ЛЕКЦІЯ 12. ЗАХИСТ ІНФОРМАЦІЇ В ОПЕРАЦІЙНИХ СИСТЕМАХ

План

18.1. Основні завдання забезпечення безпеки
18.2. Базові поняття криптографії
18.3. Принципи аутентифікації і керування доступом
18.4. Аутентифікація та керування доступом в UNIX
18.5. Аутентифікація і керування доступом у Windows XP

18.1. Основні завдання забезпечення безпеки

Забезпечення безпеки комп'ютерних систем вимагає виконання комплексу завдань, найважливішими серед яких є виконання процедур аутентифікації, авторизації та аудиту, дотримання конфіденційності, доступності та цілісності даних.

Аутентифікація

Аутентифікація (authentication)— це процес, за допомогою якого одна сторона (система) засвідчує, що інша сторона (користувач) є тим, за кого себе видає. Під час аутентифікації потрібне свідчення (credentials), що найчастіше складається з інформації, відомої обом сторонам (наприклад, ним може бути пароль). Користувач, що пред'явив коректне свідчення, дістає позитивну відповідь на вимогу аутентифікації.

Авторизація

Після того як аутентифікація відбулась успішно, користувач починає працювати із системою. Тепер йому може знадобитися доступ до різних ресурсів. *Авторизація* (authorization), або *керування доступом* (access control) — це процес, за допомогою якого перевіряють, чи має право користувач після успішної аутентифікації отримати доступ до запитаних ним ресурсів.

Авторизацію здійснюють порівнянням інформації про користувача з інформацією про права доступу, пов'язаною із ресурсом. Зазвичай вона містить тип дії та відомості про користувачів, яким дозволено цю дію виконувати. Якщо користувач має право на запитану дію із цим ресурсом, йому надають можливість її виконати.

Аудит

Під *аудитом* (auditing) розуміють збирання інформації про різні події у системі, важливі для її безпеки, і збереження цієї інформації у формі, придатній для подальшого аналізу. Подіями можуть бути успішні та безуспішні спроби аутентифікації у системі, спроби отримати доступ до об'єктів тощо. Інформацію звичайно зберігають у спеціальному системному журналі (system log). У деяких системах цей журнал має вигляд текстового файла, інші підтримують його спеціальний формат.

Конфіденційність, цілісність і доступність даних

Основним компонентом політики безпеки комп'ютерних систем є дотримання найважливіших характеристик даних.

Конфіденційність (data confidentiality) — можливість приховання даних від стороннього доступу. її зазвичай забезпечують криптографічним захистом даних за допомогою їхнього шифрування.

Цілісність (data integrity) — спроможність захистити дані від вилучення або зміни (випадкової чи навмисної). Технології підтримки цілісності даних також пов'язані із криптографічними методами, вони включають цифрові підписи і коди аутентифікації повідомлень.

Доступність (data availability) — гарантія того, що легітимний користувач після аутентифікації зможе отримати доступ до запитаного ресурсу, якщо він має на це право. Порушення доступності даних називають *відмовою від обслуговування* (denial of service), однієї із цілей політики безпеки є запобігання випадковому або навмисному доведенню системи до такої відмови.

Про ці завдання докладніше йтиметься далі.

18.2. Базові поняття криптографії

Припустимо, що одна особа збирається відіслати іншій особі повідомлення і хоче бути впевненою в тому, що ніхто інший не зможе його прочитати навіть тоді, коли воно буде перехоплене.

У криптографії повідомлення називають *вихідним текстом* (plaintext), зміну вмісту повідомлення так, що воно стає недоступним для сторонніх, — *шифруванням* (encryption), зашифроване повідомлення — *шифрованим текстом* (ciphertext), а процес отримання вихідного тексту із шифрованого — *дешифруванням* (decryption).

18.2.1. Поняття криптографічного алгоритму і протоколу

Криптографічні алгоритми

Криптографічний алгоритм, який ще називають *шифром* (cipher), — це математична функція, яку використовують для шифрування і дешифрування.

Сучасні криптографічні алгоритми проблему безпеки вирішують виключно за допомогою *ключа* (key). Ключ — це значення, яке використовують для шифрування і дешифрування, при цьому без знання ключа дешифрування має бути технічно неможливим (тобто неможливим за наявності будь-яких доступних ресурсів). Ключ має належати до множини зі значною кількістю елементів: її розмір повинен бути таким, щоб забезпечити технічну неможливість підбору ключа повним перебором усіх елементів цієї множині.

Сукупність алгоритму і множини всіх можливих вихідних текстів, шифрованих текстів і ключів називають *криптосистемою.*

Криптографічні алгоритми, засновані на ключі, поділяють на дві основні групи: *симетричні* (або *алгоритми із секретним ключем)* і *алгоритми із відкритим ключем.* Далі докладніше про особливості цих двох груп.

Криптографічні протоколи

Криптографічний протокол — це послідовність кроків, розроблена для виконання деякої задачі із забезпечення криптографічного захисту даних.

Такий протокол обов'язково включає дві або більше сторін і розглядається як обмін діями між сторонами. Для позначення сторін прийнято використовувати такі імена: *Аліса* — учасник, що робить перший крок; *Боб* — учасник, що робить крок у відповідь.

На окремих етапах протоколів використовують різні криптографічні алгоритми.

18.2.2. Криптосистеми з секретним ключем

Криптографічні алгоритми з секретним ключем

Алгоритми із секретним ключем або симетричні алгоритми — це такі алгоритми, для яких ключ для шифрування повідомлення збігається із ключем для його дешифрування.

Наведемо деякі приклади симетричних алгоритмів

Алгоритм *DES* (Data Encryption Standard) був прийнятий як національний стандарт США1977 року. Головним його недоліком є довжина ключа (56 біт), що робить DES недостатньо надійним у сучасних умовах (ключ може бути знайдений за скінчений час перебором усіх можливих ключів). Як альтернативу сьогодні використовують *потрійний DES* (послідовне шифрування трьома різними ключами).

У 2001 році було прийнято новий стандарт на симетричний криптографічний алгоритм. Він (як і сам алгоритм) дістав назву *AES* (Advanced Encryption Standard). За довжиною ключ AES значно перевищує DES (стандарт визначає ключі на 128, 192 і 256 біт), крім того, він відрізняється високою продуктивністю.

Обмін повідомленнями із використанням криптографії із секретним ключем

Подамо у вигляді протоколу обмін повідомленнями у разі використання криптографічного алгоритму із секретним ключем.

1. Аліса і Боб погоджуються, що вони використовуватимуть систему із секретним ключем.

2. Вони доходять згоди щодо спільного ключа.

3. Аліса шифрує вихідний текст ключем і посилає його Бобові.

4. Боб розшифровує повідомлення тим самим ключем.

Другий крок потрібно робити секретно через деякий альтернативний канал передавання даних. Якщо Аліса і Боб не можуть використати такий канал, вони змушені передавати ключ незашифрованим, інакше інша сторона не зможе ним скористатися. Якщо при цьому ключ перехопить зловмисник, то він далі зможе читати всю інформацію, яку передають каналом.

Цю проблему неможливо вирішити, залишаючись у межах традиційних симетричних алгоритмів. Потрібен принципово інший підхід. Він отримав назву *криптографії із відкритим ключем* (public key cryptography).

18.2.3. Криптосистеми із відкритим ключем

Криптографічні алгоритми з відкритим ключем

Алгоритми з відкритим ключем розроблено таким чином, що ключ, використаний для шифрування, відрізняється від ключа для дешифрування. Ці два ключі працюють у парі: текст, який шифрують одним ключем, дешифрують іншим відповідно до формул

Ці алгоритми так називаються тому, що один із цих ключів може бути відкритий для всіх (наприклад, опублікований у пресі), даючи змогу будь-якій особі шифрувати цим ключем, але таке повідомлення може прочитати тільки справжній адресат (що володіє другим ключем).

Ключ, що відкривають для інших осіб для виконання шифрування, називають відкритим ключем (public key), парний до нього ключ для дешифрування — закритим ключем (private key).

