МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Методи реалізації криптографічних механізмів

Лабораторна робота №4

“Реалізація для серверу в системі клієнт-сервер (серверна частина).”

Виконали:

Студент ФІ-22мн
Підгрупа 1С

Бондаренко Андрій
Гузей Дмитро
Яценко Артем

Перевірила:

Байденко П.В.

Київ-2023

Завдання

Дослідити основні задачі, що виникають при програмній реалізації криптосистем. Запропонувати методи вирішення задачі контролю доступу до ключової інформації, що зберігається в оперативній пам'яті ЕОМ для різних (обраних) операційних систем. Запропонувати методи вирішення задачі контролю правильності функціонування програми криптографічної обробки інформації. Порівняти зточки зору вирішення цих задач інтерфейси Crypto API, PKCS 11.

***Підгрупа 1С**. Реалізація для серверу в системі клієнт-сервер (серверна частина).*

Crypto APIs

Будова і можливості Crypto API

Насамперед постараємося описати коло завдань, на вирішення яких орієнтований Crypto API:

- надійне приховування даних;
- можливість передання прихованих даних третім особам;
- надійна система перевірки достовірності інформації, що надійшла від третіх осіб;
- розшифровування отриманих конфіденційних даних;
- робота з "ідентифікаційними посвідченнями" третіх осіб;
- забезпечення роботи з визнаними криптографічними стандартами;
- можливість розширення і роботи з поки що невідомими алгоритмами.

Спершу я постараюся розглянути вирішення завдання розширення. У Crypto API це завдання вирішили досить елегантно. Реалізація всіх алгоритмів (шифрування, цифрового підпису тощо) повністю виведена зі складу самого Crypto API і реалізується в окремих, незалежних динамічних бібліотеках - "криптопровайдерах" (Cryptographic Service Provider - CSP). Сам же Crypto API просто висуває певні вимоги до набору функцій (інтерфейсу) криптопровайдера і надає кінцевому користувачеві уніфікований інтерфейс роботи з CSP. Кінцевому користувачеві для повноцінного використання всіх функцій криптопровайдера достатньо знати його строкове ім'я і номер типу.

Крім завдання розширення, одним з основних завдань Crypto API є можливість однозначної ідентифікації сторони, що передає/приймає, в протоколі передачі даних. Загальновизнаним рішенням у цьому питанні є використання механізму сертифікатів. Сертифікати ніби стали "цифровими паспортами", що несуть інформацію про своїх власників. Повна розповідь про цей механізм буде дещо пізніше, а зараз варто згадати, що Crypto API також повно реалізує весь спектр

функцій роботи з ним. Більшість функцій Crypto API, що працюють з переданими даними, так чи інакше, використовують сертифікати у своїй роботі.

У програмних рішеннях рано чи пізно постає питання стандартизації переданих між додатками даних. У сфері криптографії для вирішення цього питання вже давно й успішно застосовують набір стандартів "PKCS", запропонований компанією RSA Security. У цьому комплекті стандартів враховуються всі можливі випадки, що виникають у криптографічних додатках. Передбачено стандарти для обміну сертифікатами, зашифрованими і підписаними даними та багато іншого. Crypto API, як основна бібліотека для забезпечення роботи з криптографічними даними в Windows, також досить повно підтримує цей комплект стандартів і дає змогу формувати криптографічні додатки, які можна опрацювати надалі будь-якими програмними продуктами.

Таким чином, ми можемо розділити весь інтерфейс Crypto API на 5 функціональних груп:

1. базові криптографічні функції:
    - функції шифрування/розшифрування даних;
    - функції хешування та отримання цифрового підпису даних;
    - функції ініціалізації криптопровайдера і роботи з отриманим контекстом;
    - функції генерації ключів;
    - функції обміну ключами.
2. Функції кодування/декодування. Під кодуванням у цьому випадку мається на увазі отримання на виході інформації, кодованої у форматі ASN.1 (Abstract Syntax Notation One).
3. Функції роботи з сертифікатами.
4. Високорівневі функції обробки криптографічних повідомлень.
5. Низькорівневі функції обробки криптографічних повідомлень.

Криптопровайдери

Криптопровайдером називають незалежний модуль, що забезпечує безпосередню роботу з криптографічними алгоритмами. Кожен криптопровайдер має забезпечувати:

- реалізацію стандартного інтерфейсу криптопровайдера;
- роботу з ключами шифрування, призначеними для забезпечення роботи алгоритмів, специфічних для даного криптопровайдера;
- неможливість втручання третіх осіб у схему роботи алгоритмів.

Як уже зазначалося вище, самі криптопровайдери реалізуються у вигляді динамічно завантажуваних бібліотек (DLL). Таким чином, досить важко вплинути на перебіг алгоритму, реалізованого в криптопровайдері, оскільки компоненти криптосистеми Windows (усі) повинні мати цифровий підпис (тобто підписується і DLL криптопровайдера). У криптопровайдерів мають бути відсутні можливості зміни алгоритму через установку його параметрів. У такий спосіб розв'язується завдання забезпечення цілісності алгоритмів криптопровайдера. Завдання забезпечення цілісності ключів шифрування вирішується з використанням контейнера ключів, про який розповідається нижче.

Функції роботи з криптопровайдерами можна розділити на такі групи:

- функції ініціалізації контексту та отримання параметрів криптопровайдера;
- функції генерації ключів і роботи з ними;
- функції шифрування/розшифрування даних;
- Функції хешування та отримання цифрового підпису даних.

До групи функцій ініціалізації контексту входять такі функції:

- CryptAcquireContext. За допомогою цієї функції насамперед проводиться ініціалізація контексту криптопровайдера (отримання посилання на HANDLE, яке надалі можна використовувати в інших функціях). Також за допомогою останнього параметра цієї функції можна створити або видалити контейнер ключів.
- CryptContextAddRef. Ця функція служить для збільшення внутрішнього лічильника посилань криптопровайдера. Цю функцію рекомендується використовувати під час передачі контексту криптопровайдера як члена різних структур, що передаються функціям.
- CryptReleaseContext. Ця функція призначена для звільнення контексту криптопровайдера, отриманого за допомогою функції CryptAcquireContext. Фактично, проводиться тільки зменшення внутрішнього лічильника посилань

криптопровайдера (щось на зразок механізму підрахунку посилань у COM-об'єкта). Коли внутрішній лічильник посилань стає рівним нулю, цей контекст криптопровайдера повністю звільняється і не може бути більше ніде використаний.

- CryptGetProvParam. За допомогою цієї функції можна отримати значення різних параметрів криптопровайдера. Потрібно сказати, що для всіх криптопровайдерів стандартом визначено лише обмежений набір параметрів. Набір параметрів криптопровайдера може сильно варіюватися залежно від реалізації криптопровайдера.

До групи генерації та роботи з ключами входять такі функції:

- CryptGenKey. Ця функція призначена для генерації сесійного ключа (ключ, який використовується тільки протягом поточної сесії роботи), а також для генерації пар ключів для обміну (публічний і закритий ключ) і цифрового підпису.
- CryptDuplicateKey. Функція призначена для копіювання ключа.
- CryptGetUserKey. Функція призначена для отримання значення публічного ключа для зазначеного контейнера ключів. Використовується для отримання значень публічних ключів, призначених для обміну ключами та цифрового підпису.
- CryptDestroyKey. Функція призначена для звільнення раніше отриманого хендла ключа. Функцію слід викликати завжди для запобігання витокам пам'яті в додатку.
- CryptGetKeyParam. Функція призначена для отримання різних параметрів ключа. Як параметри використовуються алгоритм ключа (його цифрове позначення в системі), прапори дозволу використання ключа (наприклад, якщо відсутній встановлений прапор ключа CRYPY_ENCYPT, даним ключем неможливо буде зашифрувати дані), дані про довжину блоку ключа і багато іншого. У практичних реалізаціях цій функції слід приділяти чималу увагу, оскільки найчастіше саме від параметрів ключа залежить робота використовуваного алгоритму криптопровайдера.

- CryptSetKeyParam. Функція, зворотна до попередньої. Використовується для встановлення параметрів ключа.
- CryptDeriveKey. Функція призначена для генерації сесійного ключа на основі хешу даних. Тобто ця функція генерує один і той самий сесійний ключ, якщо їй передаються однакові значення хешу даних. Функція корисна в разі генерації сесійного ключа на основі пароля.
- CryptGenRandom. Функція використовується для заповнення переданого їй буфера випадковими даними. Використовується, наприклад, для генерації нового імені контейнера ключів.
- CryptExportKey. Функція експорту ключа для його передавання каналами інформації. Можливі різні варіанти передавання ключа, включно з передаванням публічного ключа пари ключів, а також передаванням секретного або сесійного ключа.
- CryptImportKey. Функція, зворотна до попередньої. Призначена для отримання з каналів інформації значення ключа.

До групи функцій шифрування/розшифрування даних входять:

- CryptEncrypt. Основна базова функція шифрування даних. Як параметри використовує раніше отримані контексти криптопровайдера і сесійного ключа. Дані, що генеруються на виході цієї функції, не є форматованими і не містять жодної іншої інформації, крім шифрованого контенту (на відміну від, скажімо, стандарту PKCS #7, який використовує спільне передавання кодованих даних і експортованого сесійного ключа).
- CryptDecrypt. Основна базова функція розшифровування даних. Як параметри використовуються раніше отримані контекст криптопровайдера і хендл сесійного ключа.

До групи функцій хешування та отримання цифрового підпису входять:

- CryptCreateHash. Функція створює хеш-об'єкт, призначений для генерації хеш-значення даних. В якості основних параметрів приймає раніше отримані контекст криптопровайдера і алгоритм формування хеша даних.

