МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего профессионального образования

«Национальный исследовательский ядерный университет «МИФИ»

Факультет кибернетики и информационной безопасности

Кафедра №36 «Информационные системы и технологии»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к дипломному проекту на тему:

Реализация механизма автоматического выбора сертификата открытого ключа пользователя на основании его контекста безопасности

Группа К10-361

Студент дипломник _____ (подпись) (Воронин Д.Л.) (ФИО)

Руководитель проекта _____ (подпись) (Муравьёв С.К.) (ФИО)

Заведующий кафедрой _____ (подпись) (Шумилов Ю.Ю.) (ФИО)

Москва 2014г.

Пояснительная записка к дипломному проекту: 124 страницы, 7 приложений, 15 рисунков, 1 таблица, список литературы из 20 наименования.

Ключевые слова: инфраструктура открытых ключей PKI, сертификат открытого ключа X509, контекст безопасности, SELinux, OpenSSL, M2Crypto, PostgreSQL.

В работе предлагается реализация механизма автоматического выбора сертификата на основе контекста безопасности: проводится обзор инфраструктуры открытых ключей PKI, сертификата открытого ключа X509 и мандатной системы контроля доступа SELinux, дорабатывается модуль создания многоэкземплярных директорий pam_namespace, реализуется дополнение сертификата X509 v3_secon в OpenSSL, разрабатывается утилита создания сертификатов с контекстом безопасности. Работоспособность механизма показана на примере СУБД PostgreSQL.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего профессионального образования

«Национальный исследовательский ядерный университет «МИФИ»

Факультет Кибернетики и информационной безопасности

Кафедра №36 Информационных систем и технологий

Специальность 010501 «Прикладная математика и информатика» Группа К10-361

Утверждаю
Зав. кафедрой
« » 2014г.

ЗАДАНИЕ НА ДИПЛОМНЫЙ ПРОЕКТ (работу)

1. Фамилия, имя, отчество дипломанта Воронин Дмитрий Леонидович

2. Тема проекта (работы) Реализация механизма автоматического выбора сертификата открытого ключа пользователя на основании его контекста безопасности
   Утверждена приказом по институту от «_____» _____ 2014г. № _____

3. Срок сдачи студентом готового проекта «17» июня 2014г.

4. Руководитель дипломного проектирования Муравьёв Сергей Константинович, нач. отдела ФГУП «ЦНИИ ЭИСУ», к.т.н.

1. Исходные данные к проекту:

   Операционная система Fedora 20, SELinux, языки программирования: C, Bash, Python.

2. Содержание проекта:

   2.1. литература и обзор работ, связанных с проектом:

   Полянская О.Ю., Горбатов В.С. — Инфраструктуры открытых ключей — М.: Изд-во "Интернет-университет информационных технологий - ИНТУИТ.ру", "БИНОМ. Лаборатория знаний", 2007. - 368 с.

   2.2. расчетно-конструкторская, теоретическая, технологическая часть:

   Разработка алгоритма автоматического выбора сертификата пользователя на основании контекста безопасности;

   Разработка способа хранения контекста безопасности в сертфикате;

   Разработка средств создания сертификата пользователя с значением контекста безопасности.

   2.3. экспериментальная часть:

   Реализация структуры хранения контекста безопасности;

   Реализация средств создания сертификата с контекстом безопасности пользователя;

   Интеграция механизма автоматического выбора сертификата в СУБД PostgreSQL;

   Тестирование механизма.

   2.4. экономико-организационная и информационная часть

   2.5. охрана труда и техника безопасности, экологическая часть

3. Отчетный материал проекта:

   3.1. пояснительная записка;

   3.2. графический материал (с указанием обязательных чертежей);

   Схема стенда

   Скриншоты работы механизма

   3.3. макетно-экспериментальная часть

   Листинги кода

   Патчи доработанных программных средств

КАЛЕНДАРНЫЙ ПЛАН РАБОТЫ НАД ПРОЕКТОМ

(составляется руководителем с участием студента в течение первой недели с начала дипломного проектирования)

№ п/п	Наименование этапов работы	Сроки выполнения этапов	Степень готовности проекта в % к объему работы	Время выполнения
1	Анализ существующих подходов к выбору сертификата пользователя	28.03.14 — 03.04.14	5	04.04.14
2	Анализ литературы	04.04.14 — 08.04.14	15	08.04.14
3	Выбор подхода к решения задачи и средств реализации. Написание 1 главы пояснительной записки.	08.04.14 — 13.04.14	30	15.04.14
4	Реализация компонентов механизма выбора сертификата пользователя.	14.04.14 — 27.04.14	55	23.04.14
5	Написание 2 главы пояснительной записки	28.04.14 — 04.05.14	65	06.05.14
6	Отладка механизма, интеграция с СУБД PostgreSQL	05.05.14 — 15.05.14	80	13.05.14
7	Написание 3 главы пояснительной записки	16.06.14 — 23.05.14	95	24.05.14
8	Оформление пояснительной записки и графического материала	24.05.14 — 31.05.14	100	29.05.14

Дата выдачи задания «28» марта 2014г.

Руководитель дипломного проектирования _____

Задание принял к исполнению (дата и подпись студента) _____

Содержание

Введение

Для хранения и обработки информации в современном обществе используются информационные системы. Информация, которой они оперируют, как правило, обладает определённым уровнем конфиденциальности. Это объясняет необходимость систем контроля доступа, позволяющих конкретному субъекту (пользователю) получить допуск к объекту (информации) соответствующего уровня. Получение доступа в некоторых информационных системах осуществляется путём предъявления сертификата открытого ключа — цифрового удостоверения личности субъекта. Использование одного сертификата разрешает субъекту доступ ко всей информации. Для обеспечения доступа субъекта к объектам разного уровня конфиденциальности, а не ко всей информации в целом, необходимо несколько сертификатов соответствующих уровней, доступных субъекту. Ввиду того, что уровней может быть много, то необходима автоматизация процесса выбора сертификата. Поэтому тема данного дипломного проекта является **актуальной**.

В данной дипломной работе предлагается реализация механизма выбора сертификата открытого ключа пользователя на основании его контекста безопасности.

При реализации будет использована инфраструктура открытых ключей (англ. *PKI* [1]), один из принципов построения которой предполагает наличие удостоверяющего центра, выпускающего сертификаты открытых ключей пользователей, тем самым удостоверяя их личность. В каждом из сертификатов в дополнительном атрибуте будет содержаться значение контекста безопасности.

В качестве поставщика метки безопасности будет использоваться SELinux [2] — реализация системы мандатного контроля доступа, которая используется в некоторых дистрибутивах Linux (например, Fedora) вместе с дискреционным механизмом контроля доступа. С помощью специально описанных политик регулируется доступ субъекта (пользователя) к объекту (файлу, директории и т.д.). SELinux может работать в многоуровневом режиме (англ. *MLS*). Этот режим основан на принципе, что субъект может иметь доступ к объекту, если уровень безопасности субъекта соответствует уровню безопасности объекта.

Автоматизация процесса выбора сертификата будет осуществлена с использованием многоэкземплярности директорий — механизма создания независимых копий.

Научная новизна данной работы определяется в выборе сертификата открытого ключа пользователя на основании его контекста безопасности.

В работе показана работоспособность предложенного механизма на примере СУБД PostgreSQL [3]. Это определяет **практическую значимость** дипломной работы.

Таким образом, **целью** данной работы является разработка механизма автоматического выбора сертификата пользователя на основании его контекста безопасности. Для достижения поставленной цели были сформулированы следующие **задачи**:

1. Изучить принципы построения инфраструктуры открытых ключей PKI;
2. Исследовать современные средства выбора сертификата открытого ключа;
3. Разработать средства создания сертификатов с контекстом безопасности пользователя;
4. Автоматизировать процесс выбора сертификатов, используя механизм многоэкземплярности;
5. Показать применение разработанного механизма для СУБД PostgreSQL.

Реализация данного механизма произведена на дистрибутиве Linux Fedora 20.

1 Обзор существующих подходов к управлению сертификатами пользователя

В данной главе проводится обзор инфраструктуры открытых ключей PKI, исследуются принципы построения сертификатов формата X509, анализируются современные подходы к управлению сертификатами пользователя.

1.1 Инфраструктура открытых ключей PKI

Инфраструктура открытых ключей (*PKI*, *Public Key Infrastructure*) [4] — набор средств (технических, материальных, людских и т. д.), распределенных служб и компонентов, в совокупности используемых для поддержки криптозадач на основе закрытого и открытого ключей.

В основе PKI лежит использование криптографической системы с открытым ключом и несколько основных принципов:

- закрытый ключ известен только его владельцу;
- удостоверяющий центр создает сертификат открытого ключа, таким образом удостоверяя этот ключ;
- никто не доверяет друг другу, но все доверяют удостоверяющему центру;
- удостоверяющий центр подтверждает или опровергает принадлежность открытого ключа заданному лицу, которое владеет соответствующим закрытым ключом.

PKI реализуется в модели клиент-сервер, то есть проверка какой-либо информации, предоставляемой инфраструктурой может происходить только по инициативе пользователя.

Основные компоненты PKI:

- Удостоверяющий центр (УЦ) является основной структурой, формирующей цифровые сертификаты подчиненных центров сертификации и конечных пользователей. УЦ является главным управляющим компонентом PKI. Он является доверенной стороной.

- Сертификат открытого ключа (чаще всего просто сертификат) — это данные пользователя и его открытый ключ, скрепленные подписью удостоверяющего центра. Выпуская сертификат открытого ключа, удостоверяющий центр тем самым подтверждает, что лицо, поименованное в сертификате, владеет секретным ключом, который соответствует этому открытому ключу.

- Репозиторий — хранилище, содержащее сертификаты и списки отозванных сертификатов (СОС) и служащее для распространения этих объектов среди пользователей.

- Архив сертификатов — хранилище всех изданных когда-либо сертификатов (включая сертификаты с закончившимся сроком действия). Архив используется для проверки подлинности электронной подписи, которой заверялись документы.

- Конечные пользователи — пользователи, приложения или системы, являющиеся владельцами сертификата и использующие инфраструктуру управления открытыми ключами.

Основные задачи системы информационной безопасности, которые решает инфраструктура управления открытыми ключами:

- обеспечение конфиденциальности информации;

- обеспечение целостности информации;

- обеспечение аутентификации пользователей и ресурсов, к которым обращаются пользователи;

- обеспечение возможности подтверждения совершенных пользователями действий с информацией (неотказуемость, или апеллируемость — англ. non-repudiation).

PKI напрямую не реализует авторизацию, доверие, именование субъектов криптографии, защиту информации или линий связи, но может использоваться как одна из составляющих при их реализации.

Задачей PKI является определение политики выпуска цифровых сертификатов, выдача их и аннулирование, хранение информации, необходимой для последующей проверки правильности сертификатов.

Инфраструктура открытых ключей основана на использовании криптографической системы с открытым ключом.

1.2 Формат сертификатов открытых ключей X509

Формат сертификата открытого ключа **X509** [5] определен в рекомендациях Международного Союза по телекоммуникациям ITU (X.509) и документе RFC 3280 Certificate & CRL Profile [6] организации инженерной поддержки Интернета. В настоящее время основным принятым форматом является формат версии 3, позволяющий задавать дополнения, с помощью которых реализуется определенная политика безопасности в системе.

Сертификат открытого ключа или шифрования представляет собой структурированную двоичную запись в абстрактной синтаксической нотации ASN1. В сертификате имеются элементы данных, сопровождаемые цифровой подписью. Сертификат содержит 6 обязательных и 4 необязательных поля. К обязательным полям сертификата относятся:

- `Certificate Serial Number` — серийный номер сертификата;
- `Signature Algorithm Identifier` — идентификатор алгоритма подписи;
- `Validity` — период действия (когда выдан и до какого времени может быть использован);
- `Subject Public Key Information` — открытый ключ субъекта;
- `Subject Name` — имя субъекта.

Субъект сертификата определяет объект, контролирующая секретный ключ и обладающая открытым ключом. Поле `Version` определяет версию сертификата. Наличие необязательных полей характерно для версий 2 и 3. Структура сертификата X509 приведена на *рис. 1.1.*

<table>
<tr><td>Версия</td><td rowspan="8">Версия v1</td><td rowspan="10">Версия v2</td><td rowspan="11">Версия v3</td></tr>
<tr><td>Серийный номер</td></tr>
<tr><td>Идентификатор алгоритма подписи</td></tr>
<tr><td>Имя издателя</td></tr>
<tr><td>Период действия (не раньше/не позднее)</td></tr>
<tr><td>Имя субъекта</td></tr>
<tr><td>Информация об открытом ключе субъекта</td></tr>
<tr><td>Уникальный идентификатор издателя</td></tr>
<tr><td>Уникальный идентификатор субъекта</td></tr>
<tr><td colspan="2">Дополнения</td></tr>
<tr><td>Подпись</td><td colspan="3">Все версии</td></tr>
</table>

Рис. 1.1: Структура сертификата X509

Издатель сертификатов присваивает каждому выпускаемому сертификату серийный номер `Certificate Serial Number`, который должен быть уникален. Комбинация имени издателя и серийного номера однозначно идентифицирует каждый сертификат.

В поле `Signature Algorithm Identifier` указывается идентификатор алгоритма электронно-цифровой подписи, который использовался издателем сертификата для подписи сертификата.

Поле `Issuer Name` содержит отличительное имя издателя, который выпустил этот сертификат. В поле `Validity` (`Not Before/After`) указываются даты начала и окончания периода действия сертификата.

Поле `Subject Name` содержит владельца секретного ключа, соответствующего открытому ключу данного сертификата. Субъектом сертификата может выступать удостоверяющий центр, регистрационный центр или конечный субъект.

Поле `Subject Public Key Information` содержит информацию об открытом ключе субъекта: сам открытый ключ, необязательные параметры и идентификатор алгоритма генерации ключа. Это поле всегда должно содержать значение. Открытый ключ и необязательные параметры алгоритма используются для верификации цифровой подписи (если субъектом сертификата является удостоверяющий центр) или управления ключами.

Важная информация находится также в дополнениях сертификата. Они позволяют включать в сертификат информацию, которая отсутствует в основном содержании, определять валидность сертификата и наличие у владельца сертификата прав доступа к той или иной системе. Кроме того, в дополнениях содержится технологическая информация, позволяющая

легко проверить подлинность сертификата. Каждая организация может использовать свои частные дополнения, удовлетворяющие конкретным требованиям ведения бизнеса. Однако большинство требований включено в стандартные дополнения, поддержку которых обеспечивают коммерческие программные продукты.

Субъектом (`Subject`) сертификата может быть конечный пользователь, система или удостоверяющий центр.

Опциональное поле `Extensions` (дополнения) появляется в сертификатах третьей версии.

1.2.1 Формат дополнения сертификата X509

Формат дополнения сертификата **X509** определён рекомендациями X509 версии 3 [7]. Дополнения сертификата описываются следующей структурой:

```
Extension ::= SEQUENCE {
        extnID OBJECT IDENTIFIER,
        critical BOOLEAN DEFAULT FALSE,
        extnValue OCTET STRING
}
```

Дополнение сертификата состоит из следующих объектов:

- `extnID` — идентификатор объекта;
- `critical` — признак критичности;
- `extnValue` — строка, определяющая значение расширения.

Каждое дополнение состоит из идентификатора типа дополнения `Extension identifier`, признака критичности `Criticality flag` и собственно значения дополнения `Extension value`. Идентификатор типа дополнения задает формат и семантику значения дополнения. Признак критичности сообщает приложению, использующему данный сертификат, существенна ли информация о назначении сертификата и может ли приложение игнорировать данный тип дополнения.

Дополнения можно разделить на две категории: ограничивающие и информационные.

Первые ограничивают область применения ключа, определенного сертификатом, или самого сертификата.

Вторые содержат дополнительную информацию, которая может быть использована в прикладном программном обеспечении пользователем сертификата.

К ограничивающим дополнениям относятся:

- основные ограничения (`Basic Constraints`);
- назначение ключа (`Key Usage`);
- расширенное назначение ключа (`Extended Key Usage`);
- политики применения сертификата (`Certificates Policies, Policy Mappings, Policy Constraints`);
- ограничения на имена (`Name Constraints`).

К информационным дополнениям относятся:

- идентификаторы ключей (`Subject Key Identifier`, `Authority Key Identifier`);
- альтернативные имена (`Subject Alternative Name`, `Issuer Alternative Name`);
- пункт распространения списка аннулированных сертификатов (`CRL Distribution Point`, `Issuing Distribution Point`);
- способ доступа к информации УЦ (`Authority Access Info`).

Также стандарт открытых ключей X509v3 позволяет создавать пользовательские дополнения и использовать их в сертификатах.

1.3 Анализ современных подходов к управлению сертификатами пользователей

Возможность использовать несколько действующих сертификатов, связанных с разными политиками и назначениями ключей, связана с тем, что для каждого вида активности должен выбираться корректный (соответствующий назначению) закрытый ключ. Например, для

подписи электронных сообщений используется один ключ, для аутентификации — другой. При этом возникают ситуации, когда пользователь должен выбрать ключ, который будет использовать в каком-либо приложении.

При этом рекомендуется использовать отдельную пару ключей для решения одной определённой задачи.

Важным преимуществом использования этого подхода является удобство независимого управления сертификатами в случае их аннулирования. Если открытый ключ содержится в нескольких сертификатах, то при компрометации ключа необходимо получить все сертификаты, содержащий этот ключ, после чего аннулировать. Если хотя бы один сертификат в таком случае аннулирован не будет, это может привести к серьёзному риску нарушения безопасности системы.

Если же будет скомпрометирована одна пара ключей, то её изъять и выпустить новую пару ключей будет не так сложно.

Кроме того, сертификаты, связанные с уникальной парой ключей, независимо конструируются: они могут соответствовать разным политикам, а также иметь разные сроки действия, назначения и процедуры управления. Один сертификат может устаревать или аннулироваться независимо от других сертификатов. Использование одного и того же открытого ключа в нескольких сертификатах усложняет управление сертификатами.

Как правило, прикладное программное обеспечение выбирает сертификат пользователя автоматически на основании дополнений сертификата, в которых может быть указана цель использования ключа, различные ограничения, а также и идентификатор политики применения сертификата.

1.3.1 Назначение ключа

Дополнение `keyUsage` определяет цель использования ключа, содержащегося в сертификате. Данное дополнение может быть использовано в сертификате, когда необходимо ограничить перечень операций, в которых может быть использован ключ.

Оно может принимать следующие значения:

- `digitalSignature` — ключ, содержащийся в сертификате, используется для проверки цифровых подписей (кроме подписей сертификатов, списков CRL);

- `nonRepudiation` — ключ, содержащийся в сертификате, не может быть использован для цифровых подписей;
- `keyEncipherment` — ключ, содержащийся в сертификате, используется для шифрования закрытых ключей;
- `dataEncipherment` — ключ, содержащийся в сертификате, используется для шифрования исходных данных пользователей;
- `keyAgreement` — ключ, содержащийся в сертификате, используется для согласования ключей;
- `keyCertSign` — ключ, содержащийся в сертификате, используется для проверки сигнатур подписи публичных ключей;
- `cRLSign` — ключ, содержащийся в сертификате, используется для проверки сигнатур списка отозванных сертификатов.

1.3.2 Ограничения сертификата

Основными ограничениями сертификата являются дополнения `Basic Constraints` (основные ограничения) и `Name Constraints` (ограничение на имя сертификата).

Дополнение `Basic Constraints` используется в основном в качестве ограничения возможности использования открытого ключа для проверки цифровых подписей сертификатов.

Если в дополнении используется строка:

```
CA: TRUE
```

то она говорит о том, что пара ключей может быть использована для проверки цифровых подписей сертификата.

Для ограничения данной возможности используется следующая строка:

```
CA: FALSE
```

Дополнение `Name Constraints` используется в сертификатах удостоверяющего центра для того, чтобы имя субъекта всех сертификатов, выданных удостоверяющим центром, начинались с определённых атрибутов.

Например, если в дополнении `Name Constraints` указано

```
C=ru, ST=msk, L=msk
```

то субъект с именем

```
C=ru, ST=msk, L=msk, OU=mephi
```

считается корректным с точки зрения ограничения имени.

1.3.3 Политика применения сертификатов

Политика — набор определённых правил, предназначенные для достижения различных целей организации. Политика безопасности предназначена, главным образом, для обеспечения защиты информации организации. При этом, должен быть реализован комплекс механизмов безопасности и процедур. Механизмы безопасности являются средствами обеспечения безопасности системы, а процедуры — шаги для её обеспечения.

Политика применения сертификатов (ППС) — это документ, описывающий политику безопасности в отношении выпуска сертификатов и распространения информации о статусе сертификатов. Эта политика безопасности регламентирует операционную работу удостоверяющего центра, а также регулирует ответственность пользователей при получении и использовании сертификатов и ключей. ППС гарантирует защищенность всего жизненного цикла сертификатов, начиная от их генерации и заканчивая аннулированием или истечением срока действия.

Фактом выпуска сертификата удостоверяющий центр подтверждает пользователю сертификата (доверяющей стороне), что данный открытый ключ принадлежит данному лицу (субъекту сертификата). Степень доверия, с которой пользователю сертификата следует полагаться на его надежность, зависит от назначения сертификата, декларируемого политикой, и возможности его использования в конкретном приложении.

При принятии решения об использовании сертификата пользователь может ориентироваться на указатель ППС в сертификате формата X509v3.

Косвенным образом пользователь может получить информацию о политике PKI на основании дополнений сертификата X509v3: `Certificate Policies`, `Policy Mappings` и `Policy Constraints`:

- Дополнение `Certificate Policies` содержит набор из одного или нескольких правил, каждое из которых содержит идентификатор политики и дополнительную информацию. В сертификате конечного субъекта в дополнении указывается политика удостоверяющего центра, на основании которой был выдан сертификат. Приложениям, которым необходимо загружать сертификат на основании политики, необходимо иметь список OID разрешённых политик. Если приложение найдёт OID сертификата в этом списке, то сертификат должен быть принят, иначе — не принят.

- Дополнение `Policy Mappings` используется в сертификатах удостоверяющего центра. Его значение представляет собой пару идентификаторов OID `issuerDomainPolicy` и `subjectDomainPolicy`. Удостоверяющий центр считает политику `issuerDomainPolicy` эквивалентной политике `subjectDomainPolicy`.

- Дополнение `Policy Constraints` используется в сертификатах удостоверяющего центра. Оно используется для ограничения путей проверки двумя возможными способами. Первый из них заключается в запрете отображения одной политики на другую, а второй — в том, чтобы каждый сертификат в пути валидации сертификата содержал допустимый идентификатор политики.

Управление сертификатами пользователей осуществляется на основе дополнений сертификата X509.

Ограничивающие дополнения (`Basic Constraints`, `Name Constraints`) применяются для удостоверяющих центров.

С помощью дополнения `keyUsage` осуществляется ограничение по применению ключа. При этом желательно использовать одну пару ключей для решения одной задачи для осуществления быстрого изъятия и перевыпуска в случае компрометации.

Политики применения сертификатов — набор правил, на основании которых удостоверяющий центр или прикладное программное обеспечение принимают решение о принятии сертификата.

Анализ показал, что отсутствуют средства выбора сертификата на основании контекста безопасности пользователя, хотя существующие решения могут быть использованы для реализации механизма.