Алгоритми із секретним ключем працюють швидше за алгоритми з відкритим ключем, тому, якщо не потрібні специфічні властивості, забезпечувані відкритим ключем (наприклад, не передбачене пересилання даних відкритим каналом зв'язку), достатньо обмежитися алгоритмом із секретним ключем.

Найвідомішим алгоритмом із відкритим ключем є *RSA.*

Обмін повідомленнями з використанням криптографії із відкритим ключем

Обмін повідомленнями у разі використання криптографії з відкритим ключем наведемо у вигляді протоколу

1.Аліса і Боб погоджуються використовувати систему з відкритим ключем.

2.Боб надсилає Алісі свій відкритий ключ.

3.Аліса шифрує повідомлення відкритим ключем Боба і відсилає його Бобові.

4. Боб розшифровує це повідомлення своїм закритим ключем.

Ця послідовність кроків позбавлена недоліків, властивих для процедури обміну секретним ключем. Перехоплення відкритого ключа зловмисником не спричиняє порушення безпеки каналу, а закритий ключ між сторонами не передають

18.2.4. Гібридні криптосистеми

На практиці алгоритми із відкритим ключем не можуть цілковито замінити алгоритми із секретним ключем, насамперед через те, що вони працюють значно повільніше, і шифрувати ними великі обсяги даних неефективно.

Для того щоб об'єднати переваги двох категорій алгоритмів, використовують гібридні криптосистеми, де алгоритми із відкритим ключем використовують для шифрування не самих повідомлень, а ключів симетричних алгоритмів, якими далі (після обміну) шифрують весь канал. Такі ключі називають сесійними ключами, бо їх звичайно створюють за допомогою генераторів випадкових чисел для конкретної сесії.

Протокол роботи гібридної криптосистеми наведемо нижче.

1.Боб відсилає Алісі свій відкритий ключ.

2.Аліса генерує випадковий сесійний ключ, шифрує його відкритим ключем Боба і відсилає Бобові.

3.Боб розшифровує сесійний ключ своїм закритим ключем.

4.Обидві сторони далі обмінюються повідомленнями, зашифрованими сесійним ключем.

Гібридні криптосистеми забезпечують якісний захист ключів і прийнятну продуктивність. Про різні застосування цієї технології йтиметься далі.

18.2.5. Цифрові підписи

Ще одна галузь застосування криптографії із відкритим ключем — *цифрові підписи.* Протокол створення і перевірки цифрового підпису при цьому складається з таких кроків.

1. Аліса шифрує документ своїм закритим ключем, тим самим підписуючи його.

2. Аліса відсилає шифрований документ Бобові.

3. Боб розшифровує документ відкритим ключем Аліси, підтверджуючи цифровий підпис.

Цей протокол забезпечує підтримку основних характеристик цифрових підписів, до яких належить, зокрема, неможливість приховати зміну підписаного документа. У цьому разі спроба змінити документ без використання закритого ключа призводить до того, що документ не дешифрується після перевірки підпису.

Алгоритми із відкритим ключем, однак, неефективні для підписування повідомлень великого обсягу. Щоб підвищити продуктивність, цифрові підписи реалізують із використанням односторонніх хеш-функцій.

Односторонні хеш-функції

Хеш-функція — функція, що приймає на вхід рядок змінної довжини, який називають *вхідним образом,* а повертає рядок фіксованої (звичайно меншої) довжини — *хеш.*

За значенням *односторонньої функції* важко віднайти аргумент. *Одностороння хеш-функція* також працює в одному напрямку: легко отримати хеш із вихідного рядка, але технічно неможливо знайти вхідний образ, із якого походить цей хеш.

Односторонній хеш-функції властива свобода від колізій: технічно неможливо створити два вхідних образи з одним і тим самим значенням хеша.

Найрозповсюдженшшими односторонніми хеш-функціями є MD5 і SHA-1. Крім цифрових підписів, їх можна використати для розв'язання різних задач, які потребують такого відображення важливої інформації, що заслуговує на довіру.

Підписи із відкритим ключем і односторонніми хеш-функціями

У разі використання односторонніх хеш-функцій для цифрових підписів замість документа підписують його хеш (значно менший за обсягом). Протокол набуває такого вигляду.

1.Аліса отримує односторонній хеш документа.

2.Аліса шифрує хеш своїм закритим ключем, тим самим підписуючи документ.

3.Аліса відсилає Бобові документ і зашифрований хеш.

4.Боб розшифровує хеш, переданий Алісою, її відкритим ключем.

5. Боб отримує односторонній хеш переданого Алісою документа.

6. Боб порівнює хеші, отримані під час виконання кроків 4 і 5. Якщо вони збігаються, підпис Аліси можна вважати вірним.

Цей протокол грунтується на тому, що підпис хеша можна прирівняти до підпису документа, що випливає із властивості свободи від колізій односторонніх хеш-функцій (неможливо створити два документи, які були б перетворені на один хеш).

18.2.6. Сертифікати

Дотепер ми розглядали обмін повідомленнями між двома сторонами. На практиці значно частіше трапляється ситуація, коли ціла низка сторін домовляються про використання криптографії із відкритим ключем для обміну повідомленнями. У цьому разі доцільно розміщувати відкриті ключі кожної зі сторін у спеціальній базі даних.

Основна проблема при цьому пов'язана із базою даних, у якій зберігають відкриті ключі. Вона повинна мати такі властивості.

Можливість читати з цієї бази даних може мати будь-який користувач.

У неї не може записувати дані жоден користувач, за винятком деякої довіреної сторони. У протилежному випадку зловмисник зможе записати у базу свій відкритий ключ поверх ключа легітимного користувача і читати всі повідомлення, адресовані йому.

Водночас, навіть якщо база даних і має такі властивості, зловмисник під час передавання даних може підмінити відкритий ключ Боба своїм ключем. Для того щоб цього уникнути, довірена сторона може поставити цифровий підпис на кожен відкритий ключ.

У реальних застосуваннях довірена сторона (центр сертифікації, Certification Authority, CA) підписує не відкритий ключ, а *сертифікат* — документ, що складається із відкритого ключа та інформації, яка ідентифікує його власника. Після підписання сертифікати заносяться у базу даних.

На формат сертифікатів є стандарти, наприклад X.509.

18.3. Принципи аутентифікації і керування доступом

Цей розділ присвячено особливостям реалізації аутентифікації і контролю доступу в сучасних операційних системах.

18.3.1. Основи аутентифікації

Аутентифікація надає можливість розрізняти легітимні та нелегітимні спроби доступу до системи. Надійна аутентифікація дає змогу у багатьох випадках обмежити коло потенційних порушників легітимними користувачами системи, спрощуючи цим процедури забезпечення її безпеки.

Свідчення, які вимагаються від користувачів під час аутентифікації, найчастіше зводяться до знання секретної інформації, спільної для користувача і системи (наприклад, пароля). Саме про таку аутентифікацію і йтиметься в цьому розділі.

Розрізняють локальну і мережну аутентифікацію. У разі успішної локальної аутентифікації користувач доводить свою легітимність для використання ресурсів однієї комп'ютерної системи (свідчення користувача перевіряють локально), мережна аутентифікація дає змогу користувачу довести легітимність для використання всіх ресурсів мережі (свідчення користувача передають для перевірки на спеціальний сервер із будь-якого комп'ютера мережі).

Облікові записи

Для того щоб аутентифікація користувача була можлива, у системі має зберігатись інформація про цього користувача. Таку інформацію називають *обліковим записом* (account). Із ним звичайно пов'язують такі дані:

- ім'я користувача, яке він вказує для входу у систему;
- ідентифікатор користувача, що зазвичай є чисельним значенням, унікальним у межах комп'ютера або групи комп'ютерів (цей ідентифікатор ОС використовує під час аутентифікації і авторизації);
- інформація про пароль користувача;
- інформація про обмеження на вхід користувача у систему (термін легітимності облікового запису, періодичність зміни пароля, години і дні тижня, у які користувач може отримувати доступ у систему тощо);
- інформація про групи, до яких належить цей користувач;
- місце знаходження домашнього каталогу користувача (у якому він може створювати свої файли);
- налаштування сесії користувача (шлях до його командного інтерпретатора тощо).

