**Краткий обзор Безопасности Java™**

Платформа Java™ была разработана с сильным акцентом на безопасность. В его ядре язык самого Java безопасен с точки зрения типов и обеспечивает автоматическую сборку "мусора", улучшая устойчивость кода программы. Безопасная загрузка класса и механизм контроля над соблюдением соглашения гарантируют, что только законный код Java выполняется.

Начальная версия платформы Java, создаваемой безопасная среда для того, чтобы выполнить потенциально недоверяемый код, такой как апплеты Java, загружается с общедоступной сети. Поскольку платформа вырастила и расширила свой диапазон развертывания, архитектура безопасности Java соответственно развилась, чтобы поддерживать увеличивающийся набор служб. Сегодня архитектура включает большой набор прикладных программных интерфейсов (API), инструменты, и реализации обычно используемых алгоритмов безопасности, механизмов, и протоколов. Это предоставляет разработчику всесторонняя платформа безопасности для того, чтобы он записал приложения, и также предоставляет пользователю или администратору ряд инструментов, чтобы надежно управлять приложениями.

API безопасности Java охватывают широкий диапазон областей. Криптографический и инфраструктура управления открытыми ключами (PKI) интерфейсы обеспечивают базовое основание для того, чтобы оно разработало безопасные приложения. Интерфейсы для того, чтобы выполнить аутентификацию и управление доступом позволяют приложениям принять меры против несанкционированного доступа к защищенным ресурсам.

API учитывают многократные взаимодействующие реализации алгоритмов и других служб безопасности. Службы реализуются в *провайдерах*, которые включаются в платформу Java через стандартный интерфейс, который облегчает для приложений получать службы безопасности, не имея необходимость знать что-либо об их реализациях. Это позволяет разработчикам сосредотачиваться, как интегрировать безопасность в их приложения, а не на том, как фактически реализовать сложные механизмы безопасности.

Платформа Java включает много провайдеров, которые реализуют базовый набор служб безопасности. Это также учитывает дополнительных пользовательских провайдеров, которые будут установлены. Это позволяет разработчикам расширить платформу с помощью новых механизмов безопасности.

Эта бумага дает широкий краткий обзор безопасности в платформе Java, от безопасных функций языка до API безопасности, инструментов, и встроенных служб провайдера, выделяя ключевые пакеты и классы где применимый. Отметьте, что эта бумага основана на Java™ SE версия 7.

**2 Проверки Безопасности и Байт-кода Языка Java**

Язык Java разрабатывается, чтобы быть безопасным с точки зрения типов и удобным. Это обеспечивает автоматическое управление памятью, сборку "мусора", и проверку диапазона на массивах. Это уменьшает полное бремя программирования, помещенное в разработчиков, приводя к меньшему количеству тонких ошибок программирования и к более безопасному, большему количеству устойчивого кода.

Кроме того, язык Java определяет различные модификаторы доступа, которые могут быть присвоены классам Java, методам, и полям, позволяя разработчикам ограничить доступ к их реализациям класса как соответствующий. Определенно, язык определяет четыре отличных уровня доступа: private, protected, public, и, если неуказанный, package. Большинство спецификатора открытого доступа public доступ предоставляется любому. Самый рестриктивный модификатор private доступ не предоставляется вне определенного класса, в котором определяется член парламента, не занимающий официального поста (метод, например). protected модификатор предоставляет доступ к любому подклассу, или к другим классам в пределах того же самого пакета. Доступ на уровне пакета только предоставляет доступ к классам в пределах того же самого пакета.

Компилятор преобразовывает программы Java в машинно-независимое представление байт-кода. Верификатор байт-кода вызывается, чтобы гарантировать, что только законные байт-коды выполняются в Среде выполнения Java. Это проверяет, что байт-коды соответствуют Спецификации языка Java и не нарушают правила языка Java или ограничения пространства имен. Верификатор также проверяет на нарушения управления памятью, выходы за нижнюю границу стека или переполнения, и недопустимые преобразования типа данных. Как только байт-коды были проверены, Среда выполнения Java готовит их к выполнению.

