SSH — SECURE SHELL

Предыстория

Анализ сетевого трафика — перехват паролей (rlogin, telnet, rsh, ftp).
В 1995 г. Tatu Ylönen (ун-т Хельсинки) разработал первую версию SSH.
Сначала — свободные реализации, впоследствии — проприетарные.
Обнаружены уязвимости.
В 2006 г. — вторая версия протокола SSH-2, стандарты IETF.

С 2005 г. наиболее популярная реализация (клиент и сервер) — OpenSSH (лицензия BSD). Также популярен SSH-клиент PuTTY (лицензия MIT X11).

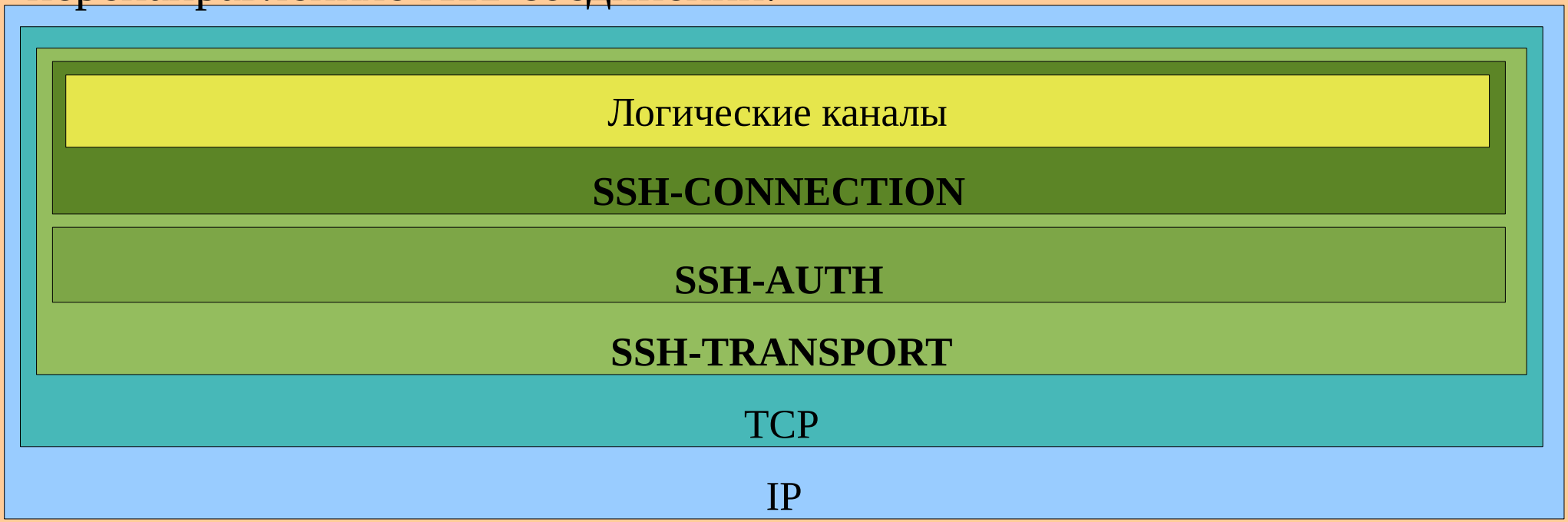
Аспекты безопасности:

* конфиденциальность данных (шифрование);
* целостность данных (имитовставка HMAC);
* man-in-the-middle (аутентификация сервера, инфраструктура ключей);
* denial of service — отказ в обслуживании (уязвим);
* скрытые каналы (уязвим);
* forward secrecy — предупреждающая безопасность (алг. Д-Х).

АРХИТЕКТУРА ПРОТОКОЛА SSH

RFC 4251 (январь 2006)

SSH-TRANPORT (RFC 4253): идентификация версий, согласование алгоритмов, аутентификация сервера.
SSH-AUTH (RFC 4252): аутентификация клиента.
SSH-CONNECTION (RFC 4254): логические каналы — интерактивные сеансы, удалённое вып. команд, перенаправление TCP-соединений, перенаправлениях X11-соединений.



