Jet Info

ИНФОРМАЦИОННЫЙ БЮЛЛЕТЕНЬ № 5 (72)/1999

Java-технология как средство создания современных корпоративных систем (с. 2)

Современная трактовка сервисов безопасности (с. 14)

КОРПОРАТИВНЫЕ СИСТЕМЫ

Java-технология как средство создания современных корпоративных систем

Олег Москаленко

СОДЕРЖАНИЕ

1. Введение 3

2. История Java как средства создания корпоративных приложений 3

3. Текущее состояние Java-технологии с корпоративной точки зрения 4

3.1. Использование Java для создания серверных приложений
3.2. Использование Java для создания клиентских приложений
3.3. Общие технологии

4. Заключение 10

5. Благодарности 11

6. Литература 11

7. Приложение. Пример использования удаленного вызова методов 11

1. Введение

Данная статья представляет собой попытку краткого обзора корпоративных технологий на основе Java. Этот обзор сознательно сделан с одной очень узкой точки зрения — гипотетического программиста-практика, не читающего ничего, кроме документации к программным продуктам: ни газет, ни рекламных объявлений, ни анонсов новых технологий и не посещающего никаких презентаций (чего, конечно, реально никогда не бывает). По мнению автора, такая сознательно суженная точка зрения становится все более важной, актуальной и полезной в наше время «войн анонсов». Именно таким специфическим взглядом можно объяснить то, что многие сделанные в этой статье выводы и утверждения отличаются от общепринятых.

Следует также сразу оговориться, что качества Java-технологий оцениваются именно с точки зрения программиста-создателя бизнес-приложений, а не с точки зрения абстрактного кодировщика. Очевидно, что программист, работающий в центре ядерных исследований или создающий спецэффекты трехмерной графики для фантастического фильма ужасов «Мозилла — сын Годзиллы», имел бы совершенно иной взгляд на достоинства и недостатки Java-технологий.

Мы надеемся, что подобная постановка вопроса окажется интересной не только техническим специалистам, но и руководителям, принимающим стратегические решения по развитию информационных систем.

Для экономии места далее опускаются вводные фразы «с точки зрения автора», «по мнению автора» и т.д. Вся статья написана с точки зрения автора и представляет исключительно его мнение.

2. История Java как средства создания корпоративных приложений

Технология Java вначале мало кем рассматривалась как серьезная платформа для корпоративных программных систем. Первое время многим казалось, что это, прежде всего, удобное средство для создания многоплатформных клиентских приложений, в основном, для Интернет (хотя, по-видимому, в число этих «многих» не входили люди, работающие в JavaSoft). Однако время шло, Java развивалась, причем в довольно неожиданном направлении, превратившись в мощную и удобную платформу для создания и развертывания корпоративных систем. Более слабы ее позиции как универсального средства создания клиентских приложений. Что касается приложений для Интернет, то здесь сколько-нибудь заметное на практике развитие остановилось из-за слабой поддержки Java web-навигаторами. Имеются следующие основания для такого вывода:

- на момент написания статьи версия Java 1.2 не поддерживалась ни одним навигатором, даже HotJava;
- версия Java 1.1 поддерживается в Internet Explorer весьма условно, есть проблемы в реализации некоторых прикладных программных интерфейсов, например, RMI;
- только сравнительно недавно Java 1.1 стала поддерживаться в Netscape Navigator;
- качество встроенной в Netscape Navigator виртуальной Java-машины — тема отдельного разговора;
- три конкурирующих несовместимых (и постоянно развивающихся) стандарта безопасности аплетов (от JavaSoft, Netscape и Microsoft) крайне затрудняют создание универсальных (рассчитанных на произвольного пользователя) аплетов;
- созданный JavaSoft Java 1.2 Plug-in следует рассматривать как спасательный круг, брошенный корпоративным разработчикам и администраторам систем, но отнюдь не как средство, которое принесет Java 1.2 на компьютеры миллионов пользователей Интернет.

Java постепенно уходит из Интернет, по крайней мере, на нынешнем витке развития. Об этом свидетельствует и направление развития Java-платформы — «аплеты», созданные средствами JDK 1.2, требуют для сносной работы мощнейших компьютеров и очень быстрых каналов связи. Простенький FTP-клиент, написанный на Java и требующий для своей работы не менее 64 Мб оперативной памяти, это не страшилка на ночь, а реальный факт, реальный коммерческий продукт. Вызывает недоумение и нестыковка действий команд Java-разработчиков из Netscape и JavaSoft.

Сторонники Java-технологии в настоящий момент должны сосредоточиться именно на корпоративных приложениях. Java весьма адекватно отражает потребности корпоративных разработчиков. Однако, если не будет решена проблема эффективного и удобного создания и выполнения аплетов и клиентских приложений (причем на обычных компьютерах, а не на суперстанциях, которыми, по-видимому, щедро оснащена JavaSoft), то Java из «великой белой надежды» может превратиться в технологию-нишу.

3. Текущее состояние Java-технологии с корпоративной точки зрения

3.1. Использование Java для создания серверных приложений

3.1.1. Общие соображения

При создании серверных приложений с помощью Java встречается минимальное число проблем и наиболее ярко проявляются достоинства Java-технологии. Здесь стоит отметить следующие положительные стороны использования Java:

- многоплатформность особенно важна именно для серверных приложений, так как в этой области, к счастью, пока не наблюдается монополии;
- программные интерфейсы Java сейчас развиты настолько, что позволяют решать стандартным образом практически любые прикладные задачи, не требующие низкоуровневого взаимодействия с аппаратурой;
- достигнута неплохая надежность работы серверных приложений (автор лично убедился в возможности создания серверного приложения, выполняющего действия ежесекундно круглые сутки в очень жестком режиме, без каких-либо сбоев и зависаний);
- удобный механизм многопотоковости облегчает создание приложений, обслуживающих одновременно множество пользователей (реализовать Corba-сервер, обслуживающий в многопотоковом режиме множество параллельных клиентских подключений, оказалось гораздо проще при помощи Java, а не C++, причем Java-программа обычно получается надежнее из-за меньшего, в среднем, числа допускаемых программистом ошибок);
- практически снята проблема скорости работы серверных приложений. Во-первых, сама технология стала достаточно зрелой и последние версии JVM, оснащенные к тому же JIT-компиляторами, предоставляют весьма эффективную среду выполнения. Во-вторых, скорость работы серверного бизнес-приложения, в основном, зависит от проработки архитектуры приложения и качества ее воплощения в программный код, а в этой области Java является, возможно, наиболее адекватным средством. В-третьих, скорость работы корпоративных бизнес-приложений, в первую очередь, зависит от эффективности взаимодействия с СУБД. С появлением нового поколения JDBC-драйверов необходимая эффективность гарантирована;
- на сервере не так остра проблема объема потребляемой оперативной памяти;
- основная проблема внутренней корпоративной разработки — преодоление хаотичности действий отдельных разработчиков и команд разработчиков — в значительной степени решается с помощью присущего Java-технологии жесткого подхода к стандартизации и реализации программного интерфейса;
- Java идеально подходит для создания промежуточного, связующего ПО, осуществляющего взаимодействие различных, часто унаследованных систем, чему особенно способствуют такие качества Java, как многоплатформность и развитые коммуникационные средства;
- именно в серверных приложениях, не загроможденных реализацией пользовательского интерфейса, наиболее ярко проявляются достоинства Java как языка программирования (в первую очередь, стройность и простота). Скорость и простота написания приложений весьма важны для корпоративных разработчиков, постоянно находящихся в цейтноте и зачастую не обремененных большим программистским опытом;
- важный фактор — широкая поддержка серверных Java-технологий всеми заметными производителями СУБД, серверов приложений и мониторов транзакций.

Наблюдения за развитием Java показывают, что в настоящее время JavaSoft больше внимания уделяет именно серверной платформе. Например, вышедшая недавно Java 2 (старое название — Java 1.2) демонстрирует завидные достижения в области производительности и масштабируемости серверных приложений, но изобилует проблемами, связанными с работой на клиентской стороне. Качество серверных технологий, встроенных в Java 1.2, заставляет настойчиво рекомендовать использовать именно этот вариант для серверов приложений.

3.1.2. Корпоративные серверные технологии

Удаленный вызов методов (RMI)

RMI является зрелой и практичной технологией для организации взаимодействия Java-программ. Класс java.rmi.Naming и пакет java.rmi.registry предоставляют удобный облегченный аналог сервиса имен OMG Corba Naming Service, позволяющий эффективно вести поиск объектов. Несмотря на название «Удаленный вызов методов» (Remote Method Invocation), иногда удобно применять эту технологию и для локального взаимодействия Java-компонентов (из соображений унификации и глобализации). Например, при создании универсальных серверов, допускающих множество вариантов конфигурирования на этапе выпол-

нения, разумно использовать RMI для организации взаимодействия всех компонентов системы, что позволит получить большую гибкость конечного программного продукта. Простые примеры использования RMI для взаимодействия объектов приведены в [5] и в приложении к данной статье.

Однако не все так просто при удаленном взаимодействии, особенно на низкоскоростных линиях связи. Коммуникации RMI опираются на программный интерфейс сериализации, что неоправдано с точки зрения эффективного использования пропускной способности сети (по крайней мере, при использовании стандартной реализации интерфейса Serializable). Поэтому, RMI имеет смысл использовать в локальных сетях и для локального взаимодействия. Лучше всего применять RMI для организации взаимодействия объектов в компонентной технологии (JavaBeans), где ярко проявляются такие достоинства RMI, как простота, унифицированность, прозрачность взаимодействия. При создании территориально распределенных систем предпочтительнее технологии более низкого (TCP/IP Sockets) или более высокого (Corba) уровней.

RMI, грубо говоря, это «Corba для чайников». Начинающий программист, открывающий для себя область распределенных вычислений, может сразу приступить к разработке несложных распределенных приложений, изучив азы RMI.

Представленная в JDK/JRE реализация RMI не отличается особой производительностью, это всего лишь «действующая модель». Компании-производители серверов приложений для Java часто предлагают собственные реализации RMI, настроенные на работу в условиях высокой нагрузки. Например, продукт WebXpress компании BEA Systems предоставляет изощренную и высококачественную реализацию RMI, основанную на оптимизированном интерфейсе Serializable и усовершенствованном механизме серверной многопотоковости. Резервы повышения производительности RMI хорошо описаны в [4].

Возможно, однако, что сейчас наблюдается процесс принципиального улучшения производительности стандартной Java-платформы. Это имеет место, по крайней мере, в области поддержки большого числа клиентов на сервере. И личный опыт автора, и результаты независимых тестов (см. [1]) показывают, что новая версия Java 1.2 способна уже в стандартом варианте (без коммерческих расширений) поддерживать действительно большое число клиентов, одновременно работающих с сервером приложения. Пока, к сожалению, таким выдающимся качеством может похвастать только версия Java 1.2 для Solaris, но есть надежда, что качественные реализации для Win32 и Linux не за горами. Личный опыт автора говорит о том, что несколько сотен подключений по протоколу RMI на платформах Solaris и Windows NT не вызывают никаких особых проблем уже для стандартной установки Java 1.2 (без продуктов третьих фирм), а на платформе Linux такая степень масштабируемости становится доступна после настройки параметров ядра системы.

