МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«ДОСЛІДЖЕННЯ KERBEROS АВТЕНТИФІКАЦІЇ В WINDOWS LSA-СЕРВІСАХ»**

Виконала: студентка 2 курсу, групи 8.1220-з
спеціальності 122 комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми комп'ютерні науки
(назва освітньої програми)
І.А. Данчук
(ініціали та прізвище)
Керівник доцент кафедри комп'ютерних наук, доцент, к.т.н., Борю С.Ю.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри програмної інженерії, доцент, к.ф.-м.н., Лісняк А.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти магістр

Спеціальність 122 комп'ютерні науки
(шифр і назва)

Освітня програма комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук, д.т.н., доцент
_____ Борю С.В.
(підпис)

« 16 » 06 2021 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Данчук Інні Анатоліївні
(прізвище, ім'я та по-батькові)

1. Тема роботи (проекту) Дослідження Kerberos автентифікації в Windows LSA-сервісах

керівник роботи (проекту) Борю Сергій Юрійович, к.т.н., доцент
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 09 » 06 2021 року № 850-с

2. Строк подання студентом роботи 19.11.21 року

3. Вихідні дані до роботи 1. Постановка завдання.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Особливості застосування протоколу безпеки Kerberos
2. Аналіз можливостей Windows LSA - сервісу
3. Дослідження Kerberos автентифікації в Windows LSA - сервісах

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 14. 06. 2021

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	28.09.2021	
2.	Збір вихідних даних.	30.09.2021	
3.	Обробка методичних та теоретичних	01.10.2021	
	джерел.		
4.	Розробка першого розділу.	04.10.2021	
5.	Розробка другого розділу.	17.10.2021	
6.	Оформлення та нормоконтроль		
	кваліфікаційної роботи.	19.11.2021	
7.	Захист кваліфікаційної роботи.	08.12.2021	

Студент _____ І.А. Данчук
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.Ю. Борю
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г. Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Дослідження Kerberos автентифікації в Windows LSA-сервісах»: 73 с., 10 рис., 1 табл., 50 джерел, 3 додатків.

АВТОРИЗАЦІЯ, БЕЗПЕКА, ВАЛІДАЦІЯ, ВЕРІФІКАЦІЯ, КВИТОК, КОРИСТУВАЧ, СЕРВІС, KERBEROS, LSA-SERVER, TGS, TGT.

Об’єкт дослідження – Windows LSA-сервіс.

Мета роботи: дослідити використання Kerberos автентифікації в Windows LSA-сервісах; проаналізувати, на основі створеного проєкту LsaLogonUserTest, як відбувається реалізація Kerberos протоколу в LSA-сервісах.

Методи дослідження: теоретичні: вивчення наукової літератури, форумів сучасних досліджень та публікацій Kerberos аутентифікації, їх аналіз; емпіричні: спостереження та аналіз реалізації Kerberos аутентифікації в LSA-сервісі.

Теоретична значимість дослідження полягає в більш глибокому розумінні роботи Windows системи при user login та методів використання функцій LsaCallAuthenticationPackage та LsaLogonUser в бібліотеці secure32.dll.

Практичне значення роботи полягає у розробці програмного забезпечення та досліджені використання Kerberos аутентифікації в LSA-сервісі.

Дослідження може бути корисним:

– для задач, пов’язаних з віртуалізацією;
– для задач, пов’язаних з автентифікацією;
– як заготівля для інших майбутніх проєктів.

SUMMARY

Master’s Qualification Thesis «Researching of Kerberos Authentication in Windows LSA-services»: 73 pages, 10 figures, 1 table, 50 references, 3 supplements.

AUTHORIZATION, KERBEROS, LSA-SERVICE, SECURITY, SERVER, TGS, TGT, TICKET, USER, VALIDATION, VERIFICATION.

Object of study - Windows LSA-service.

Objective: research using of Kerberos authentication in Windows LSA services; to do analyze the created LsaLogonUserTest project to understand how is implemented the Kerberos protocol in LSA-services.

Research methods: theoretical: research next stuff the scientific literature, the modern research both the forums and publications about Kerberos authentication, their analysis; empirical: observation and analysis of Kerberos authentication implementation in the LSA service.

The theoretical significance of the researching is in a fuller understanding the handling by the Windows system both a user login and using following functions the LsaCallAuthenticationPackage and the LsaLogonUser in the secure32.dll library.

The practical significance of the work is in software development and explored the use of Kerberos authentication in the LSA service.

Research can be useful for following:

– implementation the virtualization features for the systems;
– implementation the authentication features;
– also, it can be used as template for some projects.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ОС	Операційна система
ПЗ	Програмне забезпечення
DSA	Алгоритм цифрового підпису
ECDSA	Алгоритм з відкритим ключем для створення цифрового підпису
MIT	Університет та дослідницький центр, розташований у Кембриджі
KDC	Центр розподілу ключів
LSA	Системний сервіс, за допомогою якого система здійснює створення Logon session на локальному комп’ютері
NTLM	Університет та дослідницький центр, розташований у Кембриджі
SID	Структура даних змінної довжини
SSL	Криптографічний протокол
S/MIME	Стандарт для шифрування та підпису в електронній пошті за допомогою відкритого ключа
PGP	Комп’ютерна програма, а також бібліотека функцій, що дозволяє виконувати операції шифрування та цифрового підпису повідомлень, файлів та іншої інформації
TGS	Сервер видачі мандатів чи дозволів
TGT	Мандат для отримання мандату (концепції)

ВСТУП

З віртуалізацією знайома велика кількість користувачів: самий яскравий приклад – це термінальні служби Windows Server. Термінальний сервер надає свої розрахункові ресурси клієнтам, а клієнтський додаток виконується на сервері, клієнт отримує тільки «зображення», щоб отримати представлення. Така модель доступу дозволяє, по-перше, знизити вимоги до програмно-апаратного забезпечення сторони клієнта, по-друге – зменшити вимоги до пропускної здатності, а по третє – дозволяє підвищити безпеку. Що стосується обладнання – то в якості термінальних клієнтів можуть використовуватися навіть смартфони або старі комп'ютери, вбудовані до Pentium 166, не кажучи вже про спеціалізованих тонких клієнтів. Існують, наприклад, такі тонкі клієнти в форм - факторі розетки Legrand, які монтуються в коробці. На клієнтських робочих місцях достатньо встановити тільки монітор, клавіатуру та мишу – і можна працювати.

Для роботи з термінальним сервером не обов'язково мати високошвидкісне підключення до локальної мережі, цілком достатньо навіть стандартного підключення з пропускною здатністю 15-20 кбіт/с, тому термінальні рішення дуже підходять фірмам, що мають сильно розподілену структуру (за прикладом – мережі невеликих магазинів). Крім того, при використанні тонких клієнтів значно підвищується безпека, тому що користувачем можна запустити лише обмежений набір додатків і заборонити встановлювати власні програми. За принципом, це тільки можна зробити із повноцінними клієнтськими робочими станціями, але з використанням термінальних служб це буде зробити набагато легше

Більш того, ніяку інформацію не можна скопіювати на зовнішній носій та із зовнішнього носія, якщо це явно не дозволено в налаштуваннях термінальних служб. Тобто проблема «вірусів на флешках» відпадає автоматично. Ще одна незаперечна перевага – зниження складності

адміністрування: відбувається оновлення додатка (достатньо оновити їх на сервері), і спрощується робота служби підтримки: до термінальної сесії будь - якого користувача можна підключити видалено без встановлення додаткового програмного налаштування.

