Эссе по курсу "Защита информации", кафедра радиотехники, Московский физико-технический институт (ГУ МФТИ)

http://www.re.mipt.ru/infsec

Максимов Сергей, гр. 111.

Система безопасности Java.

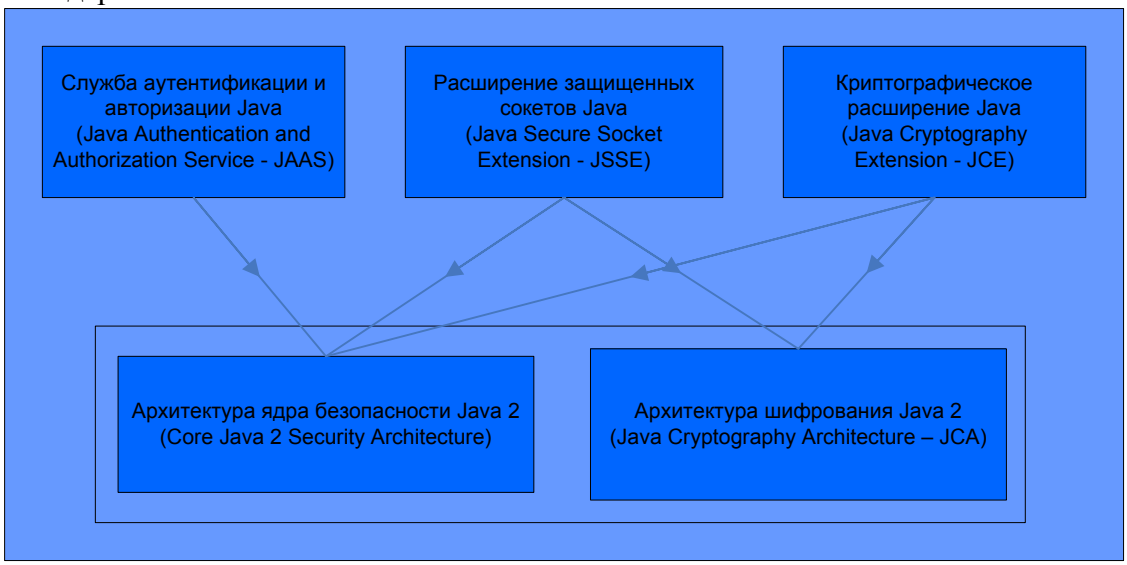
Было бы неправильно считать Java еще одним языком программирования, возникшим в конце двадцатого века. Более правильно говорить о платформе Java, которая предоставляет все необходимые средства для разработки современных надежных приложений. Изначально разработчики платформы Java руководствовались следующим принципом: “Написано однажды, работает везде” (“Write Once, Run Anywhere”), т.е. разработчик приложения на Java не должен задумываться над тем, на какой платформе будет выполняться написанный им код. Это привело к тому, что платформа Java стала одним из самых популярных средств разработки распределенных приложений. Естественно, что данная область программного обеспечения предъявляет повышенные требования к безопасности приложений.
Рассказ о средствах безопасности технологии Java логичнее всего начать со встроенных в язык средств защиты, которые помогают избежать множества неприятных моментов.

В Java удалены многие потенциально опасные возможности, такие как оператор goto или указатели. Интерпретируемый характер выполнения позволяет предотвратить выход за границы массивов, обращения к произвольным областям памяти и пустым ссылкам. Конечно, за эти дополнительные проверки приходится платить некоторой долей производительности. Хотя если учесть то, что большинство уязвимостей в современных программах существуют благодаря отсутствию подобных проверок, а программисты далеко не всегда сами о них заботятся, то такое решение кажется вполне оправданным.

Первоначально Java использовался для написания аплетов. Аплеты позволяют загружать код по сети и выполнять его на стороне клиента. Преимущества данного подхода с точки зрения пользователей очевидны. Но сразу же возникает вопрос, какую свободу вы готовы предоставить коду, который загружаете с удаленного узла? Выбор между возможностями аплетов и безопасностью клиентской системой является далеко неочевидным. Модель безопасности аплетов в Java 1.0, известная как sandbox(песочница), была довольно примитивной. Локальные программы имели доступ ко всем необходимым ресурсам, а на загруженный код накладывались серьезные ограничения, например, ограничения на доступ к файловой системе, запрет на создание новых подключений и т.д. В таких условиях сложно создать полнофункциональное приложение, поэтому разработчики очень скоро отказались от подобной политики. В соответствии с новой моделью безопасностью пользователь сам пожжет задавать ограничения на любой java код, в независимости от того является ли данный код удаленным или локальным. Появилась возможность использовать цифровую подпись, для того чтобы определить происхождение кода. После того как пользователь удостоверился в том, что код принадлежит стороне, которой можно доверять, он может самостоятельно разрешить ли ему доступ к необходимым ресурсам.

