**Введение**

Internet в последнее время стал популярной, недорогой базовой инфраструктурой. Универсальный доступ к нему заставил многие компании рассмотреть возможность создания виртуальной защищенной сети (Virtual Private Network VPN) на основе глобальной сети Internet (которая, по сути, представляет собой совокупность сетей). Преимущества VPN заключается в использовании базовой инфраструктуры Internet и для коммуникаций внутри компании (включая ее различные отделения) и для связи между компаниями, причем защищенность данных соединений остается такой же, как и в локальных корпоративных сетях.

Корпоративная сеть характеризуется тем, что все ее управление находится в руках ее владельцев, данные, передаваемые по ней, проходят только через узлы входящие в эту сеть и взаимодействие с внешним миром выражается в небольшом объеме передаваемых данных (иногда и полном отсутствие). При таких самодостаточных условиях сеть можно было сделать защищенной. VPN переводит корпоративную сеть на базис Internet. При этом, однако, возникает ряд существенных трудностей. Ни один из узлов не может управлять Internet-ом. Данные из разных источников передаются посредством общей инфраструктуры. Также, благодаря развитию электронной коммерции и бурному росту сетевых технологий, значительно возрос объем данных передаваемых между компаниями. Исходя из всех вышеперечисленных особенностей, можно сделать вывод, что структура VPN существенно отличается от старой традиционной корпоративной сети.

Координационным советом IETF (Internet Engineering Task Force) был разработан и предложен набор протоколов защиты сетевых соединений [3]. IPSec протоколы обеспечивают целостность и конфиденциальность передаваемых данных, а также управление ключевой информацией и политикой защищенных соединений. Отличительной особенностью IPSec от более ранних подобных протоколов заключается в защите всего пути следования передаваемых данных (а не фрагмента, как это было раньше).

Для защиты трафика было предложено два протокола AH (Authentication Header) и ESP (Encapsulating Security Payload). Протокол AH (Authentication Header) заключается в подсчете и проверке значения хеш-функции с ключом от передаваемых данных. Протокол AH обеспечивает целостность и аутентичность данных и защищает от атак, основанных на переповторах пакетов (replay attack). В основе ESP протокола лежит шифрование и расшифрование данных, и он обеспечивает те же функции, что и AH, и дополнительно конфиденциальность передаваемой информации.

Эти две вышеуказанные особенности привели к возникновению проблемы управления ключевой информации и политик (параметров) соединений. Простейшим решением является ручное конфигурирование соединений, при котором параметры и секретные ключи жестко прописываются при запуске системы. К единственному достоинству можно отнести относительную простоту данного метода. Самым же большим недостатком данного метода можно считать отсутствие «масштабируемости». Это означает, что для установления секретного соединения ручным конфигурированием обе стороны должны договориться о ключе, используемом для защиты трафика и о параметрах его использования (выбор протокола, алгоритма и т.п.). Если речь идет о системе, состоящей из небольшого количества пользователей, то никаких проблем не возникает. Когда же количество пользователей составляет несколько тысяч (десятков, сотен…) процедура конфигурирования становится практически невыполнимой задачей. Другим недостатком ручного конфигурирования можно считать отсутствие простого механизма смены используемого ключевого материала. Т.е. ключ шифрование будет использоваться слишком долго (для слишком большого объема данных), что снижает защищенность передаваемых данных.

Другим способом конфигурирования соединений является использование специальных «key management» протоколов. Одним из таких протоколов явился ISAKMP. Данный протокол был также предложен координационным советом IETF. Протокол работает независимо от модуля осуществляющего защиту передаваемых данных. Результатом работы протокола является договоренные между двумя партнерами параметры защищенного соединения (включает в себя набор используемых протоколов защиты данных, алгоритмы, используемые в этих протоколах, параметры алгоритмов) и ключевая информация, для используемых алгоритмов. Полученная информация является выходными данными для протокола и должна быть передана модулю защиты передаваемых данных.

Протокол ISAKMP полностью решает проблему «масштабируемости». Ключевой материал высчитывается на основе данных, передаваемых в процессе аутентификации партнера и договора параметров соединения. При расчете также используется случайные величины, генерящиеся каждой из сторон для данного соединения, что обеспечивает разный ключевой материал при двух попытках установления соединения между одними и теми же партнерами. Данное свойство также позволяет не описывать правило секретного соединения для каждого из абонентов, а объединять их по какому-либо признаку (подсеть, диапазон IP адресов, определенный протокол и т.д.) и описать для них одно правило создания секретного соединения.

Протокол также решает и проблему времени жизни ключевой информации. Время жизни ключевой информации (в секундах и килобайтах) является одним из параметров договариваемого соединения. Таким образом, легко регулируется время использования ключа или объем данных, который можно этим ключом шифровать, и появляется механизм, позволяющий запустить создание нового соединения при истечении времени жизни ключевой информации.

**Анализ методов реализации системы защиты сетевых соединений**

Как только сетевые технологии стали использоваться корпорациями для передачи конфиденциальной информации, возникла проблема защиты этой передаваемой информации.

Самым первым методом защиты сетевых соединений создание локальных корпоративных сетей. Их отличительной особенностью был полный контроль над всеми элементами, входящими в эту сеть, всеми узлами, через которые проходила информация. Локальная корпоративная сеть была самодостаточной и зачастую замкнута в себе. Она или совсем не имела выхода во внешний мир или выходящий трафик тщательно фильтровался. Связанные с этим временные задержки никого не волновали, т. к. этот трафик был весьма не значительным. При таком небольшом, полностью контролируемом оборудовании, корпоративные сети считались достаточно защищенными.

Но сегодняшние корпоративные сети развиваются согласно новым моделям компаний: корпоративная сеть сегодня это набор физически разделенных локальных сетей, соединенных посредством общедоступной сети Internet, а взаимосвязь между компаниями сегодня жизненно необходима. Эта новая модель компании вынесла на общее обозрение передаваемые данные, чего не делала старая корпоративная сеть. Разработчики и пользователи VPN должны осознавать опасность этой открытости и защищаться от нее. Хорошо продуманная политика защиты VPN позволит компании организовать соединения между локальными сетями и между компаниями с такой же гарантией, как и в старых традиционных корпоративных сетях.

Для защиты трафика внутри VPN требовались специальные протоколы передачи данных и протоколы для получения параметров соединения и ключевой информации. На данный момент к протоколам первого типа относятся AH (Authentication Header) и ESP (Encapsulating Security Payload). AH передает данные и некую подпись по этим данным, что обеспечивает их целостность и подлинность. ESP передает зашифрованные данные, что обеспечивает дополнительно конфиденциальность данных.

К протоколам, обеспечивающим AH и ESP необходимыми параметрами и ключевой информацией, относится протокол ISAKMP. Он и будет подробнее рассмотрен далее в этой главе.

Структура протокола ISAKMP

В этом разделе будет рассмотрено, как ISAKMP протокол договаривается о параметрах и обменивается ключами между двумя системами, которые хотят создать секретное соединение [4].

Для того чтобы рассмотреть все на конкретном примере примем, что метод аутентификации – заранее известный секретный ключ (preshared key).

Все пакеты, которыми обмениваются партнеры в процессе установления соединения, начинаются с ISAKMP заголовка. Он содержит некоторую идентифицирующую информацию (Initiator Cookie, Responder Cookie и Message ID), тип обмена, флаги, номер версии и длину всего пакета.

Основное тело пакета состоит из payload-ов. Payload – объем информации, несущий определенную смысловую нагрузку. В дальнейшем этот элемент будем называть «компонентом».

# Фаза 1 (Main Mode)

Целью первой фазы является создание секретного соединения, под защитой которого будут проходить все последующие обмены [5]. Фаза состоит из 6 обменов – 3 со стороны инициатора и 3 со стороны ответчика (Рис. 1).

Рис. 1. Структура фазы 1 (Main Mode)



В пакете 1 инициатор посылает SA payload, компонент, который содержит все предлагаемые варианты параметров соединения. Его структура представлена на рисунке 2.



Рис. 2. Структура SA payload

SA payload содержит внутри себя список Proposal payload-ов, каждый из которых представляет собой отдельный протокол. Proposal payload-ы могут объединяться в группы по «И» и по «ИЛИ». Это осуществляется с помощью номеров данных компонент – одинаковые номера означают объединение по «И», а разные по ИЛИ. В свою очередь Proposal payload содержит список Transform payload-ов, которые представляют алгоритмы для данного протокола. Объединены они могут быть только по «ИЛИ». Transform payload содержит список атрибутов, конкретизирующих данный алгоритм (длина ключа) и содержащих другие параметры соединения. Атрибуты не могут выбираться, или принимается весь список атрибутов или все отвергает.

Таким образом, инициатор посылает ответчику на выбор список списков протоколов и для каждого протокола на выбор список алгоритмов. Из всего этого ответчик выбирает список протоколов, причем для каждого протокола может быть выбран **только один** алгоритм и набор атрибутов для данного алгоритма не может изменяться (ни добавления / удаления, ни изменения), т.е. алгоритм или принимается со всем списком атрибутов, или отвергается. Выбранная информация оформляется также в SA payload, и отправляется инициатору вторым пакетом.

Итогом первых двух пакетов, или первого обмена, становиться договоренность относительно параметров соединения.

В пакетах 3 и 4 передаются KE payload и Nonce payload. В КЕ payload инициатор и ответчик обмениваются своими открытыми ключами для алгоритма Diffie-Hellman. Они потребуются на последующих этапах для расчета общего ключа. Nonce payload содержит случайную последовательность любого размера, которые также будут участвовать при расчете ключевой информации.

После этого обмена можно начать расчет ключевой информации. На основе чужого открытого и своего секретного ключей рассчитывается общий ключ (g^xy) по алгоритму Diffie-Hellman. Затем производится расчет некоторых служебных констант.

SKEYID = PRF (Preshared Key, Ni | Nr)

где Preshared Key – заранее известный секретный ключ.

SKEYID\_d = PRF (SKEYID, g^xy | CKY-I | CKY-R | 0)

SKEYID\_a = PRF (SKEYID, SKEYID\_d | g^xy | CookieI | CookieR | 1)

SKEYID\_e = PRF (SKEYID, SKEYID\_a | g^xy | CookieI | CookieR | 2)

Из формул видно, что в расчете всех констант (а, следовательно, и во всех последующих расчетах) участвует известный только обменивающимся сторонам секретный ключ (Preshared Key), что обеспечивает аутентификацию сторон, т. к. никто другой не сможет правильно рас читать эти константы.

Из SKEYID\_e мы получаем ключевую информацию. Остальные константы будут использованы при дальнейших расчетах.

В пакетах 5 и 6 партнеры обмениваются информацией, которая их идентифицирует (IDii и IDir) и информацией, которая их аутентифицирует (HASH\_I, HASH\_R). Идентификационная информация передается посредством Identification payload, где указывается тип идентификационной информации (IP адрес, имя пользователя, SubNet и т.п.) и собственно значение.

Аутентификационная информация передается через Hash payload. Его содержимое рассчитывается по следующим формулам (для инициатора и ответчика соответственно):

HASH\_I = PRF (SKEYID, g^xi | g^xr | CookieI | CookieR | SAi | IDii)

HASH\_R = PRF (SKEYID, g^xr | g^xi | CookieR | CookieI | SAi | IDir)

Последний обмен (пакет 5 и 6) уже передается защищенным с помощью договоренных на первом этапе алгоритмов и рассчитанной после второго пакета ключевой информацией.

# Фаза 1 (Aggressive Mode)

Aggressive Mode выполняет те же функции, что и Main Mode, но укладывается всего в три пакета [5]. Такое упрощение, однако, приводит к тому, что он более подвержен атакам, чем Main Mode. На рисунке 3 представлена структура Aggressive Mode.

В пакете 1 инициатор посылает сразу SA payload с предложением параметров соединения, KE payload со своим открытым ключом, Nonce payload со случайной информацией и идентифицирует себя с помощью Identification payload.

Сразу видны недостатки данного режима. В SA payload-е не может быть предложено более одной группы параметров для алгоритма Diffie-Hellman-а т. к. сразу же посылается открытый ключ, а его размер напрямую зависит от этих параметров. В данном режиме, в отличие от Main Mode, идентификационная информация посылается в открытом виде.

Ответчик, получив пакет 1, уже имеет достаточно информации для расчета рабочих констант и своей аутентификационной информации. Поэтому в пакет 2 состоит из тех же частей, что и пакет 1 (с соответствующим наполнением) и добавляется Hash payload, содержащий информацию, аутентифицирующую ответчика. Пакет еще не может быть зашифрован (т. к. инициатор не знает выбранного алгоритма и у него нет ключей), но можно уже провести ключевой информации, которая будет использована в будущем.

Рис. 3. Структура фазы 1 (Aggressive Mode)

Инициатор из пакета 2 берет необходимую информацию. Затем вычисляет рабочие константы, аутентификационную информацию и ключи шифрования. Пакетом 3 инициатор аутентифицирует себя.

# Фаза 2 (Quick Mode)

Целью второй фазы является получение параметров секретного соединения и ключевой информации [5] [6]. Все пакеты, передаваемые во время второй фазы, защищаются секретным соединением, созданным во время первой фазы. Одновременно с обеспечением конфиденциальности передаваемой информации обеспечивается и целостность данных путем передачи значения хеш-функции от данных.

Рис. 4. Структура фазы 2 (Quick Mode)

Режим состоит из трех пакетов. Его структура представлена на рисунке 4. В первом пакете инициатор посылает SA payload, содержащий предложения о параметрах будущего соединения, случайную информацию (Nonce payload) для создания ключевой информации. Все остальные компоненты пакета являются опциональными. Если для расчета ключевой информации требуется использовать «свежий» ключевой материал, то осуществляется еще один обмен открытыми ключами, в противном случае для расчета берется информация из первой фазы. Также, если локальная политика требует использование во второй фазе идентификационной информации отличной от информации используемой в первой фазе, добавляются соответствующие Identification payload-ы.