Большую работу по повышению производительности Java-платформы проводит корпорация IBM, сделавшая общедоступной собственную реализацию JDK для Win32. Эта реализация отличается хорошей масштабируемостью (хотя и уступает JDK 1.2 от JavaSoft для Solaris) и повышенной производительностью.

Corba

В настоящее время Corba является наиболее адекватным средством создания распределенных корпоративных систем. Данная технология сочетает относительную простоту (особенно в сравнении с DCOM), мощность и универсальность подхода. Мы, однако, не будем рассматривать Corba как таковую, а представим свои соображения о связке Java-Corba.

Среди существующих языков программирования Java является наиболее подходящим средством выражения конструкций Corba. Для доказательства достаточно сравнить спецификации «Mapping of OMG IDL to C++» и «Mapping of OMG IDL to Java». Невооруженным глазом видно, что попытка «натянуть» концепцию Corba на C++ выглядит весьма искусственной и вынуждает пользоваться самыми сложными и сравнительно новыми конструкциями C++. Напротив, Java-интерфейсы Corba выглядят простыми и естественными.

Java позволяет (с некоторыми оговорками) решить мучительную для разработчика Corba-приложений проблему корректного выделения и освобождения памяти. При написании Java-программ полностью соблюдается стандарт на автоматическое выделение и освобождение памяти и это не требует от разработчика никаких дополнительных усилий. При написании же Corba-программ на C++ не просто приходится явно выделять/освобождать память, но и пользоваться для этого нестандартными для C++ функциями.

Большим неудобством программирования Corba-приложений на C++ является то, что очень легко допустить ошибку при манипулировании указателями на Corba-объекты из-за сложного использования конструкторов присваивания в классах C++, автоматически создаваемых IDL-компилятором. Подобная проблема полностью отсутствует в программах на Java.

На примере Corba-программ особенно ясно видна архаичность (в сравнении с Java) языка C++. В настоящее время многие поставщики Corba-реализаций предлагают средства создания

Corba-приложений (как серверных, так и клиентских) с помощью Java. Это, например, такие продукты, как M3 (BeaSystems), Visibroker (Inprise), OrbixWeb (Iona), NetDynamics (Sun Microsystems), Orbacus (OOC, Inc.).

Огромным плюсом Corba с точки зрения разработчика является наличие спецификаций (и реализаций) стандартных сервисов, соответствующих часто используемой в приложениях функциональности. Автору известны следующие доступные для использования в Java-приложениях сервисы (реализованные в коммерческих продуктах): Naming Service, Event Service, Transaction Service, Property Service, Security Service, Life Cycle Service, Trading Object Service (список, конечно, не полон).

К сожалению, не все производители строго следуют спецификациям на сервисы и часто добавляют от себя всякие «усовершенствования», что размывает единое поле Corba. Например, компания Inprise изрядно «постаралась», добавив в Visibroker функциональность, дублирующую Naming Service и Event Service, но несовместимую со спецификациями OMG (правда, для Visibroker отдельно предлагается и стандартная реализация этих сервисов). С точки зрения программиста-практика это, по меньшей мере, странно и неудобно. Как образец стандартности заслуживает одобрения объектный брокер Orbacus, практически не внесший никаких «улучшений» в спецификации OMG.

Примером специализированного брокера объектных запросов является M3 (BeaSystems). Этот продукт в значительной степени сохранил совместимость со спецификациями OMG, а необходимые расширения, в основном, добавлены к окружению времени выполнения совершенно прозрачным образом для разработчиков приложений. Это единственно правильный способ добавления расширений к стандарту в тех случаях, когда они необходимы. В случае M3 расширения оправданы, так как продукт имеет специфическую область применения — это сверхбольшие приложения с огромным количеством распределенных объектов.

Популярности использования связки Corba-Java, несомненно, будет способствовать включение в JDK 1.2 основ Corba-технологии (простой объектный брокер и Naming Service) как неотъемлемой части. Конечно, для больших распределенных приложений этого недостаточно, и нужно использовать коммерческие продукты, но разработчики смогут изучить эти технологии и составить свое мнение о них.

Следует отметить такое преимущество Corba перед RMI, как возможность в одной программной системе использовать модули, написанные на разных языках программирования. Например, из клиентской программы, написанной на Visual C++, с помощью стандартных средств Corba можно обращаться к реализациям серверных объектов, созданных на Java, причем способ вызова удаленных объектов не зависит от того, на каком языке программирования они написаны. Что же касается RMI, то в настоящее время это технология исключительно для Java.

Corba имеет и такой существенный плюс, как стандартизация реализаций брокеров запросов и протокола их взаимодействия — IIOP (Internet Inter-ORB Protocol). Это позволяет использовать различные брокеры запросов в различных подсистемах большой программной системы, что может быть очень выгодно для ее оптимального построения. Например, можно применять встроенный в Netscape Navigator Visibroker для клиентского приложения и одновременно использовать M3 как мощный масштабируемый брокер для серверного приложения.

К великому сожалению, пока не видно продвижения к стандартизации взаимодействия сервисов OMG Corba, из-за чего различные брокеры запросов не могут совместно использовать, например, Naming Service, Event Service и т.п.

Как и в случае RMI, реализация брокера запросов в Java 1.2 далеко не оптимальна по производительности. Например, сейчас в этой реализации имеется только одна модель управления потоками на сервере приложений, когда на каждый запрос создается отдельный поток и по окончании запроса он уничтожается. Конечно, такая модель создает неоправданную нагрузку на серверную систему. Для серьезных приложений лучше использовать специальные реализации Corba для Java (см. выше). Продвинутые коммерческие реализации (Visibroker, M3, Orbacus) могут задействовать модель «пул потоков», когда некоторое заранее определенное число потоков распределяется между запросами. При этом не происходит создания и уничтожения потоков для обработки клиентских запросов.

JDBC, JSQL

Подавляющее большинство корпоративных приложений взаимодействует с базами данных. Крайне важно, чтобы соответствующие средства доступа были эффективными и универсальными.

Некоторые современные реализации стандарта JDBC полностью отвечают потребностям корпоративных разработчиков. Автор проводил сравнительное тестирование скорости доступа к СУБД из программы на Java и из программы на C. Использовались: СУБД Oracle 8, «родной» драйвер JDBC, препроцессор и библиотеки Embedded SQL/C (PRO-C). Платформа тестирования — SPARC Solaris 2.5.1, процессор UltraSPARC 147 МГц. Сервер СУБД и программа-клиент СУБД работали на одном компьютере, чтобы исключить погрешность, вносимую сетевыми взаимодейст-

виями. Простые SQL-запросы по выборке нескольких строк из таблицы размером 200000 записей, по индексированным столбцам, требовали в среднем от 0.002 до 0.01 секунды. Это близко к аналогичным показателям для программ, созданных с помощью PRO-C.

Приведенные данные показывают, что разница в производительности невелика и непринципиальна, а для SQL-запросов средней и большой сложности вообще незаметна. Это, в сочетании с высокой надежностью современных драйверов JDBC, позволяет считать Java подходящим средством для реализации серверных приложений, взаимодействующих с СУБД.

Имеются и примеры противоположного рода, когда реализация JDBC явно неадекватна, однако, есть надежда, что все производители драйверов JDBC со временем «подтянутся» до приемлемого уровня.

Использование JDBC в Java-программах аналогично использованию ODBC в программах на C со всеми вытекающими последствиями для удобства написания. «Многословность», присущая JDBC, особенно неприятна в больших проектах, имеющих и без того сложные исходные тексты. Для исправления этого положения компанией Oracle была выдвинута инициатива стандарта JSQL, который представляет собой спецификации препроцессора Java/SQL и, фактически, является аналогом Embedded SQL/C. Этот стандарт был поддержан остальными производителями СУБД и в настоящее время процесс написания программ на Java, взаимодействующих с СУБД, заметно упростился.

3.2. Использование Java для создания клиентских приложений

3.2.1. Общие соображения

К сожалению, в настоящий момент ресурсы, необходимые для эффективной работы Java-программ, заметно превышают возможности типового современного компьютера клиента (по крайней мере, в России). Минимальные реальные требования к клиентской платформе для Java 1.2 таковы: процессор не слабее Pentium II 200 МГц, память 32 МБ. Рекомендуемые параметры в 2-3 раза выше.

Корпоративные клиентские программы на Java в настоящее время реально имеет смысл использовать в следующих случаях:

- необходима реальная многоплатформность;
- приложение является довольно простым и скорость его работы не очень важна;
- корпоративный стандарт на клиентские компьютеры очень высок;
- очень важны такие требования, как простота разработки, удобство сопровождения, следование современным стандартам и т.п.

По-видимому, время широкого распространения клиентских Java-приложений еще не пришло. Впрочем, если отвлечься от «ресурсной проблемы», то мнение автора о Java как средстве разработки и выполнения клиентских программ самое благоприятное.

3.2.2. Корпоративные клиентские технологии

Swing

До появления пакета Swing пользовательский интерфейс Java-программ был весьма аскетичным. Сравнение средств пакета java.awt с богатством такой библиотеки, как MFC (из комплекта Microsoft Visula C++), было, мягко говоря, не в пользу Java. Появление Swing коренным образом изменило ситуацию. Java-разработчики теперь получили возможность, не используя продукты третьих фирм, создавать программы с профессиональным интерфейсом.

Вызывает восхищение стройность и продуманность пакета — он действительно был спроектирован «снизу доверху», имелась единая идея, концепция реализации. Этим Swing выгодно отличается от многих других библиотек построения интерфейсов, которые, как правило, «сложились исторически». Важным качеством является то, что компоненты Swing — это легковесные компоненты, не загружающие операционную систему лишними системными окнами и сообщениями; кроме того, Swing содержит в себе все средства для создания «многодокументного» MDI-интерфейса. Этот стандарт интерфейса предполагает использование «внутренних» фреймов, а не «настоящих» системных окон, и давно применяется в мире Windows-приложений.

Перечисленные качества заставляют предположить, что Swing — весьма эффективный пакет. Однако на практике преимущества Swing становятся заметны только для очень сложных клиентских приложений, в которых число элементов исчисляется сотнями. Более простые интерфейсы на основе Swing требуют гораздо больше ресурсов, чем аналогичные с использованием оригинального Java AWT. Это объясняется тем, что, к сожалению, Swing очень высоко «задирает планку» начальных требований к ресурсам приложения. Происходит это в силу именно стройности и «правильности» реализации пакета. Кроме того, Swing полностью написан на Java, что, конечно, идеологически правильно, но при сложной иерархии используемых классов самые простые действия часто вызывают интенсивную работу процессора.

