрощают разработку программного обеспечения, так как исходный код легче читать и поддерживать с другими членами команды программистов. Но с другой стороны упрощает анализ кода злоумышленником.

Все описанные выше и остальные уязвимости механизмов будут выявлены и показаны далее на выбранных телекоммуникационных устройствах.

3.1 Схема анализа систем лицензирования

Проведя исследования нескольких механизмов лицензирования стоит выделить общую схему анализа систем лицензирования, который также будет применяться в следующих разделах. Схема будет состоять из последовательных шагов от поиска телекоммуникационного устройства до нахождения в нем исходного кода механизма лицензирования. Но стоит отметить, что данная схема будет изменяться и адаптироваться под

телекоммуникационное устройство в силу различия архитектуры программного продукта и языка программирования.

Схема анализа систем лицензирования:

— **поиск телекоммуникационного устройства;**

Поиск программного продукта, содержащего в себе механизм лицензирования осуществляется двумя способами: общедоступным в интернете либо покупкой программного устройства (в моем случае это виртуальный образ программного устройства, который общедоступен);

— **установка;**

После удачного поиска стоит задача установки и запуска виртуального устройства, который возможно сделать следующими программами VMware, Oracle virtualbox;

— **файлы программ прошивки;**

Существует множество способов добраться до исходных программных файлов телекоммуникационного устройства, один из которых заключается в использовании возможности виртуализации, что изображено на рисунке 3;

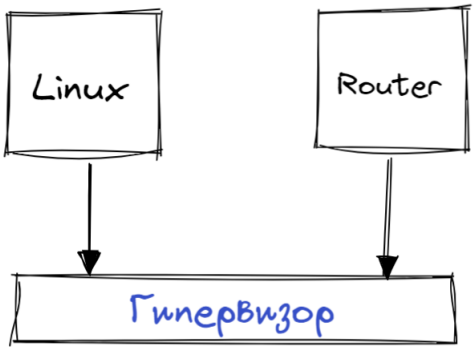


Рисунок 3 — Гипервизор

Таким образом, была создана виртуальная машина на базе linux, к которой было подключено всё дисковое пространство устройства и получен доступ к исходным файлам программ.

— **поиск механизма лицензирования;**

Данный шаг схемы всегда будет зависеть от архитектуры телекоммуникационного устройства, но стоит отметить, что аналитик может воспользоваться

поисковыми программами и сократить время поиска нежели делать это вручную;

— **анализ механизма лицензирования;**

После удачного поиска предстоит анализ механизма лицензирования, суть данного шага состоит в том, чтобы разобрать и понять идею механизма лицензирования;

— **тестирование.**

Данный этап включает в себя задачу разработки прототипа генератора ключей лицензий, усложнение на этом этапе это зависимости механизма лицензирования от остального программного обеспечения телекоммуникационного устройства.

Выведенная схема анализа может быть применена для исследования любых механизмов лицензирования находящихся в телекоммуникационном устройстве. Описанная выше схема будет показана, на реальных примерах механизмов лицензирования, в следующих разделах.

3.2 Анализ системы лицензирования Cisco PLR

PLR - запрос на постоянную лицензию (англ. permanent license request) отличается от остальных видов лицензий ознакомиться с которыми можно перейдя по следующей ссылке «Cisco Licenses» [2]. Основным различием является то что PLR снимает все ограничения на телекоммуникационном устройстве и является бессрочной. Данный механизм лицензирования содержится в маршрутизаторах:

— cisco isr 4000 series;

— cisco catalyst 3000 series;

— cisco catalyst 9000 series;

— cisco asr series;

— cisco crs1000v;

— cisco nexus series.

Схему анализа механизма PLR, которую мы будем использовать, была описана в предыдущем разделе.

1. Поиск телекоммуникационного устройства.

Посетив сайт вендора можно найти документацию по телекоммуникационным устройствам, содержащие механизм PLR. В нашем примере мы рассматриваем маршрутизатор линейки Nexus, в которые начали внедрять механизм лицензирования PLR, виртуальный образ которого можно найти в интернете в свободном доступе.

2. Установка.

В качестве виртуализации выбрана программа VMware Workstation (но аналогично можно сделать и в программе Oracle virtualbox). Создаем две виртуальные машины.

— первая виртуальная машина - linux Ubuntu

— вторая виртуальная машина - образ маршрутизатора

3. Файлы программ прошивки.

Виртуализация VMware позволяет связать по сети две созданные нами виртуальные машины. Зайдя в настройки виртуальной машины linux меняем настройки диска, подключаем диск виртуальной машины маршрутизатора. Тем самым использовав данную функцию на виртуальной машине linux мы можем взаимодействовать с файлами прошивки маршрутизатора.

Достав файлы прошивки, определяем язык программирования, в нашем случае это Java — мультиплатформенный язык программирования. Это значит, что программы, написанные на языке Java, можно выполнять на любой платформе, где установлена специальная исполняющая система Java. Такая система называется Java Virtual Machine (JVM). Для того, чтобы перевести программу из исходного кода в код, понятный JVM, нужно её скомпилировать. Код, понятный JVM называется байт-кодом и содержит набор инструкций, которые в дальнейшем будет исполнять виртуальная машина.

Для компиляции исходного кода в байт-код существует компилятор javac, входящий в поставку JDK (Java Development Kit). На вход компилятор принимает файл с расширением .java, содержащий исходный код программы, а на выходе выдает файл с расширением .class, содержащий байт-код, необходимый для исполнения программы виртуальной машиной, что изображено на следующем рисунке.