Віртуалізація дозволяє скоротити кількість серверів завдяки консолідації, тобто там, де раніше потрібно кілька серверів – тепер можна поставити один сервер, і запустити потрібну кількість гостьових операційний систем у віртуальному середовищі. Це дозволить заощадити на вартості придбання обладнання, а також знизити енергоспоживання, а значить і тепловиділення системи – і, отже, можна використовувати менш потужні, і, відповідно, дешевші системи охолодження. Але ця медаль має і зворотний бік, і не один. Справа в тому, що при впровадженні рішень на базі віртуалізації, швидше за все, доведеться купувати нові сервери, а віртуальні сервери використовують апаратні ресурси фізичного сервера, і, відповідно знадобляться потужніші процесори, більші обсяги оперативної пам’яті, а також швидкісна дискова підсистема, і, швидше за все – більшого обсягу. Крім того, деякі системи віртуалізації (зокрема MS Hyper - V) вимагають підтримки процесором апаратних технологій віртуалізації (Intel VT або AMD-V) та деяких інших функцій процесора.

1 ОСОБЛИВОСТІ ЗАСТОСУВАННЯ ПРОТОКОЛУ БЕЗПЕКИ KERBEROS

1.1 Асиметричне шифрування

Асиметричне шифрування – це метод шифрування даних, передбачуваний використання двох ключів – відкритого та закритого. Відкритий (публічний) ключ застосовується для шифрування інформації і може передаватися через незахищені канали. Закритий (приватний) ключ застосовується для розшифровки даних, зашифрованих відкритим ключем. Відкритий і закритий ключі – це дуже великі числа, пов’язані з іншою певною функцією, але такою, що, знаючи одне дуже складно вирахувати інше.

Асиметричне шифрування використовується для захисту інформації при її передачі, а також на його принципах побудови роботи електронних підписів [1].

Принцип роботи асиметричного шифрування полягає в схемі передачі даних між двома суб’єктами (А і Б) за допомогою відкритого ключа, що виглядає наступним чином:

– суб’єкт А генерує пару ключів, відкритий і закритий (публічний і приватний);

– суб’єкт А передає відкритий ключ суб’єкту Б. Передача може здійснюватися по незахищеним каналам;

– суб’єкт Б шифрує пакет даних при використанні отриманого відкритого ключа і передає його А. Передача може здійснюватися по незахищеним каналам;

– суб’єкт А розшифровує отриману від Б інформацію за допомогою секретного, закритого ключа.

У такій схемі перехоплення будь-яких даних, переданих через незахищені канали, не має сенсу, оскільки встановлювати вихідну інформацію

можливо тільки за допомогою закритого ключа, відомого лише для отримання та не потрібного передачі.

Розглянемо призначення асиметричних алгоритмів. Асиметричне шифрування вирішує головну проблему симетричного методу, при якому для кодування та відновлення використовується один і той же ключ. Якщо передати цей ключ за незахищеним каналом, він може перехопити і отримати доступ до зашифрованих даних. З іншого боку, асиметричні алгоритми набагато повільніше симетричних, тому в багатьох криптосистемах застосовуються як ті так і інші.

Наприклад, стандарти SSL і TLS використовують асиметричний алгоритм на стадії встановлення з'єднань (рукостискання): за допомогою його кодують і передають ключ від симетричного шифру, який використовується в процесі подальших даних.

Також асиметричні алгоритми застосовуються для створення електронних підписів, для підтвердження авторства та (або) цілісності даних. При цьому підпис генерується за допомогою закритого ключа, а перевіряється за допомогою відкритого.

Найпопулярніші алгоритми асиметричного шифрування:

RSA (абревіатура від Рівеста, Шаміра і Адельмана, прізвищ творців алгоритму) – алгоритм, в основі якого лежить обчислювальна складність факторизації (розкладення на множники) великих чисел. Застосовується в захищених протоколах SSL і TLS, стандартах шифрування, наприклад, в PGP і S/MIME. Використовується і для шифрування даних, і для створення цифрових підписів.

DSA (Digital Signature Algorithm, «алгоритм цифрового підпису») – алгоритм, заснований на складності обчислювання дискретних логарифмів. Використовується для генерації цифрових підписів. Слугує частиною стандарту DSS (стандарт цифрового підпису, «стандарт цифрової підписи»).

Схема Ель-Гамаля – алгоритм, заснований на складності обчислення дискретних логарифмів. Лежить в основі DSA і застарілого російського

стандарту ГОСТ 34.10–94. Призначається як для шифрування, так і для створення цифрових підписів.

ECDSA (Elliptic Curve Digital Signature Algorithm) – алгоритм, заснований на складності обчислення дискретного логарифма в групі точок еліптичної кривої. Замінюється для генерації цифрових підписів, зокрема для підтвердження транзакцій у криптовалюті Ripple.

Теоретично приватний ключ від асиметричного шифру, а також відкритий ключ та механізм можна обчислити. Але надійними вважаються шифри, для яких це доцільно з практичної точки зору. Так, на зламування шифру, виконаного за допомогою алгоритму RSA з ключем довжиною 768 біт на комп'ютер з одноядерним процесором AMD Opteron із частотою 2,2 ГГц, що працює в середині 2000-х років, пішло б до 2000 років.

При цьому фактична надійність шифрування залежить в основному від довжини ключа і складності завдань, що лежать в основі алгоритму шифрування, для існуючих технологій.

Оскільки продуктивність обчислювальних машин постійно росте, довжину ключів необхідно час від часу збільшувати. Так, у 1977 - му (рік публікації алгоритму RSA) неможливою з практичною точкою зору вважалася розшифровка повідомлень, закодованих за допомогою ключа довжиною 426 біт, а зараз для шифрування цим методом використовуються ключі від 1024 до 4096 біт, при чому перші вже переходять у категорію ненадійних .

Що стосується ефективності пошуку ключа, то вона незначно міняється з течією часу, але може значно збільшитися з появою кардинально нових технологій (наприклад, квантових комп'ютерів). У цьому випадку може знадобитися пошук альтернативних підходів до шифрування.

1.2 Аналіз базових понять дослідження

Ідентифікація в інформаційних системах – процедура, в результаті виконання якої для суб’єкта ідентифікації виявляється його ідентифікатор, однозначно ідентифікує цього суб’єкта в інформаційній системі. Для виконання процедури ідентифікації в інформаційній системі суб’єкту попередньо повинен бути призначений відповідний ідентифікатор (тобто проведена реєстрація суб’єкта в інформаційній системі) [3].

Процедура ідентифікації безпосередньо пов’язана з автентифікацією: суб’єкт проходить процедуру автентифікації, і якщо автентифікація успішна, то інформаційна система на основі факторів автентифікації визначає ідентифікатор суб’єкта. При цьому достовірність ідентифікації повністю визначається рівнем достовірності виконаної процедури автентифікації.

Присвоєння суб’єкту ідентифікатора (тобто реєстрацію суб’єкта в інформаційній системі) іноді теж називають ідентифікацією.

Нижче наведемо фактори ідентифікації.

– відносини – визначаються як спосіб, яким дві або більше концепції, об’єкти або люди пов’язані або знаходяться в стані зв’язку:
  – співробітник на організаційний підрозділ;
  – брат сиблінгу;
  – громадянин з уряду Entity;
  – клієнт з організаційної одиниці;
– атрибут – це іменований набір значень:
  – Driver License Number;
  – ступінь;
  – ім’я;
– агент – це суб’єкт, уповноважений діяти від імені принципала для виконання транзакції з третьою стороною:
  – адвокат.