Інформацію про облікові записи зберігають у *базі даних облікових записів* (account database). Адміністратор системи може змінювати будь-яку інформацію в цій базі, для інших користувачів звичайно доступна лише зміна їхнього власного пароля.

Групи користувачів

У сучасних ОС для зручності адміністрування системи користувачі можуть об'єднуватись у групи. Користувач може одночасно належати до кількох груп. Під час авторизації доступу до об'єктів перевіряють не тільки права самого користувача, але й права груп, до яких він належить.

Інформацію про групи також зберігають у базі даних облікових записів. Звичайно ОС визначає кілька стандартних груп, які створюють під час її установки, зокрема, групу адміністраторів системи (які можуть виконувати в ній будь-які дії) і групу звичайних користувачів із обмеженим доступом.

Аутентифікація з використанням односторонніх функцій

Для перевірки пароля немає потреби знати цей пароль, досить уміти відрізняти правильний пароль від неправильного. Тому замість зберігання паролів доцільно зберігати односторонні функції цих паролів. Подивимося, як виглядатиме в даному випадку протокол аутентифікації.

1. Аліса посилає системі свої ім'я і пароль.

2. Система обчислює односторонню функцію від пароля.

3. Система порівнює результат обчислення односторонньої функції зі значенням, що зберігається у базі даних облікових записів.

У результаті зменшуються втрати, які може задати зловмисник, коли отримає доступ до списку паролів, оскільки навіть у цьому разі за односторонньою функцією відновити паролі неможливо. Проте цей підхід не позбавлений недоліків.

Словникові атаки і сіль

Якщо зловмисник володіє списком паролів, зашифрованих односторонньою функцією, можлива словникова атака. Зловмисник бере набір найпоширеніших паролів, застосовує до них односторонню функцію і зберігає всі зашифровані паролі. Потім він порівнює список зашифрованих паролів із цим файлом (словником) у пошуках збігів.

Один зі способів боротьби із такою атакою пов'язаний із використанням *солі* (salt). Сіль — це випадковий рядок *S*, який додають до пароля перед шифруванням. У список шифрованих паролів заноситься рядок $S + E(S + P)$, де P — пароль, E — функція шифрування, «+» — конкатенація рядків. Якщо кількість можливих значень солі достатньо велика, то це робить словникову атаку значно складнішою, оскільки у словник потрібно вносити результати шифрування паролів із усіма можливими значеннями солі.

Солі потрібно досить багато. Наприклад, стандартний її обсяг, прийнятий в UNIX (12 біт, що дає 4096 можливих значень), є не зовсім достатнім (є словники найуживаніших паролів, об'єднані з усіма значеннями солі).

Аутентифікація за принципом «виклик-відповідь»

Більш серйозна проблема, пов'язана із використанням описаного підходу, полягає в тому, що пароль передають мережею незашифрованим, і він може бути перехоплений зловмисником. Один зі способів вирішення цієї проблеми полягає в тому, щоб передавати мережею не паролі, а їх односторонні хеші (дайджести). Цей підхід називають аутентифікацією за принципом «виклик-відповідь» (challenge-response authentication) або дайджест-аутентифікацією

Цей протокол був основним підходом до аутентифікації у системах лінії Windows XP до появи Windows 2000 і дотепер підтримується у цих системах

Більш складним протоколом аутентифікації є *протокол Kerberos*. Це розподілена система аутентифікації користувачів із можливістю аутентифікації клієнта і сервера. Протокол Kerberos є основним протоколом мережної аутентифікації у системах лінії Windows XP, починаючи із Windows 2000. Реалізація цього протоколу доступна і для UNIX-систем.

Одноразові паролі

Проблему пересилання пароля мережею можна також розв'язати, використовуючи паролі, дійсні лише один раз під час сесії користувача. Перехоплення такого *одноразового пароля* (one-time password) нічого не дає зловмисникові.

Такі паролі у сучасних системах можуть реалізовуватися за допомогою смарт-карт — електронних пристроїв, у які вбудований мікропроцесор із засобами генерації відповідних одноразових паролів.

13.3.2. Основи керування доступом

Для реалізації керування доступом операційна система має можливість визначати, що за дії і над якими об'єктами має право виконувати той чи інший користувач системи. Звичайний розподіл прав відображають у вигляді *матриці доступу,* у якій рядки відповідають суб'єктам авторизації (користувачам, групам користувачів тощо), стовпці — ресурсам (файлам, пристроям тощо), а на перетині рядка і стовпця зазначені права цього суб'єкта на виконання операцій над даним ресурсом.

Зберігання повної матриці контролю доступу неефективне (вона займатиме багато місця), тому звичайно використовують два базові підходи для її компактного відображення.

У разі реалізації ***списків*** *контролю доступу* (Access Control Lists, ACL) зберігаються стовпці матриці. Для кожного ресурсу задано список суб'єктів, які можуть використовувати цей ресурс.

У разі реалізації *можливостей* (capabilities) зберігаються рядки матриці. Для кожного суб'єкта задано список ресурсів, які йому дозволено використовувати.

Списки контролю доступу

Якщо реалізовано ACL, для кожного об'єкта задають список, що визначає, які користувачі можуть виконувати ті чи інші операції із цим об'єктом. Наприклад, для файлових систем такими об'єктами є файли або каталоги. У найзагальнішій формі елементи цього списку — це пари (користувач, набір цій), які називають *елементами контролю доступу* (access control elements, ACE).

Розглянемо деякі проблеми, які потрібно вирішити у разі реалізації списків контролю доступу.

♦ Розмір списку контролю доступу має бути не надто великим, щоб система могла ефективно ним керувати.

♦ Права мають бути досить гнучкими, але при цьому не дуже складними. Надмірне ускладнення системи прав звичайно призводить до того, що користувачі починають її «обходити», задаючи спрощені права; у результаті загальна безпека не підвищується, а ще більше знижується.

Вирішення цих проблем призвело до особливостей реалізації списків контролю доступу в різних ОС. Так, загальною концепцією у створенні списків є задання прав не лише для окремих користувачів, але й для груп користувачів, а також можливість визначити права доступу всіх користувачів системи. У деяких ОС обмежують кількість елементів у списку (наприклад, в UNIX список, як незабаром побачимо, обмежений трьома елементами). Водночас у системах лінії Windows XP можна задавати списки контролю доступу, що складаються з необмеженої кількості елементів (для окремих користувачів, груп, користувачів інших комп'ютерів тощо), і задавати різні комбінації прав — від узагальнених до детальних.

Можливості

Під час визначення можливостей для кожного користувача задають, до яких файлів він може мати доступ і як саме. При цьому із користувачем пов'язують *список можливостей* (capability list), що складається із пар (об'єкт, набір дій).

Реалізація списків можливостей дає змогу забезпечити не тільки захист, але й задання імен. Можна зробити так, щоб кожний користувач бачив тільки ті файли, до яких у нього є доступ. Якщо при цьому значенням за замовчуванням є відсутність доступу, користувачі можуть починати роботу в «порожній» системі. Загалом можливості використовують у системах, де потрібно забезпечити більшу безпеку за рахунок деякого зниження зручності роботи.

Можливості можуть бути задані не тільки для користувачів, але й для окремих процесів. У результаті під час запуску кожного процесу можна визначити його права.

18.4. Аутентифікація та керування доступом в UNIX

У цьому розділі опишемо, як реалізувати аутентифікацію і керувати доступом в UNIX-системах на прикладі Linux.

18.4.1.Облікові записи користувачів

Перш ніж розглянути реалізацію аутентифікації в UNIX, зупинимося на концепції користувача цієї системи.

Користувачі та групи користувачів

Кожному користувачу в UNIX ставлять у відповідність обліковий запис (account), що характеризується іменем користувача та ідентифікатором (uid). Як ім'я користувача, так і його ідентифікатор мають бути унікальними в межах усієї системи. Крім цього, з обліковим записом користувача пов'язують його *домашній каталог* (home directory), у який він за замовчуванням може записувати дані. Після входу користувача у систему відбувається перехід у його домашній каталог.