- CryptHashData. Основна функція хешування даних. Найважливішим параметром цієї функції є посилання на хешовані дані.
- CryptGetHashParam. Функція використовується в основному для отримання значення сформованого хешу даних. Функція CryptGetHashParam завершує процедуру створення хеш-значення, і подальші виклики функції CryptHashData повертатимуть помилку.
- CryptSetHashParam. Функція використовується для встановлення параметрів хешу. Може бути використана, скажімо, для зміни алгоритму формування хеша.
- CryptDestroyHash. Функція використовується для звільнення хеш-об'єкта.
- CryptDuplicateHash. Функція отримання копії хеша. Використовується при передачі хеша між функціями.
- CryptSignHash. Основна базова функція отримання цифрового підпису даних. Потрібно зазначити, що в Crypto API для зменшення довжини цифрового підпису (і прискорення роботи асиметричних алгоритмів, що використовуються для формування цифрового підпису) як вхідні дані використовують хеш. Функцій, що використовують для отримання цифрового підпису самі дані, у Crypto API немає.
- CryptVerifySignature. Основна базова функція перевірки цифрового підпису. Як вхідні дані знову-таки використовується значення хеша.

Контейнери ключів

Контейнером ключів називають частину бази даних ключів, яка містить пару ключів для обміну ключами та формування цифрового підпису. Як контейнери ключів (сховища пар ключів) використовують, наприклад, область тимчасової пам'яті, ділянку реєстру, файл на диску, смарт-картки. Контейнери ключів у системі можуть бути двох типів: користувацькі та рівня системи. Користувацькі контейнери існують у контексті роботи поточного користувача. За замовчуванням доступу до них більше ніхто не має (щоправда, існують можливості видачі доступу до контейнерів для інших користувачів). Контейнери ключів рівня системи використовуються, здебільшого, не в користувацьких програмах, а, наприклад, у сервісах (доступ до контейнерів ключів рівня системи можливий без інтерактивної ідентифікації користувача в системі). Контейнери ключів не існують самі по собі, а існують тільки в контексті криптопровайдера. Для кожного криптопровайдера існує свій власний

набір контейнерів ключів. Це пояснюється тим, що різні криптопровайдери можуть по-різному реалізовувати навіть один і той самий математичний алгоритм. Отже, і способи зберігання ключів можуть також сильно варіюватися від криптопровайдера до криптопровайдера.

Основними операціями з контейнерами ключів можна вважати:

- Створення нового контейнера ключів. Виконується за допомогою функції CryptAcquireContext, описаної раніше в розділі про криптопровайдери. Для створення нового контейнера ключів в якості останнього параметра даної функції передається значення CRYPT_NEWKEYSET. Необхідно також зауважити, що початково в новому контейнері ключів ніяких даних не міститься, і пари ключів необхідно генерувати самостійно за допомогою виклику функції CryptGenKey, або імпортувати.
- Видалення наявного контейнера ключів. Виконується за допомогою виклику функції CryptAcquireContext з останнім параметром, встановленим у CRYPT_DELETEKEYSET.
- Генерація нової пари ключів. Виконується за допомогою виклику функції CryptGenKey. При генерації пари ключів для обміну ключами третій параметр цієї функції встановлюють в AT_KEYEXCHANGE, а при генерації пари ключів для формування цифрового підпису - AT_SIGNATURE. Потрібно зауважити, що багато криптопровайдерів дозволяють використовувати пару ключів, створену для обміну, також і для формування цифрового підпису.
- Отримання значення пари ключів. Виконується за допомогою виклику функції CryptGetUserKey. Прапори, що передаються в цю функцію, відповідають прапорам для генерації пари ключів.
- Встановлення параметрів. Виконується за допомогою виклику функції CryptSetProvParam. Здебільшого встановлюють тільки один параметр - дескриптор безпеки контейнера ключів. Цей параметр використовується, наприклад, для видачі доступу до контейнера не тільки користувачеві, який створив контейнер. Інші параметри контейнера ключів можуть сильно варіюватися від криптопровайдера до криптопровайдера, і зазвичай добре розписані в документації, що поставляється безпосередньо з криптопровайдером.
- Отримання параметрів. Виконується за допомогою виклику функції CryptGetProvParam. Набір одержуваних параметрів також може сильно варіюватися залежно від криптопровайдера. Як цікавий стандартний параметр можна назвати параметр PP_UNIQUE_CONTAINER. Як параметр, що повертається, передається унікальний (у масштабах всієї системи) рядок, що ідентифікує контейнер ключів. Зокрема, за цим рядком часто можна дізнатися фізичне розташування контейнера ключів (наприклад, для контейнера ключів, розташованого в регістрі системи, на початку рядка, що повертається, буде "REGISTRY").

- Перерахування контейнерів ключів. Виконується окремо для кожного криптопровайдера в системі за допомогою виклику функції CryptGetProvParam з параметром dwParam встановленим у PP_ENUMCONTAINERS.

Алгоритми

У Crypto API всі криптографічні алгоритми реалізуються всередині криптопровайдерів. Таким чином, криптопровайдер дає змогу встановлювати або отримувати різні параметри алгоритму, отримувати вихідну інформацію алгоритму і багато іншого. Усередині Crypto API алгоритми прийнято ділити на такі групи:

- алгоритми шифрування/розшифрування;
- алгоритми отримання значення хеша;
- алгоритми отримання цифрового підпису;
- алгоритми обміну криптографічними ключами.

Для подальшого розуміння сутності роботи з алгоритмами необхідно також увести поняття симетричних та асиметричних алгоритмів шифрування. Симетричним алгоритмом називають алгоритм, у якому для шифрування та розшифрування використовується один і той самий ключ. Асиметричними називають алгоритми, в яких використовується пара ключів (у загальному випадку може бути і більше двох ключів), один з яких використовується для шифрування (і тільки для шифрування) інформації, а інший - для розшифровування. Як асиметричні алгоритми прийнято застосовувати так звані алгоритми з відкритим ключем. У таких алгоритмах існує пара ключів - відкритий (публічний) і закритий (секретний) ключі. У більшості випадків для шифрування інформації використовують публічний ключ, а для розшифрування - секретний. Публічний ключ не є предметом секретності і призначений для передавання відкритими каналами. Особа, яка отримала публічний ключ, може зашифрувати ним дані та надіслати їх тими самими відкритими каналами особі, яка володіє секретним ключем. Своєю чергою, на стороні одержувача розшифрування можливе тільки за наявності секретного ключа з тієї пари ключів, до якої належить публічний ключ, що застосовувався для шифрування. Таким чином, досягається можливість обмінюватися шифрованими повідомленнями, не піклуючись про те, що ключ може бути перехоплений у процесі пересилання.

Асиметричні алгоритми у звичайному житті мало застосовні. Причина тому - висока трудомісткість шифрування, зумовлена підвищеною складністю асиметричних алгоритмів. Зазвичай використовують таку схему: шифрування даних виконується симетричним криптографічним алгоритмом деяким згенерованим для цього сесійним ключем. Потім для передачі одержувачу цей ключ шифрується за допомогою асиметричного алгоритму (довжина ключа несуттєво мала порівняно з можливим розміром даних). На стороні одержувача за допомогою асиметричного алгоритму розшифровується сесійний ключ, який використовується для розшифрування отриманих даних за допомогою симетричного алгоритму. Такий

спосіб передачі шифрованої інформації дає змогу використовувати переваги обох підходів.

Саме такий підхід використано в усіх ключових алгоритмах роботи Crypto API. Контейнери ключів зберігають пари ключів, необхідні для обміну сесійними ключами, а для безпосередньо шифрування даних необхідно щоразу генерувати нові сесійні ключі.

Розглянемо основні операції, що виконуються над алгоритмами в Crypto API:

- Перерахування всіх криптографічних алгоритмів. Виконується за допомогою виклику функції CryptGetProvParam з параметром dwParam, встановленим у PP_ENUMALGS.
- Отримання параметрів алгоритмів. Зазвичай виконується прямо під час перерахування алгоритмів за допомогою виклику функції CryptGetProvParam з параметром dwParam, встановленим у PP_ENUMALGS_EX. У цьому разі отримувана структура може містити практично всю інформацію про алгоритм.

Найчастіше також потрібно знайти криптопровайдер, що реалізує якийсь відомий нам алгоритм. Для ідентифікації алгоритмів у межах системи всі алгоритми криптопровайдерів мають унікальний цифровий номер. Цей номер у системі має тип ALG_ID і являє собою просте число типу DWORD. Поряд з нумерацією алгоритмів на основі ALG_ID існує й інший ідентифікатор - OID. Цей ідентифікатор прийшов зі специфікацій криптографічних стандартів фірми RSA Security і являє собою рядок, що складається з груп цифр, розділених крапками. У стандартах жорстко закріплено значення кожної групи цифр в OID, і за цим номером можна отримати деяку додаткову інформацію (наприклад, фірму-розробника алгоритму). Перетворення між типами номерів алгоритмів здійснюється функціями CertAlgIdToOID і CertOIDToAlgId.

Сертифікати

Повне пояснення щодо цього поняття буде дано в окремому розділі. Тут же про сертифікати згадано, оскільки сертифікати все-таки використовуються у високорівневих функціях обробки криптографічних повідомлень типу CryptSignMessage. Поки що важливо розуміти, що сертифікати в системі використовуються для:

- зберігання додаткової інформації про власника сертифіката;
- зберігання значення публічного ключа власника;
- отримання посилання на контейнер ключів. Із сертифікатом, встановленим у системі, може бути фізично пов'язаний контейнер ключів. Тобто, маючи лише посилання на контекст сертифіката, можна встановити однозначний зв'язок із секретним ключем і використовувати це, наприклад, при перевірці цифрового підпису або розшифровуванні даних.

