МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРИЗОВАНИХ СИСТЕМ ЗАХИСТУ ІНФОРМАЦІЇ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____ С.В. Казмірчук
«_____»_____ 20__ р.

На правах рукопису
УДК 004.056.5:510.22(043.3)

КВАЛІФІКАЦІЙНА РОБОТА

ЗДОБУВАЧА ВИЩОЇ ОСВІТИ ОСВІТНЬОГО СТУПЕНЯ «МАГІСТР»

Тема: Удосконалений алгоритм автентифікації та авторризації протоколу RADIUS

Виконавець: А.О. Мусель
Науковий керівник: к.т.н., доцент А.Б. Петренко
Нормоконтролер: к.т.н., доцент А.Б. Петренко

Київ 2020

ВСТУП

В даний час кількість інтернет сервісів швидко зростає, що призводить до необхідності постійного вдосконалення маршрутизаторів та серверів доступу до мережі з метою задоволення нових вимог. Постійне зростання кількості користувачів також вимагає великої кількості NAS із складними конфігураціями. Але постачальники послуг розмежування доступу до ресурсів повинні не тільки пропонувати вільний порт для кожного з'єднання. Вони також повинні забезпечити захист від викрадення даних користувачів, перевірку рівня доступу користувача до ресурсів. Також можуть знадобитися дані про час підключення користувача до мережі для виставлення рахунків та подальшого планування ресурсів. Усі ці послуги вимагають координації між різними системами.

Саме вирішення цих проблем лежить в основі використання протоколу автентифікації, авторизації та аудиту (AAA), який є по суті фреймоврком, що вирішує разом всі ці окремі задачі завдяки різним мережевим технологіям та платформам, зокрема протоколу RADIUS.

Рішення цієї проблеми базується на використанні протоколу автентифікації, авторизації та аудиту (AAA), який, по суті, є фреймоврком, який вирішує всі ці окремі завдання разом за допомогою різних мережевих технологій та платформ, включаючи протокол RADIUS.

Оскільки протокол RADIUS не надто захищений, але все ще залишається одним із найбільш широко використовуваних протоколів AAA, метою цього дослідження є розробка методів вдосконалення механізмів протоколу RADIUS.

Об'єктом дослідження є механізми забезпечення процесів автентифікації та авторизації протоколу RADIUS.

Предметом дослідження є безпека та її забезпечення у протоколі RADIUS.

Досягнення мети дослідження реалізується за допомогою детального вивчення існуючих механізмів захисту протоколу, огляду існуючих рішень

захисту в криптографічних протоколах та можливості використання цих механізмів по відношенню до протоколу, який досліджується, з урахуванням специфіки автентифікації, авторизації та аудиту.

Таким чином, завдяки розробленим в ході виконання дослідження покращеним методам автентифікації та авторизації можуть бути вирішена велика кількість проблем безпеки сучасних реалізацій протоколу RADIUS, а отже підвищення рівня безпеки як користувацьких даних, так і обмежених ресурсів систем.

Отримані результати повинні бути перевірені за допомогою існуючих інструментів перевірки безпеки криптографічних протоколів, а також на відповідність міжнародним стандартам у сфері AAA.

Основні положення роботи доповідалися та обговорювалися на таких конференціях:

- Міжнародна наукова конференція «Aplikovane Vedecke Novinky». Прага, 22 липня – 30 липня.
- Міжнародна наукова конференція «Scientific Horizonts». Шеффілд, 30 вересня – 07 жовтня.

РОЗДІЛ 1. ОГЛЯД ПРОТОКОЛУ RADIUS

У першому розділі даної роботи буде розглянуто призначення протоколу автентифікації, авторизації та аудиту та механізми забезпечення цих операцій за допомогою протоколу RADIUS, а також проаналізовані основні вразливості та можливі атаки на зазначений протокол.

1.1 Огляд механізмів аутентифікації, авторизації та аудиту на основі протоколу RADIUS

Для початку розглянемо кожен елемент протоколу аутентифікації, авторизації та аудиту окремо.

Аутентифікація - це процес ідентифікації користувача, який намагається отримати доступ до ресурсу чи об’єкта. Цей процес визначає, чи має кінцевий користувач унікальну інформацію, тобто комбінацію «ім’я користувача та пароль», секретний ключ або біометричні дані (наприклад, відбитки пальців), які служать унікальними даними. Сервер AAA порівнює надані користувачем облікові дані з даними користувача, що зберігаються в базі даних сервера. Якщо облікові дані збігаються, користувачеві надається доступ до мережі. В іншому випадку сталася помилка автентифікації, і користувачеві було відмовлено у доступі до мережі. Відповідно до RFC2989 авторизація – це визначення, чи може певне право чи послуга бути надана пред’явнику ідентифікаційних даних. Таким правом/послугою можуть бути доступ до мережі, надання IP-адреси, виклик фільтра для визначення, які програми чи протоколи підтримуються. У деяких випадках процеси авторизації та автентифікації поєднані.

Авторизація - надання певній особі або групі осіб прав на виконання певних дій, а також процес перевірки (підтвердження) даних прав при спробі виконання цих дій. Часто можна почути вираз, що якийсь чоловік «авторизований» для виконання даної операції - це значить, що він має на неї прав. Авторизація виконує контроль доступу легальних користувачів до ресурсів системи після успішного проходження ними аутентифікації. Найчастіше процедури аутентифікації і авторизації поєднуються. В інформаційних технологіях за допомогою авторизації встановлюються права доступу до інформаційних ресурсів і систем обробки даних.

Аудит - паралельний з аутентифікацією і авторизацією етап, який записує в журнал успіх чи невдачу даних процесів, змогла людина проникнути в приміщення чи ні, чи отримав користувач доступ до мережних пристроїв і, якщо так, то які дії на ньому здійснював. Цей процес важливий з точки зору безпеки та контролю доступу, так як дозволяє визначати потенційні загрози і шукати «діри» в системі.

Сучасні системи використовують такі протоколи для автентифікації, авторизації та аудиту: Служба віддаленої автентифікації (Dial-In User Service) (RADIUS), Cisco Terminal Access Controller Access-Control System Plus (TACACS +) і DIAMETER. У цій роботі буде розглядатися лише протокол RADIUS.

RADIUS (remote authentication dial-in user service) - це мережевий протокол, який забезпечує клієнт-серверну аутентифікацію, авторизацію і аудит віддалених користувачів. Для підключення віддалених користувачів, в мережі повинні бути встановлені сервери доступу та засоби для віддаленого підключення. Сервер доступу запитує у користувача облікові дані для входу і передає їх на сервер RADIUS, на якому зберігаються імена користувачів і паролі. При цьому віддалений користувач є клієнтом сервера доступу, а сервер доступу є клієнтом RADIUS-сервера.

Незважаючи на те, що цей протокол був розроблений у 1991 році, він все ще широко використовується, як правило, у білінгових системах (автоматична

платіжна система "Hydra", автоматична платіжна система LANBilling) (рис. 1, додаток 1.1), для доступу адміністратора до локальної мережі підприємства з непередбачуваних місць.

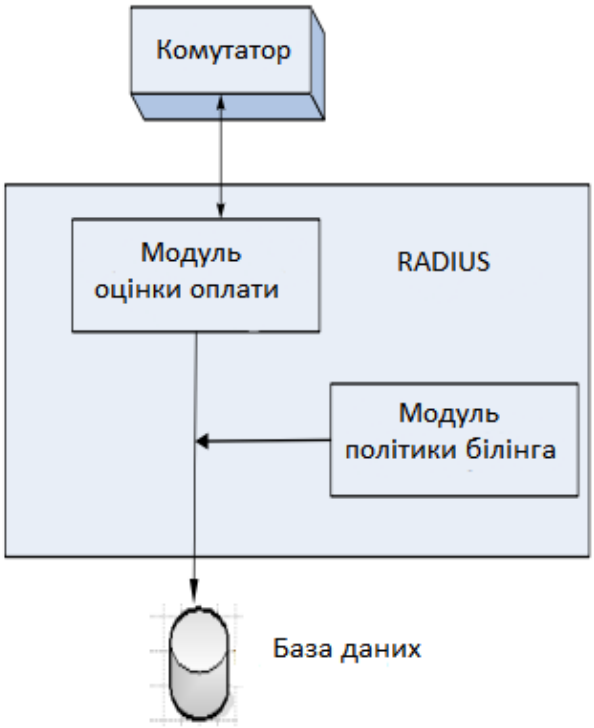

Рисунок 1.1 – Приклад архітектури білінгової системи на базі протоколу RADIUS

RADIUS також використовується в корпоративних середовищах, щоб забезпечити доступ до корпоративної мережі співробітникам під час відряджень або з дому. RADIUS дозволяє компаніям зберігати профілі користувачів в централізованій базі даних. Після успішної аутентифікації, зовнішньому користувачу присвоюється попередньо налаштований профіль, який визначає до яких ресурсів він може отримати доступ, а до яких - ні. Ця технологія дозволяє компанії мати єдину керовану точку входу, що забезпечує стандартизацію з точки зору безпеки та надає простий спосіб контролю використання віддаленого доступу і збору мережевий статистики.

RADIUS був розроблений компанією Livingston Enterprises для своєї серії серверів доступу, але потім був опублікований у вигляді стандартів RFC 2865 "RADIUS Customer Assistance Service" і RFC 2866 "RADIUS Audit". Це

відкритий протокол, який може використовувати будь-який виробник в своїх продуктах. Конфігурації і облікові дані користувачів можуть зберігатися на серверах LDAP, в різних базах даних або текстових файлах.

Повідомлення у цьому протоколі передаються у формі пакетів протоколу передачі даних без встановлення з'єднання UDP. Крім того, усі проблеми пов'язані з доступністю сервера, повторною передачей даних та часу очікування обробляються пристроями з підтримкою RADIUS, а не самим протоколом передачі. У пакеті UDP завжди є лише одне повідомлення RADIUS. Поточна реалізація протоколу використовує порт 1812 для автентифікації та порт 1813 для аудиту. Деякі сервери можуть використовувати порт 1645 для автентифікації та порт 1646 для аудиту. Протокол реалізує архітектуру клієнт-сервер, де клієнтом є сервером мережевого доступу (NAS), а сервер - це демон-процес, що працює на машині UNIX або Windows NT. NAS діє як шлюз між користувачем і сервером, відповідає за передачу інформації про користувача на сервери RADIUS, а потім діє залежно від відповіді, отриманої від сервера. RADIUS сервера отримують запроси на надання з'єднання користувачеві, автентифікують користувача, а потім повертають інформацію про конфігурацію, необхідну клієнту для надання послуг, які запросив користувач. На рис. 1.2 показано загальну схему взаємодії користувача, клієнта та сервера в системі RADIUS.

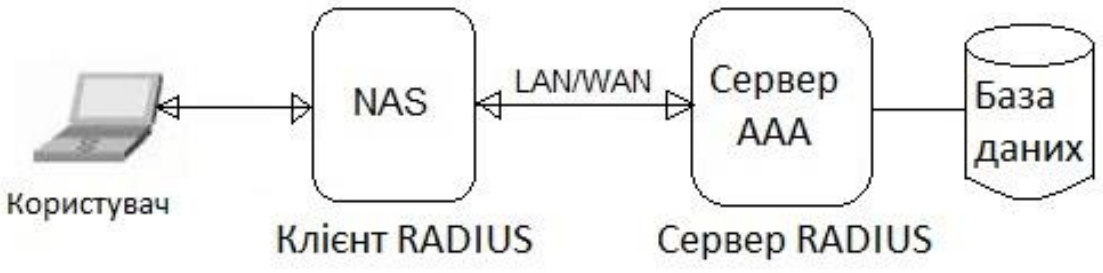

Рисунок 1.2 – Загальна схема взаємодії протоколу RADIUS

Ви можете делегувати будь-яку функціональність сервера RADIUS іншому серверу RADIUS. Потім сервер RADIUS, який отримує повідомлення від клієнта, стає проксі-сервером. Наприклад, Cisco часто використовує таку

модель взаємодії RADIUS. Реєстратор доступу Cisco (Cisco AR) виглядає як проксі-сервер, який приймає запити від клієнтів; як замовник - США. Cisco AR передає запит на отримання інформації про користувача до серверу полегшеного протоколу доступу до каталогів LDAP (Lightweight Directory Access Protocol). І на основі отриманих даних приймає рішення про автентифікацію та авторизацію користувачів. Ця схема протоколу показана на рис. 1.3.

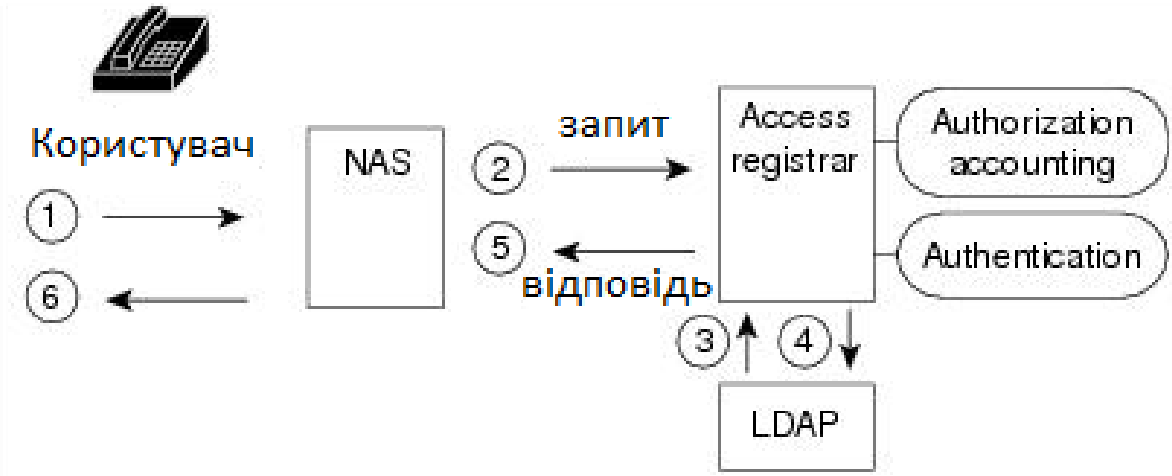

Рисунок 1.3 – Схема проксі Cisco AR з використанням LDAP серверу

Розглянемо процеси аутентифікації та авторизації RADIUS більш детально.

Аутентифікація та авторизація RADIUS описані в RFC 2865. Як правило, вхід користувача складається із запиту від NAS до сервера RADIUS (запит на доступ) та відповіді сервера (доступ приймається або забороняється). RFC 2865 та 2866 підтримують такі типи повідомлень:

1. Access-Request. Відправляється клієнтом RADIUS для запиту автентифікації та спроб авторизації при підключенні до мережі.

2. Access-Accept. Надіслане сервером у відповідь на повідомлення із запитом на доступ. Це повідомлення інформує клієнта про прийняття спроби автентифікації та авторизації.

3. Access-Reject. Надіслане сервером у відповідь на повідомлення із запитом на доступ. Це повідомлення інформує клієнта про те, що його спроба

автентифікації та авторизації була відхилена. Сервер надсилає це повідомлення, якщо інформація про користувача неправильна або спроба підключення заборонена.

4. Access-Challenge. Він містить додаткову інформацію від користувача, таку як другий пароль, PIN-код або маркер. Він також використовується в більш складних випадках автентифікації, коли між хостом і сервером користувача встановлюється захищений тунель, щоб клієнт не міг отримати доступ до даних користувача.