Ідентифікація – це будь-який процес, який може відрізнити конкретні цифрові ідентичності в певному контексті від набору анонімності до заданого рівня гарантії (таблиця 1.1) [9].

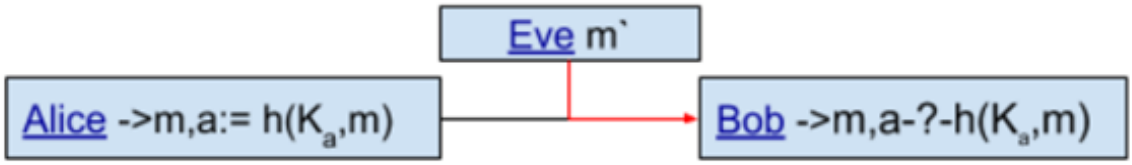
Таблиця 1.1 – Приклади ідентифікації

Варіант ідентифікації	**Фактори автентифікації**	**Результат ідентифікації (Ідентифікатор)**
ідентифікація користувача	1) логін / пароль («я знаю»)	логін
ідентифікація по банківській карті	1) мікропроцесорна банківська карта («я маю»), 2) ПІН-код («я знаю»)	обліковий номер карти (PAN) - зчитується з банківської карти
ідентифікація по банківській карті з біоверифікацією	1) мікропроцесорна банківська карта («я маю»), 2) біометричний фактор (відбиток пальця) («я є»)	обліковий номер карти (PAN) - зчитується з банківської карти
ідентифікація товару по штрих-коду	1) штрих-код («я маю»)	обліковий номер товару
ідентифікація файлу по контрольній сумі	1) контрольна сума («я є»)	Ім’я файлу
ідентифікація громадянина по електронного підпису	1) носій електронного підпису («я маю»), 2) пароль доступу до носія («я знаю»)	ідентифікатор сертифікату (СНІЛС - для сертифіката ключа перевірки підпису)

Автентифікація (AuthN) – це процес встановлення певного рівня гарантії того, що ідентифікація є справжня. Автентифікація для більшості наших цілей є процесом цифрової ідентифікації (Аліса), створивши Assertion із запиту до Verifier (Боба), який використовує методи автентифікації, щоб забезпечити рівень гарантії підтвердження.

Автентифікація включає ідентифікацію, яка необхідна для виконання авторизації, це функція підтвердження законності Позивача (тобто того, що позивач дійсно є суб'єктом, яким він себе називає). Автентифікація – це один з аспектів побудови довіри.

На прикладі проаналізуємо історію користувачів Аліси та Боба (див. рис.1.1). Уявімо історію користувача, де Аліса хоче відправити Бобу повідомлення і Єва буде підслуховувати в зв'язку. Єва могла якось змінити повідомлення. Для цього Єва повинна мати трохи більше контролю над каналом зв'язку, але це зовсім не неможливо. Аліса намагається відправити повідомлення m, але Єва втручається в канал зв'язку, і замість отримання m Боб отримує інше повідомлення m/.



m – повідомлення у вигляді відкритого тексту; h – функція MAC;

Ka – ключ автентицікації (потрібно обмін ключами);

a – код автентифікації повідомлення і розраховується як h (Ka, m)

Рисунок 1.1 – Історія користувача Аліси та Боба

Коли Аліса відправляє повідомлення, вона обчислює код автентифікації повідомлення і відправляє і повідомлення, і код автентифікації, або MAC. Коли Боб отримує повідомлення (код автентифікації повідомлення), Боб обчислює (код автентифікації повідомлення) і порівнює його зі значенням, відправленим Алісою. Боб зрозуміє, що повідомлення неправильне.

Автентифікація – це лише часткове вирішення. Єва все ще може видаляти повідомлення, які надсилає Аліса. Єва також може повторити старі повідомлення або змінити порядок повідомлень.

Процес автентифікації складається з двох основних етапів RFC 4949 [39]:

– крок ідентифікація: подання Assertion значення у вигляді запиту (наприклад, користувач ідентифікатор) до підсистеми автентифікації;

– етап перевірки: подання або створення облікових даних (наприклад, пароля або значення, підписаного закритим ключем), які діють як свідоцтво, яке підтверджує зв’язок між атрибутом і тим, для кого він запитаний.

Методи автентифікації:

– повідомлення (або автентифікації джерела даних) надають одній стороні, яка отримує повідомлення;

– впевненість (за допомогою підтверджуючих доказів) ідентичності боку, що відправила повідомлення.

Автентифікація повідомлення найбільш помітна при використанні коду автентифікації повідомлення. Наприклад, розглянемо необхідність автентифікації джерела даних. «А» відправляє «Б» повідомлення електронної пошти (e - mail). Повідомлення може переміщатися по різних мережевих комунікаційних систем і зберігатися для «Б», щоб отримати його пізніше. «А» та «Б» зазвичай не спілкуються безпосередньо. «Б» хотів би мати будь-які засоби для перевірки того, що повідомлення, отримане і ймовірно створене «А», дійсно було відправлено з «А». Автентифікація джерела даних неявно забезпечує цілісність даних, оскільки, якщо повідомлення було змінене під час передачі, «А» більше не буде відправником .

Автентифікація в контексті управління ідентифікацією та доступом включає:

– перевірку документа: перевірка, що дані правильні і дійсні підтвердженням або перевірка джерела; перевірка цілісності будь-яких

елементів захисту документа, пошук дублікатів, часто використовується в процесах реєстрації та перевірки;

– перевірка справжності облікових даних, може включати:

– форму перевірки документа, в якій облікові дані є контрольованим документом, виданим органом влади;

– форму входу користувача в систему, при якій облікові дані і автентифікатор використовуються для підтвердження того, що облікові дані представлені і контролюються справжнім власником;

– Entity Authentication – це форма входу в систему з використанням облікових даних і автентифікацію. Ця форма навмисно уникає специфікації фізичної особи юридичних осіб в порівнянні з Non - особи суб'єкта;

– федеративної автентифікації: Entity Authentication, де Identity Provider (IDP) є віддаленим або запитується окремо від ресурсу, верифікатор автентифікації повідомляє або стверджує результат автентифікації стороні, що перевіряється [35].

Під час здійснення автентифікації може виникнути ряд проблем, викликаними масовою автентифікацією, тобто коли виклики автентифікації є цільовими обліковими даними багатьох об'єктів.

Запам'ятовування різних складних імен користувачів і паролів не тільки дратує клієнтів, але сучасні хакери досвідченіші, ніж будь - коли раніше, і нікуди не подінуться. Дані не тільки крадуться, але навіть утримуються з метою викупу і становлять серйозну загрозу кібербезпеці, відому як програми-вимагачі. Порушення даних може створювати непотрібне навантаження на IT-персонал, що може призвести до зниження продуктивності на робочому місці.

Завдяки хмарним обчисленням все більше співробітників покладається на мобільне робоче місце, одночасно отримуючи доступ до внутрішніх додатків компанії та програмного забезпечення як послуги (SaaS).

Захист організації від хакерів та шкідливих програм, при цьому пропонуючи користувачам одноманітний інтерфейс, стає дедалі складнішим завданням.