У спрощеному вигляді сертифікати можна вважати деякими посвідченнями особи. Правами видавати сертифікати (як і паспорти), в ідеалі повинні володіти тільки довірені центри (наприклад, ФСБ або міліція). Але, в принципі, можна генерувати сертифікати і самому. У цьому разі той користувач, який використовуватиме цей сертифікат, має право або довіряти, або не довіряти сертифікату ("...паспорт можна і самому намалювати..."). Для цілей подальшого вивчення високорівневих функцій обробки криптографічної інформації нам важливі лише два останні випадки використання сертифікатів з перерахованих: сертифікати зберігають публічний ключ і можуть бути пов'язані з контейнером ключів, у якому зберігається секретний ключ. Цієї інформації на цьому етапі буде достатньо.

Якщо ви не мали досвіду роботи з сертифікатами, то для ознайомлення з ними можна викликати Internet Explorer, вибрати в його меню пункт "Сервіс / Властивості оглядача" ("Tools / Internet Options"), а потім на закладці "Зміст" ("Content") натиснути на кнопку "Сертифікати" ("Certificates"). Вашій увазі буде представлено список сертифікатів, встановлених у системі.

Базові функції

Шифрування

Базова функція шифрування даних має таке оголошення:

```
BOOL CryptEncrypt(HCRYPTKEY hKey,
                  HCRYPTHAS hHash,
                  BOOL Final,
                  DWORD dwFlags,
                  BYTE* pbData,
                  DWORD* pdwDataLen,
                  DWORD dwBufLen);
```

Першим параметром цієї функції передається хендл сесійного ключа, що застосовується для шифрування. Другий параметр досить рідко використовується і призначений для отримання хеша даних одночасно з їх шифруванням. Така можливість досить корисна під час формування одночасно як шифрованих даних, так і цифрового підпису цих же даних.

Ця функція може обробляти дані блоками. Тобто немає потреби одразу завантажувати в пам'ять цілком весь масив даних, а лише потім передавати посилання на нього криптографічній функції. Достатньо передавати масив даних поблочно, спеціальним чином зазначивши лише останній блок даних (це зазвичай потрібно, щоб криптопровайдер провів деякі дії після використання сесійного ключа). Для вказівки того, що це останній блок даних, у функції CryptEncrypt використовується третій параметр Final. Четвертий параметр служить покажчиком

на масив вхідних/вихідних даних. Тут потрібно відразу зазначити деяку загальну схему роботи з даними в Crypto API. Якщо дані, що повертаються, можуть бути будь-якого розміру (а це можливо, адже, скажімо, в алгоритмі може відбуватися проста заміна, коли одна буква кодується чотирма цифрами), то робота з функцією складається з двох етапів. На першому етапі у функцію передається загальний розмір вхідних даних і NULL як посилання на сам масив вихідних даних. Функція повертає довжину вихідного масиву даних, користувач ініціалізує пам'ять необхідного розміру і лише потім заново передає функції посилання на цей масив. Така ж схема використовується і в роботі з функцією CryptEncrypt. Параметр pdwDataLen служить для повернення розміру даних, які повертає функція. Параметр dwBufLen слугує для зазначення довжини вхідного буфера даних. Параметр dwFlags зазвичай не використовується і встановлюється в 0.

Експорт сесійних ключів

Після виконання операції шифрування постає проблема передачі шифрованих даних. Самі по собі дані, звісно, передавати можна, внаслідок їх захищеності. Але нагадаємо ще раз, що в Crypto API використовуються симетричні алгоритми шифрування, і якщо на стороні, що приймає, не буде використано той самий сесійний ключ, який було використано для шифрування, то розшифрувати дані на стороні, що приймає, не вдасться. У самих шифрованих даних Crypto API самостійно сесійні ключі також не передає. Натомість Crypto API надає розвинені механізми експорту значення сесійного ключа в зовнішній масив даних.

Базова функція експорту ключів має такий опис:

```
BOOL CryptExportKey(HCRYPTKEY hKey,
                    HCRYPTKEY hExpKey,
                    DWORD dwBlobType,
                    DWORD dwFlags,
                    BYTE* pdData,
                    DWORD* pdwDataLen);
```

Першим параметром цієї функції передається хендл ключа, який буде експортовано. Фактично, експорт ключа можна представити як окрему операцію шифрування ключа. Отже, для такої операції необхідний ще один ключ шифрування. Зазвичай у Crypto API сесійний ключ шифрують за допомогою асиметричного алгоритму. Параметр hExpKey у більшості випадків ініціалізують контекстом публічного ключа одержувача. Параметр dwBlobType визначає формат одержуваного блоку експорту. Можливо, скажімо, вказати, що експорту підлягатиме лише публічний ключ. У цьому разі параметр hExpKey має дорівнювати 0 (шифрування публічного ключа не потрібне) і на виході функції отримується просте значення публічного ключа. Для такого випадку параметр dwBlobType має дорівнювати PUBLICKEYBLOB. Зазвичай же, при експорті сесійного ключа використовується значення SIMPLEBLOB. Решта значень цього параметра досить специфічні і застосовуються рідко. Параметри pbData і pdwDataLen вказують на масив, виділений для отримання експортованого ключа, і на його розмір.

Імпорт сесійних ключів

Базова функція імпорту ключів має такий опис:

```
BOOL CryptImportKey(HCRYPTPROV hProv,
                    BYTE* pbData,
                    DWORD dwDataLen,
                    HCRYPTKEY hPubKey,
                    DWORD dwFlags,
                    HCRYPTKEY* phKey);
```

Як перший параметр у цю функцію передається ініціалізований контекст криптопровайдера. Мушу зазначити, що для успішного завершення роботи функції CryptImportKey необхідно, щоб під час ініціалізації криптопровайдера було вказано контейнер ключів. Зокрема, це необхідно для успішного імпорту секретних ключів у контейнер ключів. Параметр pbData являє собою посилання на імпортовані дані, параметр dwDataLen - довжину цих даних. У параметрі hPubKey вказують хендл ключа, застосовуваного під час імпорту (для розшифровування сесійного ключа). Параметр dwFlags зазвичай не застосовується і може бути встановлений в 0. У параметрі phKey повертається імпортований ключ.

Розшифрування

Базова функція розшифрування має такий опис:

```
BOOL CryptDecrypt(HCRYPTKEY hKey,
                  HCRYPTHASH hHash,
                  BOOL Final,
                  DWORD dwFlags,
                  BYTE* pbData,
                  DWORD* pdwDataLen);
```

Першим параметром цієї функції передається ініціалізований контекст сесійного ключа, застосовуваного для розшифровування даних. Другий параметр, як і в попередньому прикладі, пов'язаний, здебільшого, з функцією отримання та перевірки цифрового підпису. Зазвичай він не використовується і встановлюється в 0. Параметр dwFlags найчастіше не використовується і також встановлюється в 0. Параметри pbData і pdwDataLen використовуються так само, як і в CryptEncrypt і являють собою посилання на вхідний/вихідний масив даних і довжину цього масиву даних.

Хешування

Під хешуванням розуміють застосування деякої математичної функції (званої хеш-функцією) до деяких даних. Під час застосування хеш-функції до довільного об'єму даних завжди виходить масив даних фіксованого розміру. До хеш-значення висувається вимога "стійкості до колізій". Це означає, що хеш-функція тим краща, чим важче знайти два таких випадкових вхідних масиви даних, для яких збігалися б генеровані хеш-значення. Під час обробки одних і тих самих даних хеш-функція зобов'язана повертати одне й те саме хеш-значення. Цю властивість хеш-функцій використовують, насамперед, для контролю над цілісністю даних. Адже якщо ми змінимо хоч біт у вхідному масиві інформації, то результат роботи хеш-функції (з високою ймовірністю) буде іншим.

У Crypto API для маніпуляції з хешем використовується спеціальний хеш-об'єкт. Взаємодія з цим об'єктом здійснюється за допомогою таких трьох функцій:

- CryptCreateHash;
- CryptHashData;
- CryptGetHashParam.

Для первинної ініціалізації хеш-об'єкта застосовують функцію CryptCreateHash. Ця функція має такий опис:

```
BOOL CryptCreateHash(HCRYPTPROV hProv,
                     ALG_ID AlgId,
                     HCRYPTKEY hKey,
                     DWORD dwFlags,
                     HCRYPTHASH* phHash);
```

Як перший параметр цієї функції передається ініціалізований контекст криптопровайдера. Другим параметром вказується алгоритм отримання значення хеша. Параметр hKey необхідний лише в разі застосування спеціалізованих алгоритмів типу MAC і HMAC.

Параметр dwFlags зарезервований під можливе майбутнє використання і має завжди дорівнювати 0. Через параметр phHash функція повертає хендл створеного їй хеш-об'єкта. Після того, як хеш-об'єкт стане непотрібним, потрібно звільнити хеш-об'єкт за допомогою виклику функції CryptDestroyHash.

Після ініціалізації хеш-об'єкта можна почати передачу даних хеш-функції за допомогою виклику CryptHashData. Ця функція має такий опис**:**

```
BOOL CryptHashData(HCRYPTHASH hHash,
                   BYTE* pbData,
```

```
                    DWORD dwDataLen,
                    DWORD dwFlags);
```

Як перший параметр цієї функції передається раніше ініціалізований хендл хеш-об'єкта. Другим параметром передається порція даних для хеш-функції. Параметр dwDataLen являє собою довжину переданих даних. Параметр dwFlags зазвичай дорівнює нулю.