Користувачі об'єднуються в групи. Кожна група характеризується іменем та ідентифікатором (gid).

Загалом процес виконують із правами того користувача, який запустив відповідний виконуваний файл.

Суперкористувач root

Користувач із uid, що дорівнює нулю (звичайно його називають root) має в UNIX особливий статус — він може виконувати у системі будь-які дії без обмежень. Такого користувача називають також *суперкористувачем.* Усі інші — *звичайні користувачі;* про те, як задають права доступу для них, йтиметься в розділі 18.4.3.

Наявність єдиного суперкористувача (принцип «все або нічого») вважають головною слабкістю системи безпеки UNIX, оскільки компрометація єдиного пароля root негайно призводить до того, що зловмисник отримує повний контроль над системою. Крім цього, процеси під час виконання не можуть бути обмежені якоюсь частиною прав суперкористувача. Фактично, процес, якому потрібна лише невелика частина таких прав, змушений виконуватися під керуванням root із повним контролем над системою.

18.4.2. Аутентифікація

Для стандартної аутентифікації UNIX використовують підхід із використанням односторонньої функції від пароля і солі, описаний у пункті 18.3.1. Такою функцією традиційно був алгоритм DES, яким шифрувався рядок, що складався із нулів, при цьому в ролі ключа використовували значення пароля, об'єднане із сіллю. У сучасних версіях UNIX замість DES часто використовують односторонню хеш-функцію (звичайно MD5).

За вхід користувача у систему відповідають дві утиліти: getty — ініціалізує термінал, видає підказку «login:» і приймає ім'я користувача, і login, що видає підказку «password:», приймає значення пароля, робить аутентифікацію і починає сесію користувача (переходить у домашній каталог і запускає командний інтерпретатор).

Тіньові паролі

У традиційних UNIX-системах до файла /etc/passwd мав доступ будь-який користувач ОС. А тому зловмисникові достатньо було отримати непривілейований доступ до системи, щоб почати словникову атаку на цей файл.

Для вирішення даної проблеми потрібно заборонити доступ до цього файла для всіх користувачів, крім root. Проте таке рішення не є прийнятним, оскільки в цьому файлі, крім інформації про паролі, містяться імена та ідентифікатори користувачів, розташування їхніх домашніх каталогів та інші дані, необхідні більшості застосувань. У результаті всі програми, яким потрібна така інформація, довелося б запускати із правами суперкористувача.

У сучасних UNIX-системах (зокрема в Linux) застосовують інший підхід — технологію *тіньових паролів* (shadow passwords). При цьому

інформацію про паролі переносять із /etc/passwd в окремий тіньовий файл паролів (який зазвичай називають /etc/shadow). До нього може звертатися тільки суперкористувач. Іншу шформащю (імена та ідентифікатори користувачів тощо) зберігають у /etc/passwd, що залишається доступним для всіх користувачів. Тіньові паролі дають змогу значно підвищити надійність схеми аутентифікації системи.

18.4.3. Керування доступом

Реалізація керування доступом до файлів у UNIX збереглася, майже не змінившись від ранніх версій системи. Фактично вона є скороченою реалізацією списків контролю доступу.

Основні принципи реалізації

Наведемо принципи реалізації керування доступом в UNIX.

1. Усі файли і каталоги в UNIX мають власника (owner), що належить до певної групи (group), і права доступу (permissions).

2. Розрізняють три категорії прав доступу: для читання, записування і виконання. Для звичайних файлів назви прав відповідають їхньому змісту (зазначимо, що ядро UNIX намагатиметься завантажити у пам'ять і виконати будь-який файл, на який у поточного користувача є права на виконання).

3. Для каталогів задають той самий набір прав, але вони відрізняються за змістом від прав для файлів:

♦ право читання для каталогу означає можливість отримання списку імен файлів, що містяться в ньому (тобто читання вмісту каталогу як файла);

♦ право записування в каталог означає можливість створення в каталозі нових файлів і вилучення наявних (тобто записування в каталог як у файл); зазначимо, що для вилучення файла прав на нього самого можна й не мати — достатньо прав на каталог, де він зберігається;

♦ право виконання означає можливість пошуку в каталозі, тобто доступу до окремих файлів.

Із цього випливає, що якщо для каталогу задаються права «тільки для виконання», то доступ до окремих файлів у ньому можна отримати, коли знати їхні імена, список же всіх файлів отримати буде неможливо. Таке задання прав використовують, аби сторонні особи не могли випадково виявити файли, які, однак, потребують спільного доступу. З іншого боку, якщо задати права «тільки для читання», можна переглядати список файлів каталогу, але доступ до окремих файлів стане неможливий.

4. Права кожної категорії задають окремо для власника файла, для користувачів його групи і для всіх інших користувачів усього дев'ять базових комбінацій прав; можна сказати, що із кожним файлом пов'язаний список контролю доступу із трьох елементів.

Недоліки схеми керування доступом UNIX

Описана схема керування доступом проста для розуміння, але багато в чому обмежена. Основне обмеження пов'язане із довжиною списку контролю доступу (3 елементи). Звідси виникають такі проблеми, як

неможливість надання конкретному користувачу прав доступу до файла без створення для нього окремої групи (що не можливо зробити без наявності прав адміністратора).

Останнім часом в UNIX-системах усе ширше використовують списки контролю доступу без обмеження кількості елементів (у Linux їхня реалізація стала доступна у ядрі 2.6).

18.5. Аутентифікація і керування доступом у Windows XP

18.5.1. Загальна архітектура безпеки

Архітектури безпеки Windows XP містять такі основні компоненти.

Процес реєстрації користувачів (logon process, winlogon) обробляє запити користувачів на реєстрацію у системі та приймає дані від них.

Менеджер аутентифікації (Local Security Authority Subsystem, LSASS) безпосередньо проводить аутентифікацію користувача. Цей компонент — центральний у підсистемі безпеки. Крім аутентифікації, він контролює політику аудиту, про яку йтиметься у розділі 18.6.

Менеджер облікових записів (Security Accounts Manager, SAM) підтримує базу даних облікових записів (базу даних SAM), що містить імена локальних користувачів і груп, а також паролі. Під час перевірки прав користувача SAM взаємодіє із менеджером аутентифікації.

Довідковий монітор захисту (Security Reference Monitor, SRM) перевіряє права користувача на доступ до об'єкта і виконує необхідну дію з об'єктом за наявності цих прав. Це єдиний компонент, що виконується в режимі ядра, він реалізує політику контролю доступу, визначену менеджером аутентифікації. SRM гарантує, що будь-який користувач або процес, що дістав доступ до об'єкта, має всі права на нього.

Ця архітектура реалізована у системах, що не використовують мережної аутентифікації. Для неї інформація про облікові записи має зберігатися не у базі даних SAM, а у спеціальному централізованому сховищі даних — *активному каталозі* (Active Directory). Під час мережної аутентифікації менеджер аутентифікації звертається до служби активного каталогу віддаленого комп'ютера, на якому розташований цей каталог. База даних SAM, однак, є у будь-якій установці ОС лінії Windows XP — у ній зберігають облікові записи для локальної аутентифікації. Подальший виклад торкатиметься локальної аутентифікації.

18.5.2. Аутентифікація

Windows XP вимагає, щоб кожному користувачу відповідав обліковий запис. Він пов'язаний із *профілем захисту,* який є набором інформації щодо контролю доступу (ім'я користувача, список його груп, пароль тощо). Профілі захисту зберігають у базі даних SAM і використовують для аутентифікації.

Розглянемо послідовність кроків реєстрації користувача у системі. Процес winlogon очікує введення від користувача. Потік цього процесу виявляє спробу користувача ввійти у систему (натискання комбінації клавіш Ctrl+Alt+Del) і пропонує йому ввести ім'я облікового запису та

пароль. При цьому для реалізації такого введення winlogon звертається до спеціальної DLL графічної ідентифікації та аутентифікації (GINA). Стандартне вікно введення даних користувача реалізоване у msgina.dll, програміст може встановити свою версію цієї динамічної бібліотеки, що реалізує альтернативний метод аутентифікації (наприклад, на основі смарт-карт або біометричних даних).

