1. Загальна характеристика протоколу SSL

Протокол SSL (Secure Socket Layer) виконує функції із створення захищеного каналу між двома абонентами мережі, що включають їх взаємну автентифікацію, забезпечення конфіденційності, цілісності і автентичності даних, що передаються. Працює на сеансовому рівні моделі OSI. Працює поверх протоколу TCP, не може працювати поверх UDP.

Взаємна автентифікація сторін виконується за допомогою обміну сертифікатами відкритих ключів клієнта та сервера. Протокол SSL підтримує сертифікати, що відповідають стандарту X.509, а також стандарти інфраструктури відкритих ключів PKI (Public Key Infrastructure), за допомогою якої організовується видача і перевірка сертифікатів.

Конфіденційність забезпечується шляхом шифрування повідомлень з використанням симетричних сесійних ключів, якими сторони обмінюються при встановленні з'єднання. Сесійні ключі передаються також в зашифрованому вигляді, при цьому вони шифруються за допомогою відкритих ключів, отриманих із сертифікатів абонентів. Достовірність і цілісність інформації забезпечується за рахунок формування і перевірки електронного цифрового підпису. Для цифрових підписів та шифрування сеансових ключів використовуються алгоритми з відкритим ключем.

В якості алгоритмів асиметричного шифрування використовуються алгоритми RSA та Діффі-Хеллмана. Допустимими алгоритмами симетричного шифрування є RC2, RC4, DES, 3-DES, AES. Для обчислення хеш-функцій можуть застосовуватися стандарти MD5 та SHA-1.

1.1Підпротоколи SSL

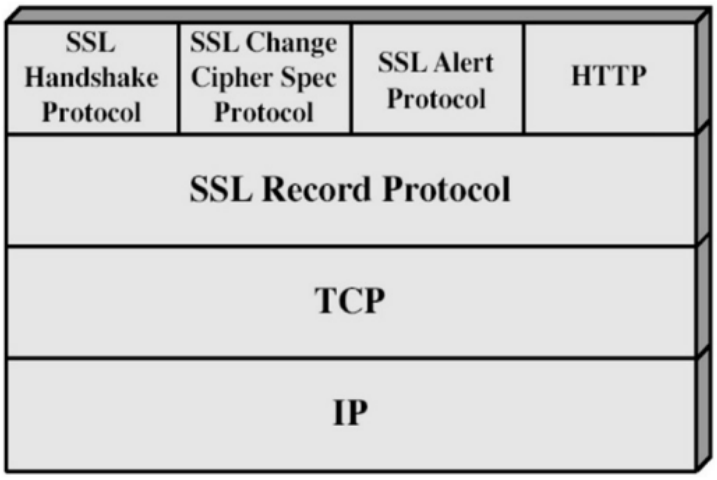

Рис.1: Стек протоколів SSL

1) Протокол Records

Базовий протокол, що використовується всіма іншими протоколами SSL. Виконує наступні функції: забезпечення конфіденційності (за допомогою шифрування) та забезпечення цілісності даних (з використанням MAC – Message Authentication Code). Виконує наступні дії:

- Розбиває повідомлення, що передається, на блоки довжиною не більше 2^{14} байт;
- Ці блоки можуть стискатися (необов'язково), без втрат, що збільшує довжину блоку не більш ніж на 1024 байти;
- Обчислюється MAC стиснутого блоку за допомогою спільного секретного ключа і додається в кінець стиснутого блоку;
- Результат шифрується з використанням симетричного алгоритму. Шифрування збільшує довжину блоку не більш ніж на 1024 байти, тому сумарна довжина блоку становить не більше ніж $2^{14} + 2048$ байти;
- До кожного блоку додається заголовок, що містить: тип повідомлення (8 біт, протокол вищого рівня, що породив

повідомлення), версію протоколу SSL (16 біт), довжину стиснутого блоку в байтах (16 біт).