Выводы по главе

1. Произведён обзор инфраструктуры открытых ключей, сертификата X509, дополнений сертификата;
2. Сертификат X509 версии 3 позволяет включать пользовательские дополнения;
3. Проанализированы современные подходы к выбору сертификата пользователя.

2 Разработка механизма автоматического выбора сертификата открытого ключа пользователя на основании его контекста безопасности

В данной главе разрабатывается алгоритм работы механизма автоматического выбора сертификата открытого ключа пользователя на основании его контекста безопасности, определяется средства реализации, проводится их обзор.

2.1 Алгоритм работы механизма

Разрабатываемый механизм должен выполнять следующий алгоритм:

1. Открытие сессии пользователя;
2. Проверяется наличие электронно-цифровой подписи пользователя. Если она была создана, выполняется переход к п. 4.
3. Создается цифровая подпись пользователя;
4. Создается закрытый ключ пользователя;
5. С помощью закрытого ключа создается запрос на подпись сертификата пользователя (CSR), содержащего текущий метку безопасности пользователя;
6. Запрос подписывается цифровой подписью пользователя.
7. Отправка подписанного CSR на удостоверяющий центр;
8. Удостоверяющий центр выполняет проверку цифровой подписи. В случае ошибки при проверке подписи, происходит незамедлительный останов;
9. Выпуск сертификата открытого ключа и пересылка сертификата пользователю;
10. При переходе пользователя на новый уровень безопасности, выполняются шаги 4-9 данного алгоритма.

Этот алгоритм представлен на *рисунке 2.1.*

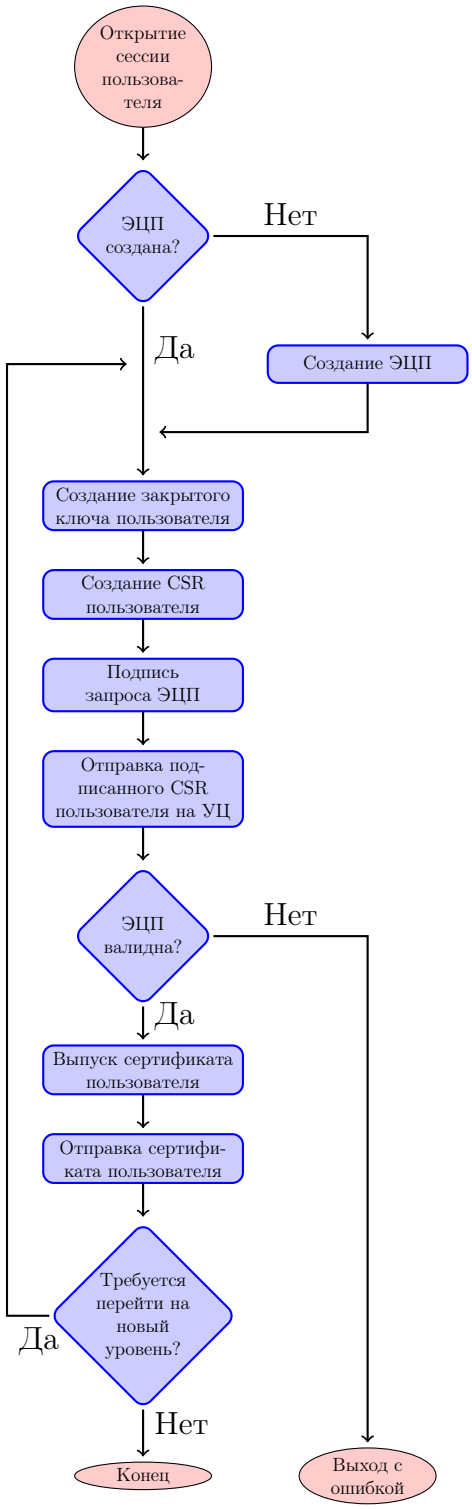

Рис. 2.1: Алгоритм работы разрабатываемого механизма

Предлагается хранить метку безопасности в дополнении сертификата X509. Каждому пользователю создаётся электронно-цифровая подпись, с помощью которой каждый запрос на подпись сертификата будет подписан этой подписью. Подпись будет проверена на удостоверяющем центре и при её валидности удостоверяющий центр выпускает сертификат.

Для обеспечения возможности выбора сертификата открытого ключа предлагается использовать механизм многоэкземплярности. С помощью данного механизма будет созданы внутри директории `/home` экземпляры домашней директории, в которых будут храниться созданные сертификаты согласно текущему уровню безопасности пользователя. При выполнении команды перехода на другой разрешённый уровень безопасности домашняя директория пользователя будет подменена соответствующим экземпляром.

2.2 SELinux

SELinux [8] — это расширение базовой модели безопасности операционной системы Linux, добавляющее механизм мандатного контроля доступа.

SELinux входит в официальное ядро Linux начиная с версии 2.6. Система разрабатывается Национальным агентством по безопасности США (*National Security Agency*, NSA) при сотрудничестве с другими исследовательскими лабораториями и коммерческими дистрибутивами Linux. Исходные тексты проекта доступны под лицензией GPL.

SELinux интегрирован в дистрибутивы, в частности, Fedora. Для функционирования в них поставляются модифицированные версии некоторых утилит (`ps`, `ls` и т.д.), которые поддерживают новые функции ядра и файловой системы.

В SELinux права доступа определяются самой системой при помощи специально определённых политик. Политики работают на уровне системных вызовов и применяются самим ядром. Все объекты (файлы, межпроцессные каналы связи, сокеты, сетевые хосты, и т.д.) и субъекты (процессы) имеют единую метку безопасности, связанную с ними. Метка безопасности состоит из четырёх элементов: пользователь, роль, идентификаторы типа и мандатные метки уровней и категорий. Обычный формат для задания или отображения метки выглядит следующим образом:

```
user:role:type:sensitivity:category
```

Рассмотрим подробнее эти объекты:

- `user` — сущность пользователя. Связывается с пользователем Linux и является неотъемлимой частью на протяжении всей пользовательской сессии;
- `role` — роль пользователя. Роль определяет, какие домены могут быть использованы. Домены, к которым имеет доступ пользовательская роль, предопределяются в конфигурационных файлах политики. Если роль не имеет доступа к заданному домену (в базе данных политики), то при попытке выполнить это действие доступ будет запрещён;
- `type` — домен для процессов или тип для объекта (файлы, директории, сокеты и т.п.). Домен определяет набор действий, которые может выполнять домен или объект;
- `sensitivity` — иерархический уровень объекта или уровень доступа субъекта. Данное поле имеет смысл в случае работы SELinux в режиме многоуровневой защиты;
- `category` — неиерахический категории, которые необходимы для разграничения доступа на одном уровне.

Для каждого объекта используются строковые идентификаторы. Метка безопасности должна содержать действующего пользователя, роль и идентификатор типа, причём идентификатор типа определяются в политике.

SELinux начинает работать после того, как дискреционный механизм контроля разрешил доступ субъекту к объекту.

2.2.1 Режимы работы SELinux

SELinux может работать в трёх режимах:

- `disabled` — полностью отключает подсистему обеспечения мандатного контроля доступа. При включении SELinux в любом режиме необходимо заново установить метки безопасности в файловой системе;
- `permissive` — разрешающий режим. Т. е. при данном режиме работы фиксируются попытки выполнения действий, противоречащих текущей политике безопасности в лог-

файл /var/log/audit/audit.log, однако фактического блокирования действий не происходит. Обычно данный режим применяется для отладки;

- enforcing — запрещающий режим. Т.е. в данном режиме будет применяться текущая политика. Все действия, противоречащие текущей политике безопасности, будут записываться в лог /var/log/audit/audit.log. При этом попытки выполнить данные действия будут блокироваться.

Режим SELinux описывается в поле SELINUX в конфигурационном файле /etc/selinux/config. Режим SELinux может изменяться с permissive на enforcing с помощью команды без перезагрузки операционной системы:

```
# setenforce 1
```

и переключаться на permissive с помощью команды:

```
# setenforce 0
```

При загрузке системы используется значение из конфигурационного файла.

2.2.2 Политики SELinux

Политики — это наборы правил, определяющие список ролей, к которым имеет доступ пользователь, какие роли имеют доступ к каким доменам и какие домены имеют доступ к каким типам.

Далее представлено типичное правило:

```
allow postfix_postdrop_t httpd_log_t:file getattr;
```

Правило означает следующее: домену postfix_postdrop_t разрешается производить действие getattr объекту file над типом httpd_log_t.

Выделяют несколько типов политик SELinux:

- target — целевая политика. Предназначена для защиты операционной систему от системных процессов, передающих и получающих сообщения через сетевые сервисы (например, NFS, DNS, HTTP). Используется по умолчанию;

- strict — строгая политика. Основана на целевой, в которой все действия, неописанные в политике по умолчанию запрещены;

- `mls` — многоуровневая политика. Политика MLS содержит не только правила, указывающие, какие объекты системы безопасности могут совершать определенные действия, и что они могут сделать, находясь на определенном уровне безопасности.

Текущий тип политики описывается в поле `SELINUXTYPE` в конфигурационном файле `/etc/selinux/config`.

При смене режима работы политики или типа политики требуется расстановка меток безопасности файловой системы. Это можно сделать с помощью создания пустого файла `.autorelabel` в корне файловой системы:

```
# touch /.autorelabel
```

Повторная расстановка меток безопасности будет произведена при следующем запуске системы.

2.2.3 Некоторые утилиты для работы с SELinux

Некоторые утилиты, используемые для работы с метками безопасности:

- `chcon` — позволяет сменить метку безопасности объекта файловой системы;
- `restorecon` — устанавливает метку безопасности файловой системы по умолчанию;
- `semanage` — используется для настройки некоторых элементов политики SELinux без необходимости модификации или повторной компиляции исходного текста политики;
- `sestatus` — выводит информацию о режиме работы, типе используемой политике и прочую информацию о работе SELinux;
- `setenforce` — позволяет изменить режим работы SELinux;
- модифицированные утилиты `ls`, `ps`, `id` и т.д. — используются для получения контекста безопасности файлов, папок, процессов и т.д.

2.3 Многоэкземплярность

Многоэкземплярность [9] — это концепция создания нескольких независимых копий для одного объекта.

В операционной системе Fedora она может быть применена для создания копий директорий по различным признаками: по имени пользователя, по уровню или контексту безопасности.

Создание многоэкземплярных директорий реализуется с помощью модуля `pam_namespace.so` PAM [10]. Во время входа пользователя в операционную систему модуль PAM создает в пределах системной многоэкземплярной директории свою личную, которую «видит» только пользователь. Он имеет право на чтение и запись как обычно. Однако, он не «видит» других экземпляров этой директории.

Механизм многоэкземплярности наглядно представлен *на рисунке 2.2*. На этом рисунке представлена структура директории `/home`, многоэкезмплярность которой настроена по методу `context`.

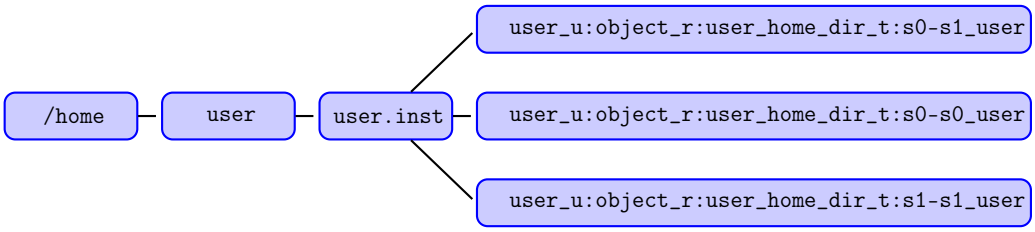

Рис. 2.2: Многоэкземплярность домашней директории пользователя `user`

Обладая определённым уровнем доступа, пользователь `user` в данном примере может видеть только один экземпляр, соответствующий его уровню.

Настройка многоэкземплярности в дистрибутивах `Fedora` предусматривает редактирование конфигурационных файлов:

- `/etc/security/namespace.conf` — файл, в котором указываются многоэкземплярные директории, которые создаст PAM при запуске системы;
- `/etc/security/namespace.init` — скрипт инициализации многоэкземплярных директорий;
- `/etc/pam.d/<deamon>` — конфигурационные файлы программ и служб, использующие для аутентификации PAM.

Синтаксис конфигурационного файла `/etc/security/namespace.conf` следующий:

```
polydir instance_prefix method list_of_uids
```

polydir — путь к многоэкземплярной директории;

instance_prefix — корневая директория для многоэкземплярной директории;

method — метод, с помощью которого создаются экземпляры, может принимать следующие значения: user, context, level;

list_of_uids — список пользователей, для которых PAM не будет создавать многоэкземплярные директории.

Далее приводится пример создания многоэкземплярных директорий /tmp и /var/tmp:

```
/tmp             /tmp/tmp-inst/          level   root
/var/tmp         /var/tmp/tmp-inst/      level   root
```

В данном случае создаются многоэкземплярные директории /tmp и /var/tmp, принцип создания которых основан на уровне пользователей для всех, за исключением суперпользователя root. В данном случае он может «видеть» директории /tmp/tmp-inst и /var/tmp/tmp-inst и все экземпляры данных папок, находящихся в них.

Далее требуется создать вручную эти многоэкземплярные директории:

```
# mkdir /tmp/tmp-inst /var/tmp/tmp-inst
```

Созданным директориям необходимо назначить права 000 и установить владельца директории root:

```
# chmod 000 /tmp/tmp-inst /var/tmp/tmp-inst
# chown root.root /tmp/tmp-inst /var/tmp/tmp-inst
```

Если какой-либо программе в операционной системе необходимо работать с многоэкземплярными директориями, в конфигурационный файл PAM этой программы добавляется строка:

```
session required        pam_namespace.so
```

2.3.1 Обзор модуля PAM pam_namespace

pam_namespace — динамически подключаемая библиотека, обеспечивающая создание многоэкземплярных директорий. Написана на языке программирования C.

В файле pam_namespace.h определяются основные структуры, используемые при реализации модуля, в честсноти, структура polydir_s, в которой хранится информация о многоэкземпляной директории, а также instance_data — структура, хранящая сведения об экземпляре многоэкземплярной директории.

Имя экземпляра директории формируется в функции poly_name:

```
static int poly_name(const struct polydir_s *polyptr, char **i_name,
security_context_t *i_context, security_context_t *origcon, struct
instance_data *idata)
```

В данной функции вызывается функция form_context, отвечающая за получение текущего контекста пользователя, необходимого для формирования имени директории:

```
static int form_context(const struct polydir_s *polyptr,
security_context_t *i_context, security_context_t *origcon, struct
instance_data *idata)
```

Вызов скрипта, инициализирующего экземпляр директории, происходит в функции

```
static int inst_init(const struct polydir_s *polyptr, const char *ipath
, struct instance_data *idata, int newdir)
```

в строке

```
if (execle(init_script, init_script, polyptr -> dir, ipath, newdir
?"1":"0", idata -> user, NULL, envp) < 0)
```

Типичный скрипт инициализации представлен на следующем листинге:

```
#!/bin/sh -p
if [ "$3" = 1 ]; then
    [ -x /sbin/restorecon ] && /sbin/restorecon "$1"
    user="$4"
    passwd=$(getent passwd "$user")
    homedir=$(echo "$passwd" | cut -f6 -d":")
    if [ "$1" = "$homedir" ]; then
        gid=$(echo "$passwd" | cut -f4 -d":")
        cp -rT /etc/skel "$homedir"
```

```
            chown -R "$user":"$gid" "$homedir"
            mask=$(awk '/^UMASK/{gsub("#.*$", "", $2); print $2; exit}' /
    etc/login.defs)
            mode=$(printf "%o" $((0777 & ~$mask)))
            chmod ${mode:-700} "$homedir"
            [ -x /sbin/restorecon ] && /sbin/restorecon -R "$homedir"
        fi
    fi
    exit 0
```

Данный скрипт принимает четыре параметра:

- `$1` — путь к многоэкземплярной директории;
- `$2` — путь к экземпляру директории;
- `$3` — флаг, равен 0, если не требуется создание экземпляра директории и 1, если требуется;
- `$4` — имя пользователя;

Данный скрипт реализует операции инициализации экземпляров многоэкземплярных директорий.

2.4 OpenSSL

OpenSSL [11] — это система защиты и сертификации данных (с англ. *«открытая система безопасных сокетов»*).

Ввиду того, что OpenSSL поддерживает очень много различных стандартов сертификации, шифрования, хеширования, то использование данной команды достаточно сложно.

Для удобства OpenSSL разделён на компоненты, которые отвечают за то или иное действие. Для получения списка доступных компонентов используется команда:

```
$ openssl list-message-digets-commands
```

Список доступных алгоритмов шифрования доступен с помощью команды:

```
$ openssl list-cipher-commands
```

Обычно OpenSSL используется для следующих операций:

- Создание и управление ключами;
- Создание запросов на подпись сертификатов;
- Создание сертификата открытых ключей;
- Осуществлять верификацию сертификатов;
- Создания удостоверяющих центров.

2.4.1 Создание удостоверяющего центра

Для создания одиночного удостоверяющего центра используется скрипт `CA`:

```
# /etc/pki/tls/misc/CA -newca
```

При выполнении данного скрипта в диалоговом режиме будет предложено ввести информацию об удостоверяющем центре:

```
Enter PEM pass phrase: <пароль>
Verifying - Enter PEM pass phrase: <пароль>
...
Country Name (2 letter code) [XX]:ru
State or Province Name (full name) []:msk
Locality Name (eg, city) [Default City]:msk
Organization Name (eg, complany) [Default Company Ltd]:mephi
Organizational Unit Name (eg, section) []:kaf36
Common Name (eg, your name or your server's hostname) []:kaf36's CA
Email Address []:root@kaf36
```

После выполнения данного скрипта будет развёрнут удостоверяющий центр, закрытый ключ которого будет сохранён в `/etc/pki/CA/private` и сертификат в `/etc/pki/CA/cacert.pem`.

2.4.2 Электронно-цифровая подпись

Электронная цифровая подпись (*ЭЦП*) [12] — реквизит электронного документа, позволяющий установить отсутствие искажения информации в электронном документе с момента формирования ЭЦП и проверить принадлежность подписи владельцу сертификата ключа ЭЦП. Значение реквизита получается в результате криптографического преобразования информации с использованием закрытого ключа ЭЦП.

ЭЦП обычно используется для идентификации лица, подписавшего электронный документ.

Использование электронной подписи позволяет осуществить:

- Контроль целостности передаваемого документа: при любом случайном или преднамеренном изменении документа подпись станет недействительной, потому что вычислена она на основании исходного состояния документа и соответствует лишь ему;
- Защиту от изменений (подделки) документа: гарантия выявления подделки при контроле целостности делает подделывание нецелесообразным в большинстве случаев;
- Невозможность отказа от авторства. Так как создать корректную подпись можно, лишь зная закрытый ключ, а он известен только владельцу, он не может отказаться от своей подписи под документом;
- Доказательное подтверждение авторства документа: Так как создать корректную подпись можно, лишь зная закрытый ключ, а он известен только владельцу, он может доказать своё авторство подписи под документом.

Поскольку подписываемые документы обычно переменной или даже большой длины, подписываются не сами документы, а их хэш.

- Обычные цифровые подписи (присоединённые);
- Цифровые подписи с восстановлением документа.

Присоединённые электронно-цифровая представляет собой подписанный хэш файла, хранящийся в отдельном файле. Для её проверки необходимо иметь файл, который был подписан этой подписью.

Цифровые подписи с восстановлением документа содержат в себе подписываемый документ: в процессе проверки подписи автоматически вычисляется и внедряется в тело документа.

2.4.3 Создание сертификата открытого ключа

При создании закрытого ключа, запроса на подпись сертификата или сертификата, по умолчанию используется конфигурационный файл `openssl.conf`. Он разделён на секции следующего вида:

```
[ section_name ]
name = value
```

где:

- `section_name` — имя секции;
- `name` — имя опции;
- `value` — значение опции.

Для создания закрытого ключа используется команда `genrsa`, синтаксис которой следующий:

```
$ openssl genrsa [-out file] [-des | -des3 | -idea] [bits]
```

Команда `genrsa` создает закрытый ключ длиной `bits` в формате `PEM`, шифрует его одним из алгоритмов: `des` (56 бит), `des3` (168 бит) или `idea` (128 бит).

При выборе алгоритма шифрования будет запрошен пароль для шифрования создаваемого закрытого ключа.

Опция `-out` говорит программе, что вывод нужно осуществлять не в `stdout`, а в файл `file`. Например, на следующем листинге приведён пример генерации закрытого ключа длиной 4096 бит с использованием алгоритма шифрования `des3`:

```
$ openssl genrsa -out ~/mykey.pem -des3 4096
Generating RSA private key
.....++*...++++++++*
```

```
Enter PEM passphrase:
Verify PEM passphrase:
```

При создании ключа таким образом требуется ввести пароль, после чего его повторить. Созданный ключ будет сохранён в домашней папке пользователя в файле `mykey.pem`.

Для создания запроса на подпись сертификата, используется команда `openssl req`. Она позволяет в диалоговом режиме или с использованием предопределённых параметров конфигурационного файла создать запрос на подпись сертификата. Данная команда требует указания пути закрытого ключа соответствующей опцией. При создании запроса на подпись сертификата по умолчанию используется конфигурационный файл `/etc/pki/tls/openssl.conf`.

Для создания запроса на подпись сертификата может быть использована команда:

```
$ openssl req -new -key mykey.key -out mycert.csr
```

Удостоверяющий центр выпускает сертификат по запросу на подпись сертификата:

```
# openssl x509 -req -days 365 -in mycert.csr -extfile /etc/pki/tls/
openssl.cnf -extensions v3_req -out mycert.crt
```

С помощью представленной выше команды будет выпущен сертификат `mycert.crt`, действительный в течение года (`-days 365`). В сертификат могут быть добавлены дополнения сертификата, указанные в секции `v3_req` опцией `-extensions` из конфигурационного файла `/etc/pki/tls/openssl.conf`. По умолчанию в сертификат добавляются следующие дополнения:

```
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
```

Дополнение `basicConstraints` указывает о невозможности использования сертификата в качестве сертификата удостоверяющего центра, `keyUsage` — политику использования ключа.

2.4.4 Создание пользовательского дополнения сертификата

Стандарт сертификатов X509v3, как это было описано ранее, позволяет включать пользовательские дополнения в сертификат.

Для реализации этой возможности могут быть использованы следующие варианты:

- С помощью модифицированного конфигурационного файла;
- Программно:
  - с помощью псевдонима на уже существующее дополнение (`alias`);
  - с помощью реализации структуры дополнения.

Первый вариант предполагает модификацию конфигурационного файла, в котором будут дополнительно объявлены идентификатор пользовательского дополнения `OID`, а также его короткое и длинное имена. При этом для идентификации данного дополнения требуется поставлять модифицированный конфигурационный файл в `openssl`.

Достоинства данного подхода — не требует расширения базового функционала OpenSSL. Недостатки — необходимость поставки модифицированного конфигурационного файла на все машины, на которых будут проводиться операции с сертификатами.

Другим вариантом является программная реализация дополнения.