Структура второго пакета аналогична первому, только заполняется информацией об ответчике. Исключение составляют только компоненты с идентификационной информацией, которые или принимаются (и тогда в таком же виде и отсылаются) или не принимаются и попытка установления соединения считается неудачной.

Третий пакет посылается инициатором в подтверждение правильности принятой информации и содержит только Hash payload, который вычисляется с помощью буфера случайных данных, посланных ответчиком во втором пакете. Содержимое Hash payload-ов вычисляются по следующим формулам:

HASH(1) = PRF (SKEYID\_a, Message ID | SA | Ni [| KE] [| IDic | IDcr])

HASH(2) = PRF (SKEYID\_a, Message ID | Ni | SA | Nr | [| KE] [| IDic | IDcr])

HASH(3) = PRF (SKEYID\_a, 0 | Message ID | Ni | Nr)

Формула для расчета окончательного ключевого материала зависит от того, был ли обмен открытыми ключами для создания нового общего ключа. Если такого обмена не было, то формула следующая:

KEYMAT = PRF (SKEYID\_d, protocol | SPI | Ni | Nr)

где protocol – номер протокола, для алгоритма которого считается ключевой материал.

Если все же вычисление общего ключа производилось, формула для расчета окончательного ключевого материала следующая:

KEYMAT = PRF (SKEYID\_d, g^xy | protocol | SPI | Ni | Nr)

Таким образом, после второй фазы мы получаем всю необходимую информацию для создания секретного соединения. Список применяемых протоколов и используемых в них алгоритмы получается после обмена SA payload-ами во второй фазе. Ключевая информация для каждого алгоритма рассчитывается по приведенным выше формулам. Следует заметить, что приведенная выше структура протокола была упрощена для простоты восприятия (отсутствует рассмотрение остальных методов аутентификации и New Group Mode).

Виды сетевых атак

Не смотря на то, что протокол сам по себе не производит защиту передаваемой информации, а лишь создает соединения для передачи данных, он сам является предметом атаки. Подвергнуться атаке в протоколе могут процесс аутентификации, процесс обеспечения целостности и конфиденциальности передаваемой информации и, наконец, сама работоспособность протокола. В этом разделе мы рассмотрим основные виды сетевых атак и то, как протокол им противостоят [4].

# Отказ в обслуживании (Denial of Service)

Данная атака является одной из самых простых и эффективных. Целью атаки является работоспособность системы или, в данном случае, протокола.

Сама атака представляет собой посылку злоумышленником большого числа запросов на создание соединения, вынуждая противоположную сторону тратить ресурсы на их обработку. Чтобы скрыть свой истинный адрес пакеты могут посылаться с фиктивных адресов. Если посылаемые ложные запросы занимают собой все ресурсы системы, то обработка приходящих правильных запросов откладывается на неопределенное время или они просто игнорируются. Со стороны внешнего мира система выглядит неработающей.

Оговоримся сразу, способа, полностью защитится от данного типа атак не существует. Атаку можно лишь сделать менее эффективной. В протоколе ISAKMP это достигается в первую очередь за счет откладывания основных «тяжелых» расчетов на более поздние обмены. В первые обмены производятся простые вычисления (выбор параметров соединения). В то же время сама работа протокола состоит из нескольких обменов, что не позволяет злоумышленнику использовать фиктивные адреса, т. к. не получив информацию от нас, он не сможет правильно сформировать следующий пакет. Т.е. если атака и станет успешной, мы будем точно знать, кто нас атаковал. Однако следует заметить, что данный способ защиты не подходит для Aggressive Mode, который, как уже подчеркивалось, работает быстрее, но менее защищен.

# Человек посередине (Man-in-the-Middle)

Целью атаки являются конфиденциальность и целостность данных. Атака заключается в том, что злоумышленник, вклиниваясь в процесс установления секретного соединения, представляется для каждой из сторон ее партнером и проводит установление соединения от ее имени. В результате вместо одного защищенного канала между двумя партнерами получается два канала между каждой из сторон и злоумышленником. Для каждого из партнеров все выглядит обычным образом, но злоумышленник получает возможность не только просматривать данные, передаваемые по «защищенному» каналу, но даже модифицировать их. Структура описанной атаки представлена на рисунке 5.

Sx – секретный ключ, Px – открытый ключ

Защита от данного вида атаки в протоколе ISAKMP заключается в процессе аутентификации. Обязательное выполнение этого процесса во время первой фазы гарантирует обеим сторонам отсутствие «человека посередине», который смог бы прослушивать и модифицировать передаваемые данные не только во второй фазе, но и при передаче основной информации. В данном случае стойкость протокола к данному типу атаки определяется надежностью метода аутентификации. Для метода заранее известного секретного ключа это определяется уникальностью данного ключа, для методов, использующих сертификаты – достоверностью полученного сертификата.

# Повтор посылки (Replay attack)

Атака заключается в перепосылке ранее записанных пакетов в расчете на неправильную реакцию атакуемого. Например, попытаться с помощью пакетов, подслушанных при аутентификации двух партнеров, представиться одним из них при установлении соединения со вторым. Даже если таким образом просто повторят уже проведенное соединение (т.е. в результате будет создано еще одно соединение совпадающие с прежним), это приведет к потере ресурсов.

Для защиты от этой атаки в протоколе был введен Nonce payload, с помощью которого стороны обмениваются случайной информацией. Эта информация потом участвует в расчетах всех констант и ключевых материалов. Использование в каждом обмене «свежей» случайной информации гарантирует защиту от атак с помощью переповторов.

В первой части данного раздела была рассмотрена структура протокола создания защищенных сетевых соединений ISAKMP. В процессе рассмотрения были приведены порядок посылки пакетов, их содержимое и объяснено назначение каждого компонента пакета. Также были даны формулы, по которым проводятся расчеты внутренних констант и окончательного ключевого материала.

Во второй части были представлены основные типы сетевых атак, объяснен принцип их действия и, на основе структуры протокола ISAKMP, показано как он противостоит этим атакам.

Разработка программы

Определение места программы в системе защиты сетевого трафика

В этом разделе мы рассмотрим, из каких основных модулей состоит система защиты сетевого трафика, назначение этих модулей и каким образом они взаимодействуют [3].

На рисунке 6 представлена структура системы защиты сетевого трафика. Рассмотрим отдельно каждый модуль.

**Модуль управления**

Данный модуль определяет общее поведение системы. Внутри него происходит считывание, проверка и хранение конфигурационной информации, согласно которой он управляет остальными модулями. Модуль имеет интерфейсы почти ко всем остальным модулям. В модуль обработки трафика он прогружает правила фильтрации трафика (входящего и исходящего), правила обработки трафика (заданные вручную в конфигурации и полученные модулем ISAKMP). Из модуля обработки трафика он получает запросы, на создание секретного соединения (правила обработки трафика), которые передает в модуль ISAKMP. Также в процессе работы модуля ISAKMP именно на нем лежит обязанность формулирования предлагаемых вариантов параметров соединения и выбор приемлемого варианта в предложенном наборе. До прогрузки секретного соединения, созданного модулем ISAKMP, оно сохраняется в модуле хранения основной ключевой информации.

# Модуль хранения основной ключевой информации

Является дублирующим местом хранения правил обработки трафика (еще одно находится в модуле обработки сетевого трафика). Необходимость дублирования информации в двух местах объясняется тем, что время жизни соединения в секундах легче отслеживать в этом модуле, а в килобайтах в модуле обработки трафика. Дополнительно появляется возможность не хранить в модуле обработки трафика информации о соединениях, которыми давно не пользовались, а запрашивать эту информацию по необходимости. Взаимодействие ведется только с модулем управления, от которого принимается информация о соединения для сохранения и команды на удаление соединения, а выдается сигнал о том, что у какого-то соединения кончилось время жизни.

# Модуль обработки сетевого трафика

Обрабатывает входящий и исходящий трафики согласно правилам фильтрации и правилам обработки трафика, которые прогружает модуль управления. Для противостояния атакам отказа в доступе, если для входящего пакета не находится правила его обработки, то он просто игнорируется. Наоборот, если подобная ситуация произойдет для исходящего пакета, то в модуль управления передастся запрос на создания такого соединения. Также в модуль управления может поступить сигнал о том, что у какого-либо соединения истекло время жизни в килобайтах. От модуля управления данный модуль может получить созданное соединение и сигнал на уничтожение соединения.

# Модуль ISAKMP

По запросу со стороны модуля управления и используя информацию из конфигурации, создает правила обработки трафика. Взаимодействует с модулем управления и модулем хранения ключевой информации ISAKMP. В модуль хранения ключевой информации сохраняются внутренние соединения, созданные во время первой фазы и использующиеся для защиты последующих фаз. От модуля управления получает запрос на создание соединения, информацию из конфигурации для формирования / выбора параметров соединения и формирования / проверки идентифицирующей информации и информацию для аутентификации себя. Обратно в модуль управления отдается созданное соединение или сигнал о неудачной попытке его создания.

# Модуль хранения ключевой информации ISAKMP

Данный модуль является хранилищем информации о секретных соединениях протокола ISAKMP, используемых им для защиты своего трафика. Данные соединения полностью скрыты для модуля управления. Модуль осуществляет прием на хранение информации о соединениях, поиск существующего соединения и отслеживание окончания времен жизни хранимых соединений (при истечении срока соединение тут же удаляется).

На основе представленной структуры можно описать, каким образом программа, реализующая протокол ISAKMP, вписывается в систему защиты сетевого трафика. Программа объединяет собой модуль ISAKMP, модуль хранения ключевой информации ISAKMP и часть модуля хранящую конфигурационную информацию необходимую для работы модуля ISAKMP, осуществляющая запросы на создание нового соединения и приема созданных правил обработки трафика. Таким образом, получается только один интерфейс со всей остальной системой, описывающий взаимоотношения между частью модуля управления вошедшей в состав программы и оставшейся частью модуля управления.

Разработка общей структуры программы

Так как представленная программа написана с использованием технологии «нитей» (Thread), то в начале данного раздела будет дано определение этому термину, описаны плюсы и минусы использования этой технологии, а затем рассмотрено из каких конкретно модулей (нитей) состоит программа, их назначение и взаимодействие между собой.

# Что такое нить (thread)?

Под нитью (иногда называемой нитью контроля) понимается независимая последовательность выполнения программного кода внутри отдельного процесса [10]. Нити разделяют между собой всю память процесса, и если одна нить пишет что-то в память, другая может читать эти данные. Нити также разделяют все остальные ресурсы процесса, например, дескриптор файла, т.е. сразу несколько нитей могут писать в один и тот же файл. Нити внутри процесса распределяются и исполняются абсолютно независимо, т.е. если одна нить ожидает ввода информации, это никаким образом не прерывает исполнение других нитей. В мультипроцессорных системах разные нити могут выполняться разными процессорами. В однопроцессорных же системах – нити могут исполняться в произвольном порядке. Обычно, нить исполняется, пока не будет заблокирована каким-либо запросом или пока не закончится отведенный ей отрезок времени (квант времени).

Не смотря на то, что использование нитей несколько усложняет процесс программирования, они дают преимущества. Рассмотрим эти преимущества подробнее.

**Производительность.** Программа, реализованная с помощью только одной нити, переходит в режим ожидания при каждом системном вызове. Использование более одной нити (и для мультипроцессорных, и для однопроцессорных систем) позволяет совместить времена ожидания выполнения системных вызовов. Нить, которая делает запрос, переходит в режим ожидания, но другая нить в данном процессе может продолжать работу. При этом одному процессу в каждый момент времени может соответствовать несколько запросов к системе. Следует заметить, что данные запросы остаются синхронными.

**Мультипроцессорные системы.** Использование нескольких нитей в одном процессе является эффективным способом использования возможности параллельной работы.

**Графический интерфейс для пользователя.** Однонитевое приложение, предоставляющее пользователю графический интерфейс, обычно замирает при ожидании реакции от пользователя (например, нажатие кнопки). Если бы это приложение было много нитевым, то с ожиданием нажатия кнопки можно было бы связать отдельную нить, а другие нити продолжали бы работать. Так же в подобных системах множество нитей позволило бы сделать незаметным для пользователя выполнение служебных действий (например, автоматическое сохранение).

**Оперативность серверных приложений.** Серверные приложения обрабатывают запросы, приходящие от клиентов. Одновременно может прийти несколько запросов. В случае однонитевого приложения запросы будут выполняться последовательно, и выполнение сложного запроса может надолго отложить выполнение других, более простых и важных запросов. Много нитевая структура в этом отношении представляется более адаптивной, т. к. каждый запрос пользователя может быть обработан согласно его сложности и важности. Другой проблемой для серверных приложений является взаимные запросы. Это происходит если сервер 1, обрабатывая клиентский запрос, делает запрос к серверу 2, который в свою очередь при его обработке обращается обратно к серверу 1. В однонитевом приложении это приведет к зависанию обоих серверов, т. к. единственная нить сервера 1 уже занята обработкой запроса и не может обработать запрос сервера 2. Использование нескольких нитей решает эту проблему, т. к. для каждого запроса выделяется отдельная нить, которая выполняется независимо от других.

Однако использование нитей несет в себе несколько опасностей, и главная из них это работа с общей памятью. Рассмотрим конкретный пример. Оператор увеличения переменной на единицу для программы выливается в 3 действия:

1. Загрузить значение переменной в регистр
2. Увеличить регистр
3. Записать значение регистра в переменную

Если две нити начнут выполнять этот оператор одновременно, то может произойти следующая последовательность действий:

**Нить 1 Нить 2**

1 Загрузить значение переменной в регистр

1 Загрузить значение переменной в регистр

2 Увеличить регистр

3 Записать значение регистра в переменную

2 Увеличить регистр

3 Записать значение регистра в переменную

В этом случае нить 1 перезапишет значение записанной нитью 2, и переменная увеличится лишь на единицу вместо двух. Такие места в коде называют критическими секциями и организуют работу так, что они гарантировано выполняются только одной нитью.

