«Особенности обеспечения безопасности в Java (на примере JSSE)»

Эссе по курсу «Защита информации»
Студента 311 гр. МФТИ(ГУ)
Сафонова А.В.

Московский Физико-Технический Университет (Государственный Университет)
2007 г

Особенности обеспечения безопасности в Java (JSSE)

Введение

Данные, передающиеся по сети, могут быть легко перехвачены неавторизованным лицом. Если эти данные включают в себя конфиденциальную информацию, такую как пароли, номера кредитных карт и т.п., то необходимо не только закрыть доступ неавторизованному лицу к этой информации, но и не позволить подменить данные. Протоколы SSL и TLS были созданы специально для обеспечения конфиденциальности и целостности данных при передаче по телекоммуникационным сетям.

JSSE обеспечивает безопасное Интернет-соединение. Оно предоставляет структуру и реализацию Java версии протоколов SSL и TLS и включает функциональность по шифрованию данных, серверной аутентификации, поддержке целостности сообщений и, опционально, клиентской аутентификации. Используя JSSE, разработчик может обеспечить безопасную передачу данных между клиентом и сервером для приложений, использующих различные протоколы и технологии: HTTP, Telnet, FTP и т.п. поверх TCP/IP.

Ранее JSSE было расширением стандартного JDK, начиная с версии J2SDK v1.4, JSSE включено в JDK.

JSSE поддерживает SSL версий 2.0, 3.0 и TLS версии 1.0. Эти протоколы включаются в стандартный двунаправленный сокет, а JSSE API добавляет прозрачным образом поддерживаемую аутентификацию, шифрование и поддержку целостности.

JSSE API позволяет использовать другую реализацию SSL/TLS и Инфраструктуры открытых ключей (Public Key Infrastructure — PKI). Для российских разработчиков это означает, что они могут использовать сертифицированные российские реализации алгоритмов безопасности.

Особенности реализации

JSSE обладает следующими ключевыми особенностями:

- Реализовано на «чистой» Java
- Может быть адоптировано к большинству стран
- Предоставляет поддержку SSL версий 2.0 и 3.0 и реализацию версии 3.0
- Включает поддержку и реализацию TLS версии 1.0
- Включает классы по созданию безопасных каналов соединений
- Предоставляет поддержку клиентской и серверной аутентификации, которые являются частью стандартного SSL взаимодействия
- Поддерживает HTTPS
- Включает поддержку основных алгоритмов шифрования, таких как:

Криптографический алгоритм	**Криптографический процесс**	**Длина ключа (бит)**
RSA	Аутентификация и обмен ключами	2048 (аутентификация) 2048 (обмен ключами) 512 (обмен ключами)
RC4	Потоковое шифрование	128 128 (40 эффективных)
DES	Потоковое шифрование	64 (56 эффективных)

		64 (40 эффективных)
Triple DES	Потоковое шифрование	192 (112 эффективных)
Diffie-Hellman	Выработка ключей	1024 512
DSA	Аутентификация	1024

Стандартный интерфейс JSSE

Стандартный API JSSE покрывает следующие аспекты:

- Безопасные (SSL) сокеты и серверные сокеты
- Фабрики по созданию сокетов, серверных сокетов, SSL сокетов и SSL серверных сокетов
- Класс для установления безопасного HTTP URL соединения
- Поддержка доверительного управления и управления ключами (key and trust management)
- Класс для обеспечения безопасных HTTP соединений (HTTPS)

Базовые классы

`SocketFactory` И `ServerSocketFactory`

Данные классы используются для создания сокетов. Предоставляют удобный и гибкий интерфейс по созданию различных типов сокетов, позволяют использовать разные типы в одном приложении.

`SSLSocketFactory` И `SSLServerSocketFactory`

Данные классы используются для создания защищенных сокетов. Фабрики защищенных сокетов скрывают от программиста различные тонкости создания и первичной конфигурации сокетов, включая работу с ключами аутентификации, проверку сертификата узла и т.п.
Данные фабрики скрывают всю «лишнюю» информацию, касающуюся инициализации сокетов и являются хорошим решением для большинства приложений, позволяющим упростить поддержку приложения.

`SSLSocket` И `SSLServerSocket`

`SSLSocket` является расширением стандартного сокета, поддерживающим защищенный режим.