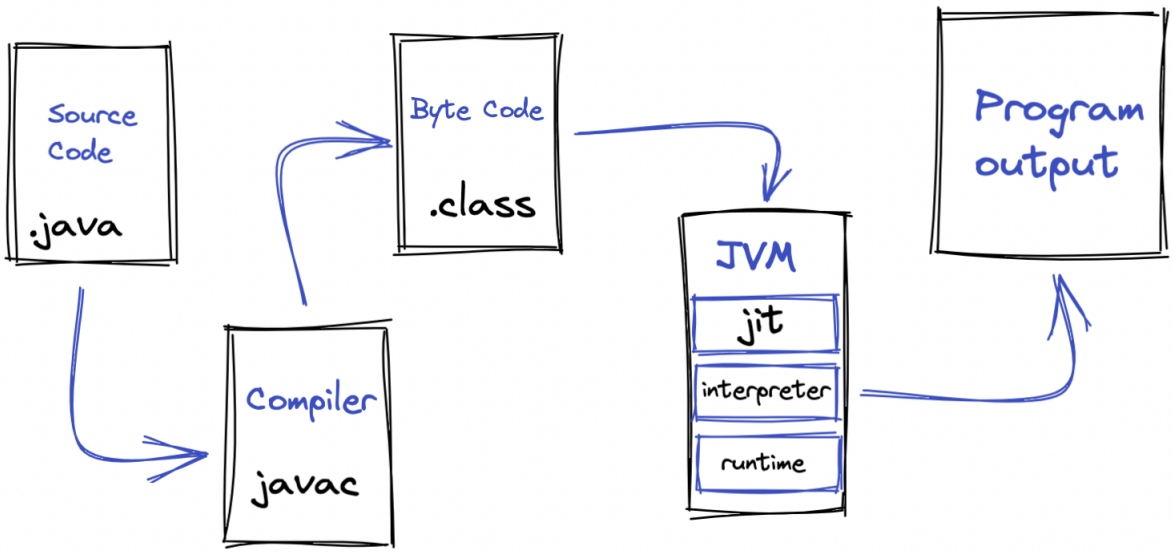


Рисунок 3 — Компиляция на языке Java

После того, как программа была скомпилирована в байт-код, она может быть выполнена с помощью виртуальной машины. Структура прошивки состоит из более тысячи файлов программ между которыми большое количество связи, что усложняет запуск отдельной программы вне прошивки маршрутизатора.

4. Поиск механизма лицензирования.

Описав уязвимости в первой главе, нашей задачей будет поиск строк логирования, комментария и общепринятых названий функций. Задача состоит в поиске строки в файлах программ, но осуществляема она только при знании определенной строки, которая каким-то образом связана с механизмом лицензирования.

Следующий шаг в достижении цели является изучение документации маршрутизатора или то, как можно ввести ключ активации в маршрутизатор. Все команды описаны по следующей ссылке «Smart Licensing PLR » [3]. Войдя в командную строку маршрутизатора нужно ввести следующие строки:

— license smart reservation;

— license smart reservation request;

— License smart reservation install;

— Reservation install successfully;

— Reservation install failed;

— PLR.

Таких строк очень много, поэтому выделенные строки считаются основными. Утилиты, такие как grep и the silver searcher, позволяют совершать поиск строки в файлах программ. Таким способом удастся сделать выборку файлов по включающимся в них строках, что упрощает поиск аналитику. Просмотрев все файлы с вхождениями строк можно найти сам механизм лицензирования. В нашем случае это оказался один .jar файл, в котором была найдена функция лицензирования PLR.

5. Анализ механизма лицензирования.

Найдя исходный код механизма лицензирования PLR стоит изучить и понять идею генерации и проверки ключа активации. На рисунке 4 изображены строки кода PLR.

```
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;
import javax.xml.bind.Unmarshaller;
import org.apache.commons.codec.digest.DigestUtils;
import org.apache.commons.io.FileUtils;
import org.apache.commons.lang3.ArrayUtils;
import org.apache.commons.lang3.StringUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.xml.sax.InputSource;


public class PLR {
    private static final Logger log = LoggerFactory.getLogger(PLR.class.getName());
    private SmartAgentRequestHandler smartAgentRequestHandler;
    private IPlatformDependent pdPlugin;
    private ICrypto cryptoPlugin;
    private DataStoreManager dsManager;
    private String tenantContextId;
    private EventLogManager eventLogManager;

    public PLR(SmartAgentRequestHandler smartAgentRequestHandler, IPlatformDependent pdPlugin, ICrypto cryptoPlugin) {
        this.smartAgentRequestHandler = smartAgentRequestHandler;
        this.pdPlugin = pdPlugin;
        this.cryptoPlugin = cryptoPlugin;
        this.dsManager = smartAgentRequestHandler.getDSManager();
        this.tenantContextId = smartAgentRequestHandler.getTenantContextId();
        this.eventLogManager = smartAgentRequestHandler.getEventLogManager();
    }
```

Рисунок 4 — Исходный код механизма лицензирования PLR, класс PLR

Для начала стоит ознакомится с методом ввода ключа от лица конечного потребителя. Первоначально клиент в командной строке маршрутизатора вводит строки

Device(config)#license smart enable

Device(config)#license smart reservation

Device(config)#license smart reservation request local

Результат которых будет 32-битная строка. Алгоритм генерации запроса, изображен на рисунке 5, выглядит следующим образом.

```
public String generateUniversalRequestCode() {
    String verStr = getVersionString();
    log.debug("PLR version string: " + verStr);
    byte newSequence = incrementSequence((byte) this.smartAgentRequestHandler.getTechSupportInfo().getTestInfo().getPlrSequence())
    this.smartAgentRequestHandler.getTechSupportInfo().getTestInfo().setPlrSequence(newSequence);
    String seqStr = getSequenceString();
    log.debug("PLR sequence string: " + seqStr);
    String udiStr = getUDIString();
    log.debug("PLR udi string: " + udiStr);
    String swIDStr = getSoftwareIDTagString(this.pdPlugin.getSoftwareID());
    log.debug("PLR  swID string: " + swIDStr);
    String preHashStr = verStr + seqStr + udiStr + swIDStr;
    String hash = DigestUtils.md5Hex(preHashStr);
    String shortHash = hash.substring(hash.length() - 2).toUpperCase();
    String finalStr = verStr + seqStr + "-" + udiStr + "-" + swIDStr + "-" + shortHash;
    log.debug("PLR UniversalRequestCode: " + finalStr);
    this.smartAgentRequestHandler.getTechSupportInfo().getTestInfo().setPlrRequestCode(finalStr);
    this.smartAgentRequestHandler.getTechSupportInfo().getTestInfo().setPlrStatus(ISmartAgentPLR.PLRStatus.IN_PROGRESS);
    this.smartAgentRequestHandler.getTechSupportInfo().getTestInfo().setPlrAuthorizationCode(null);
    this.smartAgentRequestHandler.getTechSupportInfo().getTestInfo().setPlrInProgressStartTime(Calendar.getInstance().getTime());
    this.smartAgentRequestHandler.persistAgentInfo(this.smartAgentRequestHandler.getTechSupportInfo().getTestInfo());
    GlobalNotification notification = new GlobalNotification(SmartAgentNotification.NotificationType.NotifyReservationInProgress, 
    this.smartAgentRequestHandler.getAgentListener().onNotify(notification);
    this.smartAgentRequestHandler.sendSyslog(SyslogCode.RESERVE_IN_PROGRESS, finalStr, getUDIString());
    return finalStr;
}
```