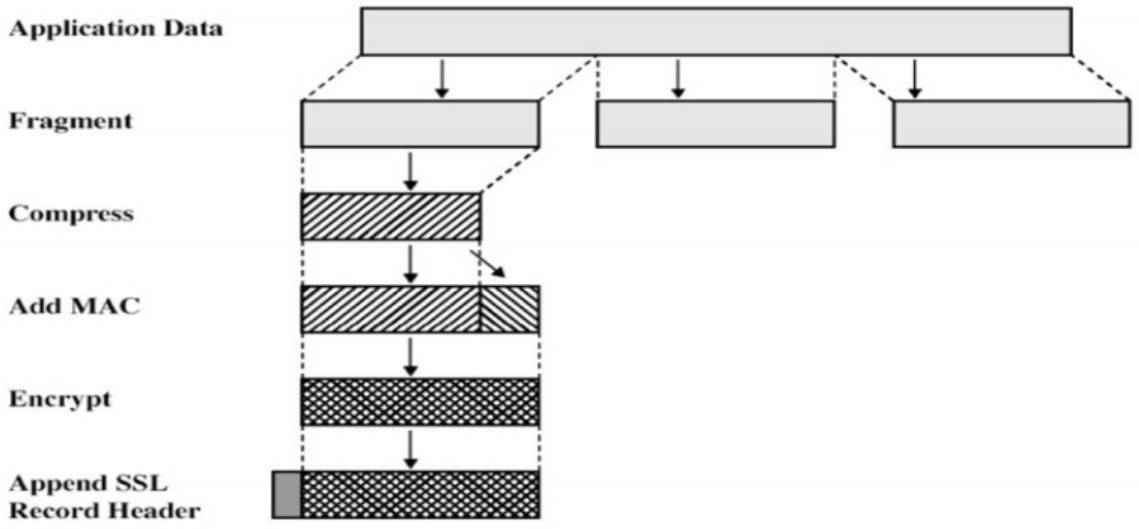

Рис. 2: Операції протоколу Records

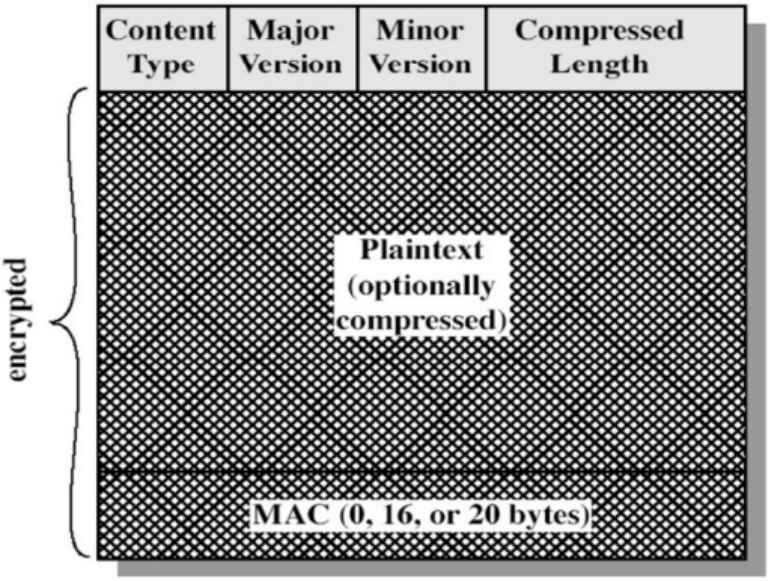

Рис. 3: Формат блоку в протоколі Records

2) Протокол Change Cipher Spec

Складається із одного повідомлення довжиною 1 байт і значенням 1. Сповіщає Records про те, що можна починати шифрування повідомлень із використанням обумовлених параметрів. Під час відновлення попередньої

сесії повідомлення change cipher spec відправляється після hello повідомлень, під час першої сесії – після обміну ключами.

3) Протокол Alert

Формує повідомлення про будь-які помилки та зміни стану в протоколі SSL. Складається із двох байтів. Перший байт приймає два значення: 1(warning) та 2 (fatal). Якщо рівень помилки fatal, то SSL негайно припиняє з'єднання. Другий байт містить код, що позначає тип помилки.

4) Протокол Handshake

Найскладніша частина протоколу SSL, що дозволяє клієнту та серверу автентифікувати одне одного та вибрати алгоритми шифрування і MAC та ключі. Цей протокол застосовується до того як application data починають пересилатися по створеному захищеному каналу. Протокол складається із серії обмінів повідомленнями між сервером та клієнтом. Кожне повідомлення має три поля: тип повідомлення (1 байт), довжина повідомлення в байтах (3 байти), контент повідомлення.