Использование многонитевого принципа построения моей программы вызвано двумя причинами:

* Необходимость постоянно прослушивать требуемый порт на наличие пришедшего пакета.
* Принцип действия программы похож на принцип работы серверного приложения (в качестве запросов клиентов выступают приходящие пакеты). В связи с этим становиться очень ценным возможность обработки пакетов согласно их важности.

# Механизм обмена информации между нитями

В процессе работы программы нитями необходимо обмениваться информацией. В основном это передача пакетов и запросов с параметрами. Для осуществления обмена использовался механизм pipe [8]. Pipe представляет собой модуль для передачи данных. Единственным его ограничением является то, что этот модуль создает однонаправленный поток данных. Создание производится с помощью функции *pipe*.

#include <unistd.h>

int pipe (int filedes[2]);

Функция возвращает 0 в случае успеха и –1 при ошибке. Параметрами в функцию предается массив из двух дескрипторов, которые заполняются внутри функции. В первый дескриптор (filedes[0]) предназначен для чтения из pipe, а второй (filedes[1]) для записи в pipe. Чтение из pipe и запись в него производятся с помощью стандартных функций *read* и *write*. Для этих функций дескриптор pipe ничем не отличается от дескриптора файла.

#include <unistd.h>

ssize\_t read (int fildes, void \*buf, size\_t nbyte);

ssize\_t read (int fildes, void \*buf, size\_t nbyte);

Вторым параметром в функции передается указатель на буфер, куда записывать данные (для *read*) или откуда считывать их (для *write*). Третьим параметром для *read* передается максимальное число читаемых данных, я для *write* число записываемых байт.

Выбор pipe в качестве средства передачи между нитями обусловлен простотой и наглядностью данного метода. Плюс корректно обрабатывают ситуацию, когда в pipe пишут сразу 2 сообщения – эти сообщения не перемешиваются, а записываются последовательно. Правда в этом случае встает задача разделения этих двух сообщений, т. к. *read* не разделяет их, а, прочитав сразу два можно или не заметить второе сообщение или посчитать неверной структуру первого. Для избежания этого формат сообщения, передаваемого в pipe в моей программе следующий.



Рис. 8. Структура передаваемых запросов

Первым байтом в сообщении идет тип данного сообщения, который говорит, какие именно данные содержаться. Если тип сообщения не предусмотрен в месте, куда это сообщение пришло, сообщается об ошибке и из pipe считывается буфер максимальной длины. Следующие за типом сообщения 4 байта содержат длину передаваемых данных. Если тип сообщения налагает какие-либо ограничения на длину данных (например, если передается IP адрес, то его длина должна быть 4 байта) и считанная длина этим ограничениям не удовлетворяет, то также сообщается об ошибке, и стараемся все вычитать из pipe. После этого из pipe достаются данные указанной длины. После обработки считанного запроса процедура повторяется заново, начиная с получения типа сообщения. Такой формат сообщения и приведенный порядок его обработки гарантирует, что никакое сообщение не будет потеряно.

# Нитевая структура программы

В этом разделе будет рассмотрено, из каких нитей состоит программа, их назначение и как они взаимодействуют друг с другом. На рисунке 9 представлена нитевая структура программа.

Рис. 9. Нитевая структура программы



На рисунке окружностями условно показаны нити, одинарными стрелками передача данными между нитями, а двойными взаимодействие с таблицей (добавление, поиск и удаление). Программа содержит 4 вида нитей:

1. Нить работы с сетью
2. Нить распределения пакетов
3. Нить выполнения первой фазы
4. Нить выполнения второй фазы

**Нить работы с сетью.** Задачей данной нити является непрерывная проверка порта на наличие пакета и прием запросов от других модулей на отсылку пакетов. Работа данной нити начинается с открытия порта (функция *socket*) и указания адреса и порта, с которым мы будем работать[9].

struct sockaddr\_in serveraddr;

if ((sockdscr = socket (AF\_INET, SOCK\_DGRAM, 0)) == -1) {

printf («Server error: cannot open socket\n»);

return NULL;

}

memset (&serveraddr, 0, sizeof(serveraddr));

serveraddr.sin\_family = AF\_INET;

serveraddr.sin\_port = htons (Conf. LocalPort);

serveraddr.sin\_addr.s\_addr = inet\_addr (Conf. LocalAddress); if (bind(sockdscr, (struct sockaddr \*)&serveraddr, sizeof(serveraddr))==-1) {

printf («Server error: cannot bind\n»);

return NULL;

}

Как видно из данного части исходного кода программы, локальный IP адрес и номер порт берутся из конфигурации. После инициализации нить должна войти в режим ожидания и реагировать только на два события приход пакета и получение запроса на отправку пакета. Данное действие выполняется с помощью функции *select*. Она предназначена для слежения за несколькими дескрипторами одновременно на предмет их готовности к чтению, записи или если произошла ошибка.

#include <sys/time.h>

int select (int nfds, fd\_set \*readfds, fd\_set \*writefds,

fd\_set \*errorfds, struct timeval \*timeout);

Первый параметр в этой функции – максимальный номер рассматриваемых дескрипторов. Три следующих параметра – массивы дескрипторов. Первый содержит номера дескрипторов, которые нужно наблюдать на предмет возможности чтения из него, второй на предмет возможности записи в него и третий на предмет возникновения в них ошибок. Последний параметр задает временной интервал, через который функция должна закончиться, если не произойдет никакого события. Данный параметр позволяет использовать данную функцию как таймер. В случае срабатывания какого-либо события возвращается число сработавших дескрипторов, и в массивах остаются только эти сработавшие дескрипторы. При выходе по таймауту функция возвращает ноль. В случае ошибки возвращается –1.

Вся нить, таким образом, представляет собой выполнение функции *select*, которая проверяет на возможность чтения из дескриптора порта и читающего конца pipe, передающего данные в эту нить. Т.к. выход по таймауту в данном случае нам не нужен, пятым параметром передается NULL.

FD\_SET (sockdscr, &rfds); /\* Добавление в массив дескриптора порта\*/

FD\_SET (pipefd[0],&rfds); /\* Добавление в массив дескриптора pipe\*/

retval=select (1024,&rfds, NULL, NULL, NULL);

if (SOCKET\_ERROR == retval) {

/\* Обработка ошибки\*/

}

if (FD\_ISSET (sockdscr, &rfds)) {

/\* Действия, выполняемые при приходе пакета \*/

}

if (FD\_ISSET (pipefd[0], &rfds)) {

/\* Действия, выполняемые при получении запроса \*/

}

В случае прихода пакета он целиком передается нити распределения пакетов (вместе с ним передаются также IP адрес и номер порта отправителя). При получении запроса на посылку пакета пакет отсылается, причем адрес и номер порта получателя должен находиться в запросе.

**Нить распределения пакетов.** В задачу данной нити входит предварительный разбор заголовка пакета, проверка правильности структуры пакета и передача пакета нити, для которой он предназначен. Вся информация для проверки пакета и нахождения нити приемника берется из ISAKMP заголовка пакета. Данный заголовок должен находиться в начале каждого пакета и служит для определения, к какой именно попытке установления соединения принадлежит данный пакет. Структура ISAKMP заголовка приведена на рисунке 10 [4].

Первые 8 байт занимает Initiator Cookie – иидентификатор попытки установления соединения со стороны инициатора. Значение данного поля выбирается на стороне инициатора (случайным или предопределенным образом) и служит при дальнейшем распределении пакетов. Responder Cookie играет такое же значение, но для ответчика.



Рис. 10. Структура ISAKMP заголовка

Следующим полем идет Next Payload, которое показывает тип компонента (payload) следующего за заголовком. Version показывает версию используемого протокола. Exchange type говорит о режиме, при котором используется данный пакет (Main Mode, Aggressive Mode, Quick Mode и т.п.). Флаги содержат информацию о состоянии пакета, например, зашифрован он или нет. Еще одним идентификатором пакета является Message ID. Последние 4 байта содержат длину всего пакета, включая сам заголовок.

Идентификация пакета проводится по следующим принципам. В первом пакете инициатор проставляет Initiator Cookie, а Responder Cookie оставляет нулевым, давая возможность ответчику при ответе заполнить его. Message ID служит для идентификации разных попыток установления соединения во второй фазе, идущих под защитой одной и той же первой фазы, а, следовательно, имеющих одинаковые CookieI и CookieR.

Порядок обработки пакета следующий

1. Проверка длины пакета. Производится простым сравнением длины полученного пакета, которую мы узнаем при чтении пакета из порта и значением соответствующего поля в ISAKMP заголовке. Данная проверка является очень простой, но в то же время весьма эффективной, т. к. позволяет быстро (фактически на самом первом этапе), без затрачивания больших ресурсов отсечь случайные пакеты. Здесь мы впервые встречаемся с проблемой разного способа хранения чисел на разных архитектурах. Если рассмотреть конкретный пример, то число 0x01020304 в системе с процессором Sun Sparc будет представлено в виде



т.е. сначала идут старшие цифры (так называемое big-endian представление), а для процессора Intel



сначала идут младшие цифры (little-endian). Из-за требования поддерживать обе платформы использовать простое проведение памяти к типу *unsigned int* нельзя, т. к. значение длины пакета, например, 100 для Sun Sparc будет 100, а для Intel 1677721600. Для решения этой проблемы были написаны макросы для перевода чисел из одного состояния в другое для обеих платформ.

#define GET\_32BIT(cp) \

(((unsigned long) (unsigned char) (cp) [3]) | \

((unsigned long) (unsigned char) (cp) [2] << 8) | \

((unsigned long) (unsigned char) (cp) [1] << 16) | \

((unsigned long) (unsigned char) (cp) [0] << 24))

#define GET\_16BIT(cp) \

(((unsigned long) (unsigned char) (cp) [1]) | \

((unsigned long) (unsigned char) (cp) [0] << 8))

#define PUT\_32BIT (cp, value)\

(cp) [3] = (value); \

(cp) [2] = (value) >> 8; \

(cp) [1] = (value) >> 16; \

(cp) [0] = (value) >> 24;

#define PUT\_16BIT (cp, value)\

(cp) [1] = (value); \

(cp) [0] = (value) >> 8;

Если проверка не прошла, то дальнейшее рассмотрение пакета заканчивается и пакет удаляется.

1. Проверяем допустимость значений некоторых других полей заголовка – Next Payload, Version, Exchange Type и Flags. Эти поля проверяются не на точное совпадение, как длина пакета, а на вхождение значения в диапазон допустимых значений. Проверка корректности данных значений будет произведена позже, при детальном рассмотрении структуры пакета
2. Поиск в таблице нитей первой фазы возможного получателя пакета. Поиск ведется на основе значений CookieI и CookieR. Т.к. некоторые пакеты могут представлять собой запросы на создание соединений с другой стороны (т.е. нить для их обработки еще не создана), переповторы этих запросов (нить уже создана и уже проставлено значение CookieR, т.е. в данную нить пакет не попадет), а также ответы на наши запросы (CookieR в таблице стоят нулевые), то порядок поиска подходящей записи в таблице следующий: «если значение CookieR в пакете или таблице нулевое, то запись считается сработавшей, если совпало только значение CookieI, иначе должны совпасть значение и CookieR, и CookieI». Если мы нашли сработавшую запись, то мы получим дескриптор записи для pipe, связывающей с нужной нитью, по которому и передадим пакет. Если запись не найдена и значение CookieR равно нулю это означает, что это первый пакет новой попытки установления соединения. Для данного пакета мы создаем новую нить, pipe для связи с этой нитью, делаем добавление записи в таблицу нитей первой фазы, после чего передаем пакет только что созданной нити. Если не выполнилось ни одного из вышеперечисленных условий, то пакет считается неверным и удаляется.

Таким образом, происходит обработка пришедшего пакета, но нить распределения пакетов может также получить запрос на создание секретного соединения в качестве инициатора. В этом случае создается нить, pipe для связи с ней и нити передается пустой пакет, как знак начала работы в качестве инициатора.

**Нить выполнения первой фазы.** Данная нить предназначена для проведения первой фазы установления соединения.

Как было указано выше нити можно представить как независимое выполнение программы. Но, не смотря на это, каждая нить имеет свой собственный стек, а значит все переменные, объявленные в функции принадлежат только нити. Этот факт используется для хранения информации, связанной только с данной попыткой установления соединения (ключи шифрования, рабочие константы, метод аутентификации, текущие CookieI и CookieR и т.д.).

Обработка пришедшего пакета начинается с более тщательной проверки пакета. Сначала проверяется то, что не изменился тип обмена (если это первый пакет, то данное значение сохраняется для последующих проверок). То же самое делается со значением версии в ISAKMP заголовке. После этого происходит расшифрование пакета (если это требуется). Дальше происходит разбор пакета, выполнение промежуточных действий (расчетов), составление и отсылка ответного пакета. Так как набор функций, реализующих перечисленные выше действия, отличается для каждого пакета, построением простого цикла задача не решается. Для решения этой проблемы была введена переменная, которая отражала текущие состояние обмена, и по значению которой можно было узнать какие функции выполнять. Для наглядности и простоты работы с этой переменной были описаны ряд *define*.