Дані від користувача передаються процесу winlogon, і аутентифікація із використанням бази даних SAM (за яку відповідає LSASS) відбувається відповідно до протоколу «виклик-відповідь». Можна використати і протокол Керберос (цей підхід тут не розглядатиметься).

Якщо аутентифікація пройшла успішно, створюють об'єкт, що унікальним чином визначає цього користувача в усіх його подальших діях. Цей об'єкт, який називають *маркером доступу* (access token), відіграє ключову роль у підсистемі захисту: він визначає, до яких системних ресурсів мають доступ потоки, створені цим користувачем.

Після успішної аутентифікації користувача LSASS створює процес і приєднує до нього маркер доступу користувача. Цей процес передають підсистемі Win32, що запускає в його адресному просторі застосування, визначене у реєстрі як оболонка системи (за замовчуванням ним є стандартна оболонка explorer.exe). Застосування формує візуальне відображення параметрів робочого столу для користувача.

Ідентифікатори безпеки

Із кожним користувачем і групою у Windows XP пов'язують унікальний *ідентифікатор безпеки* (Security Identifier, SID). Це цілочислове значення, що складається із заголовка і випадкової частини. Система безпеки звертається до користувачів і груп тільки за їхнім SID.

18.5.3. Керування доступом

Під час створення будь-якого об'єкта Windows XP, який може бути використаний більш як одним процесом (включаючи файли, поіменовані канали, синхронізаційні об'єкти тощо), йому присвоюють *дескриптор захисту* (security descriptor).

До найважливіших елементів дескриптора захисту належать:

- SID власника об'єкта (власник завжди може змінювати атрибути безпеки об'єкта, навіть якщо в нього немає прав на доступ до його даних);

- список контролю доступу (ACL), що визначає права доступу до об'єкта.

Кожний елемент списку контролю доступу (ACE) містить такі елементи:

- тип ACE (виділяють, зокрема, дозволяючі і забороняючі ACE);

- ідентифікатор безпеки (SID);

- набір прав доступу (читання, записування, повний контроль тощо).

Сума прав доступу, наданих окремими ACE, формує загальний набір прав доступу, наданих ACL.

Висновки

Основними завданнями забезпечення безпеки комп'ютерних систем є аутентифікація, керування доступом, аудит, забезпечення конфіденційності, доступності та цілісності даних.

Сучасні засоби підтримки безпеки ґрунтуються на таких криптографічних технологіях, як алгоритми із секретним і відкритим ключами, гібридні криптосистеми, цифрові підписи і сертифікати.

Аутентифікація дає змогу впевнитися, що користувач є тим, за кого себе видає, і може бути допущений у систему. Для підтвердження особи користувача звичайно використовують пароль. Сучасні технології аутентифікації дають змогу уникнути передавання пароля каналом зв'язку у незашифрованому вигляді.

Керування доступом (авторизація) дає змогу визначити дії, які авторизований користувач може виконувати із різними об'єктами у системі. Авторизація реалізована на основі визначення списків контролю доступу, пов'язаних із об'єктами у системі, а також списків можливостей, пов'язаних із окремими користувачами.

Контрольні запитання та завдання

1. Як можна реалізувати підтримку цілісності повідомлень і конфіденційності передачі даних у системі електронної пошти?

2. Наведено описи трьох систем керування доступом ОС:

а) кожен файл містить список коректних паролів; користувач зобов'язаний пред'явити один із цих паролів, щоб відкрити файл;

б) кожному файлу присвоюють ім'я з 256 випадкових символів, єдиний спосіб одержати доступ до файла — це задати його ім'я

в) для кожного файла адміністратор системи задає список користувачів, яким заборонено доступ до файла. Яку технологію керування доступом використовують у кожній з цих систем?

3. Поясніть, яким чином введення підтримки списків контролю доступу може розширити функціональність системи, яка базується на можливостях.

4. У чому різниця між авторизацією та аутентифікацією? Яка послідовність їхнього виконання?

5. Що таке "відмова від обслуговування"? Коли і чому вона можлива?

6. Чим відрізняються криптосистеми з секретним ключом від систем з відкритим ключом? Який з них є кращим?

7. Що таке цифровий підпис? Для чого в нбому використовують хеш-функції?

8. Словникові атаки і забоби боротьби з ними?

9. Тіньові паролі та їхня реалізація.

10. Особливості керування доступом в Windows XP.

АУДИТ ТА БЕЗПЕКА ДАНИХ

План

19.1. Аудит
19.2. Локальна безпека даних
19.3. Мережна безпека даних
19.4. Атаки і боротьба з ними

19.1. Аудит

19.1. 1. Загальні принципи організації аудиту

Підсистеми аудиту сучасних ОС містять такі основні компоненти.

♦ Засоби визначення *політики аудиту* (audit policy). Визначивши її, адміністратор системи може задати перелік подій, які його цікавлять. Для зручності визначення політики повідомлення поділяються на рівні залежно від важливості.

♦ Засоби генерації повідомлень аудиту. Такі повідомлення можуть бути згенеровані різними компонентами режиму ядра і режиму користувача. Для цього ОС має надавати відповідну функціональність і системні виклики.

♦ Засоби збереження інформації в журналі аудиту. Їх зазвичай реалізують централізовано в рамках окремого комп'ютера або мережі, для чого запускають спеціальний фоновий процес, якому передають усю інформацію про повідомлення аудиту.

Треба зазначити, що зловмисник може змінити інформацію в журналі аудиту, щоб приховати сліди свого перебування у системі. Для запобігання таким змінам у системах із високим ступенем захисту журнал аудиту можуть зберігати на віддаленому комп'ютері або на альтернативному носії (наприклад, відразу виводити на принтер або записувати на CD-ROM). Крім того, можна передбачити різні візуальні та звукові сигнали тривоги, що привертають увагу адміністратора.

19.1.2. Робота із системним журналом UNIX

Організація системного журналу

Для організації аудиту в UNIX-подібних системах використовують *системний журнал* (system log, syslog). За реєстрацію інформації від застосувань у ньому відповідає фоновий процес syslogd. Звичайно інформацію записують у файли, що перебувають у каталозі /var/log (керують процесом аудиту зміною конфігураційного файла /etc/syslog.conf). Усі системні фонові процеси і засоби забезпечення безпеки (наприклад, login) зберігають інформащю про свою роботу в цьому журналі. Адміністратори системи мають регулярно переглядати відповідні файли в пошуках можливих порушень політики безпеки.

Програмний інтерфейс системного журналу

Прикладні програми також можуть зберігати інформацію про свою діяльність у журналі за допомогою відповідного програмного інтерфейсу.

Зв'язок між процесом користувача і syslog здійснюють через спеціальний сокет. Перед початком використання журналу цей сокет можна відкрити за допомогою системного виклику openlog():

#include <syslog.h>

void openlog(char *ident. int options, int facility);

де: ident — рядок символів, що зберігатиметься в кожному записі журналу (звичайно це назва застосування);

options — маска прапорців, що визначають додаткові параметри використання журналу (LOG_PID — кожний запис журналу міститиме ідентифікатор процесу, LOG_C0NS — повідомлення виводяться на консоль у разі неможливості виведення в журнал).

19.1.3. Журнал подій Windows XP

Архітектура аудиту Windows XP

Повідомлення аудиту можуть бути згенеровані виконавчою підсистемою під час перевірки прав доступу, при цьому за передавання таких повідомлень у режим користувача для реєстрації відповідає SRM. Крім того, повідомлення можуть надходити від компонентів режиму користувача, зокрема від прикладних процесів (для цього у програмах необхідно використовувати відповідні функції Win32 API). Інформацію про такі повідомлення зберігають у *журналі подій* (event log). Для роботи із ним процес повинен мати необхідні привілеї [1].

Необхідність аудиту конкретної події визначає *локальна політика безпеки* (local security policy). Цю політику підтримує LSASS у межах загальної політики безпеки і передає SRM під час ініціалізації системи. LSASS отримує повідомлення аудиту від SRM або компонентів режиму користувача і відсилає їх окремому процесові журналу подій, що зберігає записи аудиту в журналі.