5. Accounting-Request. Надіслане клієнтом для ініціювання процесу аудиту прийнятого з'єднання.

6. Accounting-Response. Надіслане сервером у відповідь на Accounting-Request. Підтверджує успішне отримання запиту на аудит.

Загальна схема пакета RADIUS наведена на рис. 1.4.

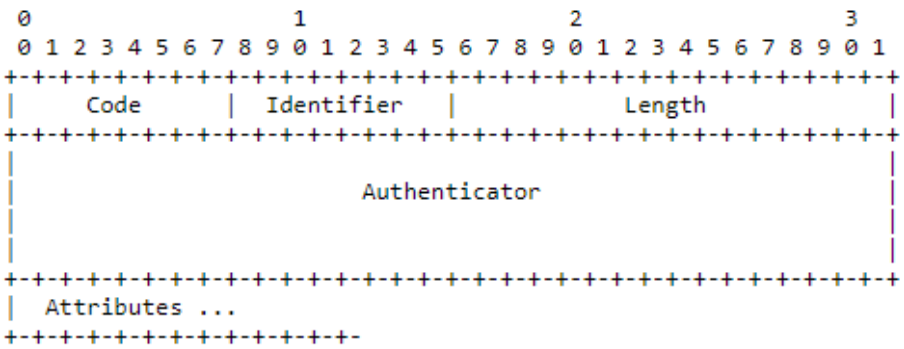

Рисунок 1.4 – Загальна схема пакету RADIUS

Кожен пакет складається з наступних полів:

1. Код. Визначає тип пакета: Access-Request, Access-Accept, Access-Reject, Access-Challenge, Accounting-Request або Accounting-Response.

2. Ідентифікатор. Містить значення, яке копіюється у відповідь сервера, щоб клієнт міг узгодити свої запити з відповіддю сервера. Він також використовується сервером для розпізнавання клієнтів при аутентифікації кількох користувачів.

3. Довжина. Служить простим механізмом перевірки помилок. Сервер відкидає пакет, якщо його довжина менше значення, вказаного в цьому полі.

Якщо довжина пакета перевищує це поле, додаткові октети ігноруються сервером.

4. Автентифікатор. Містить значення або запиту, або автентифікатора відповіді. Аутентифікатор запиту включений до пакету Access-Request. Це значення непередбачуване і унікальне. Він використовується для шифрування пароля користувача при його передачі від клієнта на сервер.

5. Атрибути. Залежить від типу пакету, до складу якого він входить. Кількість пар атрибут/значення, що містяться в цьому полі, є змінною, включаючи обов'язкову або необов'язкову для типу послуги, що вимагається. Атрибути RADIUS описані в RFC 2865, 2866, 2867, 2868, 2869 та 3162.

Кінцевий користувач знаходиться у довірчих відносинах із сервером RADIUS через логін та пароль. NAS і сервер мають довірчі відносини на основі спільного пароля. Коли користувач намагається отримати доступ до мережі, NAS передає облікові дані між користувачем та сервером RADIUS. Цей процес називається «RADIUS conversation». Усі RADIUS conversations ініціюються NAS від імені користувача.

1.2 Забезпечення процесів автентифікації та авторизації в протоколі RADIUS

Транзакції між клієнтом та сервером RADIUS здійснюються із використанням спільного пароля, який відомий і клієнту, і серверу, не надсилається по мережі та вказується системним адміністратором як текстовий рядок. Єдиним технічним обмеженням спільного секрету є те, що його довжина повинна бути більше 0, але RFC рекомендує, щоб довжина секрету становила щонайменше 16 байт. Ця вимога пов'язана з тим, що таємницю такої довжини практично неможливо зламати атакою грубої сили.

Клієнт створює пакет Access-Request, що містить принаймні ім'я користувача та пароль. Ідентифікатор пакету Access-Request генерується клієнтом. Зазвичай це звичайний лічильник, який виставляється за рахунок кожного нового запиту. Пакет Access-Request містить аутентифікатор запиту, який є випадково вибраним 16-байтовим рядком. Запит на доступ повністю захищений, за винятком атрибута User-Password, який захищений наступним механізмом. Клієнт і сервер діляться секретом. Результат конкатенації спільного пароля та автентифікатора запиту хешується за допомогою MD5 для отримання значення 16 байт, до якого додається за модулем пароль користувача. Якщо пароль користувача перевищує 16 байт, додаткові обчислення MD5 виконуються за допомогою зашифрованого блоку пароля користувача замість автентифікатора запиту.

Сервер отримує пакет запитів на доступ і перевіряє, чи має сервер спільний пароль для вказаного клієнта. Це робиться шляхом доступу сервера до бази даних (текстові файли, сервери LDAP тощо. Вони можуть служити альтернативою базі даних). Якщо відповідний клієнт не отримує таку розподілену таємницю, запит просто відхиляється. Якщо поле Request-Authenticator відсутнє або його неможливо перевірити в повідомленні клієнта, сервер також відхиляє запит.

На наступному кроці сервер використовує свою базу даних автентифікації для перевірки імені користувача та пароля. База даних RADIUS зберігає інформацію про користувача (ім'я, пароль) та дані сеансу (загальний час сеансу та статистику трафіку, що входять до / з користувача). Якщо пароль правильний або ім’я користувача відсутнє в базі даних сервера, сервер створює пакет Access-Accept, який слід відправити назад клієнту. Якщо пароль неправильний, сервер створює пакет Access-Reject для надсилання назад клієнту. Це повідомлення може супроводжуватися текстовим повідомленням, що обґрунтовує відмову. Пакети Access-Accept та Access-Reject використовують однаковий ідентифікатор пакета Access-Request, надісланого клієнтом. Сервер також розміщує Response Authenticator у полі Authenticator.

Аутентифікатор відповіді - це результат хеш-функції MD5, вхідним сигналом якої є пакет відповідей сервера із відповідним аутентифікатором запиту та загальним секретом:

$$Response\ Authenticator = MD5(Code + Id + Length + Request\ Authenticator + Attributes + ShrdSecret), \quad (1.1)$$

де *RequestAuthenticator* – автентифікатор запросу;

ShrdSecret – спільний секрет клієнта та сервера.

Коли клієнт отримує пакет відповідей від сервера, він намагається зіставити його із своїм запитом за допомогою поля ID. Якщо клієнт не надіслав запит, але отримав відповідь, така відповідь відхиляється. В іншому випадку клієнт перевіряє автентифікатор відповіді, використовуючи ті самі розрахунки, що і у формулі (1.1), а потім порівнює результат із полем Authenticator із відповіді сервера. Якщо вони не збігаються, пакет відповідей із сервера відкидається. Якщо клієнт отримує пакет Access-Accept, ім'я користувача та пароль є дійсними, а користувача аутентифіковано. Якщо клієнт отримує підтверджене повідомлення про відмову в доступі, ім’я користувача та пароль вважаються недійсними, а користувач не аутентифікований.

У RADIUS процеси авторизації та аутентифікації пов’язані, тому, як тільки аутентифікація буде виконана, користувач автоматично отримує авторизацію в системі.

1.3 Механізм аудиту протоколу RADIUS

Процес аудиту в протоколі RADIUS виконується незалежно від автентифікації та авторизації. Аудит протоколу описаний у RFC2866. Аудит дозволяє передавати дані на початку та в кінці сесії із зазначенням кількості

ресурсів, використаних під час сесії. Такими ресурсами можуть бути час, пакети, байти тощо.

Основною метою процесу аудиту є облік користувача щодо використовуваного доступу до мережі для цілей статичного та загального моніторингу мережі.

Коли NAS надає користувачеві доступ до мережі, він надсилає запис Accounting Start на сервер, щоб повідомити серверу про те, що користувач почав використовувати мережеві ресурси. Accounting Start - це пакет запитів на проводку, який має значення атрибуту Acct-Status-Type у значенні "start". Цей пакет зазвичай містить інформацію про ідентифікацію користувача, його мережеву адресу, точку доступу та унікальний ідентифікатор сеансу.

Періодично під час сеансу NAS може надсилати проміжні записи оновлення на сервер для оновлення стану поточного сеансу. В тілі таких пакунків атрибуту Acct-Status-Type встановлено значення "проміжне оновлення". Проміжні пакети зазвичай надсилають інформацію про тривалість поточного сеансу та використання даних.

Нарешті, коли сеанс припиняється і доступ користувача до мережі припиняється, NAS надсилає на сервер запис зупинки обліку, що містить інформацію про час сеансу, переданий пакет, передані дані, причину відключення та іншу інформацію, пов’язану з доступом користувача до мережі. Атрибут Acct-Status-Type у цьому записі встановлено на "зупинка". Зазвичай клієнт продовжує надсилати пакети Accounting-Request, поки не отримає пакет Accounting-Response від сервера.

Обчислення вмісту пакету Access-Request відрізняється від обчислення пакета Accounting-Request. На відміну від Request Authenticator у пакеті Access-Request, значення Request Authenticator у пакеті Authenticator-Request не є випадковим і обчислюється згідно з RFC 2866 згідно з формулою (1.2).

Response Authenticator = MD5(Code + Id + 16 нульових байтів + Attributes + Secret), (1.2)

де *Secret* – спільний секрет сервера та клієнта.

Таким чином, сервер може перевірити запит на розрахунки та відкинути пакет, якщо при перетворенні того самого значення на стороні сервера результат не збігається з отриманим від клієнта. У більшості випадків це пов’язано з неправильним спільним паролем.

1.4 Огляд протоколу аутентифікації EAP

Протокол EAP відноситься до протоколів аутентифікації, що підтримує розширення своєї функціональності за рахунок додавання нових методів. EAP зазвичай працює безпосередньо на базі протоколів канального рівня типу PPP або IEEE 802, не вимагаючи використання протоколу IP. EAP може застосовуватися на виділених і комутованих каналах, як в дротяних, так і в бездротових мережах. Даний протокол використовує покрокову схему обробки повідомлень: новий запит відправляється тільки після отримання відповіді на попередній. Тому даний протокол не призначений для передачі великих обсягів даних. EAP забезпечує власну підтримку повторної передачі повідомлень і позбавлення від дублікатів, але вона заснована на гарантованому порядку доставки повідомлень протоколом нижчого рівня. Відповідно, обробка пакетів, доставлених з порушенням порядку, не підтримується. Архітектура EAP використовує такі ролі для мережевих вузлів:

- аутентифікатор (authenticator) -мережний вузол, який починає процес аутентифікації

- партнер (peer) - мережевий вузол, який відповідає на запити аутентифікатора;
- внутрішній сервер аутентифікації (backend authentication server) - виділений сервер, що надає послуги аутентифікації (виконує методи EAP) для аутентифікатора;
- сервер EAP - об'єкт, який реалізує методи EAP; він або розміщується на сервері аутентифікації, або є частиною аутентифікатора.

Однією з переваг архітектури EAP є її гнучкість. Вибір конкретного механізму аутентифікації відбувається після того, як аутентифицирующей сторона (аутентифікатор) отримає додаткову інформацію від партнерського вузла. Замість необхідності кожен раз оновлювати аутентифікатор для підтримки нового методу аутентифікації архітектура EAP дозволяє використовувати виділений сервер, який підтримує різні методи аутентифікації, а сам аутентифікатор просто пересилає повідомлення від інших вузлів цьому серверу.

Загальна схема роботи протоколу зображено на рис. 1.5. Клієнт (партнер) запитує доступ до деякого ресурсу, звертаючись до системи доступу (аутентифікатору). Аутентифікатор передає запит з даними клієнта серверу EAP. Сервер EAP запитує додаткові дані у клієнта. Обмін повідомленнями між клієнтом і сервером EAP триває до тих пір, поки обраний метод аутентифікація не завершиться успішно або з помилкою. Аутентифікатор на підставі результату аутентифікації, отриманого від сервера EAP, приймає рішення про надання клієнту доступу до запитаного ресурсу. Таким чином, аутентифікатор виконує роль посередника між клієнтом і сервером EAP.

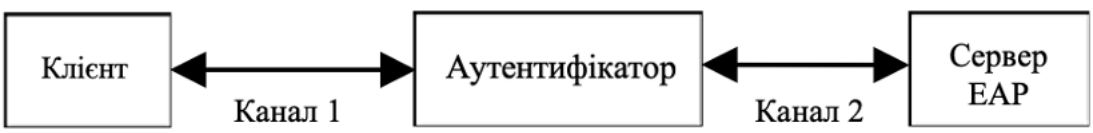

Рисунок 1.5 – Загальна схема роботи EAP

Перший комунікаційний канал між клієнтом і аутентифікатором може бути як дротовим, так і бездротовим. Досить часто в ролі клієнта може виступат мобільний пристрій, а в ролі аутентифікатора - точка доступу. Пересилання пакетів EAP між клієнтом і аутентифікатором здійснюється за допомогою інкапсуляції пакетів EAP в протокол нижчого рівня, наприклад, в протокол PPP при використанні каналів точка-точка, або в протокол EAPOL (EAP over LAN) в мережах IEEE 802. Другий канал між аутентифікатором і сервером EAP, як правило, є провідним, однак в деяких випадках (наприклад, в сценаріях роумінгу), між ними можуть розміщуватися додаткові об'єкти пересилання повідомлень.

Багато мережеві пристрої зазвичай мають досить обмежені апаратні ресурси і, наприклад, не можуть зберігати в пам'яті інформацію про велику кількість користувачів. Крім того, число користувачів може регулярно змінюватися, і виникає необхідність в єдиній базі даних. У такій системі від аутентифікатора конкретного ресурсу потрібно тільки підтримка базової функціональності протоколу EAP. А реалізація всіх методів аутентифікації і база даних з інформацією про всіх користувачів і їх права доступу знаходяться в єдиній підсистемі - сервері аутентифікації.

Пересилання пакетів EAP між внутрішнім сервером аутентифікації і клієнтом здійснюється за допомогою інкапсуляції пакетів EAP в протокол аутентифікації, авторизації та обліку, який виконується між аутентифікатором і внутрішнім сервером аутентифікації. Як протокол AAA зазвичай застосовуються саме протокол RADIUS.

1.5 Аналіз вразливостей протоколу RADIUS

Сервер RADIUS підтримує кілька протоколів автентифікації. Найчастіше використовуються протокол автентифікації пароля (PAP), протокол

автентифікації Challenge Handshake (CHAP) та перша або друга версія протоколу MS-CHAP (MS-CHAPv2), доступ до сервера клієнт-сервер 802.1X (EAP, EAP-TLS, PEAP, EAP-TTLS, EAP-MD5, EAP-GTC, LEAP), аутентифікація HTTP. Крім того, можна використовувати RADIUS з протоколом Point-to-Point. Коли сервер RADIUS надається спільно з усіма вищезазначеними протоколами, результати автентифікації користувача та сервера доступу надсилаються на сервер RADIUS, який приймає рішення про автентифікацію користувача.

Безпека цього алгоритму базується на хеші MD5. MD5 - це 128-розрядний алгоритм хешування, розроблений у 1991 році. Призначений для створення офортів будь-якої довжини. Він замінив MD4, який був недосконалим. На момент розробки протоколу хеш MD5 вважався безпечним, але станом на 2011 рік, згідно з RFC 6151, алгоритм вважався ненадійним. В даний час існує кілька типів злому хеш-алгоритму MD5:

1. Метод грубої сили. Це підхід, який передбачає перегляд усіх можливих вхідних значень, поки не отримається потрібний хеш.

2. Використання райдужних таблиць. Райдужна таблиця – це попередньо обчислена таблиця для інвертування хуш-функцій, як правило, для пошуку хешей паролей.

3. Колізія хеш-функції. Під колізією розуміють отримання одного і того ж результату обчислення хеш-функції з різними вхідними даними.