Для создания пользовательского дополнения с помощью объявления `alias` на существующее дополнение может быть использован следующий код (написан на языке программирования C):

```
int nid;
nid = OBJ_create("1.2.3.4", "mephiAlias", "My Mephi");
X509V3_EXT_add_alias(nid, exsisted_nid);
```

В данном листинге объявляется переменная `nid`, являющая порядковым идентификатором создаваемого дополнения. С помощью функции `OBJ_create()` создается объект расширения. Первым параметром передается `OID`, во втором — короткое имя, в последнем — длинное. В последней строке выполняется создание псевдонима на объект дополнения, идентификатор которого указан в переменной `extension_nid`. Для инициализации, изменения или удаления нового дополнения используются соответствующие функции существующего дополнения.

Достоинства данного подхода — простота программной реализации. Недостатки — для успешной идентификации созданного дополнения в конфигурационном файле OpenSSL требуется добавить строчки, определяющее OID объекта и его имя, а также требуется вызывать специальные функции для получения значения дополнения.

Если используется программная реализация структуры дополнения, то требуется модифицировать файл `objects.txt`, добавив туда информацию о дополнении, а также требуется реализовать функции по конвертации из `ASN1` в текстовое представления и обратно.

Недостатками данного подхода является сложность реализации. В качестве достоинств данного подхода выделяются: отсутствие необходимости поставки идентификатора объекта и его текстового представления в конфигурационных файлах, поэтому не требуется вызывать специальные функции библиотеки OpenSSL при различных манипуляциях с дополнением. В этом случае идентификация объекта в сертификате выполняется внутри библиотеки без использования конфигурационных файлов.

2.4.5 Устройство дополнения сертификата

Соответствие между кодом объекта в аннотации ASN1 и его текстового представления описывается в файле `objects.txt`.

Синтаксис этого файла следующий:

```
1 2 3 4          : shortName     : Long Name
```

`1 2 3 4` обозначает числовой код объекта в спецификации ASN1, `shortName` — короткое имя объекта (например `SN`, `C`, `ST`), `Long Name` — полное имя (например, `surname`, `countryName`, `stateOrProvinceName`).

Например, ниже приведено определение объекта `SubjectKeyIdentifier` в файле `objects.txt`:

```
!Cname subject-key-identifier
id-ce 14 : subjectKeyIdentifier : X509v3 Subject Key Identifier
```

Идентификатор объекта `SubjectKeyIdentifier` использует объявленный ранее в файле `OID id-ce`. С помощью директивы `!Cname` явно определяется имя объекта в библиотеке OpenSSL.

Чтобы можно было использовать данную информацию о дополнениях непосредственно в OpenSSL, используется скрипт `objects.pl`. Он принимает на вход файл `objects.txt` и создает с помощью директив `#define` определение числового кода в ASN1 объекта и его имени, а также его порядковый идентификатор или `NID`. Эти объявления можно найти в

файле objects.h. На следующем листинге показан результат работы данного скрипта для объекта SubjectKeyIdentifier:

```
#define SN_subject_key_identifier       "subjectKeyIdentifier"
#define LN_subject_key_identifier       "X509v3 Subject Key Identifier"
#define NID_subject_key_identifier      82
#define OBJ_subject_key_identifier      OBJ_id_ce,14L
```

Такие определения создаются для всех объектов.

Для обеспечения информации, хранимой в дополнении, требуется реализовать структуру X509V3_EXT_METHOD:

```
typedef struct v3_ext_method X509V3_EXT_METHOD;

struct v3_ext_method {
int ext_nid;
int ext_flags;
ASN1_ITEM_EXP *it;
X509V3_EXT_NEW ext_new;
X509V3_EXT_FREE ext_free;
X509V3_EXT_D2I d2i;
X509V3_EXT_I2D i2d;

X509V3_EXT_I2S i2s;
X509V3_EXT_S2I s2i;

X509V3_EXT_I2V i2v;
X509V3_EXT_V2I v2i;

X509V3_EXT_I2R i2r;
X509V3_EXT_R2I r2i;

void *usr_data;
};
```

Данная структура объявлена в заголовочном файле `x509v3.h`. В структуре объявляется поле типа `int ext_nid`, которое хранит порядковый идентификатор дополнения, в поле `ext_flags` указывается тип дополнения (`0` — однострочный, `1` — многострочный). В поле `it` определяется функция создания и удаления дополнения. Если она определена, то поля `ext_new`, `ext_free`, `d2i`, `i2d` не требуют определения. Они используются для инициализации и корректного удаления объекта дополнения в старом стиле.

Далее определяются функции, выполняющие кодирование и декодирование объекта из ASN1 в текстовый и обратно. Поля `i2s`, `s2i` определяют функции преобразования из ASN1 в строку и обратно; `v2i`, `i2v` определяют функции преобразования дополнений, в которых содержится несколько полей; `i2r`, `r2i` — функции преобразования многострочных дополнений.

В последнем поле может быть объявлена дополнительная функция, которая может потребоваться для реализации дополнения.

Поиск дополнений выполняется в таблице `standart_ext`, объявленной в заголовочном файле `ext_dat.h`:

```
static const X509V3_EXT_METHOD *standard_exts[]
```

При этом все структуры дополнений предварительно загружаются с помощью ключевого слова `extern` языка C. Данная таблица содержит адреса структур всех доступных в конкретной версии OpenSSL дополнений.

2.5 Разработка утилиты создания сертификатов

Утилита для создания сертификатов X509 должна удовлетворять следующим требованиям:

1. Возможность создавать закрытый ключ пользователя произвольной длины;
2. Создавать запросы на подпись сертификатов с пользовательским дополнением, в котором будет храниться контекст безопасности пользователя;
3. Создавать ЭЦП, а также выполнять подпись и её проверку;
4. Подписывать запрос с помощью удостоверяющего центра.

Возможности библиотеки OpenSSL и её одноимённой консольной утилиты `openssl` удовлетворяют данным требованиям. Однако для включения дополнений в сертификат необходимо предопределённое значение дополнения, которое указано в конфигурационном файле `openssl.conf`. Поэтому данный конфигурационный файл должен быть редактирован при каждом создании сертификата.

Предлагается разработать утилиту, реализующая все перечисленные выше требования, которая не будет использовать конфигурационный файл `openssl.conf`. В дополнении сертификата будет содержаться значение контекста безопасности пользователя, установить которое можно как вручную, так и с помощью системных средств. Кроме того, каждый запрос на подпись сертификата будет подписан электронно-цифровой подписью пользователя, выполнение проверки которой будет выполняться на удостоверяющем центре.

Данную утилиту предлагается реализовать на языке программирования Python с целью повышения скорости разработки. Вопросы оптимизации при разработке утилиты не рассматриваются.

На языке программирования Python существует несколько библиотек, позволяющие создавать сертификаты X509, такие как:

- PyOpenSSL;
- M2Crypto.

PyOpenSSL [13]— это интерфейс к библиотеке OpenSSL. Текущая версия — 0.14. Недостатками данной библиотеки являются неполная реализация функций OpenSSL, неполная документация и сложность модификации. Достоинством данной библиотеки является малый размер библиотеки.

M2Crypto [14] также является интерфейсом к библиотеке OpenSSL, текущая версия — 0.21. В качестве основных недостатков можно выделить больший размер M2Crypto по сравнению с PyOpenSSL. Достоинствами являются хорошая документация всех функций наиболее полная реализация функций OpenSSL, а также возможность расширения с помощью низкоуровневого API.

Поэтому для разработки утилиты создания X509 сертификатов предлагается использовать библиотеку M2Crypto.

2.5.1 Обзор структуры библиотеки M2Crypto

Библиотека M2Crypto состоит из следующих подмодулей:

- Подмодуль `M2Crypto.m2` представляет собой низкоуровневый интерфейс, который генерируется автоматически с помощью утилиты `SWIG` [15]. Данная утилита создает динамическую подключаемую библиотеку с вызовами функций библиотеки OpenSSL, которые можно использовать при создании высокоуровнего API. С помощью особых файлов спецификаций, используемых утилитой `SWIG` можно расширить библиотеку для собственных нужд.

- Подмодуль `M2Crypto.RSA` содержит класс RSA-ключей. Методы данного класса реализуют создание ключей и сохранение в различных форматах.

- Подмодуль `M2Crypto.ASN1` необходим для создания ASN1 объектов сертификата и для их идентификации.

- Подмодуль `M2Crypto.X509` содержит в классы, предоставляющие возможность работы с объектами сертификата: созданием сертификата, дополнениями, стека дополнений и т.д.

- Подмодуль `M2Crypto.BIO` содержит класс, позволяющий загружать сертификаты в память, а также выполнять различные действия с ними.

- Подмодуль `M2Crypto.SMIME` содержит класс `SMIME`, позволяющий подписывать создавать подписи с восстановлением документа, а также их проверять.

Выводы по главе

1. Произведён обзор мандатной системы контроля доступа SELinux;

2. Модуль `pam_namespace` в существующей реализации не предполагает передачу скрипту инициализации многоэкземплярных директорий `namespace.init` текущего контекста безопасности пользователя;

3. OpenSSL предоставляет несколько вариантов добавления дополнений X509v3 в сертификат, наиболее предпочтительным из которых является программная реализация структуры дополнения;

4. Предлагается создать специализированную утилиту генерации сертификатов с меткой безопасности пользователя, которая будет храниться в дополнении сертификата, на языке программирования Python;

5. Была выбрана библиотека M2Crypto для создания утилиты генерации сертификатов.

3 Реализация механизма автоматического выбора сертификата открытого ключа пользователя на основании его контекста безопасности

В данной главе осуществляется доработка выбранных средств реализации механизма автоматического выбора сертификата открытого ключа пользователя, выполняется настройка механизма на стенде из двух машин. Работоспособность данного механизма показана на примере СУБД PostgreSQL, выполняется тестирование механизма.

3.1 Разработка компонентов механизма

3.1.1 Модификация pam_namespace.so

Текущий уровень безопасности пользователя можно получить из названия экземпляра директории, однако, получить полный контекст безопасности пользователя весьма затруднительно.

Поэтому предлагается объявить расширить структуру `polydir_s` дополнительным полем `secontext`, в котором будет храниться текущее значение контекста безопасности пользователя. Значение данного поля будет передаваться в скрипт инициализации `namespace.init`. Таким образом, расширенная структура `polydir_s` будет иметь следующий вид:

```
struct polydir_s {
    char dir[PATH_MAX];
    char rdir[PATH_MAX];
    char instance_prefix[PATH_MAX];
    enum polymethod method;
    unsigned int num_uids;
    uid_t *uid;
    unsigned int flags;
    char *init_script;
    char *mount_opts;
    uid_t owner;
```

```
    gid_t group;
    mode_t mode;
    char *secontext;
    struct polydir_s *next;
};
```

Получение контекста пользователя представлено на следующем листинге:

```
FILE *fp;
size_t size = 0;
fp = popen("id -Z", "r");
if (NULL == fp) {
    pam_syslog(idata -> pamh, LOG_ERR, "Command ‘id -Z‘ return error");
        return PAM_SESSION_ERR;
}
int length = getline(&polyptr -> secontext, &size, fp);
if (length < 0) {
    pam_syslog(idata -> pamh, LOG_ERR, "Can’t get selinux context");
    return PAM_SESSION_ERR;
}
polyptr -> secontext[length - 1] = ’\0’;
pclose(fp);
```

Функцией `popen()` открывает новый процесс, в котором вызывается системная команда получения контекста пользователя `id -Z`. Вторым аргументом функции указывается тип создаваемого канала. Так как необходимо получить результат выполнения команды, используется режим чтения `’r’`. Функция `popen()` возвращает объект потока `FILE*` в переменной `fp`.

Далее результат вызова проверяется на равенство `NULL`. В случае равенства этой переменной значению `NULL` в лог-файл PAM пишется текст ошибки и возвращается `PAM_SESSION_ERR`.

Иначе с помощью функции `getline()` выполняется запись из потока в поле `secontext` структуры директории `polyptr`:

```
int length = getline(&polyptr -> secontext, &size, fp);
```

Функция `getline()` записывает из потока `fp` в строку `polydir->secontext`, строку длиной `size` байт. Возвращаемое значение, хранящееся в переменной `length` — это количество символов, записанных в строку. В случае ошибки возвращается значение `-1`.

Обработка возвращаемого значения осуществляется в блоке

```
if (result < 0) {
    pam_syslog(idata -> pamh, LOG_ERR, "Can't get selinux context");
    return PAM_SESSION_ERR;
}
```

Иначе, в строку `polyptr -> secontext` предпоследним символом устанавливается знак окончания строки `'\0'`, после чего функцией `pclose()` закрывается поток.

В функции `inst_init()` изменяется строка с вызовом скрипта:

```
if (execle(init_script, init_script, polyptr->dir, ipath, newdir
?"1":"0", idata->user, polyptr->secontext, NULL, envp) < 0)
```

Так как значение поле `secontext` хранится в динамической памяти, то требуется очистка памяти, занимаемой этим полем. Поэтому добавляется строка

```
free(poly->secontext);
```

в функцию `del_polydir()`.

Так был реализован патч, с которым подробно можно ознакомиться в *приложении 2*.

После наложения патча на модуль `pam_namespace` и его пересборки, скрипту инициализации `namespace.init` будет передан в параметре `$5` текущий контекст безопасности пользователя.

3.1.2 Реализация дополнения сертификата selinuxContext в OpenSSL

С помощью описанной в *пункте 2.4.5* было реализовано однострочное дополнение X509v3 на языке программирования C, которое позволяет хранить контекст пользователя. Полный текст дополнения приводится в *приложении 3*.

С помощью директив `#include` подключаются заголовочные файлы, необходимые для реализации дополнения. Далее объявляются функции, необходимые для кодирования и декодирования объекта ASN1 в текстовый и обратно:

```
    static char *i2s_ASN1_IA5STRING(X509V3_EXT_METHOD *method,
ASN1_IA5STRING *asn1_string);
    static ASN1_IA5STRING *s2i_ASN1_IA5STRING(X509V3_EXT_METHOD *method,
X509V3_CTX *ctx, char *string);
```

Данные функции определены с ключевым словом `static`, что гарантирует их видимость в данном файле и исключает возможность обращения из другого файла. Первая функция возвращает текстовое значение поля дополнения. Её параметры — указатель на структуру дополнения `method` и указатель на строку в формате ASN1 `asn1_string`, декодирование которой необходимо выполнить. Вторая функция возвращает кодированную в ASN1 строку. Ей передаются: указатель на структуру дополнения `method`, указатель на структуру контекста `ctx`, а также указатель на текстовую строку `string`.

Далее описывается определение структуры `v3_secon`:

```
const X509V3_EXT_METHOD v3_secon = EXT_IA5STRING(NID_selinux_context);
```

`EXT_IA5STRING` — это макрос, объявленный в заголовочном файле `x509v3.h` следующим образом:

```
{ nid, 0, ASN1_ITEM_ref(ASN1_IA5STRING), \
0,0,0,0, \
(X509V3_EXT_I2S)i2s_ASN1_IA5STRING, \
(X509V3_EXT_S2I)s2i_ASN1_IA5STRING, \
0,0,0,0, \
NULL}
```

Таким образом он реализует структуру `X509V3_EXT_METHOD` для однострочных дополнений x509v3. Необходимо реализовать функции конверации `i2s_ASN1_IA5STRING` (преобразование в ASN1-строки в текстовую) и `s2i_ASN1_IA5STRING` (преобразование из текстовой строки в ASN1-строку.

`ASN1_IA5STRING` — это `typedef` структуры `asn1_string_st`. Её объявление находится в заголовочном файле `asn1.h`.

В функции `i2s_ASN1_IA5STRING()` объявляется переменная `string`, в которой будет храниться результат конвертирования ASN1 строки в текстовую. При этом проверяются условия на корректность переданного значения ASN1-строки в функцию:

```
if ((NULL == asn1_string) || (NULL == asn1_string -> length))
```

Если передан нулевой указатель на ASN1-строку или её длина равна нулю, функцией возвращается `NULL`.

Далее предпринимается попытка выделить память для текстовой строки:

```
if (NULL == (string = OPENSSL_malloc(asn1_string -> length + 1)))
```

При этом следует отметить, что длина текстовой строки должна быть на 1 больше, чем ASN1. Это объясняется тем, что в C-строке последний символ должен быть обязательно завершающим (’\0’). Если память выделить для текстовой строки не удалось, также возвращается `NULL`.

Далее копируется данные из ASN1-строки в текстовую, последний символ устанавливается ’\0’:

```
memcpy(string, asn1_string -> data, asn1_string -> length);
string[asn1_string -> length] = 0;
```

После выполнения данных операций, функцией `i2s_ASN1_IA5STRING()` возвращается значение переменной `string`.

При реализации обратной конвертации текстовой строки в ASN1-строку выполняются аналогичная последовательность действий, но уже применительно к текстовой строке.

Для обеспечения возможности создания дополнения `v3_secon` был реализован патч, ознакомиться можно также в *приложении 2*. В нём была расширена таблица дополнений `standart_exts` и добавлена правила сборки для `v3_secon` в `Makefile`, а также объявлен объект `selinuxContext` в `objects.txt`.

Для того, чтобы применить разработанные дополнения, необходимо наложить патч на `openssl` и пересобрать её.

Таким образом, стало возможным создать пользовательское дополнение `selinuxContext` программно, причём его создание реализовано с использованием собственной структуры дополнения.

3.1.3 Расширение функционала библиотеки M2Crypto

При генерации сертификатов часто встречается задача извлечения дополнений из запроса на подпись сертификата для установки их в сертификат. При этом функционал библиотеки M2Crypto не позволяет это сделать в текущей реализации.

Поэтому предлагается расширить класс `X509_Request` соответствующими методами.

Для получения объекта расширения `X509_Extension`, была написана функция `get_extension_by_name(self, name)`:

```
def get_extension_by_name(self, name):
  request_stack = m2.x509_req_get_ext(self.req)
  extension_count = m2.sk_x509_extension_num(request_stack)
  for i in range(0, extension_count):
    ext_ptr = m2.sk_x509_extension_value(request_stack, i)
    extension = X509_Extension(ext_ptr)
    if extension.get_name() == name:
      return extension
  return None
```

При этом с использованием утилиты `SWIG` был произведено добавление функции в библиотеку. Для этого в файле `_X509.i`, в котором содержатся все функции модуля `M2Crypto.X509` была импортирована функция OpenSSL `X509_REQ_get_extensions()`:

```
%inline {
...
STACK_OF(X509_EXTENSION) *x509_req_get_ext(X509_REQ *request) {
    return X509_REQ_get_extensions(request);
}
...
%}
```

Реализованный метод `get_extension_by_name()` принимает два аргумента: указатель на сам объект `self` и `name` — имя дополнения. С помощью добавленной функции реализуется получение стека дополнений в переменной `request_stack`, далее подсчитывается число дополнений в стеке. На каждой итерации имя дополнения сравнивается с переданным в метод

именем желаемого дополнения `name`. Если дополнение было найдено, то оно возвращается, если после прохода всех элементов не было найдено дополнения с именем `name`, то возвращается `None`.

Данные модификации библиотеки оформлены в виде патча. Он приведён в *приложении* 1. Кроме реализации указанного метода были реализованы методы, позволяющие получить стек дополнений `get_extebsions()` (объект класса `X509_Extension_Stack`), а также количество дополнений в сертификате `get_extensions_count()`. Реализованные методы могут быть использованы при разработке иных приложений с использованием библиотеки M2Crypto.

3.1.4 Реализация утилиты создания сертификатов

Утилита `pgcert` исполняется интерпретатором Python версии 2.7. Ознакомиться с исходным текстом можно в *приложении* 4. Скриншот запуска приведён *на рисунке 3.1.*

Её исполнение начинается с точки входа:

```
if __name__ == "__main__"
```

Для удобного отображения опций командной строке утилиты, используется класс `OptionParser`. Он позволяет создать парсер опций командной строки и передавать функциям на исполнение. Создание объекта `OptionParser` реализовано с помощью:

```
  parser = OptionParser(usage="usage: %prog [Main Options] options",
add_help_option=True, description="This program use M2Crypto library and can
 generate X509 certificate with X509v3 extension SELinux Context")
```

В качестве параметров передаются значения: `add_help_options=True` — создать страницу помощи, а в поле `description` указывается описание утилиты. В поле `usage` описывается правила использования утилиты.

```
[dimv36@dimv36 ~]$ pgcert
Usage: pgcert [Main Options] options

This program use M2Crypto library and can generate X509 certificate with
X509v3 extension SELinux Context

Options:
  -h, --help            show this help message and exit

  Main Options:
    --genkey            generate private key
    --genreq            generate certificate request
    --gencert           generate certificate for user
    --sign              sign request by user's digital signature
    --verify            verify signature of request by user digital signature

  Private key options:
    --bits=BITS         set length of private key, default: 2048

  Request options:
    --user=USER         set CN of request, default: dimv36
    --secontext=SECONTEXT
                        add SELinux context to request
    --critical          set critical of selinuxContext extension, default:
                        False

  Certificate options:
    --signature         add extension keyUsage with value 'Digital signature'
                        to certificate, default: False

  Input options:
    --pkey=PKEY         set location of private key
    --request=REQUEST   set location of certificate request
    --certificate=CERTIFICATE
                        set location of certificate
    --cakey=CAKEY       set location of ca private key, default:
                        /etc/pki/CA/private/cakey.pem
    --cacert=CACERT     set location of ca certificate, default:
                        /etc/pki/CA/cacert.pem

  Output options:
    --output=OUTPUT     save to file
    --text              print request or certificate

  Info options:
    --issuer            get issuer of certificate
    --subject           get subject of certificate
    --extension=EXTENSION
                        get extension of certificate
[dimv36@dimv36 ~]$
```

Рис. 3.1: Запуск утилиты `pgcert`

Чтобы разделить опции на логические группы, используется класс `OptionGroup`. При его создании нужно передать объект парсера, а также заголовок группы. Например, с помощью следующей строки создается группа `Main Options`:

```
main_options = OptionGroup(parser, "Main Options")
```

Чтобы отобразить группу в справочной странице, необходимо вызвать метод `add_group_option()`:

```
parser.add_group_option(main_options)
```

Чтобы добавить опцию в группу, используется метод `add_option()`:

```
main_options.add_option("--genkey", dest="genkey", action="store_true",
default=False, help="generate private key")
```

Первым параметром указывается имя опции, аргумент которого будет сохраняться в поле, указанной в переменной `dest`. В поле `actions` указывается действие при парсинге опций. В данном случае при наборе данной опции в поле `dest` будет храниться значение `True`. Кроме того, можно добавить значение опции по умолчанию с помощью поля `default`. В поле `help` указывается, что реализует данная опция. Дополнительно может быть указан параметр `type`. В нём указывается ожидаемый тип значения аргумента.

С помощью метода `parse_args()` выполняется парсинг аргументов командной строки, переданные скрипту значения, которые хранятся в полях переменной `options`:

```
options, args = parser.parse_args()
```

После чего выполняется проверка на корректность переданных параметров командной строки и их параметров. Если данный набор не удовлетворяет ни одному условию, выводится страница помощи.

Вспомогательные функции утилиты

В утилите объявлены следующие константы:

- `DEFAULT_FIELDS` — словарь, хранящий значение субъекта по умолчанию;
- `CAKEY` — путь к закрытому ключу удостоверяющего центра по умолчанию;
- `CACERT` — путь к сертификату удостоверяющего центра по умолчанию;
- `DIGITAL_SIGNATURE_KEY` — путь к приватному ключу ЭЦП пользователя по умолчанию;
- `DIGITAL_SIGNATURE_CERT` — путь к сертификату ЭЦП пользователя по умолчанию;
- `DEFAULT_PASSWORD` — пароль, используемый при создании закрытого ключа.