19.2. Локальна безпека даних

Розглянемо основні принципи організації локальної безпеки даних в операційних системах. Безпека даних пов'язана із забезпеченням конфіденційності інформації в рамках локального комп'ютера. Як приклад реалізації технології буде наведено автоматичне шифрування даних на файлових системах.

Зазначимо, що є й інші підходи до реалізації локальної безпеки системи. Більшість сучасних ОС реалізують бібліотеки, що дають змогу програмістові шифрувати дані та виконувати інші операції, пов'язані з їхнім криптографічним перетворенням, безпосередньо у прикладних програмах. У системах лінії Windows XP для цього призначено інтерфейси CryptoAPI і SSPI [1], в UNIX-системах можна використати системний виклик crypt ().

19.2.1. Принципи шифрування даних на файлових системах

Механізми керування доступом до файлів не можуть запобігти несанкціонованому доступу до інформації у разі фізичного викрадення

жорсткого диска. Зробивши це, зловмисник може підключити диск до комп'ютера із сумісною версією операційної системи, у якій він є привілейованим користувачем. Після реєстрації із правами такого користувача можна отримати доступ до всіх файлів на викраденому диску незалежно від того, які для них задані списки контролю доступу. Така проблема характерна для переносних комп'ютерів (ноутбуків), оскільки вони частіше потрапляють у чужі руки.

Щоб запобігти такому розвитку подій, необхідно організувати конфіденційне зберігання цінної інформації. Найчастіше це реалізують за допомогою шифрування даних на файловій системі.

Таке шифрування звичайно здійснюють на рівні драйвера файлової системи, який перехоплює спроби доступу до файла і шифрує та дешифрує його «на льоту» у разі, коли користувач надав необхідні дані (наприклад, ключ) для виконання цих операцій.

Далі в цьому розділі йтиметься про особливості підтримки таких файлових систем у Linux і Windows XP.

19.2.2.Підтримка шифрувальних файлових систем у Linux

У Linux є кілька реалізацій файлових систем із підтримкою шифрування даних. Найвідоміші з них CFS і TCFS.

Підтримка файлової системи CFS реалізована в режимі користувача і дає змогу шифрувати дані на будь-якій наявній файловій системі ціною певної втрати продуктивності. Як алгоритм шифрування використовують потрійний DES. Шифрування може бути застосоване для окремих каталогів і файлів.

Підтримка системи TCFS реалізована в режимі ядра, що дає змогу досягти вищої продуктивності і ступеня захисту. Встановлення підтримки цієї файлової системи, однак, складніше (потрібні внесення змін у код ядра Linux і його перекомпіляція). Ця файлова система реалізує концепцію динамічних модулів шифрування, надаючи користувачу можливість вибору алгоритму і режиму шифрування, які будуть використані для конкретної файлової системи, окремого каталогу або файла.

19.2.3. Шифрувальна файлова система Windows XP

Засоби підтримки шифрування файлів у ОС лінії Windows XP описані під загальною назвою *шифрувальної файлової системи* (Encrypting File System, EFS). Для цього використовують драйвер файлової системи, розташований над драйвером NTFS.

Принципи роботи EFS

Реалізація EFS — це гібридна криптосистема із кількома рівнями шифрування.

♦ Безпосередньо для шифрування даних файла використовують симетричний алгоритм (посилений аналог DES, можливе використання інших алгоритмів). Ключ для нього *(ключ шифрування файла* — File Encryption Key, FEK) генерують випадково під час кожної спроби зашифрувати файл.

♦ Для шифрування FEK використовують алгоритм із відкритим ключем (RSA). Для кожного користувача генерують пару RSA-ключів, при цьому FEK шифрують відкритим ключем цієї пари. Результат шифрування FEK зберігають у заголовку файла. Крім того, передбачена можливість дешифрування файла довіреною особою *(агентом відновлення,* recovery agent) у разі втрати ключа користувачем. Для цього результат шифрування FEK відкритими ключами довірених агентів також зберігають у заголовку файла.

♦ Відкритий ключ користувача зберігають у вигляді сертифіката у *сховищі сертифікатів* (certificate store), розташованому в домашньому каталозі користувача на локальному комп'ютері. Крім того, у цьому сховищі містяться сертифікати всіх агентів відновлення.

♦ Для шифрування закритого ключа користувача використовують симетричний алгоритм RC4 із ключем, який система генерує випадково і періодично обновлює. Цей ключ називають *майстер-ключем* (master key). Результат шифрування закритого ключа майстер-ключем зберігають на файловій системі в домашньому каталозі користувача.

4 Для шифрування майстер-ключа теж використовують симетричний алгоритм RC4, але із більшою довжиною ключа. Його генерують на основі застосування односторонньої хеш-функції SHA-1 до даних облікового запису користувача (його SID і паролю). Результат шифрування майстер-ключа цим ключем також зберігають на файловій системі.

Програмний інтерфейс EFS

Для шифрування файла або каталогу використовують функцію EncryptFile(), а для дешифрування — DecryptFile():

EncryptFile("myfile.txt");

DecryptFileC'myfile.txt", 0);

Для перевірки того, чи файл зашифрований, використовують функцію File-EncryptionStatus():

BOOL FileEncryptionStatus(LPCTSTR fname. LPDWORD Status);

де: fname — ім'я файла або каталогу;

status — покажчик на змінну, у яку заноситься інформація про підтримку шифрування для файла (FILE_ENCRYPTABLE — файл може бути зашифрований, FILE_IS_ENCRYPTED — файл зашифрований; інші значення показують, що шифрування не підтримується).

Ця функція повертає FALSE, якщо під час перевірки виникла помилка. DWORD status:

if (FileEncryptionStatusC'myfile.txt". Sstatus)) {

if (status = FILE_IS_E!CRYPTED) printf ("Файл зашифрований\n"):

}

Для деяких каталогів має сенс заборонити шифрування. Для цього необхідно помістити в такий каталог файл desktop.ini з інформацією:

[Encryption] Disable=l

19.3. Мережна безпека даних

У цьому розділі йтиметься про організацію мережної безпеки даних в операційних системах, яка пов'язана із забезпеченням конфіденційності інформації у разі її передавання мережею [10, 40]. Як приклад буде розглянуто засоби забезпечення мережної безпеки на різних рівнях мережної архітектури TCP/IP.

19.3.1. Шифрування каналів зв'язку

Захист даних у протоколах стека TCP/IP

Протоколи стека TCP/IP самі не мають достатнього рівня захисту. Фактично, основною їхньою метою була реалізація передавання даних між хостами і застосуваннями. Передбачалося, що для підтримки таких можливостей, як аутентифікація сторін або шифрування даних під час їхнього передавання через канал, ці протоколи потрібно розширити.

Розглянемо, які розширення цих протоколів можуть бути запропоновані для реалізації безпеки даних.

Види шифрування каналів зв'язку

Різні рівні шифрування каналів зв'язку в рамках стека протоколів TCP/IP наведено нижче.

1. На канальному рівні *шифрують канали зв'язку* (link-by-link encryption). Для цього пакети автоматично шифрують під час передавання через незахищений фізичний канал і дешифрують після отримання. Ці дії звичайно виконує спеціалізоване апаратне забезпечення, у сучасних умовах таке шифрування обмежене модемними або виділеними лініями.

2. На мережному рівні виконують ***наскрізне шифрування*** (end-to-end encryption), при цьому пакети мережного рівня (наприклад, IP-дейтаграми) шифрують на хості-джерелі, залишають зашифрованими впродовж усього маршруту і автоматично дешифрують хостом-одержувачем. Передавання таких пакетів мережею не змінюється, тому що їхні заголовки залишаються незашифрованими. Прикладом реалізації шифрування на мережному рівні є архітектура протоколів IPSec.

3. На межі між транспортним і прикладним рівнями теж можна шифрувати дані. Наприклад, застосування може бути модифіковане таким чином, що у разі реалізації прикладного протоколу в ньому використовуватиметься не інтерфейс сокетів, а інтерфейс альтернативної бібліотеки, яка реалізує шифрування даних. Такий підхід застосовують у протоколі SSL/TLS.