Крім того, RADIUS має вразливі місця безпеки, що виникають як через сам протокол, так і через погану реалізацію клієнта.

Однією з основних проблем протоколу, що призводить до можливості багаторазових атак на конкретний протокол, є відсутність перевірки на стороні сервера джерела вхідних повідомлень про доступ-запит. Сервер перевіряє запит на доступ з IP-адреси для налаштованого клієнта RADIUS, але IP-адреси можна легко підробити.

Рішенням цієї проблеми є використання атрибута Message-Authenticator у пакетах Access-Request. Цей атрибут визначений у RFC 3579. Він

використовується для перевірки пакета Access-Request на стороні сервера та містить код автентифікації повідомлень на основі хешу (HMAC), який служить механізмом перевірки цілісності інформації для забезпечення того, що дані, надіслані від клієнта на сервер, не були змінені. третіми сторонами, тобто запобігання нападу, який опосередковується людиною.

Перед відправкою пакету Access-Request клієнт обчислює MD5 HMAC з даних пакета, який він має намір надіслати на сервер, і додає отримане хеш-значення як атрибут Message-Authenticator. Зі свого боку, сервер перевіряє поле Message-Authenticator, виконуючи ті самі дії. Якщо пакет не був протестований на стороні сервера, сервер повинен скинути пакет. Формула для обчислення атрибута Message-Authenticator така:

Message-Authenticator = HMAC-MD5 (Type, Identifier, Length, Request Authenticator, Attributes) (1.3)

Цей атрибут повинен використовуватися для будь-якого пакету, що містить повідомлення розширеного протоколу автентифікації. Атрибут необов’язковий для інших протоколів.

Зверніть увагу, що перший пакет у сеансі 802.1x RADIUS ніколи не містить повідомлення EAP. У цьому випадку поле Message-Authenticator не генерується, і неможливо перевірити клієнтський пакет на стороні сервера. Але клієнт все ще може перевірити відповідь сервера за допомогою поля Request Authenticator.

Відомі наступні атаки на конкретний протокол:

1. Атака, отримана розподіленим секретом за допомогою атрибута логін-пароль.

Зловмисник може отримати спільну секретну інформацію про пароль, якщо зможе контролювати мережевий трафік. Це полягає в наступному: зловмисник намагається здійснити автентифікацію, надіславши клієнту будь-який пароль. Потім зловмисник перехоплює пакет запитів на доступ, який

клієнт надсилає на сервер для перевірки облікових даних користувача, та виконує операцію XOR, операндами якої є пароль, за допомогою якого він намагався автентифікуватися, та атрибути зашифрованого пароля входу з захопленого пакету запиту на доступ. Таким чином, він отримує значення MD5 (загальний секрет + аутентифікатор запиту) та аутентифікатор запиту, і може обчислити значення спільного секрету за формулою (1.4).

$$Attributes = (pass\ XOR\ MD5(ShrdSecret + Request\ Authenticator))\ XOR\ pass = MD5(ShrdSecret + Request\ Authenticator), \quad (1.4)$$

де *pass* – пароль користувача;

ShrdSecret – спільний секрет сервера та клієнта;

Request Authenticator – аутентифікатор запиту.

2. Атака на отримання пароля за допомогою пари атрибутів логін-пароль.

Спочатку зловмисник намагається пройти автентифікацію, використовуючи валідне ім’я користувача та будь-який пароль. Наступним кроком атакуючий перехоплює пакет Access-Request, що надходить від клієнта до сервера, та визначає результат MD5(спільний секрет + автентифікатор запиту), як в попередній атаці. Далі зловмисник може відсилати на сервер від імені клієнта пакети Access-Request, використовуючи автентифікатор запиту та MD5(спільний секрет + автентифікатор запиту) та перебирати паролі, при цьому змінюючи значення атрибуту логін-пароль, до тих пір, поки не буде отриманий пакет Access-Accept від сервера, тобто знайдений відповідний логіну пароль. Ця атака можлива за умови відсутності обмежень на кількість запитів на стороні сервера від клієнта. Також для цього виду атаки необхідно, щоб довжина паролю становила менше, ніж 16 символів, оскільки механізм потокового шифру пари логін-пароль використовує зашифровані на минулому етапі дані, якщо ті перевищують 16 байтів виводу. Будь-яка потужна

автентифікація, наприклад двухфакторна автентифікація, унеможливлює цю атаку.

Рівень безпеки RADIUS залежить від генерації модуля Request Authenticator. Аутентифікатор запитів повинен бути унікальним та непередбачуваним. Багато реалізацій протоколів використовують ненадійні генератори псевдовипадкових чисел для отримання автентифікатора запитів. Для проведення наступних двох атак зловмисникові потрібно, щоб клієнт згенерував певне значення автентифікатора запитів. Зазвичай це не є серйозною проблемою, оскільки це поле не було розроблено як функція захисту. Протокол не описує процес генерації автентифікатора запитів, але найпоширенішим способом його створення є збільшення одного байта для кожного нового запиту та використання цього лічильника як значення цього поля. Таким чином, зловмисник може надсилати додаткові запити, змушуючи лічильник збільшувати значення набагато частіше, ніж у звичайному режимі.

3. Пасивна компрометація пари атрибутів логін-пароль завдяки повторюваному значенню Request Authenticator.

Якщо зловмисник міг контролювати трафік між клієнтом та сервером RADIUS, він міг створити словник, де кожне значення автентифікатора запиту асоціюється із захищеною парою атрибутів логін-пароль. Якщо зловмисник помітить повторну автентифікацію запиту для двох запитів, загальне секретне значення не матиме значення, оскільки XORing двох відомих пар логін-пароль може призвести до перших 16 байт незахищених паролів, для яких використовується операція XOR:

Attributes1 = pass1 XOR *MD5(ShrdSecret + RequestAuthenticator)*

Attributes2 = pass2 XOR *MD5(ShrdSecret + RequestAuthenticator)*

Attributes1 XOR *Attributes2 = pass1* XOR *pass2* XOR *MD5(ShrdSecret + RequestAuthenticator)* XOR *MD5(ShrdSecret + RequestAuthenticator) = pass1* XOR *pass2,* (1.5)

де *pass1, pass2* – паролі користувачів з двох запитів відповідно;

ShrdSecret – спільний секрет сервера та клієнта;

RequestAuthenticator – аутентифікатор запиту.

Успішність цієї атаки залежить від вимог до вибору користувачів паролів. Якщо всі користувачі обрали випадкові паролі однакової довжини, зловмисник не зможе нічого отримати з вилучених даних. Це досі малоймовірне явище. На практиці зазвичай зустрічаються випадки, коли користувачі вибирають паролі різної довжини (як правило, менше 16 символів) та різного ступеня випадковості.

Найпростішим випадком отримання паролів зі знайденого результату операції XOR є випадок, коли паролі значно відрізняються за довжиною. У цьому випадку один з паролів має більше символів, тому символи цього пароля, які виходять за рамки довжини іншого пароля, будуть незмінні (тобто ці символи будуть результатом операції XOR з 0). Це призводить до того, що символи, що не перекриваються довшим паролем, піддаються дії зловмисника без аналізу.

Більше складним є випадок, коли зловмисник припускає, що користувачі обрали паролі з низькою ентропією. У цій ситуації можна здійснити атаку за словником, керуючись статистичним аналізом тих біт, що перекриваються у двох паролів.

Навіть паролі довші за 16 символів підпадають під цю атаку, оскільки зловмисник все одно отримує інформацію про перші 16 символів пароля.

4. Активна компрометація пари атрибутів логін-пароль завдяки повторюваному значенню Request Authenticator.

Ця атака зводиться до того, що зловмисник пробує багато разів пройти автентифікацію, використовуючи різні значення паролів, кожного разу перехоплюючи згенерований клієнтом пакет Access-Request, отримуючи автентифікатор запиту та відповідну йому захищену пару логін-пароль. Наступним кроком атакуючий отримає значення MD5(спільний секрет + автентифікатор запиту), здійснюючи операцію XOR над паролем, який

використовувався при спробах автентифікації та захищеним атрибутом логін-пароль, як показано у формулі (1.4). Зловмисник генерує словник значень, де кожному автентифікатору запита ставиться у відповідність значення MD5(спільний секрет + автентифікатор запиту).

Коли зловмисник перехоплює валідний пакет Access-Request, який містить значення автентифікатора запиту, що присутнє у його словнику, він може здійснити операцію XOR над захищеною парою логін-пароль та значенням MD5(спільний секрет + автентифікатор запиту), що знаходиться у його словнику, таким чином отримуючи значення паролю:

$$MD5(ShrdSecret + RequestAuthenticator)\ XOR\ Attributes = MD5(ShrdSecret + RequestAuthenticator)\ XOR\ pass\ XOR\ MD5(ShrdSecret + RequestAuthenticator) = pass, \quad (1.6)$$

де *pass* – пароль користувача;

ShrdSecret – спільний секрет сервера та клієнта;

Request Authenticator – аутентифікатор запиту.

5. Атака на повторення відповідей від сервера за допомогою повторюваного Request Authenticator.

Зловмисник може скласти словник, де кожному Request Authenticator та ідентифікатору ставиться у відповідність пакет з відповіддю від сервера. Коли зловмисник бачить запит, у якому знаходиться Request Authenticator та пов’язаний з ним ідентифікатор, який присутній у словнику, він може маскуватися як сервер та відсилати відповідь клієнту, яку він спостерігав раніше при складанні словника.

Більш того, якщо серед відповідей сервера у словнику міститься хоч один пакет Access-Accept, зловмисник може здійснити спробу проходження автентифікації, змусивши клієнта створити пакет Access-Request з тим самим Request Authenticator та ідентифікатором, що у словнику відповідають пакету Access-Accept, а потім підмінити відповідь від сервера на ту, що знаходиться у

словнику, таким чином успішно підтвердити автентифікацію клієнта, насправді не знаючи пароля.

6. DOS атака, що можлива завдяки передбачуваному значенню Request Authenticator.

Завдяки тому, що зловмисник може передбачати майбутні значення Request Authenticator, він може діяти як клієнт та створити словник майбутніх значень Request Authenticator та пов’язаних з ними відповідей сервера. Наступним кроком зловмисник від імені сервера на кожен клієнтський запит Access-Reject, створюючи таким чином відмову в обслуговуванні.

7. Оскільки замість атрибута User-Name у повідомленнях RADIUS можуть використовуватися атрибути NAS-IP-Address або NAS-Identifier, які містять відповідно IP адресу NAS або ідентифікатор NAS, то зловмисник, який має доступ до трафіку RADIUS, може визначити географічне розташування учасників протоколу у режимі реального часу.

8. Атака переговорів (Negotiation attack).

У ході цієї атаки NAS, RADIUS проксі-сервер або RADIUS сервер атакуючого намагаються змусити хост, який забезпечує процес автентифікації вибрати менш безпечний метод автентифікації. Наприклад, сеанс автентифікаці, що використовує EAP, замість цього буде використовувати CHAP або PAP; з’єднання, яке було встановлено за допомогою EAP-TLS, під час цієї атаки може бути здійснено за допомогою EAP-MD5-Challenge і т.д.

Крім того, однією із слабких сторін стандарту протоколу є те, що він дозволяє спільно використовувати один і той же пароль із кількома клієнтами, щоб один вразливий клієнт міг взяти під контроль кілька комп'ютерів. Усі клієнти RADIUS з однаковим загальним секретом можуть розглядатися як один клієнт RADIUS для всіх вищезазначених атак, оскільки жоден механізм захисту протоколу не використовує адресу клієнта або іншу унікальну інформацію у своїй роботі.

Обмеження, що існують у структурі спільних секретів, також можуть вплинути на позицію безпеки. Більшість реалізацій клієнта та сервера

дозволяють вводити лише загальні секрети у вигляді рядків ASCII. Є лише 94 різних символів ASCII, які можна ввести зі стандартної американської клавіатури (з 256 можливих). Багато реалізацій протоколів також обмежують загальну довжину спільного секрету до 16 символів або менше. Обидва ці обмеження штучно зменшують розмір значень, які зловмисник повинен шукати, щоб отримати загальний секрет.

Висновки до розділу 1

В першому розділі були розглянуті основні механізми реалізації AAA на базі протоколу RADIUS, проаналізовані вразливості та проблеми реалізації даного протоколу. В результаті можна виділити наступні головні проблеми безпеки даного протоколу:

1. Погана реалізація механізму захисту атрибуту логін-пароль через використання алгоритму хешування MD5 для шифрування пари логін-пароль.
2. Погана реалізація використання Request Authenticator, головними недоліками якої є недостатня ступінь випадковості Request Authenticator та його не унікальність.
3. Перевірка пакету Access-Request на стороні сервера здійснюється тільки при використанні повідомлень розширеного протоколу автентифікації у пакетах протоколу.
4. Вимоги на створення спільних секретів, через які системні адміністратори задають спільні секрети з недостатньою інформаційною ентропією.

Таким чином для покращення механізмів безпеки протоколу RADIUS необхідно замінити алгоритм хешування з MD5 на більш стійкий або додати додаткові операції у процес захисту даних користувача, а також визначити

політику практичної реалізації протоколу, яка регламентувала б вимоги на формат та генерацію Request Authenticator та спільного секрету.

РОЗДІЛ 2. АНАЛІЗ МЕТОДІВ АВТОРИЗАЦІЇ ТА АУТЕНТИФІКАЦІЇ ПРОТОКОЛУ RADIUS

У другому розділі будуть встановленні основні вимоги до протоколів AAA та методи для запобігання атакам, які обумовлені вразливостями протоколу RADIUS, описаними у першому розділі даної роботи.

2.1 Встановлення вимог до протоколів AAA

Насамперед при розробці будь-яких модулів покращення роботи протоколу AAA чи введенні додаткових функцій з цією ж самою метою повинні враховуватися вимоги стандарту RFC2989 «Критерії оцінки протоколів AAA для доступу до мережі».

Для всіх протоколів AAA необхідно визначити основні вимоги до їх реалізації. Можна запропонувати такі вимоги:

1. Масштабованість. Кожна реалізація протоколу повинна мати можливість обробляти мільйони користувачів і десятки тисяч одночасних запитів.

2. Архітектура реалізації протоколу повинна вміти обробляти десятки тисяч пристроїв.

3. Протокол забезпечує механізм автентифікації, цілісності та захисту конфіденційності на рівні передачі даних. Безпека повинна бути забезпечена між двома пристроями передачі даних, а не лише між кінцевими пристроями в ланцюжку процесу автентифікації.

4. У разі невдалого з'єднання з сервером, протокол повинен передбачати механізм зміни доступу до іншого сервера.

5. Повинен бути визначений механізм відновлення стану роботи після втрати даних через несправності, такі як перезавантаження серверів NAS або AAA та втрата зв’язку з NAS/AAA.

6. Повинен бути визначений механізм надсилання запиту із сервера на NAS, що вимагає завершення поточного сеансу авторизації

7. Повинна бути забезпечена конфіденційність на рівні об'єкта, тобто передані дані не можуть бути змінені проміжними вузлами і повинні розшифровуватися лише тими суб’єктами AAA, для яких вони призначені.

8. Контроль за механізмом ретрансляції повинен здійснюватися через конкретну практичну реалізацію, а не за протоколом.

9. Протокол повинен мати можливість розширювання, тобто можливість визначення додаткових атрибутів.

10. Протокол повинен використовувати повторну авторизацію користувача для запиту клієнта.

11. Аудит повинен проводитися в режимі реального часу.