Message Type	Parameters
hello_request	null
client_hello	version, random, session id, cipher suite, compression method
server_hello	version, random, session id, cipher suite, compression method
certificate	chain of X.509v3 certificates
server_key_exchange	parameters, signature
certificate_request	type, authorities
server_done	null
certificate_verify	signature
client_key_exchange	parameters, signature
finished	hash value

Рис. 4: Типи повідомлень в протоколі Handshake

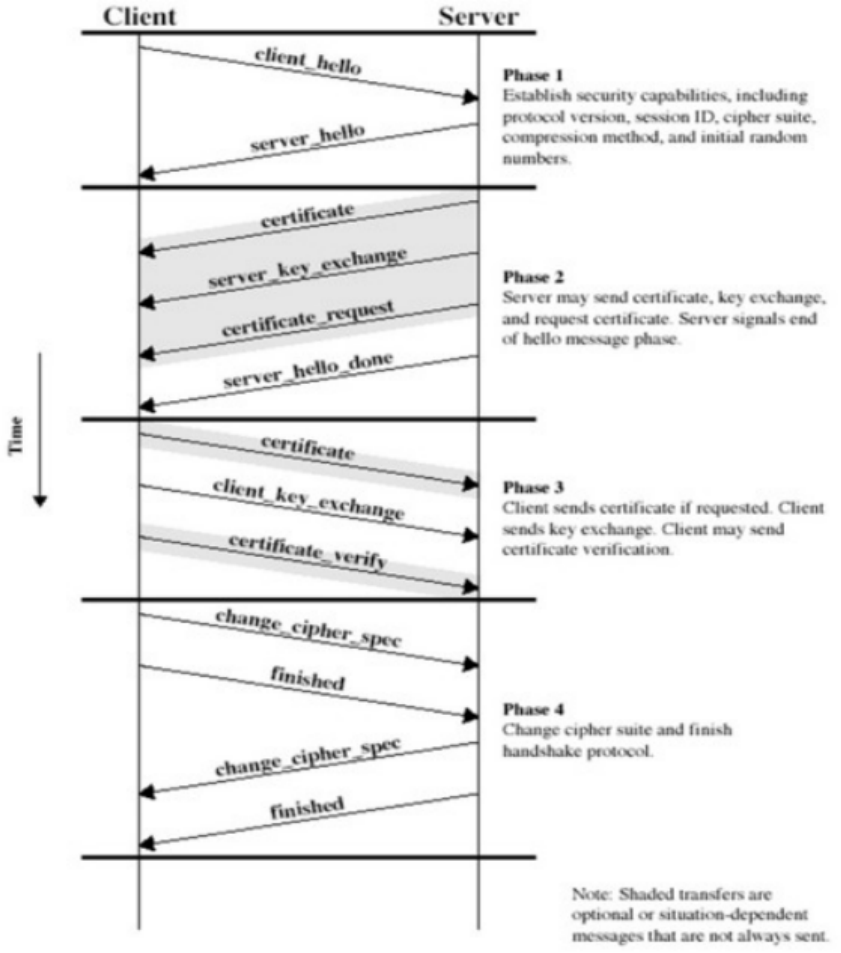

Рис. 5: Схема роботи протоколу Handshake

Hello request: може бути послано сервером у будь-який час, але буде ігноруватися клієнтом, якщо протокол handshake вже почав роботу. Це просте сповіщення про те, що клієнт може починати новий процес обговорення шляхом посилання повідомлення client hello коли йому буде зручно.

Client hello: посилається клієнтом, коли він хоче підключитися до сервера. Містить найвищу версію SSL, яка підтримується клієнтом; випадкове значення, що складається із мітки часу та 28 байтів, згенерованих криптографічно безпечним генератором псевдовипадкових послідовностей, що використовуються далі у протоколі для створення ключів; ідентифікатор

сесії; список алгоритмів шифрування, хешування та обміну ключами, які готовий використовувати клієнт (2 8-бітних числа).