#define INITIATOR 0x0

#define RESPONDER 0x80

#define MAIN 0x0

#define AGGRESSIVE 0x40

#define ABSENT 0x0 /\* Метод аутентификации еще не определен\*/

#define PRESHARED 0x8

#define DSA\_SIGN 0x10

#define RSA\_SIGN 0x18

#define RSA\_ENC 0x20

#define RSA\_REV\_ENC 0x28

#define RENCRYPT 0x30

#define GET\_ROLE(State) (State&0x80)

#define GET\_EXCH(State) (State&0x40)

#define GET\_MODE(State) (State&0x38)

#define GET\_STEP(State) (State&0x03)

#define SET\_ROLE (State, Role) {State &= 0x80; State+=Role;}

#define SET\_MODE (State, Meth) {State &= 0xC7; State+=Meth;}

#define SET\_EXCH (State, Meth) {State &= 0xBF; State+=Meth;}

#define STATE (role, exch\_type, mode, step) (role+exch\_type+mode+step)

Пример использования данной переменной при работе с программой будет приведен далее. Переход к требуемому набору функций осуществляется с помощью оператора выбора *switch*. В конце каждого набора функций переменная текущего состояния должна переводиться в следующее состояние (чаще всего это просто увеличение номера пакета) или должен выставляться флаг, говорящий об окончании первой фазы.

/\*Получение о предварительная обработка пакета\*/

switch (state. State) /\* Выбор набора требуемых функций \*/

{

case STATE (RESPONDER, AGGRESSIVE, ABSENT, 0):

/\* Набор функций для данного состояния \*/

break;

case STATE (RESPONDER, MAIN, ABSENT, 0):

/\* Набор функций для данного состояния \*/

break;

………………………

case STATE (INITIATOR, MAIN, DSA\_SIGN, 2):

/\* Набор функций для данного состояния \*/

break;

}

Переменная текущего состояния также активно используется при расчетах, где формулы отличаются в зависимости от метода аутентификации или исполняемой роли (инициатор или ответчик). Отсылка созданного пакета осуществляется через доступный каждой нити дескриптор записи pipe нити работы с сетью

После окончания первой фазы нить переходит в режим управления нитями второй фазы. Пакеты для этих нитей по значению CookieI и CookieR приходят в данную нить, а затем согласно значению Message ID отправляются в нити второй фазы или инициируют их создание. Для проведения правильной идентификации пакетов каждая нить первой фазы содержит свою таблицу нитей второй фазы, по которой и проводит поиск.

**Нить выполнения второй фазы.** Задача данной нити – проведение второй фазы установления соединения. Структура и принцип работы полностью такие же, как и в нити первой фазы. При создании вместе с пакетом нити передаются также значения некоторых переменных (рабочие константы, ключи шифрования и т.п.) необходимые для нормальной работоспособности нити. По завершению второй фазы нить выдает полученные результаты, удаляет себя из таблицы нитей второй фазы и заканчивает работу.

# Таблицы поиска нитей

Таблица нитей выполняющих первую фазу представляет собой список *CookieTable\_t* структур

struct CookieTable\_t;

typedef struct CookieTable\_t {

uchar CookieI[8]; /\*Initiator Cookie \*/

uchar CookieR[8]; /\* Responder Cookie \*/

int pd; /\* Pipe Descriptor \*/

uchar Ready; /\* Ready Flag \*/

struct in\_addr AlienAddr; /\* Peer IP Address \*/

struct CookieTable\_t \*next; /\* Next Member (NULL if last) \*/

};

Необъясненными в данной структуре остались два поля: флаг Ready и структура IP адреса. В структуре IP адреса находится адрес партнера, с которым мы ведем процесс установления соединения. Флаг Ready показывает, закончилось или нет проведение первой фазы. Он используется, если со стороны модуля управления пришло 2 запроса на инициацию попытки установления соединения. В этом случае просматривается таблица нитей первой фазы в поисках записи с указанным IP адресом. Если флаг Ready в данной записи говорит о том, что первая фаза уже завершена, то запрос формируется на проведение сразу второй фазы. IP адрес может также использоваться при поиске нити для пришедшего пакета.

Ниже приведен пример функции поиска записи в таблице по совпадению и CookieI и CookieR.

CookieTable\_t \*FindCookieRecord (uchar \*CI, uchar \*CR) {

uchar Test[8] = {0, 0, 0, 0, 0, 0, 0, 0};

struct CookieTable\_t \*ptr = CookieTable;

while(ptr) {

if (MEMCMP(CI, ptr->CookieI, 8) ||

(MEMCMP (Test, ptr->CookieR, 8)&&MEMCMP (CR, ptr->CookieR, 8)))

ptr = ptr->next;

else

return ptr;

}

return NULL;

}

Таблица нитей второй фазы тоже представляет собой список структур.

struct Phase2Table\_t;

typedef struct Phase2Table\_t {

uchar MessageID[4]; /\* Message ID \*/

int pd; /\* Pipe Descriptor \*/

struct SPIlist\_t SPIlist; /\* List of SPIs \*/

struct Phase2Table\_t next; /\* Next Member (NULL if last) \*/

};

Метод работы со списками *Phase2Table\_t* аналогичен вышеприведенному примеру.

Входные и выходные данные

Общими входными данными (те которые используются для инициатора и для ответчика) является список возможных параметров соединения для первой и второй фаз. Данная информация считывается из конфигурационного файла при запуске программы и хранится в глобальных переменных, доступная для всех нитей. Структуры, описывающие эту информацию, повторяют структуру SA payload, которые предназначены для передачи этих самых вариантов параметров и выбранного случая. Рассмотрим структуры для описания параметров соединения подробнее.

typedef struct Proposal\_t {

uchar ProposalN; /\* Номер Proposal \*/

uchar ProtocolID; /\* Номер протокола \*/

NewGroup\_t \*NewGroup;

uchar NTransforms;

Proposal\_t \*nextPor;

Proposal\_t \*nextPand;

Transform\_t \*nextT;

};

Структура *Proposal\_t* описывает Proposal payload, входящий в состав SA payload. Данная структура содержит все поля необходимые для создания и заполнения данного компонента пакета. Полями структуры являются номер компонента (ProposalN), идентификатор протокола, представляемого этим компонентом (ProtocolID), указатель на структуру содержащую параметры для New Group Mode (если он равен NULL, данные режим не нужен), количество Transform payload. Для связи между собой в данных структурах предусмотрено два указателя – указывающий на следующую структуру объединенную по «И» (nextPand), и указывающий на первую структуру следующей группы, объединенной по ИЛИ (nextPor). Также есть указатель на список соответствующих Transform payload структур.

typedef struct Transform\_t {

uchar TransformN;

uchar TransformID;

Transform\_t \*nextT;

Attributes\_t \*nextA;

};

Структура содержит номер структуры в данном списке, идентификатор представляемого алгоритма, указатель на следующий элемент (напомним, что список одномерный) и указатель на принадлежащий Transform payload список атрибутов.

typedef struct Attributes\_t {

uint type;

ushort SmallVal;

BUFFER BigVal;

Attributes\_t \*nextA;

};

Для атрибутов различают два представления – короткое и длинное. При коротком представлении значение атрибута не превышает 65535 (2 байта), а при длинном задается длина и буфер, содержащий значение атрибута. То, в каком представлении задан (прислан) атрибут определяется первым битом поля *type*. Если он выставлен, то представление короткое. Остальные биты поля *type* показывают, какой это именно атрибут (длина ключа, метод аутентификации и т.п.). Поля SmallVal и BigVal предназначены для хранения значения атрибута при соответственно коротком и длинном представлении. Для каждого типа атрибута определено его представление. Если атрибут считается длинным, то допускается его представление в коротком формате (если значение умещается в 2 байта). Но обратное утверждение не верно – короткий атрибут всегда остается коротким.

Для хранения информации из конфигурации в программе описаны два указателя на структуру *Proposal\_t*, которые содержат набор параметров соответственно для первой и второй фаз. При получении SA payload от партнера, его содержимое также переводится в данные структуры для удобства работы с информацией.

С помощью этих же структур происходит выдача результатов работы программы. Но, т. к. ключевой материал не входит в структуры, то буфер посчитанным ключевым материалом передается отдельно.

Алгоритм обработки входящего пакета

При обработке входящего пакета решается две задачи. Первая задача это отбраковать плохие пакеты (посланные случайно или преднамеренно) по внешним признакам, чтобы избежать лишней траты ресурсов. Вторая задача это найти нить, для которой предназначен пакет.

При решении первой задачи рассматриваются два признака, по которым пакеты проверяются. Первый – значение поля Initiator Cookie. Это поле ни в одном пакете не может быть нулевым. Второй пункт при проверке это длина пакета. Т.к. ее значение передается в ISAKMP заголовке, а он никогда не шифруется, то для каждого пакета мы можем узнать заявленную длину и сравнить с длиной реальной. Вторая задача решается на основе значений полей CookieI и CookieR путем поиска требуемой нити в таблице нитей первой фазы. Способ решения данных задач была подробнее рассмотрена при описании нити распределения пакета, поэтому в данном разделе будет объяснен алгоритм разбора пакета на его компоненты (payload).

Данный заголовок стоит в начале каждого компонента и служит для связывания компонент в список. Первым полем в нем указан тип следующего компонента (тип первого компонента указывается в соответствующем поле ISAKMP заголовка). У последнего компонента данное поле должно быть равно нулю. Второй байт в заголовке является зарезервированным для будущего использования и должен равняться нулю. Последние два байта содержат длину компонента вместе с общим заголовком.

Далее будет рассмотрена функция *CheckPacket*, которая осуществляет проверку структуры списка компонент.

int CheckPacket (State\_t \*state)

{

int next = state->FirstPayload, Length = ISAKMP\_HEADER\_SIZE;

state->LastPacket = 0;

state->ptr = state->Packet.buf + ISAKMP\_HEADER\_SIZE;

do

{

if (state->ptr[1]!= 0) return ERROR /\*Поле RESERVED не ноль\*/

Length += GET\_16BIT (state->ptr+2);

if (Length > state->Packet.len) return ERROR

/\* Вышли за пределы пакета \*/

switch(next)

{

case 1: if (CheckSA(state) < 0) return ERROR;

state->LastPacket |= PAYLOAD\_SA;

break;

case 4: if (CheckKE(state) < 0) return ERROR;

state->LastPacket |= PAYLOAD\_KEY;

break;

………………………………………….

case 13: if (CheckVendor(state) < 0) return ERROR;

state->LastPacket |= PAYLOAD\_VENDOR;

break;

default:

return ERROR;

break;

}

next = state->ptr[0]

state->ptr += GET\_16BIT (state->ptr+2);

} while(next);

return 0;

}

В функцию в качестве параметра передается структура *state*, которая содержит всю информацию, относящуюся к данной нити. В данном случае нам потребовалось значение типа первого компонента *state->FirstPayload* (оно было получено при разборе ISAKMP заголовка), указатель на буфер, содержащий пришедший пакет и длина пакета. Т.к. ISAKMP заголовок уже был разобран, временный указатель смещен на начало первого компонента. Затем начинается цикл по всем компонентам. Сначала проверяется правильность общего заголовка. Для этого проверяем равенство нулю зарезервированного поля. Длину компонента добавляем к сумматору общей длины пакета (*Length)* и проверяем, что заявленная длина компонент не больше длины самого пакета. Затем стоит оператор выбора (*case*), который анализирует значение типа заголовка. Этим выполняется проверка правильности типа компонента, и если тип оказывается неизвестным (или неподдерживаемым), то программа заканчивается с ошибкой. Для каждого известного компонента сначала происходит проверка правильности структуры компонента. Это обусловлено наличием в некоторых из них зарезервированных полей, а также тем, что некоторые из них содержат в себе другие компоненты (SA payload). После проверки выставляется флаг наличия данного компонента в пакете. Данные флаги будут необходимы при семантическом анализе пакета. Следующим действием мы присваиваем переменной содержащей тип следующего компонента новое значение и передвигаем указатель на начало следующего компонента. Выход из данного цикла осуществляется по нулевому значению поля общего заголовка Next payload. Гарантией того, что процесс проверки вообще когда-нибудь кончиться служит проверка того, что длина разбираемых компонент меньше длины пакета. Нормальный выход из цикла означает правильную структуру пакета.

Написание программы и проведение тестирования

В данном разделе будет описан процесс написания и тестирования отдельных функций и модулей. При написании программы, реализующей протокол ISAKMP, тестирование приходилось проводить после написания почти каждой функции обработки очередного пакета. Сначала будут описаны служебные функции и модули, а затем модули непосредственно реализующие протокол ISAKMP.

Служебные функции и модули.

К служебным функциям относятся функции, реализующие некоторые вспомогательные действия, с помощью которых реализуется сам протокол.

# Функции работы с памятью.

Наряду с системными функциями работы в моей программе были реализованы функции работы со структурой виртуального буфера *BUFFER*

typedef struct BUFFER {

uchar \*buf;

int len;

int Len;

};

Данная структура кроме указателя содержит две длины – размер зарезервированного буфера и сколько байт используется в настоящее время. Набор функций выполняет стандартный набор действий (создание, копирование, сравнение, обнуление и удаление), а также конкатенацию двух буферов. Размер виртуального буфера увеличивался динамически при добавлении новых данных. Следующий пример показывает это.

int MEMCPY (BUFFER\* dst, int offset, uchar \*src, int len)

{

uchar \*tmp = NULL;

if(! dst) return ERR\_BADPARAM;

if (offset + len > dst->Len) /\* Проверка достаточности буфера\*/

{

dst->Len = ((offset+len)/ALLOC\_SIZE+1)\*ALLOC\_SIZE;

tmp = (uchar\*) MALLOC (dst->Len); /\* Занятие нового буфера \*/

if(! tmp) return ERR\_NOMEMORY;

memcpy (tmp, dst->buf, dst->len);/\*Копирование старого значения\*/

FREE (dst->buf); /\* Удаление старого буфера \*/

dst->buf = tmp;

}

dst->len = offset + len; /\* Новая длина \*/

memcpy (dst->buf + offset, src, len); /\* Копирование нового значения \*/

return 0;

}

Тестирование данных функций производилось с помощью утилиты, которая считывала буферы из файла, и производила над ними требуемые действия. Результаты по желанию выводились на экран или обратно в файл, где и анализировались.