Після повної передачі всього масиву вхідних даних функції CryptHashData виникає необхідність в отриманні значення хеш-функції. Це завдання вирішується із застосуванням функції CryptGetHashParam. Ця функція має такий опис:

```
BOOL CryptGetHashParam(HCRYPTHASH hHash,
                       DWORD dwParam,
                       BYTE* pbData,
                       DWORD* pdwDataLen,
                       DWORD dwFlags);
```

Як перший параметр цієї функції передається раніше ініціалізований хендл хеш-об'єкта. Другий параметр, dwParam, функції визначає тип запитуваного значення. Для отримання хеш-значення необхідно передати другим аргументом значення HP_HASHVAL. Параметри pdData і pdwDataLen відповідають за блок пам'яті, який використовується під значення, що повертається. Параметр dwFlags зарезервований для майбутнього використання і має дорівнювати нулю.

Для перевірки правильності хеш-значення потрібно отримати хеш-значення даних і звірити його з хеш-значенням, що перевіряється.

Цифровий підпис

Під цифровим підписом розуміють якусь похідну даних, за якою однозначно можна визначити цілісність і відправника надісланих даних. У найпростішому випадку під цифровим підписом можна розуміти навіть власне шифрований контент у разі, коли ключ шифрування не скомпрометований і однозначно належить відомому відправнику інформації. На практиці ж використовують набагато цікавіший метод: первинно використовують отримання хешу даних, а потім шифрують отримане хеш-значення за допомогою алгоритму з відкритим ключем. Таким чином, за отриманим цифровим підписом можна судити як про цілісність даних (розшифрувавши цифровий підпис, ми можемо потім перевірити отримане значення хешу), так і про відправника даних (для отримання

цифрового підпису використовують не публічний ключ відправника, а секретний, який може бути використано тільки самим відправником).

Саме такий підхід до формування цифрового підпису використовується в базових функціях Crypto API для роботи з підписом. Базова функція отримання підпису хеша даних має такий опис:

```
BOOL CryptSignHash(HCRYPTHASH hHash,
                   DWORD dwKeySpec,
                   LPCTSTR sDescription,
                   DWORD dwFlags,
                   BYTE* pbSignature,
                   DWORD* pdwSigLen);
```

Як перший параметр використовується значення хендла хеш-об'єкта, вже ініціалізованого даними (за допомогою функції CryptHashData). Параметр dwKeySpec визначає, яка саме пара ключів буде використана для формування підпису (AT_KEYEXCHANGE (пара для обміну ключами) або AT_SIGNATURE (пара для формування цифрового підпису)). Ще раз хочеться звернути увагу читача, що в багатьох криптопровайдерах пара ключів, призначена для обміну ключами, може також використовуватися і для формування цифрового підпису (але не у всіх криптопровайдерах). Параметр sDescription більше не використовується в даній функції і його значення повинно завжди бути встановлено в NULL. Параметр dwFlags зазвичай встановлюють також у 0. Параметри pbSignature і pdwSigLen використовують для коректної вказівки посилання на масив вихідних даних і його розміру.

Перевірка цифрового підпису

Для перевірки цифрового підпису хеш-значення використовується базова функція, що має такий опис:

```
BOOL CryptVerifySignature(HCRYPTHASH hHash,
                          BYTE* pbSignature,
                          DWORD dwSigLen,
                          HCRYPTKEY hPubKey,
                          LPCTSTR sDescription,
                          DWORD dwFlags);
```

Як перший параметр у функцію передається хендл хеш-об'єкта, попередньо ініціалізований даними за допомогою функції CryptHashData. Другий і третій параметри відповідають за передачу значення підпису, що перевіряється. Параметр hPubKey використовується для зазначення хендла публічного ключа відправника підпису (того, хто власне сформував цифровий підпис). Параметр sDescription наразі більше не використовується і його значення має бути

встановлено в NULL. Параметр dwFlags також зазвичай не несе корисного навантаження і встановлюється в 0.

CAPICOM

Найчастіше застосування безпосередньо функцій Crypto API досить проблематичне. Наприклад, у Web-клієнтах, де виклики процедур безпосередньо неможливі. Для подібних цілей, а також для спрощення роботи з Crypto API було створено тип об'єктів CAPICOM (Crypto API COM-object). У своїй реалізації цей об'єкт майже повністю охоплює все те, про що йшлося вище в цій статті: від шифрування до роботи з сертифікатами. Завдяки використанню дуальних інтерфейсів, доступ до цього об'єкта можливий як із клієнтів із раннім зв'язуванням (C++), так і з пізнім (VBscript).

В основі роботи всіх функцій CAPICOM лежить використання високорівневих функцій роботи з криптографічними повідомленнями, які були розглянуті раніше. Таким чином, базовим стандартом для вихідних даних є стандарт PKCS #7, вторинно кодований Bas64. Так само, як і у випадку високорівневих функцій, робота ведеться тільки з відносно невеликою ділянкою даних, завантажених у пам'ять. Робота з даними великого обсягу або робота з поблоковим завантаженням даних в об'єкті не передбачена.

Web Crypto API

Web Crypto API - це набір продуктів, пов'язаних з блокчейном і криптовалютою, які допоможуть вам скоротити витрати на розробку та інфраструктуру. Це рівень інфраструктури, який значно скорочує час виходу на ринок.

- Гаманець як послуга(**Wallet as a Service**) - це цифровий гаманець, який включає в себе найкращі функції, безпеку та процеси авторизації, що існують на сьогоднішній день. Базується на MPC, використовує TSS та розподілену генерацію ключів, підкріплений ескроу третьої сторони. Розміщений на мульти-розподілених вузлах, Crypto APIs Wallet as a Service може запропонувати клієнтам як серверні, так і мобільні вузли. Різноманітні методи безпеки забезпечують регулярний моніторинг, аудит і перевірку системи. Функціональність веб-хуків, вбудована разом з білим списком, рівнем управління, модифікацією кворуму (M/N вузли) і багато іншого.

- Вузол як послуга(**Node as a Service**) - потужні спільні та виділені вузли з Crypto API. Обирайте між загальними, виділеними або самокерованими вузлами для посилення ваших блокчейн- та крипто-проектів.
- **Blockchain Data** - інтегрувавши його один раз, ви зможете отримувати найважливіші дані для найпоширеніших блокчейнів. Отримуйте результати для кожного підтримуваного протоколу блокчейну, змінюючи лише один параметр. Отримуйте як історичні дані, так і дані в реальному часі, які можуть бути уніфікованими, необробленими, доступними тільки для читання або сегментованими. Швидкий час відгуку 25 мс гарантується масштабованою і збалансованою за навантаженням базою даних, де всі дані синхронізуються та індексуються. Ви можете отримати інформацію про такі дані блокчейну, як блоки, адреси, гаманці HD (xPub, yPub, zPub), баланси, непідтверджені транзакції, підтверджені транзакції, дядьки, епохи, токени, смарт-контракти, внутрішні транзакції, мемпул, комісії та багато іншого.
- **Blockchain Events** - працює за принципом Webhook. Ви можете підписатися на певні події і отримувати сповіщення кожного разу, коли вони відбудуться. Немає необхідності в хостингу або підтримці будь-якого вузла, а середній час доставки оцінюється в 100 мс. Розглянуті події можуть включати: новий видобутий блок, нові непідтверджені та видобуті транзакції, певну кількість підтверджень (блоків) для нової транзакції тощо.
- Автоматизація блокчейну(**Blockchain Automations**) - підписавшись на певну автоматизацію, засновану на певній події, ви можете отримати зворотний дзвінок, коли вона буде виконана. Блокчейн-автоматизації не вимагають піклування про гарячі (тимчасові) адреси. Більше того, ви можете згенерувати стільки адрес, скільки бажаєте, а потім призначити їх своїм користувачам, де всі отримані кошти будуть автоматично переводитися на ваш безпечний гаманець (теплий/холодний). Такі автомати можуть автоматично переміщати отримані токени та/або кошти на інший гаманець.
- **Blockchain Tools** - колекція корисних кінцевих точок API, які забезпечують простий спосіб підключення до блокчейну та виконання певних операцій. Кінцеві точки REST API скорочують час розробки та зменшують витрати на сервери. Ці інструменти для блокчейну включають кінцеві точки для перевірки дійсності адреси, отримання HD (xPub, yPub, zPub) адрес, їх отримання та зміни, трансляції підписаних транзакцій та багато іншого.
- **Crypto Market Data** - отримуйте інформацію про курси валют, монети та біржі в реальному часі та історичні дані.

Microsoft CryptoAPI

Криптографічний інтерфейс прикладного програмування для платформи Microsoft Windows (також відомий як CryptoAPI, Microsoft Cryptography API, MS-CAPI або просто CAPI) - це інтерфейс прикладного програмування, що входить до складу операційних систем Microsoft Windows, який надає послуги, що дозволяють розробникам захищати додатки на базі Windows за допомогою криптографії. Це набір динамічно зв'язаних бібліотек, що забезпечує рівень абстракції, який ізолює програмістів від коду, що використовується для шифрування даних. Вперше Crypto API було представлено в Windows NT 4.0[1] і вдосконалено в наступних версіях.

CryptoAPI підтримує як криптографію з відкритим ключем, так і криптографію з симетричним ключем, хоча постійні симетричні ключі не підтримуються. Він включає функціональність для шифрування і дешифрування даних і для автентифікації за допомогою цифрових сертифікатів. Він також включає криптографічно захищену функцію генератора псевдовипадкових чисел CryptGenRandom.

CryptoAPI працює з декількома CSP (постачальниками криптографічних послуг), встановленими на комп'ютері. CSP - це модулі, які виконують фактичну роботу з кодування та декодування даних, виконуючи криптографічні функції. Постачальники HSM можуть постачати CSP, які працюють з їхнім обладнанням.