Більшість підприємств також дозволяють співробітникам отримувати доступ до внутрішніх додатків зі своїх різних особистих пристроїв, що наражає на ризик особисті даних певної компанії і може зробити її ресурси вразливими для хакерів та шкідливих програм.

У робочих місць різні потреби, особливо в сьогоднішньому цифровому ландшафті, що постійно змінюється. Деякі з клієнтів можуть регулярно здійснювати транзакції з різних місць, у той час як у разі офісного клерка може бути достатньо ключа автентифікації. Розподілена мережа додатків, що зростають, які використовуються різними клієнтами, посилює необхідність управління безліччю ідентифікаційних даних користувачів, не кажучи вже про додаткові витрати та вимоги до персоналу.

1.3 Основна концепція протоколу Kerberos

Ідентифікація та перевірка справжності користувачів (автентифікація) – це основний засіб захисту інформаційних систем від однієї з головних загроз – стороннього втручання. Якщо у зловмисника немає засобів для нелегального доступу, можливості інформаційного нападу істотно зменшуються [49].

Під ідентифікацією в дослідженні розуміється процедура, за допомогою якої користувач або комп’ютерний процес повідомляє відомості про себе (називає себе). Перевірка справжності, або автентифікація – це процедура перевірки достовірності пред’явлених даних.

Сучасні інформаційні системи являють собою сукупність різнорідних (реалізованих на різних апаратно-програмних платформах), територіально рознесених компонентів.

Як правило, більшості користувачів потрібен доступ до багатьох сервісів, що надаються системами. У той же час ні їм, ні системним адміністраторам не хочеться розмножувати реєстраційну інформацію і особливим чином входити в кожен сервер.

Відповідно до технології клієнт/сервер, вихід полягає у створенні спеціального сервера перевірки автентичності, послугами якого будуть користуватися інші сервери і клієнти інформаційної системи.

На сьогоднішній день на роль фактичного стандарту сервера автентифікації є тільки один реальний претендент – Kerberos, продукт, розроблений в середині 1980-х років в Массачусетському технологічному інституті. Клієнтські компоненти Kerberos присутні в більшості сучасних операційних систем; в такі системи, як Solaris, Kerberos проник досить глибоко, а концерн OSF зробив Kerberos частиною свого розподіленого комп'ютерного середовища (DCE).

Виділення особливого сервера автентифікації дозволяє реалізовувати власні прикладні системи зі своєю системою понять, але зі стандартною процедурою перевірки автентичності, що істотно полегшує управління правами доступу користувачів. Kerberos (точніше, його версія, реалізована в MIT) є вільно поширюваним програмним продуктом, доступним в вихідних текстах (крім, можливо, криптографічних компонентів).

Таким чином, в руках розробників прикладних систем і осіб, відповідальних за інформаційну безпеку, виявляється не "чорний", а "прозорий ящик", що функціонує зрозумілим чином і доступний для "підгонки" з урахуванням потреб конкретної організації.

Постає питання: від чого захищає і від чого не захищає Kerberos?

Kerberos надає засіб перевірки дійсності суб'єктів, що працюють в відкритій (незахищеній) мережі, тобто мережі, що не виключає можливості перехоплення, модифікації і поповнення інформації, що пересилається. Kerberos не покладається на засоби автентифікації, реалізовані в операційних системах мережевих комп'ютерів, на справжність мережевих адрес, на фізичну захищеність мережевих комп'ютерів (крім тих, на яких працює сервер Kerberos).

З точки зору реалізації Kerberos – це один або кілька серверів перевірки автентичності, що функціонують на фізично захищених комп’ютерах. Сервери підтримують базу даних суб’єктів і їх секретних ключів.

В даному документі не розглядається протокол адміністрування, що дозволяє безпечним чином змінювати базу даних суб’єктів, а також протокол реплікації цієї бази.

Kerberos не захищає від:

– атак на доступність, відображення подібних атак (як і реакція на "нормальні" відмови) покладається на користувачів і адміністраторів;

– крадіжки секретних ключів, суб’єкти повинні самі дбати про секретність своїх ключів;

– вгадування паролів.

Погано обраний пароль може не встояти проти підбору за допомогою словника, обчислення по паролю секретного ключа з подальшою спробою розшифрувати отриману від Kerberos інформацію. Відповідним чином змінена програма введення пароля користувача ("троянський кінь") дозволить зловмисникові дізнатися паролі багатьох користувачів.

Таким чином, Kerberos все ж передбачає певний рівень захищеності обслуговуються комп’ютерів:

– повторного використання ідентифікаторів суб’єктів, в принципі можлива ситуація, коли новий суб’єкт Kerberos отримає той же ідентифікатор, що був у раніше вибулого суб’єкта, можливо, що цей ідентифікатор залишився в списках управління доступом будь-якої системи в мережі, в такому випадку новий суб’єкт успадкує права доступу вибулого;

– неузгодженості годин на комп’ютерах, Kerberos покладається на приблизну узгодженість годин мережевих комп’ютерів, подібне припущення спрощує виявлення відтворення (дублювання) інформації, допустима похибка годин може встановлюватися індивідуально для кожного сервера.

1.4 Постановка завдання та практичне застосування Kerberos

Система Kerberos призначена для вирішення наступного завдання. Є відкрита (незахищена) мережа, в вузлах якої зосереджені суб'єкти – користувачі, а також клієнтські і серверні програмні системи. Кожен суб'єкт має секретний ключ. Щоб суб'єкт зміг довести свою справжність суб'єкту «S» (без цього «S» не стане обслуговувати «C»), він повинен не тільки назвати себе, але і продемонструвати знання свого секретного ключа. «C» не може просто послати «S» свій секретний ключ, по-перше, тому, що мережа відкрита (доступна для пасивного і активного прослуховування), а, по-друге, тому, що «S» не знає (і не повинен) знати секретний ключ «C». Тобто потрібен менш прямолінійний спосіб демонстрації знання секретного ключа.

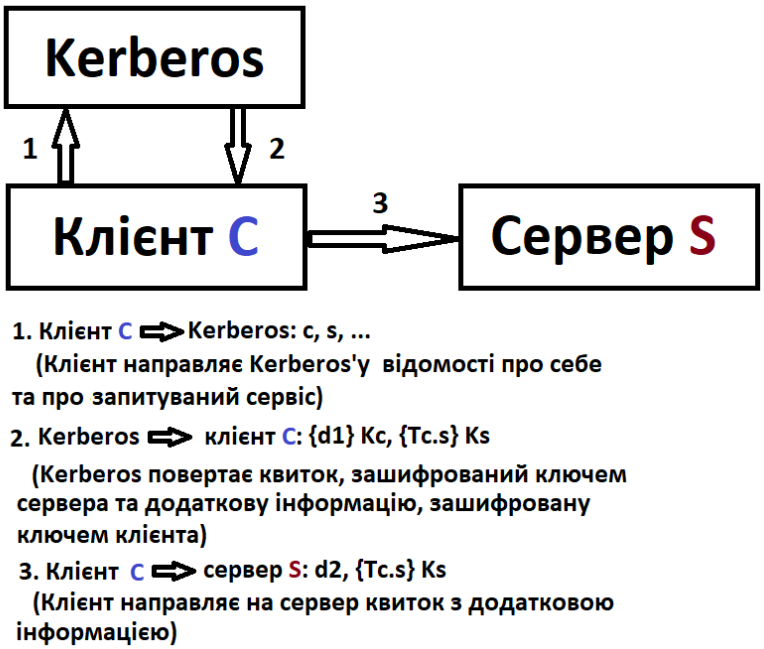
Система Kerberos є надійною третьою стороною (тобто стороною, якій довіряють всі), що володіє секретними ключами обслуговуючих суб'єктів і допомагає їм в попарній перевірці автентичності.