Рисунок 5 — Функция генерации запроса PLR

Далее клиент регистрируется на портале вендора и вводит данную строку из 32 бит, узнает стоимость лицензии и только после оплаты возвращается ключ активации в виде строки символов. Данный ключ нужно ввести в маршрутизатор следующей командой

Device(config)# license smart reservation install [Response Code]

где [Response Code] - ключ активации, который вводится в маршрутизатор и пройдя все проверки снимает все ограничения на маршрутизаторе. Исходный код данной проверки и генерации ключа активации изображен на рисунке 6.

```java
public boolean validateUniversalAuthorizationCode(String authCode) {
    HashMap<String, String> addlData = new HashMap<>();
    addlData.put(Constants.UDI, this.eventLogManager.getUDI());
    log.debug("authCode: " + authCode);
    if (authCode == null || authCode.isEmpty()) {
        this.eventLogManager.saveEventLog(SmartAgentEventLog.SAEVT_RESERVE_VALIDATION_FAIL, addlData, this.smartAgentRequestHandler
        log.debug("validateUniversalAuthorizationCode error: authCode is empty");
        return false;
    }
    String verStr = authCode.substring(0, 1);
    log.debug("verStr: " + verStr);
    String flagStr = authCode.substring(1, 2);
    log.debug("flagStr: " + flagStr);
    String piidStr = getAuthCodePIIDString(authCode);
    log.debug("piidStr: " + piidStr);
    String sigStrDash = authCode.substring(6);
    log.debug("sigStr with -: " + sigStrDash);
    if (sigStrDash == null || sigStrDash.length() == 0) {
        this.eventLogManager.saveEventLog(SmartAgentEventLog.SAEVT_RESERVE_VALIDATION_FAIL, addlData, this.smartAgentRequestHandler
        log.debug("signature is null or empty");
        return false;
    }
    String sigStr = sigStrDash.replaceAll("-", "");
    String sigData1 = "export:true" + getSoftwareIDTagString(this.pdPlugin.getSoftwareID()) + getUDIString() + getSequenceString() +
    log.debug("sigData1: " + sigData1);
    String sigData2 = "export:false" + getSoftwareIDTagString(this.pdPlugin.getSoftwareID()) + getUDIString() + getSequenceString()
    log.debug("sigData2: " + sigData2);
    boolean valid = false;
    String hmac1 = Utility.calculateHMAC(sigData1, this.cryptoPlugin);
    String hmac2 = Utility.calculateHMAC(sigData2, this.cryptoPlugin);
    log.debug("hmac1:           " + hmac1);
    log.debug("hmac2:           " + hmac2);
    log.debug("expected sigStr: " + sigStr);
```

Рисунок 6 - Функция обработки запроса и генерации ключа PLR

6. Тестирование.

После нахождения механизма лицензирования, стоит задача разработки прототипа генератора ключа и тестирования на исследуемом телекоммуникационном устройстве. В итоге аналитик оперирует двумя основными функциями из выше представленных рисунков 5 и 6. На рисунке 5 представлена генерация 32-битной строки которая в итоге будет использоваться в функции отображенной на рисунке 6. Давайте обратив внимание на следующие важные строки

```
0 String verStr = getVersionString();
1 String seqStr = getSequenceString();
2 log.debug("PLR sequence string: "+ seqStr);
3 String udiStr = getUDIString();
4 log.debug("PLR udi string: "+ udiStr);
5 String swIDStr = getSoftwareIDTagString(this.pdPlugin.getSoftwareID());
6 log.debug("PLR swID string: "+ swIDStr);
7 String preHashStr = verStr + seqStr + udiStr + swIDStr;
8 String hash = DigestUtils.md5Hex(preHashStr);
9 String shortHash = hash.substring(hash.length() - 2).toUpperCase();
```

Строки с началом log.debug означают вывод в режим отладки. Интересующие строки кода все остальные, давайте обратим внимание на последнюю строку в которой строковая переменная состоящая из конкатенации нескольких перемен. К примеру в переменной **verStr** содержится результат функции **getVersionString**. Найдя данную функцию можно понять идею, забегая вперед название переменной говорит нам это version string, что проясняется точно на следующем рисунке 7

```
public String getVersionString() {
    byte[] inByte = {Constants.PLR_VERSION};
    String versionStr = Base58.doEncode(inByte);
    log.debug("version " + ((int) Constants.PLR_VERSION) + " encoded to " + versionStr);
    return versionStr;
}
```

Рисунок 7 — Функция генерации версии

Понять составляющую остальных переменных, конкатенация которых образуют, переменную **finalStr**, можно так же. Далее данная строка поступает на генерацию ключа лицензирования, изображенного на рисунке 6, проходя внутренние проверки результатом которой является ключ активации.

Забегая вперед хочу выделить то что основные составляющие генерации строяться на основных характеристиках телекомууникацинного устройсва. Что является слабой составляющей механизма лицензирования, так как данные характеристики можно найти в документации, через логи или в командной строке самого устройства.

Результат анализа механизма - написан прототип генератора ключей активации на языке Java. Критерий выбора языка основывались на языке используемом в прошивке устройства, так как в алгоритме использовались стандартные библиотеки. Основные сложности, которые возникли при анализе это поиск, это запуск механизма лицензирования, так как механизм содержит в себе более ста ссылок в другие файлы прошивки, что требует больше времени для копирования и воссоздания самого алгоритма лицензирования вне маршрутизатора. В итоге прототип генератора можно использовать и для других телекоммуникационных устройств содержащих данный механизм, вследствие нескольких факторов разработки механизма лицензирования которые описаны выше.