Далі в цьому розділі буде розглянуто особливості реалізації двох підходів до шифрування каналів зв'язку: наскрізного шифрування на мережному та шифрування на транспортному рівнях.

19.3.2. Захист інформації на мережному рівні

Стандартним підходом для реалізації захисту інформації на мережному рівні стека протоколів TCP/IP є архітектура IPSec. Це набір протоколів, призначених для забезпечення наскрізного захисту пакетів, які передають між двома хостами. Розглянемо мережну взаємодію у разі використання IPSec.

Хост Аліси генерує IP-дейтаграми для передавання мережею Бобові. Ці дейтаграми порівнюють із фільтрами IPSec, при цьому перевіряють, чи потрібно забезпечувати їхній захист. Фільтри IPSec задають у рамках політики безпеки IPSec, визначеної для хосту Аліси.

1. Якщо дейтаграма відповідає умові фільтра, хост Аліси починає взаємодіяти з хостом Боба із використанням протоколу ***обміну ключами Інтернету*** (Internet Key Exchange, IKE). При цьому відбувається взаємна аутентифікація сторін (на підставі сертифікатів або пароля), узгодження протоколу захисту пакетів і обмін ключами для шифрування інформації відповідно до протоколу роботи гібридної криптосистеми.

2. Програмне забезпечення хоста Аліси перетворює пакети відповідно до погодженого протоколу захисту пакетів і відсилає їх хосту Боба. У рамках IPSec є два протоколи захисту пакетів:

♦ ***заголовка аутентифікації*** (Authentication Header, AH), при використанні якого пакети супроводжують одностороннім хешем для забезпечення цілісності, але не шифрують;

♦ ***інкапсулюючого захищеного навантаження*** (Encapsulating Security Payload, ESP), який забезпечує наскрізне шифрування пакетів.

Перетворені пакети — це звичайні IP-дейтаграми, їхнє пересилання мережею відбувається стандартним способом. Додаткові дані, необхідні для протоколів AH і ESP, інкапсулюють усередині дейтаграм.

4. Програмне забезпечення комп'ютера Боба перевіряє пакети на цілісність і дешифрує їхній вміст (для протоколу ESP). Далі IP-пакети передають звичайним способом на верхні рівні реалізації стека протоколів (транспортний і прикладний).

Оскільки протоколи IPSec визначені на мережному рівні, то у разі переходу на IPSec не потрібно змінювати наявне програмне забезпечення — усі дані, передані за допомогою TCP або UDP, будуть автоматично захищені.

19.3.3. Захист інформації на транспортному рівні

Протокол SSL/TLS

Найвідомішим протоколом захисту інформації на транспортному рівні був *протокол захищених сокетів* (Secure Socket Layer, SSL). Остання його версія (SSL 3.0) із невеликими змінами була прийнята як стандартний *протокол безпеки транспортного рівня* (Transport Layer Security, TLS 1.0); цей протокол називатимемо *SSL/TLS.*

Його розташовують між протоколом транспортного рівня (TCP) і протоколами прикладного рівня (наприклад, HTTP), а реалізацію зазвичай постачають у вигляді бібліотеки користувача, інтерфейс якої можна використати у застосуваннях замість інтерфейсу сокетів. Найвідомішою реалізацією бібліотеки підтримки цього протоколу є OpenSSL.

Застосування може реалізовувати протокол прикладного рівня поверх інтерфейсу SSL/TLS. Є кілька реалізацій стандартних протоколів прикладного рівня, які базуються на SSL/TLS, серед них реалізація

протоколу HTTP (HTTPS), підтримувана більшістю сучасних веб-серверів і веб-браузерів.

Розглянемо послідовність кроків у разі використання SSL/TLS на прикладі HTTPS. Цей протокол ґрунтується на протоколі роботи гібридної криптосистеми.

1. Клієнт (веб-браузер) зв'язується із сервером із використанням протоколу HTTPS (наприклад, запросивши веб-документ, URL якого починається із https://). При цьому встановлюється TCP-з'єднання із портом 443 (стандартний порт HTTPS).
2. Клієнт і сервер погоджують алгоритми шифрування, які використовуватимуться під час обміну даними.
3. Сервер відсилає клієнтові свій відкритий ключ у складі сертифіката.
4. Клієнт верифікує сертифікат сервера за допомогою відкритого ключа довіреної сторони (центру сертифікації), що видала цей сертифікат. Сучасні веб-браузери постачають із ключами деяких центрів сертифікації, можна також встановлювати додаткові ключі. Якщо сертифікат вірний, аутентифікацію сервера вважають успішною.
5. Клієнт генерує сесійний ключ, шифрує його відкритим ключем сервера, шифрує HTTP-запит сесійним ключем і відсилає всю цю інформацію серверу.
6. Сервер розшифровує сесійний ключ своїм закритим ключем, HTTP-запит — сесійним ключем, формує HTTP-відповідь, шифрує її сесійним ключем, доповнює одностороннім хешем і відсилає клієнтові.

Клієнт дешифрує HTTP-відповідь сесійним ключем і перевіряє цілісність (для чого обчислює її односторонній хеш і порівнює із хешем, отриманим від сервера). Якщо цілісність дотримана, документ відобразиться.

Протокол SSH

Ще одним важливим протоколом безпеки транспортного рівня є протокол *безпечного командного інтерпретатора* (Secure Shell, *SSH).* Він багато в чому подібний до SSL/TLS, але використовується інакше. Найчастіше він застосовується для реалізації захищеного віддаленого доступу до системи.

Після встановлення з'єднання між SSH-клієнтом і SSH-сервером (коли завершена аутентифікація сторін і обмін ключами) SSH-сервер на віддаленому комп'ютері запускає копію командного інтерпретатора, яка використовує псевдотермінал. SSH-клієнт може взаємодіяти із цим інтерпретатором, емулюючи термінал.

Взаємодія користувача із системою фактично відбувається аналогічно до протоколу telnet, але при цьому весь канал зв'язку шифрують. Сьогодні у багатьох UNIX-системах використання SSH є обов'язковим (протокол telnet блокують із міркувань безпеки).

19.4. Атаки і боротьба з ними

У цьому розділі ознайомимося із деякими підходами, які можна використати для атаки на систему безпеки ОС. Через брак місця виклад буде обмежено атаками переповнення буфера і найпростішими прикладами відмови в обслуговуванні. Як технології запобігання атакам буде розглянуто організацію квот на ресурси і зміну кореневого каталогу застосування.

19.4.1. Переповнення буфера

Розповсюдженим типом атак на програмний код у сучасних ОС є *атаки переповнення буфера* (buffer overflow attacks). Усі вони використовують некоректний програмний код (переважно мовою С), що не перевіряє довжину буфера, у який записують зовнішні дані, отримані від користувача. Ось приклад такого коду:

```
#include <stdio.h>
void f()
{
char buf[128];
gets(buf): // небезпечне одержання рядка даних зі стандартного вводу
}
```

Функція gets(), що входить у стандартну бібліотеку мови С, вводить рядок символів довільної довжини зі стандартного вводу і розміщує їх у буфері buf. При цьому сама функція не перевіряє, скільки символів насправді було введено і чи достатньо для них місця в буфері. У ситуації, коли користувач ввів більше ніж 128 символів, програма запише дані у пам'ять, розташовану за buf.

Для того щоб зрозуміти, у чому тут полягає небезпека, розглянемо організацію пам'яті, у якій виконують застосування [1, p.18].

Адреса повернення функції і локальні змінні (зокрема і буфер buf) розміщені у програмному стеку. Коли зловмисник введе значно більше символів, ніж може бути розміщено у buf, вони переповнять буфер і будуть записані поверх адреси повернення функції. У результаті після повернення із функції відбудеться перехід за адресою, взятою із введеного зловмисником рядка. При цьому вміст рядка може бути ретельно підібраний так, щоб цей його фрагмент містив адресу коду, який бажає виконати зловмисник. Згаданий код може перебувати в іншій частині цього самого рядка і, наприклад, запускати командний інтерпретатор. Якщо застосування виконувалося із правами суперкористувача, запущений інтерпретатор дасть зловмисникові повний контроль над системою.