# Функции работы с сетью.

Включает в себя функции инициализации порта, чтения и записи информации. Для тестирования было написано две программы: клиент и сервер. Задачей клиента было прослушивание заданного порта и вывод полученной информации на экран. Сервер запрашивал адрес клиента и отсылал информацию из файла по указанному адресу. Тестирование заключалось в сравнении отосланной и полученной информации. Также были протестированы случаи прихода сразу нескольких пакетов.

# Функции криптоалгоритмов

В программе используются алгоритмы шифрования DES и Triple DES, алгоритмы хеширования MD5 и SHA1и алгоритмы с открытым ключом RSA и DSA. Реализация всех алгоритмов была взята извне. Исключение составляет только Triple DES, реализация которой основана на основе функций реализации DES.

void des\_3cbc\_encrypt (DESContext \*ks1, DESContext \*ks2,

DESContext \*ks3, u\_char \*iv, u\_char \*dest, const u\_char \*src, u\_long len)

{

word32 iv0, iv1, out[2], out1 [2], out2 [2];

u\_long i;

iv0 = GET\_32BIT(iv); /\* Считывание значения IV \*/

iv1 = GET\_32BIT (iv + 4);

for (i = 0; i < len; i += 8) /\*Обработка в цикле по 8 байт \*/

{

iv0 ^= GET\_32BIT (src + i);

iv1 ^= GET\_32BIT (src + i + 4);

des\_encrypt (iv0, iv1, out1, ks1, 1); /\* Шифрование первым ключом \*/

des\_encrypt (out1 [0], out1 [1], out2, ks2, 0); /\* Расшифрование вторым\*/

des\_encrypt (out2 [0], out2 [1], out, ks3, 1); /\* Шифрование третим \*/

iv0 = out[0];

iv1 = out[1];

PUT\_32BIT (dest + i, iv0);

PUT\_32BIT (dest + i + 4, iv1);

}

PUT\_32BIT\_DES (iv, iv0);

PUT\_32BIT\_DES (iv + 4, iv1);

}

Тестирование алгоритмов шифрования и алгоритмов с открытым ключом проводилось следующим образом. Сначала тестировалась работа самим с собой, т.е. функцией зашифровывался буфер, затем расшифровывался и сравнивался с оригинальным значением. После этого проверялась работа с тестовыми последовательностями, взятыми из стандарта по данным алгоритмам. Алгоритмы хеширования сразу тестировались на стандартных тестовых последовательностях.

# Создание нитей и организации передачи данных между ними.

Нить создается стандартной функцией *pthread\_create.*

#include <pthread.h>

int pthread\_create (pthread\_t\* thread\_ID, const pthread\_attr\_t \*attr,

void \* (\*start\_func) (void \*), void \*arg);

Первый параметр в данной функции это некий дескриптор нити, с помощью которого создающий процесс (нить) может потом ею управлять. Третий параметр – имя функции, выполнением которой и займется новая нить. Как видно из описания функция имеет определенный формат. И последним параметром передается указатель на параметры, передаваемые этой нити при старте. В программе описано 5 функций для запуска нитей – работа с сетью, распределения пакетов, выполнения первой фазы, выполнения второй фазы (одна для New Group Mode и одна для Quick Mode). Первые две нити создаются при старте программы, остальные по мере надобности. Рассмотрим пример создания нити для первой фазы.

#define THREAD\_T pthread\_t

#define THREAD\_SIMPLE\_CREATE (start\_func, arg, tid) \

pthread\_create (tid, NULL, start\_func, arg)

THREAD\_T tid;

……………………….

Record = AddCookieRecord(); /\* Добавление в таблицу нитей первой фазы \*/

if (NULL == Record) return ERROR\_MEMORY;

MEMCPY (Record->CookieI, buff, 8);

Record->Ready = 0;

BUFptr = (BUFFER\*) MALLOC (sizeof(BUFFER)); /\* Создание параметра \*/

if (NULL == BUFptr) return ERROR\_MEMORY;

MEMINIT(BUFptr);

MEMADD (BUFptr, null, 1);

PUT\_32BIT (addr\_tmp, clientaddr.sin\_addr.s\_addr)/\* Запись адреса партнера \*/

MEMADD (BUFptr, addr\_tmp, 4);

PUT\_16BIT (null, length);

MEMADD (BUFptr, null, 2);

MEMADD (BUFptr, buff, length); /\* Запись длины пакета \*/

if (THREAD\_SIMPLE\_CREATE (WorkThread, (void\*) BUFptr, &tid)) {

printf («Thread creation failed\n»);

return 1;

}

……………………….

После принятия решения о том, что для принятого пакета нужно создавать отдельную нить я сначала добавляю новую запись в таблицу нитей первой фазы. В эту запись заноситься Cookie Initiator и флаг Ready обнуляется как знак того, что первая фаза еще не закончена. После этого формируется массив, содержащий информацию необходимую для начала работы нити. В него входит IP адрес отославшего сообщение, длина пакета и собственно пакет. Указатель на виртуальный буфер, содержащий эту информацию, передается в качестве параметра в функцию нити. В самом начале работы каждая нить создает pipe, для того чтобы ей можно было передавать пакеты. Читающий дескриптор она оставляет у себя, а дескриптор для записи она записывает в таблицу нитей первой фазы (запись она находит, зная значение Cookie Initiator). Туда же записывается и значение Cookie Responder, после того как оно будет определено. Для нитей второй фазы процесс создания во многом похож, за исключением того, что в качестве параметра передается указатель на структуру *state*, которая содержит всю требуемую информацию (ключи шифрования, рабочие константы, адреса и т.п.). Процесс распределения дескрипторов pipe для связи остается таким же. Дескриптор записи в нить работы с сетью является глобальной переменной и доступен каждой нити.

Тестирование проводилось путем создания простой модели программы. Простота заключается в отсутствии кода реализующего протокол. Нить просто принимает пакет, увеличивает внутренний счетчик и посылает ответ. При достижении счетчика определенного значения нити первой фазы начинают создавать нити второй фазы, а нити второй фазы, при том же значении счетчика заканчивают работу. Программа для данного тестирования явилась каркасом будущей программы, т. к. потом увеличение счетчика было заменено обработкой и анализом пакета. В данном тесте также проводилось тестирование и функций работы с памятью и функций работы с сетью.

Модули реализации протокола ISAKMP

Данные модули включают в себя функции анализа пакета, обработки и проверки пришедших данных, выполнение требуемых расчетов и создание данных необходимых для сборки пакета.

Как уже упоминалось раньше, весь процесс работы рабочих нитей представляет собой переход из одного так называемого состояния в другое. Под состоянием здесь понимается набор определенных значений некоторых параметров. В программе рассматриваются следующие параметры: роль играемая нитью (инициатор или ответчик), тип обмена, номер пакета и метод аутентификации (только для первой фазы). Все параметры, кроме последнего, могут быть определены в самом начале работы, поэтому для метода аутентификации добавляется значение NONE, означающее неопределенность параметра. Каждый набор параметров однозначно указывает на какое-либо состояние.

Состояние, в свою очередь, состоит из следующих частей: анализ и проверка данных из пакета, проведение расчетов, составление данных для отсылаемого пакета и определение следующего состояния.

Анализ и проверка пришедших данных (часто называемый «семантическим разбором пакета») включает в себя в первую очередь анализ качественного состава пакета, т.е. проверяется все ли из необходимых в данном состоянии компонент присутствуют и нет ли лишних. Все эти проверки осуществляются при анализе переменной, содержащей флаги имеющихся компонент (то, как она определяется, описано в разделе, описывающем работу нити первой фазы). Затем проходит проверка пришедших данных. Для каждого компонента происходит своя проверка, которая может и отсутствовать, например, Nonce payload значение которого просто запоминается. Для SA payload проверка заключается в сравнении пришедшей политики с описанной в конфигурации для ответчика и в проверке правильности присланного ответа для инициатора. Для KE payload происходит проверка длины присланной ключевой информации согласно параметрам, присланным в политике. В Certificate и Certificate Request payloads проверяется тип используемых сертификатов (у меня в программе допустимы только x509 сертификаты). В Identification payload проверяется тип присылаемой информации (допускается только IP адреса) и собственно содержимое компонента. Компонент со значением хеш-функции (Hash payload) проверяется путем подсчета своего варианта и сравнения его с пришедшим значением. В примере приведена функция, вычисляющая значение хеш-функции противоположной стороны.

int CalculateAlienHash (State\_t \*state, BUFFER \*Hash)

{

uchar save;

BUFFER DATA;

MEMINIT(&DATA); /\* Инициализация буфера \*/

if(! ((state->SKEYID).len))

CalculateSKEYID(state); /\* Подсчет SKEYID \*/

MEMADD (&DATA, &(state->AlienKE)); /\* g^x (i/r) \*/

MEMADD (&DATA, &(state->MyKE)); /\* g^x (i/r) \*/

if (GET\_ROLE (state->State) == INITIATOR) {

MEMADD (&DATA, state->CookieR, 8);

MEMADD (&DATA, state->CookieI, 8);

} else {

MEMADD (&DATA, state->CookieI, 8);

MEMADD (&DATA, state->CookieR, 8);

}

MEMADD (&DATA, &(state->SAi\_b));

MEMADD (&DATA, &((state->AlienIdent).Type), 1);

MEMADD (&DATA, (state->AlienIdent).DOI, 3);

MEMADD (&DATA, &((state->AlienIdent).Data));

if (GET\_MODE (state->State) == DSA\_SIGN) {

save = state->HashAlg;

state->HashAlg = HASH\_ALG\_SHA;

M (PRF(state, &(state->SKEYID), &DATA, Hash));

state->HashAlg = save;

} else

M (PRF(state, &(state->SKEYID), &DATA, Hash));

return 0;

}

Формулы, реализованные этой функцией, были представлены при описании фазы 1 (Main Mode). Следует заметить, что эта функция считает не значение инициатора или ответчика, а значение хеш-функции противоположной стороны. Внутри функции это достигается анализом переменной показывающей текущее состояние. Для проверки подписи (располагается в Signature payload) считается данное значение хеш-функции и подписывается требуемым алгоритмом. В приведенном примере есть еще один пример использования переменной состояния. DSA алгоритм может подписывать хеш только от алгоритма SHA. Специально для этого случая значение текущего алгоритма хеширования принудительно приравнивается значению алгоритму SHA. Следует заметить, что в процессе проверки может поменяться текущее значение состояния. Это может произойти при сравнении политик, когда партнеры договариваются о методе аутентификации

После проверки проводятся необходимые расчеты. Большинство формул для расчетов были приведены в разделе о первой фазе.

int CalculateSKEYID\_d (State\_t \*state)

{

uchar z0 = 0;

BUFFER DATA;

MEMINIT(&DATA);

if(! ((state->SKEYID).len))

M (CalculateSKEYID(state));

MEMADD (&DATA, &(state->SharedKey));

MEMADD (&DATA, state->CookieI, 8);

MEMADD (&DATA, state->CookieR, 8);

MEMADD (&DATA, &z0, 1));

PRF (state, &(state->SKEYID), &DATA, &(state->SKEYID\_d));

MEMZERO(&DATA);

return 0;

}

Выше приведен пример расчета одной из рабочих констант.

Рассмотрим пример одной из реализации функций состояния. Состояние возьмем для инициатора, режим – Aggressive Mode, пакет – приход ответа от ответчика.

………………………….

case STATE (INITIATOR, AGGRESSIVE, ABSENT, 1):

if (state. LastPacket & PAYLOAD\_SA)

{

Log (PAYLOAD\_MALFORMED, &state, NULL);

goto THREAD\_END;

}

answer = ProvePolicy (&MyISAKMPPolicy, &(state. AlienPolicy), &state);

if (answer==ERR\_NOPROPOSAL)

{

Log (PAYLOAD\_MALFORMED, &state, NULL);

goto THREAD\_END;

}

switch (state. AuthMeth)

{

case 1: SET\_MODE (state. State, PRESHARED); break;

case 2: SET\_MODE (state. State, DSA\_SIGN); break;

case 3: SET\_MODE (state. State, RSA\_SIGN); break;

case 4: case 5:

default:

Log (NOT\_SUPPORTED, &state, NULL);

goto THREAD\_END;

}

ONE\_MORE = 1;

break;

……………………….

Данный пример весьма показателен тем, что здесь происходит смена значения текущего состояния при проверке пакета. Пакет содержит выбранную ответчиком политику, а, следовательно, и выбранный метод аутентификации. По этому значению происходит смена значения текущего состояния и выставляется флаг, говорящий, что надо повторить оператор выбора состояния, чтобы выполнить действия необходимые именно данному методу аутентификации.

case STATE (INITIATOR, AGGRESSIVE, PRESHARED, 1):

ERR (CheckHash(&state));

ERR (CalculateEncKeysPhase1 (&state));

ERR (MakeISAKMP(&state));

ERR (MakeHash(&state));

ERR (SendPacket(&state, 0));

NOT\_END = 0;

break;

Приведенный набор действий будет выполнен, если в предыдущем примере партнеры договорились о методе аутентификации с помощью заранее известного секретного ключа. На данном примере видны все три составляющих состояния. Сначала проходит аутентификация ответчика путем проверки значения Hash payload. Затем производится расчет ключевой информации для первой фазы. Завершают все функции создания компонент пакета и отсылка самого пакета. Так как это последний пакет со стороны инициатора, то значение состояния не изменяется и выставляется флаг, говорящий об окончании фазы.

Описанный принцип состояний использовался для написания логики повеления во всех обменах. Любая рабочая нить имела следующую структуру:

/\* Инициализация нити \*/

while (NOT\_END) {/\* Проверка признака окончания работы \*/

if (GET\_STEP (state. State)) /\* Если это не первый пакет \*/

{

ERR (RecivePacket(&state)); /\* Ожидание пакета \*/

ERR (DecryptPacket(&state)); /\* Расшифрование пакета \*/

}

ONE\_MORE = 1;

while (ONE\_MORE) {/\* Новый выбор без посылки пакета \*/

ONE\_MORE = 0;

switch (state. State) {

case STATE (RESPONDER, AGGRESSIVE, ABSENT, 0):

case STATE (RESPONDER, MAIN, ABSENT, 0):

FreePolicy (state. AlienPolicy);

ERR (CheckPacket(&state));

ONE\_MORE = 1;

break;

case STATE (RESPONDER, MAIN, RSA\_SIGN, 0):

…….

state. State++; /\* Следующий номер пакета \*/

break;

……………….