Недостатки защиты метода лицензирования:

— незашифрованный диск маршрутизатора, что позволяло свободно обозревать исходный код;

— отсутствие каких-либо методов защиты исходного кода;

— использование общепринятых библиотек программирования.

3.3 Анализ системы лицензирования Cisco OPLR

OPLR - вид лицензий позволяющий раскрыть криптографический модуль телекоммуникационного устройства, который доступен после лицензирования маршрутизатора. Модели маршрутизаторов и межсетевых экранов использующие механизм лицензирования OPLR:

— cisco C3560 Series;

— cisco Nexus Series;

— cisco Asa 5500.

Активация лицензий на телекоммуникационном устройстве включает в себя пропускную способность и количество зашифрованных каналов, создаваемых на телекоммуникационном устройстве. Схему анализа механизма лицензирования, которую мы будем использовать, была описана в предыдущих разделах.

1. Поиск телекоммуникационного устройства.

Посетив сайт вендора можно найти документацию по телекоммуникационным устройствам, содержащие механизм OPLR. Данный криптографический модуль содержится в следующих устройствах «Cisco Smart Licensing OPLR» [4] и называется HSEC (англ. high security) В нашем примере мы рассматриваем маршрутизатор линейки Nexus, виртуальный образ которого можно найти в интернете в свободном доступе.

2. Установка.

В качестве виртуализации выбрана программа VMware Workstation (но аналогично можно сделать и в программе Oracle virtualbox). Создаем две виртуальные машины:

— первая виртуальная машина - linux Ubuntu;

— вторая виртуальная машина - образ маршрутизатора.

3. Файлы программ прошивки.

Виртуализация VMware позволяет связать по сети две созданные нами виртуальные машины. Зайдя в настройки виртуальной машины linux меняем настройки диска, подключаем диск виртуальной машины маршрутизатора. Тем самым использовав данную функцию на виртуальной машине linux мы можем взаимодействовать с файлами прошивки маршрутизатора.

В итоге данный маршрутизатор содержит один большой бинарный файл прошивки.

4. Поиск механизма лицензирования.

На поиск механизма лицензированная OPLR потребовалось больше времени, в отличие от предыдущего механизма лицензирования PLR, так как программное обеспечение исследуемого телекоммуникационного устройства написана на C++. Аналитику придется работать с бинарным файлом, что значительно усложняет задачу. Для дальнейшего анализа исходного кода программы стоит разобраться в процессе компиляции программы на языке C++, что изображено на следующем рисунке 8.

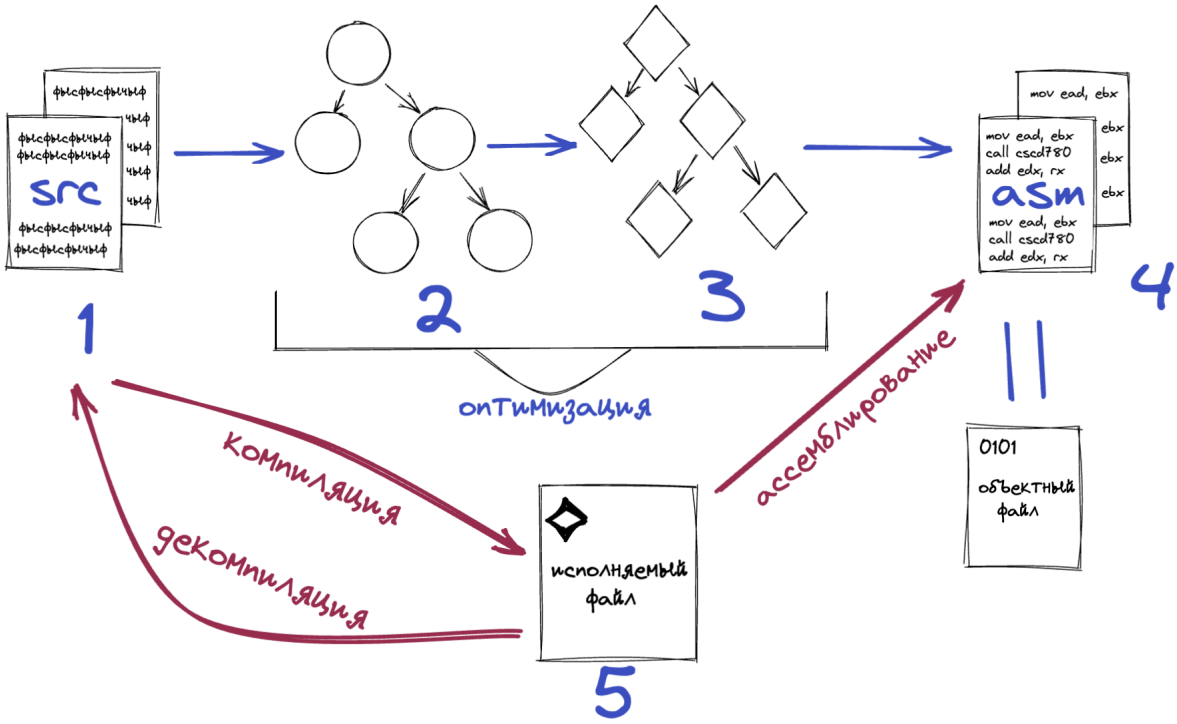


Рисунок 8 — Процесс компиляции программы

1. **Исходный код программы.**

Это текст программы, написанный на определенном языке программирования, который будет подвергаться компиляции.

2. **Лексер.**

На этом этапе срабатывает лексер и стадии лексического анализа это получить на вход много букв и сгруппировать их по каким-то категориям — «токенам». Дальше парсер эту последовательность токенов анализирует и представляет в виде синтаксической конструкции. В результате работы парсера получается конкретное синтаксическое дерево. Лексический анализ ещё называют «токенизацией». Это самая первая стадия обработки текста, которую производит каждый существующий компилятор.

3. **Внутреннее представление (intermediate language).**

Некоторые современные языки программирования на следующей стадии строят внутреннее представление для упрощения дальнейшей работы.

4. **Объектный файл.**

Строится набор ассемблерных файлов, объектный файл.