Server hello: посилається у відповідь на client hello. Містить версію SSL клієнта, якщо вона підтримується сервером, або ж найвищу версію SSL, яку підтримує сервер; випадкове значення, що складається із мітки часу та 28 байтів, згенерованих криптографічно безпечним генератором псевдовипадкових послідовностей; ідентифікатор сесії (необов'язково); список алгоритмів, вибраних сервером із списку, наданого клієнтом.

Server certificate: якщо сервер повинен бути автентифікований (що відбувається у більшості випадків), то він надсилає послідовність сертифікатів (першим свій сертифікат, останнім сертифікат CA), зазвичай у форматі X.509.

Server key exchange: посилається сервером, якщо у нього немає сертифікату з ключем для шифрування.

Certificate request: посилається сервером у випадку, коли він хоче, щоб клієнт ідентифікував себе (необов'язково). Містить список допустимих CA.

Server hello done: посилається сервером, щоб повідомити клієнта про завершення server hello та пов'язаних повідомлень. Після посилання цього повідомлення сервер буде чекати на відповідь клієнта. Після отримання цього повідомлення клієнт повинен перевірити, що сервер надав коректний сертифікат і що параметри server hello є прийнятними.

Client certificate: посилається лише якщо сервер запросив сертифікат клієнта.

Client key exchange: якщо використовується RSA, то містить pre-master secret, зашифрований з використанням відкритого ключа сервера (закодований за допомогою PKCS #1); pre-master secret складається із максимальної версії SSL, що підтримується клієнтом, і 48 байтів, згенерованих криптографічним генератором псевдовипадкових послідовностей; у випадку використання алгоритму Діффі-Хеллмана містить

публічне значення алгоритму Діффі-Хеллмана, що дозволяє клієнту і серверу обчислити однаковий pre-master secret.

Certificate verify: відправляється лише якщо був відправлений сертифікат клієнта. Використовується для явної перевірки коректності сертифікату клієнта. Клієнт підписує усі повідомлення, передані у ході протоколу handshake, і відправляє серверу.

Finished: відправляється обома сторонами одразу після повідомлення change cipher spec, щоб запевнитися, що обмін ключами і автентифікація пройшли успішно; це перше повідомлення, зашифроване за допомогою щойно обумовлених алгоритмів і ключів.

1.2 Відмінності між різними версіями протоколу SSL

1) Відмінності SSL 3.0 від SSL 2.0:

- розділення транспортування даних та шару повідомлень;
- використання всіх 128 біт матеріалу для ключів, навіть при використанні шифру Export;
- здатність клієнта та сервера відправляти ланцюжки сертифікатів, що дозволяє організаціям використовувати ієрархію сертифікатів глибиною більше, ніж 2 сертифікати;
- реалізація узагальненого протоколу обміну ключами, що дозволяє обмін ключами з використанням алгоритмів Діффі-Хеллмана та Fortezza, а також використання інших сертифікатів, крім RSA-сертифікатів;
- можливість стиснення та декомпресії даних в протоколі Records;
- можливість використання SSL 2.0, якщо зустрічається клієнт, що не підтримує SSL 3.0.

2) Відмінності між TLS 1.0 та SSL 3.0:

- різні функції утворення ключів;

- різні MAC;
- різні повідомлення Finished;
- у TLS більше повідомлень alert;
- для TLS вимагається підтримка DSS/DH.

3) Зміни у TLS 1.1. в порівнянні з TLS 1.0:

- неявний вектор ініціалізації (Initialization vector, IV) замінений на явний, щоб уникнути атак зв'язування блоків (Cipher block chaining);
- обробка padded errors використовує bad_record_mac alert повідомлення замість decryption_failed для захисту від CBC атак;
- для параметрів протоколу визначені реєстри IANA (Internet Assigned Numbers Authority);
- сесія може бути відновлена, навіть якщо вона була закрита передчасно.

4) Зміни у протоколі TLS 1.2:

- комбінація MD5/SHA-1 у псевдовипадковій функції замінена на використання псевдовипадкової функції в залежності від набору функцій шифрування;
- комбінація MD5/SHA-1 в підписаному елементі замінена на єдиний хеш; підписані елементи тепер містять поле, що явно вказує алгоритм хешування, який використовується;
- після certificate_request, у випадку, якщо сертифікату немає, клієнт повинен посилати пустий список сертифікатів;
- TLS_RSA_WITH_AES_128_CBC_SHA став обов'язковим для реалізації;
- додавання HMAC-SHA256;
- видалення IDEA та DES як застарілих.