Щоб за допомогою Kerberos отримати доступ до «S» (зазвичай це сервер), «C» (як правило клієнт) посилає Kerberos'у запит, який містить відомості про нього (клієнта) і про запитувані послуги. У відповідь Kerberos повертає дві порції інформації: так званий квиток, зашифрований секретним ключем сервера, і копію частини інформації з квитка, зашифровану секретним ключем клієнта.

Клієнт повинен розшифрувати другу порцію даних і переслати її разом з квитком сервера. Сервер, розшифрувавши квиток, може порівняти його (квитка) вміст з додатковою інформацією, надісланою клієнтом.

Збіг свідчить про те, що клієнт зміг розшифрувати призначені йому дані (адже вміст квитка нікому, крім сервера і Kerberos, недоступно), тобто продемонстрував знання свого секретного ключа. Значить, клієнт – саме той суб'єкт, за кого себе видає.

Описану процедуру проілюстровано на рисунку 1.2.



c і s - відомості (наприклад, ім’я), відповідно, про клієнта і сервері, d1 і d2 - додаткова (по відношенню до квитка) інформація. Tc.s - квиток для клієнта C на обслуговування у сервера S, Kc і Ks - секретні ключі клієнта і сервера, {info} K - інформація info, зашифрована ключем K.

Рисунок 1.2 – Перевірка сервером S автентичності клієнта C (варіант 1)

Підкреслимо, що секретні ключі в процесі перевірки справжності не передавалися по мережі (навіть в зашифрованому вигляді) – вони тільки використовувалися для шифрування. Представлений варіант 1 – вкрай спрощена версія реальної процедури автентифікації. Насамперед, необхідно уточнити структуру запитів, що пересилаються, і відповідей, що повертаються. І квиток, і друга порція інформації, що надається Kerberos, містять випадковий ключ, придатний для шифрування повідомлень, що пересилаються між клієнтом і сервером. Цей ключ називається сеансовим і є загальною секретною інформацією, що розділяється для «C» і «S». Поява подібної загальної інформації та можливість організації конфіденційної взаємодії – важливий побічний продукт автентифікації засобами Kerberos.

Квиток, який видає Kerberos має обмежений термін придатності. Доречно поширити цей термін і на сеансовий ключ. Коли термін придатності закінчується, необхідно провести повторну автентифікацію. Тривалість стандартного терміну придатності залежить від політики безпеки, яку проводить організація. Типовий термін придатності – 8 годин.

Цілком імовірно, що клієнт «C» захоче переконатися в справжності сервера «S», який його обслуговує. Щоб це було можливим, клієнт шифрує додаткову інформацію, що передається серверу, за допомогою сеансового ключа. Сервер може дізнатися про сеансовий ключ, тільки розшифрувавши квиток. Якщо це станеться і сервер поверне клієнту свідчення того, що він зумів прочитати додаткову інформацію, «C» має право вважати справжність сервера «S» встановленою.

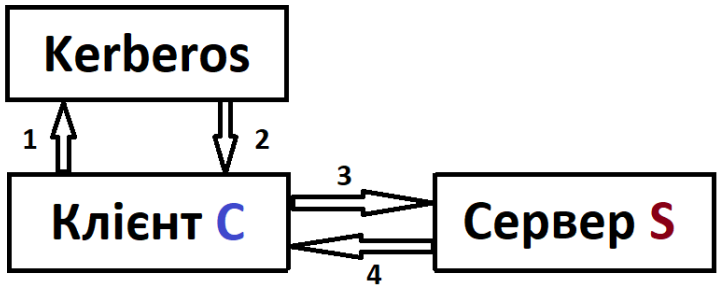
Далі, необхідно захистити інформацію, що пересилається по мережі від відтворення. Якщо цього не зробити, зловмисник зможе надіслати серверу дублікати квитка та додаткової інформації та успішно видати себе за "другого C". Стандартний спосіб захисту від відтворення – включення до неї тимчасових штампів та/або так званих одноразових номерів, що дозволяють переконатися в "свіжості" повідомлень та їх неповторності (і, більше того, в цілісності послідовності повідомлень). Одноразові номери дозволяють асоціювати відповіді з попередніми запитами.

На окреме пояснення заслуговує позначення s1 у першому запиті. Взагалі кажучи, клієнту потрібен не конкретний сервер, а певний сервіс, про що він і просить Kerberos. У відповідь клієнт отримує адресу сервера, здатного надати послугу, що запитується.

Порції інформації в повідомленнях 3 та 4, які шифруються сеансовими ключами, називають автентифікаторами. Вони дозволяють переконатися в справжності показавши їх суб’єкта. Надалі автентифікацію клієнта позначено Ac, автентифікатор сервера – As.

Варіант 2 набагато практичний за варіант 1, однак і він потребує уточнення. Природно припустити, що клієнту знадобляться кілька серверів.

Відповідно, щоб довести свою справжність, клієнту доведеться кілька разів повторювати діалог із Kerberos та використати свій секретний ключ. На жаль, довго тримати напоготові секретний ключ небезпечно його можуть вкрасти. Щоб впоратися із зазначеною проблемою, сервер Kerberos "роздвоюється" на сервер початкової автентифікації (A_S - Authentication Server) та сервер видачі квитків (TGS - Ticket Granting Server). Клієнт запитує у A_S за схемою, зображеною на Рисунку 1.4, квиток до TGS ("квиток на отримання квитків", "квиток на квитки" –TGT, Ticket-Granting Ticket). TGT містить сеансовий ключ для засекречування спілкування між клієнтом та сервером видачі квитків. Квитки до інших серверів клієнт отримує у TGS на підставі "квитка на квитки" (див. рис. 1.3).



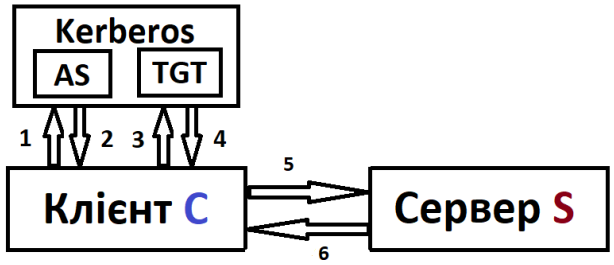
1. Клієнт C ⇨ Kerberos: c, s1, timeexp, n
(Клієнт направляє Kerberos'у відомості про себе , про запитуваний сервіс та додаткову інформацію, що є захистом відтворення)
2. Kerberos ⇨ клієнт C: {Kc.s, s, timeexp, n, ...} Kc, {Tc.s} Ks
(Kerberos повертає квиток, зашифрований ключем сервера, а також сеансовий ключ та додаткову інформацію, зашифровану ключем клієнта)
3. Клієнт C ⇨ сервер S: {ts, ck, ...} Kc.s, {Tc.s} Ks
(Клієнт посилає серверу квиток, а також додаткову інформацію, зашифровану сеансовим ключем)
4. Сервер S ⇨ клієнт C: {ts, ...} Kc.s
(Сервер повертає клієнту додаткову інформацію, зашифровану сеансовим ключем і цим доводить свою справжність)
Tc.s=Kc.s, c, timeexp, ...

timeexp – термін придатності квитка, n – одноразовий номер, Kc.s – сеансовий ключ, ts – тимчасовий штамп, ck – контрольна сума.