5. **Исполняемый файл.**

Из объектного файла мы получаем исполняемый файл (иначе говоря бинарный), скомпилированный в двоичном виде

Объектом анализа будет исполняемый файл, так как только эта составляющая всего процесса компиляции доступна всем. Для того, чтобы исследовать бинарный файл нужно будет использовать специальные программы и обладать некоторыми техниками которые описываются далее.

Существует два основных подхода к анализу: статистический и динамический. При статистическом анализе программу изучают не запуская её, а при динамическом производят запуски программы и манипуляции с запущенным процессом в оперативной памяти. Забегая вперед мы будет рассматривать статистический анализ.

Для исследования исполняемого файла мы будем использовать специальные программы позволяющее из исполняемого файла получить ассемблерный файл. В данной работе использовалась программа IDA, но так же стоит отметить аналог такой как Hydra. Сложность поиска состоит в предстоящем нахождении механизма лицензирования в ассемблерном коде, где нет явных названий функций и переменных, но есть строго заданные строки константы, по которым можно сделать поиск. На следующем рисунке 9 сделан поиск по строке HSEC

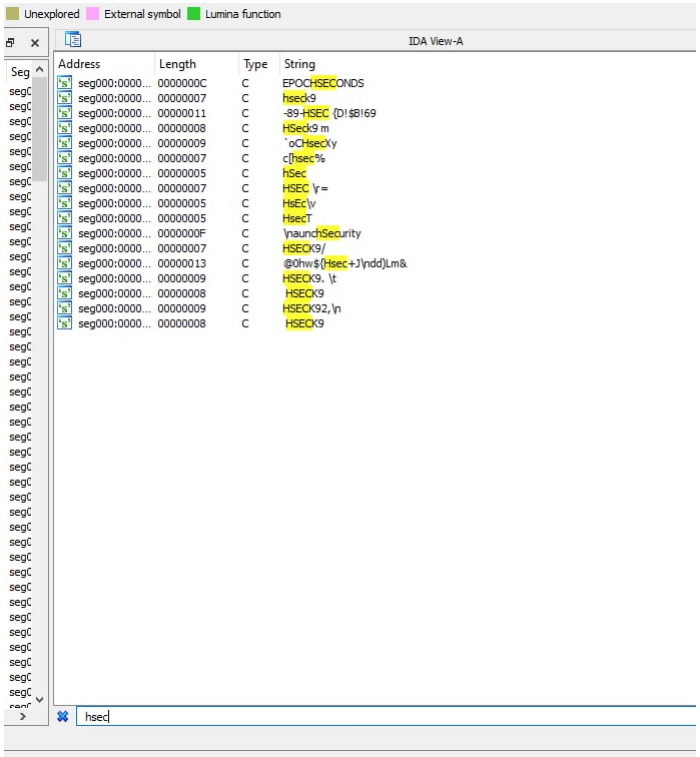


Рисунок 9 — Поиск по строке HSEC в файле прошивки маршрутизатора

Для поиска механизма стоит ознакомится в документации телекоммуникационного устройства способ активации. Так как можно отметить себе диалоговые строки по которым можно будет делать далее поиск. Следующие строки были взяты из документации:

— auth failed for Base64 string;

— failed to generate aes key;

— nll AuthCodeBase58 painter;

— auth failed for Base68 string.

данные строки можно найти, на следующем рисунке 10

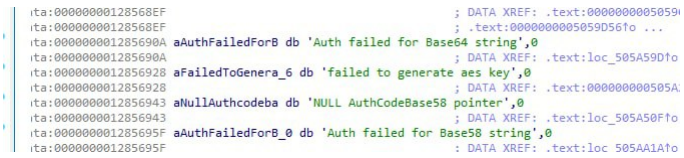


Рисунок 10 — Диалоговые строки в файле прошивки маршрутизатора

Очередной сложностью на пути анализа стало отличие библиотечного и пользовательского кода. При разработке объемных программных продуктов стоит отметить,

что исходный код будет состоять из общих библиотек и кода, написанных разработчиками компаний. Зачастую в таком коде содержится библиотечный код, который сложно отличить от кода пользователя.

Решением данной проблемы стал модуль программы IDA, который позволяет определять библиотечный код. Данный модуль называется FLIRT (англ. fast library identification and recognition technology), который в свою очередь может облегчить анализ механизма лицензирования за счет определения библиотечного кода, тем самым обращая наше внимание на код разработчиков компании.

Теперь нашим объектом анализа являются тысячи строк механизма лицензирования на языке ассемблера, что изображено на следующем рисунке 11.

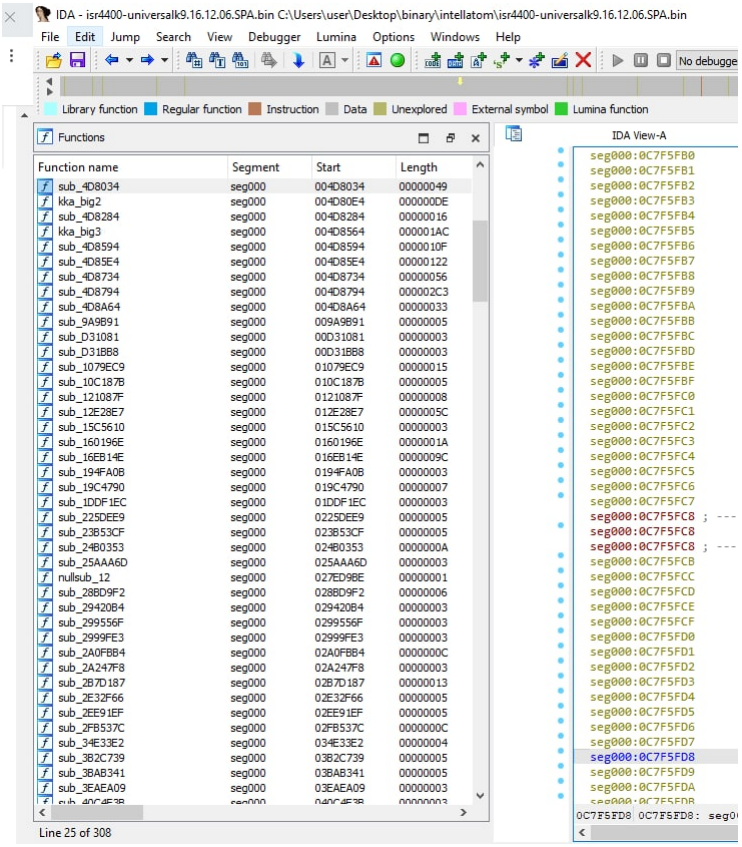


Рисунок 11 — Все функции в файле прошивки маршрутизатора

Используя выше описанные техники можно найти механизм лицензирования OPLR для дальнейшего анализа.