Для захисту від таких атак насамперед необхідно підвищувати якість розробки програмного забезпечення. Необхідно повністю відмовитися від використання функцій, які не перевіряють обсягу введених даних, замінивши їх варіантами, що роблять таку перевірку.

Захист від переповнення буфера на рівні ОС може полягати в цілковитій забороні виконання коду, що перебуває у програмному стеку. Для Linux є виправлення до коду ядра, що реалізують це обмеження.

19.4.2. Відмова від обслуговування

Найпростіший приклад атаки відмови в обслуговуванні для UNIX-систем — так звана «fork-бомба»:

```
void main()
 {
for (; ;) fork();
}
```

Очевидно, що процес, завантажений у пам'ять внаслідок запуску такої «бомби», почне негайно створювати свої власні копії, те саме продовжать робити ці копії і т. д. У старих версіях UNIX це могло швидко привести систему до нероботоздатного стану, і навіть сьогодні запуск такого застосування у системі із малим обсягом ресурсів здатний значно понизити її продуктивність.

Для боротьби із такими атаками необхідно встановлювати ліміти на ресурси. Зокрема, потрібно, щоб у системі було встановлено ліміт на максимальну кількість процесів, які можуть бути запущені під обліковим записом користувача. За замовчуванням ця кількість дорівнює 256, змінити її можна за допомогою системного виклику setrlimit():

#include <sys/resource.h> #include <unistd.h> struct rlimit plimit; plimit.rlimjnax = 100: setr1imit(RLIMITJPR0C. Splimit);

Для ліквідації наслідків такої атаки не можна намагатися негайно завершити всі процеси-бомби, виконавши, наприклад, команду вилучення всіх процесів із заданим ім'ям:

```
#       killall -KILL bomb
```

Річ у тому, що після запуску «бомби» у системі швидко створюється стільки процесів, що ліміт на їхню кількість виявиться вичерпаним. Після цього всі процеси-бомби перестають створювати нащадків і залишаються у нескінченному циклі. Як тільки після виконання killall завершиться один із таких процесів (а вони всі не можуть завершитись одночасно), загальна кількість процесів стане менша за ліміт. У результаті якийсь інший процес набору відразу отримує можливість виконати fork() і встигає це зробити до свого завершення. Фактично після знищення поточного набору процесів його місце негайно займає новий.

Правильним підходом буде спочатку призупинити всі процеси-бомби, а потім послати їм сигнал завершення:

```
#killall -STOP bomb
#killall -KILL bomb
```

Системний виклик setrlimit() може також бути використаний для встановлення інших квот на ресурси, зокрема обмежень на кількість відкритих файлів (першим параметром потрібно задати RLIMIT_NOFILE), на частку процесорного часу, яку використовує процес (RLIMITCPU), на розмір процесу у пам'яті (RLIMIT_AS).

Другим важливим засобом обмеження споживання ресурсів є квоти дискового простору.

19.4.3. Квоти дискового простору

Дискову квоту — встановлюване адміністратором обмеження на використання різних ресурсів файлової системи — задають для кожного користувача. Зазвичай до таких ресурсів належать кількість виділених кластерів або кількість створюваних файлів. Операційна система гарантує, що користувачі не зможуть перевищити виділені їм квоти.

Квоти звичайно реалізовують так. На диску організовують спеціальний файл (файл квот), що містить таблицю квот, куди заносять інформацію про квоти різних користувачів і те, які поточні обсяги ресурсів ними витрачені. Якщо користувач відкрив хоча б один файл, відповідний цьому користувачу елемент таблиці квот зчитують у пам'ять.

Файловий дескриптор, відкритий процесом, містить покажчик на елемент таблиці квот, що відповідає користувачу, який створив процес. У разі зміни розміру якогось із відкритих файлів інформацію про нову сумарну витрату ресурсу зберігають в елементі таблиці у пам'яті. Після того як користувач закриє всі свої файли, елемент таблиці квот зберігають на диску у файлі квот.

В елементі таблиці квот міститься така інформація: гнучкий і жорсткий ліміт на ресурс; поточний стан витрати ресурсу; максимально допустима і поточна кількість попереджень про перевищення гнучкого ліміту.

Спроба перевищити жорсткий ліміт завжди спричиняє помилку. Гнучкий ліміт може бути перевищений користувачем, але він при цьому отримує попередження (для кожного користувача визначено скінчений запас таких попереджень). Якщо користувач витратить усі попередження про перевищення гнучкого ліміту, його більше не допускають у систему без дозволу адміністратора.

Дискові квоти підтримують у Linux і в лінії Windows XP (для файлової системи NTFS).

19.4.4. Зміна кореневого каталогу застосування

Одним із ефективних способів запобігання або ослаблення дії різних атак в UNIX-системах є обмеження видимості файлової системи для застосувань. У такому разі кореневим каталогом ("/") для процесу стає не справжній кореневий каталог файлової системи, а один із його підкаталогів. Поза даним каталогом файлова система для цього процесу буде недоступною. Таку зміну відображення файлової системи здійснюють за допомогою системного виклику chroot(), у даному випадку кажуть про застосування зі зміненим кореневим каталогом або chroot-застосування.

Застосування зі зміненим кореневим каталогом не має доступу до більшості каталогів системи, тому заподіяна шкода від зловмисника, який отримав контроль над системою за допомогою такого застосування

(наприклад, за допомогою атаки переповненням буфера), буде значно обмежена.

Виклик chroot() може бути виконаний тільки процесом із правами суперкористувача. Наведемо приклад його використання для того щоб зробити кореневим каталогом поточного процесу /home/user/newroot:

#inc1ude <unistd.h>

chroot("/home/user/newroot");

Зміну кореневого каталогу часто використовують у різних серверах. Так, деякі сучасні версії стандартних мережних серверів (сервер імен bind, поштовий сервер sendmail) як один із режимів підтримують роботу за умов зміненого кореневого каталогу Розглянемо послідовність завантаження сервера у разі використання chroot().

1.Здійснюють підготовку до обслуговування клієнтів (завантажують необхідні динамічні бібліотеки і конфігураційні файли, відкривають файли журналу).

2.Виконують виклик chroot ().

3.Сервер починає очікування з'єднань від клієнтів та їхнє обслуговування. При цьому рекомендовано перейти в режим виконання із правами звичайного користувача за допомогою системного виклику setuid().

Висновки

Основними завданнями забезпечення безпеки комп'ютерних систем є аутентифікація, керування доступом, аудит, забезпечення конфіденційності, доступності та цілісності даних.

Організація аудиту дає змогу зберігати інформацію про різні події у системі для подальшого аналізу. Інформацію зберігають у спеціальному журналі аудиту (системному журналі, журналі подій), вибір подій для реєстрації в цьому журналі визначає політика аудиту.

Для забезпечення конфіденційності даних у рамках локальної системи використовують шифрувальні файлові системи або криптографічні API. Мережна безпека даних забезпечена їхнім шифруванням на різних рівнях мережної архітектури. Найчастіше таке шифрування виконують на мережному рівні або між транспортним і прикладним рівнями.

Контрольні запитання та завдання

1. Як виконати атаку відмови в обслуговуванні з використанням підсистеми керування віртуальною пам'яттю сучасних ОС?
2. Мережні хробаки — це застосування, що автоматично поширюють свої копії доступними мережними з'єднаннями. Яким чином поява в системі такого хробака може призвести до атаки відмови в обслуговуванні, навіть якщо він не містить коду, що зумисно здійснює таку атаку?
3. Яким чином запобігають модифікації журналу аудиту з боку зловмисника?

4. Яке призначення системного журналу, в якому файлі він зберігається?

5. Який процес ОС відповідає за реєстрацію інформації в системному журналі?

6. Чим відрізняються принципи організації локальної та мережевої безпеки даних?

7. Принципи роботи шифрувальної файлової системи Windows XP. Які шифрувальні алгоритими використовуються в ній?

8. Принципи організації мережної безпеки. Рівні захисту?

9. Особливості мережної безпеки на різних рівнях захисту.

10. Основні типи атак. Переповнення буфера. Як захистити ПК від атаки переповнення буфера?