К сожалению, имеется одна крайне неприятная вещь: Swing невозможно использовать в России, начиная с версии Java 1.2! Русские буквы просто не показываются на платформе Windows 95 (хотя под Solaris и Linux, вроде бы, все нормально).

Этот «сильный ход» вызывает, по меньшей мере, недоумение у отечественных разработчиков. Данная проблема была известна еще до окончательного выхода Java 1.2, но не была исправлена ни в окончательной редакции, ни в вышедшей совсем недавно версии Java 1.2.1. JavaSoft хранит застенчивое молчание по поводу такого «экспортного ограничения», а отечественным разработчикам приходится по-прежнему пользоваться Java 1.1.

Программный интерфейс печати (Printer API)

Корпоративное приложение, взаимодействующее с пользователем и не использующее вывод информации на печать, является большой редкостью. До появления версии 1.2 Java часто справедливо критиковалась из-за трудности взаимодействия программного кода с принтером. Сейчас эта проблема в значительной степени решена с помощью нового пакета java.awt.print.

При использовании этого пакета пользователь имеет дело со стандартными для текущей операционной системы утилитами управления печатью, причем все особенности конкретной платформы скрыты от разработчика. Однако имеются и проблемы, связанные с тем, что нет возможности автоматически управлять принтером, без участия человека. Видимо, эта функциональность оставлена для последующих версий.

Другая проблема: новый интерфейс, как и Swing, не поддерживает русских кодировок символов. По крайней мере, тут JavaSoft последовательна в своей политике.

Еще раз RMI

Для клиентских Java-программ RMI представляется достаточно естественным средством коммуникации с сервером из-за своей простоты и удобства. Однако есть и проблемы, тесно связанные с достоинствами. Во-первых, с помощью RMI клиентская программа может общаться только с программой на Java. Во-вторых, RMI, в отличие от Corba, как уже отмечалось выше, не имеет стандартных сервисов, берущих на себя часто используемую и очень ответственную функциональность. Сейчас положение может частично измениться в связи с планами JavaSoft выпустить вариант RMI, выполненный на основе упоминавшегося выше протокола IIOP. Это позволит RMI-клиенту взаимодействовать с Corba-объектами.

Такое многообразие похожих способов взаимодействия (Corba, RMI, RMI поверх IIOP) в Java может вызвать недоумение. Однако история возникновения подобного положения весьма прозрачна. Вначале JavaSoft разработала собственный стандарт взаимодействия объектов, и это был RMI. Затем, встретив трудности на пути продвижения Java как средства создания корпоративных приложений, компания была вынуждена заняться поиском союзников в этом нелегком деле. К тому времени уже произошла поляризация производителей: часть поддерживала стандарт OMG/Corba, часть — OLE/DCOM. Понятно, что новый стандарт — RMI — был встречен весьма прохладно. JavaSoft пришлось выбрать один из двух лагерей (легко догадаться, что это был лагерь Corba). Дальнейшее развитие пошло по линии увеличения роли Corba в Java-технологии. Вначале появилась собственно реализация Corba в Java 1.2. Затем RMI начал видоизменяться в сторону Corba. С выходом RMI/IIOP данная технология превратилась в облегченное средство работы с Corba-объектами. Скорее всего, это хорошо, как для Java, так и для Corba. Остается только дождаться включения RMI/IIOP в стандартный JDK.

Еще раз Corba

Corba-клиент свободен от недостатков, присущих RMI. На уровне протокола IIOP все реализации Corba совместимы. Мы легко можем заставить взаимодействовать аплет под управлением Netscape (в который встроены клиентские библиотеки брокера запросов VisiBroker) и сервер под управлением Java 1.2, имеющий собственный брокер. Кстати, тот факт, что в Netscape Navigator встроен брокер запросов, делает его более удобным средством в корпоративных приложениях, чем Internet Explorer, такого брокера не имеющий, и не предоставляющий никаких средств интеграции собственной DCOM-модели с Corba.

При использовании Corba клиентский аплет (или полноценное клиентское Java-приложение) может взаимодействовать с сервером, написанном на любом языке, например, на C++. Это часто бывает необходимо, особенно при наличии в организации унаследованных приложений.

Надстройки Java 1.2 для навигаторов

Осознав, что поддержка Интернет-навигаторами новых версий Java задерживается, JavaSoft решила переломить ситуацию и взять процесс под свой контроль. Кое-что в этом деле удалось. Java 1.2 plug-in позволяет получить полную поддержку Java 1.2 в свежих версиях навигаторов. Следует отметить удобство и продуманность процесса установки этого продукта. Предусмотрена автоматическая установка надстройки (при ее отсутствии на компьютере клиента) по локальной сети (для Internet Explorer). Это позволяет программистам предприятия одним махом решить проблему версий Java. Поместив где-нибудь в корпоративной сети на сервере Java 1.2 plug-in, разработчики и администраторы могут быть уверены, что при первом запуске аплета, созданного для Java 1.2, на компьютер пользователя будет автоматически установлена соответствующая версия виртуальной машины. Очень удобно. Но есть не-

достаток, неожиданный для Java, — полное отсутствие многоплатформности: данная надстройка существует только для Win32!

3.3. Общие технологии

3.3.1. Интерфейс безопасности (Security API)

Для решения проблемы аутентификации и шифрования компания JavaSoft разработала набор спецификаций. Это пакеты java.security и javax.crypto. Первый из них организует общий «каркас» системы защиты приложений и содержит реализацию протоколов электронной подписи «по умолчанию», от JavaSoft. Второй набор реализует протоколы шифрования данных и подпадает под экспортные ограничения США, поэтому он не входит в стандартный дистрибутив Java. Резиденты США получают эту реализацию непосредственно от JavaSoft, остальной мир — от независимых производителей.

Модель «поставщиков услуг безопасности», примененная JavaSoft, позволяет использовать реализации протоколов защиты как сменные модули, причем в одной программе можно переключаться между ними. Совершенно свободны производители и в выборе используемых алгоритмов шифрования и электронной подписи. Обычно это стандартный набор — RSA, DSA, DES плюс какие-нибудь дополнительные алгоритмы. Например, австрийская фирма IAIK среди алгоритмов шифрования предлагает и российский ГОСТ симметричного шифрования с ключом 256 бит.

3.3.2. Corba, RMI, сервлеты как средства создания сложных многоуровневых распределенных приложений, не ограниченных стандартной жесткой схемой «клиент-сервер»

В данной статье технологии рассматривались с точки зрения самой распространенной, но далеко не единственной схемы организации корпоративного приложения — жестко заданной схемы «клиент-сервер», когда имеется сервер приложения и множество клиентов, «требующих» от сервера «исполнения взятых на себя обязательств» по предоставлению услуг. Такая звездообразная схема часто оказывается недостаточной для создания требуемой конкретным приложением функциональности. И тогда размываются барьеры между клиентами и серверами. Приведем несколько примеров возможного усложнения схемы.

Рассмотрим случай, когда серверное приложение должно вызывать в клиентском некоторые асинхронные события. Например, перед операционистом в банке должно возникать сообщение о приходе денег на определенный счет. Это, конечно, можно сделать с помощью периодического опроса сервера клиентом, но если у нас есть, допустим, 100 операционистов, контролирующих по 100 счетов каждый, то нагрузка на сервер при таком периодическом опросе будет немалой и совершенно неоправданной. Проблему сообщений клиенту легко решить с помощью Event Service (сервис передачи сообщений) при использовании Corba как средства взаимодействия, однако, возникает сразу несколько проблем, делающих такое решение далеко не универсальным:

- оно непригодно в случае использования RMI;
- оно заставляет использовать одну и ту же реализацию брокера запросов и на клиенте, и на сервере (выше мы упоминали, что реализации сервисов, в том числе и Event Service, для разных брокеров несовместимы);
- нельзя использовать реализацию брокера, не имеющую Event Service (например, ее не имеет в стандартной поставке брокер из JDK/JRE 1.2);
- такое решение ограничивает нас только асинхронными сообщениями, однако, могут понадобиться и сообщения синхронные. В нашем примере это может быть необходимость спрашивать у операциониста разрешения о снятии денег со счета (то есть сервер спрашивает у клиента: «а можно ... ?»).

Мы можем легко избежать этих проблем, отказавшись от жесткого распределения ролей в схеме «клиент-сервер» и согласившись с тем, что каждый процесс на каждом компьютере в сети может выступать как в роли сервера, так и в роли клиента. В нашем примере «клиентское» приложение у операциониста могло бы играть роль сервера для некоторых запросов «серверного приложения». Когда «сервер приложения» осуществляет перевод денег со счета, он может (с помощью параметров, приписанных к счету) определить операциониста, отвечающего за счет; затем, по имени операциониста, определить IP-адрес его компьютера и ссылку на клиентскую программу; затем «сервер приложения» может вызвать на «клиентском приложении» функцию, запускающую диалог с пользователем и возвращающую ответ последнего. При таком взаимодействии отличить принципиально «приложение-клиент» от «приложения-сервера» можно только по наличию или отсутствию пользовательского интерфейса.

Как Corba, так и RMI позволяют реализовать подобную схему без всяких специальных усилий. Таким образом, мы решаем все четыре проблемы, связанные с применением Event Service. Пожалуй, только с появлением Java такие типы взаимодействия для «клиента» и «сервера» стали простыми и естественными.

Здесь следует отметить, что данный подход является просто новым воплощением давно использующейся технологии «обратных вызовов» (call-back). Заслугой Java является то, что подобное «обратное взаимодействие» клиента и серве-

ра стало настолько простым и удобным в реализации, что теперь применение этой технологии станет обычным не только в системном программном обеспечении, но и в корпоративных системах, даже в сравнительно простых.

В [4] приведены различные схемы организации «обратного взаимодействия».

Дальнейшее усложнение предыдущего варианта — развитие ситуации вширь. При этом, как часть общего приложения может функционировать множество разнотипных клиентов и серверов, связанных различными зависимостями класса «клиент-сервер». В банковском примере мы можем вообразить, что кроме «приложения-клиента» для операциониста есть еще «приложение-клиент» для оператора карточной системы; кроме сервера банковских транзакций есть еще сервер карточной платежной системы. «Приложение-клиент» карточной системы выступает строго как «клиент» для сервера транзакций, но при этом является и «клиентом», и «сервером» для сервера карточной системы. Сервер карточной системы выступает как «клиент» сервера транзакций. Это реальная схема построения программной системы. Очевидно, что такая схема требует особенно тщательной проработки архитектуры приложения.

Подобная усложненная архитектура может быть успешно и с разумными трудозатратами реализована внутри предприятия именно при условии одновременного применения Java и Corba, как наиболее развитых и простых современных средств организации взаимодействия распределенных объектов и программных систем.