Для реализации основных функций утилиты были созданы дополнительные функции:

- `password(*args, **kwargs)` — необходима, чтобы реализовать генерацию закрытого ключа по парольной фразе, ввод которой реализован в этой функции;
- `check_selinux_context(context)` — проверяет корректность контекста безопасности пользователя при создании запроса на подпись сертификата;
- `make_level_and_category_sets(context)` — создает список множеств допустимых уровней и категорий по переданному контексту;
- `verify_user_context(user, current_context)` — проверяет, является ли текущий контекст пользователя `current_context` допустимым при выпуске сертификата пользователя;

Создание закрытого ключа

Создание закрытого ключа реализуется в функции `make_private_key(bits, output)`. Аргументы функции:

- `bits` — длина ключа
- `output` — путь к файлу, в который будет сохранён ключ.

Создание ключа реализуется с помощью метода `gen_key()` модуля `RSA`:

```
private_key = RSA.gen_key(bits, 65537, callback=password)
```

Вторым параметром передается экспонента — простое число, используемое при создании шифра парольной фразы в алгоритме RSA. Для автоматического ввода пароля используется функция `password()`, описание которой приведено ниже.

Для сохранения ключа на жесткий диск используется метод `save()`:

```
private_key.save_key(output, None)
```

Функция допускает сохранение ключа по «альтернативному» пути, в случае, если функции передано пустое значение в переменной `output`. В таком случае путь файла, в который сохраняется ключ генерируется на основе пути текущей директории и имени ключа `mykey.pem`.

Создание запроса на подпись сертификата

Создание запроса на подпись сертификата реализовано в функции `make_request(private_key_path, username, user_context, critical, output, is_printed)`.

Аргументы функции:

- `private_key_path` — путь к закрытому ключу пользователя;
- `username` — имя владельца сертификата;
- `user_context` — контекст безопасности пользователя;
- `critical` — флаг, определяющий критичность дополнения `selinuxContext`. Равен `False`, если дополнение не является критичным и `True`, если является;
- `output` — путь к файлу, в который будет сохранён запрос;
- `is_printed` — флаг, определяющий, необходимо ли распечатать запрос после его создания. Принимает два значения — `True` (будет произведена печать запроса) или `False` (если не будет).

Работа данной функции начинается с попытки загрузки закрытого ключа пользователя:

```
private_key = None
try:
        private_key = RSA.load_key(private_key_path, callback=password)
except (RSA.RSAError, IOError):
        print('ERROR request: Could not load key pair from %s' %
   private_key_path)
    exit(1)
```

Использование механизма исключений позволяет отловить ошибку, если загрузить приватный ключ не удалось.

Далее создается объект запроса на подпись сертификата `X509.X509_Request`:

```
request = X509.Request()
```

По загруженному закрытому ключу устанавливается открытый ключ:

```
request.set_pubkey(private_key)
```

Субъект сертификата устанавливается автоматически по значениям полей переменной DEFAULT_FIELDS. Значение поля CN субъекта корректируется значением переменной username, если оно непустое:

```
if username:
        name.CN = username
```

Если в переменной user_context не был передан в переменной user_context, то выполняется вызов функции SELinux getcon_raw(). Функция getcon_raw() возвращает список, состоящий из результата вызова (0, если вызов функции успешен) и значение контекста. Так как требуется получить значение контекста, берётся 2й элемент списка. Это значение хранится в переменной context. Если же после выполнения этой функции в переменной context не содержится значение, то выводится ошибка на экран и прекращается дальнейшая работа:

```
if user_context:
        context = user_context
    else:
        context = getcon_raw()[1]
    if not context:
        print('ERROR request: Could not get SELinux context for user %s
   ' % username)
        exit(1)
```

Создание дополнений реализуется посредством создания объекта X509_Extension_Stack и добавления дополнения в него:

```
stack = X509.X509_Extension_Stack()
stack.push(X509.new_extension("selinuxContext", context, int(critical))
)
```

При создании объекта дополнения передается его имя, значение и флаг критичности.

Для добавления стека дополнений в запрос:

```
request.add_extensions(stack)
```

Объект запроса подписывается закрытым ключом пользователя и сохраняется по пути `output`. Аналогично функции `make_private_key()` допускается «альтернативное» сохранение запроса на подпись сертификата. Если функции был передан флаг `is_printed` со значением `True`, выполняется печать запроса.

Подпись запроса ЭЦП

Для того, чтобы удостоверяющий центр мог проверить, что запрос на подпись сертификата действительно пришёл от пользователя, пользователь выполняет подпись ЭЦП CSR, после чего подписанный запрос отправляется на удостоверяющий центр.

В данной утилите подпись запроса ЭЦП реализована в функции `sign(private_key_path, certificate_path, request_path)`.

Аргументы функции:

- `private_key_path` — путь к приватному ключу ЭЦП пользователя;
- `certificate_path` — путь к сертификату ЭЦП пользователя;
- `request_path` — путь к запросу на подпись сертификата;

Выполняется проверка на использование пары ключей для подписи запроса:

```
    if not get_extension(certificate_path, 'keyUsage') == 'Digital
Signature':
            print('ERROR sign: key pair %s and %s could not be used for
       signing file because policy' % (private_key_path, certificate_path))
        exit(1)
```

Если значение дополнения `keyUsage` не совпадает с `Digital Signature`, т.е. сертификат и закрытый ключ не предназначены для подписи запросов, то выводится сообщение и происходит выход из функции с кодом.

Выполняется загрузка запроса из файла. При этом с помощью механизма исключений осуществляется обработка ошибочных ситуаций:

```
    request = None
    try:
```

```
        request = X509.load_request(request_path)
except (IOError, X509.X509Error):
    print('ERROR sign: Could not load request from %s' % request_path)
        exit(1)
```

Аналогично выполняется загрузка из файла закрытого ключа, с помощью которого будет осуществлена подпись:

```
try:
        private_key = RSA.load_key(private_key_path, password)
except (IOError, RSA.RSAError):
        print('ERROR sign: Could not load private key')
    exit(1)
```

На основе текста запроса создаётся присоединённая подпись. Для этого создаётся объект класса `SMIME`, текст запроса загружается в память с помощью модуля `BIO`:

```
text = BIO.MemoryBuffer(request.as_pem())
smime = SMIME.SMIME()
```

Чтобы иметь возможность подписывать файлы цифровой подписи, необходимо загрузить закрытый ключ и сертификат. Это становится возможным с помощью метода `load_key()`. Параметрами данного метода являются путь к закрытому ключу подписи и путь к сертификату.

С помощью метода `sign()` выполняется подпись текста запроса:

```
sign_request = smime.sign(text)
```

Текст подписи сохраняется в файл:

```
sign_request_file = BIO.openfile(request_path + '.sign', 'w')
smime.write(sign_request_file, sign_request)
sign_request_file.close()
```

После чего выводится сообщение об успешности выполненной операции:

```
print('Signature was saved to %s.signature' % request_path)
```

Таким образом будет создана подпись запроса с восстановлением запроса.

Верификация подписи запроса

Для проверки цифровой подписи и восстановления исходного текста запроса реализуется в функции `verify(certificate_path, ca_certificate_path, sign_request_path, output)`.

Аргументы функции:

- `certificate_path` — путь к сертификату цифровой подписи пользователя;
- `ca_certificate_path` — путь к сертификату удостоверяющего центра;
- `sign_request_path` — путь к цифровой подписи;
- `output` — путь, по которому будет сохранён исходный текст запроса.

Для верификации подписи пользователя необходим публичный ключ, который может быть получен из объекта сертификата. Для этого сертификат загружается:

```
certificate = None
try:
        certificate = X509.load_cert(certificate_path)
except (X509.X509Error, ValueError):
        print('ERROR verify: Could not load certificate for
   verification')
    exit(1)
```

Для восстановления исходного текста запроса создаётся объект класса `SMIME`, после чего выполняется загрузка в него сертификатов цифровой подписи и удостоверяющего центра:

```
stack = X509.X509_Stack()
stack.push(certificate)
smime.set_x509_stack(stack)
store = X509.X509_Store()
store.load_info(ca_certificate_path)
smime.set_x509_store(store)
```

В следующих строках выполняется загрузка подписи из файла, а также восстановления исходного текста запроса:

```
pks7, data = SMIME.smime_load_pkcs7(sign_request_path)
clear_text = smime.verify(pks7, data)
```

Если верификация подписи прошла успешно, то в переменной `clear_text` будет содержаться текст запроса. На основе этого текста создаётся объект `X509.X509_Request`. С помощью метода `save()`запрос сохраняется по пути `output`:

```
if clear_text:
        request = X509.load_request_string(clear_text)
    request.save(output)
    print('Verification OK')
    print('Request file was saved to %s' % output)
else:
        print('Verification failed')
```

Таким образом, при успешной верификации подписи будет выведено сообщение `Verification OK`, в случае ошибки — `Verification failed`.

Создание сертификата

Создание сертификата пользователя реализовано в функции `make_certificate(request_path, ca_private_key_path, ca_certificate_path, output, is_printed)`. Подписать сертификат может только суперпользователь `root`. Данное ограничение сделано из соображений безопасности.

Аргументы функции:

- `request_path` — путь к запросу на подпись;
- `ca_private_key_path` — путь к закрытому ключу удостоверяющего центра;
- `ca_certificate_path` — путь к сертификату удостоверяющего центра;
- `output` — путь к файлу, в который будет сохранён сертификат;
- `is_digital` — флаг, определяющий необходимость выпуска сертификата цифровой подписи;

- `is_printed` — флаг, определяющий, необходимо ли распечатать запрос после его создания. Принимает два значения — `True` (будет произведена печать запроса) или `False` (если не будет).

Работа данной функции начинается с загрузки запроса:

```
request = None
try:
        request = X509.load_request(request_path)
except X509.X509Error:
        print('ERROR certificate: Could not load request from %s' %
   request_path)
    exit(1)
```

Создается объект сертификата, вызовом конструктора `X509.X509()`. Порядковый номер создается на основе `UNIX_TIME`:

```
certificate.set_serial_number(time().as_integer_ratio()[0])
```

Срок действия сертификата устанавливается следующим образом:

```
now = int(time() - timezone)
not_before.set_time(now)
not_after = ASN1.ASN1_UTCTIME()
not_after.set_time(now + 60 * 60 * 24 * 365)
certificate.set_not_before(not_before)
certificate.set_not_after(not_after)
```

Таким образом, сертификат действителен с момента выпуска его удостоверяющим центром в течение 1 года.

Выполняется загрузка сертификата удостоверяющего центра и закрытого ключа в переменные `ca_certificate` и `ca_private_key`. Подписчиком сертификата пользователя является удостоверяющий центр:

```
issuer = ca_certificate.get_issuer()
certificate.set_issuer(issuer)
```

Публичный ключ берётся из объекта `request` и устанавливается в сертификат:

```
public_key = request.get_pubkey()
...
certificate.set_pubkey(public_key)
```

Разработанная функция позволяет получить объект дополнения непосредственно из запроса на подпись сертификата:

```
selinux_extension = request.get_extension_by_name('selinuxContext')
if not selinux_extension:
        print('ERROR certificate: No extension selinuxContext in
   request %s' % request_path)
    exit(1)
```

Если после выполнения функции в переменной `selinux_extension` равна `None` выводится сообщение о том, что дополнения `selinuxContext` не содержится в запросе и происходит аварийное завершение работы программы.

Проверка текущего контекста безопасности реализуется на основе сертификата цифровой подписи, хранящейся на удостоверяющем центре:

```
if not is_digital:
        if not verify_user_context(subject.CN, selinux_extension.
   get_value()):
        print('ERROR certificate: Invalid SELinux context in request
   file %s' % request_path)
        exit(1)
```

Если данные проверки прошли успешно, дополнение `selinuxContext` добавляется в сертификат методом `add_ext()`.

Дополнительно в сертификат добавляется дополнение `basicConstraints`, значение которого запрещает использовать данный сертификат в качестве сертификата удостоверяющего центра:

```
certificate.add_ext(X509.new_extension('basicConstraints', 'CA:FALSE',
1))
```

Сертификат подписывается закрытым ключом удостоверяющего центра, затем сохраняется по пути, указанном в переменной `output`. Аналогично функциям создания закрытого ключа и запроса на подпись сертификата, допускается сохранение сертификата по «альтернативному» пути. Если в переменной `is_printed` содержалось значение `True`, сертификат распечатывается сразу после создания.

Описание дополнительных функций

Для удобства работы с утилитой `pgcert` реализованы другие функции, выполняющие различные действия с сертификатами и запросами на подпись:

- `print_certificate(certificate_file_path)` — распечатать сертификат в текстовом виде;
- `print_request(request_file_path)` — распечатать запрос на создание сертификата в текстовом виде;
- `get_subject(certificate_file_path)` — получить имя субъекта сертификата;
- `get_issuer(certificate_file_path)` — получить имя подписывающего сертификата;
- `get_extension(certificate_file_path, name)` — получить имя дополнения из сертификата, путь к которому содержится в переменной `certificate_file_path`, а имя дополнения — `name`;

Примеры команд, реализуемых утилитой pgcert

В данном разделе собраны некоторые варианты команд, с помощью которых пользователь может выполнить различные действия с сертификатами.

1. Создать закрытый ключ длины `length` бита:

```
$ pgcert --genkey --bits <length>
```

2. Создать закрытый ключ `pkey`:

```
$ pgcert --genkey --output <pkey>
```

3. Создать запрос на подпись сертификата пользователь `user` с контекстом безопастности `context` по закрытому ключу `pkey`:

```
$ pgcert --genreq --pkey <pkey> --user <user> --secontext <context>
```

4. Подписать запрос и распечатать созданный сертификат:

```
# pgcert --gencert --request <request> --text
```

5. Создать сертификат, закрытый ключ которого используется для цифровой подписи:

```
# pgcert --gencert --request <request> --createdsa
```

6. Проверить цифровую подпись запроса:

```
# pgcert --verify --request <request.sign> --output <request>
```

7. Просмотреть сертификат:

```
$ pgcert --text --certificate user.crt
```

8. Получить имя субъекта:

```
$ pgcert --subject --certificate mycert.pem
```

9. Получить имя подписывающего сертификат:

```
$ pgcert --issuer --certificate mycert.pem
```

10. Получить значение дополнения `basicConstraints`:

```
$ pgcert --certificate mycert.pem --extension basicConstraints
```

Разработанная таким образом утилита позволяет выполнить стандартные действия с сертификатами: создание закрытого ключа, создание запроса на подпись сертификата, а также подпись сертификат удостоверяющим центром. Функционал позволяет создать электронно-цифровую подпись, с помощью которой можно подписать запрос или проверить подпись. Благодаря функциональному стилю программирования дополнение нового функционала несложно: достаточно реализовать новую функцию и добавить опцию в парсер.

3.2 Структура тестового стенда

Для выполнения настройки механизма автоматического выбора сертификата пользователя на основании его контекста безопасности, необходимо развернуть стенд из двух машин: удостоверяющий центр (`IP: 192.168.100.2`) и клиентскую машину (`IP: 192.168.100.3`). На клиентской машине созданы пользователи операционной системы `user1`, `user2`, `user3` с разными контекстами безопасности. Схема стенда приведена *на рисунке* 3.2.

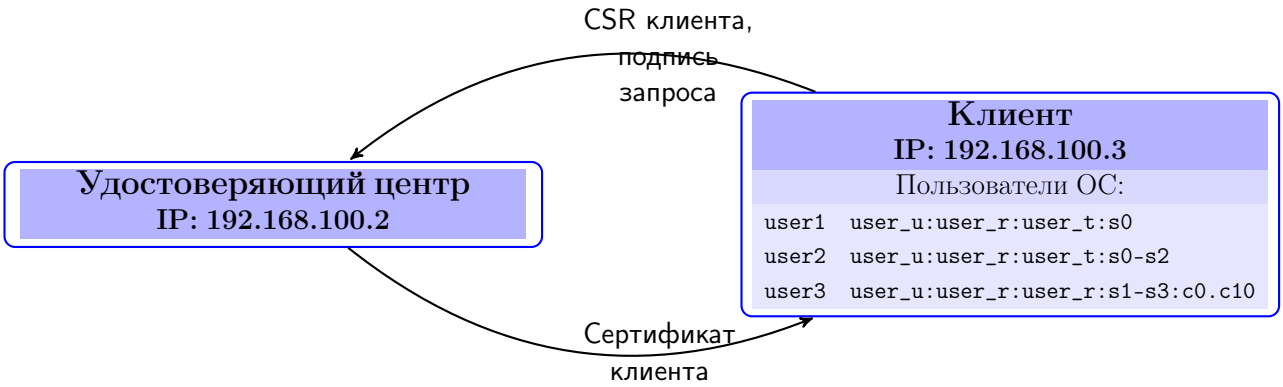

Рис. 3.2: Схема разворачиваемого стенда

В качестве операционных систем на всех машинах используется `Fedora 20`.

3.2.1 Общая настройка машин стенда

Действия, указанные в данном разделе, требуется произвести на каждой машине стенда.

1. Перевести SELinux в разрешающий режим работы. Для этого в конфигурационном файле `/etc/selinux/config` заменяется строка

   ```
   SELINUX=enforcing
   ```

   на строку

   ```
   SELINUX=permissive
   ```

2. Установить многоуровневую политику:

   ```
   # yum install selinux-policy-mls policycoreutils-python -y
   ```

Первый пакет содержит набор бинарных модулей многоуровневой политики SELinux, второй — содержит утилиты, упрощающие написание политик SELinux и их редактирование.

3. В конфигурационном файле SELinux изменяется тип политики, используемой по умолчанию:

```
SELINUXTYPE=mls
```

4. После чего расставляются метки безопасности в файловой системе:

```
# touch /.autorelabel
# reboot
```

5. Установить модифицированные библиотеки `openssl` и `m2crypto`, наложив патч и выполнив установку;

6. Скопировать утилиту `pgcert` в каталог исполняемых файлов. Для этого:

```
# cp pgcert.py /usr/bin/pgcert
```

3.2.2 Настройка удостоверяющего центра

Настройка удостоверяющего центра состоит в разворачивании удостоверяющего центра. Это можно сделать с помощью скрипта `CA`:

```
# /etc/pki/tls/misc/CA -newca
```

При выполнении данного скрипта в диалоговом режиме будет предложено ввести информацию об удостоверяющем центре:

```
Enter PEM pass phrase: <пароль>
Verifying - Enter PEM pass phrase: <пароль>
...
Country Name (2 letter code) [XX]:ru
State or Province Name (full name) []:msk
Locality Name (eg, city) [Default City]:msk
Organization Name (eg, complany) [Default Company Ltd]:mephi
```

```
Organizational Unit Name (eg, section) []:kaf36
Common Name (eg, your name or your server's hostname) []:CA
```

После выполнения данного скрипта будет развёрнут удостоверяющий центр, закрытый ключ которого будет расположен в директории `/etc/pki/CA/private` и сертификат удостоверяющего центра `cacert.pem`, расположенного в `/etc/pki/CA`.

Создается директория `certs`, в которую будут присылаться запросы на подпись сертификатов и в которой будут содержаться сертификаты:

```
# mkdir /root/certs
```

3.2.3 Настройка клиента

Настройка клиентской машины состоит в создании пользователей операционной системы, назначение им контекста безопасности, а также в настройке скрипта инициализации многоэкземплярных папок `namespace.init` и дополнительной установке пакетов.

1. Устанавливается модифицированный модуль `pam` (его модификация проводилось *в разделе 3.1.1*), с помощью которого становится возможным получить контекст пользователя в скрипте инициализации:

2. Устанавливается пакет `sshpass`, необходимый при автоматическом вводе пароля при подключении по протоколу SSH:

   ```
   # yum install -y sshpass
   ```

3. Настроить механизм многоэкземплярности папок.

   3.1. Для этого в конфигурационном файле `/etc/security/namespace.conf` записать следующие строки:

   ```
   /tmp             /tmp/tmp-inst/      level root
   /var/tmp         /var/tpm/tmp-inst/  level root
   $HOME            $HOME/$USER.int/    level root
   /etc/pki/certs/ /etc/pki/certs/certs.inst/ user root
   ```

Таким образом, будут созданные многоэкземплярные директории `/tmp`, `/var/tmp` и пользовательские домашние директории для всех пользователей, за исключением суперпользователя `root`. Последняя строка отвечает за создание директории с электронно-цифровыми подписями клиентов, используемых для генерации сертификатов.

3.2. В конфигурационный файл PAM `/etc/pam.d/sshd` дописать следующую строку:

```
session required        pam_namespace.so
```

Данная строка позволяет работать с многоэкземплярными директориями демону `SSHD`.

4. Создается пользователь SELinux `user_u`, сущность которого будет сопоставлена с пользователями клиентской машины:

```
# semanage user -m -r s0-s3:c0.c1023 user_u
```

5. Модифицируется скрипт инициализации многоэкземплярных директорий `/etc/security/namespace.init`:

Скрипт `namespace.init` исполняется интерпретатором `BASH`. Полный текст скрипта приведён в *приложении 5*. Скрипт принимает пять параметра:

- `$1` — путь к многоэкземплярной директории;
- `$2` — путь к экземпляру директории;
- `$3` — флаг, равен 0, если не требуется создание экземпляра директории и 1, если требуется;
- `$4` — имя пользователя;
- `$5` — текущий контекст безопасности пользователя.

Для улучшения читаемости скрипта были объявлены следующие глобальные переменные:

```
    polydir_path="$1"
    instance_path="$2"
    need_to_create="$3"
    user="$4"
    secontext="$5"

    log="$polydir_path/log"                  # лог-файл, в который
выводится вывод утилиты pgcert
    certdir="$polydir_path/certs"            # каталог, в котором хранится
закрытый ключ и сертификат клиента
    signerdir="/etc/pki/keys"                # каталог, в котором хранится
электронно-цифровая подпись клиента

    ca_ip="192.168.100.2"                    # IP-адрес удостоверяющего
центра
    ca_password="123456"                     # пароль пользователя root
удостоверяющего центра
    ca_certdir="/root/certs"                 # директория, в которую будут
копироваться запросы на подпись сертификата и в которой будут создаваться
    ca_signaturedir="/etc/pki/certs"         # директория, в которой
хранятся открытые ключи электронно-цифровой подписи клиентов
```

Если экземпляр многоэкземплярной директории требует создания, выполняются команды по инициализации директории. Контекст безопасности устанавливается по умолчанию:

```
[ -x /sbin/restorecon ] && /sbin/restorecon "$polydir_path"
```

Далее определяется, является ли экземпляр многоэкземплярной папки домашней директорией:

```
passwd=$(getent passwd "$user")
homedir=$(echo "$passwd" | cut -f6 -d":")
if [ "$polydir_path" = "$homedir" ]; then
        ...
fi
```

Если является, то выполняются дополнительные действия по инициализации: копирование шаблона-каталога из `/etc/skel`, назначение прав и т.д.

После выполнения этих действий выполняется создание сертификата пользователя. Сначала осуществляется проверка наличия электронно-цифровой подписи у клиента. Если её нет, она создаётся с помощью функции `create_signature`. В ней описываются действия по созданию подписи и копированию в каталог `$ca_signaturedir` удостоверяющего центра.