**3 Основных Архитектуры безопасности**

Платформа Java определяет ряд API, охватывающих главные области безопасности, включая криптографию, инфраструктуру управления открытыми ключами, аутентификацию, безопасную передачу, и управление доступом. Эти API позволяют разработчикам легко интегрировать безопасность в свой код программы. Они были разработаны вокруг следующих принципов:

***Независимость реализации***

Приложения не должны реализовать безопасность самостоятельно. Скорее они могут запросить службы безопасности с платформы Java. Службы безопасности реализуются в провайдерах (см. ниже), которые включаются в платформу Java через стандартный интерфейс. Приложение может положиться на многократных независимых провайдеров для функциональности безопасности.

***Функциональная совместимость реализации***

Провайдеры являются взаимодействующими через приложения. Определенно, приложение не связывается с определенным провайдером, и провайдер не связывается с определенным приложением.

***Расширяемость алгоритма***

Платформа Java включает много встроенных провайдеров, которые реализуют основной набор служб безопасности, которые широко используются сегодня. Однако, некоторые приложения могут положиться на появляющиеся стандарты, еще реализованные, или на собственных службах. Платформа Java поддерживает установку пользовательских провайдеров, которые реализуют такие службы.

**Поставщики систем обеспечения безопасности**

java.security.Provider класс инкапсулирует понятие поставщика систем обеспечения безопасности в платформе Java. Это определяет имя провайдера и перечисляет службы безопасности, которые это реализует. Многократные провайдеры могут быть сконфигурированы одновременно, и перечисляются в порядке предпочтения. Когда службу безопасности требуют, самый высокий приоритетный провайдер, который реализует ту службу, выбирается.

Приложения полагаются на соответствующее getInstance метод, чтобы получить службу безопасности из базового провайдера. Например, создание обзора сообщения представляет один тип службы, доступной от провайдеров. (Раздел 4 обсуждает обзоры сообщения и другие криптографические службы.) Приложение вызывает getInstance метод в java.security.MessageDigest класс, чтобы получить реализацию определенного алгоритма обзора сообщения, такого как MD5.

MessageDigest md = MessageDigest.getInstance("MD5");

Программа может дополнительно запросить реализацию от определенного провайдера, указывая на имя провайдера, как в следующем:

MessageDigest md =

MessageDigest.getInstance("MD5", "ProviderC");

Рисунки 1 и 2 иллюстрируют эти опции для того, чтобы запросить реализацию обзора сообщения MD5. Оба данных показывают трех провайдеров что алгоритмы обзора сообщения реализации. Провайдеры упорядочиваются предпочтением слева направо (1-3). В рисунке 1 приложение запрашивает реализацию алгоритма MD5, не определяя имя провайдера. Провайдеры ищутся в привилегированном порядке и реализации от первого провайдера, предоставляющего, что определенный алгоритм, ProviderB, возвращается. В рисунке 2 приложение запрашивает реализацию алгоритма MD5 от определенного провайдера, ProviderC. На сей раз реализация от того провайдера возвращается, даже при том, что провайдер с более высоким привилегированным порядком, ProviderB, также предоставляет реализацию MD5.

|  |  |
| --- | --- |
| схема показывая приложение, запрашивающее MD5 algorithem, не определяя имя провайдера | схема показывая приложение, запрашивающее MD5 algorithem от определенного провайдера |
| **Поиск Провайдера рисунка 1** | **Рисунок 2 Определенный провайдер требуют** |

Реализация платформы Java от Oracle включает много предварительно сконфигурированных провайдеров по умолчанию, которые реализуют основной набор служб безопасности, которые могут использоваться приложениями. Отметьте, что другие реализации поставщика платформы Java могут включать различные наборы провайдеров, которые инкапсулируют специфичные для поставщика наборы служб безопасности. Когда эта бумага упоминает встроенных провайдеров по умолчанию, она ссылается на доступных в реализации Oracle.

Разделы ниже по различным областям безопасности (криптография, аутентификация, и т.д.) каждый включает описания соответствующих служб, предоставленных провайдерами по умолчанию. Таблица в Приложении C суммирует всех провайдеров по умолчанию.