Cryptography API: Next Generation

Windows Vista містить оновлення інтерфейсу Crypto API, відомого як Cryptography API: Наступне покоління (CNG). Він має кращий факторинг API, що дозволяє виконувати ті самі функції за допомогою широкого спектру криптографічних алгоритмів, а також включає низку нових алгоритмів, які є частиною набору B Агентства національної безпеки (АНБ).[2] Він також є гнучким, оскільки підтримує підключення користувацьких криптографічних API до середовища виконання CNG. Однак, провайдери зберігання ключів CNG все ще не підтримують симетричні ключі.[3] CNG працює як в режимі користувача, так і в режимі ядра, а також підтримує всі алгоритми з CryptoAPI. Провайдер Microsoft, який реалізує CNG, розміщений в Bcrypt.dll.

CNG також підтримує криптографію еліптичної кривої, яка, оскільки використовує коротші ключі для того ж очікуваного рівня безпеки, є більш ефективною, ніж RSA.[4] API CNG інтегрується з підсистемою смарт-карток шляхом включення модуля Base Smart Card Cryptographic Service Provider (Base

CSP), який інкапсулює API смарт-картки. Виробникам смарт-карток потрібно лише зробити свої пристрої сумісними з ним, а не створювати рішення з нуля.

CNG також додає підтримку Dual_EC_DRBG,[5] генератора псевдовипадкових чисел, визначеного в NIST SP 800-90A, який може наразити користувача на прослуховування з боку Агентства національної безпеки, оскільки містить клептографічний бекдор, якщо тільки розробник не згадає згенерувати нові базові точки за допомогою іншого криптографічно захищеного генератора псевдовипадкових чисел або справжнього генератора випадкових чисел, а потім опублікувати згенероване зерно, щоб видалити бекдор АНБ. Він також є дуже повільним[6] і використовується лише тоді, коли це явно вимагається.

CNG також замінює стандартний ГПСЧ на CTR_DRBG, використовуючи AES в якості блочного шифру, тому що попередній ГПСЧ, який визначений у вже застарілому стандарті FIPS 186-2, заснований на DES або SHA-1, обидва з яких були зламані.[7] CTR_DRBG є одним з двох алгоритмів в NIST SP 800-90, схвалених Шнайером, іншим є Hash_DRBG.[6]

Crypto API (Linux)

Crypto API - це криптографічний фреймворк у ядрі Linux для різних частин ядра, які займаються криптографією, таких як IPsec і dm-crypt. Він був представлений у ядрі версії 2.5.45 і з тих пір розширився, включивши в себе практично всі популярні блокові шифри і хеш-функції.

Userspace interfaces

Багато платформ, які забезпечують апаратне прискорення шифрування AES, надають цю можливість програмам через розширення архітектури набору інструкцій (ISA) різних чипсетів (наприклад, набір інструкцій AES для x86). Завдяки такій реалізації будь-яка програма (в режимі ядра або в користувацькому просторі) може використовувати ці можливості напряму.

Однак деякі платформи, такі як процесори ARM Kirkwood SheevaPlug та AMD Geode, не реалізовано як розширення ISA, і вони доступні лише через драйвери режиму ядра. Для того, щоб користувацькі програми, які використовують шифрування, такі як wolfSSL, OpenSSL або GnuTLS, могли скористатися перевагами такого прискорення, вони повинні взаємодіяти з ядром.

AF_ALG

: Інтерфейс на основі мережевих посилань, який додає сімейство адрес AF_ALG; його було включено у версію 2.6.38 ядра Linux. Колись існував плагін до OpenSSL для підтримки AF_ALG, який було подано на злиття. У версії 1.1.0 OpenSSL отримав ще один патч для AF_ALG, наданий Intel. wolfSSL може використовувати AF_ALG і cryptodev

cryptodev

: Інтерфейс OpenBSD Cryptographic Framework /dev/crypto з OpenBSD було перенесено на Linux, але так і не було об'єднано.

PKCS 11

Стандарт інтерфейсу криптографічних токенів PKCS#11, також відомий як Cryptoki, є одним із стандартів криптографії з відкритим ключем, розроблений RSA Security. PKCS#11 визначає інтерфейс між програмою та криптографічним пристроєм. У цій главі наведено загальний опис PKCS#11 та деякі з його основних понять. Якщо читач не знайомий з PKCS#11, ми наполегливо рекомендуємо йому звернутися до PKCS #11: *Стандарт інтерфейсу криптографічних токенів*.

PKCS#11 використовується як низькорівневий інтерфейс для виконання криптографічних операцій без необхідності для програми безпосередньо взаємодіяти з пристроєм через його драйвер. PKCS#11 представляє криптографічні пристрої, використовуючи загальну модель, яка називається просто токеном. Таким чином, програма може виконувати криптографічні операції на будь-якому пристрої або токені, використовуючи один і той же незалежний набір команд.

SafeNet ProtectToolkit-C - це постачальник криптографічних послуг, що використовує стандарт інтерфейсу прикладного програмування (API) PKCS #11, визначений RSA Labs. Він включає в себе легкий, власний Java API для доступу до функцій PKCS #11 з Java.

API PKCS #11, також відомий як Cryptoki, включає в себе набір криптографічних сервісів для шифрування, дешифрування, генерації підписів, перевірки підписів та постійного зберігання ключів. Програмне забезпечення, що міститься на інсталяційному DVD, сумісне з PKCS #11 v. 2.20. Найновіші версії клієнтського програмного забезпечення та мікропрограми HSM можна знайти на Порталі технічної підтримки клієнтів Thales.

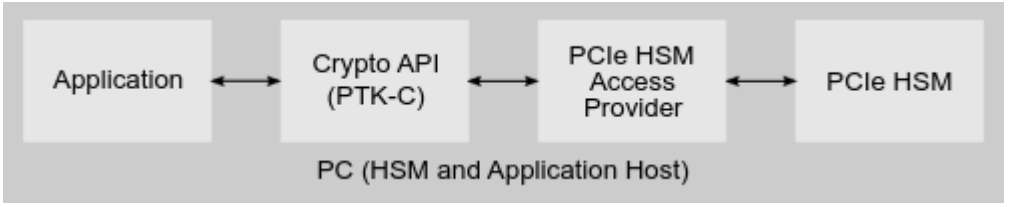
Для забезпечення найвищого рівня безпеки SafeNet ProtectToolkit-C взаємодіє з програмним забезпеченням провайдера доступу SafeNet і лінійкою апаратних модулів безпеки (HSM) SafeNet:

- SafeNet ProtectServer Network HSM
- SafeNet ProtectServer PCIe HSM

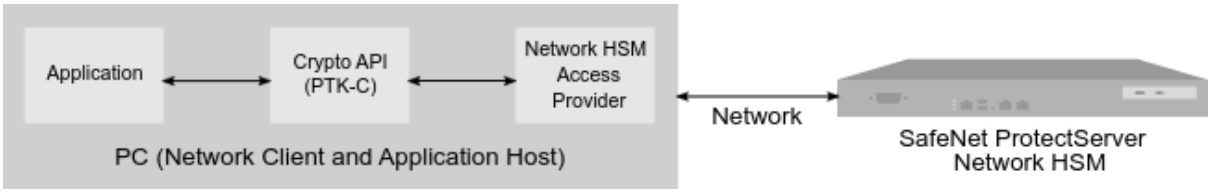
HSM включають в себе високошвидкісне апаратне прискорення DES і RSA, а також загальну обробку безпеки. Безпечне, постійне, стійке до несанкціонованого доступу сховище CMOS-ключів включено. На одному комп'ютері можна використовувати кілька адаптерів для підвищення пропускної здатності або забезпечення надмірності. HSM можуть бути встановлені локально, в тій же системі, що і SafeNet ProtectToolkit-C, або віддалено через мережу.

SafeNet ProtectToolkit-C можна використовувати в одному з трьох режимів роботи. Це

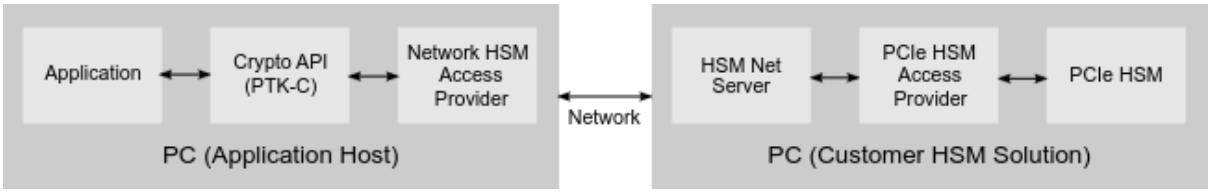
- **Режим PCI** в поєднанні з локально встановленим SafeNet ProtectServer PCIe HSM.



- Мережевий режим через мережу TCP/IP у поєднанні з сумісним продуктом, таким як SafeNet ProtectServer Network HSM.



Комп'ютер, на якому встановлено SafeNet ProtectServer PCIe HSM, також можна використовувати як сервер у мережевому режимі.



- Тільки програмний режим, на локальній машині без доступу до апаратного модуля безпеки.