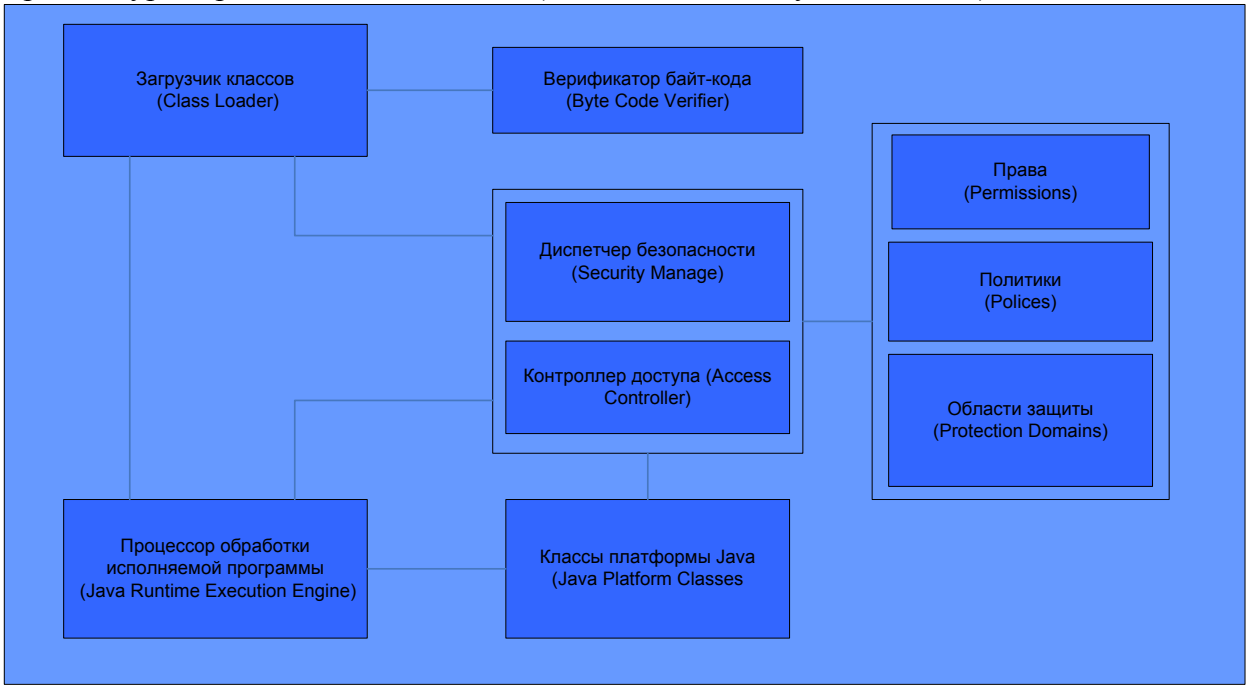
Обратимся теперь к общей структуре системы безопасности.

Стандартные компоненты системы безопасности Java 2



Для того чтобы понять, как происходит загрузка и выполнение java-кода необходимо рассмотреть архитектуру ядра безопасности Java.

Архитектура ядра безопасности Java 2 (Core Java 2 Security Architecture)



Прежде чем загружаемый класс будет зарегистрирован в среде выполнения, он отдается на проверку верификатору байт-кода(Byte Code Verifier). Вот некоторые типичные проверки, которые происходят в этот момент:

- классы должны начинаться с 0xCAFEBABE ;)
- нет лишних или потерянных байтов
- соответствие спецификации
- проверка стека

После этого загрузчик классов(Class Loader) выполняет трансляцию байт кодов. Во время трансляции перехватываются вызовы к API Java платформы, и уже диспетчер безопасности решает, корректно ли это обращение или нет в данном контексте. В Java 2 применен расширенный подход, который заключается в использовании

контроллера доступа(Access Controller) который, основываясь на конфигурируемых правах доступа(Permissions), политиках безопасности(Policies) и областях защиты(Domains) принимает решение разрешить обращение или нет. Такой подход намного более надежный, т.к. позволяет действительно гибко увязать права доступа с ценными ресурсами в определенных областях защиты. Каждая реализация процесса в java-машине допускает существование единственного диспетчера безопасности.