**Расположение файлов**

Определенные аспекты безопасности Java, упомянутой в этой газете, включая конфигурацию провайдеров, могут быть настроены, устанавливая свойства безопасности. Можно установить свойства безопасности статически в файле свойств безопасности, который по умолчанию является *java.security* файлом в каталоге *lib/безопасности* каталога, где Среда выполнения Java™ (JRE) устанавливается. Свойства безопасности могут также быть установлены динамически, вызывая соответствующие методы Security класс (в java.security пакет).

Инструменты и команды, упомянутые в этой газете, являются всеми в ~jre/bin каталог, где ~jre стенды для каталога, в котором устанавливается JRE. *cacerts* файл, упомянутый в Разделе 5, находится в ~jre/lib/security.

**4 Криптографии**

Архитектура криптографии Java является платформой для доступа и разработки криптографической функциональности для платформы Java. Это включает API для большого разнообразия криптографических служб, включая

* Алгоритмы обзора сообщения
* Алгоритмы цифровой подписи
* Симметричное шифрование больших объемов данных
* Симметричное поточное шифрование
* Асимметричное шифрование
* Основанное на пароле шифрование (PBE)
* Шифрование в эллиптических кривых (ECC)
* Алгоритмы согласования ключей
* Ключевые генераторы
* Коды аутентификации сообщений (MACs)
* (Псевдо-) генераторы случайных чисел

Для исторического (контроль над экспортом) причины криптографические API-функции организуются в два отличных пакета. java.security пакет содержит классы, которые *не* подвергаются контролям над экспортом (как Signature и MessageDigest). javax.crypto пакет содержит классы, которые подвергаются контролям над экспортом (как Cipher и KeyAgreement).

Криптографические интерфейсы основаны на провайдере, учитывая многократные и взаимодействующие реализации криптографии. Некоторые провайдеры могут выполнить криптографические операции в программном обеспечении; другие могут выполнить операции на аппаратном маркере (например на smartcard устройстве или на аппаратных средствах криптографический акселератор). Провайдеры, которые реализуют управляемые экспортом службы, должны быть в цифровой форме подписаны.

Платформа Java включает встроенных провайдеров для многих из обычно используемых криптографических алгоритмов, включая RSA, DSA, и алгоритмы подписи ECDSA, DES, AES, и алгоритмы шифрования ARCFOUR, MD5, SHA 1, и SHA 256 алгоритмов обзора сообщения, и Diffie-Hellman и алгоритмы согласования ключей ECDH. Эти провайдеры по умолчанию реализуют криптографические алгоритмы в коде Java.

Платформа Java также включает встроенного провайдера, который действует как мост к собственному PKCS#11 (v2.x) маркер. Этот провайдер, названный SunPKCS11, позволяет приложения Java легко криптографическим службам доступа, расположенным на PKCS#11-compliant маркеры.

На Windows платформа Java включает встроенного провайдера, который действует как мост к собственной Microsoft CryptoAPI. Этот провайдер, названный MSCAPI, позволяет приложения Java легко криптографическим службам доступа на Windows через CryptoAPI.

**5 Инфраструктур управления открытыми ключами**

Инфраструктура управления открытыми ключами (PKI) является термином, использованным для платформы, которая включает безопасному обмену информацией, основанному на шифровании с открытым ключом. Это позволяет идентификационным данным (людей, организаций, и т.д.) быть связанными с цифровыми сертификатами и обеспечивает средство проверки подлинности сертификатов. PKI охватывает ключи, сертификаты, шифрование с открытым ключом, aand доверяло Центрам сертификации (АВАРИЯ), кто генерирует и в цифровой форме подписывает сертификаты.

Платформа Java включает API и поддержку провайдера цифровых сертификатов X.509 и Списков аннулированных сертификатов (CRL), так же как PKIX-совместимое здание пути сертификации и проверка допустимости. Классы, связанные с PKI, располагаются в java.security и java.security.cert пакеты.