} /\* switch \*/

} /\* while (ONE\_MORE) \*/

} /\* while (NOT\_END)

/\* Действия выполняемые в конце нити \*/

Основу нити составляет цикл ожидания конца работы. Внутри нити сначала происходит ожидание пакета (если это не самое начало обмена), затем идет цикл, позволяющий проводит насколько выборов состояний. Затем идет непосредственно выбор состояния и выполнение соответствующих состоянию набора функций.

Тестирование функций реализации протокола должно проходить после реализации каждого состояния. В самом начале в тестах поможет утилита клиента, оставшаяся от тестов функций работы с сетью. Ею можно будет просматривать пакеты, если еще не созданы состояния для ответчика. Тестирование заключается в запускании процесса установления соединения (пусть даже оно заведомо не завершиться) и анализе информации выдаваемой программой в процессе работы на экран или в файл. В процессе тестирования могут быть обнаружены следующие типы ошибок. Программа просто не работает, т.е. присутствует некая ошибка исполнения программы. Если она не очевидна сразу, то ее поиск производится с помощью отладчика. Второй вариант – программа работает, но вырабатываются неправильные данные или есть неверная реакция на правильные данные. В большинстве случаев подобная ситуация решается анализом информации выдаваемой программой в процессе работы (лог). Если из существующего лога ошибка не находится, то можно сначала попробовать использовать более детальный лог. Если это не помогает, то используется отладчик, хотя использование этого метода для подобных ошибок может стать весьма времязанимающим занятием. После того, как какой-то этап, был протестирован на работу самим с собой можно, переходить на тестирование с другими реализациями протокола. Причем это делалось также после написания каждого состояния.

Тестирование с другими реализациями протокола ISAKMP

Во время написания и отладки моей программы, внешней реализацией для тестирования был выбран сайт www.ssh.fi, содержащий свою реализацию данного протокола. Данный сервер имеет хорошо разветвленную систему настроек политики чужой стороны, что позволяет определять посылаемый мне SA payload с точностью до атрибутов (что очень помогло при тестировании выбора приемлемых атрибутов). Также на сервере хорошо реализована система показа процесса работы программы. Расчет каждой величины предваряется перечислением входных данных и используемых алгоритмов. Это очень помогает при поиске ошибок, связанных с неправильным вычислением констант или неправильном использовании алгоритмов, когда видно, что входные данные те же, а результат другой.

Отладка с данной реализацией проводилась методом описанном в предыдущем разделе. Т.е. после каждого созданного состояния проводилось тестирование при разных входных данных.

Заключение

В данном разделе были рассмотрены общие принципы создания программы реализации протокола ISAKMP. Рассмотрены способы использования многонитевых структур программы, указан необходимый набор функций для создания нити и обеспечения передачи данных между нитями. Представлена структура программы, описаны использующиеся в ней нити, порядок их реализации и интерфейс общения между ними. Рассмотрен порядок разработки программы и методика проведения тестирования программы в процессе ее создания.

**Технология реализации протокола ISAKMP**

Исходя из технического задания, данная реализация протокола ISAKMP должна содержать 4 обмена (Main Mode, Aggressive Mode, New Group Mode и Quick Mode) и должна поддерживать 4 метода аутентификации (методом заранее известного секретного ключа, цифровой подписью с помощью алгоритмов DSS и RSA, шифрования открытым ключом с помощью алгоритма RSA). Весь процесс написания программы можно разбить на 6 частей:

1. Подготовительная часть
2. Реализация Main mode с методом аутентификации заранее известного секретного ключа
3. Реализация Quick Mode
4. Реализация остальных методов аутентификации для Main Mode
5. Реализация Aggressive Mode со всеми методами аутентификации
6. Реализация New Group Mode

При таком порядке написания программы сначала будет написана версия с минимальной функциональностью (для простейшей конфигурации протокол будет работать уже после третьего этапа), а уже затем происходит выполнение всех требований технического задания.

Подготовительная часть

Подготовительная часть включает в себя написание вспомогательных модулей, которые можно выделить в отдельные задачи.

* Реализация криптоалгоритмов. Включает в себя написание функции реализующих алгоритмы шифрования DES и Triple DES, алгоритмы хеширования MD5 и SHA и алгоритмы с открытым ключом RSA и DSA. Реализация включает в себя проверку правильности работы функций. Для алгоритмов шифрования сначала происходит проверка работы функций самих с собой (шифрование произвольный текст, расшифрование и проверка идентичность), затем проверка работоспособности с другими реализациями данного алгоритма и проверка тестовых последовательностей. Для алгоритмов хеширования возможна только проверка тестовых последовательностей.
* Реализация алгоритма Diffie-Hellman. Включает в себя написание функций подсчета открытого ключа по известному секретному ключу и расчета общего ключа для любых заданных параметров алгоритма. Тестирование сначала производится между собой (расчет двух пар ключей, расчет общего ключа для обеих сторон и проверка идентичности), затем с другими реализациями данного алгоритма.
* Работа с сетью. Включает в себя функции инициализации и закрытия портов, отсылки и приема пакетов. Тестирование заключается в проверке идентичности отправленных и полученных пакетов.
* Конфигурация. Включает в себя считывание конфигурации из указанного файла, проверка ее правильности и заполнение, согласно ей, внутренних структур.
* Лог. Включает в себя функции инициализации и записи в лог-файл. Функция записи пишется с расчетом на много нитевую программу.
* Работа с буферами переменной длины. Сам буфер представляется в виде структуры. Функции реализуют следующие действия: создание, очистка, копирование в буфер и из него, добавление информации в буфер, распечатка, удаление. При тестировании следует написать тестовую программу, реализующую все требуемые действия.
* Работа с сертификатами. Реализуются функции загрузки сертификата из файла, проверка правильности сертификата, получение информации из сертификатов. В список перечисленных действий не входит создание сертификатов, т.е. нам придется сертификаты, созданные третьей стороной. При тестировании происходит проверка работоспособности всех функций, причем все тесты должны проходить для сертификатов, полученных из разных мест.
* Проверка структуры пакета. Протокол ISAKMP четко определяет структуру пакетов. Тестирование осуществляется путем ручного задания правильно и неправильно сформированных пакетов.

Часть из вышеперечисленных задач можно делать параллельно с выполнением основной задачи (например, работа с сертификатами нужно лишь при реализации методов аутентификации цифровой подписью и шифрованием открытым ключом).

Реализация Main mode c методом аутентификации заранее известного секретного ключа

Main mode состоит из 6-и посылок пакетов – 3 от каждой из сторон (инициатора и ответчика). Также здесь реализуется прием задания для стороны инициатора.

Реализация на данном этапе только одного из 4 методов аутентификации продиктовано желанием получить сначала работающую версию, а уже затем добавлять в нее необходимую функциональность. Не смотря на это, в программе учитываются все методы аутентификации, но вместо всех не нужных подставляются «заглушки».

Каждый из обменов реализуется сразу как для инициатора, так и для ответчика. После выполнения каждого из обменов происходит тестирование. Сначала пробуется работа двух реализаций, а затем тестируется работа с другой реализацией протокола.

Порядок реализации данного этапа следующий:

1. Получение задания на установление соединения со стороны инициатора, проверка необходимости проведения первой фазы (возможно для данного партнера ISAKMP SA уже создана и тогда сразу переходим ко второй фазе);
2. Составление и отсылка инициатором первого. Наиболее сложным здесь является запись политики заданной в конфигурационном файле в структуру пакета ISAKMP;
3. Прием и разбор первого пакета ответчиком. Выбор из предложенной политики приемлемого варианта. Составление и отсылка второго пакета, содержащего выбранную политику;
4. Прием и разбор второго пакета инициатором. Проверка корректности выбранной политики. Получение секретного и отрытого ключей для алгоритма Diffie-Hellman с параметрами, оговоренными в политике. Составление и отсылка третьего пакета, содержащего свой открытый ключ и Nonce – случайную последовательность для данного соединения;
5. Прием и разбор третьего пакета ответчиком. Сохранение Nonce и открытого ключа инициатора, вычисление своей пары Diffie-Hellman ключей. Составление и отсылка четвертого пакета, содержащего открытый ключ и Nonce ответчика;
6. Прием и разбор четвертого пакета инициатором. Вычисление общего ключа для алгоритма Diffie-Hellman с помощью своего секретного ключа и открытого ключа ответчика. Получение аутентифицирующей информации Составление пятого пакета, содержащего информацию, идентифицирующую и аутентифицирующую инициатора. Вычисление ключей шифрования. Шифрование полученного пакета с помощью алгоритмов, оговоренных в политике, и только что вычисленных ключей. Отсылка зашифрованного пакета;
7. Прием пятого пакета ответчиком. Вычисление общего ключа для алгоритма Diffie-Hellman. Вычисление ключей шифрования. Расшифровка пакета и его разбор. Проверка идентифицирующей информации инициатора. Вычисление аутентифицирующей инициатора информации и сравнение со значением находящимся в пакете. Вычисление информации идентифицирующей и аутентифицирующей ответчика. Составление, шифрование и отсылка шестого пакета;
8. Прием шестого пакета инициатором. Расшифровка пакета и проверка информации идентифицирующей и аутентифицирующей ответчика.

Результатом первой фазы становиться ISAKMP SA, которая включает в себя алгоритмы, о которых стороны договорились во время обмена политиками, и ключевая информация, которая считается при следующих обменах. ISAKMP SA используется для защиты всех последующих обменов.

Ошибки, всплывающие в процессе работы и при тестировании, можно разделить на ошибки влияющие на работоспособность программы (критические ошибки периода исполнения, ошибки доступа к ресурсам – т.е. когда программа прекращает свою работу) и ошибки связанные с реализацией протокола (неправильно составленный пакет, неправильный разбор пришедшего пакета и т.д.). Ошибки первого типа в основном обнаруживаются с помощью отладчика. Ошибки второго типа обнаружить сложнее, т. к. зачастую они скрываются в логике программы. Первым шагом при их поиске является тщательное изучение логов с обеих сторон, так как неправильная реакция на пришедший пакет на одной из сторон может быть вызвана не соблюдением правил составления пакета с другой стороны. Возможно, в процессе анализа потребуется сделать лог более подробным для всего процесса установления соединения или для отдельных моментов. В результате анализа лога должно быть найдено место, где программа повела себя неправильно. Если ошибка, вызвавшая это неправильное действие очевидна, то она исправляется. Иначе ее поиск продолжается, но уже с помощью отладчика.

После написания всех трех обменов происходит тестирования Main mode в целом. Сначала тестируется работа двух реализаций программы в роли и инициатора и ответчика. Затем тестируется совместимость с другими реализациями протокола (в процессе разработки для тестирования совместимости использовался сервер www.ssh.fi). Тестирование включает в себе проведение одиночного соединения и сразу нескольких (причем каждая из сторон в разных соединениях играет разную роль), проведение удачных и неудачных соединений. Также в процессе тестирования проверяется одинаковая работа программы на разных процессорах (Sun Sparc и Intel x86).

Реализация Quick mode

Quick Mode, представляющий собой вторую фазу установления соединения, состоит из 3-х посылок пакетов – две со стороны инициатора и одна со стороны ответчика. Порядок разработки и тестирования такой же, как на предыдущем этапе. Данный режим проходит под защитой ISAKMP SA, полеченной во время первой фазы.

На этом этапе сначала реализуется рабочий минимум (т.е. без повторного обмена ключевой информацией и посылки идентификационной информации), а затем, когда этот минимум будет полностью оттестирован, доделываются остальные части. Таким образом, порядок реализации данного этапа будет следующим:

1. Получение инициатором из конфигурационного файла политики для второй фазы. Составление первого пакета, содержащего предлагаемую политику и Nonce. Вычисление значения хэш-функции от пакета, добавление этого значение в пакет и шифрование пакета. Отсылка пакета ответчику.
2. Прием пакета, расшифрование и проверка целостности путем вычисления значение хеш-функции от пакета и сравнение с присланным значением. Выбор из присланной политики приемлемого варианта политики. Составление второго пакета, содержащего выбранную политику и свой Nonce. Вычисление значения хеш-функции от пакета, шифрования пакета и отсылка его инициатору.
3. Прием инициатором второго пакета. Расшифрование пакета и проверка целостности. Проверка корректности выбранного варианта политики. Вычисление значения хеш-функции от Nonce-ов. Составление третьего пакета, его шифрование и отсылка ответчику. Вычисление выходных результатов.
4. Прием ответчиком третьего пакета. Расшифровка пакета, вычисление значения хеш-функции от Nonce-ов и сравнение с присланным значением. Вычисление выходных результатов.
5. Если в конфигурации указана необходимость в обмене ключами осуществить расчет пар ключей с каждой из сторон, обмен открытыми ключами и получение общего ключа аналогично тому, как это делалось на предыдущем этапе. Также учесть наличие общего ключа при расчете выходных результатов.
6. Если в конфигурации указана необходимость посылки идентификационной информации, то со стороны инициатора обеспечить включение этой информации в первый пакет, а со стороны ответчика ее проверку.

Все мероприятия связанные с тестированием проводятся также как и на первом этапе. Должны проводиться тесты после реализации каждого взаимного обмена и финальные тесты, включающие в себя одновременное проведение нескольких соединений, использование различных конфигураций (провести вторую фазу в разных объемах), тестирование с другими реализациями протокола и тестирование работы программы на разных процессорах.