Действия по созданию сертификата клиента объединены в функции `create_certificate()`.

В этой функции объявлены локальные переменные `$private_key`, `$request`, `$certificate`:

```
private_key="private.key" # файл, в который будет храниться закрытый
ключ
request="$user.csr"       # файл, в котором будет храниться запрос на
подпись сертификата
certificate="$user.crt"   # файл, в котором будет храниться сертификат
```

Далее создаётся директория для хранения сертификатов `$certdir`:

```
mkdir $certdir
```

С помощью утилиты `pgcert` выполняется создание закрытого ключа, после чего утилитой `chmod` выполняется назначение ему прав `0600` (права на чтение, запись, исполнение только для владельца ключа):

```
pgcert --genkey --bits 2048 --output $certdir/$private_key >> $log
chmod 0600 $certdir/$private_key
```

Для создания запроса на подпись сертификата используется следующая команда:

```
pgcert --genreq --pkey $certdir/$private_key --user $user --secontext
$secontext --output $certdir/$request >> $log
```

Так будет создан запрос на подпись сертификата `user.csr`, где `user` — имя клиента.

Запрос подписывается цифровой подписью клиента:

```
pgcert --sign --pkey $signerdir/private.key --certificate $signerdir/
$user.crt --request $certdir/$request >> $log
```

После выполнения данной команды создаётся файл $certdir/$request.sign, в котором содержится цифровая подпись с восстановлением.

Подпись пересылаются на удостоверяющий центр с помощью команды:

```
sshpass -p $ca_password scp -o StrictHostKeyChecking=no $certdir/
$request.sign root@$ca_ip:$ca_certdir/$request.sign >> $log
```

Данная команда создает SSH соединение без проверки ключей (StrictHostKeyChecking=no), при просьбе ввести пароль от удалённой машины, утилита sshpass вводит пароль, хранящийся в переменной $ca_password. После этого файл запроса копируется в директорию $ca_certdir удостоверяющего центра.

На удостоверяющем центре выполняется проверка цифровой подписи клиента. Результатом работы является дешифрованный запрос на подпись сертификата:

```
sshpass -p $ca_password ssh -o StrictHostKeyChecking=no -T root@$ca_ip
"pgcert --verify --certificate /etc/pki/certs/$user.crt --request
$ca_certdir/$request.sign --output $ca_certdir/$request" >> $log
```

С помощью расшифрованного запроса на подпись сертификата удостоверяющей центр выпускает сертификат клиента и сохраняет его в директорию $ca_certdir:

```
shpass -p $ca_password ssh -o StrictHostKeyChecking=no -T root@$ca_ip "
pgcert --gencert --request $ca_certdir/$request --output $ca_certdir/
$certificate" >> $log
```

Если код возвращаемого значения данной командой равен 0, то сертификат пересылается клиенту в директорию $certdir:

```
sshpass -p $ca_password scp -o StrictHostKeyChecking=no root@$ca_ip:
$ca_certdir/$certificate $certdir/$certificate >> $log
```

В директорию $certdir с удостоверяющего центра копируется его сертификат:

```
sshpass -p $ca_password scp -o StrictHostKeyChecking=no root@$ca:/etc/
pki/CA/cacert.pem $certdir/root.crt
```

Всем файлам в директории устанавливается владелец $user.$user:

```
chown -R $user.$user $certdir
```

Таким образом, при аутентификации клиента в операционной системе будут созданы:

- закрытый ключ `/etc/pki/keys/private.key` и сертификат цифровой подписи `/etc/pki/keys/$user.crt`, создаваемые при первом входе в систему;
- закрытый ключ `/home/$user/certs/private.key` и сертификат `/home/$user/certs/$user.crt`, которые могут быть использованы в различных программных средствах.

3.3 Тестирование механизма

Для того, чтобы убедиться в корректности работы разработанного механизма автоматического выбора сертификата, осуществляется проверка на соответствие контекста безопасности пользователя, полученное средствами операционной системы, контексту безопасности пользователя, указанного в дополнении сертификата клиента. Данный сертификат должен храниться в домашнем каталоге клиента.

3.3.1 Подготовка к тестированию

На клиентской машине создаются пользователи `user1`, `user2`, `user3`. Их контексты безопасности представлены в *таблице 1.*

Пользователь	контекст безопасности
`user1`	`user_u:user_r:user_t:s0`
`user2`	`user_u:user_r:user_t:s0-s2`
`user3`	`user_u:user_r:user_t:s1-s3:c0.c10`

Таблица 1: Контексты пользователей клиентской машины

Далее будет приведена последовательность команд, необходимых для создания пользователей на и для назначения требуемого контекста безопасности.

1. Создается пользователь системы `user1`:

```
# useradd user1
```

2. Назначается произвольный пароль с помощью утилиты `passwd`:

```
# passwd user1
```

3. Пользователю сопоставляется пользователь SELinux `user_u` и назначается контекст согласно *таблице 1*:

```
# semanage login -a -s user_u -r s0 user1
```

4. Меняется контекст домашней директории пользователя в соответствии с его уровнем:

```
# chcon -R -l 's0' /home/user1
```

Для пользователей `user2` и `user3` выполняются аналогичные действия.

3.3.2 Алгоритм тестирования

Для проверки корректности работы автоматического механизма выбора сертификатов пользователей клиентом PostgreSQL выполняются следующие действия:

1. Зайти за пользователя `user` в систему на клиентской машине.

2. Выполнить команду, возвращающей текущий контекст пользователя:

```
$ id -Z
```

3. Удостовериться, что была создана директория `certs` и в ней находится 3 файла: `private.key`, `user.crt` и `root.crt`:

```
$ ls ~/certs
```

4. Удостовериться, что была создана ЭЦП пользователя. Для этого:

   4.1. Убедиться, что были созданы закрытый ключ ЭЦП `private.key` и сертификат `user.crt`:

```
$ ls ~/etc/pki/keys/
```

4.2. Убедиться в наличии дополнения `keyUsage` в сертификате ЭЦП `user.crt` и его значением является `Digiral Signature`:

```
$ pgcert --text --certificate /etc/pki/keys/user.crt
```

5. Удостовериться, что существует дополнение `selinuxContext` в сертификате пользователя `~/.postgresql/postgresql.crt` и его значение соответствует текущему контексту безопасности пользователя:

   ```
   $ pgcert --text --certificate ~/certs/user.crt
   ```

   Проверка по данному пункту может быть реализована с помощью утилиты `openssl`:

   ```
   $ openssl x509 -in ~/certs/user.crt -text
   ```

6. Сменить контекст безопасности с помощью команды:

   ```
   $ newrole -l <уровень>
   ```

   При выполнении данной команды необходимо ввести пароль пользователя `user`.

7. Т.к. при выполнении шага 5 происходит «подмена» экземпляра домашней директории, необходимо вновь войти в неё:

   ```
   $ cd user/
   ```

8. Повторить шаги (2-6) для всех возможных комбинаций уровней безопасности и категорий клиента.

3.3.3 Пример проверки работы механизма для пользователя user2

При проведении проверки считается, что пользователь `user2` входит в систему первый раз.

На рисунке 3.3 пользователь `user2` входит в операционную систему на клиентской машине. Его «базовый» контекст безопасности, т.е. тот, который назначается модулем `pam_namespace` по умолчанию, является `user_u:user_r:user_t:s0-s2`.

```
Fedora release 20 (Heisenbug)
Kernel 3.13.6-200.fc20.x86_64 on an x86_64 (tty3)

client login: user2
Password:
Last login: Fri May 23 16:11:04 on tty4
[user2@client ~]$ id -Z
user_u:user_r:user_t:s0-s2
[user2@client ~]$ ls ~/certs/
private.key  root.crt  user2.crt
[user2@client ~]$ _
```

Рис. 3.3: Контекст безопасности пользователя `user2` на мандатном уровне `s0-s2`

Пользователю генерируется ЭЦП с помощью создания пары закрытый ключ – сертификат пользователя. В дополнении сертификата `selinuxContext` содержится значение «базового» контекста безопасности (*рисунок 3.4*). Сертификат копируется на удостоверяющий центр и используется для верификации ЭЦП клиента, а также при проверке контекста безопасности.

```
Fedora release 20 (Heisenbug)
Kernel 3.13.6-200.fc20.x86_64 on an x86_64 (tty3)

client login: user2
Password:
Last login: Fri May 23 16:30:17 on tty3
[user2@client ~]$ id -Z
user_u:user_r:user_t:s0-s2
[user2@client ~]$ ls ~/certs/
private.key  root.crt  user2.crt
[user2@client ~]$ ls /etc/pki/keys/
private.key  user2.crt
[user2@client ~]$ _
```

```
        X509v3 extensions:
            X509v3 Selinux Context:
                user_u:user_r:user_t:s0-s2
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Key Usage: critical
                Digital Signature
    Signature Algorithm: sha1WithRSAEncryption
         b1:de:ca:a5:68:b5:d7:fa:5e:e8:7a:4a:7c:f1:2a:74:90:00:
         62:c3:8a:5a:d7:07:ae:24:16:f9:dd:94:9b:99:2d:18:bf:00:
         d9:09:1f:65:94:dd:3c:2e:e0:d6:da:bb:ab:2e:77:0d:76:fd:
         bd:86:6b:64:32:49:e0:7d:66:dc:1a:04:2d:0f:a1:84:29:75:
         31:28:c1:bb:5b:5f:0c:cb:80:b2:ad:54:a1:15:9b:a6:28:9b:
         28:5d:c5:41:e5:77:80:1f:4a:42:99:17:91:cd:14:cd:fa:dc:
         bb:68:5d:e1:fd:31:b4:39:ed:a3:f9:e2:28:cf:a8:9b:b5:52:
         7a:d3:9a:66:fc:b2:2d:c3:e7:c1:34:70:b0:dc:dd:d1:66:d8:
         b4:f7:ce:00:92:dc:fe:7a:f8:ed:c9:b0:36:7d:f0:66:cc:ff:
         90:4d:ce:d7:d0:5c:60:a7:15:81:aa:d7:7d:ca:b9:78:5e:c5:
         c2:8e:48:d6:28:7f:05:95:2a:de:1b:50:98:c0:50:9d:36:91:
         e1:e3:6b:99:bd:00:06:f1:ae:12:ba:40:d2:a5:6f:25:19:4f:
         6e:f5:96:10:af:66:68:1c:22:61:64:30:7e:40:9a:4f:cb:0c:
         5c:42:73:4a:24:ec:25:e7:82:cd:b1:f4:22:f4:5a:b7:7f:e2:
         0f:74:a8:b7

[user2@client ~]$ _
```

Рис. 3.4: Сертификат пользователя `user2` на мандатном уровне `s0-s2`

На клиенте создаётся запрос на подпись сертификата, в дополнении `selinuxContext` которого содержится текущий контекст безопасности. Он подписывается ЭЦП и пересылается на удостоверяющий центр. На удостоверяющем центре выполняется проверка ЭЦП и проверка

на допустимость контекста безопасности, указанной в запросе. Если проверка успешна, удостоверяющий центр выпускает сертификат. Он сохраняется в папку `~/certs` (*рисунок* 3.5).

```
                    52:5f
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Selinux Context:
                user_u:user_r:user_t:s0-s2
            X509v3 Basic Constraints: critical
                CA:FALSE
    Signature Algorithm: sha1WithRSAEncryption
         42:85:e6:16:ef:5a:bf:b4:a7:87:44:0d:50:3f:2b:79:2e:87:
         a3:8f:cc:da:0d:0d:85:62:b1:a7:53:7b:bd:15:86:f6:cd:3f:
         b7:38:0d:91:d6:8b:a4:58:45:66:49:45:3c:13:00:b9:1b:e7:
         3c:44:6b:51:cf:7e:6c:1f:fc:01:03:84:6d:42:7a:96:26:33:
         e5:44:33:cd:c2:55:63:9e:92:21:f3:31:75:7b:da:38:bd:e7:
         2f:fe:ab:c1:a8:55:ea:d1:12:2f:aa:0f:fe:63:c4:a0:c0:b7:
         ef:f7:8b:49:83:60:78:34:dc:50:48:31:00:21:e4:09:ed:38:
         96:9f:6d:5f:9f:a9:6a:83:86:5f:a1:ec:be:4d:d3:99:f0:80:
         f2:e3:47:36:47:f4:c7:b6:d9:4b:0d:be:e5:9c:7e:85:7a:ec:
         4f:ad:46:e9:21:38:08:84:71:be:3b:fd:ab:0a:42:d6:e7:d5:
         14:8b:9d:9c:b5:45:78:22:39:bb:3e:52:da:1b:c3:3b:10:5c:
         40:9c:4d:50:6d:ee:1d:f3:4c:3a:be:ee:47:53:54:d6:b9:e5:
         e3:67:ac:c4:75:87:55:39:65:23:88:68:cd:b8:9f:e6:bd:c4:
         e8:af:88:11:d1:fd:d3:f0:47:d7:71:3c:df:b0:00:5a:ac:ec:
         91:ea:1a:f2

[user2@client ~]$ _
```

Рис. 3.5: Сертификат пользователя `user2` на мандатном уровне `s0-s2`

Осуществляется переход на мандатный уровень `s1-s1` (*рисунок 3.6*).

```
          42:85:e6:16:ef:5a:bf:b4:a7:87:44:0d:50:3f:2b:79:2e:87:
          a3:8f:cc:da:0d:0d:85:62:b1:a7:53:7b:bd:15:86:f6:cd:3f:
          b7:38:0d:91:d6:8b:a4:58:45:66:49:45:3c:13:00:b9:1b:e7:
          3c:44:6b:51:cf:7e:6c:1f:fc:01:03:84:6d:42:7a:96:26:33:
          e5:44:33:cd:c2:55:63:9e:92:21:f3:31:75:7b:da:38:bd:e7:
          2f:fe:ab:c1:a8:55:ea:d1:12:2f:aa:0f:fe:63:c4:a0:c0:b7:
          ef:f7:8b:49:83:60:78:34:dc:50:48:31:00:21:e4:09:ed:38:
          96:9f:6d:5f:9f:a9:6a:83:86:5f:a1:ec:be:4d:d3:99:f0:80:
          f2:e3:47:36:47:f4:c7:b6:d9:4b:0d:be:e5:9c:7e:85:7a:ec:
          4f:ad:46:e9:21:38:08:84:71:be:3b:fd:ab:0a:42:d6:e7:d5:
          14:8b:9d:9c:b5:45:78:22:39:bb:3e:52:da:1b:c3:3b:10:5c:
          40:9c:4d:50:6d:ee:1d:f3:4c:3a:be:ee:47:53:54:d6:b9:e5:
          e3:67:ac:c4:75:87:55:39:65:23:88:68:cd:b8:9f:e6:bd:c4:
          e8:af:88:11:d1:fd:d3:f0:47:d7:71:3c:df:b0:00:5a:ac:ec:
          91:ea:1a:f2

[user2@client ~]$ newrole -l s1-s1
Пароль:
Warning: Permanently added '192.168.100.2' (ECDSA) to the list of known hosts.
[user2@client home]$ cd user2/
[user2@client ~]$ id -Z
user_u:user_r:user_t:s1
[user2@client ~]$ ls ~/certs/
private.key  root.crt  user2.crt
[user2@client ~]$ _
```

Рис. 3.6: Переход на мандатный уровень `s1`

Т.к. данный уровень является «подуровнем» базового уровня, будет выпущен сертификат с данным контекстом безопасности (*рисунок* 3.7).

```
                    f5:a9
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Selinux Context:
                user_u:user_r:user_t:s1
            X509v3 Basic Constraints: critical
                CA:FALSE
    Signature Algorithm: sha1WithRSAEncryption
         97:22:4b:24:56:2c:02:03:06:5e:59:c4:35:d5:e8:c3:d7:1e:
         4e:9a:d9:65:bc:77:0c:49:42:87:88:46:68:e4:84:ef:a8:95:
         65:b7:c6:2a:8e:ba:37:40:d6:d2:05:52:92:23:8b:05:0e:e6:
         e3:61:fa:b9:4e:68:82:81:1a:bd:f1:0a:e8:95:4a:48:69:6d:
         7e:37:59:36:df:fc:51:30:7a:8f:e9:f6:e0:77:24:3f:1b:48:
         09:2a:3a:aa:15:ed:de:a1:b2:0e:dc:f8:73:cb:83:ae:78:26:
         74:c7:d3:3a:9c:79:46:b4:5b:e3:f9:dd:87:59:06:20:20:f2:
         bb:de:d7:85:2b:3b:16:ab:5e:56:82:08:09:53:b6:ea:69:9a:
         2e:a1:16:bc:b8:ba:e7:cf:7c:95:ec:19:22:83:46:32:20:0b:
         ea:cd:75:f3:9f:3a:61:b3:e5:a1:9a:c3:a1:47:64:9d:1c:e6:
         f7:86:bd:9a:1b:aa:40:44:df:d4:7f:fb:fd:41:b6:a7:1f:5a:
         ca:f7:84:69:d8:44:33:ba:3e:9f:52:01:cd:b4:55:dc:78:1b:
         9f:9a:a9:a0:8e:79:ee:ce:4e:7e:92:a4:e0:a0:0a:1f:0c:48:
         3d:5d:b4:09:f8:4b:13:b7:42:df:23:82:91:68:d2:21:ff:54:
         2e:51:75:74

[user2@client ~]$ _
```

Рис. 3.7: Сертификат пользователя `user2` на мандатном уровне `s1`

Если же пользователь попробует осуществить переход на уровень, недопустимый с точки зрения удостоверяющего центра, сертификат не будет (*рисунок 3.8*). В этом случае в лог-файле `~/init.log` будет указана соответствующая ошибка.

```
                CA:FALSE
    Signature Algorithm: sha1WithRSAEncryption
         5c:af:0c:11:95:eb:a8:50:72:4f:70:09:b0:d4:4a:e9:f3:ef:
         e5:38:ec:11:20:b8:03:fb:9e:8f:90:32:08:ac:20:e2:47:2c:
         76:4d:4a:83:c9:3d:95:91:84:b1:15:e0:07:f9:fd:30:4b:7a:
         3d:99:02:85:fe:b1:0b:f1:ad:da:ca:2a:d1:6f:02:d4:4c:b9:
         95:c7:83:2c:c6:41:40:fa:09:f1:03:8c:e3:63:76:64:91:2b:
         a5:4a:30:0d:7d:e7:4c:67:28:e9:7e:19:4b:1f:8e:5e:44:19:
         e5:73:3f:59:25:8b:71:f7:c5:9d:4d:e1:73:c5:46:72:a4:a9:
         62:fc:95:62:fb:17:34:45:bf:5d:a0:5c:86:98:3d:6c:9f:e6:
         d0:d3:e7:c0:bd:e2:8f:10:c2:9f:28:43:5a:e9:92:b2:1b:b5:
         61:43:4f:ab:57:da:b7:09:6b:f1:3a:69:ac:27:e1:86:64:41:
         f9:ee:32:4d:e9:77:c4:c3:1b:32:cf:df:2b:91:ba:5b:a0:fe:
         49:78:8d:12:c9:96:c7:b8:6a:70:69:d4:33:36:e7:bd:e2:d9:
         3e:18:f0:65:0b:30:87:76:70:8e:dc:5d:8d:88:0a:9c:eb:87:
         9f:46:8e:a9:04:b9:01:48:8d:73:6c:8f:97:23:ad:1c:4c:b1:
         dd:80:43:0b

[user2@client ~]$ newrole -l s0-s2:c0.c10
Пароль:
Warning: Permanently added '192.168.100.2' (ECDSA) to the list of known hosts.
[user2@client home]$ cd user2/
[user2@client ~]$ ls ~/certs/
private.key  root.crt
[user2@client ~]$ _
```

Рис. 3.8: Переход на недопустимый уровень

Аналогичные проверки выполнялись и для остальных пользователей.

3.4 Применение разработанного механизма для СУБД PostgreSQL

Механизм автоматического выбора сертификата открытого ключа может быть использован для аутентификации клиента СУБД PostgreSQL. В этом случае необходимо добавить в стенд третью машину, а также адаптировать скрипт инициализации многоэкземплярных директорий `namespace.init`. Расширенная схема стенда представлена *на рисунке3.9.* На сервере СУБД используется PostgreSQL версии 9.3.4.

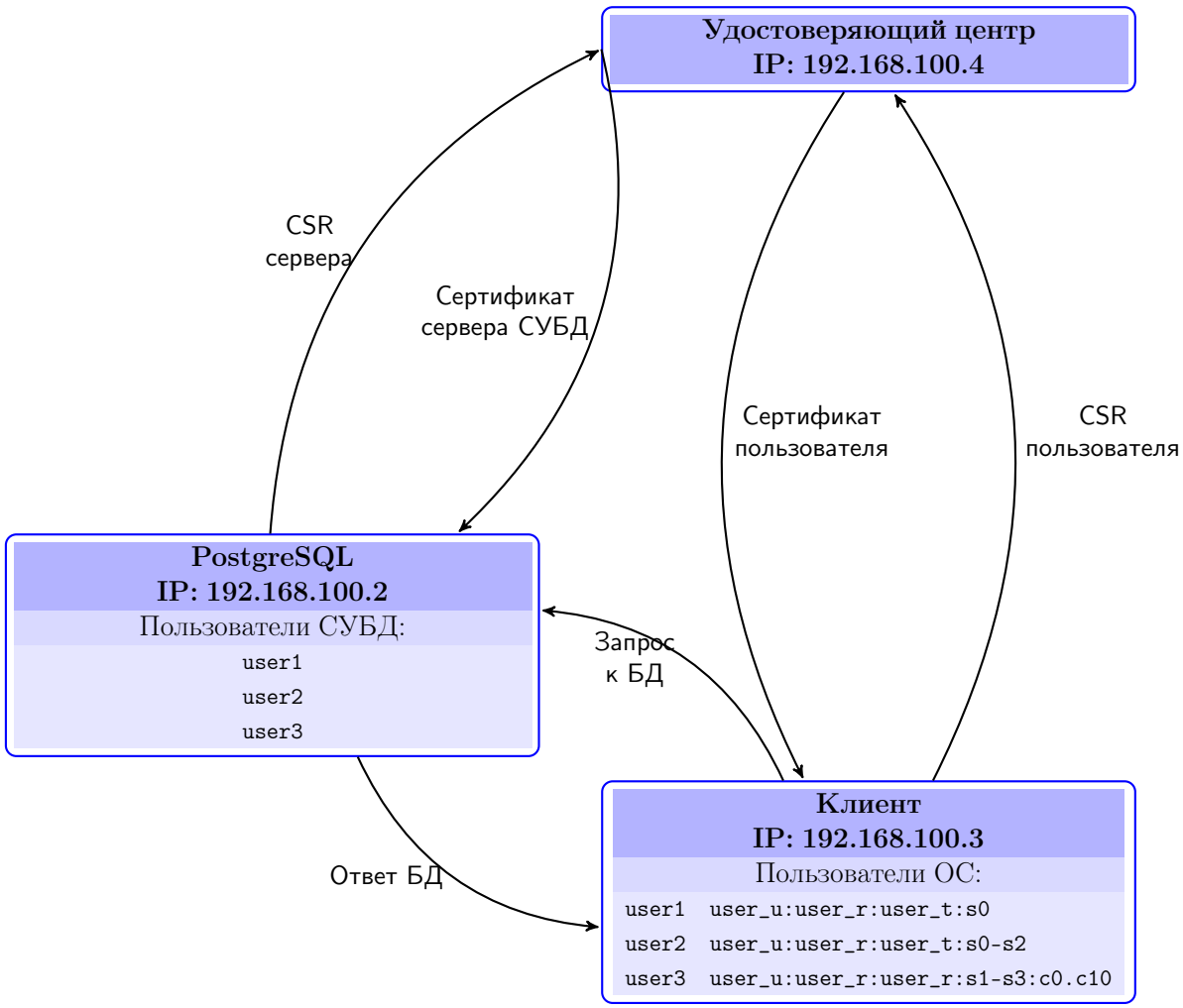

Рис. 3.9: Расширенная схема стенда

3.4.1 Настройка сервера PostgreSQL

Для установки PostgreSQL требуется подключить репозиторий PostgreSQL 9.3.4 и выполнить следующую команду:

```
# yum install -y postgresql93 postgresql93-server postgresql93-contrib
```

Инициализация сервера базы данных выполняется командой:

```
# /usr/pgsql-9.3/bin/postgresql93-setup initdb
```

Администратору базы данных, созданному при установке пакетов, назначается произвольный пароль утилитой `passwd`:

```
# passwd postgres
```

Редактируются конфигурационный файл /var/lib/pgsql/9.3/data/postgresql.conf:

```
listen_addresses = '*'
ssl = on
ssl_cert_file = 'server.crt'
ssl_key_file = 'server.key'
ssl_ca_file = 'root.crt'
shared_preload_libraries = 'sepgsql'
```

В первой строке указывается диапазон прослушиваемых адресов. В данном случае PostgreSQL будет прослушивать весь диапазон адресов.