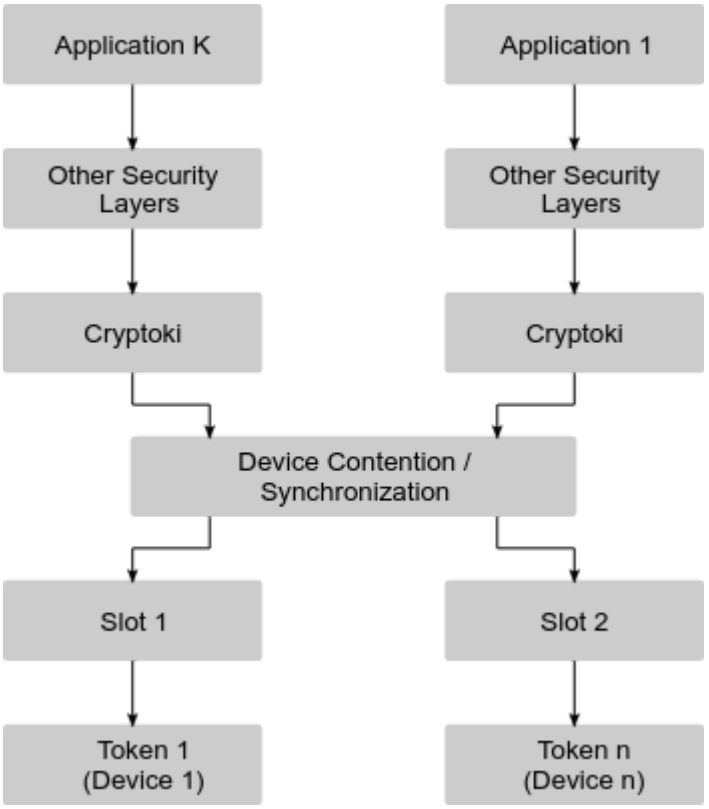
У клієнт-серверному середовищі виконання сервер виконує криптографічну обробку за запитом клієнта. Сам сервер буде працювати тільки в одному з апаратних режимів виконання.

Версія тільки для програмного забезпечення доступна для різних платформ, включаючи Windows NT і Solaris, і зазвичай використовується як середовище розробки і тестування додатків, які в кінцевому підсумку будуть використовувати апаратний варіант SafeNet ProtectToolkit-C.

Модель PKCS#11

Модель для PKCS#11 можна побачити на ілюстрації нижче, яка демонструє, як додаток передає свої запити токену через інтерфейс PKCS#11. Термін "слот" позначає фізичний інтерфейс пристрою. Наприклад, зчитувач смарт-карт буде представляти слот, а смарт-карта - токен. Також можливо, що декілька слотів можуть використовувати один і той самий токен.

Загальна модель PKCS#11:



В рамках PKCS#11 токен розглядається як пристрій, який зберігає об'єкти і може виконувати криптографічні функції. Об'єкти зазвичай визначаються в одному з чотирьох класів:

- Об'єкти даних, які визначаються додатком
- об'єкти сертифікатів, які є цифровими сертифікатами, такими як X.509
- об'єкти ключів, які можуть бути відкритими, приватними або секретними криптографічними ключами
- Об'єкти, визначені постачальником

Об'єкти в PKCS#11 також визначаються як об'єкти токенів або об'єкти сеансів. Об'єкти токенів є видимими для будь-якої програми, яка має достатній дозвіл на доступ і підключена до цього токену. Важливим атрибутом об'єкта токена є те, що він залишається на токені до тих пір, поки не буде виконана певна дія для його видалення.

З'єднання між токеном і додатком називається сеансом. Сесійні об'єкти є тимчасовими і існують лише доти, доки сесія відкрита. Об'єкти сеансу видимі лише для програми, яка їх створила.

Доступ до об'єктів у PKCS#11 визначається типом об'єкта. Загальнодоступні об'єкти видимі будь-якому користувачеві або додатку, тоді як приватні об'єкти вимагають, щоб користувач був зареєстрований в цьому токені, щоб їх переглянути. PKCS#11 розпізнає два типи користувачів, а саме: співробітник служби безпеки (SO) або звичайний користувач. Єдина роль співробітника служби безпеки полягає в ініціалізації токену та встановленні PIN-коду доступу звичайного користувача.

Контроль доступу до ключової інформації ЗІ СТОРОНИ СЕРВЕРА в системі клієнт-сервер.

Ви можете захистити ресурси, які знаходяться на вашому веб-сервері, за допомогою декількох служб і механізмів безпеки, включаючи автентифікацію, авторизацію і контроль доступу. У цьому розділі описано деякі з підтримуваних механізмів контролю доступу до вашого веб-сервера

Що таке контроль доступу

Аутентифікація - це процес підтвердження особи. Авторизація означає надання особі доступу до обмеженого ресурсу, а механізми контролю доступу забезпечують дотримання цих обмежень. Аутентифікацію та авторизацію можна забезпечити за допомогою низки моделей безпеки (безпека веб-додатків, htaccess, Authentication Realm тощо) та сервісів.

Контроль доступу дозволяє вам визначити:

- Хто може отримати доступ до вашого сервера адміністрування
- До яких додатків вони можуть отримати доступ
- Хто може отримати доступ до файлів або каталогів на вашому веб-сайті

Ви можете контролювати доступ до всього сервера або до окремих його частин, а також до файлів або каталогів на вашому веб-сайті. Ви створюєте ієрархію правил, які називаються записами контролю доступу (ACE), щоб дозволити або заборонити доступ. Колекція створених вами ACE називається списком управління доступом (ACL).

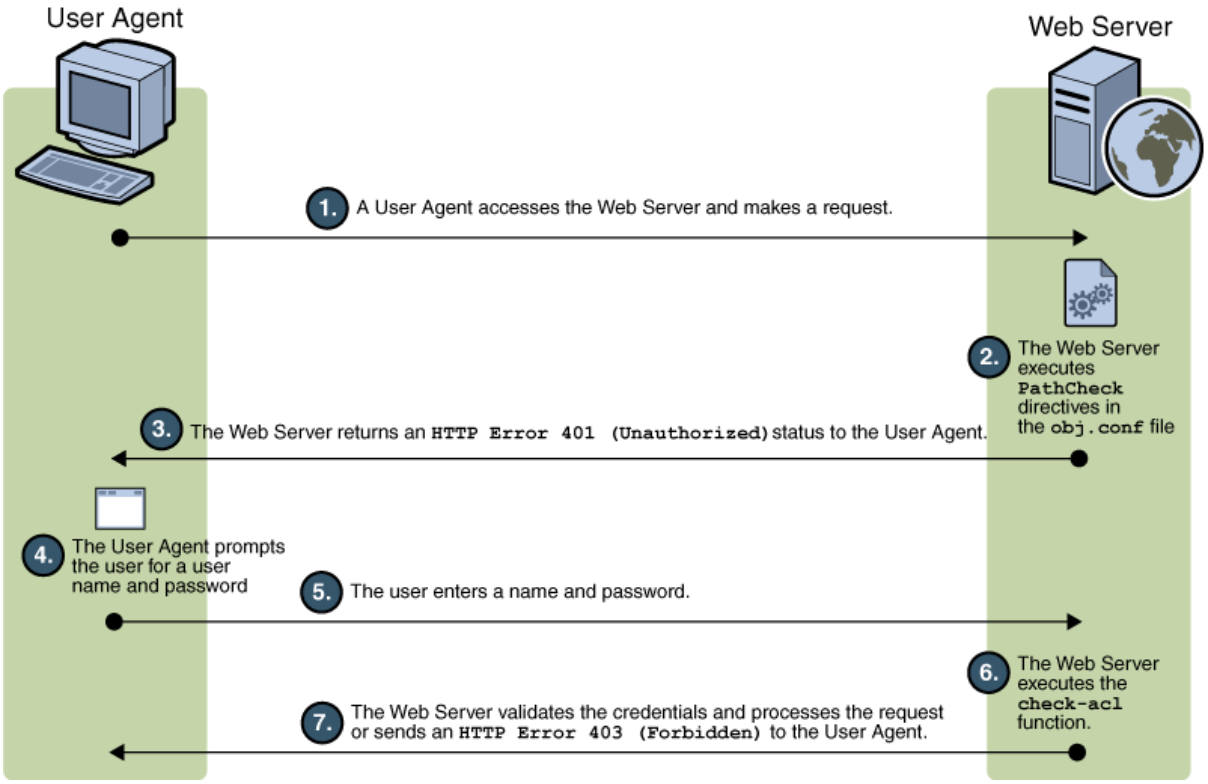
За замовчуванням сервер має один файл ACL, який містить декілька ACL. Після визначення віртуального сервера, який буде використовуватися для вхідного запиту, сервер перевіряє, чи налаштовані для цього віртуального сервера будь-які ACL. Якщо знайдено ACL, які застосовуються до поточного запиту, сервер оцінює їхні ACE, щоб визначити, чи слід надати доступ або заборонити його.

Ви дозволяєте або забороняєте доступ на основі:

- Хто робить запит (група користувачів)
- Звідки надходить запит (Host-IP)
- Коли відбувається запит (наприклад, час доби)
- Який тип з'єднання використовується (SSL)

Як працює контроль доступу

Коли сервер отримує запит на сторінку, він використовує правила у файлі ACL, щоб визначити, чи надавати доступ до неї, чи ні. Правила можуть посилатися на ім'я хоста або IP-адресу комп'ютера, що надсилає запит. Правила також можуть посилатися на користувачів і групи, що зберігаються у каталозі LDAP.



На попередньому рисунку показано, як працює контроль доступу у Web Server 7.0.9. Користувацький агент (клієнт) отримує доступ до веб-сервера, а потім веб-сервер виконує директиви PathCheck у файлі obj.conf. Веб-сервер повертає клієнту HTTP 401 (неавторизований). Клієнт запитує користувача для аутентифікації. У випадку, якщо клієнтом є браузер, з'являється діалогове вікно для входу в систему. Користувач вводить інформацію для входу. Веб-сервер виконує внутрішню функцію перевірки входу. Веб-сервер перевіряє облікові дані користувача і обробляє запит.