Реализация остальных методов аутентификации для Main mode

На данном этапе добавляются возможные методы аутентификации для первой фазы. Все эти методы используют алгоритмы шифрования с открытым ключом и сертификаты как способ передачи открытых ключей. Следует заметить, что необходимость в функциях работы с сертификатами и реализациях алгоритмов DSA и RSA встречается впервые именно на этом этапе, что позволяет распараллелить работы над этими задачами.

Как и на предыдущих этапах, сначала реализуется рабочий минимум, который включает в себя собственно вычисление и проверку аутентификационной информации, затем добавить возможности обмена сертификатами.

Порядок реализации данного этапа следующий:

1. Реализация ветвления по разным методам аутентификации в зависимости от договоренной политики;
2. Реализация методов аутентификации с помощью подписи алгоритмами DSA и RSA. Включает в себя вычисление подписи каждой из сторон для специально вычисляемого значения хеш-функции и проверка подписи на другой стороне. Сертификаты для проверки подписи другой стороны задаются явно.
3. Реализация метода аутентификации шифрованием открытым ключом алгоритмом RSA. Сертификаты для расшифрования также задаются явно.
4. Реализация обмена сертификатами. Включает в себя реализацию запроса сертификата другой стороны и отсылку своего сертификата на запрос и без запроса. Информация о том надо ли отсылать свой сертификат, запрашивать ли чужой и т.п. берется из конфигурационного файла.

После реализации данного этапа проходит тот же набор тестов, что проводился по окончанию второго этапа. Но в данном случае делается упор на правильность работы именно вновь добавленных методов аутентификации. Особое внимание уделяется правильной диагностике ошибок возникающих при не нахождении сертификатов и получении сертификат, чей тип отличается от ожидаемого значения.

Реализация Aggressive mode со всеми методами аутентификации

На этом этапе реализуется другой режим для проведения первой фазы. От Main mode его отличает меньшее количество посылаемых пакетов, но вместе с этим большее количество информации передаваемых в пакетах и большая трудоемкость при их обработке. Реально данный режим реализуется на основе кода написанного для Main mode путем перестановки основных функций, но при этом надо помнить, что простая перестановка не поможет и требуется некий дополнительный анализ. Например, в политике, передаваемой инициатором, не может быть предложен выбор между различными Oakley группами, т. к. в этом же пакете посылается публичный ключ однозначно определяющий группу. Все подобные проверки должны проводиться для Aggressive Mode.

Порядок реализации этапа следующий:

1. Реализация ветвления между Main и Aggressive режимами. Для инициатора вид режима определяется конфигурацией. Ответчик проверяет приемлемость предложенного режима, также основываясь на конфигурации и, в случае не приемлемого значения, может отказаться продолжать обмены. Инициатор должен в этом случае уметь предложить другой режим, конечно, если это позволяет реализация.
2. Реализация метода аутентификации заранее известного секретного ключа. После реализации проводят тесты, проводимые на втором этапе, но с использованием Aggressive mode.
3. Реализация остальных методов аутентификации. Этапы реализации и тесты аналогичны такие же, как на четвертом этапе.

Как уже упоминалось выше, тесты во время разработки и по ее окончанию аналогичны тестам проводимых на втором и четвертом этапах, но с упором на использование различных режимов в первой фазе. Особо стоит обратить внимание на повторную попытку инициатора провести соединение в другом режиме после отказа со стороны ответчика.

Реализация New Group Mode

Данный режим предназначен для согласования нестандартных Oakley групп и заключается лишь в обмене политиками. Но основная сложность заключается во встраивании использования этого режима и правильном использовании результатов работы режима во второй фазе. Сам режим, также как и Quick mode, проходит под защитой ISAKMP SA.

Порядок реализации этапа следующий:

1. Реализация со стороны инициатора ответвления на New Group mode в зависимости от конфигурации. Должна быть предусмотрена проверка существования уже созданной нестандартной Oakley группы.
2. Создания инициатором пакета содержащего информацию о предлагаемой группе, подсчет значения хеш-функции от этого пакета, шифрования пакета (с помощью алгоритмов и ключей из ISAKMP SA) и отсылка пакета;
3. Получение пакета, его расшифровка и проверка целостности. Проверка приемлемости предлагаемой группы. Составление ответного пакета, подсчет значения хеш-функции, шифрование и отсылка этого пакета. Сохранение информации о группе для второй фазы;
4. Получение пакета, его расшифровка и проверка целостности. Проверка корректности присланной информации. Сохранение информации о группе для второй фазы.
5. Реализация использования во время второй фазы групп, о которых договаривались во время New Group mode.

При тестировании проверяется правильная работа в New Group mode и инициатора, и ответчика. Особое внимание стоит уделить правильной работе программы при попытке договориться о нескольких Oakley группах.

В данном разделе был рассмотрен порядок реализации протокола ISAKMP, предложен вариант разбиения общей задачи на подзадачи, даны рекомендации по организации работ над этими задачами и предложен порядок проведения тестов.

Следует заметить, что данный порядок реализации протокола ISAKMP был предложен в расчете на определенные технические требования к программе и в расчете на работу одного человека. При других требованиях или если количество работающих над программой человек изменяется, порядок реализации протокола может существенно измениться как на уровне этапов, так и на более низких уровнях.

**Сегментация рынка пользователей программы, реализующей протокол ISAKMP**

Успех продвижения на рынке новых товаров во многом зависит от всестороннего исследования требований рынка. Исследуемая информация касается спроса на товары и услуги различных уровней, уже имеющихся и потенциально возможных конкурентов, а также требований, предъявляемых потребителями. Сбор подобной информации требует значительных затрат времени и средств. Это заставляет предприятия нацеливаться на отдельные части рынка, которые представляют собой сегменты групп потребителей с примерно общими требованиями. Поиск таких однородных сегментов потребителей среди различных вариантов требований, предъявляемых к товару, называется сегментацией рынка, а данный найденный участок рынка – сегментом рынка.

При разумном делении рынка на сегменты все инструменты маркетинга внутри него могут быть оптимально скоординированы. Именно поэтому сегментация рынка считается очень важным аспектом деятельности предприятия.

Методика расчёта сегментации рынка

При первичной сегментации всего рынка целесообразно выделить сегменты товаров потребительского рынка или производственного назначения. Такая классификация важна, поскольку подчеркивает различия в характеристиках продуктов и последствия для маркетолога.

Для дальнейшего деления рынка на сегменты можно воспользоваться различными критериями в зависимости от следующих факторов:

* географического положения потребителей (регион, страна);
* типа потребителя (величина предприятия, интенсивность потребления, отрасль, место в производственном процессе);
* типа процесса, для которого приобретается продукция (административная деятельность, движение товара, производственный процесс);
* покупательского спроса (клиент / потенциальный клиент, связь с поставщиком, частота и величина закупок);

На рынках сбыта товаров широкого потребления используют другие критерии. Классическими являются следующие показатели:

* социально-экономические (образования, доходы);
* демографические (возраст, пол, состав семьи);
* географические

Однако следует учитывать, что всех потребителей на рынке не так-то легко разделить по категориям. Поведение потребителя в последнее время становиться все более дифференцированным, возникают различные «стили жизни» внутри общества.

Для формирования сегментации рынка используются элементы таксономического анализа – построение диаграмм Чекановского. Исходным шагом, предопределяющим правильность конечных результатов, является оформление матрицы наблюдений.

Признаки, включенные в матрицу, могут быть неоднородны, поскольку описывают разные свойства объектов. Кроме того, различаются единицы их измерения. Поэтому надлежит выполнить предварительное преобразование, которое заключается в стандартизации признаков.

Таблица 1. Неупорядоченная диаграмма Чекановского

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номера  единиц | 1 | 2 | … |  |
| 1 | X |  |  | X |
| 2 |  |  | X |  |
| … | … | … | … | … |
| w |  |  |  | X |

В приведенной неупорядоченной диаграмме очередность записи единиц целиком случайна. На это указывает явственный разброс символов, обозначающих разницу между изучаемыми единицами: наименьшее численное расстояние – Χ; наибольшее расстояние, т.е. пары единиц, наиболее разнящиеся между собой, – Для их линейного упорядочения следует произвести перегруппировку знаков Χ и. Перегруппировка должна выполняться таким образом, чтобы указанные знаки оказались как можно ближе к главной диагонали диаграммы. С этой целью строки и столбцы таблицы переставляются до тех пор, пока не получится упорядоченная диаграмма.

Поиск сегментов рынка для программы установки защищенных сетевых соединений с помощью протокола ISAKMP

Взрывной характер развития компьютерных технологий и резко возросшее количество действий совершаемых с помощью глобальной сети Internet (развитие электронной торговли, предоставление компаниями через сеть ряда услуг для своих клиентов и т.д.) привело к резкому увеличению объемов информации передаваемой по сети. Также претерпел изменения и качественный состав передаваемой информации – возросла доля конфиденциальной информации. Вместе с тем возникла необходимость аутентификации другой стороны при всех вышеперечисленных действиях. Все это привело к увеличению рынка программных продуктов предназначенных для защиты информации в сети. Представленная программа сама по себе не производит непосредственно защиту передаваемой информации. Задачей программы является аутентификация другой стороны и подготовка информации, необходимой для непосредственной защиты передаваемой информации.

Выделим потребителей программы:

1. Системы авторизации доступа, т.е. система, которой требуется провести идентификацию и аутентификацию пользователей. Примером может служить сервер компании, предоставляющий ряд услуг клиентам компании. Программа в данном случае поможет отсеять постороннего посетителя от клиента.
2. Структура доступа для внутри корпоративной сети. Программа поможет произвести ограничение доступа к информации различных отделов и подразделений. Например, рядовые работники имеют доступ к рабочим серверам компании, но не могут воспользоваться информацией бухгалтерии. Также возможно сегментирование информации внутри отдельного подразделения. Например, отдел маркетинга может воспользоваться информацией из бухгалтерии о финансовом состоянии компании, но не имеет доступ к информации о заработной плате сотрудников.
3. Виртуальная корпоративная сеть. Обычно корпоративная сеть представляет собой замкнутую, самодостаточную локальную сеть, которая общается с внешним миром через компьютер фильтрующий проходящую через него информацию. При этом никакая секретная информация из локальной сети не попадает во внешнюю сеть. Недостатком данного подхода являлась то, что все участники этой сети должны были находиться в непосредственной близости друг от друга. Программа позволяет построить виртуальную защищенную сеть на основе глобальной сети Internet, шифруя всю информацию, передаваемую между участниками этой виртуальной сети. При таком подходе можно организовывать сеть даже между людьми, находящимися в разных странах.
4. Сеть банкоматов. Программа поможет создать защищенные соединения с банком для обмена информации о произведенных операциях.
5. Межкорпоративная сеть и сеть для связи с филиалами компании. Программа обеспечивает создание защищенного соединения для передачи информации между внутренними сетями разных компаний или разных филиалов одной компании (это может быть или локальная сеть, или виртуальная защищенная сеть).

Рассмотрим параметры программы, которые влияют на ее функциональность и на способ использования программы:

1. Простота настроек и обслуживания. Определяет уровень теоретической подготовки оператора программы.
2. Объем настроек. Показывает, сколько имеется параметров для настройки программы. Больший объем настроек позволяет более гибко настроить программу.
3. Полнота методов аутентификации. Сколько реализовано методов аутентификации и их функциональная полнота.
4. Полнота сведений о процессе работы программы. Способ сохранения этих сведений, способ задания количественных и качественных параметров обрабатываемых событий.
5. Полнота реализации протокола ISAKMP и совместимость с продуктами третьей стороны.

**X**= **Z**=



По формуле (4) рассчитаем матрицу расстояний:

**C**=



Разбиваем полученную матрицу на классы, где X – соответствует наименьшему численному расстоянию между изучаемыми задачами (0–1) и получаем неупорядоченную матрицу Чекановского:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** |
| **1** | **X** | X |  | **X** |  |
| **2** | **X** | **X** |  | **X** |  |
| **3** |  |  | **X** |  |  |
| **4** | **X** | **X** |  | **X** |  |
| **5** |  |  |  |  | **X** |

Произведя перегруппировку строк и столбцов (поменяли местами строки / столбцы 3 и 4) получаем упорядоченную диаграмму:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **4** | **3** | **5** |
| **1** | **X** | **X** | **X** |  |  |
| **2** | X | **X** | **X** |  |  |
| **4** | **X** | **X** |  | **X** |  |
| **3** |  |  | **X** |  |  |
| **5** |  |  |  |  | **X** |

В результате выполненных вычислений выделился сегмент пользователей программы, включающий в себя пользователей использующих программу для создания систем авторизации доступа, систем разграничения доступа и сетей банкоматов.

В результате сегментации рынка пользователей программы установления защищенных сетевых соединений с помощью протокола ISAKMP был выделен объединенный сегмент рынка, включающий в себя использование программы для создания систем авторизации доступа, систем разграничения доступа и сетей банкоматов. Для этого сегмента характерны повышенные требования к реализации методов аутентификации, системе протоколирования произошедших событий и объему возможных настроек программы.

**Разработка мероприятий по безопасности работы с монитором ПК**

Вычислительные комплексы на базе персональных ЭВМ являются одним из основных средств труда разработчика на всех этапах создания программы (проектирование, написание, тестирование и отладка).

Рассмотрев факторы обитаемости в данной производственной среде, можно выделить следующие факторы, оказывающие вредное воздействие на организм человека.