Во второй указана активность шифрования соединения. В третьей, четвертой и пятой строках указывается имена файлов, хранящие сертификат, закрытый ключ сервера и удостоверяющего центра. В параметре `shared_preload_libraries` указывается список динамически подключаемых библиотек, которые будут подключаться при запуске сервера СУБД. В данном случае подключается модифицированная библиотека `sepgsql`.

Файл `pg_hba.conf` представляет собой таблицу со следующими полями:

```
TYPE DATABASE USER ADDRESS METHOD
```

Поле `TYPE` устанавливает тип подключения (`local` — локальное, `host` — удаленное нешифрованное по протоколу TCP/IP и т.д.).

`DATABASE` указывает имена баз данных, к которым разрешено подключаться.

`USER` определяет пользователей, которые могут подключаться.

`ADDRESS` — IP-адреса, с которых возможно подключение.

`METHOD` — Метод аутентификации, используемый в подключении.

Например, для установки метода аутентификации `cert` используется следующая строка [20]:

```
hostssl all all 192.168.100.0/24 ldap cert clientcert=1
```

Она означает, что разрешено подключение ко всем базам данных всем пользователям, IP-адреса клиентов которых находятся в подсети `192.168.100.0` при успешной аутентификации по методу cert. При этом подключение между клиентской машиной и сервером PostgreSQL будет зашифровано по протоколу SSL (на это указывает параметр `hostssl`). Дополнитель-

ный параметр метода аутентификации `clientcert=1` указывает на проверку сертификатов, высылаемых клиентами базы данных. В случае некорректных сертификатов в подключении к базе данных отказывается.

Для создания сертификата пользователя выполняются следующие действия:

1. Создается закрытый ключ и запрос на подпись сертификата. На закрытый ключ сервера назначаются права 0600:

```
$ cd /var/lib/pgsql/9.3/data
$ openssl req -new -newkey rsa:2048 -nodes -keyout mykey.pem -out
myreq.pem
$ chmod 0600 server.key
```

2. Запрос на подпись сертификата копируется[1] на удостоверяющий центр:

```
$ scp server.csr root@192.168.100.2:/root/certs/
```

3. Удостоверяющий центр подписывает сертификат:

```
# openssl ca -out server.crt -infiles server.csr
```

4. С удостоверяющего центра копируется сертификат сервера СУБД и сертификат удостоверяющего центра в директорию `PGDATA`:

```
$ scp root@192.168.100.2:/root/certs/server.crt /var/lib/pgsql/9.3/
data/
$ scp root@192.168.100.2:/etc/pki/cacert.pem /var/lib/pgsql/9.3/
data/root.crt
```

5. Изменяется владелец на сертификаты:

```
$ su -c "chown postgres.postgres *"
```

При выполнении данной команды требуется ввести пароль от пользователя `root`.

Для реализации возможности устанавливать контекст безопасности пользователем СУБД с помощью модуля `sepgsql` необходимо создать пользователя SELinux `user_u`:

[1]При выполнении шагов 3-6 требуется ввести пароль суперпользователя `root` удостоверяющего центра

```
# semanage user -a -r s0-s3:c0.c1023 user_u
```

СУБД PostgreSQL принимает по умолчанию подключения на порте `5432`. Чтобы иметь возможность подключаться к серверу СУБД по сети требуется открыть этот порт в межсетевом экране `firewall`:

```
# firewall-cmd --permanent --add-port 5432/tcp
```

Изменение настроек межсетевого экрана требует перезапуска:

```
# serverice firewalld restart
```

Изменение конфигурационных файлов PostgreSQL требует перезапуска сервера СУБД:

```
# service postgresql-9.3 restart
```

Сервис базы данных добавляется в список автозагрузки:

```
# chkconfig postgresql-9.3 on
```

3.4.2 Дополнительная настройка клиентской машины

Чтобы иметь возможность удалённо подключаться к серверу PostgreSQL, необходимо установить клиентское приложение `psql`. Для этого необходимо подключить репозиторий PosgreSQL и выполнить команду:

```
# yum install -y postgresql93
```

Ввиду того, что сервер СУБД ищет сертификаты в папке `.postgresql`, изменяется скрипт инициализации многоэкземплярных директорий `namespace.init`:

```
certdir="$polydir_path/.postgresql"
```

В функции создания сертификатов `create_certificate()` изменяются значения пути к закрытому ключу, запросу на подпись сертфиката и сертификату:

```
private_key="postgresql.key"
request="postgresql.csr"
certificate="postgresql.crt"
```

После изменения значений переменных закрытый ключ пользователя, его сертификат и сертификат удостоверяющего центра будут хранится в папке `~/.postgresql`.

3.4.3 Доработка модуля sslinfo

В СУБД PostgreSQL существует возможность подключения дополнительных функций в виде расширений (`extension`).

Модуль, предоставляющий функции для работы с сертификатами клиента, `sslinfo` [16], в настоящее время не позволяет получить информацию о дополнениях сертификата. Предлагается расширить `sslinfo` соответствующими функциями.

Данный модуль написан на языке программирования C с использованием библиотеки `libpq` [17, 18], предоставляющей интерфейс к базе данных. Модуль представляет собой динамически подключаемую библиотеку `sslinfo.so`, находящаяся в папке `/usr/pgsql-9.3/lib`, а также SQL-файл для загрузки функций в базу данных.

Реализация всех функций находится в файле `sslinfo.c`. При разработке функций модуля используется специальные типы данных, объявленные в библиотеке `libpq`.

Для того, чтобы собрать модуль `sslinfo` в динамически подключаемую библиотеку и использовать её в СУБД PostgreSQL необходимо установить пакет с заголовочными файлами базы данных:

```
# yum install postgresql93-devel -y
```

В файле модуля дополнительно подключается заголовочный файл OpenSSL `v3x509.h`, в котором объявляются функции для работы с дополнениями X509v3:

```
#include <openssl/x509v3.h>
```

Реализация функций должна начинается со строки `PG_MODULE_MAGIC;`, которая говорит базе данных, что в данном модуле реализуются хранимые процедуры.

В PostgreSQL существует возможность создания функций в «старом» и «новом» стиле. Для объявления функции в «новом» стиле перед реализацией функции используется макрос `PG_FUNCTION_INFO_V1()`, в качестве аргумента которого передается имя хранимой процедуры.

После этого реализуется функция, предназначенная для получения контекста пользователя:

```
Datum ssl_get_extension_by_name(PG_FUNCTION_ARGS);
```

При этом тип возвращаемого значения Datum — это унифицированный тип возвращаемого значения процедуры, в качестве аргумента функции передаётся PG_FUNCTION_ARGS — массив значений, передаваемые функции. В коде процедуры эти значения могут быть получены с помощью семейства функций PG_GETARG. В функции объявляются следующие переменные:

```
X509 *certificate = MyProcPort -> peer;
char *extension_name = text_to_cstring(PG_GETARG_TEXT_P(0));
X509_EXTENSION *extension = NULL;
BIO *bio = BIO_new(BIO_s_mem());
char *value = NULL;
text *result = NULL;
```

certificate — указатель на сертификат пользователя, extension_name — указатель на строку типа char*, в котором будет храниться значение переданного процедуре имени дополнения; extension — указатель на объект дополнения; bio — указатель на структуру BIO, используемую для вывода значения дополнения, value — указатель на строку, в которой будет храниться значение расширения и result — указатель на строку, выведенную после выполнения процедуры в консоль PostgreSQL.

```
if (NULL == certificate)
        PG_RETURN_NULL();
```

Если не было передано сертификата серверу СУБД (не используется SSL для шифрования соединения), то возвращается NULL.

Иначе объект дополнения получается по имени с помощью вспомогательной функции get_extension(X509 *certificate, char *extension_name). В данной функции выполняется преобразование имени дополнения в порядковый числовой идентификатор NID, с помощью которого возвращается структура X509_Extension. Если было передано некорректное имя дополнения, возвращается NULL.

Если после вызова функции get_extension() в переменной extension не содержится дополнения, то выводится соответствующее сообщение и выполнение процедуры прекращается:

```
if (NULL == extension)
```

```
    elog(ERROR, "Extension with name \"%s\" is not found in certificate
", extension_name);
```

Далее в следующем блоке кода значение дополнения считывается в переменную `value`:

```
char nullterm = '\0';
X509V3_EXT_print(bio, extension, -1, -1);
BIO_write(bio, &nullterm, 1);
BIO_get_mem_data(bio, &value);
```

Функцией `X509V3_EXT_print()` значение объекта записывается в буфер `bio`, записывается знак конца строки `nullterm`, а с помощью функции `BIO_get_mem_data()` значение расширения записывается в переменную `value`.

С помощью функции `cstring_to_text()` выполняется преобразование строки `char*` в тип строки `text*` базы данных:

```
result = cstring_to_text(value);
```

Для избежания утечек памяти очищаются переменные, объявленные в начале процедуры:

```
BIO_free(bio);
pfree(extension_name);
```

Результат выполнения функции возвращается с помощью макроса `PG_RETURN_TEXT_P()`.

Дополнительно были разработаны хранимые процедуры:

- `ssl_is_critical_extension(text)` — определяет по имени дополнения, является ли дополнение критическим. Возвращаемые значения: `t` (истина), если дополнение критическое и `f` (ложь), если некритическое;
- `ssl_get_extensions_count()` — возвращает количество дополнений в сертификате. Не принимает параметров, возвращает целое число.

В PostgreSQL существует возможность объединения разработанных процедур в модули с помощью конструкции `CREATE EXTENSION`. Загруженные таким образом процедуры не могут быть добавлены или удалены по отдельности, более того, обновление или откат к предыдущей версии процедур становится проще.

SQL-файл `sslinfo--1.0.sql`, в котором описывается создание процедур модуля, дополняется следующими строками:

```
CREATE OR REPLACE FUNCTION ssl_get_extension_by_name(text)
RETURNS text AS
'MODULE_PATHNAME', 'ssl_get_extension_by_name'
LANGUAGE C STRICT;

CREATE OR REPLACE FUNCTION ssl_is_critical_extension(text)
RETURNS text AS
'MODULE_PATHNAME', 'ssl_is_critical_extension'
LANGUAGE C STRICT;

CREATE OR REPLACE FUNCTION ssl_get_extensions_count()
RETURNS text AS
'MODULE_PATHNAME', 'ssl_get_extensions_count'
LANGUAGE C STRICT;
```

В данном листинге описывается создание функций с их сигнатурами вызова, ключевым словом `RETURNS` определяется тип возвращаемого значения, с помощью ключевого слова `AS` определяется путь к хранимой процедуре. В данном случае в переменную `MODULE_PATHNAME` подставляется путь из файла `sslinfo.control`, через запятую указывается имя функции, объявленной в библиотеке. Далее указывается, на каком языке была написана процедура.

Расширение модуля `sslinfo` реализовано с помощью патча, полный текст которого доступен в *приложении* 6.

Для применения изменений, необходимо наложить патч на модуль `sslinfo`, после чего выполнить пересборку:

```
$ cc -I /usr/pgsql-9.3/include/server -fpic -O2 -g -pipe -Wall -Wp,-
D_FORTIFY_SOURCE=2 -fexceptions -lpq -fstack-protector --param=ssp-buffer-
size=4 -m64 -mtune=generic -c -o sslinfo.o sslinfo.c
$ cc  -shared -o sslinfo.so sslinfo.o
```

Созданная таким образом динамически подключаемая библиотека, копируется в

папку /usr/pgsql-9.3/lib/, а модифицированный файл sslinfo--1.0.sql — в папку /usr/pgsql-9.3/share/extension/. Для загрузки функций из дополнения используется следующая конструкция:

```
CREATE EXTENSION sslinfo;
```

в клиенте СУБД PostgreSQL.

3.4.4 Доработка модуля sepgsql

Модуль sepgsql [19] используется в СУБД PostgreSQL для реализации мандатной системы контроля доступа с помощью системы SELinux. Для реализации возможности установки контекста из сертификата был доработан модуль sepgsql. В файле label.c была реализована функция

```
int set_label_from_certificate()
```

Эта функция предпринимает попытку установки метки безопасности клиента в глобальную переменную client_label_peer, в которой содержится контекст безопасности пользователя. Может возвращать следующие значения:

- SEPG_SSL_NOT_USED — соединение SSL не используется между клиентом и сервером;
- SEPG_SSL_EXT_ERROR — дополнение с именем selinuxContext не было найдено в сертификате клиента;
- 0 — нормальное завершение работы функции.

SEPG_SSL_NOT_USED, SEPG_SSL_NID_ERROR, SEPG_SSL_EXT_ERROR — это макросы, описанные в файле sepgsql.h:

```
#define SEPG_SSL_NOT_USED                       100
#define SEPG_SSL_EXT_ERROR                      101
```

В функции объявляются следующие переменные:

```
X509 *certificate = MyProcPort -> peer;
X509_EXTENSION *extension = NULL;
BIO *bio = NULL;
```

В переменной certificate определяется указатель на сертификат X509 клиента; extension — указатель на объект дополнения, bio — указатель на структуру BIO, используемой для распечатки дополнения.

Следующим условием проверяется активность SSL соединения:

```
if (NULL == certificate)
        return SEPG_SSL_NOT_USED;
```

Если не используется SSL соединение, возвращается код ошибки SEPG_SSL_NOT_USED. Иначе выполняется получение дополнения:

```
int locate = X509_get_ext_by_NID(certificate, NID_selinux_context,  -1)
;

extension = X509_get_ext(certificate, locate);
```

В переменной locate вычисляется положение дополнений в стеке сертификатов. После чего с помощью функции X509_get_ext() получается дополнение selinuxContext.

Выполняется проверка, было ли получено дополнение selinuxContext из сертификата:

```
if (NULL == extension) {
        elog(WARNING, "set_label_from_certificate: extension by name \"
   selinuxContext\" is not found in certificate");
        return SEPG_SSL_EXT_ERROR;
}
```

В случае отсутствия дополнения в сертификате клиента, в лог-файл PostgreSQL заносится соответствующее сообщение и возвращается код ошибки SEPG_SSL_EXT_ERROR.

В следующем блоке кода выполняется получение значения контекста безопасности из дополнения в переменную client_label_peer:

```
bio = BIO_new(BIO_s_mem());
char nullterm = '\0';
X509V3_EXT_print(bio, extension, -1, -1);
BIO_write(bio, &nullterm, 1);
BIO_get_mem_data(bio, &client_label_peer);
```

и возвращается 0 — код нормального завершения работы функции.

Данная модификация модуля `sepgsql` оформлена в виде патча, с которым ознакомиться можно в *приложении 7*.

3.5 Тестирование применения разработанного механизма для СУБД PostgreSQL

3.5.1 Подготовка к тестированию

На сервере СУБД выполняются следующие действия:

1. Создаётся тестовая база данных `testdb` от имени пользователя `postgres`:

```
$ createdb testdb
```

2. Создаются пользователи базы данных `user1`, `user2`, `user3`:

```
$ psql -c "CREATE USER user1 login;"
$ psql -c "CREATE USER user1 login;"
$ psql -c "CREATE USER user1 login;"
```

3. Модифицированные библиотеки `sepgsql.so` и `sslinfo.so` копируются в каталог `/usr/pgsql-9.3/lib`:

```
# cp sepgsql.so /usr/pgsql-9.3/lib/
# cp sslinfo.so /usr/pgsql-9.3/lib/
```

4. Модифицированный файл загрузки расширения `sslinfo` копируется в каталог `/usr/pgsql-9.3/share/extension`:

```
# cp sslinfo-1.0.sql /usr/pgsql-9.3/share/extension/
```

5. Загружается созданный модифицированный модуль `sslinfo` в базу данных `testdb`:

```
$ psql testdb -c "CREATE EXTENSION sslinfo;"
```

6. Загружается модуль `sepgsql` в базу данных `testdb`:

```
$ psql testdb -f /usr/pgsql-9.3/share/contrib/sepgsql.sql
```

7. Выполняется перезапуск сервера СУБД:

```
# service postgresql-9.3 restart
```

3.5.2 Алгоритм тестирования

Тестирование применения механизма автоматического выбора сертификата для СУБД PostgreSQL проводилось при настроенном методе аутентификации `cert`, хотя может быть проведено для других методов аутентификации с активным SSL-шифрованием.

1. Зайти в систему на клиентской машине за пользователя `user`.

2. Удостовериться, что пользователю был создан сертификат с текущим уровнем безопасности (*см. п. 3.3.2*).

3. Подключиться удалённо к базе данных `testdb` с помощью клиента `psql`:

   ```
   $ psql -h 192.168.100.4 testdb
   ```

   и убедиться в активности SSL-соединения (должны быть выведен текст «`SSL-соединение`», а также его шифр).

4. Убедиться в том, что в базе данных `testdb` содержатся дополнительные функции модуля `sslinfo`, разработка которых велась в *п. 3.4.3*. Для этого необходимо выполнить команду `\df` и убедиться в наличии данных функций.

5. Получить значение дополнения `selinuxContext` клиентского сертификата с помощью функции `ssl_get_extension_by_name()` модуля `sslinfo`:

   ```
   SELECT ssl_get_extension_by_name('selinuxContext');
   ```

   и сравнить вывод этой функции с значением контекста безопасности, полученного в п. 2 данного алгоритма.

6. Убедиться в том, что пользователю установлен контекст безопасности модулем `sepgsql` и он соответствует контексту безопасности контексту безопасности пользователя, полученного в п. 2 данного алгоритма:

```
SELECT sepgsql_getcon();
```

7. Выйти из клиента `psql` с помощью команды `\q`, сменить контекст безопасности с помощью команды `newrole` и повторить п. 2-7.

3.5.3 Пример проверки для пользователя user2

Шаг 2 настоящего алгоритма был проверен *в п. 3.3.3*. Пользователю `user2` были созданы закрытый ключ и его сертификат в каталоге `~/.postgresql`. В этом каталоге также содержится сертификат удостоверяющего центра `root.crt`. Контекст безопасности пользователя `user2`, согласно *таблице1* `user_u:user_r:user_t:s0-s2`.

На рисунке 3.10 приведён скриншот подключения пользователя `user2` к базе данных `testdb`. При этом используется шифрование соединения по протоколу SSL/

```
Fedora release 20 (Heisenbug)
Kernel 3.13.6-200.fc20.x86_64 on an x86_64 (tty2)

client login: user2
Password:
Last login: Sun May 25 19:19:24 on tty2
[user2@client ~]$ psql -h 192.168.100.4 testdb
psql (9.3.4)
SSL-соединение (шифр: DHE-RSA-AES256-SHA, бит: 256)
Введите "help", чтобы получить справку.

testdb=> _
```

Рис. 3.10: Подключение к СУБД PostgreSQL

На рисунке 3.11 приведён вывод хранимой процедуры из расширенного модуля `sslinfo` `ssl_get_extension_by_name()` и `sepgsql_getcon()`, функция, с помощью которой можно получить текущий контекст безопасности пользователя. Контекст безопасности, установлен-

ный пользователю СУБД `user2` совпадает с контекстом безопасности, указанном в сертификате пользователя `user2`.

```
     ¦ обычная
 public ¦ ssl_get_extension_by_name ¦ text                  ¦ text
     ¦ обычная
 public ¦ ssl_get_extensions_count  ¦ text                  ¦
     ¦ обычная
 public ¦ ssl_is_critical_extension ¦ text                  ¦ text
     ¦ обычная
 public ¦ ssl_is_used               ¦ boolean               ¦
     ¦ обычная
 public ¦ ssl_issuer_dn             ¦ text                  ¦
     ¦ обычная
 public ¦ ssl_issuer_field          ¦ text                  ¦ text
testdb=> SELECT ssl_get_extension_by_name('selinuxContext');
 ssl_get_extension_by_name
---------------------------
 user_u:user_r:user_t:s0-s2
(1 строка)

testdb=> SELECT sepgsql_getcon();
       sepgsql_getcon
---------------------------
 user_u:user_r:user_t:s0-s2
(1 строка)

testdb=> _
```

Рис. 3.11: Контекст безопасности пользователя `user2` на `s0-s2`

При смене уровня безопасности с помощью команды `newrole` контекст безопасности пользователя СУБД меняется согласно уровню безопасности пользователя (см. *рисунок 3.12*).

```
(1 строка)

testdb=> \q
[user2@client ~]$ newrole -l s1-s1
Пароль:
[user2@client home]$ id -Z
user_u:user_r:user_t:s1
[user2@client home]$ psql -h 192.168.100.4 testdb
psql (9.3.4)
SSL-соединение (шифр: DHE-RSA-AES256-SHA, бит: 256)
Введите "help", чтобы получить справку.

testdb=> SELECT ssl_get_extension_by_name('selinuxContext');
 ssl_get_extension_by_name
---------------------------
 user_u:user_r:user_t:s1
(1 строка)

testdb=> SELECT sepgsql_getcon();
     sepgsql_getcon
------------------------
 user_u:user_r:user_t:s1
(1 строка)

testdb=> _
```

Рис. 3.12: Контекст безопасности пользователя `user2` на `s1`

Аналогичные проверки выполнялись и для остальных пользователей.

Выводы по главе

1. Доработанный модуль `pam_namespace` позволяет передавать текущий контекст безопасности пользователя скрипту `namespace.init` в качестве параметра `$5`;

2. Реализованное дополнение `v3_secon` использовано для хранения контекста безопасности пользователя в сертификате открытого ключа;

3. Разработана утилита `pgcert` с использованием библиотеки M2Crypto. Функционал утилиты позволяет создавать закрытые ключи, запросы на подписи сертификата, сертификаты, электронно-цифровые подписи. С её помощью стало возможным реализовать механизм автоматического создания сертификата на основе контекста безопасности;

4. Показана применимость механизма для использования в СУБД PostgreSQL. Для этого vодуль `sslinfo` был дополнен функциями, предоставляющие справочную информацию

о дополнениях сертификата пользователя, а модуль `sepgsql` был дополнен возможностью установки метки безопасности пользователя из сертификата клиента.