Налаштування контролю доступу для групи користувачів

Ви можете обмежити доступ до веб-сервера певним користувачам або групам. Контроль доступу за групами користувачів вимагає від користувачів введення імені користувача і пароля перед отриманням доступу до сервера. Сервер порівнює інформацію у сертифікаті клієнта із записом у каталозі сервера.

Сервер адміністрування використовує лише базову автентифікацію. Якщо ви хочете вимагати автентифікацію клієнта на вашому сервері адміністрування, ви повинні вручну відредагувати файли ACL, змінивши метод на SSL.

Автентифікація групи користувачів виконується веб-сервером шляхом читання записів у базі даних груп користувачів. Інформація, яку служба каталогів

використовує для реалізації контролю доступу, може надходити з будь-якого з наступних джерел:

- Внутрішня пласка база даних файлового типу
- Зовнішня база даних LDAP

Коли сервер використовує зовнішню службу каталогів на основі LDAP, він підтримує наступні типи методів автентифікації групи користувачів для екземплярів сервера:

- За замовчуванням
- Базовий
- SSL
- Дайджест
- Інші

Якщо сервер використовує внутрішню службу каталогів на основі файлів, підтримувані ним методи автентифікації для екземплярів сервера - "Користувач-група":

- За замовчуванням
- Базовий
- Дайджест

Автентифікація групи користувачів вимагає від користувачів автентифікації перед отриманням доступу до сервера або файлів і каталогів на вашому веб-сайті. Процес автентифікації передбачає, що користувачі підтверджують свою особу, вводячи ім'я користувача і пароль, використовуючи клієнтський сертифікат. Клієнтські сертифікати потрібні лише для SSL-з'єднання.

Автентифікація за замовчуванням

Автентифікація за замовчуванням
Автентифікація за замовчуванням є найкращим методом автентифікації. Параметр За замовчуванням використовує метод за замовчуванням у файлі server.xml, або "Базовий", якщо у файлі server.xml не вказано жодних параметрів. Якщо ви виберете "За замовчуванням", правило ACL не вказуватиме метод у файлі ACL. Вибір значення "За замовчуванням" дозволяє вам легко змінити методи для всіх ACL, відредагувавши один рядок у файлі obj.conf.

Базова автентифікація

Базова автентифікація вимагає, щоб користувачі вводили ім'я користувача та пароль для доступу до вашого веб-сервера або веб-сайту. Базова автентифікація використовується за замовчуванням, і для того, щоб її використовувати, ви повинні створити і зберегти список користувачів і груп у базі даних LDAP, наприклад, Oracle Directory Server Enterprise Edition, або у файлі. Ви повинні використовувати сервер каталогів, встановлений на іншому серверному корені, ніж ваш веб-сервер, або сервер каталогів, встановлений на віддаленому комп'ютері.

Коли користувачі намагаються отримати доступ до ресурсу, який має автентифікацію "користувач-група" на сервері адміністрування або на вашому веб-сайті, веб-браузер відображає діалогове вікно з проханням ввести ім'я користувача та пароль. Сервер отримує цю інформацію в зашифрованому або незашифрованому вигляді, залежно від того, чи ввімкнено шифрування на вашому сервері.

SSL-автентифікація

Сервер може підтвердити особу користувача за допомогою сертифікатів безпеки двома способами:

- Використання інформації в клієнтському сертифікаті як підтвердження особи
- Перевірка клієнтського сертифіката, опублікованого в каталозі LDAP (додатково)

Коли ви налаштовуєте сервер на використання інформації сертифіката для автентифікації клієнта, сервер:

- Спочатку перевіряє, чи сертифікат отримано від довіреного центру сертифікації. Якщо ні, автентифікація не пройде і транзакція буде завершена.
- Зіставляє сертифікат із записом користувача за допомогою файлу certmap.conf, якщо сертифікат від довіреного центру сертифікації (ЦС).
- Перевіряє правила ACL, вказані для цього користувача, якщо сертифікат зіставлено правильно. Навіть якщо сертифікат зіставлено правильно, правила ACL можуть відмовити користувачеві у доступі.

Вимога автентифікації клієнта для контролю доступу до певних ресурсів відрізняється від вимоги автентифікації клієнта для всіх підключень до сервера. Якщо ви налаштуєте сервер так, щоб він вимагав автентифікацію клієнта для всіх

підключень, клієнту потрібно лише надати дійсний сертифікат, виданий довіреним центром сертифікації. Якщо ви налаштуєте керування доступом на сервері на використання методу SSL для автентифікації користувачів і груп, клієнту потрібно буде

- Надати дійсний сертифікат, виданий довіреним центром сертифікації
- Сертифікат повинен бути зіставлений з дійсним користувачем в LDAP
- Список управління доступом повинен оцінюватися належним чином

Якщо вам потрібна автентифікація клієнта з контролем доступу, вам потрібно увімкнути SSL-шифри на вашому веб-сервері.

Щоб успішно отримати доступ до ресурсу, автентифікованого за допомогою SSL, клієнтський сертифікат повинен бути виданий центром сертифікації, якому довіряє веб-сервер. Сертифікат клієнта повинен бути опублікований на сервері каталогів, якщо файл certmap.conf веб-сервера налаштований на порівняння сертифіката клієнта в браузері з сертифікатом клієнта на сервері каталогів. Однак файл certmap.conf можна налаштувати так, щоб він порівнював лише вибрану інформацію з сертифіката із записом на сервері каталогів. Наприклад, ви можете налаштувати файл certmap.conf на порівняння лише ідентифікатора користувача та адреси електронної пошти у сертифікаті браузера із записом на сервері каталогів.

Digest автентифікація

Сервер можна налаштувати на виконання дайджест-автентифікації за допомогою служби каталогів на основі LDAP або файлової служби каталогів.

Дайджест-аутентифікація дозволяє користувачеві пройти автентифікацію на основі імені користувача та пароля без надсилання імені користувача та пароля у вигляді відкритого тексту. Браузер використовує алгоритм MD5 для створення дайджесту, використовуючи пароль користувача і деяку інформацію, надану веб-сервером.

Коли сервер використовує службу каталогів на основі LDAP для виконання дайджест-аутентифікації, це значення дайджесту також обчислюється на стороні сервера за допомогою плагіна Digest Authentication і порівнюється зі значенням дайджесту, наданим клієнтом. Якщо значення дайджесту збігаються, користувач автентифікований. Для того, щоб це спрацювало, серверу каталогів потрібен доступ до пароля користувача у відкритому вигляді. Oracle Directory Server Enterprise Edition включає в себе плагін оборотного пароля, який використовує симетричний алгоритм шифрування для зберігання даних у зашифрованому

вигляді, який пізніше можна розшифрувати до початкової форми. Ключ до даних зберігається лише на сервері каталогів.

Для автентифікації дайджесту на основі LDAP вам потрібно ввімкнути плагін зі змінним паролем і спеціальний плагін для автентифікації дайджесту, що входить до складу сервера. Щоб налаштувати веб-сервер на обробку дайджест-автентифікації, встановіть властивість digestauth у визначенні бази даних у файлі dbswitch.conf.

Якщо ви не вкажете метод ACL, сервер використовуватиме digest або basic, якщо автентифікація потрібна, або basic, якщо автентифікація не потрібна. Перевагу слід надавати методу digest.

Table 7–1 Digest Authentication Challenge Generation

ACL Method	Digest Authentication Supported by Authentication Database	Digest Authentication Not Supported by Authentication Database
“default” none specified	digest and basic	basic
“basic”	basic	basic
“digest”	digest	ERROR

При обробці ACL з методом = digest, сервер намагається автентифікуватись за:

- Перевірка заголовка запиту на авторизацію. Якщо його не знайдено, генерується відповідь 401 з викликом Digest, і процес зупиняється.
- Перевірка типу Авторизації. Якщо тип автентифікації - Digest, то сервер перевіряє:
    - Перевіряє nonce. Якщо не дійсний, свіжий nonce, згенерований цим сервером, генерується відповідь 401, і процес зупиняється. Якщо

застарілий, генерується відповідь 401 зі значенням stale=true, і процес зупиняється.

- Ви можете налаштувати час, протягом якого nonce залишається свіжим, змінивши значення параметра DigestStaleTimeout у файлі magnus.conf, розташованому в каталозі server_root/https-server_name/config/. Щоб встановити значення, додайте в magnus.conf наступний рядок:

  DigestStaleTimeout seconds

- де seconds - кількість секунд, протягом яких nonce залишатиметься свіжим. Після закінчення вказаних секунд nonce стає недійсним, і від користувача вимагається нова автентифікація.
- Перевіряє домен. Якщо домен не збігається, генерує відповідь 401, і процес зупиняється.
- Перевіряє наявність користувача у каталозі LDAP, якщо каталог автентифікації базується на LDAP, або перевіряє наявність користувача у файловій базі даних, якщо каталог автентифікації базується на файлах. Якщо користувача не знайдено, генерує відповідь 401, і процес зупиняється.
- Отримує значення запиту-дайджесту з сервера каталогів або файлової бази даних і перевіряє на збіг з клієнтським запитом-дайджестом. Якщо не знайдено, генерує відповідь 401 і процес зупиняється.
- Створює заголовок Authorization-Info і вставляє його в заголовки сервера.