5. Анализ механизма лицензирования.

При использовании IDA стоит узнать все характеристики маршрутизатора. Такие как:

— архитектура процессора;

— разрядность.

что показано на следующем рисунке 12 при попытке открыть файл прошивки

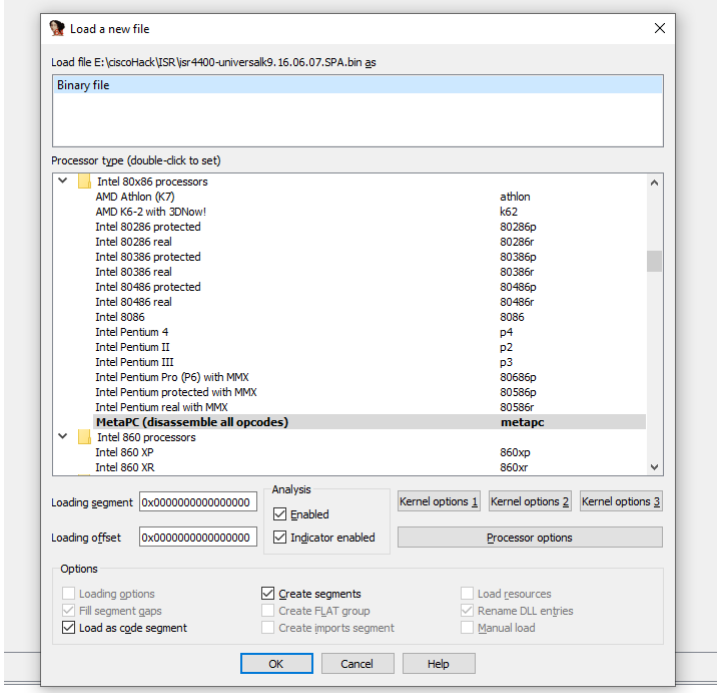


Рисунок 12 — Открытие файла прошивки маршрутизатора

Данный этап очень важен, так как на выбранном вами процессоре и разрядности программы будет строиться ассемблерный файл с разметками, характерными данному процессору. Информацию о исследуемом нами маршрутизаторе можно узнать на сайте вендора.

Подход, использованный в прошлом методе не даст никаких результатов, так как скопировать сам механизм не удастся. Механизм лицензирования OPLR немного схож с PLR, но с модификациями в виде использования новых общепринятых библиотек. Ключ активации механизма лицензирования OPLR это конкатенация нескольких строк, которые в свою очередь хранятся или генерируются. Следующий рисунок 13 содержится в себе некоторые проверки ключа.

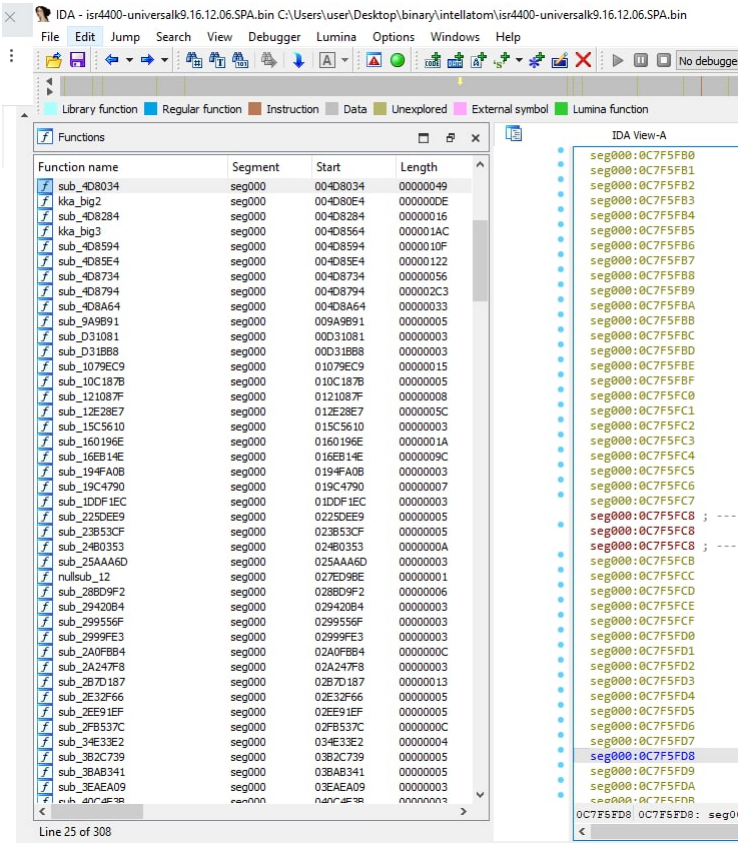


Рисунок 13 — Проверка ключа активации в файле прошивки маршрутизатора

Изучив каждую строку и критерии конкатенации строк можно восстановить механизм лицензирования. Для разбора ассемблерного кода стоит понимать конструкции функций, описание которых есть в книге Дениса Юричева « «Reverse Engineering для начинающих»» [5].

6. Тестирование.

Написание прототипа генератора механизма лицензирования OPLR возможно только при полном разборе ассемблерного кода и реализации его на любом языке программирования.

Результат Реализован прототип генератора механизма лицензирования на языке C#. Критерий выбора языка программирования осуществлялся по интересующемуся

Недостатки защиты метода лицензирования:

— незашифрованный диск маршрутизатора, что позволяло свободно обозревать исходный код;

— отсутствие каких-либо методов защиты исходного кода;

— использование общепринятых библиотек программирования.