5. Разработанный механизм был протестирован;

6. Работоспособность данного механизма показана на примере СУБД PostgreSQL.

Заключение

В данной дипломной работе предложена реализация механизма автоматического выбора сертификата открытого ключа пользователя на основании его контекста безопасности.

Для реализации использовался дистрибутив Linux Fedora 20, в качестве поставщика меток безопасности использовалась система SELinux, настроенная в многоуровневом режиме работы.

Был проведён обзор инфраструктуры открытых ключей PKI. Для реализации механизма был выбран стандарт сертификата открытого ключа X509. Проведён анализ современных подходов к управлению сертификатами открытого ключа пользователя.

Для обеспечения возможности хранения контекста безопасности в сертификате X509 программно реализовано дополнение `selinuxContext` в библиотеке OpenSSL.

Для утилиты создания сертификатов с этим дополнением была выбрана библиотека M2Crypto на языке Python. С помощью доработанного функционала библиотеки реализована утилита `pgcert`, с помощью которой обеспечивается возможность создания сертификатов с дополнением `selinuxContext`.

Для автоматизации процесса создания сертификатов использовался механизм многоэкземплярности директорий, позволяющий изолировать файлы и директории клиентов разного уровня. Модуль PAM `pam_namespace`, отвечающий за создание многоэкземплярных директорий, был доработан для обеспечения возможности передачи контекста безопасности в скрипт инициализации многоэкземплярных директорий `namespace.init`.

Работоспособность механизма показана на примере СУБД PostgreSQL, в которой серверный процесс выполняет запросы в том контексте безопасности, который соответствует метке из сертификата пользователя. При этом были доработаны модули `sepgsql` и `sslinfo`.

Все изменения программных модулей были оформлены в виде патчей.

Патч для модуля СУБД PostgreSQL `sslinfo` был отправлен мировому сообществу PostgreSQL на предмет включения в состав дистрибутива.

Работа механизма автоматического выбора сертификата открытого ключа пользователя на основании его контекста безопасности была протестирована на различных диапазонах уровней безопасности пользователей.

Таким образом была достигнута цель данной дипломной работы.

Список литературы

[1] Public-Key Infrastructure (X.509) [Электронный ресурс] — http://datatracker.ietf.org/wg/pkix/charter

[2] Frank Mayer, Karl MacMillan, David Caplan — SELinux by Example. — New Jersey: Prentice Hall, 2006. 425 с.

[3] PostgreSQL: The world's most advanced opensource database [Электронный ресурс] — http://www.postgresql.org

[4] Полянская О.Ю., Горбатов В.С. — Инфраструктуры открытых ключей — М.: Изд-во "Интернет-университет информационных технологий - ИНТУИТ.ру", "БИНОМ. Лаборатория знаний", 2007. - 368 с.: ил., ISBN: 978-5-9556-0081-9

[5] Формат сертификатов открытых ключей X.509 [Электронный ресурс] — http://www.inssl.com/x509-open-key-specifications.html

[6] Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile [Электронный ресурс] — http://www.ietf.org/rfc/rfc3280

[7] RFC 5280 [Электронный ресурс] — http://tools.ietf.org/html/rfc5280

[8] Введение в SELinux (security acl selinux limit linux kernel) [Электронный ресурс] — http://www.opennet.ru/base/sec/intro_selinux.txt.html

[9] Robb R. Romans — Improve security with polyinstantiation [Электронный ресурс] — http://www.ibm.com/developerworks/linux/library/l-polyinstantiation/index.html?S_TACT=105AGX99&S_CMP=CP

[10] Как работает PAM [Электронный ресурс] — http://www.opennet.ru/base/net/pam_linux.txt.html

[11] OpenSSL: The Open Source toolkit for SSL/TLS [Электронный ресурс] — http://ww.openssl.org

[12] Электронно-цифровая подпись [Электронный ресурс] — http://mind-control.wikia.com/wiki/Электронная_цифровая_подпись

[13] PyOpenSSL [Электронный ресурс] — http://pythonhosted.org/pyOpenSSL/

[14] M2Crypto [Электронный ресурс] — https://github.com/martinpaljak/M2Crypto

[15] SWIG Basics [Электронный ресурс] — http://www.swig.org/Doc1.3/SWIG.html

[16] PostgreSQL: sslinfo [Электронный ресурс] — http://www.postgresql.org/docs/9.3/static/sslinfo.html

[17] PostgreSQL: libpq — C library [Электронный ресурс] — http://www.postgresql.org/docs/9.3/interactive/libpq.html

[18] PostgreSQL: Documentation 9.3: C-Language Functions [Электронный ресурс] — http://www.postgresql.org/docs/9.3/static/xfunc-c.html

[19] PostgreSQL: sepgsql [Электронный ресурс] — http://www.postgresql.org/docs/9.3/static/sepgsql.html

[20] PostgreSQL: 19.3.10. Certificate Authentication [Электронный ресурс] — http://www.postgresql.org/docs/9.3/static/auth-methods.html#AUTH-CERT

Приложения

Приложение 1. Патч для библиотеки M2Crypto

Данный патч добавляет функции для работы с объектами дополнений в M2Crypto.

m2crypto-0.21.1-req-functions.patch

```
--- M2Crypto-0.21.2/SWIG/_x509.i        2011-01-15 22:10:06.000000000
+0300
+++ M2Crypto-0.21.2/SWIG/_x509.i        2014-03-30 21:05:01.000000000
+0400
@@ -507,6 +507,10 @@
     return X509_REQ_add_extensions(req, exts);
 }

+STACK_OF(X509_EXTENSION) *x509_req_get_ext(X509_REQ *request) {
+    return X509_REQ_get_extensions(request);
+}
+
 X509_NAME_ENTRY *x509_name_entry_create_by_txt(X509_NAME_ENTRY **ne,
 char *field, int type, char *bytes, int len) {
     return X509_NAME_ENTRY_create_by_txt( ne, field, type, (unsigned
char *)bytes, len);
 }
--- M2Crypto-0.21.2/M2Crypto/X509.py    2011-01-15 22:10:05.000000000
+0300
+++ M2Crypto-0.21.2/M2Crypto/X509.py    2014-03-30 23:59:07.712541164
+0400
@@ -497,6 +497,7 @@
         """
         assert m2.x509_type_check(self.x509), "'x509' type error"
         return m2.x509_add_ext(self.x509, ext.x509_ext, -1)
+

     def get_ext(self, name):
         """
@@ -967,6 +968,46 @@
         @param ext_stack: Stack of extensions to add.
         """
         return m2.x509_req_add_extensions(self.req, ext_stack._ptr())
+
+    def get_extensions(self):
+        """
+        Get all extensions of request
+
+        """
+        request_stack = m2.x509_req_get_ext(self.req)
```

```
+        extension_count = m2.sk_x509_extension_num(request_stack)
+        stack = X509_Extension_Stack()
+        for i in range(0, extension_count):
+            ext_ptr = m2.sk_x509_extension_value(request_stack, i)
+            extension = X509_Extension(ext_ptr)
+            stack.push(extension)
+        return stack
+
+    def get_extension_by_name(self, name):
+        """
+        Get an Extension by short name
+
+        @type name:      str
+        @param name:     short name of extension
+        @rtype: M2Crypto.X509.X509_Extension
+        @return: M2Crypto.X509.X509_Extension object
+
+        """
+        request_stack = m2.x509_req_get_ext(self.req)
+        extension_count = m2.sk_x509_extension_num(request_stack)
+        for i in range(0, extension_count):
+            ext_ptr = m2.sk_x509_extension_value(request_stack, i)
+            extension = X509_Extension(ext_ptr)
+            if extension.get_name() == name:
+                return extension
+        return None
+
+    def get_extension_count(self):
+        """
+        Get count of request extensions
+        """
+        request_stack = m2.x509_req_get_ext(self.req)
+        return m2.sk_x509_extension_num(request_stack)

    def verify(self, pkey):
        return m2.x509_req_verify(self.req, pkey.pkey)
```

Приложение 2. Патч для модуля pam_namespace

Данный патч позволяет получить текущий контекст пользователя при инициализации многоэкземплярной директории и передать его значение скрипту инициализации `namespace.init`.

pam-1.1.8-selinux-context.patch

```
--- Linux-PAM-1.1.8/modules/pam_namespace/pam_namespace.h
2013-06-18 18:11:21.000000000 +0400
+++ Linux-PAM-1.1.8/modules/pam_namespace/pam_namespace.h
2014-04-04 11:04:27.000000000 +0400
@@ -169,6 +169,7 @@
     uid_t owner;                        /* user which should own the
polydir */
     gid_t group;                        /* group which should own the
polydir */
     mode_t mode;                        /* mode of the polydir */
+    char *secontext;                    /* SELinux context of user */
     struct polydir_s *next;             /* pointer to the next polydir
entry */
 };

--- Linux-PAM-1.1.8/modules/pam_namespace/pam_namespace.c
2013-06-18 18:11:21.000000000 +0400
+++ Linux-PAM-1.1.8/modules/pam_namespace/pam_namespace.c
2014-04-04 13:20:42.000000000 +0400
@@ -64,6 +64,7 @@
         if (poly) {
                 free(poly->uid);
                 free(poly->init_script);
+                free(poly->secontext);
                 free(poly->mount_opts);
                 free(poly);
         }
@@ -733,7 +734,23 @@
         int rc = PAM_SUCCESS;
         security_context_t scon = NULL;
         security_class_t tclass;
-
+
+
+       FILE *fp;
+       size_t length = 0;
+       fp = popen("id -Z", "r");
+
+       if (NULL == fp) {
+               pam_syslog(idata -> pamh, LOG_ERR, "Command `id -Z`
return error");
```

```
     +                return PAM_SESSION_ERR;
     +        }
     +        int result = getline(&polyptr -> secontext, &length, fp);
     +        if (result < 0) {
     +                pam_syslog(idata -> pamh, LOG_ERR, "Can't get selinux
context");
     +                return PAM_SESSION_ERR;
     +        }
     +        polyptr -> secontext[result - 1] = '\0';
     +        pclose(fp);
     +
              /*
               * Get the security context of the directory to polyinstantiate
          .
               */
     @@ -870,7 +887,7 @@
            * polyinstantiation method.
            */

     -     pm = polyptr->method;
     +//      pm = polyptr->method;
           if (pm == LEVEL || pm == CONTEXT)
      #ifdef WITH_SELINUX
               if (!(idata->flags & PAMNS_CTXT_BASED_INST)) {
     @@ -1206,7 +1223,7 @@
                                       }
      #endif
                                       if (execle(init_script, init_script,
     -                                         polyptr->dir, ipath, newdir
?"1":"0", idata->user, NULL, envp) < 0)
     +                                         polyptr->dir, ipath, newdir
?"1":"0", idata->user, polyptr->secontext, NULL, envp) < 0)
                                               _exit(1);
                                } else if (pid > 0) {
                                       while (((rc = waitpid(pid, &status, 0))
                                           == (pid_t)-1) &&
```

Приложение 3. Дополнение v3_secon для OpenSSL

Приведенный в данном разделе программный код позволяет добавить в сертификат X509 дополнение, в котором будет храниться метка безопасности пользователя.

v3_secon.c

```
#include <stdio.h>
#include "cryptlib.h"
#include <openssl/asn1.h>
#include <openssl/conf.h>
#include <openssl/x509v3.h>

static char *i2s_ASN1_IA5STRING(X509V3_EXT_METHOD *method,
ASN1_IA5STRING *asn1_string);
static ASN1_IA5STRING *s2i_ASN1_IA5STRING(X509V3_EXT_METHOD *method,
X509V3_CTX *ctx, char *string);
const X509V3_EXT_METHOD v3_secon = EXT_IA5STRING(NID_selinux_context);

static char *i2s_ASN1_IA5STRING(X509V3_EXT_METHOD *method,
ASN1_IA5STRING *asn1_string)
{
        char *string;
        if ((NULL == asn1_string) || (NULL == asn1_string -> length))
                return NULL;
        if (NULL == (string = OPENSSL_malloc(asn1_string -> length + 1)
   )) {
                X509V3err(X509V3_F_I2S_ASN1_IA5STRING,
           ERR_R_MALLOC_FAILURE);
                return NULL;
        }
        memcpy(string, asn1_string -> data, asn1_string -> length);
        string[asn1_string -> length] = 0;
        return string;
}

static ASN1_IA5STRING *s2i_ASN1_IA5STRING(X509V3_EXT_METHOD *method,
X509V3_CTX *ctx, char *string)
{
        ASN1_IA5STRING *asn1_string;
        if (NULL == string) {
                X509V3err(X509V3_F_S2I_ASN1_IA5STRING,
           X509V3_R_INVALID_NULL_ARGUMENT);
                return NULL;
        }
        if (NULL == (asn1_string = M_ASN1_IA5STRING_new())) {
                X509V3err(X509V3_F_S2I_ASN1_IA5STRING,
           ERR_R_MALLOC_FAILURE);
```

```
                return NULL;
        }
        if (NULL == ASN1_STRING_set((ASN1_STRING *)asn1_string, (
    unsigned char*)string, strlen(string))) {
                M_ASN1_IA5STRING_free(asn1_string);
                X509V3err(X509V3_F_S2I_ASN1_IA5STRING,
           ERR_R_MALLOC_FAILURE);
                return NULL;
        }
        return asn1_string;
}
```

openssl-1.0.1e-selinux-extension.patch

```
--- openssl-1.0.1e/crypto/objects/objects.txt   2014-03-30
11:23:07.502735506 +0400
+++ openssl-1.0.1e/crypto/objects/objects.txt   2014-03-30
11:22:45.069485059 +0400
@@ -774,7 +774,8 @@
 id-ce 55               : targetInformation     : X509v3 AC Targeting
 !Cname no-rev-avail
 id-ce 56               : noRevAvail            : X509v3 No Revocation
 Available
-
+!Cname selinux-context
+id-ce 57               : selinuxContext        : Selinux Context
 # From RFC5280
 ext-key-usage 0                : anyExtendedKeyUsage   : Any Extended
 Key Usage

--- openssl-1.0.1e/crypto/x509v3/Makefile       2014-03-30
11:25:06.234055425 +0400
+++ openssl-1.0.1e/crypto/x509v3/Makefile       2014-03-30
10:53:03.000000000 +0400
@@ -22,13 +22,13 @@
 v3_int.c v3_enum.c v3_sxnet.c v3_cpols.c v3_crld.c v3_purp.c v3_info.c
  \
 v3_ocsp.c v3_akeya.c v3_pmaps.c v3_pcons.c v3_ncons.c v3_pcia.c v3_pci
 .c \
 pcy_cache.c pcy_node.c pcy_data.c pcy_map.c pcy_tree.c pcy_lib.c \
-v3_asid.c v3_addr.c
+v3_asid.c v3_addr.c v3_secon.c
 LIBOBJ= v3_bcons.o v3_bitst.o v3_conf.o v3_extku.o v3_ia5.o v3_lib.o \
 v3_prn.o v3_utl.o v3err.o v3_genn.o v3_alt.o v3_skey.o v3_akey.o
 v3_pku.o \
 v3_int.o v3_enum.o v3_sxnet.o v3_cpols.o v3_crld.o v3_purp.o v3_info.o
  \
 v3_ocsp.o v3_akeya.o v3_pmaps.o v3_pcons.o v3_ncons.o v3_pcia.o v3_pci
```

```
 .o \
       pcy_cache.o pcy_node.o pcy_data.o pcy_map.o pcy_tree.o pcy_lib.o \
      -v3_asid.o v3_addr.o
      +v3_asid.o v3_addr.o v3_secon.o

       SRC= $(LIBSRC)

      @@ -367,6 +367,21 @@
       v3_ia5.o: ../../include/openssl/stack.h ../../include/openssl/symhacks
 .h
       v3_ia5.o: ../../include/openssl/x509.h ../../include/openssl/x509_vfy.
 h
       v3_ia5.o: ../../include/openssl/x509v3.h ../cryptlib.h v3_ia5.c
      +
      +v3_secon.o: ../../e_os.h ../../include/openssl/asn1.h ../../include/
openssl/bio.h
      +v3_secon.o: ../../include/openssl/buffer.h ../../include/openssl/conf.
h
      +v3_secon.o: ../../include/openssl/crypto.h ../../include/openssl/e_os2
.h
      +v3_secon.o: ../../include/openssl/ec.h ../../include/openssl/ecdh.h
      +v3_secon.o: ../../include/openssl/ecdsa.h ../../include/openssl/err.h
      +v3_secon.o: ../../include/openssl/evp.h ../../include/openssl/lhash.h
      +v3_secon.o: ../../include/openssl/obj_mac.h ../../include/openssl/
objects.h
      +v3_secon.o: ../../include/openssl/opensslconf.h ../../include/openssl/
opensslv.h
      +v3_secon.o: ../../include/openssl/ossl_typ.h ../../include/openssl/
pkcs7.h
      +v3_secon.o: ../../include/openssl/safestack.h ../../include/openssl/
sha.h
      +v3_secon.o: ../../include/openssl/stack.h ../../include/openssl/
symhacks.h
      +v3_secon.o: ../../include/openssl/x509.h ../../include/openssl/
x509_vfy.h
      +v3_secon.o: ../../include/openssl/x509v3.h ../cryptlib.h v3_secon.c
      +
       v3_info.o: ../../e_os.h ../../include/openssl/asn1.h
       v3_info.o: ../../include/openssl/asn1t.h ../../include/openssl/bio.h
       v3_info.o: ../../include/openssl/buffer.h ../../include/openssl/conf.h
      --- openssl-1.0.1e/crypto/x509v3/ext_dat.h      2014-03-30
11:23:47.124176855 +0400
      +++ openssl-1.0.1e/crypto/x509v3/ext_dat.h      2014-03-30
11:24:18.092521126 +0400
      @@ -68,6 +68,7 @@
       extern X509V3_EXT_METHOD v3_policy_mappings, v3_policy_constraints;
       extern X509V3_EXT_METHOD v3_name_constraints, v3_inhibit_anyp, v3_idp;
       extern X509V3_EXT_METHOD v3_addr, v3_asid;
```

```
+extern X509V3_EXT_METHOD v3_secon;

 /* This table will be searched using OBJ_bsearch so it *must* kept in
  * order of the ext_nid values.
@@ -124,6 +125,7 @@
 &v3_idp,
 &v3_alt[2],
 &v3_freshest_crl,
+&v3_secon,
 };

 /* Number of standard extensions */
```

Приложение 4. Программный код утилиты pgcert

Утилита `pgcert` — это программа, написанная на языке программирования Python, позволяющая создавать сертификаты X509, закрытые ключи, запросы на подпись сертификатов с разработанным дополнением X509v3 `selinuxContext`, в котором хранится метка безопасности клиента, выполнять их подпись. В утилите реализован дополнительный набор функций, упрощающий просмотр информации о сертификатах.