12. Формат даних аудиту повинен бути компактним. Метою цієї вимоги є забезпечення можливості буферизації чи збереження декількох аудиторських пакетів для спільної передачі пізніше.

13. Повинні підтримуватися такі протоколи автентифікації: CHAP, EAP, PAP, а також відкрита передача пароля для систем з одним паролем.

2.2 Аналіз методів покращення алгоритму хешування MD5

Однією з основних проблем безпеки з RADIUS є використання алгоритму хешування MD5 для захисту пароля користувача під час транзиту між клієнтом та сервером..

Зазвичай одним з найкращих варіантів у цьому випадку є заміна ненадійної хеш-функції більш стабільними опціями, такими як SHA256 або

SHA512. Однак використання цих алгоритмів призводить до результуючого періоду довшого, ніж у оригінальній версії алгоритму з використанням алгоритму MD5. У випадку, якщо результат обчислення хеш-функції залежить від сторонніх служб або інших полів бази даних, це може призвести до значних загальносистемних змін, які можуть бути дорогими для власника. У таких ситуаціях застосування додаткових захисних механізмів у існуючих рішеннях може вирішити цю проблему.

Загалом, MD5 - 128-бітова функція, вихідне повідомлення якої розбивається на блоки по 512 біт. Останній блок доповнюється до потрібної довжини, після чого до нього дописується довжина вихідного повідомлення в бітах. В алгоритмі використовується 128-бітовий проміжний стан, яке розбивається на 4 32-розрядних слова. Функція стиснення складається з 4 раундів, в кожному з яких виконується перемішування блоку повідомлення і проміжного стану. Перемішування являє собою комбінацію операцій XOR, AND, OR і операцій циклічного зсуву бітів над 32-бітними словами. У кожному раунді цілий блок повідомлення перемішується з проміжним станом, тому кожне слово повідомлення фактично використовується 4 рази. Після 4 раундів результат і вхідний проміжний стан складаються і виходить вихідне значення функції. Цей алгоритм особливо ефективний в системах з 32-розрядної архітектурою.

Як вже зазначалось у першому розділі даної роботи, MD5 вразливий до трьох видів атак: атаки за словником, атаки за допомогою райдужних таблиць та атак на основі колізій хеш-функцій.

Для забезпечення більш надійного захисту пароля при хешуванні MD5 можуть бути використані наступні методи:

1. Використання модифікатора(солі) при хешуванні.

Сіль – це вторинна інформація, яка складається з рядка символів, який додається до основних даних та хешується. Цей процес робить дані більш стійкими до використання до них райдужних таблиць, оскільки модифіковані

захешовані дані матимуть вищу ентропію, і, отже набагато менше шансів, що ці данні існують у райдужних таблицях.

Існує декілька підходів до використання модифікатора при хешуванні. Перший - використовувати одне і те ж значення для всіх хешів. Цей підхід слід використовувати для різних значень даних, до яких застосовано хеш-функцію. Але оскільки в цьому випадку аргументи хеш-функції є спільним секретом, загальним для кожної пари клієнт-сервер, та Request Authenticator, який можна повторити, висока ймовірність того, що результат хешування буде однаковим.

Інший підхід полягає у використанні для кожної унікальної інформації власного значення модифікатора, яке буде зберігатися разом з відповідними даними у базі даних. Однак, оскільки доступ до бази даних має лише сервер, і процес хешування даних повинен виконуватися на стороні клієнта, який нічого не знає про сіль, такий підхід призведе до нового запиту до сервера, в результаті чого сервер поверне клієнтську сіль цим користувачам. Окрім доступу до сервера, ця опція вимагатиме повторного доступу до бази даних та створення додаткового атрибута в пакеті даних. Крім того, хоча цей метод допоможе протидіяти використанню райдужних таблиць, дані все одно будуть настільки ж вразливими до словникових атак, оскільки сіль буде передаватися у відкритому вигляді мережею.

Інший підхід полягає у використанні випадково сформованого числа як модифікатора для поточного сеансу аутентифікації на основі даних, відомих як клієнту, так і серверу, щоб уникнути передачі модифікатора по відкритому каналу зв'язку. Таким чином, сіль може обчислювати як клієнт, так і сервер. Спільні дані можуть бути загальним секретом між клієнтом і сервером.

2. Ітеративне хешування.

У цьому випадку процес хешування повториться кілька разів. Оскільки md5 - це алгоритм швидкого хешування, для уповільнення процесу хеш-атаки потрібно щонайменше 1000 ітерацій хешування.

3. Попереднє перетворення даних, що подаються на вхід алгоритму хешування.

Цей підхід можна застосовувати наступним чином: перед використанням алгоритму MD5 до результату конкатенації спільного секрету з автентифікатором запиту, це значення може передаватися будь-яким симетричним шифром, де ключем може бути загальний секрет клієнта та сервера. Оскільки загальний секрет та елемент аутентифікації запиту не є значущими текстами, криптоаналіз даних, отриманих після симетричного шифрування даних, можливий тільки за умови повного перебору.

4. Метод ланцюжка.

Метод ланцюжка використовує хеш-таблицю, де крім основних полів для кожного елемента додано ще одне поле, в якому може міститися посилання на будь-який елемент таблиці. Метод працює наступним чином: якщо з якої-небудь причини при формуванні таблиці у внесеного елемента N адреса збігається з адресою вже існуючого елемента E, то елемент N заноситься в таблицю за певним іншою адресою, а в поле посилання елемента E заноситься покажчик на елемент N (це може бути, для прикладу, його адресу або індекс в таблиці). У свою чергу дане поле посилання у елемента N також в силу колізії може виявитися заповненим, тобто виникають ланцюжки з елементів таблиці.

2.3 Аналіз протоколу EAP-MD5 на предмет підвищення його криптостійкості

Атаці за словником також підлягає протокол EAP-MD5, що використовується у якості протоколу автентифікації протоколу RADIUS. Протокол EAP-MD5 є одним із протоколів стандарту IEEE802.1x, що містить у своїй архітектурі аппліканта, автентифікатора та сервер. Незважаючи на те, що на даний момент існують більше безпечні версії протоколу EAP, наприклад EAP-TLS, EAP-TTLS, EAP-FAST, EAP-MD5 досі залишається широкорозповсюдженим протоколом завдяки високій швидкості роботи та

відносно легкій конфігурації. Також EAP-MD5 працює з інструментами доступу даних, які оперують з паролями, які зберігаються у відкритому вигляді такими, як LDAP, SQL, FILE, DBFILE і т.д. У цьому випадку легше оптимізувати процеси захисту даних користувача, ніж повністю змінювати структуру бази даних для того, щоб мати можливість працювати з більш сучасними протоколами. Таким чином підвищення рівня безпеки EAP-MD5 залишається актуальною задачею у сучасних реалізаціях протоколу RADIUS.

Послідовна реалізація протоколу складається з 10 етапів наступним чином:

1. Заявник надсилає запит на початок сесії.

2. Автентифікатор надсилає ідентифікатор розміром 1 байт заявнику.

3. Заявник надсилає своє ім’я користувача, яке є його посвідченням особи, автентифікатору.

4. Автентифікатор надсилає точно таку ж інформацію постачальнику послуг (лише його структура форматування інша).

5. На цьому етапі сервер перевірить, чи є це ім’я користувача у базі даних чи ні. та за наявності - випадковим чином генерує 128-розрядний challenge і надсилає разом з ID + 1 автентифікатору.

6. Аутентифікатор надсилає заявникові абсолютно однакову інформацію (лише його структура форматування інша).

7. На цьому етапі заявник подає три значення ID + 1 та свій пароль для їхнього challenge разом із хешем (MD5) з них та враховує своє ім’я для відповіді. Потім він надсилає цей хеш аутентифікатору.

8. Аутентифікатор надсилає точно таку саму інформацію на сервер (лише її форматування відрізняється).

9. На цьому етапі сервер виконує етап 7 (він також має challenge, пароль та ідентифікатор). Якщо кількість отриманого хешу дорівнює отриманій відповіді на challenge заявника, знак «accept» буде надіслано автентифікатору, інакше знак «reject» буде надіслано автентифікатору.

10. Аутентифікатор надсилає заявникові абсолютно однакову інформацію (лише її форматування відрізняється).

Маючи два з трьох вхідних параметра хеш-функції, наступним кроком атакуючий може здійснити атаку грубої сили, використовуючи словник значень паролів для того, щоб згенерувати хеш від значення ідентифікатора та випадкового параметру challenge отриманих при перехваті трафіку мережі та паролів зі словника по черзі до тих пір, поки не буде знайдений потрібний пароль.

Фактично атакуючий використовує відповідь від сервера для того, щоб перевірити свої здогадки про пароль. Причина простоти процесу злому полягає в тому, що противник має два з трьох вхідних параметрів: ідентифікатор запиту та параметр challenge. То ж якщо один з відкритих параметрів може бути переданий не у відкритому вигляді, зловмиснику буде важче знайти пароль користувача. Так як параметр challenge має більший розмір, то більш доцільно саме цей вхідний параметр хеш-функції передавати мережою у захешованому вигляді.

2.4 Додаткові поради щодо практичної реалізації протоколу RADIUS

Серед додаткових порад щодо покращення рівня безпеки протоколу можна виділити наступні:

1. Використання захищеного з’єднання для транспортування пакетів RADIUS.

SSL або IPsec можуть виступати в якості таких з'єднань. Або використовування готових рішень для здійснення таких з’єднань. Наприклад, RadSec, який передає пакети RADIUS за допомогою протоколів TCP і TLS.

2. Встановлення вимоги до формату спільного секрету про унікальне значення спільного секрету для кожного окремого RADIUS клієнта.

3. Встановлення вимоги до довжини спільного секрету сервера та клієнта.

Якщо загальний вибір пароля обмежений символами, які можна ввести з клавіатури, рядок символів повинен мати принаймні 22 символи та містити малі, великі, цифрові та спеціальні символи. Це тому, що для отримання значення ентропії 128 біт кожен символ повинен мати ентропію 8 бітів. Коли ви вибираєте символи лише з клавіатури, ентропія 8-бітового символу зменшується до 5.8 біт. Щоб отримати необхідний рівень ентропії, необхідно 22 символи.

4. Використання криптографічно стійких генераторів псевдовипадкових чисел у процесі генерації автентифікатора запиту для забезпечення його непередбачуваності.

Можуть бути використані методи, які були зазначені у якості методів генерації випадкового параметра challenge.

5. Формування пакетів RADIUS таким чином, щоб не всі сервера/проксі-сервера на шляху пакету автентифікації користувача мали доступ до даних цього пакета.

Цю проблему може вирішити клієнт, який зберігає окремий загальний секрет для кожного сервера, що бере участь у трансляції пакета RADIUS. Потім дані, необхідні серверу, можуть бути зашифровані за допомогою спільного пароля клієнта з цим сервером. Але такий підхід вимагає від клієнта знання всього маршруту передачі пакетів та зберігання великої кількості загальних секретів, а отже, і поганої масштабованості такої системи.

Проблему несанкціонованої зміни атрибутів пакетів проксі-серверами в маршруті передачі пакетів RADIUS можна частково вирішити, встановивши політики, що описують атрибути, які можна змінювати, а які – ні. Для атрибутів, які не можуть бути модифіковані, повинні бути визначені механізми перевірки цілісності на стороні клієнта та сервера.

6. Для захисту проти атаки переговорів, коли атакуючий змушує використовувати більш безпечний протокол автентифікації мусить бути визначена політика взаємодії сервера автентифікації з іншими хостами мережі,

згідно з якою сервер автентифікації може здійснювати більш потужний метод автентифікації для одних сервісів, та менш потужний - для інших.

Згідно цієї політики, якщо сервер автентифікації отримує від клієнта пакет з типом протоколу автентифікації з рівнем безпеки нижчим, ніж від нього очікує сервер, то такий пакет повинен бути проігнорований.

7. Задля запобігання підробки ідентифікаційних даних про джерело надходження повідомлення, які міститься у полях NAS-IP-adress, NAS-IPv6-Address, Called-Station-Id, повинні бути визначені механізми захисту цих полів у пакеті RADIUS.

8. Щоб запобігти підробці даних аудиту проксі-серверами для отримання додаткової плати, клієнт і сервер повинні мати доступ до логів автентифікації та забезпечувати порівняння даних аудиту, отриманих з проксі-серверів, з даними логів.

Висновки до розділу 2

В другому розділі були розглянуті методи покращення механізму захисту користувацьких даних у протоколі RADIUS, а також метод підвищення рівня безпеки протоколу EAP-MD5 при використанні його у якості протоколу автентифікації на базі протоколу RADIUS, а також були запропоновані поради для підвищення рівня безпеки протоколу при його практичній реалізації.

РОЗДІЛ 3. СТВОРЕННЯ УДОСКОНАЛЕНИХ АЛГОРИТМІВ МЕХАНІЗМІВ АВТЕНТИФІКАЦІЇ ТА АВТОРИЗАЦІЇ ПРОТОКОЛУ RADIUS

В третьому розділі були запропоновані метод покращення алгоритму хешування MD5, а також алгоритм удосконаленої роботи протоколу EAP-MD5 шляхом зміни етапів автентифікації таким чином, щоб один з відкритих параметрів став закритим.

3.1 Пропозиція щодо покращення алгоритму хешування MD5

Розглядаючи всі вищезазначені методи з попереднього розділу для поліпшення обчислення хеш-функції MD5, можливо отримати такий алгоритм хешування, не вдаючись до реалізації більш складних і повільних методів хешування:

1. Вхідне значення для хешування (у випадку пакета типу Access-Request це результат конкатенації автентифікатора запиту та спільного секрету) перетворюється у значення, отримане за допомогою стовпчастого шифру перестановки, де ключ є загальним секретом.

Припустимо, загальним секретом є "ShrdSecret ", а аутентифікатором запиту є "reqAuthenticator". Ми отримуємо матрицю, яка може бути представлена у вигляді таблиці 3.1, в якій порожні комірки таблиці заповнені символами «*» та «@».

Таблиця 3.1 - Матриця символів спільного секрету та автентифікатора запиту

10	7	1	8	4	3	11	5	2	9	6	12
s	h	a	r	e	d	S	e	c	r	e	t
r	e	q	A	u	t	h	e	n	t	i	c
a	t	o	r	*	@	*	@	*	@	*	@

Застосовуючи стовпчастий шифр перестановки, ми отримуємо такий результат:

aqocn*dt@eu*ee@ei*hetrArrt@sraSh*tc@

2. Наступним кроком обчислюємо значення модифікатора, шляхом посимвольного виконання операції XOR, операндами якої є спільний секрет та строка, отримана із перестановочного шифру. Отримаємо наступний результат:

*salt = aqocn*dt@eu*ee@ei*hetrArrt@sraSh*tc@ XOR ShrdSecret ShrdSecret ShrdSecret = N7#^!N;$!I4*

3. Хешування. Хешування відбувається за кілька ітерацій. Результатом попереднього кроку хешування є вхідним значенням для поточної операції хешування. Розрахунок хеш-значення здійснюється наступним чином:

$HV_0 = MD5(ComplexPass, salt)$

$HV1 = MD5(ComplexPass, HV_0, salt)$

...

...

...

$HV_n = MD5(ComplexPass, HV_{n-1}, salt)$

$HV = HV_0 + HV_1 + \ldots + HV_n,$ (3.1)

де *ComplexPass* – значення отримане в результаті стовпчастого шифру перестановки на першому етапі алгоритму;

salt – значення отримане в результаті операції XOR та спільного секрету на другому етапі алгоритму.