Архитектура шифрования Java 2 (JCA) обеспечивает основные сервисы необходимые для шифрования на платформе Java. Вообще говоря, JCA является только интерфейсом, различные реализации которого могут быть использованы. В стандартной платформе применяется базовая реализация этого интерфейса. В состав JCA входит:

- основные классы и интерфейсы, формирующие набор сервис-провайдеров JCA и API криптографических операций.
- набор классов и интерфейсов управления сертификатами
- набор классов для управления ключами

Криптографический сервис-провайдер(Cryptographic Service Provide - CSP) представляет собой класс, реализующий одну или несколько криптографических функций, которые подчиняются интерфейсам, определенным в JCA.
Вот основные классы криптографических механизмов JCA:

- MessageDigest – работа с хэшем
- Signature – работа с цифровой подписью
- KeyPairGenerator – создает пары закрытый открытый ключ
- KeyFactory и KeyStore – работа с ключами и их хранение
- CertificateFactory – создание аннулирование сертификатов
- AlgorithmParameters – задание параметров криптографических алгоритмов
- AlgorithmParametersGenerator – создание параметров для алгоритмов
- SecureRandom – генерация случайных чисел.

В стандартном пакете поставки идет CSP от Sun реализующей следующие алгоритмы и механизмы: MD5 и SHA-1, DSA, хранилище ключей JKS, производитель сертификатов по стандарту X.509 и алгоритм генерации псевдо случайных чисел SHA1PRNG(стандарт IEEE).

Криптографическое расширение Java (JCE)
Американские законы запрещают экспорт некоторых видов криптографического ПО за пределы США и Канады или разрешают экспорт с урезанными ключами. Стандартные классы из JCA не попадают под это ограничение, поэтому они поставляются в составе платформы Java 2, а JCE поставляется отдельно. Наиболее известная открытая реализация совместимая с JCE 1.2 это пакет Cryptix JCE, который включает в себя:

- Шифры Blowfish, CAST5, DES, IDEA, MARS, RC2, RC4, RC6, Rijndael, Serpent, SKIPJACK, Square, TripleDES, Twofish.
- Протоколы обмена ключами – Диффи-Хелмана
- Методы шифрования CBC, ECB, OFB
- Хэш-функции – MD2, MD4, MD5, RIPEMD-128, RIPEMD-160, SHA-0, SHA-1, Tiger
- MAC-коды – HMAC-MD2, HMAC-MD4, HMAC-MD5, HMAC-RIPEMD-128, HMAC-RIPEMD-160, HMAC-SHA-0, HMAC-SHA-1, HMAC-Tiger
- Подписи – RawDSA, RSA
- Асимметричные шифры – ElGamal, RSA

Расширение защищенных сокетов (JSSE) является стандартным интерфейсом Java для работы с SSL(Secure Socket Layer). SSL является связным протоколом для обеспечения безопасной связи на основе стека протоколов TCP/IP. Хотя может быть реализован поверх любого надежного транспортного протокола. SSL поддерживает шифрование данных, поддержку целостности данных, возможность аутентификации, неаннулируемости как клиента, так и сервера. Поверх SSL могут работать более высокоуровневые протоколы такие как HTTP и IIOP. Протокол TLS(Transport Layer Protocol) расширяет возможности SSL в плане аутентификации.
JSSE поддерживает протоколы SSL версий 2 и 3, а так же TSL первой версии.

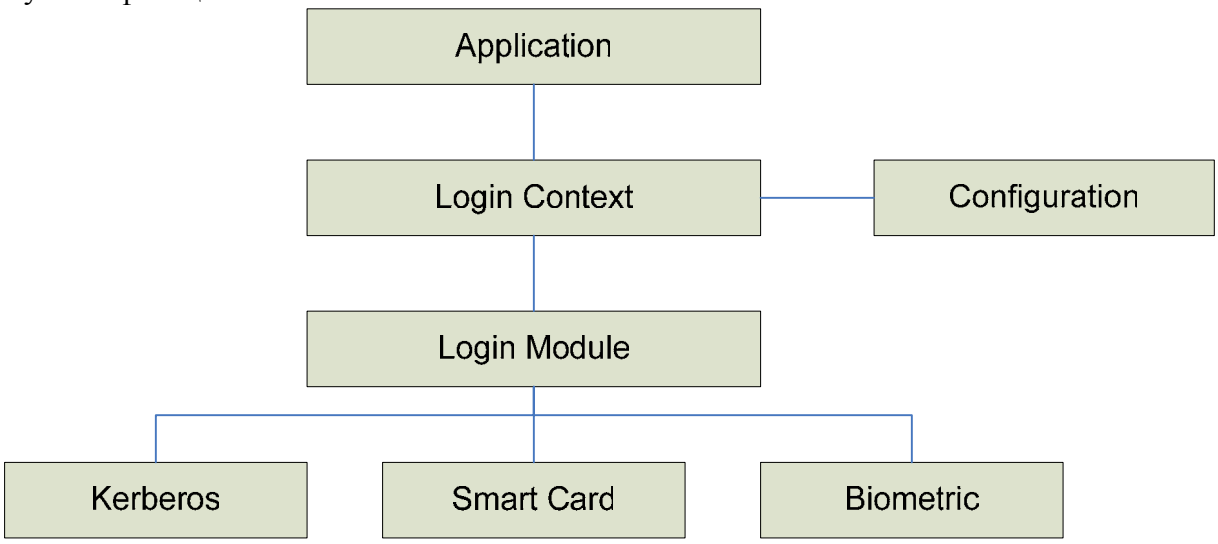
Служба аутентификации и авторизации Java(JAAS) обеспечивает стандартный способ ограничения доступа к ресурсам на основе аутентификации пользователей. Основные понятия данной службы:

- Субъект (Subject). Применяется для представления объекта или лица, претендующего на использование некоторой части системы. Subject может представлять пользователя или группу пользователей, часть системы, внешнюю систему, т.е. любую сущность, которая может использовать ценный ресурс с точки зрения безопасности.
- Принципал (Principal). У субъекта может быть несколько способов идентифицировать себя в системе. Например, если речь идет о пользователе, то он может быть представлен с помощью имени, номера кредитки, номера паспорта и так далее.
- Мандат (Credential). Некоторая информация для подтверждения полномочий данного субъекта. Credentials бывают открытые(public) и закрытые(private). К относятся, например, публичные сертификаты, открытые ключи(open keys), к закрытым относятся секретные ключи.
- Контекст входа в систему (Login Context). Объект, связывающий сервис-провайдер регистрационного модуля и саму систему. В JAAS реализован шаблон PAM (Pluggable Authentication Module - сменный модуль аутентификации), т.е. любой сервис провайдер совместимый с JAAS реализуют определенный абстрактный интерфейс, с помощью которого система может получить данные для аутентификации. Сама реализации данного модуля может быть разной от простого ввода пароля до сканирования радужной оболочки глаза.
- Configuration (Объект настроек). Login Context обращается к данному объекту, чтобы узнать каким модули регистрации необходимо загрузить. Настройки считываются из определенного файла, где описывается каждый модуль регистрации. С каждым модулем могут быть связаны следующие флаги:
    - необходимый (required) – необходимый для аутентификации модуль, в любом случае процесс будет продолжен дальше
    - обязательный (requisite) – обязательный, продолжает процесс аутентификации, в случае неудачи, процесс прерывается.
    - достаточный (sufficient) – необязательный, в случае успеха весь процесс также считается успешным
    - необязательный (optional) – необязательный, в случае неудачи процесс продолжается дальше

Процесс аутентификации происходит следующим образом, клиент запускает метод login() объекта Login Context. Объекту Login Context может быть передан уже существующий Subject с набором Principals и Credentials для аутентификации в данном контексте, либо этот объект создается в процессе аутентификации. Для этого необходимо взаимодействие с пользователем, чтобы с создаваемым объектом можно было связать некоторый набор Principals и Credentials. Для этого используется механизм обратных вызовов(callbacks). Механизм обратных вызовов позволяет следить за ходом аутентификации и обеспечивать

необходимую для этого информацию. Фактически с помощью этого механизма осуществляется взаимодействие с пользователем. Результатом аутентификации является объект Subject, с которым связаны Principals and Credentials.

Аутентификация в JAAS



После этого может быть использован механизм авторизации, который ограничивает доступ к защищенным ресурсам на основе информации полученной в процесс аутентификации. Методы Subject.doAs() и Subject.doAsPrivileged используются для выполнения небезопасных операций. Они получают на вход один аргумент объект Subject. Разрешена или нет данная операция данному субъекту, решается исходя из существующей политике безопасности, которая применена в системе. Политики безопасности настраивают в отдельном конфигурационном файле, где ставятся в соответствие определенные Principals и операции в системе.

В заключении можно сказать, что круг задач, которые приходится решать системе безопасности Java, очень широк. Это связано с тем, что данная платформа применяется для разработки очень разных приложений. Насколько успешно эти задачи решаются можно судить по тому, что платформа Java, а конкретнее J2EE, стала стандартом для разработки больших распределенных корпоративных приложений. Как известно в этой области требования к безопасности самые высокие и без надлежащей системы безопасности лидирующее положение Java платформы было бы невозможно.

Ссылки:

1) “Java Security Handbook” Jamie Jaworski and Paul Perrone
2) http://java.sun.com/security/