SSH: ТРАНСПОРТНЫЙ ПРОТОКОЛ

RFC 4253 (январь 2006)

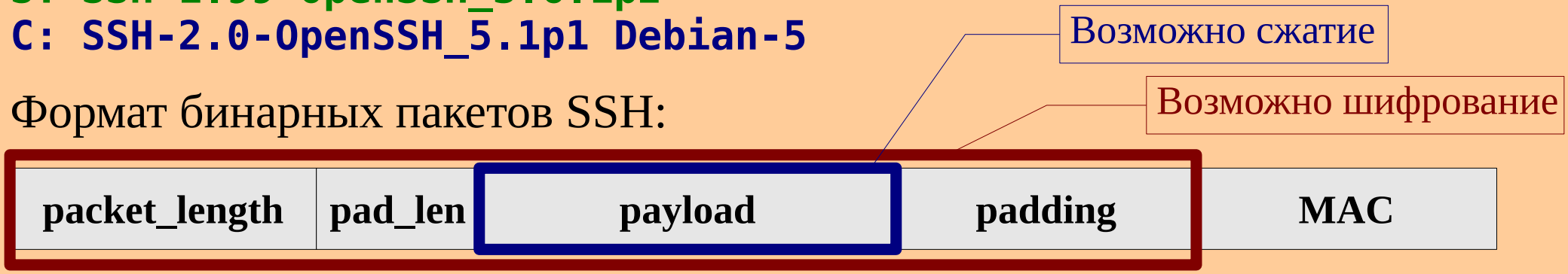
Сервер ожидает соединение на TCP-порт 22. Клиент инициирует соединение. После установления соединения происходит обмен идентификационными строками:

SSH-*protoversion-softwareversion* SP *comments* CR LF

Пример:

```
S: SSH-1.99-OpenSSH_3.6.1p2
C: SSH-2.0-OpenSSH_5.1p1 Debian-5
```

Формат бинарных пакетов SSH:



packet_length (uint32) — длина пакета (исключая packet_length и MAC)
padding_length (byte) — длина поля padding
payload (byte[n_1]) — контент SSH-сообщения (n_1=packet_length-padding_length-1)
padding (byte[n_2]) — байты выравнивания со случайным содержимым (n_2>4)
MAC (byte[m]) — Message Authentication Code, MAC = hmac(key, seqno || payload)

SSH: ТРАНСПОРТНЫЙ ПРОТОКОЛ

Каждое SSH-сообщение содержит байт типа сообщения и параметры, специфичные для данного типа.

* 1-19 — общие сообщения транспортного протокола;
* 20-29 — согласование алгоритмов;
* 30-49 — обмен ключами;

* 50-59 — общие сообщения протокола аутентификации;
* 60-79 — специфические сообщения протокола аутентификации;

* 80-89 — общие сообщения протокола соединения;
* 90-127 — управление логическими каналами;

* 128-191 — резерв;

* 192-255 — локальные расширения.

Рекомендуемая максимальная длина SSH-сообщения — 32768, всего пакета — 35000 байт.

SSH: ТРАНСПОРТНЫЙ ПРОТОКОЛ

Согласование алгоритмов:

* алгоритм выбора ключей сеанса
  diffie-hellman-group1-sha1, diffie-hellman-group14-sha1
* алгоритм проверки ключа сервера
  ssh-dss, ssh-rsa, pgp-sign-rsa, pgp-sign-dss
* алгоритм шифрования сеанса
  3des-cbc, blowfish-cbc, aes128-cbc, aes256-cbc, arcfour, ..., none
* алгоритм проверки целостности (MAC)
  hmac-sha1, hmac-sha1-96, hmac-md5, hmac-md5-96, none
* алгоритм сжатия
  none, zlib
* язык сообщений

SSH: ТРАНСПОРТНЫЙ ПРОТОКОЛ

Выбор ключей сеанса

C: SSH_MSG_KEXINIT(...) — согласование алгоритмов

S: SSH_MSG_KEXINIT(...) — согласование алгоритмов

C: генерирует *a*, вычисляет *A*
SSH_MSG_KEXDH_INIT(*A*)

S: генерирует *b*, вычисляет B, *K*, *H* = hash(Vc || Vs || Ic || Is || Ks^{pub} || A || B || *K*),
вычисляет ЭЦП от *H* на основе своего закрытого ключа *s* = sign(*H*,Ks^{pri})
SSH_MSG_KEXDH_REPLY(Ks^{pub}, *B*, *s*)