Вывод. В данной главе показано применение схемы анализа безопасности механизма лицензирования на практике. Схема была применена на механизмах Cisco PLR и OPLR. Также описанны уявзиоспсти и недостатки найденные после применения схемы анализа безопасности механизмов лицензирования.

4 Методы защиты систем лицензирования Cisco PLR/OPLR

Анализ механизмов лицензирования зачастую происходит двумя типами злоумышленникам. Первые - любители, результат которых чаще всего является положительным из-за пренебрежительного отношения компании к безопасности программного обеспечения. Использовав методы защиты, которые были описаны в предыдущей главе, вы значительно усложните анализ, что предполагает наличие большей квалификации от злоумышленника. Но также есть продвинутые аналитики - второй тип исследователей, которые занимаются анализом зачастую за определенную плату и на заказ. Следующий описанный способ определенно будет казаться невозможным для первого типа злоумышленников и довольно затруднительным для второго типа, что потребует больших ресурсов, времени и денег.

4.1 Общее описание методов защиты

Первым важным и одним из эффективных методов стоит выделить **обфускацию**. Обфускация, а иначе шифратор, программа позволяющая в большей или меньшей мере внести изменения в байт-код или исходный код приложения. Познакомившись с процессом компиляции на языках Java и C++, вспомним что исходный код преобразуется в байт-код или бинарный файл. Естественно есть и обратный процесс - декомпиляция. Количество программ позволяющие декомпилировать множество, причем некоторые их них извлекают исходный код программы в довольно хорошем качестве. Для того чтобы защититься используют обфускаторы, которые можно классифицировать следующим образом:

— стандартная обфускация;

— обфускация хода выполнения программы;

— структурная обфускация.

Стандартная обфускация изменяет структуру программы следующими способами: удалением информации отладчика и переименованием идентификаторов. Важно отметить что изменение идентификаторов имеет место при работе как с исходным кодом программы, так с байт-кодом. Следующие примеры будут на языке Java. Например

переименование может превратить понятный класс:

```
1 public class Configuration {
2 public String getHttpPort() {
3
4 {
5 }
6 // место вызова
7 Configuration c = new Configuration()
8 String port = c.getHttpPort()
```

в следующую непонятную структуру:

```
1 public class a {
2 public String b() {
3
4 {
5 }
6 // место вызова
7 a c = new a()
8 String port = c.b()
```

При компиляции Java классов компилятор может сохранять отладочную информацию, которой можно воспользоваться, например для удаленной отладки приложения. Следовательно стоит ее отключить, сделать это можно добавлением опции **-g:none** при компиляции:

javac -g:none MyClass.java

Объединив вышеперечисленные способы - это простой, стандартный способ обфускации. Который уже во многом затрудняет понимание кода взломщиком, но логика работы программы абсолютно никак не скрывается.

Продвинутые способы обфускации. Следующий способ изменяет ход выполнения программы (англ. flow obfuscators). Чаще всего при этом участвуют условные конструкции такие как **if**, **switch** и циклы **for** и **while**. На примере языка Java, байт код

стараются изменить так, чтобы он имел прямых аналогов в языке Java. Данный способ значительно усложняет работу злоумышленнику.

Последний пример обфускатора связан с изменением структурных данных (structural obfuscator). Изменение структурных данных заключается в изменении принципов наследования. Например, в иерархии можно создать несколько промежуточных классов. Разбиение можно применять не только к классам но и к массивам. Вдобавок, было бы уместно в нашем случае, кодировать символы строковых констант и переменных.

Использование второго и третьего способа обфускации вместе значительно усложняет код программы, что восстановление оригинального кода становиться практически невозможным.

После обфускации стоит задача защитить файлы программ от модификации и запуска. Для защиты исправления исходного кода программного обеспечения телекоммуникационного устройства существует механизм **целостность кода** (англ. Integrity Check). Без использования механизма целостности кода, злоумышленник сможет модифицировать или вовсе удалить механизм из программного обеспечения. Целостность кода — это функция защиты от угроз, которая проверяет драйверы и системные файлы на вашем устройстве на наличие признаков повреждения или вредоносного программного обеспечения. Данная техника защиты подробно описана здесь « Integrity Check » [6]. Идея данного метода состоит в контроле хеш суммы размера файла, то-есть изменение какого-либо файла может привести к нарушению целостности кода. Также часто встречается проверка целостности самого механизма проверки целостности кода. Внедрив в прошивку маршрутизатора данный метод мы сможем защитить код от модификаций и запуска измененной злоумышленником прошивки.

Следующий описанный метод это **упаковщик** (англ. packers). Упаковка исполняемых файлов заключается в сжатии исполняемого файла. Данную упаковку применяют по ряду следующих причин:

— некоторые упаковщики совмещены с шифрованием содержимого файла для предотвращения анализа программы;

— упакованный файл занимает меньше места на носители информации, что способствует быстрой загрузке программы в память.

Общий принцип упаковщика таков: после нажатия на EXE-файле и его запуска выполнение основного кода программы начинается с так называемой точки входа (Entry

Point) — адреса, по которому передается управление после загрузки программы в оперативную память. Когда программа запакована, алгоритм работы несколько изменяется. Упаковщик запоминает точку входа EP, а затем, используя алгоритмы архивирования, сжимает содержимое файла (как правило, это секция кода и данных), после чего дописывает свою сигнатуру после либо до сжатого кода программы и перенаправляет ее не в основной код программы, а в код упаковщика (точнее сказать — распаковщика). Сам же код распаковщика, находящийся теперь внутри файла, получает управление первым и распаковывает упакованные секции кода/данных в памяти. На диске исходный файл остается нетронутым, то есть упакованным, неизменным. После того, как код и данные программы распакованы, код распаковщика восстанавливает таблицу импорта и передает управление основному коду программы на бывшую точку входа, которая в упакованных программах называется оригинальной точкой входа (Original Entry Point). Сжатие данных (упаковка) основывается на свойстве энтропии информации, а алгоритмы по своей сути очень схожи с теми, что применяются в архиваторах, только в отличие от первых упаковщики для исполняемых файлов распаковывают данные в оперативную память.