**Манипулируйте и Хранение Сертификата**

Платформа Java обеспечивает в течение длительного срока персистентное хранение криптографических ключей и сертификатов через хранилища сертификата и ключ. Определенно, java.security.KeyStore класс представляет *базу ключей*, безопасный репозитарий криптографических ключей и/или доверял сертификатам (чтобы использоваться, например, во время проверки допустимости пути сертификации), и java.security.cert.CertStore класс представляет *хранилище сертификата*, общедоступный и потенциально обширный репозитарий несвязанных и обычно недоверяемых сертификатов. A CertStore май также хранит CRL.

KeyStore и CertStore реализации отличают типы. Платформа Java включает стандартный *PKCS11* и типы базы ключей *PKCS12* (чьи реализации совместимы с соответствующими спецификациями PKCS из Безопасности RSA), так же как собственный основанный на файле тип базы ключей под названием *JKS* (который обозначает "Базу ключей Java").

Платформа Java включает специальную встроенную базу ключей JKS, *cacerts*, который содержит много сертификатов для известной, доверяемой АВАРИИ. keytool утилита в состоянии перечислить сертификаты, включенные в *cacerts* (см., что документация средств защиты соединяется в Разделе 10).

Провайдер SunPKCS11, упомянутый в разделе "Криптографии" (Раздел 4), включает *PKCS11* KeyStore реализация. Это означает, что к ключам и сертификатам, находящимся в безопасных аппаратных средствах (таким как smartcard), можно получить доступ и использоваться приложениями Java через KeyStore API. Отметьте, что smartcard ключам нельзя разрешить оставить устройство. В таких случаях, java.security.Key ссылка на объект, возвращенная KeyStore API может просто быть ссылкой на ключ (то есть, это не содержало бы фактический ключевой материал). Такой Key объект может только использоваться, чтобы выполнить криптографические операции на устройстве, где фактический ключ находится.

Платформа Java также включает тип хранилища сертификата *LDAP* (для того, чтобы получить доступ к сертификатам, сохраненным в каталоге LDAP), так же как тип хранилища сертификата *Набора* в памяти (для того, чтобы получить доступ к сертификатам, которыми управляют в a java.util.Collection объект).

**Инструменты PKI**

Есть два встроенных инструмента для того, чтобы работать с ключами, сертификатами, и базами ключей:

**keytool** используется, чтобы создать и управлять базами ключей. Это может

* Создайте пары "открытый/закрытый ключ"
* Дисплей, импорт, и экспорт X.509 v1, v2, и v3 сертификаты, сохраненные как файлы
* Создайте самоподписанные сертификаты
* Сертификат проблемы (PKCS#10) запрашивает быть отправленным АВАРИИ
* Создайте сертификаты, основанные на запросах сертификата
* Ответы сертификата импорта (полученный из АВАРИИ отправил запросы сертификата),
* Определяйте сертификаты с открытым ключом как доверяющийся

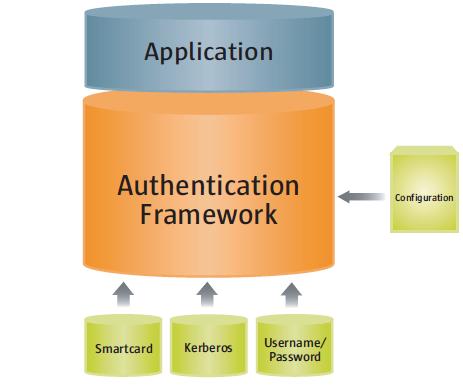
**jarsigner** инструмент используется, чтобы подписать файлы JAR, или проверить подписи на файлах JAR со знаком. Архив Java (JAR) формат файла включает связыванию многократных файлов в единственный файл. Обычно файл JAR содержит файлы класса и вспомогательные ресурсы, связанные с апплетами и приложениями. Когда Вы хотите в цифровой форме подписать код, Вы сначала используете keytool, чтобы генерировать или импортировать соответствующие ключи и сертификаты в Вашу базу ключей (если они уже не там), затем используйте инструмент **фляги**, чтобы поместить код в файл JAR, и наконец использовать jarsigner инструмент, чтобы подписать файл JAR. jarsigner инструмент получает доступ к базе ключей, чтобы найти любые ключи, и сертификаты должны были подписать файл JAR или проверять подпись файла JAR со знаком. Отметьте: **jarsigner** может дополнительно генерировать подписи, которые включают метку времени. Системы (такие как Плагин Java), которые проверяют подписи файла JAR, могут проверить метку времени и принять файл JAR, который был подписан, в то время как сертификат подписания был допустим вместо того, чтобы требовать, чтобы сертификат был текущим. (Ежегодно сертификаты обычно истекают, и не разумно ожидать, что создатели файла JAR оставят развернутые файлы JAR ежегодно.)