C: проверяет доверие ключу сервера Ks^{pub} (по локальной базе .known_hosts,
по записи DNS SSHFP (RFC 4255) или запросив у пользователя),
вычисляет *K* и *H*, проверяет аутентичность сервера на основе *s*
SSH_MSG_NEWKEYS() или SSH_MSG_DISCONNECT(*reason*, *descr*, *lang*)

S: SSH_MSG_NEWKEYS() или SSH_MSG_DISCONNECT(*reason*, *descr*, *lang*)

H — хэш обмена; первое значение *H* используется как session_id

K — общий секрет (перегенерируется через 1 час или 1 Гбайт)

key = hash(*K* || *H* || *X* || session_id), *X* — лат. буква «A» ... «F».

Если длины хэша не хватает для формирования ключа:

K2 = hash(*K* || *H* || *K1*); *K3* = hash(*K* || *H* || *K1* || *K2*); ...

key = *K1* || *K2* || *K3* || ...

SSH: ТРАНСПОРТНЫЙ ПРОТОКОЛ

После завершения фазы согласования ключей клиент запрашивает сервис *service_name*:

«ssh-userauth» — протокол аутентификации пользователя SSH,
«ssh-connection» — протокол соединения SSH

C: SSH_MSG_SERVICE_REQUEST(*service_name*)

S: SSH_MSG_SERVICE_ACCEPT(*service_name*) или
SSH_MSG_DISCONNECT(*reason, descr, lang*)

SSH: ПРОТОКОЛ АУТЕНТИФИКАЦИИ

RFC 4252 (январь 2006)

C: SSH_MSG_USERAUTH_REQUEST(*user_name, service_name, method, ...*) = UAR

S: SSH_MSG_USERAUTH_FAILURE(*methods_list, partial*) или
SSH_MSG_USERAUTH_SUCCESS() или
SSH_MSG_DISCONNECT(*reason, descr, lang*)

Методы аутентификации:

publickey — клиент высылает ЭЦП S_c=sign(session_id || UAR, Kc^{pri}), сервер проверяет доверие открытому ключу клиента Kc^{pub} по имеющейся на сервере копии ключа, затем проверяет аутентичность клиента по Sc.

password — клиент подтверждает свою аутентичность паролем.

hostbased — аналогично publickey, только используется пара ключей для клиентского хоста; подтвердив аутентичность хоста, сервер доверяет имени пользователя.

none — без аутентификации.

SSH: ПРОТОКОЛ СОЕДИНЕНИЯ

RFC 4254 (январь 2006)

Прокол соединения допускает мультиплексирование нескольких логических каналов в рамках одного соединения (SSH-сеанса). Любая сторона может создать канал. Каналы идентифицируются по номеру (независимая нумерация на каждой стороне).

→ SSH_MSG_CHANNEL_OPEN(type, s_no, ...)
← SSH_MSG_CHANNEL_OPEN_CONFIRMATION(r_no, s_no, ...) или
SSH_MSG_CHANNEL_OPEN_FAILURE(r_no, reason, descr, lang)
↔ SSH_MSG_CHANNEL_DATA(r_no, data)
↔ SSH_MSG_CHANNEL_EOF(r_no)
↔ SSH_MSG_CHANNEL_CLOSE(r_no)

Многие каналы имеют расширения, работающие по схеме запрос-ответ:

→ SSH_MSG_CHANNEL_REQUEST(r_no, request, want_reply, ...)
← SSH_MSG_CHANNEL_SUCCESS(r_no) или SSH_MSG_CHANNEL_FAILURE(r_no)

SSH: ПРОТОКОЛ СОЕДИНЕНИЯ

Интерактивные сеансы

Канал типа ***session*** предназначен для удалённого запуска приложений (шелл, программа, подсистема). Поддерживаются следующие запросы:

«pty-req», «env», «shell», «exec», «subsystem», «window-change», «xon-xoff», «signal», «exit-status», «exit-signal»

Перенаправление X11 (X11 Forwarding)

Запрос «x11-req» в рамках канала интерактивного сеанса (***session***):на удалённой стороне генерируется фиктивный код mit-magic-cookie для прокси-X-сервера, открывается TCP-порт для приема соединений на прокси-X-сервер. При подключения удалённого X-клиента к прокси-X-серверу создаётся новый канал типа ***x11***. В X11-запросах фиктивный код заменяется кодом локального X-сервера.