Зазначений вище хеш-алгоритм вирішує проблему атаки, коли зловмисник отримує загальний секрет, ініціюючи процес автентифікації з будь-яким паролем, перехоплює пакет Access-Request і отримує значення хешу. Незважаючи на те, що зловмисник має хеш-значення, неможливо буде отримати загальне секретне значення, оскільки немає сенсу використовувати райдужні таблиці, поки не буде отриманий результат цього алгоритму. Крім того, отримання однакових початкових значень хешу можливо лише в тому випадку, якщо користувачі вибрали однакові паролі, а клієнт згенерував однаковий аутентифікатор запиту.

Припустимо, що всі вибрані користувачами паролі однаково ймовірні, тобто їх частота у зразку однакова. Тоді ймовірність того, що двоє користувачів виберуть однаковий пароль серед усіх можливих, дорівнює:

$$p = \frac{1}{\sum_{i=1}^{n} k^i} = \frac{1}{\sum_{i=1}^{n} 95^i}, \tag{3.2}$$

де n – середня довжина пароля користувача;

k – кількість ASCII символів, які може вводити користувач у якості символів пароля.

Ймовірність того, що клієнт вибере одне і те ж значення аутентифікатора запиту, залежить від вибору алгоритму отримання цього аутентифікатора. Отже, ми отримуємо ймовірність отримання того самого хеш-значення у наведеному алгоритмі:

$$p = p_{reqAuth} * \frac{1}{\sum_{i=1}^{n} 95^i}, \tag{3.3}$$

де n – середня довжина пароля користувача;

k – кількість ASCII символів, які може вводити користувач у якості символів пароля;

$p_{reqAuth}$ – ймовірність обрання сервером однакового значення аутентифікатора запиту.

Отримана ймовірність дуже мала для того, щоб зловмисник зміг перехопити два однакові результати хешування і таким чином отримати будь-які значення на основі отриманих статистичних даних.

Вищевказаний спосіб вирішує проблему застосування райдужних таблиць до отриманого значення хеш-функції, а також колізій внаслідок використанню значення модифікатора. Але все одно можна скористатись словниковою атакою, в якій зловмисник буде змушений пройти всі можливі значення загальної таємниці, тобто перебрати 95 значень (95 – ASCII символи, які можна ввести з клавіатури, тобто встановити як символи для загального секрету клієнта та сервера) і обчислити наступну кількість хешів:

$$hashesNumber = iterations*(95^{\wedge}shSLength), \quad (3.4)$$

де *iterations* – кількість ітерацій хешування;

shSLength – довжина спільного секрету клієнта та сервера.

Вибираючи кількості хеш-ітерацій, які сповільнюватимуть зловмисника шукати правильне загальне значення секрету, також потрібно зважати на те, що це уповільнить продуктивність сервера. При великій кількості запитів до сервера це може призвести до повільної обробки кожного запиту, і тому сервер стає більш вразливим до атак DoS. DoS-атака (від англ. Denial of Service - «відмова в обслуговуванні») - атака, яка використовується для виведення з ладу і злому обчислювальної техніки і створення технічних і економічних труднощів у цілі. Здійснюється за допомогою створення великої кількості запитів і серйозного навантаження на техніку, найчастіше на великі сервера.

Але проблема із передбаченими та повторюваними значеннями автентифікації запиту, де ви можете скласти словник, що містить значення аутентифікації запиту та відповідний хеш, залишається. Щоб вирішити цю

проблему в процесі хешування паролів, ви можете використовувати не значення автентифікатора запиту, а унікальне значення, яке за один сеанс генерує сервер і надсилає клієнту, так званий параметр challenge.

Однак використання параметру challenge не захищає від повної пошукової атаки, оскільки значення challenge передається у відкритому вигляді мережою. Це означає, що зловмисник має додаткове хеш-значення.

3.2 Пропозиція щодо покращення протоколу EAP-MD5

Для реалізації розглянутого в минулому розділі рішення може бути запропонований наступний алгоритм роботи протоколу EAP-MD5:

1. Аплікант надсилає початковий запит.
2. Автентифікатор надсилає ідентифікатор розміром 1 байт аплікату.
3. Аплікант надсилає ім’я та ідентифікатор автентифікатору.
4. Автентифікатор пересилає отриману інформацію апліканта на сервер.
5. Сервер перевіряє наявність користувача та спільний секрет для автентифікатора, від якого прийшов запит, у базі. Якщо такі дані існують, він генерує випадкову строку розміром 16 байт та обчислює значення запиту до сервера у відповідності до (2.5).

$$Request = MD5\ (ID + challenge)\ XOR\ password, \tag{3.5}$$

де *challenge* – випадково згенерована сервером строка;

password – пароль користувача.

6. Автентифікатор надсилає ту ж саму інформацію аплікатну.
7. Отримавши пакет з зашифрованим паролем, аплікант отримує значення зашифрованого параметру challenge у відповідності з (2.6).

Request XOR password = MD5 (ShrdSecret + challenge) XOR password XOR password = MD5 (ShrdSecret + challenge), (3.6)

де *Request* – зашифроване значення паролю, отримане від сервера;

password – пароль користувача;

ShrdSecret – спільний пароль сервера та користувача.

Наступним кроком він генерує значення поточного часу та обчислює відповідь Response згідно з (2.7).

Response = MD5 (hashedChallenge + timestamp), (3.7)

де *hashedChallenge* - отриманий на минулому етапі роботи алгоритму результат хешування сервером параметра challenge;

timestamp – поточний час у мілісекундах.

8. Автентифікатор надсилає інформацію отриману від апліканта на сервер.

9. При отриманні відповіді від клієнта сервер насамперед перевіряє значення відмітки часу. Якщо пройшло багато часу з моменту генерації відповіді клієнта, сервера відповідає пакетом Access-Reject. В іншому випадку сервер обчислює те ж саме значення хеш-функції, що й клієнт. Якщо обчислене значення збігається з тим, що надійшло від клієнта, сервер відповідає пакетом Access-Accept.

Для того, щоб сервер не виконував обчислення хеш-функції для отримання значення *hashedChallenge* ще один раз, на кроці 5 сервер може формувати словник, де кожному ідентифікатору клієнта ставиться у відповідність обчислене значення *MD5 (ID + challenge).* У цьому випадку при отриманні сервером запиту від клієнта з відміткою часу, йому необхідно лише знайти відповідний запис у створеному словнику за ідентифікатором клієнта та обчислити значення хеш-функції з вхідними параметрами *hashedChallenge* та відмітки часу, що прийшла від клієнта.

На рис. 3.1 зображена схема взаємодії аплікaнта, автентифікатора та сервера згідно описаного протоколу.

Таким чином згенероване сервером значення challenge не передається каналом зв’язку, атакуючий не має два з трьох вхідних параметрів хеш-функції, а значить атака перебору паролів для успішного проходження автентифікцації є неефективною.

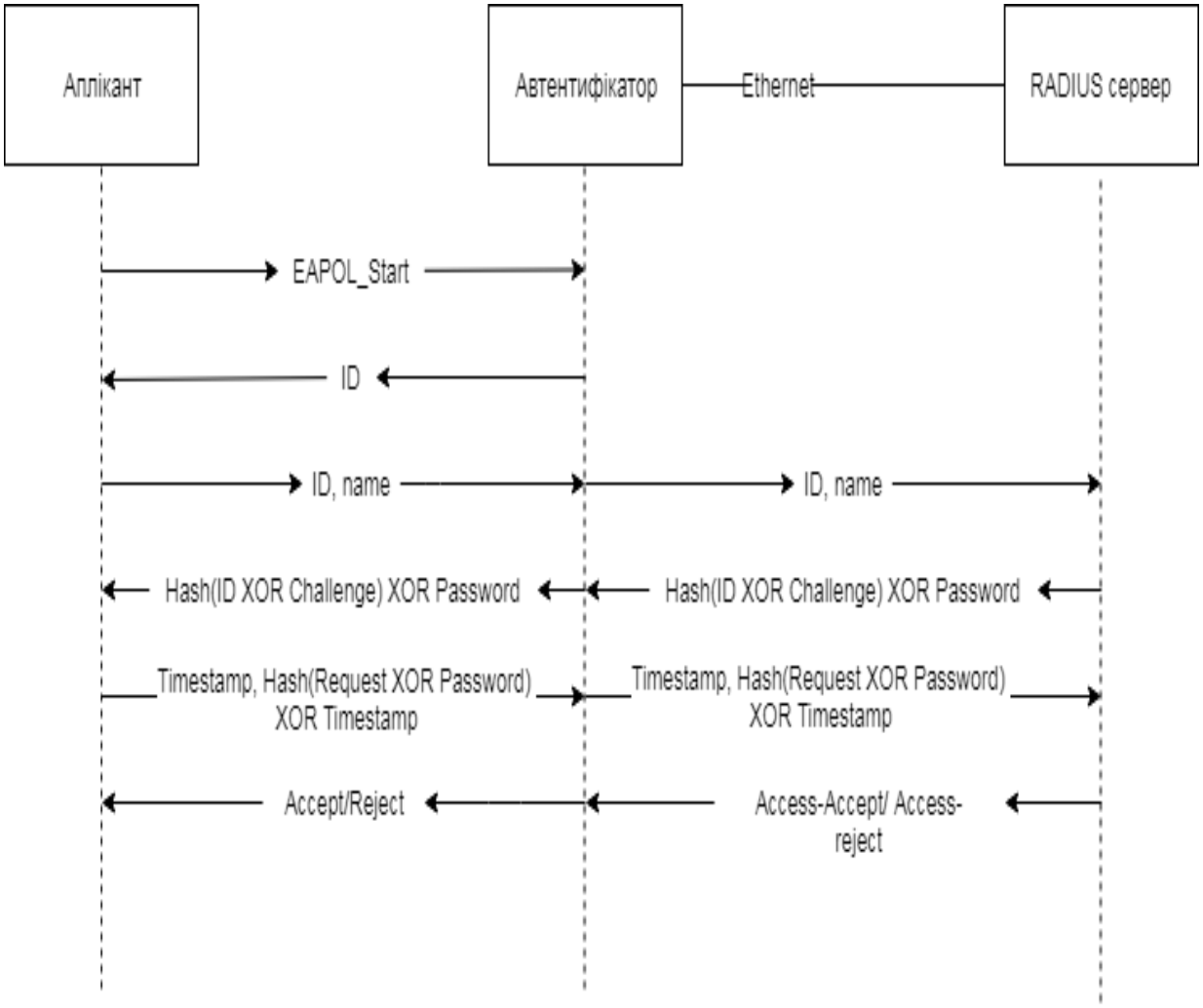

Рисунок 3.1 – Схема взаємодії аплікaнта, аутентифікатора та сервера згідно запропонованому алгоритму аутентифікації

У цьому випадку виникає проблема генерації достатньо випадкового параметру challenge, який важко передбачити. Для генерації криптографічно стійких випадкових даних, наприклад, можна використовувати значення поточного часу, як це відбувається у UNIX-подібних операційних системах для

генерації модифікатора для використання у функції хешування паролей користувачем crypt() для зберігання у файлі /etc/passwd. Або криптографічно безпечний генератор псевдовипадкових чисел з джерел, алгоритми яких явно не можливо спостерігати, наприклад, API генератора випадкових чисел операційної системи сервера. У цьому випадку випадковість отриманих даних забезпечується даними з апаратних джерел (дисперсія шуму вентилятора або жорсткого диску), рухів мишки або генераторів випадкових чисел. Наприклад, ядро Linux (більшість RADIUS серверів працюють на хості з цією операційною системою) генерує ентропію на підставі часу рухів мишки або апаратного генератора випадкових чисел процесора, та робить отримані випадкові дані доступні другим системним процесам за допомогою файлів /dev/random та /dev/urandom, які надають інтерфейс до системного генератора випадкових/псевдовипадкових чисел. При цьому /dev/random – інтерфейс до даних генератора випадкових чисел, який виводить лише випадкові байти, які повністю складаються з бітів шума «хаотичного» пула операційної системи. Якщо пул пустий, інтерфейс не видасть нічого, а програма буде чекати отримання наступного випадково згенерованого байта з пула. На відміну від /dev/random, /dev/urandom поверне стільки байтів, скільки було запитано. Якщо в пулі недостатньо бітів, теоретично можна знайти вразливість алгоритму, який використовує випадкові дані цього пристрою. Але практично нема жодних даних про реалізацію такої атаки.

У Windows задля отримання випадкових даних використовується CryptoAPI, механізм збору ентропії системи схожий на механізм у Linux системі. Але його алгоритм генерації даних є закритим.

Серед інших вимог, які висуваються до процесу генерації параметру challenge потрібно виділити наступні: для кожної окремої сесії генерація параметру challenge відбувається заново. А також згенеровані для різних сесій параметри не повинні залежати один від одного. Механізми, що використовуються для генерації параметра, повинні гарантувати, що розкриття

параметра challenge не призведе до розкриття зловмисником будь-якого іншого параметра для іншої сесії.

Запропонований метод покращення роботи протоколу зберігає швидкість реалізації протоколу завдяки використанню тієї ж самої хеш-функції, але вводить додаткове випадкове значення, яке не передається каналом зв'язку у відкритому вигляді. Такий алгоритм роботи протоколу унеможливлює отримання зловмисником спільного секрету сервера та клієнта, а також створення словника, де кожному значенню аутентифікатора запиту зіставляється значення обчисленої хеш-функції, яке передається каналом зв'язку.

Оскільки значення challenge, що надсилається сервером на першому етапі алгоритму, є випадковим числом та ніколи не передається по мережі у відкритому вигляді, то для отримання паролю користувача, окрім атаки за словником, необхідно також перебрати увесь простір можливих значень параметру challenge.

Оскільки довжина значення параметру challenge дорівнює 16 байтам = 128 бітам, простір можливих значень challenge дорівнює $2^{\wedge 128}$. Тобто якщо складність атаки на оригінальний протокол дорівнює 2^{x} (x залежить від довжини пароля та символів, з яких він складається), то складність атаки проти запропонованого протоколу при таких же умовах складає $2^{(128+x)}$. Якщо усі значення параметру challenge та користувацького пароля рівно ймовірні, тоді потрібні значення challenge та пароля у середньому будуть отримані після $2^{(64+x/2)}$ спроб. У реальному житті паролі користувача не є рівномірно розподіленою величиною. Але навіть враховуючи частоту використання різних значень паролів, величина спроб для отримання потрібного значення пароля є досить великою для того, щоб обчислити її за прийнятний проміжок часу.

Запропонований метод покращення протоколу EAP-MD5 відповідає вимозі стандарту RFC4962 про забезпечення механізмів захисту від атаки відтворення.

У таблиці 3.2 наведено порівняння оригінального протоколу EAP-MD5 та запропонованого покращеного алгоритму.

Таблиця 3.2 - Порівняльний аналіз оригінального та запропонованого алгоритмів

	Швидкість роботи	Стійкість проти атаки відтворення	Складність атаки
Оригінальний протокол	Висока	Не стійкий	2^x
Запропонований протокол	Висока	Стійкий	2^{128+x}

Однією з головних переваг запропонованого протоколу є повторне використання хеш-функції, що збільшує складність проведення атаки. А використання та перевірка відмітки часу запобігає атаці відтворення.

Запропоновані методи покращення ефективності автентифікації та авторизації RADIUS відповідають вимогам стандарту RFC6421 «Вимоги до крипто-динамічності RADIUS», що встановлює вимоги до здатності протоколу пристосовуватися до змін криптографії та вимог безпеки. Наприклад, ці вимоги можуть бути застосовані до нових модульних механізмів, які дозволяють оновити криптографічні алгоритми без істотних порушень в існуючих полях пакетів RADIUS, як це і відбувається у вищезазначених методах. Зокрема запропоновані методи відповідають таким вимогам цього стандарту: запропоновані методи не повинні вимагати нових форматів атрибутів пакету; запропоновані методи повинні підтримувати усі типи пакетів RADIUS.