**6 Аутентификации**

Аутентификация является процессом определения идентификационных данных пользователя. В контексте среды выполнения Java это - процесс идентификации пользователя выполняющейся программы Java. В определенных случаях этот процесс может положиться на службы, описанные в разделе "Криптографии" (Раздел 4).

Платформа Java обеспечивает API, которые позволяют приложению выполнить пользовательскую аутентификацию через сменные модули входа в систему. Приложения вызывают в LoginContext класс (в javax.security.auth.login пакет), который поочередно ссылается на конфигурацию. Конфигурация определяет который модуль входа в систему (реализация javax.security.auth.spi.LoginModule интерфейс), должен использоваться, чтобы выполнить фактическую аутентификацию.

Так как приложения исключительно говорят со стандартом LoginContext API, они могут остаться независимыми от базовых сменных модулей. Новые или обновленные модули могут быть включены для приложения, не имея необходимость изменять приложение непосредственно. Рисунок 3 иллюстрирует независимость между приложениями и базовыми модулями входа в систему:



**Модули входа в систему Аутентификации рисунка 3, включающие платформу аутентификации**

Важно отметить, что, хотя модули входа в систему являются сменными компонентами, которые могут быть сконфигурированы в платформу Java, они не включаются на пути поставщики систем обеспечения безопасности. Поэтому, они не следуют за моделью поиска Провайдера, описанной в Разделе 3. Вместо этого как показывается в вышеупомянутой схеме, модули входа в систему администрируемы их собственной уникальной конфигурацией.

Платформа Java предоставляет следующему встроенному LoginModules, всем в com.sun.security.auth.module пакет:

* Krb5LoginModule для аутентификации, используя протоколы Kerberos
* JndiLoginModule для имени пользователя/аутентификации по паролю, используя LDAP или базы данных NIS
* KeyStoreLoginModule для того, чтобы зарегистрировать в любой тип базы ключей, включая PKCS#11 маркерную базу ключей

Аутентификация может также быть достигнута во время процесса установления безопасного канала связи между двумя коллегами. Платформа Java обеспечивает реализации многих стандартных протоколов связи, которые обсуждаются в следующем разделе.

**7 Безопасной Передачи**

К данным, которые перемещаются через сеть, может получить доступ кто-то, кто не предполагаемый получатель. Когда данные включают частную информацию, такую как пароли и номера кредитной карточки, шаги должны быть сделаны, чтобы сделать данные непонятными к несанкционированным сторонам. Также важно гарантировать, что Вы отправляете данные соответствующей стороне, и что данные не были изменены, или преднамеренно или неумышленно, во время транспорта.

Криптография формирует основание, требуемое для безопасной передачи, и это описывается в Разделе 4. Платформа Java также оказывает поддержку API и реализации провайдера для многих стандартных безопасных протоколов связи.

**SSL/TLS**

Платформа Java обеспечивает API и реализацию SSL и протоколов TLS, который включает функциональность для шифрования данных, целостности сообщения, аутентификации сервера, и дополнительной аутентификации клиента. Приложения могут использовать SSL/TLS, чтобы предусмотреть безопасный проход данных между два, смотрит на любой протокол приложения, такой как HTTP сверху TCP/IP.

javax.net.ssl.SSLSocket класс представляет сетевой сокет, который инкапсулирует поддержку SSL/TLS сверху нормального потокового сокета (java.net.Socket). Некоторые приложения могли бы хотеть использовать альтернативные транспортные абстракции данных (например, New-I/O); javax.net.ssl.SSLEngine класс доступен, чтобы произвести и использовать пакеты SSL/TLS.