SSH: ПРОТОКОЛ СОЕДИНЕНИЯ

Перенаправление TCP-портов (TCP Forwarding)

Запрос перенаправления на удалённой стороне:

→ SSH_MSG_GLOBAL_REQUEST("tcpip-forward", *want_reply*, *addr*, *port*)

← SSH_MSG_REQUEST_SUCCESS(*port*)

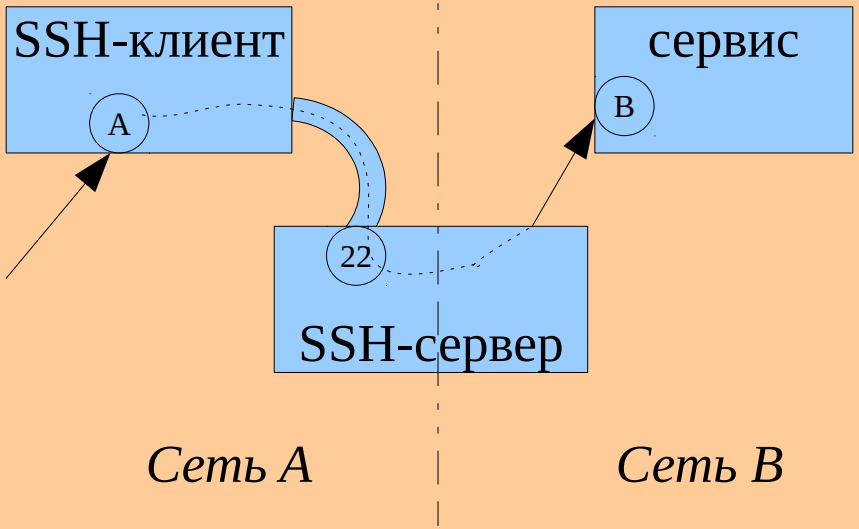
→ SSH_MSG_GLOBAL_REQUEST("cancel-tcpip-forward", *want_reply*, *addr*, *port*)

Каналы:

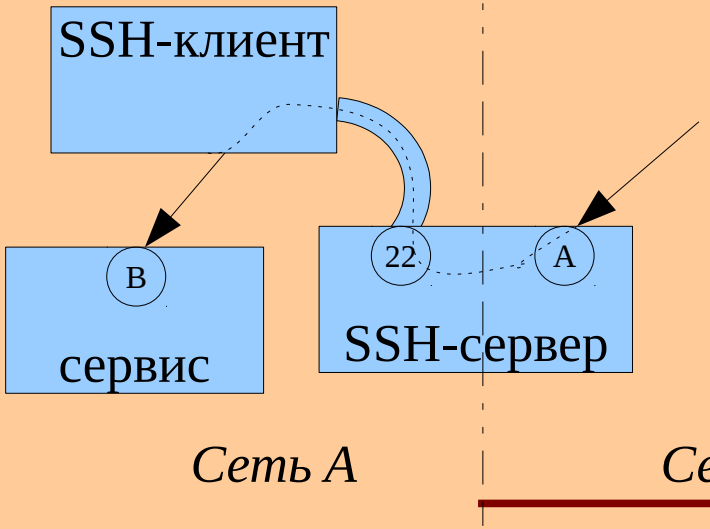
forwarded-tcpip — подключение к порту удалённой стороны

direct-tcpip — подключение к порту локальной стороны

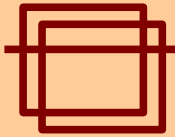
Local Forwarding (direct-tcpip)



Remote Forwarding (forwarded-tcpip)



SSH КАК ТРАНСПОРТ (ТУННЕЛЬ)



* SFTP (реализовано как «subsystem») → SSHFS
* SCP (реализовано как «exec» для rcp)
* FISH (реализовано как «shell»)
* SVN+SSH
* VPN (только в OpenSSH)

Существует библиотека libssh, предоставляющая API для использования SSH в качестве универсального транспорта.