Рисунок 1.3 – Взаємна перевірка клієнтом C та сервером S автентичності один одного (варіант 2)

Таким чином, секретний ключ потрібен клієнту лише на короткий час для отримання "квитка на квитки". Надалі використовується сеансовий ключ, який розділяє клієнт і сервер видачі квитків. Компрометація сеансового ключа, що має обмежений термін придатності, річ значно менш небезпечна, ніж розкриття секретного ключа.

A_S, всупереч своїй назві, не перевіряє справжності суб'єктів, що до нього звернулися. Квиток на квитки може отримати будь-хто, у тому числі й зловмисник, але скористатися цим квитком він зможе, тільки якщо знає секретний ключ суб'єкта, від імені якого він виступає. Відзначимо також, що якщо для програмних суб'єктів (клієнтів та серверів) спосіб зберігання секретного ключа не визначено, то для користувачів специфікується алгоритм перетворення пароля на ключ. Іншими словами, користувачеві достатньо пам'ятати лише свій пароль. В результаті схема отримання доступу до сервера набуває вигляду, зображеного на рисунку 1.4.



1. Клієнт C ⇨ AS: c, tgs1, ...
   (Клієнт запитує у сервера аутентицікації "квиток на квитки")
2. AS ⇨ клієнт C: {Kc.tgs, tgs< ...} Kc. {Tc.tgs} Ktgs
   (Сервер аутентицікації видає "квиток на квитки")
3. Клієнт C ⇨ TGS: {Ac}Kc.tgs, {Tc.tgs} Ktgs, s1. ...
   (Клієнт звертається до сервісу видачі квитків з проханням про доступ до сервера S)
4. TGS ⇨ клієнт C: {Kc.s, s, ...} Kc.tgs, {Tc.s} Ks
   (Сервіс видачі квитків повертає клієнту квиток до серверу S)
5. Клієнт C ⇨ сервер S: {As} Kc.s, {Tc.s} Ks
   (Клієнт доводить свою справжність серверу S)
6. Сервер S ⇨ клієнт C: {As} Kc.s
   (Сервер S доводить свою справжнітсь клієнту)

Рисунок 1.4 – Взаємна перевірка клієнтом «C» та сервером «S» автентичності один одного (варіант 3)

Кожен сервер Kerberos обслуговує певну область керування. Вище розглядався випадок, коли клієнт та сервер знаходилися в межах однієї області та взаємодіяли за допомогою локального Kerberos. Тепер звернемося до ситуації, коли клієнт та сервер зареєстровані у різних галузях.

Щоб суб'єкти з різних галузей керування змогли спілкуватися один з одним, між ними має бути укладена угода, а сервери Kerberos мають обмінятися секретними ключами. (Ключі можуть бути різними для різних пар серверів Kerberos.) Оскільки між областями управління Kerberos існують стандартні ієрархічні відносини, кожній області достатньо укласти угоду і обмінятися ключами з батьком і всіма нащадками. У принципі, можливе укладання угоди між двома довільними областями, суб'єкти яких активно спілкуються один з одним. Рисунок 1.5 ілюструє відносини між областями управління.

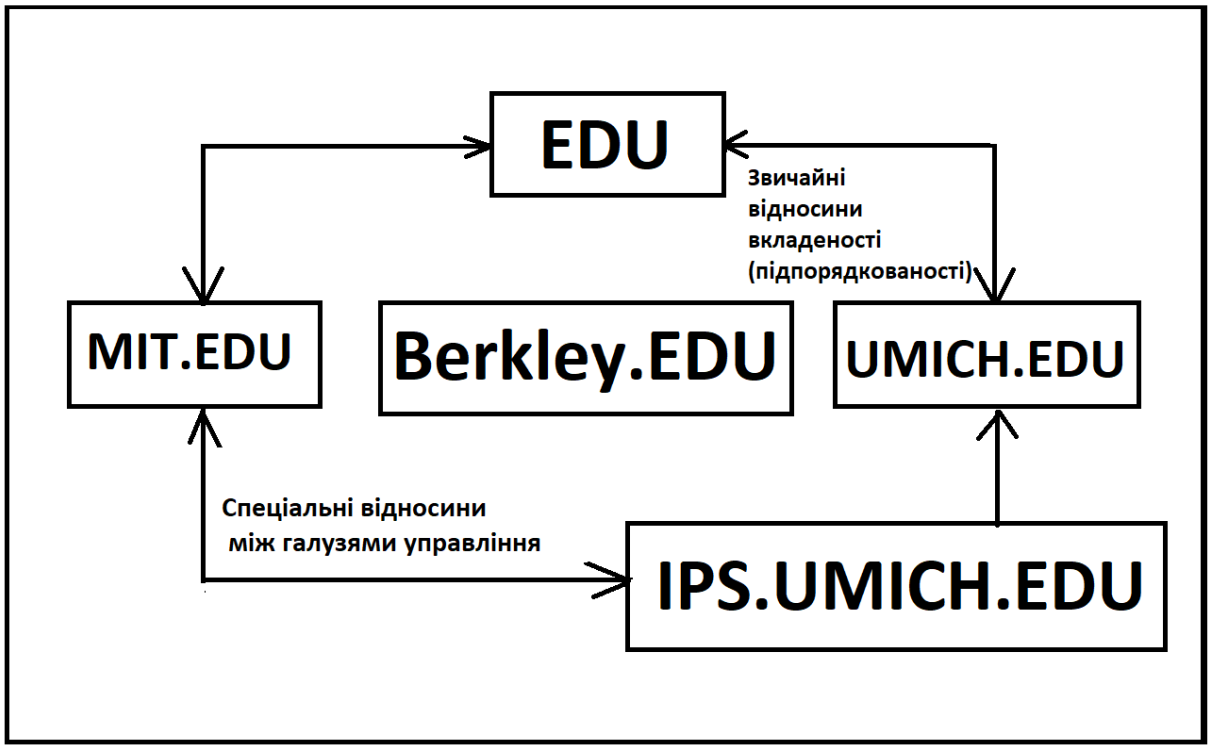
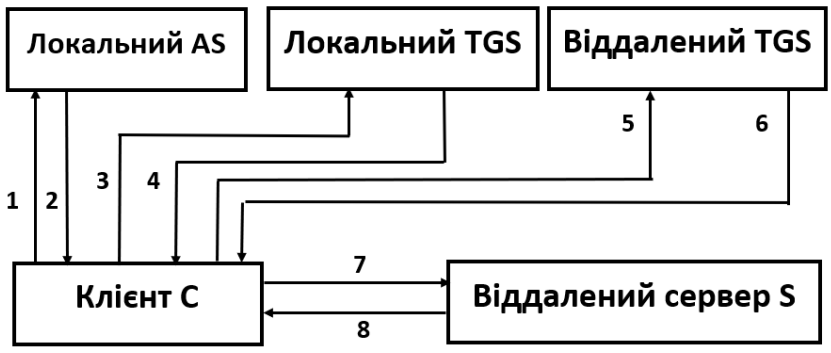


Рисунок 1.5 – Відносини між областями керування Kerberos

За наявності договірних відносин локальний сервер видачі квитків може видати квиток до віддаленого TGS, який, своєю чергою, видає квиток віддаленому серверу. Послідовність обміну повідомленнями цього випадку показано на рисунку 1.6.