Можно усложнить схему и «вглубь», когда количество уровней программной системы «клиент-сервер» становится больше двух (не считая сервера базы данных). При этом приложения в средних уровнях являются клиентами для более низких уровней и серверами — для более высоких. Например, на этом принципе основана Java-архитектура в компании Sun Microsystems (см. [2]).

На клиентском компьютере при такой схеме работают только приложения, осуществляющие исключительно экранные функции и функции ввода с клавиатуры.

На следующем уровне — «хостов» (ни строго клиентами, ни строго серверами эти компьютеры назвать уже нельзя) размещаются приложения, подготавливающие данные для клиентских приложений и осуществляющие критичные, с точки зрения безопасности, функции — печать, доступ к диску, к различным серверам сети (возможно, глобальной) и т.п.

На более глубоком уровне находится собственно сервер (или несколько серверов) приложений, содержащий в себе логику обработки данных и обращающийся за этими данными к четвертому уровню — серверу базы данных.

Самым интересным для нас является второй уровень — уровень «суррогатного клиента». Этот «клиент» чаще всего создается с помощью Web-сервера и специальных расширений для него, позволяющих вместо CGI-процедур запускать Java-программы (сервлеты). Сервлеты поддерживаются таким Web-сервером, как Apache, а это примерно 60% всех серверов Интернет (см. http://www.netcraft.com). При использовании сервлетов нет необходимости даже в наличии Java на компьютере пользователя — достаточно обычного Web-навигатора.

4. Заключение

Основным препятствием на пути широкого распространения Java в корпоративных приложениях является качество реализации Java-платформы. В настоящее время Java несколько оторвалась от своих корней и вызывает справедливые нарекания в таких областях, как надежность, безопасность, многоплатформность (!).

Очень важна технологическая сдержанность JavaSoft: скорость роста ресурсных требований Java-платформы не должна быть выше скорости роста «вооружения» типового компьютера. Очень многие Java-разработчики были неприятно поражены степенью роста «прожорливости» Java 1.2 по сравнению с Java 1.1.

Автор видит выход в том, чтобы временно приостановить развитие Java как платформы, уделить основное внимание качеству реализации и дать возможность «подтянуться» явно поотставшим независимым производителям как коммерческих приложений, так и операционных систем (не успевающим делать качественные реализации Java для своих ОС).

Распространению Java в компаниях не способствует «проблема Linux». Автор сформулировал бы ее так: «самая перспективная ОС имеет самую плохую реализацию Java-машины». Различные независимые исследования, например, [3], подтверждают, что в настоящее время Linux становится одной из основных ОС на предприятиях, с наиболее высоким процентом роста инсталляционной базы.

К сожалению, качество реализации JVM для Linux до сих пор удручающее, и это не дает возможности Java «оседлать» волну популярности Linux. Невысокое качество JVM для Linux подтверждают как личный опыт автора, так и данные независимого тестирования (см. [1]). Никакая крупная компания — ни JavaSoft, ни IBM — не производит JVM для Linux. Для IBM это был бы особенно естественный шаг, она объявила о полной поддержке Linux в корпорациях и уже выпускает JVM для множества платформ — OS/2, Win32, AIX, MVS, OS/400 и имеет в этом деле большой опыт.

Java необходима для Linux, поскольку помогает устранить такие изначально присущие Linux недостатки, как:

- отсутствие реальной компонентной технологии (JavaBeans тут как нельзя более кстати);
- сложность создания пользовательского интерфейса. Программирование для X Window по плечу немногим специалистам. Кроме внутренне присущего Java удобства построения графического интерфейса, имеется множество удобных визуальных редакторов такого интерфейса.

Несмотря на все сложности развития Java, автор совершенно согласен с высказанной в [2] мыслью, что «Java, несомненно, культовая технология». Направление развития компьютерных технологий в первой половине 90-х годов вызывало откровенное недоумение у значительной части специалистов, на разных уровнях выражавшееся по-разному. На «школьном» уровне появился бесхитростный «пролетарский» лозунг «Windows Must Die», на «студенческом» организовалась команда разработчиков Linux, «академический» и «корпоративный» слои профессионалов нашли свое выражение в Java.

С появлением Java компьютерный мир изменился, причем степень этого изменения такова, что позволяет считать, что произошло что-то большее, чем просто появление языка программирования нового поколения. Это технологический прорыв, который уже сейчас начинает коренным образом преобразовывать ландшафт корпоративных компьютерных систем.

5. Благодарности

Автор выражает благодарность Владимиру Николаевичу Родионову (Java Certified Programmer) за консультации и обсуждение сложных вопросов.

6. Литература

1. Neffenger J. The Volano Report: Which Java platform is fastest, most scalable?. — JavaWorld, 1999, March. http://www.javaworld.com/jw-03-1999/jw-03-volanomark.html.

2. Таранов А., Цишевский В. Java в три года. — Jet Info, 1998, 11/12.

3. Отчеты IDC (International Data Corporation, http://www.idc.com): — Linux Operating System Market Overview (IDC #W18662); — What's Happening in Linux Servers? (IDC #W18831); — Strategies for Windows NT in the Enterprise (IDC #W18426).

4. Krumel A. Write high-performance RMI servers and Swing clients. — JavaWorld, 1999, April. http://www.javaworld.com/jw-04-1999/jw-04-enterprise.html.

5. Mohseril P. Exploit distributed Java computing with RMI, Part Two. — NC World, 1998, February. http://www.ncworldmag.com/ncw-02-1998/ncw-02-rmi2.html.

7. Приложение. Пример использования удаленного вызова методов

В этом примере показывается, как можно создать распределенный JavaBean-компонент. Он состоит из двух подкомпонентов — серверного и клиентского. При этом каждый подкомпонент сам по себе является полноценным (хотя и очень простым) JavaBean-компонентом и при создании приложения им можно манипулировать с помощью визуального редактора, поддерживающего спецификации JavaBeans. Клиентский компонент можно использовать в интерфейсе программы-клиента (вместо JTextField), экземпляр серверного компонента должен быть создан в процессе-сервере.

```
// Определение RMI-интерфейса, описывающего удаленное взаимодействие клиента и сервера. Методы
// интерфейса предназначены для получения и изменения единственного свойства серверного
// компонента - содержимого файла (Content).

public interface FileBeanServerInt extends java.rmi.Remote {

  // Получить свойство:
  public String getContent() throws java.rmi.RemoteException;

  // Установить свойство:
  public void setContent(String s) throws java.rmi.RemoteException;

}  // Конец интерфейса

// Клиентский компонент.
// Клиентское приложение расширяет класс JTextField (который сам по себе уже есть JavaBean),
// представляющий собой экранный компонент ввода строки символов. У клиентского компонента есть
// три новых, неунаследованных свойства: содержимое файла (Content), логическое имя сервера
// (Server) и показатель подключения к серверу (Connected). С помощью методов получения и
// установления этих свойств клиентская программа легко может синхронизировать содержимое
// строки на экране и содержимое удаленного файла.
```

```
public class FileBeanClient extends JTextField implements Serializable {

  // Константа клиентского компонента:

  private static final int port=1099;

  // Внутренние переменные класса:

  private FileBeanServerInt server=null;
  private String servername="localhost";
  private boolean connected=false;

  // Конструктор

  public FileBeanClient() {
    super();
    setEditable(false);  // Пока мы не подключились к серверу, редактировать строку нельзя
  }

  // Получить значение свойства Server

  public String getServer() {
    return servername;
  }

  // Установить значение свойства Server

  public void setServer(String s) {
    servername=s;
  }

  // Получить значение свойства Connected

  public boolean getConnected() {
    return connected;
  }

  // Установить значение свойства Connected (и при этом подключиться к серверу)

  public void setConnected(boolean b) throws java.rmi.RemoteException {
    if (b) {
      if (!connected) {
        try {
   // Находим серверный объект:
          server=(FileBeanServerInt)Naming.lookup("//"+servername+
                        ":"+port+"/FileBeanServer");
          connected=true;
        } catch(Exception ex) {
   // Мы не смогли соединиться. Напечатаем сообщение:
          System.out.println("connection exception:"+ex);
        }
      }
      if(connected) {
         setText(server.getContent());
         setEditable(true);    // При установленном подключении синхронизируем содержимое файла
                               // и строки и разрешаем редактировать эту строку
      }
    } else {
      connected=false;
      server=null;
      setEditable(false);
    }
  }

  // Получить содержимое файла

  public String getContent() throws java.rmi.RemoteException {
    //Синхронизируем содержимое:
    if (connected&&(server!=null)) server.setContent(getText());

    return getText();
  }

  // Устанавливаем новое содержимое

  public void setContent(String s) throws java.rmi.RemoteException {
    setText(s);
    // Синхронизация клиентского и серверного компонентов:
    if (connected&&(server!=null)) server.setContent(getText());
    repaint();
  }

} // Конец клиентского компонента
```

```
// Серверный компонент:

public class FileBeanServer extends UnicastRemoteObject
     implements FileBeanServerInt {

   // Константы серверного компонента:

   private static final String filename="contentFile.txt";
   private static final int port=1099;

   // Конструктор:

   public FileBeanServer() throws RemoteException {
     super();
     doRegistry(); // Зарегистрировать серверный компонент
   }

   // Процедура регистрации:

   public void doRegistry() {

     try {
       Registry registry = null;
       try {
         registry=LocateRegistry.getRegistry(port);
       } catch (Exception exr) {}
       if (registry==null) {
         registry = LocateRegistry.createRegistry(port);
       }
       registry.rebind("FileBeanServer", this);
     } catch (Exception e) {
         ...
     }

   }

   // Методы установки и получения содержимого синхронизированы
   // для минимизации возможных конфликтов

   // Установка содержимого:

   public synchronized void setContent(String s)
       throws java.rmi.RemoteException {
     if (filename!=null) {
       try {
         FileOutputStream fos=new FileOutputStream(filename);
         OutputStreamWriter osw=new OutputStreamWriter(fos);
         BufferedWriter bw=new BufferedWriter(fos);
         bw.write(s,0,s.length());
         bw.flush();
         osw.flush();
         fos.flush();
         bw.close();
         osw.close();
         fos.close();
       } catch(Exception ex) {
         throw new java.rmi.RemoteException(ex.getMessage());
       }
     }
   }

   // Получение содержимого:

   public synchronized String getContent()
       throws java.rmi.RemoteException {
     if(filename!=null) {
       try {
         FileInputStream fis=new FileInputStream(filename);
         InputStreamReader isr=new InputStreamReader(fis);
         BufferedReader br=new BufferedReader(isr);
         String out=br.readLine();
         br.close();
         isr.close();
         fis.close();
         return out;
       } catch(Exception ex) {
         throw new java.rmi.RemoteException(ex.getMessage());
       }
     }
     return "";
   }
} // Конец серверного компонента
```

Современная трактовка сервисов безопасности

Владимир Галатенко

СОДЕРЖАНИЕ

1. Введение 15

2. Особенности современных информационных систем, существенные с точки зрения безопасности 15

3. Стандарты в области безопасности распределенных систем 16

3.1. Интерпретация "Оранжевой книги" для сетевых конфигураций
3.2. Международные стандарты X.800 и X.509
3.3. Рекомендации IETF

4. Архитектурная безопасность 19

5. Сервисы безопасности 20

5.1. Идентификация/аутентификация
5.2. Разграничение доступа
5.3. Протоколирование/аудит
5.4. Экранирование
5.5. Туннелирование
5.6. Шифрование
5.7. Контроль целостности
5.8. Контроль защищенности
5.9. Обнаружение отказов и оперативное восстановление
5.10. Управление
5.11. Место сервисов безопасности в архитектуре информационных систем

6. Возможность практической реализации полного спектра сервисов безопасности в России 24

7. Заключение 24

8. Литература 24

1. Введение

Информационная безопасность относится к числу дисциплин, развивающихся чрезвычайно быстрыми темпами. Этому способствуют как общий прогресс информационных технологий, так и постоянное противоборство нападающих и защищающихся.