Висновки до розділу 3

В другому розділі були запропоновані метод покращення механізму захисту користувацьких даних у протоколі RADIUS, а також метод підвищення рівня безпеки протоколу EAP-MD5 при використанні його у якості протоколу автентифікації на базі протоколу RADIUS. Запропонований метод покращує захист протоколу EAP-MD5 від атак на словники і робить їх практично неефективними.

РОЗДІЛ 4. ПРАКТИЧНА РЕАЛІЗАЦІЯ ЗАПРОПОНОВАНИХ ПОКРАЩЕНЬ ПРОТОКОЛУ RADIUS

У четвертій частині роботи буде проаналізовано запропоноване вдосконалення протоколу RADIUS, описано практичну реалізацію модифікованого алгоритму EAP-MD5, механізми, що використовуються для написання практичної реалізації, а також аналіз запропонованого алгоритму за допомогою ProVerif.

4.1 Варіативний аналіз і обґрунтування вибору засобів реалізації

При виборі засобів для розробки програмного проекту необхідно врахувати надзвичайно велику кількість різноманітних аспектів, найбільш важливим із яких є мова програмування, тому що вона в значній мірі визначає інші доступні засоби. При виборі мови програмування основними критеріями були продуктивність програмування, продуктивність роботи додатку та ефективність використання пам’яті. Для розробки програмних систем такого рівня існує три мови, які відповідають вимозі універсальності: Java, C# та Python.

Java — об'єктно-орієнтована мова програмування, випущена компанією Sun Microsystems у 1995 році як основний компонент платформи Java. Зараз мовою займається компанія Oracle, яка придбала Sun Microsystems у 2009 році. Синтаксис мови багато в чому походить від C та C++. У офіційній реалізації, Java програми компілюються у байткод, який при виконанні інтерпретується віртуальною машиною для конкретної платформи. Oracle надає компілятор Java

та віртуальну машину Java, які задовольняють специфікації Java Community Process, під ліцезією GNU General Public License.

C# — об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET. Розроблена Андерсом Гейлсбергом, Скотом Вілтамутом та Пітером Гольде під егідою Microsoft Research (при фірмі Microsoft).

Синтаксис C# близький до C++ і Java. Мова має строгу статичну типізацію, підтримує поліморфізм, перевантаження операторів, вказівники на функції- члени класів, атрибути, події, властивості, винятки, коментарі у форматі XML. Перейнявши багато що від своїх попередників — мов C++, Delphi, Модула і Smalltalk — C#, спираючись на практику їхнього використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем, наприклад множинне спадкування класів (на відміну від C++).

Python — інтерпретована об'єктно-орієнтована мова програмування високого рівня з динамічною семантикою. Розроблена в 1990 році Гвідо ван Россумом. Структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднання існуючих компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні як у скомпільованій так і у вихідній формі на всіх основних платформах. В мові програмування Python підтримується декілька парадигм програмування, зокрема: об'єктно-орієнтована, процедурна, функціональна та аспектно-орієнтована.

Основним недоліком мови C# перед Java та Python є її пропріетарна ліцензія та можливість використання усіх її можливостей в повному обсязі лише на ОС Windows. Оскільки розроблювана система є серверною на повинна витримувати досить високі навантаження бажаною є повна підтримка UNIX-

подібних операційних систем, зокрема GNU/Linux. Тому реальним є вибір між Java та Python.

Однією з основних відмінностей при виборі Python або Java є швидкість. Java швидше ніж Python, це незаперечний факт. Однак варто зазначити, що це може бути застосовано лише для певних проектів - на сьогоднішній день сучасні процесори здатні компенсувати цей недолік повільних мов програмування. Хоча навіть з огляду на сказане, Java вважається самим швидких з цих двох. Основною причиною цього є те, що Python інтерпретована мова і визначає тип даних під час виконання.

Також варто врахувати специфіку нашого програмного модуля – це клієнт/серверна архітектура. Обидві обрані мови можуть реалізувати дану архітектуру, та все ж Java має більш гнучкі засоби, а також зручну роботу з потоками. Виходячи з усього наведеного, вибір мови падає саме на Java.

4.2 Вибір середовища розробки

Для програмування на Java в основному використовують три найвідоміші IDE: NetBeans, Eclipse і IntelliJ IDEA.

NetBeans IDE – вільне інтегроване середовище розробки (IDE) для мов програмування Java, JavaFX, C/C++, PHP, JavaScript, HTML5, Python, Groovy. Середовище може бути встановлене і для підтримки окремих мов, і у повній конфігурації. Середовище розробки NetBeans за умовчанням підтримує розробку для платформ J2SE і J2EE.

Поширюється у сирцевих текстах під ліцензіями GPLv2 і CDDL. Проект NetBeans IDE підтримувався і спонсорувався фірмою Sun Microsystems і після придбання Sun – Oracle, проте розробка NetBean ведеться незалежно співтовариством розробників (NetBeans Community) і компанією NetBeans.Org.

NetBeans IDE доступна для платформ Microsoft Windows, GNU/Linux, FreeBSD, і Solaris (як SPARC, так x86). Для інших платформ доступна можливість зібрати NetBeans самостійно із сирцевих текстів.

За якістю і можливостям останні версії NetBeans IDE змагаються з найкращим інтегрованими середовищами розробки для мови Java, підтримуючи рефакторинг, профілювання, виділення синтаксичних конструкцій кольором, автодоповнення мовних конструкцій на льоту, шаблони коду та інше.

Розробка середовища NetBeans почалася в 1996 під назвою Xelfi (гра букв на основі Delphi), як проект студентів зі створення Java IDE під керівництвом факультету математики і фізики Карлова Університету в Празі. У 1997 році Роман Станек сформував компанію навколо проекту і став випускати комерційні версії середовища NetBeans до передачі всіх прав на IDE корпорації Sun Microsystems в 1999 році. Sun відкрила сирцеві коди середовища розробки NetBeans IDE в червні наступного року. Відтоді спільнота NetBeans постійно розвивається і росте завдяки людям і компаніям, що використовують і підтримує проект.

NetBeans IDE 6.0, створена на основі попередньої версії 5.5.1, надала гнучку підтримку створення модулів для IDE і інтернет-застосунків, заснованих на платформі NetBeans, новий дизайнер користувацьких інтерфейсів (відомий під назвою “Проект Matisse”), нову і перероблену підтримку системи управління версіями CVS, підтримку Weblogic 9 і JBoss 4, і багато покращень в редакторі. NetBeans 6.0 поставляється в складі дистрибутивів Ubuntu 8.04 і Debian.

NetBeans IDE 6.5, випущена в листопаді 2008 року, розширює можливості Java EE (включаючи підтримку Java Persistence, EJB 3 та JAX- WS). Додатково, NetBeans Enterprise Pack підтримує розробку застосунків Java EE 5 Enterprise, включаючи візуальні засоби SOA, засоби для роботи з XML schema, роботу з веб-сервісами (для BPEL), і моделювання на мові UML. Збірка NetBeans IDE Bundle for C/C++ підтримує проекти на мовах C/C++.

NetBeans 7.0, що вийшла у квітні 2011, реалізувала підтримку розробки застосунків з використанням попередньої версії JDK7, були додані засоби для інтеграції з Oracle WebLogic Server 11g і забезпечена підтримка Oracle Database, GlassFish Server Open Source Edition 3.1 і Oracle GlassFish Server 3.1. Версія 7.0 вилучила зі складу модулі з реалізацією засобів розробки мови Ruby і MVC-фреймворка Ruby on Rails. В якості причини названа низька популярність NetBeans серед розробників мовою Ruby.

За заявою Oracle NetBeans IDE 7.1, що вийшов у грудні 2011, став першим середовищем розробки, який повною мірою підтримує останні варіанти специфікацій і стандартів на платформу Java, включаючи повну підтримку циклу розробки з використанням JavaFX і JDK7. Основними нововведеннями NetBeans 7.1 є забезпечення повноцінної підтримки розробки з використанням JavaFX 2.0, значне розширення можливостей Swing GUI Builder, підтримка CSS3, нові інструменти для візуального зневадження інтерфейсу застосунків на базі Swing і JavaFX, інтеграція підтримки Git, додані засоби для інтеграції з Oracle WebLogic Server 12c.

У випуску 7.4 у жовтні 2013 продовжено розвиток засобів для розробки з використанням технологій HTML5, додана підтримка створення гібридних HTML5-застосунків для платформ Android і Apple iOS з використанням фреймворку Apache Cordova, реалізовані засоби використання HTML5 в проектах Java EE і PHP, представлена експериментальна підтримка майбутнього випуску JDK8.

NetBeans 8 вийшов 18 березня 2014. У випуску реалізовані засоби для розробки з використанням Java SE 8, Java SE Embedded 8 і Java ME Embedded 8, розширена підтримка Maven і Java EE з PrimeFaces, додані нові інструменти для HTML5 і, зокрема, фреймворк AngularJS, покращена підтримка PHP (підтримка системи unit-тестування Nette Tester і аналізатора коду PHP-CS-Fixer; поліпшення підтримки Twig, Latte, Neon) і C/C++ (зокрема додана консоль зневаджувача GDB).

Eclipse – вільне модульне інтегроване середовище розробки програмного забезпечення. Розробляється і підтримується Eclipse Foundation і включає проекти, такі як платформа Eclipse, набір інструментів для розробників на мові Java, засоби для управління сирцевими кодами, візуальні побудовники GUI тощо. Написаний в основному на Java, може бути використаний для розробки застосунків на Java і, за допомогою різних плаґінів, на інших мовах програмування, включаючи Ada, C, C++, COBOL, Fortran, Perl, PHP, Python, R, Ruby (включно з Ruby on Rails), Scala, Clojure та Scheme. Середовища розробки зокрема включають Eclipse ADT (Ada Development Toolkit) для Ada, Eclipse CDT для C/C++, Eclipse JDT для Java, Eclipse PDT для PHP.

Випущена на умовах Eclipse Public License, Eclipse є вільним програмним забезпеченням. Він став одним з перших IDE під GNU Classpath. IntelliJ IDEA – комерційне інтегроване середовище розробки для різних мов програмування (Java, Python, Scala, PHP та ін.) від компанії JetBrains. Система поставляється у вигляді урізаної по функціональності безкоштовної версії "Community Edition" і повнофункціональної комерційної версії "Ultimate Edition", для якої активні розробники відкритих проектів мають можливість отримати безкоштовну ліцензію. Вихідний код Community-версії поширюються рамках ліцензії Apache 2.0. Бінарні файли підготовлені для Linux, Mac OS X і Windows.

Перша версія IntelliJ IDEA з’явилася у січні 2001 року й швидко здобула популярність, як перша Java IDE із широким набором інтегрованих інструментів для рефакторингу, що дозволяла програмістам швидко реорганізовувати сирцевий код програм. Дизайн середовища орієнтовано на продуктивність праці програмістів, дозволяючи їм сконцентруватися на розробці функціональності, тоді як IntelliJ IDEA бере на себе виконання рутинних операцій.

Починаючи з шостої версії продукту IntelliJ IDEA надає інтегрований інструментарій для розробки графічного користувацького інтерфейсу.

З версії 9.0 є безплатний варіант Community Edition з відкритими кодами. Сирцеві коди відкритої версії IntelliJ IDEA Community Edition поширюються

рамках ліцензії Apache 2.0. Бінарні пакунки підготовлені для Linux, Mac OS X і Windows.

До складу IntelliJ IDEA включені напрацювання, створені в результаті спільної роботи з компанією Google, яка використовувала IntelliJ IDEA в якості базису для своєї нового відкритого середовища розробки Android Studio. Завдяки співпраці істотно розширені штатні можливості IntelliJ IDEA з розробки застосунків для платформи Android.

Community версія середовища IntelliJ IDEA підтримує інструменти для проведення тестування TestNG і JUnit, системи контролю версій CVS, Subversion, Mercurial і Git, засоби складання Maven і Ant, мови програмування Java, Java ME, Scala, Clojure, Groovy і Dart. Підтримується розробка застосунків для мобільної платформи Android. До складу входить модуль візуального проектування GUI-інтерфейсу Swing UI Designer, XML- редактор, редактор регулярних виразів, система перевірки коректності коду, система контролю за виконанням завдань і доповнення для імпорту та експорту проектів з Eclipse. Доступні засоби інтеграції з системами відстеження помилок JIRA, Trac, Redmine, Pivotal Tracker, GitHub, YouTrack, Lighthouse.

Комерційна версія “Ultimate Edition” відрізняється наявністю підтримки додаткових мов програмування (наприклад, PHP, Ruby, Python, JavaScript, CoffeeScript, HTML, CSS, SQL), підтримкою технологій Java EE, UML-діаграм, підрахунок покриття коду, можливістю роботи з фреймворками (Rails, Grails, Google Web Toolkit, Spring, Play Java Web framework і Hibernate), засобами інтеграції з Perforce, Microsoft Team Foundation Server і Rational ClearCase.

Для розробки програми було обрано IntelliJ IDEA.

4.3 Аналіз швидкості роботи запропонованого алгоритму

У другому розділі пропонується метод для вдосконалення процесу хешування паролів користувача за допомогою симетричного алгоритму шифрування та збільшення кількості ітерацій хешування. Дослідимо швидкість запропонованого алгоритму, змінивши кількість ітерацій у процесі хешування.

Для обчислення результату хешування використовується клас Java MessageDigest, який представляє криптографічну хеш-функцію, яка обчислює хеш з двоїчних даних. У якості вхідних даних цієї функції будемо використовувати значення модифікатора довжиною 16 байтів та значення зашифрованого симетричним шифром паролю довжиною теж 16 байтів. Тести проводились на процесорі Intel® Pentium® N3540 2159 МГц в одному потоці. Отриманий результат зображений на рис. 4.1.

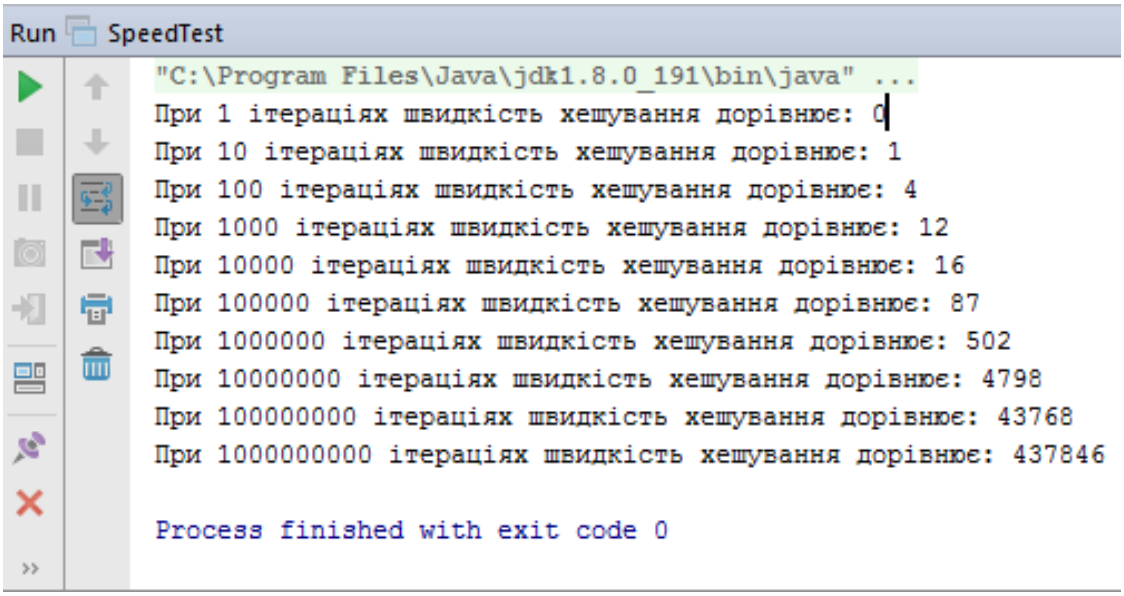

Рисунок 4.1 – Швидкість виконання хешування в мілісекундах при різній кількості ітерацій

Збільшення часу роботи програми стає помітним вже при 10000000 ітераціях, а при 1000000000 час обчислення значення хешу неприпустимо уповільнює загальну роботу програми. На рис. 4.2 зображен графік залежності швидкості роботи програми від кількості ітерацій хешування.