Использование упаковщика значительно усложняет анализ. Помимо этого, разработчики могут использовать многократную упаковку (так называемый послойный пак), применять малоизвестные или вовсе самописные сигнатуры, которые будут отсутствовать в общем доступе. Интересно, что любой упаковщик оставляет свою уникальную сигнатуру в бинарном файле, а соответственно, умея пользоваться Hex-редакторами, можно определить его сигнатуру и без анализаторов.

Минус данного метода состоит в том, что существуют анализаторы, которые позволяют распаковывать данный механизм, но стоит отметить, что написание своих сигнатур упаковки значительно усложнит анализ кода программы.

Преимущество следующего метода в том, что при анализе программного обеспечения злоумышленник не доходит даже до самих программ, не говоря уже о коде. В чем состоит идея данного метода изображено на следующем рисунке 14.

Рисунок 14 — Метод защиты важных механизмов программного обеспечения

В диске программного устройства всегда будут присутствовать файлы операционной системы и файлы прошивки. В предложенном методе стоит разделить файлы операционной системы, не критически важных механизмов от важных механизмов. Далее **зашифровать** файлы критически важных механизмов для того, чтобы злоумышленник не смог просматривать код программ. Так же схема анализа которая была предложена в данной работе будет бессильна, так как все нужные файлы будут зашифрованы.

Минус данного метода состоит в том, что данный механизм шифрования будет также находиться на программном устройстве, так как операционная система должна каким-то образом взаимодействовать с данными механизмами, что является потенциальной угрозой.

Следует отметить что не один из вышеописанных методов не был использован в исследуемых телекоммуникационных устройствах. Методы защиты описанные выше могут защитить от схемы анализа описанной в данной работе. Для более стойкой защиты рекомендуется использовать комбинированный метод.

Вывод. В данной главе описаны методы защиты механизмов лицензирования и предложен метод защиты от схемы анализа, представленный в предыдущей главе. Также стоит отметить что ни один из данных методов не был использован в исследуемых телекоммуникационных устройствах.

ЗАКЛЮЧЕНИЕ

Прежде всего хотелось бы отметить факт обнаружения уязвимостей и недостатков безопасности прошивки и механизмов лицензирования отправлены команде безопасности Cisco.

В данной работе представлены методы защиты механизмов лицензирования от предложенной схемы анализа. В результате были выявлены слабые стороны защиты механизмов лицензирования PLR, OPLR, и разработаны генераторов ключей на языке Java для PLR и на языке C# для OPLR. Предложен метод защиты от описанного алгоритма анализа. Пройдя по следующим ссылкам можно увидеть программные устройства содержащие механизм лицензирования OPLR «Device with OPLR » [7]. Далее будут представлены семейства телекоммуникационных устройств содержащие механизм лицензирования PLR: Cisco ISR, Cisco Catalyst, Cisco ASR, Cisco ISR и Cisco CSR. В сумме которых входят несколько десятков из маршрутизаторов, межсетевых экранов и коммутаторов содержащих в себе данные уязвимые методы лицензирования.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Cracking Cisco's Sourcefire licensing system. — URL: https://blog.hackercat.ninja/assets/files/cracking_sf_license_system.pdf ; (дата обращения:17.09.2022).
2. Cisco Licenses. — URL: https://www.cisco.com/c/en/us/td/docs/net_mgmt/prime/access_registrar/9-1/installation/guide/incfg/smartlicensing.pdf ; (дата обращения:20.12.2022).
3. Smart Licensing PLR. — URL: https://www.cisco.com/c/en/us/td/docs/net_mgmt/prime/access_registrar/9-2/installation/guide/incfg/smartlicensing.pdf ; (дата обращения:17.09.2022).
4. Cisco Smart Licensing OPLR. — URL: https://www.cisco.com/c/en/us/td/docs/routers/sl_using_policy/b-sl-using-policy/task_library.html#Cisco_Reference.dita_c382a6c3-99ac-4756-9956-f2612e8e7f09 ; (дата обращения:01.11.2022).
5. Reverse Engineering для начинающих. — URL: http://old-dos.ru/index.php?page=lib&do=show&id=669 ; (дата обращения:25.10.2022).
6. Integrity Check. — URL: https://habr.com/ru/company/pt/blog/177189/ ; (дата обращения:08.12.2022).
7. Device with OPLR. — URL: https://www.cisco.com/c/en/us/td/docs/routers/sl_using_policy/b-sl-using-policy/task_library.html#Cisco_Reference.dita_c382a6c3-99ac-4756-9956-f2612e8e7f09 ; (дата обращения:20.12.2022).

АНТИПЛАГИАТ
ОБНАРУЖЕНИЕ ЗАИМСТВОВАНИЙ

Отчет о проверке на заимствования №1

Автор: Раимбеков Азим
Проверяющий: Раимбеков Азим

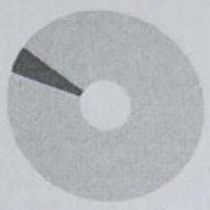
Отчет предоставлен сервисом «Антиплагиат» - http://users.antiplagiat.ru

ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 1
Начало загрузки: 16.01.2023 16:15:53
Длительность загрузки: 00:00:01
Имя исходного файла: Raimbekov_v8.pdf
Название документа: Raimbekov_v8
Размер текста: 42 кБ
Символов в тексте: 43271
Слов в тексте: 5324
Число предложений: 298

ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Начало проверки: 16.01.2023 13:15:56
Длительность проверки: 00:00:02
Комментарии: не указано
Модули поиска: Интернет Free



СОВПАДЕНИЯ	САМОЦИТИРОВАНИЯ	ЦИТИРОВАНИЯ	ОРИГИНАЛЬНОСТЬ
4,81%	0%	0%	95,19%

Совпадения — 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.
Совпадения, 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]	3,55%	Реверсинг малвари для начинающих. Вскрываем упаковщики, ломаем протекторы | «Хакер» https://xakep.ru	19 Мар 2020	Интернет Free	
[02]	0%	Реверсинг малвари для начинающих. Вскрываем упаковщики, ломаем протекторы — «Хакер» https://xakep.ru	09 Дек 2019	Интернет Free	
[03]	0%	eXeL@B ::: Распаковка: от самого простого к чуть более сложному (1/2) https://exelab.ru	08 Янв 2016	Интернет Free	

Еще источников: 6
Еще совпадений: 1,26%