Налаштування кешу користувачів ACL

За замовчуванням сервер зберігає результати автентифікації користувачів і груп у кеші користувачів ACL. Ви можете контролювати час, протягом якого кеш користувача ACL є дійсним, за допомогою директиви ACLCacheLifetime у файлі magnus.conf. Кожного разу, коли відбувається посилання на запис у кеші, обчислюється його вік і перевіряється за допомогою ACLCacheLifetime. Запис не використовується, якщо його вік більший або дорівнює ACLCacheLifetime. Значення за замовчуванням - 120 секунд. Встановлення значення 0 (нуль) вимикає кеш. Якщо ви використовуєте велике число для цього значення, вам може знадобитися перезавантаження сервера кожного разу, коли ви вносите зміни до записів LDAP. Наприклад, якщо це значення встановлено в 120 секунд, сервер може бути не синхронізований з каталогом LDAP протягом двох хвилин. Встановлюйте велике значення лише у тому випадку, якщо ваш каталог LDAP не змінюватиметься часто.

За допомогою параметра magnus.conf ACLUserCacheSize ви можете налаштувати максимальну кількість записів, які можуть зберігатися у кеші. За замовчуванням цей параметр має значення 200. Нові записи додаються до початку списку, а записи у кінці списку переробляються для створення нових записів, коли кеш досягає максимального розміру.

Ви також можете задати максимальну кількість членств у групах, яку можна кешувати для одного запису користувача, за допомогою параметра magnus.conf, ACLGroupCacheSize. Значення за замовчуванням для цього параметра дорівнює 4. На жаль, якщо користувач не є членом групи, він не кешується, і це призведе до декількох звернень до каталогів LDAP при кожному запиті.

Для отримання додаткової інформації про директиви файлів ACL див. Посібник розробника NSAPI.

Налаштування властивостей кешу ACL

Щоб налаштувати властивості кешу ACL за допомогою CLI, виконайте наступну команду.

```
wadm> set-acl-cache-prop --user=admin --password-file=admin.pwd --host=serverhost
--port=8989 --config=config1 property=value
```

Допустимі властивості, які ви можете встановити::

- enabled - вказує, чи кешує сервер вміст файлів і метаінформацію. Значення за замовчуванням - true.
- max-age - Максимальний час (у секундах) для кешування вмісту файлу та метаінформації. Діапазон значень від 0.001 до 3600.
-
- max-groups-per-user - Максимальна кількість груп для одного користувача, для яких сервер буде кешувати інформацію про членство. Діапазон значень від 1 до 1024.
-
- max-age - Максимальний час (у секундах), протягом якого сервер буде зберігати інформацію про автентифікацію. Діапазон значень від 0.001 до 3600.

Налаштування контролю доступу

Сервер підтримує автентифікацію та авторизацію за допомогою локально збережених списків управління доступом (ACL), які описують, які права доступу має користувач до ресурсу. Наприклад, запис в ACL може надати користувачеві на ім'я John право на читання певної папки misc.

У цьому розділі описано процес обмеження доступу до файлів або каталогів на вашому веб-сайті. Ви можете встановити глобальні правила контролю доступу для всіх серверів, а також індивідуально для певних серверів. Наприклад, відділ кадрів може створити списки ACL, які дозволять усім автентифікованим користувачам переглядати власні дані про заробітну плату, але обмежать доступ до оновлення даних лише працівникам відділу кадрів, відповідальним за нарахування заробітної плати.

Основні списки ACL, що підтримуються сервером, включають три типи автентифікації: базову, SSL і дайджест.

Щоб відредагувати налаштування контролю доступу, виконайте такі дії:

1. Перейдіть на вкладку Конфігурації і виберіть потрібну конфігурацію.
2. Перейдіть на вкладку Безпека > вкладка Контроль доступу.
3. Натисніть кнопку Додати ACL, щоб додати новий ACL, або клацніть існуючий ACL, щоб змінити налаштування.

Додавання списку контролю доступу (ACL)

У наступному розділі описано процес додавання нового ACL до конфігурації.

1. Перейдіть на вкладку Конфігурації і виберіть конфігурацію.
2. Перейдіть на вкладку Контроль доступу > вкладка Списки контролю доступу.
3. Натисніть кнопку Створити, щоб додати новий ACL.

Налаштуйте такі параметри:

Таблиця 7-2 Параметри ACL

Parameter	Description
Resource	Named/URI/Path. Select the type of resource you need to set access restriction and specify the value. Example for URI resource — “/sales”. Example for Path resource — “/opt/oracle/webserver7/docs/cgi-bin/*”.
Authentication DB	дозволяє вибрати базу даних, яку сервер використовуватиме для автентифікації користувачів. За замовчуванням використовується ключовий файл
Authentication Method	1. Базовий - використовує метод HTTP Basic для отримання автентифікаційної інформації від клієнта. Ім'я користувача та пароль шифруються в мережі, тільки якщо на сервері ввімкнено SSL. 2. SSL - використовує клієнтський сертифікат для автентифікації користувача. Для використання цього методу на сервері має бути ввімкнено SSL. Якщо шифрування ввімкнено, ви можете комбінувати базовий метод і метод SSL. 3. Дайджест - використовує механізм автентифікації, який дозволяє браузеру автентифікуватися на основі імені користувача та пароля без надсилання імені користувача та пароля у вигляді відкритого тексту. Браузер використовує алгоритм MD5 для створення значення дайджесту, використовуючи пароль користувача і деяку інформацію, надану веб-сервером. Зауважте, що для того, щоб використовувати дайджест, базова база даних auth-db також повинна підтримувати дайджест. Це означає, що має бути присутня або файлова база даних з використанням digestfile, або база даних LDAP, якщо встановлено плагін Digest Authentication Plug-in.

	4. Інше - використовує власний метод, створений за допомогою API контролю доступу.
Prompt for Authentication	Опція **Запит на автентифікацію** дозволяє вам ввести текст повідомлення, яке з'явиться у діалоговому вікні автентифікації. Ви можете використовувати цей текст для опису того, що користувач повинен ввести. Залежно від браузера, користувач побачить перші 40 символів запиту. Веб-браузери зазвичай кешують ім'я користувача та пароль і пов'язують їх з текстом запиту. Коли користувач отримує доступ до файлів і каталогів сервера за допомогою того самого запиту, імена користувачів і паролі не потрібно буде вводити знову. Якщо ви хочете, щоб користувачі проходили повторну автентифікацію для доступу до певних файлів і каталогів, вам просто потрібно змінити запит для ACL на цьому ресурсі.
Denied Access Response	Вказати дію у відповідь на відмову в доступі до ресурсу. 1. Відповідати стандартним повідомленням - виберіть цю опцію, щоб відобразити стандартне повідомлення про відмову в доступі від сервера. 2. Відповідати за URL-адресою - виберіть цю опцію, щоб перенаправити запит на будь-яку іншу зовнішню URL-адресу або сторінку помилки.

Використання файлу .htaccess

Сервер підтримує динамічні конфігураційні файли .htaccess. Ви можете ввімкнути файли .htaccess або через інтерфейс користувача, або вручну змінивши конфігураційні файли.

Ви можете використовувати файли .htaccess у поєднанні зі стандартним управлінням доступом на сервері. Стандартні засоби керування доступом завжди застосовуються перед будь-якими засобами керування доступом .htaccess, незалежно від порядку розташування директив PathCheck. Не вимагайте автентифікації користувачів як за допомогою стандартних, так і за допомогою .htaccess, якщо автентифікація групи користувачів має значення "Базова". Використовуйте автентифікацію SSL-клієнта за допомогою стандартного контролю доступу до сервера, а також вимагайте автентифікацію HTTP "Базова" за допомогою файлу .htaccess.

Якщо ви ввімкнули файли .htaccess, сервер перевіряє наявність файлів .htaccess перед тим, як обслуговувати ресурси. Сервер шукає файли .htaccess у тому ж каталозі, що і ресурс, а також у батьківських каталогах цього каталогу, аж до кореня документа включно. Наприклад, якщо каталог основного документа встановлено у /oracle/server/docs і клієнт запитує /oracle/server/docs/reports/index.html, сервер перевірить наявність файлів .htaccess у каталогах /oracle/server/docs/reports/.htaccess і /oracle/server/docs/.htaccess.

Зауважте, що функціональність додаткових каталогів документів і каталогів CGI на сервері дозволяє адміністратору визначати альтернативні корені документів. Існування альтернативних коренів документів впливає на обробку файлів .htaccess. Наприклад, розглянемо сервер з основним каталогом документів у /oracle/server/docs і CGI-програмою в /oracle/server/docs/cgi-bin/program.cgi. Якщо ви ввімкнете CGI як тип файлу, сервер оцінюватиме вміст файлів /oracle/server/docs/.htaccess і /oracle/server/docs/cgi-bin/.htaccess, коли клієнт надсилатиме запит до CGI-програми. Однак, якщо ви замість цього налаштуєте каталог CGI в /oracle/server/docs/cgi-bin, сервер перевірить /oracle/server/docs/cgi-bin/.htaccess, але не /oracle/server/docs/.htaccess. Це відбувається тому, що вказівка /oracle/server/docs/cgi-bin як каталогу CGI позначає його як альтернативний корінь документа.

Запобігання DoS-атакам

Атака на відмову в обслуговуванні (DoS) - це явна спроба зловмисників перешкодити законним користувачам користуватися послугами сервера. Така атака може бути здійснена шляхом надсилання безперервних запитів до сервера на певний веб-ресурс.

Web Server 7.0.9 може виявити DoS-атаку, відстежуючи часто відвідувані URI і відхиляючи запити, якщо частота запитів висока.

У наступних розділах описано, як запобігти DoS-атакам на рівні віртуального сервера.

Контроль доступу також допомагає:

- Обмежити запити до сервера
- Обмежити максимальну кількісті з'єднань
- Запобігти атакам Cross Site Scripting

Висновок

У ході даної лабораторної роботи було розглянуто описано CryptoAPI та PKCS#11. А також розглянуто модель контролю доступу до ключової інформації для мережі “клієнт-сервер” із серверної частини.