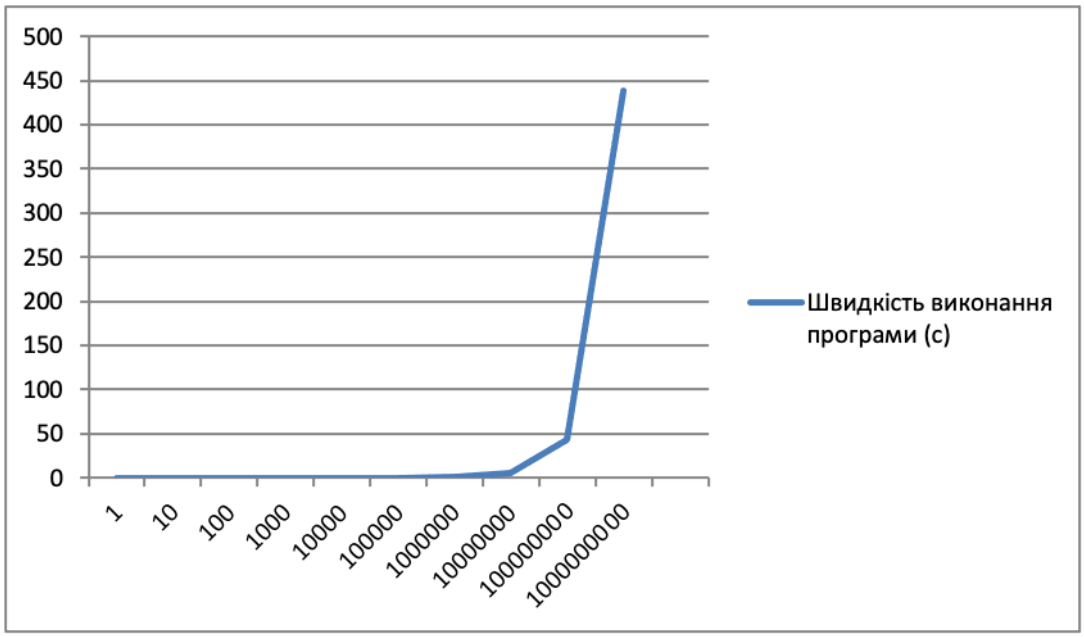

Рисунок 4.2 – Графік залежності швидкості виконання програми від кількості ітерацій хешування

Отримані дані вказують, що оптимальне значення кількості ітерацій для того, щоб це не призвело до помітних затримок у роботі сервера, але уповільнило використання зловмисником початкового значення пароля за словником, приблизно дорівнює 10^6 ітерацій.

4.4 Опис практичної реалізації покращеного алгоритму EAP-MD5

Для реалізації запропонованого у другому розділі методу покращення безпеки протоколу EAP-MD5, була обрана мова програмування Java. У якості апліканта та аутентифікатора виступає один процес, а у якості RADIUS сервера інший. Обидва процеси були запущені на одному хості. Тоді схема взаємодії головних учасників протоколу буде мати вигляд, який можна зобразити за допомогою рисунку 4.3.

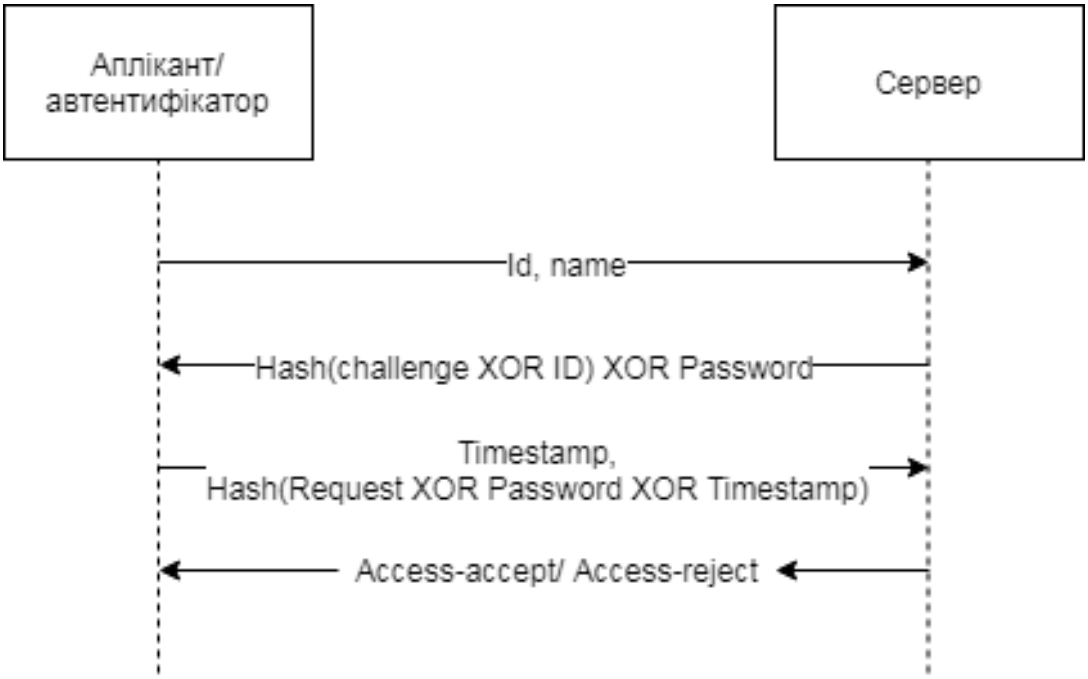

Рисунок 4.3 – Схема взаємодії учасників протоколу EAP-MD5 у практичній реалізації запропонованого алгоритму

Для реалізації прийняття рішення сервером про аутентифікацію на останньому кроку алгоритму була складена блок-схема, яка зображена на рисунку 4.4.

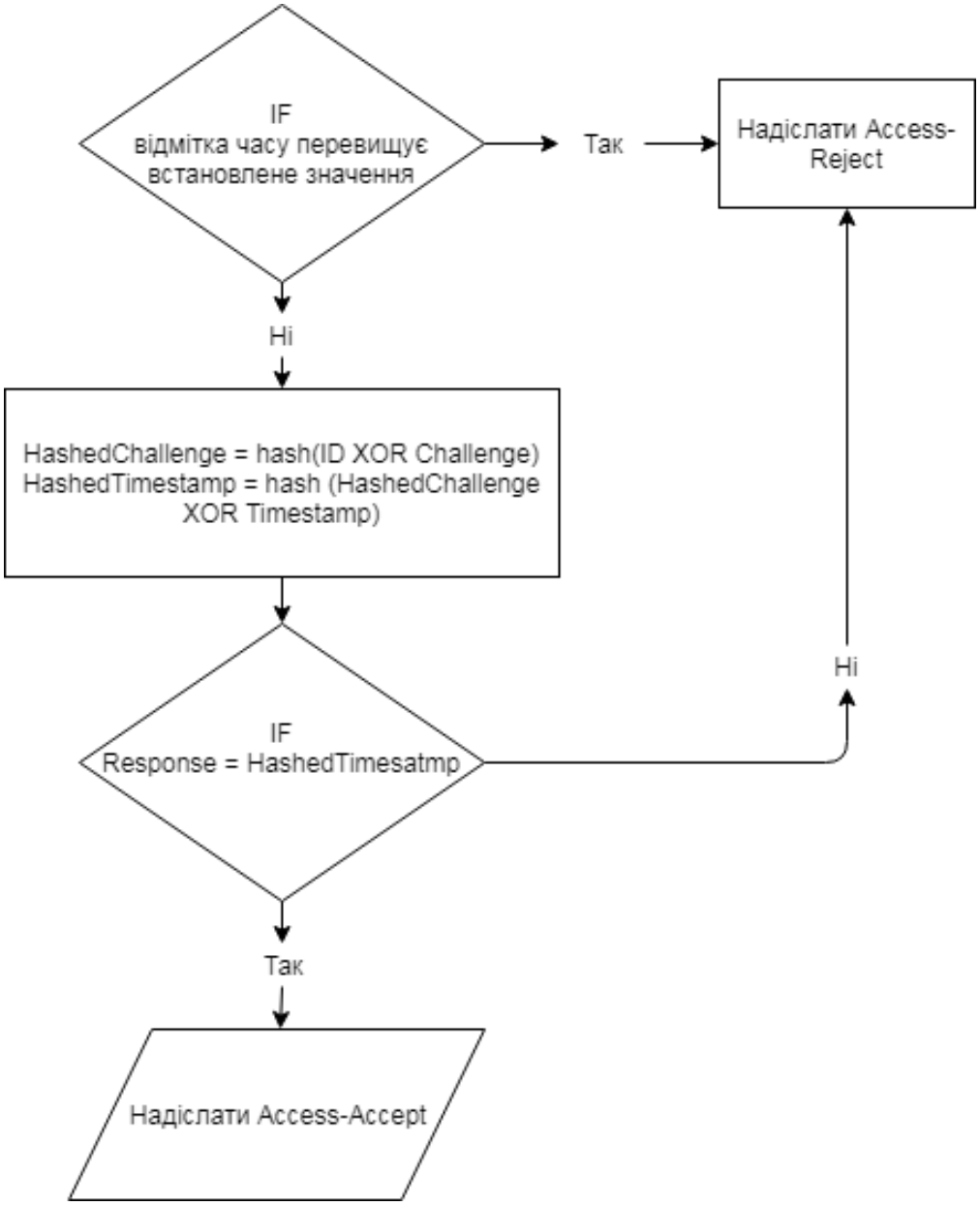

Рисунок 4.4 – Блок-схема прийняття рішення про аутентифікацію сервером RADIUS у протоколі EAP-MD5

Сервер RADIUS представлений у вигляді двох паралельно працюючих потоків: для аутентифікації на порту 1812 та для аудиту на порту 1813. Результат запуску сервера зображений на рис. 4.5.

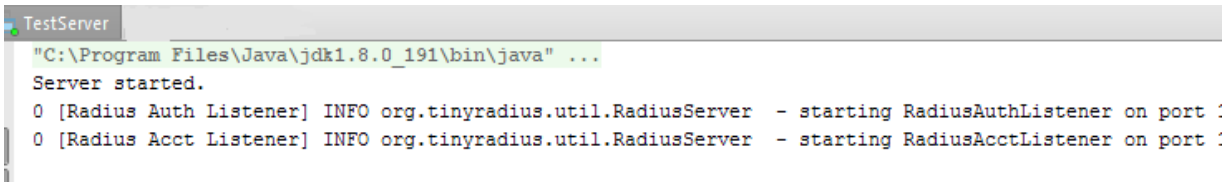

Рисунок 4.5 – Результат запуску сервера RADIUS

Клієнт RADIUS створює пакет типу Access-Request та надсилає його серверу за допомогою методу authenticate() (рис. 4.6).

```
RadiusClient rc = new RadiusClient(host, shared);

AccessRequest ar = new AccessRequest(user, pass);
ar.setAuthProtocol(AccessRequest.AUTH_EAP);
ar.addAttribute( typeName: "NAS-Identifier", value: "this.is.my.nas-identifier.de");
ar.addAttribute( typeName: "NAS-IP-Address", value: "192.168.0.100");
ar.addAttribute( typeName: "Service-Type", value: "Login-User");
ar.addAttribute( typeName: "WISPr-Redirection-URL", value: "http://www.sourceforge.net/");
ar.addAttribute( typeName: "WISPr-Location-ID", value: "net.sourceforge.ap1");
ar.addAttribute( typeName: "EAP-Message", value: "1");

RadiusPacket response = rc.authenticate(ar);
```

Рисунок 4.6 – Створення пакету, яким розпочинається RADIUS conversation

Отримавши запит від клієнта на аутентифікацію користувача, сервер перевіряє, пакет якого протоколу аутентифікації він отримав (PAP, CHAP, EAP) та діє у відповідності з механізмом аутентифікації-авторизації відповідного протоколу. У даному випадку сервер обробляє пакет протоколу EAP-MD5 (рис. 4.7).

```
public RadiusPacket accessRequestReceived(AccessRequest accessRequest, InetSocketAddress client) throws RadiusException {
    String plaintext = getUserPassword(accessRequest.getUserName());
    int type = RadiusPacket.ACCESS_REJECT;
    if (accessRequest.getAuthProtocol().equals(AUTH_EAP)) {
        RadiusPacket eapResponse = new RadiusPacket();
        EAPAuthenticator authenticator = new EAPAuthenticator();
        if (accessRequest.getPacketIdentifier() == 1) {
            accessRequest.removeAttribute(accessRequest.getAttribute( type: "EAP-Message"));
            eapResponse = authenticator.processRequest(accessRequest, plaintext);
            return eapResponse;
        } else if (accessRequest.getPacketIdentifier() == 3) {
            RadiusPacket authResult = authenticator.verifyEapRequest(accessRequest)
                    ? new RadiusPacket(ACCESS_ACCEPT) : new RadiusPacket(ACCESS_REJECT);
            return authResult;
        }

    } else if (accessRequest.getAuthProtocol().equals(AUTH_PAP)) {
        handlePAPPacket();
    } else if (accessRequest.getAuthProtocol().equals(AUTH_CHAP)) {
        handleCHAPPacket();
    } else if (accessRequest.getAuthProtocol().equals(AUTH_MS_CHAP_V2)) {
        handleMSCHAPPacket();
    }

    RadiusPacket answer = new RadiusPacket(type, accessRequest.getPacketIdentifier());
    copyProxyState(accessRequest, answer);
    return answer;
}
```

Рисунок 4.7 – Обробка сервером початкового пакету клієнта

Для формування відповіді клієнту сервер генерує значення challenge.

Для генерації значення challenge був обраний java.security.SecureRandom класс. Цей клас надає криптографічно стійкий генератор випадкових чисел. Згенероване цим класом випадкове число відповідає тестам генератора статистично випадкових чисел, визначеним у стандарті FIPS 140-2 «Вимоги безпеки до криптографічних модулів». На відміну від класу java.util.Random, який використовує лінійний конгруентний генератор для генерації випадкових чисел, SecureRandom клас використовує у своїй роботі криптографічно стійкий генератор псевдовипадкових чисел. Кожен екземпляр цього класу створюється з початковим значенням зерна, яке є основою для генерації випадкового числа та змінюється кожен раз при генерації нового значення.