К сожалению, подобная динамичность объективно затрудняет обеспечение надежной защиты. Причин тому несколько:

- повышение быстродействия микросхем, развитие архитектур с высокой степенью параллелизма позволяет методом грубой силы преодолевать барьеры (прежде всего криптографические), ранее казавшиеся неприступными;
- развитие сетей, увеличение числа связей между информационными системами, рост пропускной способности каналов расширяют число злоумышленников, имеющих техническую возможность осуществить нападение;
- появление новых информационных сервисов ведет и к появлению новых угроз как «внутри» сервисов, так и на их стыках;
- конкуренция среди производителей программного обеспечения заставляет сокращать сроки разработки, что ведет к снижению качества тестирования и выпуску продуктов с дефектами защиты;
- навязываемая потребителям парадигма постоянного наращивания аппаратного и программного обеспечения вступает в конфликт с бюджетными ограничениями, из-за чего снижается доля ассигнований на безопасность.

Если ограничиться только программно-технической гранью информационной безопасности (что мы и делаем в данной статье), то можно констатировать, что развитие затрагивает все ее уровни — от теоретических основ и международных стандартов до оперативного администрирования. Конечно, накладывать заплаты на программные системы приходится гораздо чаще, чем пересматривать стандарты, но сверять интерпретацию этих стандартов с реальным положением дел раз в 2 — 3 года необходимо.

В настоящей статье мы, отправляясь от основополагающих стандартов в области безопасности распределенных систем, попытаемся проинтерпретировать содержащиеся в них положения применительно к современным конфигурациям и коммерчески доступным защитным средствам. В заключительной части статьи будет проанализирована применимость этой интерпретации к российским условиям.

2. Особенности современных информационных систем, существенные с точки зрения безопасности

Рассмотрим базовые компоненты информационной системы типичной современной организации (см. рис. 1). С точки зрения безопасности существенными представляются следующие моменты:

- корпоративная сеть имеет несколько территориально разнесенных частей (поскольку организация располагается на нескольких производственных площадках), связи между которыми находятся в ведении внешнего поставщика сетевых услуг, выходя за пределы контролируемой зоны;
- корпоративная сеть имеет одно или несколько подключений к Интернет;

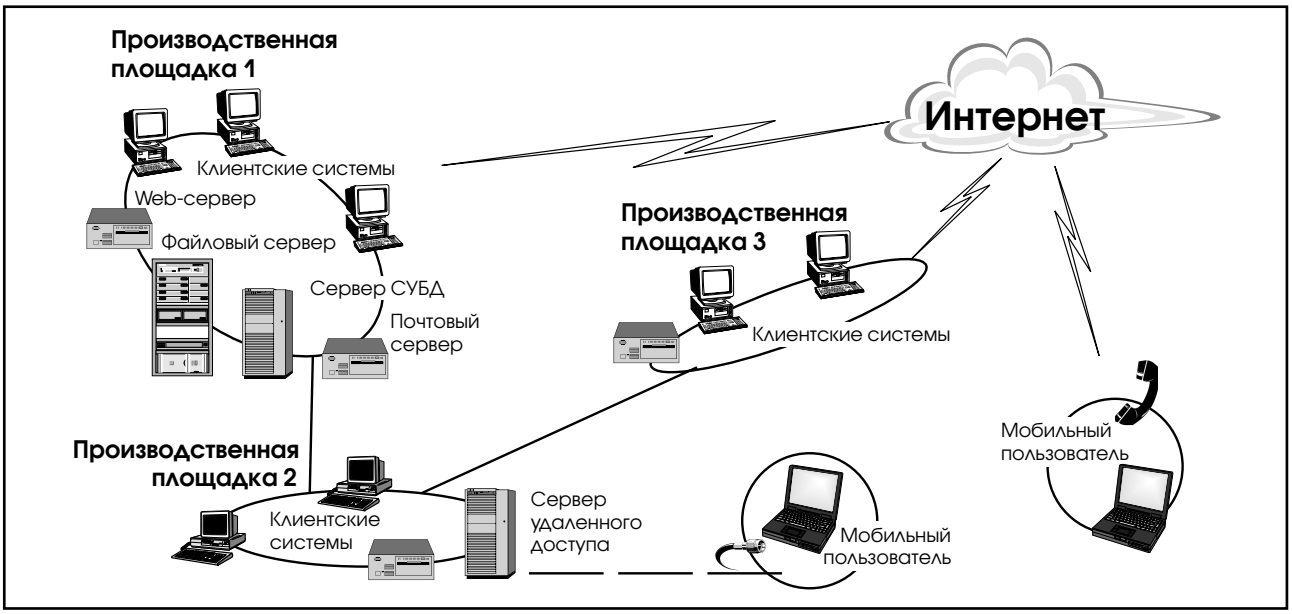

Рис. 1. Базовые компоненты информационной системы типичной современной организации.

- на каждой из производственных площадок могут находиться критически важные серверы, в доступе к которым нуждаются работники, базирующиеся на других площадках, мобильные работники и, возможно, сотрудники сторонних организаций и другие внешние пользователи;
- для доступа пользователей могут применяться не только компьютеры, но и потребительские устройства, использующие, в частности, беспроводную связь;
- в течение сеанса работы пользователю приходится обращаться к нескольким информационным сервисам, опирающимся на разные аппаратно-программные платформы;
- к доступности информационных сервисов предъявляются жесткие требования, обычно выражающиеся в необходимости круглосуточного функционирования с максимальным временем простоя порядка минут или десятков минут;
- информационная система представляет собой сеть с активными агентами, то есть в процессе работы программные компоненты, такие как аплеты или сервлеты, передаются с одной машины на другую и выполняются в целевой среде, поддерживая связь с удаленными компонентами;
- не все пользовательские системы контролируются администраторами организации;
- программное обеспечение, особенно полученное по сети, не может считаться безопасным, в нем могут присутствовать зловредные элементы или ошибки, создающие слабости в защите;
- конфигурация информационной системы постоянно изменяется на уровнях административных данных, программ и аппаратуры (меняется состав пользователей, их привилегии, версии программ, появляются новые сервисы, новая аппаратура и т.п.).

Следует учитывать также, что основная угроза информационной безопасности организаций по-прежнему исходит не от внешних хакеров, а от собственных сотрудников, по той или иной причине не являющихся лояльными.

В силу изложенных причин далее будут рассматриваться распределенные, разнородные, многосервисные, эволюционирующие системы. Соответственно, нас будут интересовать стандарты и решения, ориентированные на подобные конфигурации.

3. Стандарты в области безопасности распределенных систем

Стандартов в области безопасности распределенных систем много. Мы выделим три основные группы:

- интерпретация «Оранжевой книги» для сетевых конфигураций [1];
- международные стандарты X.800 [2] и X.509 [3];
- рекомендации IETF.

Стандарты будут интересовать нас с конструктивной точки зрения, то есть как руководство к действию, способное помочь при разработке архитектуры и реализации современных информационных систем.

3.1. Интерпретация «Оранжевой книги» для сетевых конфигураций

«Оранжевая книга» и близкие ей по духу работы (такие, например, как первые Руководящие документы Гостехкомиссии России) сейчас представляют исключительно исторический интерес. Дело не в том, что устарели сформулированные в них идеи. Просто исчезли объекты и субъекты, на которые они были ориентированы, — изолированные компьютеры с надежным программным обеспечением и злоумышленные пользователи, располагающие только штатными средствами. Можно продолжать сертификацию по классическим образцам, но когда на выходе оказывается Windows NT без НГМД и сетевого интерфейса, становится понятно, что такая сертификация является формальной.

Существенно более интересной является интерпретация «Оранжевой книги» для сетевых конфигураций. В ней содержатся две основные группы идей:

- как получить сетевую надежную вычислительную базу;
- какие сервисы безопасности должны присутствовать в сетевой среде.

Согласно «Интерпретации ...», сетевая надежная вычислительная база формируется из всех частей всех компонентов сети, обеспечивающих информационную безопасность. Надежная сетевая система должна обеспечивать такое распределение защитных механизмов, чтобы общая политика безопасности проводилась в жизнь, несмотря на уязвимость коммуникационных путей и на параллельную, асинхронную работу компонентов.

Основное назначение надежной вычислительной базы — выполнять функции монитора обращений. В «Интерпретации ...» формулируется следующее фундаментальное утверждение.

Пусть каждый субъект может осуществлять непосредственный доступ к объектам только в пределах одного компонента. Пусть, далее, каждый компонент содержит свой монитор обращений, отслеживающий все локальные попытки доступа, и все мониторы проводят в жизнь согласованную политику безопасности. Пусть, наконец, коммуникационные каналы, связывающие компоненты, сохраняют конфиденциальность и целостность передаваемой информации. Тогда сово-

купность всех мониторов образует единый монитор обращений для всей сетевой конфигурации.

Мы обратим внимание на два момента, содержащиеся в приведенном утверждении:

- необходимость выработки и проведения в жизнь единой политики безопасности;
- необходимость обеспечения конфиденциальности и целостности при сетевых взаимодействиях.

Подчеркнем, что нужно не просто один раз выработать единую политику безопасности. Требуется контролировать сохранение согласованности при всех административных действиях. Подобный контроль — одна из важнейших составляющих управления информационными системами.

Требование конфиденциальности и целостности коммуникаций впрямую обращается к такому понятию, как виртуальная частная сеть. Прочие меры безопасности (например, использование выделенных каналов, контролируемых организацией) представляются неэффективными.

«Интерпретация ...», благодаря компонентному подходу, сделала возможным анализ отдельных сервисов безопасности и их комплексирование для получения защищенных конфигураций. Кроме того, она, по сравнению с «Оранжевой книгой», принципиальным образом расширила спектр рассматриваемых сервисов безопасности. Помимо традиционных:

- идентификации/аутентификации;
- управления доступом;
- протоколирования/аудита

появились:

- проверка подлинности и корректности функционирования компонентов перед их включением в сеть;
- протокол оперативной взаимной проверки компонентами живучести и корректности функционирования друг друга;
- криптография как основа обеспечения конфиденциальности и контроля целостности.