Платформа Java также включает API, которые поддерживают понятие сменных (основанных на провайдере) ключевых менеджеров и доверяют менеджерам. *Ключевой менеджер* инкапсулируется javax.net.ssl.KeyManager класс, и управляет ключами, используемыми, чтобы выполнить аутентификацию. *Доверительный менеджер* инкапсулируется TrustManager класс (в том же самом пакете), и принимает решения относительно того, кто доверять основанный на сертификатах в базе ключей, которой это управляет.

Платформа Java включает встроенного провайдера, который реализует протоколы SSL/TLS:

* SSLv3
* TLSv1
* TLSv1.1
* TLSv1.2

**SASL**

Простой Уровень Аутентификации и Безопасности (SASL) является интернет-стандартом, который определяет протокол для аутентификации и дополнительного установления уровня безопасности между приложениями клиента и сервера. SASL определяет, как данные аутентификации должны быть переданы, но самостоятельно не определяют содержание тех данных. Это - платформа, в который определенные механизмы аутентификации, которые определяют содержание, и семантика данных аутентификации может соответствовать. Есть много стандартных механизмов SASL, определенных интернет-сообществом для различных уровней безопасности и сценариев развертывания.

Java API SASL определяет классы и интерфейсы для приложений то использование механизмы SASL. Это определяется, чтобы быть нейтральным механизмом; приложение, которое использует API, не должно быть предрасположено в использование никакого определенного механизма SASL. Приложения могут выбрать механизм, чтобы использовать основанный на требуемых средствах защиты. API поддерживает оба приложения клиента и сервера. javax.security.sasl.Sasl класс используется, чтобы создать SaslClient и SaslServer объекты.

Реализации механизма SASL предоставляются в пакетах провайдера. Каждый провайдер может поддерживать один или более механизмов SASL и регистрируется и вызывается через стандартную архитектуру провайдера.

Платформа Java включает встроенного провайдера, который реализует следующие механизмы SASL:

* CRAM-MD5, ОБЗОР-MD5, ВНЕШНИЙ, GSSAPI, NTLM, и ПРОСТЫЕ клиентские механизмы
* CRAM-MD5, ОБЗОР-MD5, GSSAPI, и механизмы сервера NTLM

**GSS-API и Kerberos**

Платформа Java содержит API с привязками к языку Java для Универсального Прикладного программного интерфейса Службы безопасности (GSS-API). GSS-API предлагает универсальный доступ прикладных программистов к службам безопасности на множестве базовых механизмов безопасности. GSS-API Java в настоящий момент требует использования Kerberos v5 механизм, и платформа Java включает встроенную реализацию этого механизма. В это время не возможно включить дополнительные механизмы. Отметьте: Krb5LoginModule упомянутый в Разделе 6 может использоваться в соединении с механизмом Kerberos GSS.

Платформа Java также включает встроенную реализацию Простого и Защищенного Механизма Согласования GSSAPI (SPNEGO) механизм GSS-API.

Прежде, чем два приложения могут использовать GSS-API Java, чтобы надежно обмениваться сообщениями между ними, они должны установить объединенный контекст защиты. Контекст инкапсулирует информацию об общем состоянии, которая могла бы включать, например, криптографические ключи. Оба приложения создают и используют org.ietf.jgss.GSSContext объект установить и поддержать общую информацию, которая составляет контекст защиты. Как только контекст защиты был установлен, он может использоваться, чтобы подготовить безопасные сообщения к обмену.

Java API GSS находится в org.ietf.jgss пакет. Платформа Java также определяет основные классы Kerberos, как KerberosPrincipal, KerberosTicket, KerberosKey, и KeyTab, которые располагаются в javax.security.auth.kerberos пакет.