2. Атаки на SSL

2.1 BEAST

Атака на основі вибраного відкритого тексту. Використовує той факт, що при використанні режиму зчеплення блоків у кожному наступному блоці в якості вектора ініціалізації використовується шифртекст із попереднього блоку.

Нехай зловмисник хоче перевірити, що повідомлення m_i дорівнює x. Він обирає наступний блок відкритого тексту як $m_j = m_i \oplus c_{(i-1)} \oplus c_{(j-1)}$. Тоді $c_j = E\left(k, c_{j-1} \oplus m_j\right) = E\left(k, c_{j-1} \oplus c_{j-1} \oplus m_i\right) \oplus c_{(i-1)} = E\left(k, m_i \oplus c_{(i-1)}\right)$ Якщо $c_j = c_i$, то x вгадано правильно.

Вгадування усіх 16 байтів випадкового кукі сесії є непрактичним, але атака стає набагато простішою, якщо у зловмисника є можливість керувати межами блоків, щоб можна було вгадувати по одному байту за раз.

2.2 CRIME (Compression Ratio Info-leak Made Easy)

Це брутфорс атака, що використовує лише інформацію про зміну довжини стиснених повідомлень в протоколі TLS.

Нехай HTTP запит від клієнта має наступний вигляд:

```
POST / HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:14.0) Gecko/20100101 Firefox/14.0.1
Cookie: secretcookie=7xc89f94wa96fd7cb4cb0031ba249ca2
Accept-Language: en-US,en;q=0.8

(... body of the request ...)
```

Розмір контенту дорівнює length(encrypt(compress(header + body)). Зловмисник знає цю дожину, а також те, що клієнт буде передавати «Cookie: secretcookie=», і хоче дізнатися це секретне значення. Зловмисник створює запит, що містить «Cookie: secretcookie=0» і виглядає наступним чином:

```
POST /secretcookie=0 HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:14.0) Gecko/20100101 Firefox/14.0.1
Cookie: secretcookie=7xc89f94wa96fd7cb4cb0031ba249ca2
Accept-Language: en-US,en;q=0.8

( ... body of the request ...)
```

Через надлишковість, пов’язану із додатковим «secretcookie», довжина другого запиту після стиснення буде меншою, ніж якби зловмисник додав до запиту дані, які не співпадають із жодним рядком у запиті клієнта. Ідея полягає в тому, щоб змінювати дані, які додає до запиту зловмисник, і порівнювати довжини стиснених даних. Далі зловмисник додає до запиту «secretcookie=1», потім «secretcookie=2» і т.д. Коли зловмисник додасть до запиту «secretcookie=7», то запит стиснеться краще, що означатиме, що секретне значення починається на «7». Потім зловмисник буде додавати до запиту значення «secretcookie=70», «secretcookie=71», і так далі.

2.3 LUCKY 13

Перед зашифруванням блок із повідомленням виглядає наступним чином:

```
|  data  | MAC of plaintext | padding |
```

Після розшифрування спочатку перевіряється padding, і якщо він коректний, то лише тоді перевіряється MAC, інакше сервер повертає помилку про те, що padding або MAC були некоректними. Атака padding oracle використовує розшифрування в режимі зчеплення блоків шляхом зміни шифртексту в попередньому блоці. Зловмисник може модифікувати зашифроване повідомлення на основі повідомлень про помилки і після повторюваних запитів отримати повідомлення, розшифроване сервером без ключа зашифрування.

TLS зазвичай використовує HMAC разом із MD5, SHA1 або SHA256 в якості хеш-функції. Кожна з цих хеш-функцій обробляє 64-байтні блоки повідомлень. Хеш-фунція включає 8-байтне поле і padding, що означає, що один блок може складатися не більш ніж з 55 байтів реальних даних (що також включає 13-байтний хедер record).

```
| Sequence | Header | App Data |
| 8 bytes  | 5 bytes|  n bytes |
```

Зашифровані дані:

Enc App Data	Enc MAC Tag	Enc Padding
n bytes	20-bytes	p-bytes

Нехай в пакеті 85 байт і сервер інтерпретував пакет наступним чином:

Header	IV	Enc App Data	Enc MAC Tag	Enc Padding
5	16	44 – p	20	p

Сервер розшифровує:

Enc App Data	Enc MAC Tag	Enc Padding
44 – p	20	p

Автентифікація проводиться на:

Sequence	Header	App Data
8	5	44 – p

Якщо немає валідних padding байтів, тоді p = 0 байтів, а app data – 44 байти. Можна розглядати наступні варіанти:

- padding невірний і 57 байтів – це MAC;
- останній байт відкритого тексту 0×00 і 56 байтів – це MAC;
- останні 2 байти відкритого тексту 0×01, 0×01 і 55 байтів – це MAC.

Атака випробовує різні значення в блоках шифртексту з другого по останній, цього разу в останніх двох байтах, намагаючись зробити останній блок відкритого тексту коректними байтами padding. Зловмиснику потрібно, щоб було аутентифіковано 55 байтів за допомогою HMAC-SHA1, тому що це можна обчислювально відрізними від 56 і 57 байтів, тому що якщо байтів буде більше за 55, хеш-функції доведеться виконати ще один цикл, що створить маленьку затримку.

2.4 SSL strip

Ця атака може відбутися, якщо браузерна сесія починається з використанням HTTP і переключається на HTTPS за допомогою посилання чи перенаправлення. Це часто використовується, наприклад, для переходу на безпечну сторінку логіну із небезпечної домашньої сторінки.

Зловмисник діє за схемою Man in the middle. Щоб знешкодити протокол безпеки, зловмисник надсилає HTTP 301 – тимчасове перенаправлення запитів, і замінює будь-яку появу https:// на http://. Це

змушує клієнта перейти на сторінку перенаправлення і спілкуватися без шифрування і автентифікації (у випадку, якщо атака успішна, і клієнт не помічає, що його обманули). Вкінці зловмисник відкриває нову сесію із сервером і отримує будь-які дані клієнта чи сервера.

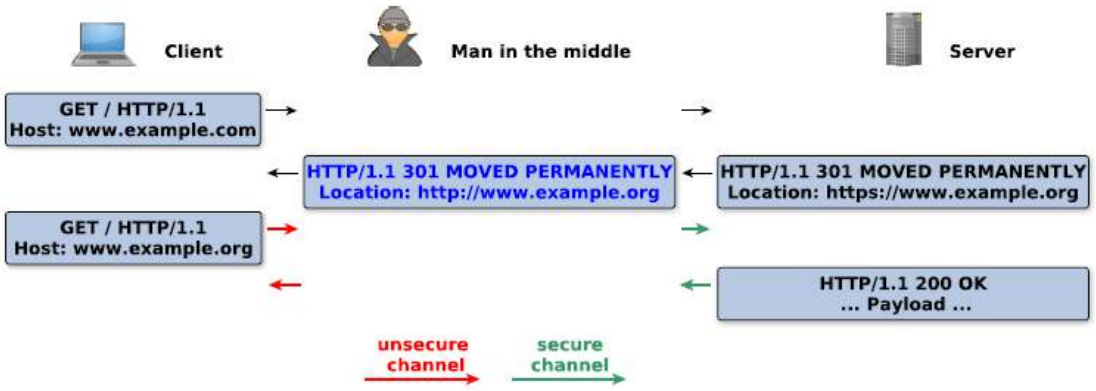

Рис. 6: Атака SSL strip

3. Структура сертифікату X.509

- TBS (to be signed, дані, які підписуються)
  - Версія (3)
  - Серійний номер
  - Ідентифікатор алгоритму
  - Видавець (RDN)
  - Дійсний
    - Дата початку
    - Дата кінця
  - Власник (RDN)
  - Інформація про відкритий ключ власника
    - Алгоритм
    - Ключ
  - Унікальний ідентифікатор видавця (не обов’язково)
  - Унікальний ідентифікатор власника (не обов’язково)
  - Розширення (не обов’язково)
    - ...
- Алгоритм підпису власника (OID)
- Цифровий підпис видавця.

Приклад OID: 1.2.840.113549.1.1.5

Пояснення:

1 - ISO

2 - Member body

840 - USA

113549 - RSA Labs

1 - PKCS

1 - PKCS #1

5 - RSA signature with SHA-1 hash