Рассмотрение вопросов доступности — одно из важнейших достоинств «Интерпретации ...». Информационный сервис перестает быть доступным, когда пропускная способность коммуникационных каналов падает ниже минимально допустимого уровня или сервис не в состоянии обслуживать запросы. Удаленный ресурс может стать недоступным и вследствие нарушения равноправия в обслуживании пользователей. Надежная система должна быть в состоянии обнаруживать ситуации недоступности, уметь возвращаться к нормальной работе и противостоять атакам на доступность.

Для обеспечения доступности (непрерывности функционирования) могут применяться следующие защитные меры:

- Внесение в конфигурацию той или иной формы избыточности (резервное оборудование, запасные каналы связи и т.п.). Это элемент архитектурной безопасности, рассматриваемой нами в следующем разделе.
- Наличие средств обнаружения отказов. Если требуется постоянная высокая готовность, необходим специализированный сервис. В остальных случаях достаточно протоколирования/аудита в квазиреальном времени.
- Наличие средств реконфигурирования для восстановления, изоляции и/или замены компонентов, отказавших или подвергшихся атаке на доступность. Это или специализированная функция, или одна из функций управления.
- Рассредоточенность сетевого управления, отсутствие единой точки отказа. Это, как и следующий пункт, — элементы архитектурной безопасности.
- Выделение подсетей и изоляция групп пользователей друг от друга. Данная мера ограничивает зону поражения при возможных нарушениях информационной безопасности.

Каждый компонент, вообще говоря, не обязан поддерживать все перечисленные выше сервисы безопасности. Важно, чтобы он обладал программными и/или протокольными интерфейсами для получения недостающих сервисов от других компонентов и чтобы не существовало возможности обхода основных и дополнительных защитных средств.

Интерпретация «Оранжевой книги» для сетевых конфигураций увидела свет в 1987 году. Нашла ли она отражение в отечественных нормативных документах по информационной безопасности? Во многих ли информационных системах реализованы изложенные в ней архитектурные принципы, задействованы описанные сервисы безопасности? Увы, на все эти вопросы приходится ответить отрицательно.

3.2. Международные стандарты X.800 и X.509

Стандарт X.800, появившийся приблизительно в то же время, что и «Интерпретация ...», описывает основы безопасности в привязке к эталонной семиуровневой модели. Это довольно обширный документ. Мы совсем коротко остановимся на предлагаемых в нем сервисах (функциях) безопасности и на администрировании средств безопасности.

Стандарт предусматривает следующие сервисы безопасности:

- Аутентификация. Имеются в виду две разные вещи: аутентификация партнеров по общению и аутентификация источника данных.
- Управление доступом. Оно обеспечивает защиту от несанкционированного использования ресурсов, доступных по сети.

- Конфиденциальность данных. В X.800 под этим названием объединены существенно разные вещи — от защиты отдельной порции данных до конфиденциальности трафика. На наш взгляд, подобное объединение носит формальный характер, от него нет ни теоретической, ни практической пользы.
- Целостность данных. Данный сервис подразделяется на подвиды в зависимости от того, что контролируется — целостность сообщений или потока данных, обеспечивается ли восстановление в случае нарушения целостности.
- Неотказуемость. Данный сервис относится к прикладному уровню, то есть имеется в виду невозможность отказаться от содержательных действий, таких, например, как отправка или прочтение письма. Трудно сказать, насколько целесообразно выделение этого сервиса, поскольку на практике его покрывают протоколирование и аутентификация источника данных с контролем целостности.

Администрирование средств безопасности включает в себя распространение информации, необходимой для работы сервисов безопасности, а также сбор и анализ информации об их функционировании. Примерами могут служить распространение криптографических ключей, установка прав доступа, анализ регистрационного журнала и т.п.

Концептуальной основой администрирования является информационная база управления безопасностью. Эта база может не существовать как единое (распределенное) хранилище, но каждый компонент системы должен располагать информацией, достаточной для проведения в жизнь избранной политики безопасности.

Отметим, что в условиях глобальной связности администрирование перестает быть внутренним делом организации. Во-первых, плохо защищенная система может стать плацдармом для подготовки и проведения злоумышленных действий. Во-вторых, прослеживание нарушителя эффективно лишь при согласованных действиях многих администраторов. К последнему замечанию мы вернемся в разделе «Протоколирование/аудит».

Стандарт X.509 описывает процедуру аутентификации с использованием службы каталогов. Впрочем, наиболее ценной в стандарте оказалась не сама процедура, а ее служебный элемент — структура сертификатов, хранящих имя пользователя, криптографические ключи и сопутствующую информацию. Подобные сертификаты — важнейший элемент современных схем аутентификации и контроля целостности.

3.3. Рекомендации IETF

Сообществом Интернет под эгидой Тематической группы по технологии Интернет (Internet Engineering Task Force, IETF) разработано много рекомендаций по отдельным аспектам сетевой безопасности. Тем не менее, какой-либо целостной концепции или архитектуры безопасности пока предложено не было («Руководство по информационной безопасности предприятия», Site Security Handbook, RFC 2196 [4], разумеется, не в счет).

Рекомендации периодически организуемых конференций по архитектуре безопасности Интернет (последняя из которых состоялась в марте 1997 года, см. [5]) носят весьма общий, а порой и формальный характер. Основная идея состоит в том, чтобы средствами оконечных систем обеспечивать сквозную безопасность. От сетевой инфраструктуры в лучшем случае ожидается устойчивость по отношению к атакам на доступность. На наш взгляд, подход, при котором сеть трактуется как пассивная сущность, предназначенная исключительно для транспортировки данных, является устаревшим. Эту мысль подтверждают и многочисленные публикации об интеллектуальности Интернет нового поколения (Интернет-2).

В трудах конференции 1997 года выделены базовые протоколы, наиболее полезные с точки зрения безопасности. К ним относятся IPsec, DNSsec, S/MIME, X.509v3, TLS и ассоциированные с ними. Далее мы кратко остановимся на двух — IPsec и TLS. Здесь же заметим, что существование (и важнейшая роль) спецификаций DNSsec опровергают положение о том, что от сетевой инфраструктуры ожидается лишь высокая доступность.

Наиболее проработанными на сегодняшний день являются вопросы защиты на IP-уровне. Спецификации семейства IPsec (см., например, [6]) регламентируют следующие аспекты:

- управление доступом;
- контроль целостности на уровне пакетов;
- аутентификация источника данных;
- защита от воспроизведения;
- конфиденциальность (включая частичную защиту от анализа трафика);
- администрирование (управление криптографическими ключами).

Протоколы обеспечения аутентичности и конфиденциальности могут использоваться в двух режимах: транспортном и туннельном. В первом случае защищается только содержимое пакетов и, быть может, некоторые поля заголовков. Как правило, транспортный режим используется хостами. В туннельном режиме защищается весь пакет — он инкапсулируется в другой IP-пакет. Туннельный режим обычно реализуют на специально выделенных защитных шлюзах (в роли которых могут выступать маршрутизаторы или межсетевые экраны). Это обстоятельство будет рассмотрено детальнее в разделе «Туннелирование».

Следует отметить, что IP-уровень можно считать оптимальным для размещения защитных

средств, поскольку при этом достигается удачный компромисс между защищенностью, эффективностью функционирования и прозрачностью для приложений. Стандартизованными механизмами IP-безопасности могут (и должны) пользоваться протоколы более высоких уровней и, в частности, управляющие протоколы, протоколы конфигурирования и маршрутизации.

На транспортном уровне аутентичность, конфиденциальность и целостность потоков данных обеспечивается протоколом TLS (Transport Layer Security, RFC 2246 [7]). Подчеркнем, что здесь объектом защиты являются не отдельные сетевые пакеты, а именно потоки данных (последовательности пакетов). Злоумышленник не сможет переупорядочить пакеты, удалить некоторые из них или вставить свои.

На основе TLS могут строиться защищенные протоколы прикладного уровня. В частности, предложены спецификации для HTTP над TLS.

4. Архитектурная безопасность

Сервисы безопасности, какими бы мощными и стойкими они ни были, сами по себе не могут гарантировать надежность программно-технического уровня защиты. Только разумная, проверенная архитектура способна сделать эффективным объединение сервисов, обеспечить управляемость информационной системы, ее способность развиваться и противостоять новым угрозам при сохранении таких свойств, как высокая производительность, простота и удобство использования.

В контексте данной статьи важнейшим является архитектурный принцип невозможности обхода защитных средств. Опасность обхода существует по двум причинам:

- при реализации системы защиты в виде совокупности сервисов может существовать способ миновать отдельные защитные средства, нарушая тем самым целостность планируемой цепочки сервисов безопасности (пример — наличие модемных выходов в Интернет, идущих в обход межсетевых экранов);
- при построении информационной системы в многоуровневой архитектуре клиент/сервер может существовать способ обхода одного или нескольких уровней, на которых сосредоточены средства защиты.

Последнее обстоятельство можно проиллюстрировать простым примером. Пусть в системе, построенной в архитектуре Интернет/Интранет, в качестве информационного концентратора используется Web-сервер (см. рис. 2). Пусть, далее, на этом сервере сосредоточены средства аутентификации, управления доступом и протоколирования/аудита. Если злоумышленник на клиентской системе способен напрямую воспользоваться программным интерфейсом, предоставляемым сервером СУБД, он тем самым обойдет штатные защитные средства и, возможно, получит неконтролируемый доступ к базам данных.

Еще одним важным архитектурным принципом следует признать минимизацию объема защитных средств, выносимых на клиентские системы. Причин тому несколько:

- для доступа в корпоративную сеть могут использовать потребительские устройства с ограниченной функциональностью;
- конфигурацию клиентских систем трудно или невозможно контролировать.

К необходимому минимуму следует отнести реализацию сервисов безопасности на сетевом и транспортном уровнях стека протоколов TCP/IP и поддержку механизмов аутентификации, устойчивых к сетевым угрозам.