У загальному випадку клієнту доведеться пройти через кілька проміжних серверів видачі квитків. Втім, оскільки глибина вкладеності областей управління зазвичай невелика (3 - 4), не має бути довгим і шлях до віддаленого сервера.

Такою є ідейна основа механізму автентифікації, прийнятого в Kerberos. Слід зазначити, що крім викладених, сервер Kerberos надає багато додаткових можливостей. Так, насправді час життя квитка визначається як пара (початковий час, кінцевий час). В результаті можна отримувати квитки на потім, наприклад, для пакетних завдань, що виконуються вночі. Існує механізм квитків з правом передачі взаємної перевірки клієнтом, що дозволяє серверам виконувати певні дії від імені клієнтів, що звернулися до них (наприклад, серверу друку здійснювати доступ до файлів користувача).



1. Клієнт **C** → локальний **AS** : **c, tgs, …**
2. Локальний **AS** → клієнт **C** : **{Kc.tgs, tgs, …} Kc, {Tc.tgs} Ktgs**
3. Клієнт **C** → локальний **TGS** : **{Ac} Kc.tgs, {Tc.tgs} Ktgs, tgs-rem, …**
4. Локальний **TGS** → клієнт **C** : **{Kc.tgs-rem, …} Kc.tgs, {Tc.tgs-rem} Ktgs-rem**
5. Клієнт **C** → віддалений **TGS** : **{Ac} Kc.tgs-rem, {Tc.tgs-rem) Ktgs-rem, s-rem, …**
6. Віддалений **TGS** →Клієнт **C** : **{Kc.s-rem, s-rem, …} Kc.tgs-rem, {Tc.s-rem} Ks-rem**
7. Клієнт **C** → віддалений сервер **S** : **{Ac} Kc.s-rem, {Tc.s-rem} Ks-rem**
8. Віддалений сервер **S** → клієнт **C** : **{As-rem} Kc.s-rem**

Рисунок 1.6 – Взаємна перевірка клієнтом «C» та віддаленим сервером «S»-rem автентичності один одного (варіант 4)

На реалізацію та використання Kerberos можна дивитися з різних точок зору.

– користувач бачить Kerberos як набір команд. (наприклад, kinit – початок Kerberos-сеансу, kdestroy – руйнування Kerberos-квитків та закінчення сеансу, klist – видача поточного списку квитків, kpasswd – зміна Kerberos-паролю тощо);

– користувач не бачить Kerberos взагалі , якщо відповідні дзвінки вбудовані в утиліти та команди операційної системи, такі як login, logout, passwd.

Для програміста Kerberos – набір бібліотек, дуже розумно структурованих, так що можна легко замінювати різні компоненти, наприклад, криптографічні.

Дві функції основної бібліотеки – krb_mk_req та krb_rd_req, дозволяють, відповідно, побудувати автентифікаційний запит до сервера та обробити його. При цьому прикладний програміст позбавляється рутинних аспектів спілкування з Kerberos.

"Керберизація" додатків, тобто вбудовування в них засобів автентифікації, значною мірою зводиться до додавання викликів згаданих функцій, що, природно, не надто складно.

Як зазначено вище, одна з основних завдань віртуалізації полягає в реалізації user login на термінальних службах швидко та безпечно.

На даний момент таким безпечним способом здійснення user login є Kerberos authentication, яка підтримується на усіх нових Windows системах. Тому вивчення та аналіз роботи Kerberos автентифікації в Windows системах є актуальним питанням на сьогодні.

Мета даної кваліфікаційної роботи:

– вивчення API LSA-Service;

– отримання параметрів для виклику API функцій;

– аналіз відповідей отриманих від API LSA-Service;

– створення проєкту LsaLogonUserTest для отримання більш глибоких знань в роботі системи Windows;

– описання припущення реалізації всередині API функції для досягнення Kerberos-автентифікації;

– надання можливості застосування debug для більш глибокого розуміння використання LSA API.

У ході кваліфікаційної роботи були поставлені наступні завдання:

– проаналізувати MSDN документацію [47];

– дослідити опис роботи Kerberos-протоколу на основі іноземної літератури та форумів стосовно проблеми дослідження;

– розробити проєкт LsaLogonUserTest з відкритим доступом;

– експериментально перевірити роботу використання Kerberos автентифікації в Windows LSA-сервісах;

– провести тестування за допомогою консольних додатків:

  – тестового (серверного «ServerApp.exe»);

  – клієнтського «ClientApp.exe»).

Також в кваліфікаційній роботі планувалося значну увагу приділити використанню Unit-тестів. Unit-тести відіграють важливу роль в створенні проєкту, так як за допомогою них в дебазі можна без використання клієнта та сервера на одній робочій машині виявити як формуються параметри для LSA логін юзера, а також як парситься сама відповідь.

Розроблений проєкт планується використовувати як стенд для аналізу та використання системних функцій LSA-сервера.

2 АНАЛІЗ МОЖЛИВОСТЕЙ WSNDOWS LSA-СЕРВІСУ

2.1 Особливості автентифікації в Windows

Автентифікація, як зазначалося вище, є одним із компонентів будь - якої комп'ютерної системи управління доступом. Як показано на рисунку 2.1, системи керування доступом забезпечують ідентифікацію, автентифікацію, авторизацію та звітність.

Рисунок 2.1 – Механізми керування доступом та автентифікації Windows

У процесі ідентифікації використовується набір даних, що унікально ідентифікує об'єкт безпеки (наприклад, користувача, групу, комп'ютер, обліковий запис служби) у спільній службі каталогів. Служба каталогів, така як Active Directory (AD), дозволяє унікально ідентифікувати об'єкти, подібно

до того, як DNS засвідчує, що дві особи не можуть мати однакові адреси електронної пошти. У внутрішніх механізмах Windows використовують SID, глобально унікальні ідентифікатори (globally unique identifier, GUID) та інші унікальні теги. У більшості випадків для ідентифікації достатньо ввести унікальне ім’я облікового запису, наприклад innaDanchuk. У великому обсязі AD доводиться застосовувати повні імена користувачів (user principal name, UPN), наприклад, innaDanchuk@banneretcs.com. При використанні смарт-карток суб’єкт безпеки може надати свій цифровий сертифікат або ключ.

Після того, як суб’єкт безпеки вводить з клавіатури або в інший спосіб надає необхідну для ідентифікації інформацію (наприклад, ім’я користувача, маркер безпеки), він повинен ввести з клавіатури або надати приватну інформацію для автентифікації (наприклад, пароль та PIN-код). У Windows суб’єкт безпеки вводить цю інформацію на екрані реєстрації за допомогою Microsoft Graphical Identification and Authentication DLL (msgina.dll) і Winlogon.exe. Протокол автентифікації та механізм системи кодують подану інформацію на настільному комп’ютері та передають запит автентифікації. Автентифікація Windows може бути базою даних SAM або AD.

База даних SAM (старий спосіб) обслуговує локальні процедури реєстрації та реєстрацію на контролерах домену Windows NT 4.0. AD автентефікує запити у Windows 2000 або доменах пізніших версій цієї операційної системи. Протоколи автентифікації (наприклад, LAN Manager, NT LAN Manager, NTLM, NTLMv2, Kerberos), які можуть використовується для транспортування запитів автентифікації та подальших транзакцій між екраном реєстрації та службою автентифікації.

Нижче кожен протокол автентифікації буде розглянуто окремо.