pgcert

```
#!/usr/bin/python
__author__ = 'dimv36'
from M2Crypto import RSA, X509, ASN1, BIO, SMIME
from selinux import security_check_context_raw, getcon_raw
from optparse import OptionParser, OptionGroup
from os import path, getlogin
from time import time, timezone
from re import findall

DEFAULT_FIELDS = dict(C='ru', ST='msk', L='msk', O='mephi', OU='kaf36',
CN=getlogin())
CAKEY = '/etc/pki/CA/private/cakey.pem'
CACERT = '/etc/pki/CA/cacert.pem'
DIGITAL_SIGNATURE_KEY = '/etc/pki/certs/private.key'
DIGITAL_SIGNATURE_CERT = '/etc/pki/certs/%s.crt'
DEFAULT_PASSWORD = '123456'


def password(*args, **kwargs):
    return DEFAULT_PASSWORD


def check_selinux_context(context):
    if context:
        try:
            security_check_context_raw(options.secontext)
        except OSError:
            print('ERROR: Invalid SELinux context in argument')
            exit(1)


def make_level_and_category_sets(context):
    level_range = findall(r's(\d+)', context.split(':')[3])
    level_range = [int(element) for element in level_range]
    level_set = set()
    if len(level_range) == 1:
        level_set.add(level_range[0])
    else:
```

```
        level_set = {element for element in range(level_range[0],
    level_range[1] + 1)}

    category = str()
    try:
        category = context.split(':')[4]
    except IndexError:
        pass
    category_set = set()
    if category:
        category_range = findall(r'c(\d+)\.c(\d+)', category)
        for subrange in category_range:
            replace = str()
            for index in range(int(subrange[0]), int(subrange[1]) + 1):
                replace += 'c%s,' % index
            replace = replace[:-1]
            category = category.replace(str(r'c%s.c%s' % (subrange[0],
       subrange[1])), replace)
    category_set = set(findall(r'c(\d+)', category))
    category_set = {int(element) for element in category_set}
    return level_set, category_set


def verify_user_context(user, current_context):
    main_user_context = get_extension(DIGITAL_SIGNATURE_CERT % user, '
selinuxContext')
    if not main_user_context:
        return False
    main_level, main_category = make_level_and_category_sets(
main_user_context)
    current_level, current_category = make_level_and_category_sets(
current_context)
    if current_level.issubset(main_level) and current_category.issubset
(main_category):
        return True
    else:
        return False


def sign(private_key_path, certificate_path, request_path):
    if not get_extension(certificate_path, 'keyUsage') == 'Digital
Signature':
        print('ERROR sign: key pair %s and %s could not be used for
    signing file because policy' %
              (private_key_path, certificate_path))
        exit(1)
    request = None
    try:
```

```
            request = X509.load_request(request_path)
        except (IOError, X509.X509Error):
            print('ERROR sign: Could not load request from %s' %
       request_path)
            exit(1)
        text = BIO.MemoryBuffer(request.as_pem())
        smime = SMIME.SMIME()
        try:
            smime.load_key(private_key_path, certificate_path)
        except (ValueError, IOError, X509.X509Error):
            print('ERROR sign: Could not load digital signature')
            exit(1)
        sign_request = smime.sign(text)
        sign_request_file = BIO.openfile(request_path + '.sign', 'w')
        smime.write(sign_request_file, sign_request)
        sign_request_file.close()
        print('Signing request was saved to %s' % request_path + '.sign')


    def verify(certificate_path, ca_certificate_path, sign_request_path,
output):
        certificate = None
        try:
            certificate = X509.load_cert(certificate_path)
        except (X509.X509Error, IOError):
            print('ERROR verify: Could not load certificate for verifying')
            exit(1)
        smime = SMIME.SMIME()
        stack = X509.X509_Stack()
        stack.push(certificate)
        smime.set_x509_stack(stack)
        store = X509.X509_Store()
        store.load_info(ca_certificate_path)
        smime.set_x509_store(store)
        pks7, data = SMIME.smime_load_pkcs7(sign_request_path)
        clear_text = smime.verify(pks7, data)
        if not output:
            output = path.abspath(path.curdir) + '/%s.csr' % DEFAULT_FIELDS
       ['CN']
        if clear_text:
            request = X509.load_request_string(clear_text)
            request.save(output)
            print('Verification OK')
            print('Request file was saved to %s' % output)
        else:
            print('Verification failed')
```

```
    def make_private_key(bits, output):
        private_key = RSA.gen_key(bits, 65537, callback=password)
        if not output:
            output = path.abspath(path.curdir) + '/mykey.pem'
        private_key.save_key(output, None)
        print('Key was saved to %s' % output)


    def make_request(private_key_path, username, user_context, critical,
output, is_printed):
        private_key = None
        try:
            private_key = RSA.load_key(private_key_path, callback=password)
        except (IOError, RSA.RSAError):
            print('ERROR request: Could not load key pair from %s' %
       private_key_path)
            exit(1)
        request = X509.Request()
        request.set_pubkey(private_key)
        request.set_version(2)
        name = X509.X509_Name()
        name.C = DEFAULT_FIELDS['C']
        name.ST = DEFAULT_FIELDS['ST']
        name.L = DEFAULT_FIELDS['L']
        name.O = DEFAULT_FIELDS['O']
        name.OU = DEFAULT_FIELDS['OU']
        name.CN = username
        if user_context:
            context = user_context
        else:
            context = getcon_raw()[1]
        if not context:
            print('ERROR request: Could not get SELinux context for user %s
       ' % username)
            exit(1)
        request.set_subject_name(name)
        stack = X509.X509_Extension_Stack()
        stack.push(X509.new_extension('selinuxContext', context, int(
    critical)))
        request.add_extensions(stack)
        request.sign(private_key, 'sha1')
        if not output:
            output = path.abspath(path.curdir) + '/%s.csr' % DEFAULT_FIELDS
       ['CN']
        request.save_pem(output)
        if is_printed:
            print(request.as_text())
        print('Request was saved to %s' % output)
```

```
    def make_certificate(request_path, ca_private_key_file,
ca_certificate_file, output, is_digital, is_printed):
        request = None
        try:
            request = X509.load_request(request_path)
        except X509.X509Error:
            print('ERROR certificate: Could not load request from %s' %
       request_path)
            exit(1)
        public_key = request.get_pubkey()
        subject = request.get_subject()
        ca_certificate = None
        try:
            ca_certificate = X509.load_cert(ca_certificate_file)
        except (IOError, BIO.BIOError):
            print('ERROR certificate: Could not load ca certificate file.
       Check permissions and try again')
            exit(1)
        ca_private_key = None
        try:
            ca_private_key = RSA.load_key(ca_private_key_file, callback=
       password)
        except (IOError, BIO.BIOError):
            print('ERROR certificate: Could not load ca private key file.
       Check permissions and try again')
            exit(1)
        certificate = X509.X509()
        certificate.set_serial_number(time().as_integer_ratio()[0])
        certificate.set_version(2)
        certificate.set_subject(subject)
        issuer = ca_certificate.get_issuer()
        not_before = ASN1.ASN1_UTCTIME()
        now = int(time() - timezone)
        not_before.set_time(now)
        not_after = ASN1.ASN1_UTCTIME()
        not_after.set_time(now + 60 * 60 * 24 * 365)
        certificate.set_not_before(not_before)
        certificate.set_not_after(not_after)
        certificate.set_issuer(issuer)
        certificate.set_pubkey(public_key)
        selinux_extension = request.get_extension_by_name('selinuxContext')
        if not selinux_extension:
            print('ERROR certificate: No extension selinuxContext in
       request %s' % request_path)
            exit(1)
        if not is_digital:
```

```
        if not verify_user_context(subject.CN, selinux_extension.
    get_value()):
            print('ERROR certificate: Invalid SELinux context in
        request file %s' % request_path)
            exit(1)
    certificate.add_ext(selinux_extension)
    certificate.add_ext(X509.new_extension('basicConstraints', 'CA:
FALSE', 1))
    if is_digital:
        certificate.add_ext(X509.new_extension('keyUsage', 'Digital
    Signature', 1))
    if not output:
        output = path.abspath(path.curdir) + '/%s.crt' % DEFAULT_FIELDS
    ['CN']
    certificate.sign(ca_private_key, 'sha1')
    certificate.save(output)
    if is_printed:
        print(certificate.as_text())
    print('Certificate was saved to %s' % output)


def print_certificate(certificate_file_path):
    certificate = None
    try:
        certificate = X509.load_cert(certificate_file_path)
    except (X509.X509Error, ValueError):
        print('ERROR print: Could not load certificate from %s' %
    certificate_file_path)
        exit(1)
    print(certificate.as_text())


def print_request(request_file_path):
    request = None
    try:
        request = X509.load_request(request_file_path)
    except X509.X509Error:
        print('ERROR print: Could not load request from %s' %
    request_file_path)
        exit(1)
    print(request.as_text())


def get_subject(certificate_file_path):
    certificate = None
    try:
        certificate = X509.load_cert(certificate_file_path)
    except (X509.X509Error, ValueError):
```

```
        print('ERROR subject: Could not load certificate from %s' %
    certificate_file_path)
        exit(1)
    print(certificate.get_subject().as_text())


def get_issuer(certificate_file_path):
    certificate = None
    try:
        certificate = X509.load_cert(certificate_file_path)
    except (X509.X509Error, ValueError):
        print('ERROR issuer: Could not load certificate from %s' %
    certificate_file_path)
        exit(1)
    print(certificate.get_issuer().as_text())


def get_extension(certificate_file_path, name):
    certificate = None
    try:
        certificate = X509.load_cert(certificate_file_path)
    except (X509.X509Error, IOError):
        print('ERROR extension: Could not load certificate from %s' %
    certificate_file_path)
        exit(1)
    try:
        extension = certificate.get_ext(name)
    except LookupError:
        print('Certificate %s does not has extension %s' % (
    certificate_file_path, name))
    else:
        return extension.get_value()


if __name__ == '__main__':
    parser = OptionParser(usage='usage: %prog [Main Options] options',
                          add_help_option=True,
                          description='This program use M2Crypto
                      library and can generate X509 certificate '
                                      'with X509v3 extension SELinux
                                  Context')
    main_options = OptionGroup(parser, 'Main Options')
    main_options.add_option('--genkey', dest='genkey', action='
store_true', default=False,
                            help='generate private key')
    main_options.add_option('--genreq', dest='genreq', action='
store_true', default=False,
                            help='generate certificate request')
```

```
    main_options.add_option('--gencert', dest='gencert', action='
store_true', default=False,
                            help='generate certificate for user')
    main_options.add_option('--sign', dest='sign', action='store_true',
 default=False,
                            help='sign request by user\'s digital
                        signature')
    main_options.add_option('--verify', dest='verify', default=False,
action="store_true",
                            help='verify signature of request by user
                        digital signature')
    parser.add_option_group(main_options)

    pkey_group = OptionGroup(parser, 'Private key options')
    pkey_group.add_option('--bits', dest='bits', type=int, default
=2048,
                          help='set length of private key, default: %
                      default')
    parser.add_option_group(pkey_group)

    req_group = OptionGroup(parser, 'Request options')
    req_group.add_option('--user', dest='user', default=DEFAULT_FIELDS
['CN'],
                         help='set CN of request, default: %default')
    req_group.add_option('--secontext', dest='secontext', help='add
SELinux context to request')
    req_group.add_option('--critical', dest='critical', action='
store_true', default=False,
                         help='set critical of selinuxContext extension
                    , default: %default')
    parser.add_option_group(req_group)

    certificate_group = OptionGroup(parser, 'Certificate options')
    certificate_group.add_option('--createdsa', dest='createdsa',
action='store_true', default=False,
                                 help='add extension keyUsage with
                            value \'Digital signature\' to certificate,
                             '
                                      'default: %default')
    parser.add_option_group(certificate_group)

    input_options = OptionGroup(parser, 'Input options')
    input_options.add_option('--pkey', dest='pkey', help='set location
of private key')
    input_options.add_option('--request', dest='request', help='set
location of certificate request')
    input_options.add_option('--certificate', dest='certificate', help
='set location of certificate')
```

```
    input_options.add_option('--cakey', dest='cakey', default=CAKEY,
                             help='set location of ca private key,
                         default: %default')
    input_options.add_option('--cacert', dest='cacert', default=CACERT,
                             help='set location of ca certificate,
                         default: %default')
    input_options.add_option('--signature', dest='signature', help='set
 location of signature file of user\'s request')
    parser.add_option_group(input_options)

    output_options = OptionGroup(parser, 'Output options')
    output_options.add_option('--output', dest='output', help='save to
file')
    output_options.add_option('--text', dest='text', action='store_true
', default=False,
                              help='print request or certificate')
    parser.add_option_group(output_options)

    info_options = OptionGroup(parser, 'Info options')
    info_options.add_option('--issuer', dest='issuer', action='
store_true', default=False,
                            help='get issuer of certificate')
    info_options.add_option('--subject', dest='subject', action='
store_true', default=False,
                            help='get subject of certificate')
    info_options.add_option('--extension', dest='extension', help='get
extension of certificate')
    parser.add_option_group(info_options)

    options, args = parser.parse_args()
    if options.genkey and options.bits:
        make_private_key(options.bits, options.output)
    elif options.genreq and options.pkey:
        check_selinux_context(options.secontext)
        make_request(options.pkey, options.user, options.secontext,
   options.critical, options.output, options.text)
    elif options.gencert and options.request:
        make_certificate(options.request, options.cakey, options.cacert
   ,
                         options.output, options.createdsa, options.
                   text)
    elif options.sign and options.request:
        if not options.pkey:
            options.pkey = DIGITAL_SIGNATURE_KEY
        if not options.certificate:
            options.certificate = DIGITAL_SIGNATURE_CERT % ['CN']
        sign(options.pkey, options.certificate, options.request)
    elif options.verify and options.request:
```

```
    if not options.certificate:
        options.certificate = DIGITAL_SIGNATURE_CERT %
   DEFAULT_FIELDS['CN']
    verify(options.certificate, options.cacert, options.request,
options.output)
 elif options.issuer and options.certificate:
    get_issuer(options.certificate)
 elif options.subject and options.certificate:
    get_subject(options.certificate)
 elif options.text and options.certificate:
    print_certificate(options.certificate)
 elif options.certificate and options.extension:
    print(get_extension(options.certificate, options.extension))
 elif options.text and options.request:
    print_request(options.request)
 else:
    parser.print_help()
```

Приложение 5. Скрипт инициализации namespace.init

Скрипт используется для инициализации многоэкземплярных папок пользователей, а также при переходе на другой уровень безопасности с помощью команды `newrole`.

namespace.init

```
#!/bin/sh -p

polydir_path="$1"
instance_path="$2"
need_to_create="$3"
user="$4"
secontext="$5"

log="certs.log"

certdir="$polydir_path/.postgresql"
signerdir="/etc/pki/keys"

ca_ip="192.168.100.4"
ca_password="123456"
ca_certdir="/root/certs"
ca_signaturedir="/etc/pki/certs"

create_certificate() {
        private_key="private.key"
        request="$user.csr"
        certificate="$user.crt"

        mkdir $certdir
        pgcert --genkey --output $certdir/$private_key >> $log
        chmod 0600 $certdir/$private_key
        pgcert --genreq --pkey $certdir/$private_key --user $user --
   secontext $secontext --output $certdir/$request >> $log
        pgcert --sign --pkey $signerdir/private.key --certificate
   $signerdir/$user.crt --request $certdir/$request >> $log
        sshpass -p $ca_password scp -o StrictHostKeyChecking=no
   $certdir/$request.sign root@$ca_ip:$ca_certdir/$request.sign >> $log
        sshpass -p $ca_password ssh -o StrictHostKeyChecking=no -T
   root@$ca_ip "pgcert --verify --certificate /etc/pki/certs/$user.crt
   --request $ca_certdir/$request.sign --output $ca_certdir/$request"
   >> $log
        if [ $? -eq 0 ]
        then
                sshpass -p $ca_password ssh -o StrictHostKeyChecking=no
            -T root@$ca_ip "pgcert --gencert --request $ca_certdir/
           $request --output $ca_certdir/$certificate" >> $log
```

```
                if [ $? -eq 0 ]
                then
                        sshpass -p $ca_password scp -o
                   StrictHostKeyChecking=no root@$ca_ip:$ca_certdir/
                   $certificate $certdir/$certificate >> $log
                fi
                sshpass -p $ca_password scp -o StrictHostKeyChecking=no
            root@$ca_ip:/etc/pki/CA/cacert.pem $certdir/root.crt >>
           $log
        fi
        sshpass -p $ca_password ssh -o StrictHostKeyChecking=no -T
   root@$ca_ip "rm -f $ca_certdir/*"
        rm -f $certdir/$request*
        chown -R $user.$user $certdir
}


create_signature() {
        private_key=$signerdir/private.key
        request=$signerdir/$user.csr
        certificate=$signerdir/$user.crt

        pgcert --genkey --output $private_key >> $log
        pgcert --genreq --pkey $private_key --user $user --secontext
   $secontext --output $request >> $log
        sshpass -p $ca_password ssh -T root@$ca_ip bash -s <<-EOF
        if ! [ -d $ca_signaturedir ]
        then
                mkdir -p $ca_signaturedir
        fi
        EOF
        sshpass -p $ca_password scp -o StrictHostKeyChecking=no
   $request root@$ca_ip:$ca_signaturedir/$user.csr
        sshpass -p $ca_password ssh -o StrictHostKeyChecking=no -T
   root@$ca_ip "pgcert --gencert --request $ca_signaturedir/$user.csr
   --signature --output $ca_signaturedir/$user.crt" >> $log
        sshpass -p $ca_password ssh -o StrictHostKeyChecking=no -T
   root@$ca_ip "rm -f ca_signaturedir/$user.csr"
        sshpass -p $ca_password scp -o StrictHostKeyChecking=no
   root@$ca_ip:$ca_signaturedir/$user.crt $certificate
rm -f $request
}


if [ "$need_to_create" = 1 ]; then
    # This line will fix the labeling on all newly created directories
    [ -x /sbin/restorecon ] && /sbin/restorecon "$polydir_path"
    user="$4"
```

```
        passwd=$(getent passwd "$user")
        homedir=$(echo "$passwd" | cut -f6 -d":")
        if [ "$polydir_path" = "$homedir" ]; then
            gid=$(echo "$passwd" | cut -f4 -d":")
            cp -rT /etc/skel "$homedir"
            chown -R "$user":"$gid" "$homedir"
            mask=$(awk '/^UMASK/{gsub("#.*$", "", $2); print $2; exit}' /
       etc/login.defs)
            mode=$(printf "%o" $((0777 & ~$mask)))
            chmod ${mode:-700} "$homedir"
            [ -x /sbin/restorecon ] && /sbin/restorecon -R "$homedir"

                    if ! [ -f "$signerdir/private.key" ]
                    then
                            create_signature
                    fi
                    create_certificate
            fi
    fi

    exit 0
```

Приложение 6. Патч для модуля sslinfo

Данный патч расширяет модуль `sslinfo` процедурами, позволяющие получить информацию о дополнениях.

postgresql-9.3.4-sslinfo.patch

```
    --- postgresql-9.3.4/contrib/sslinfo/sslinfo.c  2014-03-17
23:35:47.000000000 +0400
    +++ postgresql-9.3.4/contrib/sslinfo/sslinfo.c  2014-04-07
10:52:04.004629335 +0400
    @@ -17,6 +17,7 @@

     #include <openssl/x509.h>
     #include <openssl/asn1.h>
    +#include <openssl/x509v3.h>


     PG_MODULE_MAGIC;
    @@ -34,6 +35,9 @@
     Datum           X509_NAME_field_to_text(X509_NAME *name, text *
 fieldName);
     Datum           X509_NAME_to_text(X509_NAME *name);
     Datum           ASN1_STRING_to_text(ASN1_STRING *str);
    +Datum           ssl_get_extension_by_name(PG_FUNCTION_ARGS);
    +Datum           ssl_is_critical_extension(PG_FUNCTION_ARGS);
```

```
+Datum          ssl_get_extensions_count(PG_FUNCTION_ARGS);


 /*
@@ -371,3 +375,78 @@
                 PG_RETURN_NULL();
         return X509_NAME_to_text(X509_get_issuer_name(MyProcPort->peer)
   );
 }
+
+
+X509_EXTENSION *get_extension(X509* certificate, char *name) {
+       int extension_nid = OBJ_sn2nid(name);
+       if (0 == extension_nid) {
+           extension_nid = OBJ_ln2nid(name);
+           if (0 == extension_nid)
+               return NULL;
+       }
+       int locate = X509_get_ext_by_NID(certificate, extension_nid,
-1);
+       return X509_get_ext(certificate, locate);
+}
+
+
+PG_FUNCTION_INFO_V1(ssl_get_extension_by_name);
+Datum
+ssl_get_extension_by_name(PG_FUNCTION_ARGS)
+{
+       X509 *certificate = MyProcPort -> peer;
+       char *extension_name = text_to_cstring(PG_GETARG_TEXT_P(0));
+       X509_EXTENSION *extension = NULL;
+       BIO *bio = BIO_new(BIO_s_mem());
+       char *value = NULL;
+       text *result = NULL;
+
+       if (NULL == certificate)
+           PG_RETURN_NULL();
+
+       extension = get_extension(certificate, extension_name);
+       if (NULL == extension)
+           elog(ERROR, "Extension by name \"%s\" is not found in
certificate", extension_name);
+
+       char nullterm = '\0';
+       X509V3_EXT_print(bio, extension, -1, -1);
+       BIO_write(bio, &nullterm, 1);
+       BIO_get_mem_data(bio, &value);
+
```

```
     +       result = cstring_to_text(value);
     +       BIO_free(bio);
     +       pfree(extension_name);
     +
     +       PG_RETURN_TEXT_P(result);
     +}
     +
     +
     +PG_FUNCTION_INFO_V1(ssl_is_critical_extension);
     +Datum
     +ssl_is_critical_extension(PG_FUNCTION_ARGS) {
     +       X509 *certificate = MyProcPort -> peer;
     +       char *extension_name = text_to_cstring(PG_GETARG_TEXT_P(0));
     +       X509_EXTENSION *extension = NULL;
     +
     +       if (NULL == certificate)
     +         PG_RETURN_NULL();
     +
     +       extension = get_extension(certificate, extension_name);
     +       if (NULL == extension)
     +           elog(ERROR, "Extension name \"%s\" is not found in
certificate", extension_name);
     +       int critical = extension -> critical;
     +
     +       PG_RETURN_BOOL(critical > 0);
     +}
     +
     +
     +PG_FUNCTION_INFO_V1(ssl_get_extensions_count);
     +Datum
     +ssl_get_extensions_count(PG_FUNCTION_ARGS) {
     +       X509 *certificate = MyProcPort -> peer;
     +
     +       if (NULL == certificate)
     +         PG_RETURN_NULL();
     +
     +       int extension_count = X509_get_ext_count(certificate);
     +       PG_RETURN_INT32(extension_count);
     +}
     --- postgresql-9.3.4/contrib/sslinfo/sslinfo--1.0.sql   2014-03-17
23:35:47.000000000 +0400
     +++ postgresql-9.3.4/contrib/sslinfo/sslinfo--1.0.sql   2014-04-07
10:54:16.242850797 +0400
     @@ -38,3 +38,18 @@
      CREATE FUNCTION ssl_issuer_dn() RETURNS text
      AS 'MODULE_PATHNAME', 'ssl_issuer_dn'
      LANGUAGE C STRICT;
     +
```

```
+CREATE OR REPLACE FUNCTION ssl_get_extension_by_name(text)
+RETURNS text AS
+'MODULE_PATHNAME', 'ssl_get_extension_by_name'
+LANGUAGE C STRICT;
+
+CREATE OR REPLACE FUNCTION ssl_is_critical_extension(text)
+RETURNS text AS
+'MODULE_PATHNAME', 'ssl_is_critical_extension'
+LANGUAGE C STRICT;
+
+CREATE OR REPLACE FUNCTION ssl_get_extensions_count()
+RETURNS text AS
+'MODULE_PATHNAME', 'ssl_get_extensions_count'
+LANGUAGE C STRICT;
```

Приложение 7. Патч для модуля sepgsql

Данный патч расширяет модуль `sepgsql`, позволяя получать метку безопасности из дополнения сертификата `selinuxContext` клиента.

postgresql-9.3.4-sepgsql.patch

```
    --- postgresql-9.3.4/contrib/sepgsql/label.c    2014-03-17
23:35:47.000000000 +0400
    +++ postgresql-9.3.4/contrib/sepgsql/label.c    2014-04-07
12:14:39.000000000 +0400
    @@ -36,8 +36,8 @@
     #include "utils/tqual.h"

     #include "sepgsql.h"
    -
     #include <selinux/label.h>
    +#include <openssl/x509v3.h>

     /*
      * Saved hook entries (if stacked)
    @@ -236,6 +236,32 @@
      * It switches the client label according to getpeercon(), and the
  current
      * performing mode according to the GUC setting.
      */
    +
    +int set_label_from_certificate() {
    +       X509 *certificate = MyProcPort -> peer;
    +       X509_EXTENSION *extension = NULL;
    +       BIO *bio = NULL;
    +
    +       if (NULL == certificate)
    +           return SEPG_SSL_NOT_USED;
    +
    +       int locate = X509_get_ext_by_NID(certificate,
NID_selinux_context,  -1);
    +       extension = X509_get_ext(certificate, locate);
    +
    +       if (NULL == extension) {
    +           elog(WARNING, "set_label_from_certificate: extension by
name \"selinuxContext\" is not found in certificate");
    +           return SEPG_SSL_EXT_ERROR;
    +       }
    +
    +       bio = BIO_new(BIO_s_mem());
    +       char nullterm = '\0';
    +       X509V3_EXT_print(bio, extension, -1, -1);
```

```
+       BIO_write(bio, &nullterm, 1);
+       BIO_get_mem_data(bio, &client_label_peer);
+       return 0;
+}
+
+
 static void
 sepgsql_client_auth(Port *port, int status)
 {
@@ -252,11 +278,14 @@
         /*
          * Getting security label of the peer process using API of
    libselinux.
          */
-       if (getpeercon_raw(port->sock, &client_label_peer) < 0)
-               ereport(FATAL,
+       int res = set_label_from_certificate();
+       if (res > 0) {
+               if (getpeercon_raw(port->sock, &client_label_peer) < 0)
{
+                       ereport(FATAL,
                                        (errcode(ERRCODE_INTERNAL_ERROR),
                                         errmsg("SELinux: unable to get peer
                                    label: %m")));
-
+               }
+       }
         /*
          * Switch the current performing mode from INTERNAL to either
    DEFAULT or
          * PERMISSIVE.
@@ -266,7 +295,6 @@
         else
                 sepgsql_set_mode(SEPGSQL_MODE_DEFAULT);
 }
-
 /*
  * sepgsql_needs_fmgr_hook
  *
--- postgresql-9.3.4/contrib/sepgsql/sepgsql.h  2014-03-17
23:35:47.000000000 +0400
+++ postgresql-9.3.4/contrib/sepgsql/sepgsql.h  2014-04-07
10:08:00.000000000 +0400
@@ -53,6 +53,10 @@
 #define SEPG_CLASS_DB_VIEW                     17
 #define SEPG_CLASS_MAX                         18

+#define SEPG_SSL_NOT_USED                      100
```

```
+#define SEPG_SSL_EXT_ERROR                      101
+
 /*
  * Internally used code of access vectors
  */
```