**8 Управлений доступом**

Архитектура управления доступом в платформе Java защищает доступ к чувствительным ресурсам (например, локальные файлы) или чувствительный код программы (например, методы в классе). Все решения управления доступом устанавливаются менеджером безопасности, представленным java.lang.SecurityManager класс. A SecurityManager должен быть установлен в Среду выполнения Java, чтобы активировать проверки управления доступом.

Апплеты Java и веб-приложения Запуска Java™ автоматически выполняются с a SecurityManager установленный. Однако, местные применения, выполняемые через команду **java**, по умолчанию не выполняются с a SecurityManager установленный. Чтобы выполнить местные применения с SecurityManager, любой, которого само приложение должно программно установить один через setSecurityManager метод (в java.lang.System класс), или **java** должен быть вызван с a -Djava.security.manager параметр на командной строке.

**Полномочия**

Когда код Java загружается загрузчиком класса в Среду выполнения Java, загрузчик класса автоматически связывает следующую информацию с тем кодом:

* Где код был загружен из
* Кто подписал код (если любой)
* Полномочия по умолчанию, предоставленные коду

Эта информация связывается с кодом независимо от того, загружается ли код по ненадежной сети (например, апплет) или загружается из файловой системы (например, местное применение). Расположение, из которого был загружен код, представляется URL, подписывающее лицо кода представляется цепочкой сертификата подписывающего лица, и полномочия по умолчанию представляются java.security.Permission объекты.

Полномочия по умолчанию, автоматически предоставленные загруженному коду, включают возможность сделать сетевые соединения назад с узлом, из которого это произошло. Полномочия по умолчанию, автоматически предоставленные кодировать загруженный из локальной файловой системы, включают возможность считать файлы из каталога, это прибыло из, и также из подкаталогов того каталога.

Отметьте, что идентификационные данные пользователя, выполняющего код, не доступны во время загрузки класса. Это - обязанность кода программы аутентифицировать конечного пользователя в случае необходимости (например, как описано в Разделе 6). Как только пользователь аутентифицировался, приложение может динамически связать того пользователя с выполняющимся кодом, вызывая doAs метод в javax.security.auth.Subject класс.

**Политика**

Как отмечалось ранее, ограниченный набор полномочий по умолчанию предоставляются кодировать загрузчиками класса. У администраторов есть возможность гибко управлять полномочиями дополнительного кода через политику безопасности.

Платформа Java инкапсулирует понятие политики безопасности в java.security.Policy класс. Есть только один Policy объект, установленный в Среду выполнения Java в любой момент времени. Основная ответственность Policy объект состоит в том, чтобы определить, разрешают ли доступу к защищенному ресурсу кодировать (характеризуемый тем, где это было загружено из, кто подписал это, и кто выполняет это). Как a Policy объект делает это определение, является зависящим от реализации. Например, это может консультироваться с базой данных, содержащей данные авторизации, или это может связаться с другой службой.

Платформа Java включает значение по умолчанию Policy реализация, которая читает ее данные авторизации из одного или более ASCII (UTF-8) файлы, сконфигурированные в файле свойств безопасности. Эти файлы политики содержат точные наборы полномочий, предоставленных кодировать: определенно, точные наборы полномочий, предоставленных кодировать загруженный из определенных расположений, подписанных определенными объектами, и выполняющийся как определенные пользователи. Записи политики в каждом файле должны соответствовать задокументированному собственному синтаксису, и могут быть составлены через простой текстовый редактор или графическую **policytool** утилиту.

**Осуществление Управления доступом**

Среда выполнения Java отслеживает последовательность вызовов Java, которые выполняются, поскольку программа выполняется. Когда доступ к защищенному ресурсу запрашивается, весь стек вызовов, по умолчанию, оценивается, чтобы определить, разрешается ли запрошенный доступ.