* Эмиссионные:
* Повышенный уровень электромагнитных излучений:
* низкочастотного электромагнитного поля (51 ц-400кГц);
* низкоэффективного (мягкого) рентгеновского излучения (при напряжении на ЭЛТ 15 кВ и выше);
* Повышенный уровень электростатического поля;
* Эргономические:
* Не эргономичность визуальных параметров дисплея. Не эргономичность конструкции дисплея и клавиатуры;
* Не эргономичность рабочего стола и рабочего стула (кресла);
* Физические:
* Повышенная температура, пониженная влажность воздуха рабочей зоны;
* Повышенный уровень шума на рабочем месте;
* Недостаточная освещенность рабочих поверхностей;
* Повышенная яркость света в плоскости экрана дисплея;
* Прямая и отраженная блескость;
* Повышенная пульсация освещенности от газоразрядных источников света;
* Ионизация воздуха;

Психофизиологические:

* нервно-психические перегрузки:
* перенапряжение зрительного анализатора;
* умственное перенапряжение;
* эмоциональные перегрузки;
* монотонность труда;
* Физические перегрузки:
* статические перегрузки костно-мышечного аппарата;
* локальные динамические перегрузки мышц кистей рук;

Источником значительной части перечисленных выше вредных воздействий является монитор персональной ЭВМ.

Электромагнитное излучение монитора ЭВМ

Основным источником эргономических проблем, связанных с охраной здоровья людей, использующих в своей работе персональные компьютеры, являются дисплеи (мониторы), особенно дисплеи с электронно-лучевыми трубками. Они представляют собой источники наиболее вредных излучений, неблагоприятно влияющих на здоровье операторов. Электромагнитные излучения рабочей аппаратуры обусловлены некачественным экранированием источников излучения в аппаратуре. Кроме этого, оператор подвергается воздействию излучения от рабочей поверхности электронно-лучевой трубки.

Приложение общих положений теории электродинамических явлений к конструкции конкретных электрических приборов, в частности монитора ЭВМ, позволяет сделать некоторые выводы относительно источников и конфигурации электрических и магнитных полей, излучаемых этими приборами. Известно, что электрическое поле излучается теми частями электрических установок, в которых используются высокие напряжения, а магнитное поле излучается сильными токами.

В компьютере высокие напряжения используются в ускорительной системе электроннолучевой трубки (ЭЛТ) монитора, а сильные токи текут в системе управления электронными лучами трубки и цепях блока питания. Именно эти части монитора ЭВМ и являются основными источниками электромагнитного излучения. Силовые линии электрического поля можно представить начинающимися в области вблизи заднего конца ЭЛТ и оканчивающимися на поверхностях, находящихся вблизи монитора, в том числе и на поверхности тела пользователя ЭВМ, сидящего перед компьютером.

Силовые линии магнитного поля образуют замкнутые конфигурации, начинающиеся и заканчивающиеся на магнитных кольцах фокусирующей системы ЭЛТ. Непосредственно перед экраном монитора плотность магнитного потока достигает величин единиц мкТл, но быстро убывает с расстоянием от монитора.

Обнаружено, что:

1. Электромагнитное поле возбуждается на частотах кадровой (60 Гц) и строчной (22 кГц) разверток и их гармоник;
2. Электрическое поле ВДТ близко к электрическому полю точечного заряда, а магнитное – к полю магнитного диполя, находящихся в геометрическом центре ВДТ. При этом частоту 60 Гц излучает система токов, близкая к горизонтальному диполю, а 22 кГц – к вертикальному;
3. При удалении от экрана ВДТ поля быстро спадают. Например, электрическое поле спадает в ~ 40 раз при удалении от экрана на расстояние 1,25 м.

Длительное воздействие на организм человека электромагнитных излучений, превышающих допустимые нормы, может привести к некоторым функциональным изменениям в организме или даже повреждениям тканей и органов. Симптомами являются головная боль, нарушение сна, повышенная утомляемость. Функциональные изменения, вызываемые электромагнитными излучениями, способны накапливаться в организме.

При выборе монитора следует обращать внимание на наличие на шильдике (табличка с перечнем заводских параметров изделия) надписи о том, что данная модель прошла тестирование на предмет соответствия ТСО 95 (стандарт Шведской конференции профсоюзов) или MPR II (стандарт Шведского национального комитета по защите от излучений). Желательно также получить сведения о наличии Гигиенического Сертификата либо сертификатов, выданных другими организациями. В то же время, следует иметь в виду, что разброс возможных уровней электромагнитного излучения мониторов одной и той же модели может достигать 50%.

Пользователям персональных компьютеров, желающим снизить уровень облучения переменными магнитными полями, следует расположить мониторы так, чтобы расстояние до них составляло величину, равную расстоянию вытянутой руки (с вытянутыми пальцами). Поскольку магнитные поля сзади и по бокам большинства мониторов значительно сильнее, чем перед экраном, пользователи должны располагать свои рабочие места на расстоянии не менее 1.22 м от боковых и задних стенок других компьютеров. Также для защиты от электромагнитных излучений рекомендуется использовать специальные защитные экраны. Они изготавливаются из особого стекла и устанавливаются между рабочей поверхностью монитора и оператором. Такая защита обеспечивает задержку от 30 до 90 процентов всех вредных излучений. Такого же результата можно добиться путем удаления источника излучения от оператора. Тем не менее, не рекомендуется проводить за экраном дисплея более 3-х часов в день.

Электроопасность и пожароопасность

Мониторы ПК питаются от сети переменного тока напряжением 220 В с частотой 50 Гц, что являет само по себе серьезную опасность для жизни и здоровья человека.

Действие электрического тока на живую ткань носит разносторонний характер: термическое воздействие, электрическое и биологическое действия. Все это ведет к электрическим травмам и электрическим ударам, что в свою очередь может привести к нарушению и даже к полному прекращению жизнедеятельности организма.

Исход воздействия электрического тока на организм зависит от ряда факторов, в том числе и от электрического сопротивления тела, величины и продолжительности воздействия тока, рода и частоты тока. Пороговый ощутимый ток составляет 0,6…1,5 мА для постоянного тока.

Безопасный ток, который может в течение длительного времени проходить через человека, не вызывая никаких ощущений, составляет приблизительно 50 мкА (для переменного тока с частотой 50 Гц) и 100 мкА (для постоянного тока). При увеличении величины тока до 10…15 мА боль становится едва переносимой, и судороги мышц становятся настолько значительными, что человек не в состоянии их преодолеть. Таким образом, пороговый не отпускающий ток составляет 10…15 мА для частоты 50 Гц и 50…80 мА для постоянного тока. Ток величиной 100 мА (частотой 50 Гц) и 300 мА (постоянный ток) и более вызывают прекращение деятельности сердца через 1–2 с.

Помещение для работы с ЭВМ и с ее внешними устройствами обычно относят к категории помещений с повышенной опасностью, т. к. имеется возможность поражения электрическим током. Чаще всего источниками поражения являются блоки ЭВМ, корпуса устройств и приборов в случае возникновения неисправности (например, при нарушении защитного заземления или изоляции проводов, а также при применении неправильных приемов включения в сеть и выключения из сети вилок электропитания).

Защитой от прикосновения к токоведущим частям электроустановок служит:

изоляция проводников;

использование защитных кожухов;

использование инструментов с изолирующими ручками при ремонте оборудования ЭВМ.

Проведем расчет сопротивления изоляции.

Правила электробезопасности устанавливают нормы сопротивления изоляции и требования к ее диэлектрической прочности. Для электрических машин и аппаратов минимальное значение сопротивления изоляции определяется по формуле:

[МОм],



где U – напряжение, В;

N – мощность установки, кВт.

Отсюда следует, что при напряжении питания 220 В и мощности монитора 250 Вт сопротивление изоляции должно быть не менее чем:

.



Очень важным организационным мероприятием является также проведение обязательного и периодически повторяемого инструктажа по электро – и пожаробезопасности всех лиц, которые допускаются к работе на ЭВМ. При проведении периодически повторяемых противопожарных инструктажей необходимо обязательно добиваться, чтобы персонал практически умел пользоваться первичными средствами тушения пожара и средствами связи

Для тушения пожара должны применяться ручные огнетушители и переносные установки. Электросети и электроустановки, которые находятся под напряжением, тушить водой нельзя ни в коем случае, т. к. через струю воды может произойти поражение электрическим током. Именно поэтому для тушения пожара, который возник из-за неисправности электроприборов, применяют только пенные огнетушители.

Возможность быстрой ликвидации пожара во многом зависит от своевременного оповещения о пожаре. Обычно на предприятиях электронной промышленности весьма распространенным средством оповещения является телефонная связь.

Требования к освещению при работе с монитором ПК

Сохранность зрения человека, состояние его центральной нервной системы и безопасность на производстве в значительной мере зависят от условий освещения.

Производственное освещение должно удовлетворять следующим требованиям:

1. Освещенность должна соответствовать характеру труда, который определяется объектом различия, фоном, контрастом объекта с фоном.

2. Необходимо обеспечить достаточно равномерное распределение яркости на рабочей поверхности, а также в пределах окружающего пространства. Светлая окраска потолка, стен и производственного оборудования способствует созданию равномерного распределения яркости в поле зрения.

3. На рабочей поверхности должны отсутствовать резкие тени. Особенно вредны движущиеся тени, которые могут привести к травмам. Тени необходимо смягчать, применяя, например, светильники со светорассеивающими молочными стеклами. На окнах необходимо предусматривать солнцезащитные устройства (например, жалюзи).

4. В поле зрения должна отсутствовать блескость. Блескость – повышенная яркость светящихся поверхностей, вызывающая нарушение зрительных функций (ослепленность), т.е. ухудшение видимости объектов. Блескость снижают уменьшением яркости источника света или выбором рациональных углов светильника.

5. Величина освещенности должна быть постоянной во времени. Колебания освещенности, вызванные резким изменением напряжения в сети, приводят к значительному утомлению. Пульсация освещенности связана также с особенностями работы газоразрядной лампы. Снижение коэффициента пульсации с 55 до 5% (при трехфазном включении) приводит к повышению производительности труда на 15%.

6. Следует выбирать оптимальную направленность светового потока. Наибольшая видимость достигается при падении света под углом 60 градусов к его нормали, а наихудшая при нуле градусов.

7. Следует выбирать необходимый состав спектра освещения. Это существенно при работах, где требуется правильная цветопередача.

8. Все элементы осветительных установок должны быть достаточно долговечными, электро – и взрывобезопасными.

Обеспечение этого условия достигается применением зануления или заземления, ограничением напряжения для питания местных или переносных светильников до 42 вольт и ниже.

Анализируя условия работы программиста, получаем следующие требования к производственному освещению:

– наименьшая допустимая освещенность от общего освещения составляет 300 лк;

– при работе за компьютером желательно, чтобы освещенность рабочего места не превышала 2/3 нормальной освещенности помещения;

– экран дисплея не должен быть ориентирован в сторону источников света (окон, настольных ламп и т.п.);

при размещении рабочего места рядом с окном угол между экраном дисплея и плоскостью окна должен составлять не менее 90 градусов (для исключения бликов), прилегающую часть окна желательно зашторить;

– не следует располагать дисплей непосредственно под источником освещения или вплотную с ним;

– стена позади дисплея должна быть освещена примерно так же, как и его экран;

– яркость для блестящих поверхностей более 0.2 кв. м не должна превышать 500 кд/кв. м;

– показатель ослепленности не должен превышать 40 единиц;

– коэффициент пульсаций 10 – 20%.

Специфика работы за ЭВМ, состоит в том, что работать приходится с так называемым самосветящимся объектом.

Свечение со стороны экрана, а также частая смена заставок на экране при большой продолжительности трудовой деятельности может отрицательно воздействовать на зрение. Такой режим работы утомляет зрительные органы. Поэтому разработчику программного обеспечения следует учитывать этот фактор при проектировании программного обеспечения и его отладке за компьютером.

В данной главе рассмотрены вопросы безопасной работы с монитором ПК и даны практические рекомендации по созданию оптимальных условий труда при работе с монитором ПК.

Были рассмотрены природа электромагнитных излучений, их влияние на организм человека и способы защиты от них.

Рассмотрены вопросы электробезопасности и пожаробезопасности при работе с монитором. Проведен расчет требуемого сопротивления изоляции.

Приведены требования к освещению в производственном помещении в целом и при работе с монитором ПК.

**Заключение**

В представленной работе – «Программа установки защищенных сетевых соединений с использованием протокола ISAKMP», – были решены следующие задачи:

* Исследована структура протокола ISAKMP и уязвимость его сетевым атакам
* Разработана и проанализирована общая структура защиты информации при передаче по сети.
* Определено место разрабатываемой программы в этой системе и определены интерфейсы к ней.
* Разработана и написана программа, реализующая протокол ISAKM
* Произведены необходимые отладка и тестирование программы, методы проведения тестирования были приведены в соответствующем разделе работы.

В процессе исследования была рассмотрена структура протокола ISAKMP, рассмотрены состав и порядок посылки пакетов, и порядок проведения необходимых расчетов. Также рассмотрены основные типы сетевых атак, и то, как протокол противостоит им.

При описании процесса разработки программы были представлена многонитевая структура программы и обоснована необходимость и преимущества использование именно этой структуры. Были подробно объяснены процесс создания нити и передача информации между нитями.

**Литература**

1. Единая система программной документации ГОСТ 19.001–77, 19.002–80, 19.003–80, 19.101–77, 19.102–77, 19.505–79.
2. Зубов Н.Н., Пьянзин А.Я. Методические указания к дипломному проектированию по специальности «Программное обеспечение вычислительной техники и автоматизированных систем» /Под ред. В.Ф. Шаньгина; МИЭТ. М., 1990
3. «Security Architecture for the Internet Protocol» RFC2401 Ноябрь 1998 г.
4. «Internet Security Association and Key Management Protocol (ISAKMP)» RFC2408 Ноябрь 1998 г.
5. «The Internet Key Exchange (IKE)» RFC2409 Ноябрь 1998 г.
6. «The Internet IP Security Domain of Interpretation for ISAKMP» RFC2407 Ноябрь 1998 г.
7. Bruce Schneier «Applied Cryptography Second Edition: protocols, algorithms, and source code in C» 1996 г.
8. «Advanced Programming in the UNIX Environment» W. Richard Stevens 1994 г.
9. «UNIX System V Network Programming» Stephen A. Rago 1994 г.
10. «Programming with Threads» Steve Kleiman, Devang Shah, Bart Smaalders 1996 г.