Принцип простоты использования нацелен на то, чтобы сделать работу сервисов безопасности прозрачной для пользователей. Полностью добиться этого, конечно, невозможно (например,

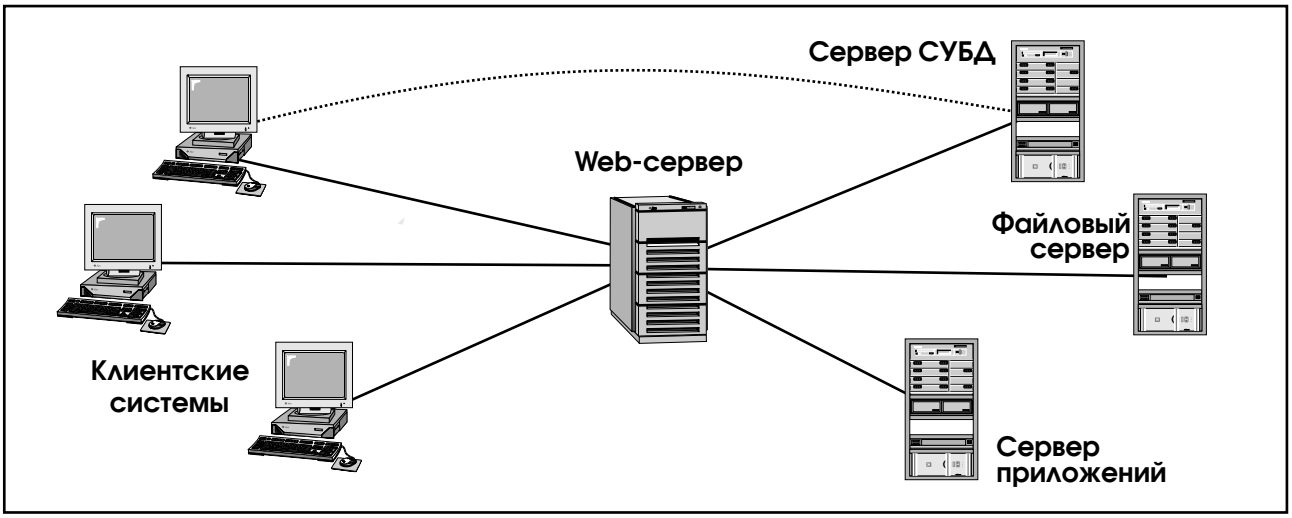

Рис. 2. Опасность обхода защитных средств в многоуровневой архитектуре клиент/сервер.

без аутентификации с участием пользователя не обойтись), но некий порог сложности превышать нельзя. Пользователи — самое слабое звено любой системы, и чем меньше они должны (и могут) сделать, тем лучше.

Важность принципа простоты архитектуры доказана всем ходом развития информационных технологий, кризисами программирования, неудачами при создании больших систем. В полной мере он относится и к подсистеме безопасности, причем на всех уровнях — от реализации отдельных функций до объединения сервисов. Следует стремиться к минимизации числа связей между компонентами информационной системы, поскольку именно оно определяет сложность. Перефразируя поговорку, можно утверждать, что в информационных технологиях «иная сложность хуже воровства».

Вообще говоря, с принципом простоты архитектуры конфликтует необходимость внесения в систему определенной избыточности, обеспечивающей устойчивость по отношению к сбоям и отказам. По целому ряду причин реальная архитектура оказывается результатом многочисленных компромиссов, однако, если опираться на отработанные решения по повышению доступности, можно не слишком поступиться простотой и эффективностью использования ресурсов.

5. Сервисы безопасности

Мы приступаем к описанию сервисов безопасности, совокупность которых, на наш взгляд, достаточна для построения защиты, соответствующей современным требованиям.

5.1. Идентификация/аутентификация

Современные средства идентификации/аутентификации должны удовлетворять двум условиям:

- быть устойчивыми к сетевым угрозам (пассивному и активному прослушиванию сети);
- поддерживать концепцию единого входа в сеть.

Первое требование можно выполнить, используя криптографические методы. (Еще раз подчеркнем тот очевидный факт, что современная криптография есть нечто гораздо большее, чем шифрование; соответственно, разные ветви этой дисциплины нуждаются в дифференцированном подходе с нормативной точки зрения.) В настоящее время общепринятыми являются подходы, основанные на системе Kerberos или службе каталогов с сертификатами в стандарте X.509.

Единый вход в сеть — это, в первую очередь, требование удобства для пользователей. Если в корпоративной сети много информационных сервисов, допускающих независимое обращение, то многократная идентификация/аутентификация становится слишком обременительной. К сожалению, пока нельзя сказать, что единый вход в сеть стал нормой, доминирующие решения пока не сформировались.

Дополнительные удобства создает применение биометрических методов аутентификации, основанных на анализе отпечатков (точнее, результатов сканирования) пальцев. В отличие от специальных карт, которые нужно хранить, пальцы «всегда под рукой» (правда, под рукой должен быть и сканер). Подчеркнем, что и здесь защита от нарушения целостности и перехвата с последующим воспроизведением осуществляется методами криптографии.

5.2. Разграничение доступа

Разграничение доступа, вероятно, является самой исследованной областью информационной безопасности. «Дискреционное» и «мандатное» управление вошли во все теоретические курсы и критерии оценки. Доминируют они и на практике.

К сожалению, в настоящее время следует признать устаревшим (или, по крайней мере, не полностью соответствующим действительности) положение о том, что разграничение доступа направлено на защиту от злоумышленных пользователей. Современные информационные системы характеризуются чрезвычайной сложностью и их внутренние ошибки представляют не меньшую опасность.

Динамичность современной программной среды в сочетании со сложностью отдельных компонентов существенно сужает область применимости самой употребительной — дискреционной модели управления доступом (называемой также моделью с произвольным управлением). При определении допустимости доступа важно не только (и не столько) то, кто обратился к объекту, но и то, какова семантика действия. Без привлечения семантики нельзя выявить троянские программы, противостоять которым произвольное управление доступом, как известно, не в состоянии.

В последнее время появляются новые модели управления доступом, например, модель «песочницы» в Java-технологии. К сожалению, и она не учитывает семантику программ, что, на наш взгляд, является основной причиной выявляемых слабостей в системе безопасности.

Активно развиваемое ролевое управление доступом решает не столько проблемы безопасности, сколько улучшает управляемость систем (что, конечно, очень важно). Суть его в том, что между пользователями и их привилегиями помещаются промежуточные сущности — роли. Для каждого пользователя одновременно могут быть активны-

ми несколько ролей, каждая из которых дает ему определенные права.

Мы уже отмечали, что сложность информационной системы характеризуется, прежде всего, числом имеющихся в ней связей. Поскольку ролей много меньше, чем пользователей и привилегий, их (ролей) использование способствует понижению сложности и, следовательно, улучшению управляемости. Кроме того, на основании ролевой модели можно реализовать такие важные принципы, как разделение обязанностей (невозможность в одиночку скомпрометировать критически важный процесс). Между ролями могут быть определены статические или динамические отношения несовместимости (невозможности одному субъекту по очереди или одновременно активизировать обе роли), что и обеспечивает требуемую защиту.

Для некоторых употребительных сервисов, таких как Web, ролевое управление доступом может быть реализовано относительно просто (в Web-случае — на основе cgi-процедур). Правда, следует позаботиться о средствах администрирования, но, разумеется, существуют и они. Так что в данном случае слово за системными администраторами.

5.3. Протоколирование/аудит

Протоколирование/аудит традиционно являлись последним рубежом обороны, обеспечивающим анализ последствий нарушения информационной безопасности и выявление злоумышленников. Такой аудит можно назвать пассивным.

Довольно очевидным обобщением пассивного аудита для сетевой среды является совместный анализ регистрационных журналов отдельных компонентов на предмет выявления противоречий, что важно в случаях, когда злоумышленнику удалось отключить протоколирование или модифицировать журналы.

В современный арсенал защитных средств несколько лет назад вошел активный аудит, направленный на выявление подозрительных действий в реальном масштабе времени. Активный аудит включает два вида действий:

- выявление нетипичного поведения (пользователей, программ или аппаратуры);
- выявление начала злоумышленной активности.

Нетипичное поведение выявляется статистическими методами, путем сопоставления с предварительно полученными образцами. Начало злоумышленной активности обнаруживается по совпадению с сигнатурами известных атак. За обнаружением следует заранее запрограммированная реакция (как минимум — информирование системного администратора, как максимум — контратака на систему предполагаемого злоумышленника).

Важным элементом современной трактовки протоколирования/аудита является протокол автоматизированного обмена информацией о нарушениях безопасности между корпоративными системами, подключенными к одной внешней сети. Работа над этим протоколом ведется под эгидой IETF. В наше время системы не могут считаться изолированными, они не должны жить по закону «каждый за себя»; угрозам следует противостоять сообща.

5.4. Экранирование

Экранирование как сервис безопасности выполняет следующие функции:

- разграничение межсетевого доступа путем фильтрации передаваемых данных;
- преобразование передаваемых данных.

Современные межсетевые экраны фильтруют данные на основе заранее заданной базы правил, что позволяет, по сравнению с традиционными операционными системами, реализовывать гораздо более гибкую политику безопасности. При комплексной фильтрации, охватывающей сетевой, транспортный и прикладной уровни, в правилах могут фигурировать сетевые адреса, количество переданных данных, операции прикладного уровня, параметры окружения (например, время) и т.п.

Преобразование передаваемых данных может затрагивать как служебные поля пакетов, так и прикладные данные. В первом случае обычно имеется в виду трансляция адресов, помогающая скрыть топологию защищаемой системы. Это — уникальное свойство сервиса экранирования, позволяющее скрывать существование некоторых объектов доступа. Преобразование данных может состоять, например, в их шифровании.

В процессе фильтрации (точнее, параллельно с ней) может выполняться дополнительный контроль (например, антивирусный). Возможны и дополнительные преобразования, наиболее актуальным из которых является исправление заголовков или иной служебной информации, ставшей некорректной после наступления 2000 года. Отметим, впрочем, что протоколы TCP/IP практически свободны от Проблемы 2000.

Применение межсетевого экранирования поставщиками Интернет-услуг в соответствии с рекомендациями [8] позволило бы существенно снизить шансы злоумышленников и облегчить их прослеживание. Данная мера еще раз показывает, как важно рассматривать каждую информационную систему как часть глобальной инфраструктуры и принимать на себя долю ответственности за общую информационную безопасность.

5.5. Туннелирование

На наш взгляд, туннелирование, как и экранирование, следует рассматривать как самостоятельный сервис безопасности. Его суть состоит в том, чтобы «упаковать» передаваемую порцию данных, вместе со служебными полями, в новый «конверт». Данный сервис может применяться для нескольких целей:

- осуществление перехода между сетями с разными протоколами (например, IPv4 и IPv6);
- обеспечение конфиденциальности и целостности всей передаваемой порции, включая служебные поля.

Туннелирование может применяться как на сетевом, так и прикладном уровнях. Например, стандартизовано туннелирование для IP и двойное конвертование для почты X.400.

Комбинация туннелирования и шифрования (с необходимой криптографической инфраструктурой) на выделенных шлюзах позволяет реализовать такое важное в современных условиях защитное средство, как виртуальные частные сети. Такие сети, наложенные обычно поверх Интернет, существенно дешевле и гораздо безопаснее, чем действительно собственные сети организации, построенные на выделенных каналах. Коммуникации на всем их протяжении физически защитить невозможно, поэтому лучше изначально исходить из предположения об уязвимости и соответственно обеспечивать защиту. Современные протоколы, направленные на поддержку классов обслуживания, помогут гарантировать для виртуальных частных сетей заданную пропускную способность, величину задержек и т.п., ликвидируя тем самым единственное на сегодняшний день реальное преимущество сетей собственных.