Как отмечалось ранее ресурсы защищаются SecurityManager. Чувствительный к безопасности код в платформе Java и в приложениях защищает доступ к ресурсам через код как следующее:

SecurityManager sm = System.getSecurityManager();

if (sm != null) {

sm.checkPermission(perm);

}

где перманентом является Объект полномочий, который соответствует запрошенному доступу. Например, если попытка предпринимается, чтобы считать файл*/tmp/abc*, разрешение может быть создано следующим образом:

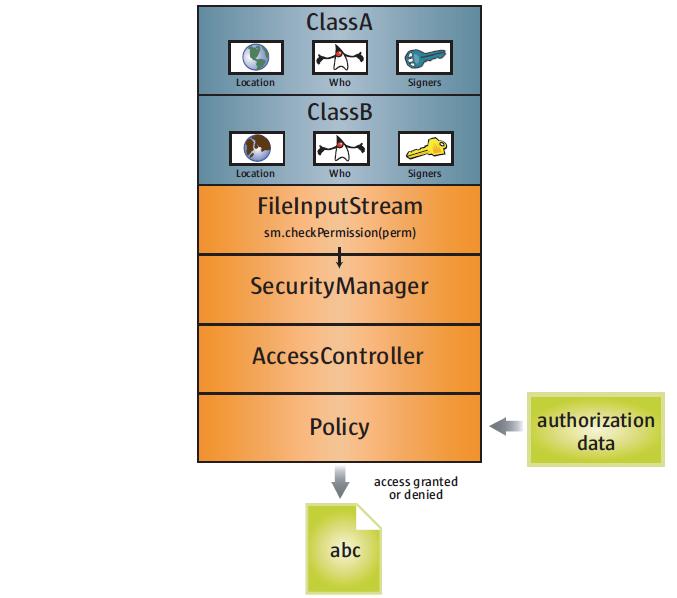
Permission perm =

new java.io.FilePermission("/tmp/abc", "read");

Реализация по умолчанию SecurityManager делегирует его решение к java.security.AccessController реализация. AccessController пересекает стек вызовов, передавая к установленной безопасности Policy каждый элемент кода в стеке, наряду с требуемым разрешением (например, FilePermission в вышеупомянутом примере). Policy определяет, предоставляют ли запрошенный доступ, основан на полномочиях, сконфигурированных администратором. Если доступ не предоставляется, AccessController броски a java.lang.SecurityException.

Рисунок 4 иллюстрирует осуществление управления доступом. В этом определенном примере есть первоначально два элемента на стеке вызовов, ClassA и ClassB. ClassA вызывает метод в ClassB, который тогда пытается получить доступ к файлу*/tmp/abc*, создавая экземпляр java.io.FileInputStream. FileInputStream конструктор создает a FilePermission, perm, как показано выше, и затем передает perm к SecurityManager's checkPermission метод. В данном случае только полномочия для ClassA и ClassB должны быть проверены, потому что весь системный код, включая FileInputStream, SecurityManager, и AccessController, автоматически получает все полномочия.

В этом примере у ClassA и ClassB есть различные характеристики кода – они прибывают из различных расположений и имеют различные подписывающие лица. Каждому, возможно, предоставили различный набор полномочий. AccessController только предоставляет доступ к требуемому файлу если Policy указывает, что обоим классам предоставили необходимое FilePermission.



**Рисунок 4, Управляющий доступом к ресурсам**

**9 XML-подписей**

XML Java API Цифровой подписи является стандартным API Java для генерирования и проверки допустимости XML-подписей.

XML-подписи могут быть применены к данным любого типа, XML или двоичного файла (см. <http://www.w3.org/TR/xmldsig-core/>). Получающаяся подпись представляется в XML. XML-подпись может использоваться, чтобы защитить Ваши данные и обеспечить целостность данных, аутентификацию сообщений, и аутентификацию подписывающего лица.

API разрабатывается, чтобы поддерживать все необходимые или рекомендуемые функции Рекомендации W3C для Синтаксиса XML-подписи и Обработки. API является расширяемым и сменным и является основанным на Архитектуре Поставщика услуг Криптографии Java.

XML Java API Цифровой подписи состоят из шести пакетов:

* javax.xml.crypto
* javax.xml.crypto.dsig
* javax.xml.crypto.dsig.keyinfo
* javax.xml.crypto.dsig.spec
* javax.xml.crypto.dom
* javax.xml.crypto.dsig.dom