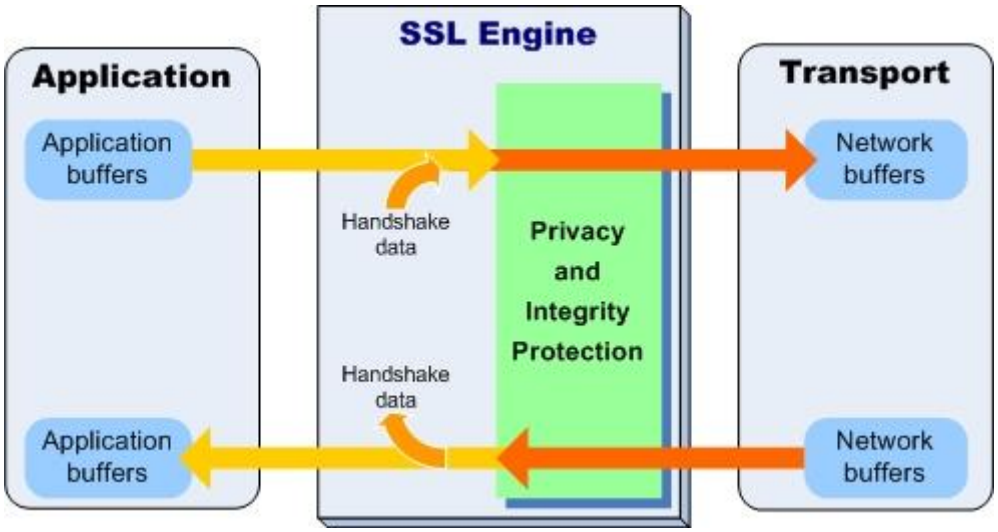
Неблокируемый ввод/вывод с использованием `SSLEngine`

SSL/TLS становятся все более популярными. Эта технология используется в огромном количестве различных приложений для всех возможных платформ и устройств. С ростом популярности возникает необходимость использовать эту технологию с различными способами ввода-вывода, различными способами организации многопоточности приложения для того, чтобы удовлетворять требованиям приложения к надежности, расширяемости, производительности и т.п. Есть необходимость использовать SSL/TLS с блокируемыми и неблокируемыми каналами ввода-вывода, асинхронным вводом-выводом и т.п. Необходимо обеспечить очень надежную, расширяемую, крайне критичную к производительности работу SSL/TLS, требующую тысячами сетевых соединений.

В версии Java 1.5.0 была значительно расширена функциональность по работе с защищенными каналами, что позволило поддерживать многочисленные организации многопоточности приложения и различные механизмы ввода-вывода. Тем не менее, такая гибкость требует от приложения некоторых действий, обеспечивающих стабильную работу SSL/TLS.

`SSLEngine`

Ключевой класс новой модели - `javax.net.ssl.SSLEngine`. Он включает в себя работу с состояниями SSL/TLS, входящими и исходящими потоками данных, поставляемыми пользователем в `SSLEngine`. На следующей диаграмме демонстрируется поток данных от приложения к `SSLEngine`, «в сеть» и обратно:



Приложение, показанное слева, заполняет буфер приложения данными (открытый текст) и передает его `SSLEngine`. `SSLEngine` обрабатывает данные из буфера, в т.ч. и всю служебную информацию, образуя тем самым закодированные с помощью SSL/TLS данные, которые передаются в сетевой буфер, предоставляемый приложением. Далее приложение уже отвечает за передачу данных необходимым образом. При получении закодированных данных, приложение помещает данные в сетевой буфер и передает его на обработку `SSLEngine`. `SSLEngine` обрабатывает информацию, получая служебную информацию и данные.
Итак, `SSLEngine` может быть в пяти различных состояниях:

1. Creation – готов к конфигурированию
2. Initial handshaking – аутентификация и договор о параметрах соединения
3. Application data – готов к обмену данными
4. Rehandshaking – повторная аутентификация, обсуждение параметров
5. Closure – готов к закрытию соединения

Некоторые известные уязвимости

1. В начале 2003 года было объявлено о найденной уязвимости: JSSE может неправильно проверять достоверность цифрового сертификата сайта. В результате, враждебный сайт может быть заверен для SSL транзакций.
   Согласно заявлению Sun, если SSLContext инициализирован через функцию SSLContext.init() с независимым образцом X509TrustManager выполнения, JSSE неверно вызовет метод isClientTrusted() для определения доверенного сертификата.

Также сообщается что Java Plug-in и Java Web Start могут некорректно утверждать цифровые сертификаты подписанных JAR файлов. В результате, злонамеренный код может подтвердить подлинность как доверенный код.
Уязвимость обнаружена в JSSE 1.0.3 и более ранних; а также в JSSE из SDK и JRE 1.4.0_01.

2. Злонамеренный пользователь может получить информацию об Cipher Block Chaining (CBC) в зашифрованной SSL/TLS сессии. Также удаленный атакующий может, при определенных обстоятельствах, раскрыть закрытый ключ SSL сервера.
Java Secure Socket Extension (JSSE) 1.x, Sun Java JRE 1.4.x, Sun Java SDK 1.4.x
Исправлено в версиях SDK и JRE 1.4.1_03

Литература:

1. http://java.sun.com/j2se/1.5.0/docs/guide/security/jsse/JSSERefGuide.html
2. http://ru.wikipedia.org/wiki/JSSE
3. http://www.securitylab.ru/