5.6. Шифрование

Шифрование — важнейшее средство обеспечения конфиденциальности и, одновременно, самое конфликтное место информационной безопасности (практически во всех странах, не только в России).

Мы, разумеется, не будем вдаваться в тонкости криптографии. Нам хотелось бы обратить внимание на то, что у компьютерной криптографии две стороны — собственно криптографическая и интерфейсная, позволяющая сопрягаться с другими частями информационной системы. Важно, чтобы были обеспечены достаточное функциональное богатство интерфейсов и их стандартизация. Криптографией, в особенности шифрованием, должны, разумеется, заниматься профессионалы. От них требуется разработка защищенных инвариантных компонентов, которые можно было бы свободно (по крайней мере, с технической точки зрения) встраивать в существующие и перспективные конфигурации.

Отметим, что у современного шифрования есть и внутренние проблемы, как технические, так и нормативные. Из технических наиболее острой является проблема производительности. Программная реализация на универсальных процессорах не является адекватным средством (здесь можно провести аналогию с компрессией видеоизображений). Еще одна техническая задача — разработка широкого спектра продуктов, предназначенных для использования во всех видах компьютерного и сетевого оборудования — от персональных коммуникаторов до мощных шлюзов.

Из нормативных проблем отметим необходимость официального признания допустимости использования зарубежных средств и алгоритмов (поскольку это предписывается, например, спецификациями IPsec).

5.7. Контроль целостности

Контроль целостности относится к числу «благополучных» сервисов безопасности, несмотря на его криптографическую природу. Здесь и проблема производительности стоит не так остро, как в случае шифрования, и отечественные стандарты лучше согласуются с международными.

В современных системах контроль целостности должен распространяться не только на отдельные порции данных, аппаратные или программные компоненты. Он обязан охватывать распределенные конфигурации, защищать от несанкционированной модификации потоки данных.

В настоящее время существует достаточно решений для контроля целостности и с системной, и с сетевой направленностью (обычно контроль выполняется прозрачным для приложений образом как часть общей протокольной активности). Стандартизован программный интерфейс к этому сервису (как часть общего интерфейса службы безопасности, см. [9]).

5.8. Контроль защищенности

Контроль защищенности по сути представляет собой попытку «взлома» информационной системы, осуществляемого силами самой организации или уполномоченными лицами. Идея данного сервиса в том, чтобы обнаружить слабости в защите раньше злоумышленников. В первую очередь, имеются в виду не архитектурные (их ликвидировать сложно), а «оперативные» бреши, появившиеся в результате ошибок администрирования или из-за невнимания к обновлению версий программного обеспечения.

Средства контроля защищенности позволяют накапливать и многократно использовать знания об известных атаках. Очевидна их схожесть с антивирусными средствами; формально последние можно считать их подмножеством. Очевиден и реактивный, запаздывающий характер подобного контроля (он не защищает от новых атак). Впрочем, следует помнить, что оборона должна быть эшелонированной, так что в качестве одного из рубежей контроль защищенности вполне адекватен. Отметим также, что подавляющее большинство атак носит рутинный характер; они возможны только потому, что известные слабости годами остаются неустраненными.

Существуют как коммерческие, так и свободно распространяемые продукты для контроля защищенности. Впрочем, в данном случае важно не просто один раз получить и установить их, но и постоянно обновлять базу данных слабостей. Это может оказаться не проще, чем следить за информацией о новых атаках и рекомендуемых способах противодействия.

5.9. Обнаружение отказов и оперативное восстановление

Обнаружение отказов и оперативное восстановление относится к числу сервисов, обеспечивающих высокую доступность (готовность). Его работа опирается на элементы архитектурной безопасности, а именно на существование избыточности в аппаратно-программной конфигурации.

В настоящее время спектр программных и аппаратных средств данного класса можно считать сформировавшимся. На программном уровне соответствующие функции берет на себя программное обеспечение промежуточного слоя. Среди аппаратно-программных продуктов стандартом стали кластерные конфигурации. Восстановление производится действительно оперативно (десятки секунд, в крайнем случае минуты), прозрачным для приложений образом.

Важно отметить, что обнаружение отказов и оперативное восстановление может играть по отношению к другим средствам безопасности инфраструктурную роль, обеспечивая высокую готовность последних. Это особенно важно для межсетевых экранов, средств поддержки виртуальных частных сетей, серверов аутентификации, нормальное функционирование которых критически важно для корпоративной информационной системы в целом. Такие комбинированные продукты получают все более широкое распространение.

5.10. Управление

Управление можно отнести к числу инфраструктурных сервисов, обеспечивающих нормальную работу функционально полезных компонентов и средств безопасности. Сложность современных систем такова, что без правильно организованного управления они постепенно (а иногда и довольно быстро) деградируют как в плане эффективности, так и в плане защищенности.

В контексте данной статьи особенно важной функцией управления является контроль согласованности конфигураций различных компонентов (имеется в виду семантическая согласованность, относящаяся, например, к наборам правил нескольких межсетевых экранов). Процесс администрирования идет постоянно; требуется, однако, чтобы при этом не нарушалась политика безопасности.

Современное управление, на наш взгляд, вступило в переломный этап. Начинают появляться продукты, обладающие достаточной интеллектуальностью, открытостью, расширяемостью, масштабируемостью, продукты, приемлемые по цене и по потребляемым ресурсам. Вероятно, должно пройти еще некоторое время, чтобы они стали достаточно зрелыми, стабилизировались.

5.11. Место сервисов безопасности в архитектуре информационных систем

Мы перечислили десяток сервисов безопасности. Как объединить их для создания эшелонированной обороны, каково их место в общей архитектуре информационных систем?

На внешнем рубеже располагаются средства выявления злоумышленной активности и контроля защищенности. Далее идут межсетевые экраны, защищающие внешние подключения. Они, вместе со средствами поддержки виртуальных частных сетей (обычно объединяемых с межсетевыми экранами) образуют периметр безопасности, отделяющий корпоративную систему от внешнего мира.

Сервис активного аудита должен присутствовать во всех критически важных компонентах и, в частности, в защитных. Это позволит быстро обнаружить атаку, даже если по каким-либо причинам она окажется успешной.

Управление доступом также должно присутствовать на всех сервисах, функционально полезных и инфраструктурных. Доступу должна предшествовать идентификация и аутентификация субъектов.

Криптографические средства целесообразно выносить на специальные шлюзы, где им может быть обеспечено квалифицированное администрирование. Масштабы пользовательской криптографии следует минимизировать.

Наконец, последний рубеж образуют средства пассивного аудита, помогающие оценить последствия нарушения безопасности, найти виновного, выяснить, почему успех атаки стал возможным.

Расположение средств обеспечения высокой доступности определяется критичностью соответствующих сервисов или их компонентов.

6. Возможность практической реализации полного спектра сервисов безопасности в России

Главным препятствием на пути практической реализации полного спектра сервисов безопасности в России является преобладание «точечного» подхода, суть которого сводится к установке отдельных защитных средств без оценки архитектуры в целом и без учета изменений, неизбежных в процессе жизненного цикла информационной системы. К сожалению, большие системы устроены довольно сложно, и «откупиться» от этой сложности, приобретя несколько готовых продуктов, невозможно.

В силу специфики российского законодательства некоторые защитные средства, прежде всего, криптографические, должны разрабатываться уполномоченными организациями, имеющими соответствующие лицензии. На сегодняшний день спектр таких организаций (равно как и произведенных ими продуктов) весьма узок; очевидна тенденция к увеличению отставания от потребностей практики. Вероятно, только объединение усилий Гостехкомиссии России, ФАПСИ, разработчиков и системных интеграторов позволит переломить ситуацию.

Почти все описанные выше сервисы находятся вне сферы действия нормативных документов по сертификации. Опубликованные критерии оценки защищенности есть только у Гостехкомиссии России. Они адекватно описывают межсетевые экраны, с некоторой натяжкой их можно применять к средствам пассивного аудита и разграничения доступа. Все остальное остается белым пятном. Конечно, известны примеры сертификации новых защитных средств (например, средств контроля защищенности), но проводилась она по неадекватным критериям и, следовательно, носит формальный характер, не позволяющий оценить реальные качества анализировавшихся продуктов.

7. Заключение

Обеспечение информационной безопасности современных информационных систем требует комплексного подхода. Оно невозможно без применения широкого спектра защитных средств, объединенных в продуманную архитектуру. Далеко не все эти средства получили достаточное распространение в России, некоторые из них даже «в мировом масштабе» находятся в стадии становления.

В этих условиях позиция по отношению к информационной безопасности должна быть особенно динамичной. Теоретические воззрения, стандарты, сложившиеся порядки необходимо постоянно сверять с требованиями практики. От атак не защититься книгой (даже оранжевой) или сертификатом. Реальная безопасность нуждается в каждодневной работе всех заинтересованных сторон.

8. Литература

1. National Computer Security Center. Trusted Network Interpretation. — NCSC-TG-005, 1987.
2. Security Architecture for Open Systems Interconnection for CCITT Applications. Recommendation X.800. — CCITT, Geneva, 1991.
3. Recommendation X.509. The Directory — Authentication Framework. — Melbourne, 1988.
4. Holbrook P., Reynolds J. (редакторы). Руководство по информационной безопасности предприятия (Site Security Handbook). — Jet Info, 1996, 10/11.
5. Bellovin S. Report of the IAB Security Architecture Workshop. — RFC 2316, 1998. ftp://ftp.isi.edu/in-notes/rfc2316.txt.
6. Галатенко В.А., Макстенек М.И., Трифаленков И.А. Сетевые протоколы нового поколения. — Jet Info, 1998, 7/8.
7. Dierks T., Allen C. The TLS Protocol. Version 1.0. — RFC 2246, 1999. ftp://ftp.isi.edu/in-notes/rfc2246.txt.
8. Ferguson P., Senie D. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. — RFC 2267, 1998. ftp://ftp.isi.edu/in-notes/rfc2267.txt.
9. Галатенко В.А. Обобщенный прикладной программный интерфейс службы безопасности. — Jet Info, 1996, 12/13.

Jet Info
ИНФОРМАЦИОННЫЙ БЮЛЛЕТЕНЬ
Издается с 1995 года
Издатель: компания Джет Инфо Паблишер

Главный редактор: Галатенко В.А. *(galat@jet.msk.su)*
Технический редактор: Антонов А.Н. *(silver@jet.msk.su)*

Россия, 103006, Москва, Краснопролетарская, 6
тел. (095) 972 11 82, 972 13 32
факс (095) 972 07 91
e-mail: JetInfo@jet.msk.su

Подписной индекс по каталогу Роспечати 32555