Наступним кроком відбувається хешування значення параметра challenge та Id клієнта та операція XOR, операндами якої виступають захешоване значення параметру challenge та пароль користувача, отриманий сервером з бази даних за допомогою імені користувача. Усі вищезазначені методи

знаходяться у класі EAPAuthenticator, який відповідає за надання функціоналу для здійснення аутентифікації за допомогою EAP (рис.4.8).

```
private static SecureRandom random = new SecureRandom();
private MessageDigest md5Digest = null;

public RadiusPacket processRequest(RadiusPacket p, String password) {
    byte[] challenge = createChallenge(p.getAttributeValue( type: "User-Name"));
    byte[] encodedChallenge = encodePassword(challenge, password);
    p.addAttribute( typeName: "EAP-Message", encodedChallenge);
    return p;
}

private byte[] createChallenge(String username) {
    byte[] challenge = new byte[16];
    random.nextBytes(challenge);
    challenges.put(username, challenge);
    return challenge;
}

private byte[] encodePassword(byte[] challenge, String password) {
    byte[] eapHash = encodeChallenge(challenge);
    byte[] passwordArray = new byte[eapHash.length];
    System.arraycopy(
            RadiusUtil.getUtf8Bytes(password),
             srcPos: 0,
            passwordArray,
             destPos: 0,
            RadiusUtil.getUtf8Bytes(password).length);
    byte[] responseHash = new byte[eapHash.length];
    for (int i = 0; i < eapHash.length; i++) responseHash[i] = (byte) (eapHash[i] ^ passwordArray[i]);
    return responseHash;
}

private byte[] encodeChallenge(byte[] challenge) {
    MessageDigest md5 = getMd5Digest();
    md5.reset();
    md5.update(RadiusUtil.getUtf8Bytes(getId()));
    md5.update(challenge);
    return md5.digest();
}
```

Рисунок 4.8 – Методи для перевірки клієнтських пакетів на сервері

Отримавши пакет з зашифрованим паролем користувач генерує фінальний запит за допомогою метода generateTimestampPacket(), а поточну відмітку часу розміщує у полі «Event-Timestamp».

Отримавши пакет від клієнта сервер в першу чергу за допомогою методу verifyEapRequest() перевіряє, скільки часу пройшло з моменту відправки пакету клієнтом (рис. 4.9). Це значення не повинно перевищувати встановлене на

сервері значення. У практичній реалізації, опис якої приводиться, це значення дорівнює 300 мілісекундам. У реальному житті для встановлення цього значення повинна враховуватись архітектура мережі та можливі затримки при передачі даних мережою.

```
public boolean verifyEapRequest(RadiusPacket accessRequest) {
    int currentTime = (int)System.currentTimeMillis() / 1000;
    int dateInSec = ByteBuffer.wrap(accessRequest.getAttribute( type: "Event-Timestamp").getAttributeData()).getI
    if((currentTime - dateInSec) > 300) {
        return false;
    }
    String username = accessRequest.getAttributeValue( type: "User-Name");
    byte[] userChallenge = challenges.get(username);
    byte[] hashedChallenge = encodeChallenge(userChallenge);
    return Arrays.equals(accessRequest.getAttribute(EAP_MESSAGE).getAttributeData(),
            encodeTimestampPacket(hashedChallenge, timestamp: dateInSec * 1000) );
}
```

Рисунок 4.9 – Обробка сервером клієнтського пакету з відміткою часу

Для збереження на сервері пар значень «id клієнта – захешований параметр challenge» був використаний класс HashMap. Цей клас використовує хеш-таблицю для зберігання значення. Будь-яке значення у хеш-таблиці зберігається за індексом, який обчислюється знаходженням хеша від заданого ключа. Таким чином забезпечується висока швидкість пошуку потрібного значення серед усіх користувачів, яких на даний час обслуговує сервер.

Результат RADIUS conversation між клієнтом та сервером відповідно до запропонованого алгоритму EAP-MD5 зображений на рисунку 4.10.

```
Request from client:
Access-Request, ID 1
User-Name: mw
NAS-Identifier: this.is.my.nas-identifier.de
NAS-IP-Address: 192.168.0.100
Service-Type: Login-User
Vendor-Specific: WISPr (14122)
  WISPr-Redirection-URL: http://www.sourceforge.net/
Vendor-Specific: WISPr (14122)
  WISPr-Location-ID: net.sourceforge.ap1
EAP-Message: 0x31
```

```
First response from server:
Access-Request, ID 1
User-Name: mw
NAS-Identifier: this.is.my.nas-identifier.de
NAS-IP-Address: 192.168.0.100
Service-Type: Login-User
Vendor-Specific: WISPr (14122)
  WISPr-Redirection-URL: http://www.sourceforge.net/
Vendor-Specific: WISPr (14122)
  WISPr-Location-ID: net.sourceforge.ap1
EAP-Message: 0x9d0e90f253edfd5c61e8d3c7f92e2608

Second response from server:
Access-Accept, ID 3
Reply-Message: Welcome mw!
```

Рисунок 4.10 – Результат RADIUS conversation між клієнтом та сервером

4.5 Аналіз запропонованого рішення за допомогою інструменту Pro Verif

Для аналізу запропонованого протоколу був обраний інструмент ProVerif. ProVerif – це інструмент для автоматичного аналізу безпеки криптографічних протоколів, які можуть бути описані за допомогою моделі загроз Долева-Яо. ProVerif підтримує, але не обмежується лише ними, наступні криптографічні примітиви:

1. Симетричне та асиметричне шифрування;
2. Цифрові підписи;
3. Хеш функції;
4. Протокол bit commitment;
5. Неінтерактивний протокол доказу з нульовим розголошенням (Non-interactive zero-knowledge proof).

За допомогою ProVerif можна перевірити властивості доступності, припущення про відповідність деяким ознакам та еквівалентність твердженням,

таким чином надаючи змогу проаналізувати властивості секретності та процесу автентифікації в протоколах, що аналізуються. Також можуть бути враховані властивості конфіденційності, процесу передачі даних та верифікації. При аналізі протоколу припускається, що при роботі протоколу може бути використана необмежена кількість сеансів та необмежений простір повідомлень. За допомогою ProVerif можна симулювати можливу атаку на протокол.

Для аналізу криптографічного протоколу за допомогою ProVerif необхідно скласти модель криптографічного протоколу, описану за допомогою синтаксису π-обчислень, які описують паралельні обчислення у мережі, конфігурація якої з часом може змінюватись, та властивості безпеки, які необхідно перевірити. Модель протоколу зображена на рис. 3.11.

```
type userId.
type challenge.
type timestamp.
type password.

free c : channel.
free id: userId.
free pass: password[private].
free time: timestamp.

fun md5Challenge (userId, challenge) : bitstring.
fun encPass(bitstring, password) : bitstring.
fun encTimestamp (bitstring, timestamp): bitstring.
fun md5ChallengePassword (bitstring): bitstring.

table d(userId, bitstring).
reduc forall m: bitstring, k: password; decPass(encPass(m, k), k) = m.
reduc forall m: bitstring, k: timestamp; decTimestamp(encTimestamp (m, k), k) = m.

event getId(userId).
event getChallenge(bitstring).
event getTimestamp(timestamp).
event authenticate().

query attacker(pass).

query gotTimestamp: timestamp;event(authenticate()) ==> event(getTimestamp(gotTimestamp)).
```

```
let clientA () =
   out(c, id);
   in(c, hidePass: bitstring);
   out(c, encTimestamp(hidePass, time)).

let serverB () =
   event getId(id);
   in(c, XId: userId);
   new chall: challenge;
   out(c, encPass(md5Challenge(XId, chall), pass));
   in(c, clientTimestamp: bitstring);
   event getTimestamp(time);
   let encChallengePassword = decTimestamp(clientTimestamp, time) in
   if encChallengePassword = md5ChallengePassword(encPass(md5Challenge(XId, chall), pass)) then event authenticate();
   0.

process
   ((!clientA) | (!serverB))
```

Рисунок 4.11 – Модель протоколу відповідно до синтаксису ProVerif

Спочатку описуються типи даних, які використовуються у протоколі. У даному випадку маємо наступні типи даних: клієнтський id clientId; challenge для опису випадкового значення, яке генерується сервером; timestamp для опису поточної відмітки часу та password для опису пароля користувача.

Наступним кроком описуються ті змінні та функції, які використовуються у процесі аутентифікації. Для опису процесу комунікації необхідно визначити канал передачі – змінна типу channel.

Маємо чотири функції: md5Challenge() для хешування параметру challenge; encPass() для операції XOR над паролем користувача та результатом обчислення md5Challenge(); encTimestamp() для операції XOR над зашифрованим паролем та відміткою часу та md5ChallengePassword, яка виконує хешування над захешованим параметром challenge та паролем користувача у якості першого оператора та відмітки часу як другого оператора. Так як операція XOR є зворотною функцією, потрібно визначити зворотні функції для методів encPass() та encTimestamp() за допомогою оператора reduc.

За допомогою оператору query робляться припущення про можливість отримання зловмисником пароля та порядок виконання дій у протоколі, на основі яких ProVerif проводить аналіз даного протоколу. У даному випадку робляться два припущення: про те, що зловмисник не може отримати пароль:

query attacker(pass).

Та про те, що подія, яка позначає аутентифікцаію клієнта, відбудеться тільки після того, як будуть виконані усі попередні кроки:

if encChallengePassword = md5ChallengePassword(encPass(md5Challenge(XId, chall), pass)) then event authenticate();

Далі описуються дії, які виконують clientA та serverB.

При запуску вищезазначеного скрипта маємо результат, який зображений на рисунку 4.12.

```
Process:
(
    {1}!
    {2}out(c, id);
    {3}in(c, hidePass: bitstring);
    {4}out(c, encTimestamp(hidePass,time))
) | (
    {5}!
    {6}event getId(id);
    {7}in(c, XId: userId);
    {8}new chall: challenge;
    {9}out(c, encPass(md5Challenge(XId,chall),pass));
    {10}in(c, clientTimestamp: bitstring);
    {11}event getTimestamp(time);
    {12}let encChallengePassword: bitstring = decTimestamp(clientTimestamp,time)
 in
    {13}if (encChallengePassword = md5ChallengePassword(encPass(md5Challenge(XId
,chall),pass))) then
    {14}event authenticate
)

-- Query not attacker(pass[])
Completing...
Starting query not attacker(pass[])
RESULT not attacker(pass[]) is true.
-- Query event(authenticate) ==> event(getTimestamp(gotTimestamp))
Completing...
Starting query event(authenticate) ==> event(getTimestamp(gotTimestamp))
goal reachable: begin(getTimestamp(time[])) -> end(authenticate)
RESULT event(authenticate) ==> event(getTimestamp(gotTimestamp)) is true.
```

Рисунок 4.12 – Результат роботи ProVerif

Усі припущення справдилися, про що свідчить наступні дві строчки:

```
RESULT not attacker(pass[]) is true.

RESULT event(authenticate) ==> event(getTimestamp(gotTimestamp)) is true.
```

Отримані результати означають, що зловмисник не має доступу до змінної pass, яка містить пароль користувача, та подія authenticate(), що позначає аутентифікацію користувача, відбудеться тільки після того, як будуть пройдені минулі кроки алгоритму.

Висновки до розділу 4

В третьому розділі була проаналізована швидкість роботи запропонованого покращення процесу хешування за допомогою алгоритму MD5 для того, щоб знайти оптимальне значення кількості ітерацій для уповільнення повного перебору значень зловмисником, але без втрати

швидкості роботи основного алгоритму. Таке значення приблизно дорівнює 10^6 ітерацій.

Також була описана детальна схема та практична реалізація запропонованого покращеного алгоритму EAP-MD5, запропонованого у минулому розділі роботи. А також їх аналіз за допомогою існуючого рішення для аналізу криптографічних протоколів. В результаті використання інструменту тестування ProVerif було доведено, що атакуючий не зможе отримати пароль користувача, а автентифікація буде здійснена тільки після пройдешня усіх кроків алгоритму.

ВИСНОВКИ

В ході виконання роботи були ретельно досліджені існуючи проблеми безпеки в сучасних реалізаціях протоколів автентифікації, авторизації та аудиту та розглянуті можливі методи покращення рівня безпеки для запобігання відомим атакам на протокол RADIUS. Розроблена програмна реалізація запропонованого покращеного алгоритму EAP-MD5 мовою програмування Java, що робить дану програмну реалізацію платформо незалежною.

Також були запропоновані загальні поради та встановлені вимоги, які повинні виконуватися при розробці та імплементації будь-яких реалізацій протоколу RADIUS для його безпечного використання.

Був проведений аналіз стійкості отриманої практичної реалізації протоколу за допомогою інструменту ProVerif.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. RFC 6151. Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms. [Електронний ресурс] – Режим доступу: https://tools.ietf.org/html/rfc6151

2. RFC 2865. Remote Authentication Dial In User Service (RADIUS). [Електронний ресурс] – Режим доступу: https://tools.ietf.org/html/rfc2865

3. RFC 2903. Generic AAA Architecture. [Електронний ресурс] – https://tools.ietf.org/html/rfc2903

4. RFC 2866. RADIUS Accounting. [Електронний ресурс] – https://tools.ietf.org/html/rfc2866

5. RFC 4764. RADIUS Accounting. [Електронний ресурс] – https://tools.ietf.org/html/rfc2866

6. RFC 2989. Criteria for Evaluating AAA Protocols for Network Access. [Електронний ресурс] – https://tools.ietf.org/html/rfc2989

7. Design in the Authentication and Billing System Based on Radius and 802.1x Protocol / Chaoyi Chen, Jianyong Zhang, Junli Liu., 2015. – 6с. - (International Symposium on Computers & Informatic) – [Текст]

8. RFC 3579. RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP). [Електронний ресурс] – https://tools.ietf.org/html/rfc3579

9. RFC 3748. Extensible Authentication Protocol (EAP). [Електронний ресурс] – https://tools.ietf.org/html/rfc3748

10. RFC 4086. Randomness Requirements for Security. [Електронний ресурс] – https://tools.ietf.org/html/rfc4086

11. RFC 5931. Extensible Authentication Protocol (EAP) Authentication Using Only a Password. [Електронний ресурс] – https://tools.ietf.org/html/rfc5931

12. RFC 4962. Guidance for Authenticaton, Authorization, and Accounting (AAA) Key Management. [Електронний ресурс] – https://tools.ietf.org/html/rfc4962

13. Jeffrey, H. An Introduction to Mathematical Cryptography / H. Jeffrey, P. Jill, H. Joseph. – Berlin: Springer, 2008

14. Панасенко, С.П. Алгоритмы Шифрования. Специальный Справочник / С.П. Панасенко. — М. : Академический Проект, 2006

15. Алферов А.П. Основы криптографии: учебное пособие./ Алферов А.П М.: Гелиос АРВ, 2002

16. An Analysis of the RADIUS Authentication Protocol. [Електронний ресурс] – https://www.untruth.org/~josh/security/radius/radius-auth.html

17. Security Analysis of MD5 Algorithm in Password Storage / Mary Cindy Ah Kioon, ZhaoShun Wang, Shubra Deb Das., 2013. – 4 с. – (Atlantis Press) – [Текст]

18. How to Break MD5 and Other Hash Functions / Xiaoyun Wang, Hongbo Yu., - 17 с. – [Текст]

19. Comparison of the RADIUS and Diameter Protocols / Mladen Stanke, Mile Sikic., 2008. – 7с. – [Текст]

20. Diffie W. New Directions in Cryptography / W. Diffie and M. Hellman // IEEE Transactions on Information Theory. – 1976, Nov. – V. IT-22, n. 6. – с. 44-54

21. Рябко Б.Я. Криптографические методы защиты информации./ Рябко Б.Я. // Учебное пособие для вузов, 2-е изд. М.: Горячая линия - Телеком, 2012. – 229 с.