Розглянемо протокол автентифікації, який повинний виконувати принаймні два завдання:

– по-перше, він повинен безпечно передавати транзакції від запитувача до бази даних автентифікації та на будь-який інший комп’ютер, на якому розміщено відповідний ресурс;

– по-друге, він повинен безпечно та надійно зберігати пароль чи маркер.

Останнє представляє особливий інтерес для хакерів паролів. Протокол автентифікації повинен захистити введену користувачем інформацію при пересиланні до бази даних автентифікації (тобто SAM або AD). Для цього протокол підписує, приховує чи шифрує транзакцію. Крім того, їй надається тимчасова мітка, щоб зломщик не міг скористатися обліковими даними в майбутньому. Щоб не дозволити негайно вийняти пароль користувача з бази даних, протокол повинен забезпечити потайне зберігання паролів у базі даних автентифікації.

Протягом більш ніж десяти років протоколи автентифікації в основному забезпечували захист шляхом збереження паролів у прихованій формі (зазвичай хешованої) у базі даних автентифікації та повної заборони на передачу паролів між запитувачем та базою даних автентифікації простим текстом (навіть у прихованій формі).

Опишемо процес запит-відповідь:

– комп’ютер отримує дані для ідентифікації та автентифікації від користувача та запитує автентифікацію на відповідному сервері;

– сервер автентифікації генерує випадкове довільне значення (зване запитом - Challenge) і посилає його запитувачу;

– запитувач отримує запит і здійснює над ним і прихованою формою пароля математичні операції, а потім передає результат (званий відповіддю - response) серверу автентифікації;

– сервер автентифікації також виконує математичні маніпуляції із запитом методом, ідентичним використовуваному на робочій станції, та порівнює результат з отриманою відповіддю. Якщо результати збігаються, запитувач вважається успішно автентифікованим.

У протоколах автентифікації використовується процес запит - відповідь, тому пароль ніколи не передається через мережу.

При реєстрації користувача одне з перших завдань Windows – визначити, чи процедура стосується лише локальної машини або облікового запису домену. Користувачі, які реєструються від імені локального облікового запису, мають доступ лише до ресурсів комп'ютера, і лише якщо інформація про обліковий запис користувача міститься в локальній базі даних SAM.

Якщо користувачам потрібно звернутися до ресурсів на віддаленому комп'ютері без автентифікації в домені, їх облікові записи повинні бути продубльовано в локальній базі даних SAM кожного доступного комп'ютера. Облікові записи на кожному комп'ютері - учаснику повинні бути синхронізовані (однакові імена реєстрації, паролі та терміни дії облікових даних на всіх машинах). Інакше становище значно ускладняється. Важко обслуговувати однорангові (P2P) мережі середніх розмірів, у яких застосовуються лише локальні процедури реєстрації.

На DC не поширюється вимога синхронізації кількох облікових записів користувачів різних комп'ютерах. Під час доменної автентифікації комп'ютери, зареєстровані в домені, шукають контролери DC, щоб пред'явити облікові дані доменного облікового запису користувача при запитах автентифікації.

Таким чином, якщо віддалений користувач намагається отримати доступ до локального ресурсу якоїсь машини, то цей комп'ютер просить DC перевірити ідентичність користувача, що запитує. Облікові записи користувача домену розміщуються лише на DC і створюються лише один раз.

Будь-який комп'ютер-учасник, якому потрібно засвідчити обліковий запис у домені, може звернутися до контролерів DC у будь-який час. Проблеми синхронізації імен реєстрації, паролів та термінів їх дії не виникає, оскільки облікові дані та керування обліковим записом здійснюються лише в одному місці – на DC. Незалежно від типу реєстрації (локальної або доменної), Windows має автентифікувати запит користувача.

Далі розглянемо протоколи автентифікації Windows. Як зазначалося вище, у Windows застосовується чотири основні протоколи автентифікації:

LAN Manager, NTLM, NTLMv2 та Kerberos. LAN Manager з’явився за часів DOS та продовжував використовуватися з першими версіями Windows. NTLM був випущений разом із NT. Нововведенням пакета оновлень NT Server 4.0 Service Pack 4 (SP4) став NTLMv2, а Windows 2000 привнесла Kerberos. За замовчуванням всі комп’ютери з Windows 2000 та новими операційними системами сумісні з усіма чотирма протоколами автентифікації. Передаючи в ці системи відповідні команди, інші робочі станції та сервери можуть вибирати протокол обробки запиту автентифікації. Системи Windows XP та пізніші з повним набором програмних виправлень сумісні з LM, NTLM та NTLMv2. На платформі Microsoft Kerberos може використовуватися лише клієнтами Windows 2000 (або новими) під час звернення до доменів Windows 2000 (і вище). Комп’ютер з Windows 2000 або нова версія операційної системи повинен мати Kerberos і принаймні ще один з протоколів автентифікації.

Дослідження в галузі безпеки [34] показали, що більш старі протоколи (LM та NTLM) уразливі у разі прослуховування та атак з розгадуванням пароля.

Тому, якщо можливо, рекомендується використовувати лише Kerberos та NTLMv2. Щоб переконатися у правильності цієї поради, слід оцінити можливості кожного протоколу.

IBM розробила протокол LAN Manager, застосувавши його в ранніх версіях Windows і мережах Windows. Як усі протоколи автентифікації Microsoft, LAN Manager генерує хеш паролів (LM hash), який зберігається та використовується відправником та одержувачем у процесі автентифікації. LAN Manager формує LM-хеш, змінюючи всі літери пароля на верхній регістр, розбиваючи пароль на дві 7-символьні половини, а потім шифруючи.

Надалі LM-хеш використовується в декількох послідовних операціях, аналогічних процесу запит-відповідь, описаному вище.

Якщо раніше LAN Manager був цілком прийнятним, то зараз він вважається дуже ненадійним. За допомогою спеціальних інструментів паролі, зашифровані методом хешування LAN Manager, можна лише за кілька секунд

перетворити на простий текст. LM - хешам властиві важливі недоліки, а також є ряд вразливих місць:

– паролі можуть складатися з обмеженої послідовності 128 символів ASCII;

– довжина пароля вбирається у 14 символів;

– якщо пароль містить менше 14 символів, то відсутні символи замінюються легко вгадується хешованої формою, що дозволяє точно визначити довжину пароля;

– перед кешуванням LAN Manager перетворює всі літерні символи пароля у верхній регістр.

Чому LAN Manager досі не вийшов із вжитку?

З метою зворотної сумісності він активний за умовчанням у всіх комп'ютерах Windows, зокрема Windows Server 2003. У новітніх базах даних автентифікації Windows слабкий LM-хеш зберігається поруч із більш надійними просто у разі, якщо доведеться виконати транзакцію LAN Manager. Якщо на підприємстві не використовуються інші програми, що вимагають автентифікації LAN Manager, можна (і потрібно) LAN Manager відключити.

У результаті з'ясувалося, що і NTLM уразливий, і фахівці Microsoft підготували NTLMv2, який досі вважається досить надійним, хоча зараз кращий протокол Kerberos. NTLMv2, як і раніше, широко використовується для локальної реєстрації і в деяких інших випадках. NTLMv2 схожий на NTLM, але в хеші пароля NTLMv2 використовується автентифікація повідомлень HMAC - MD5, а послідовності запит-відповідь надається мітка часу, щоб запобігти атакам, в ході яких зломщик записує облікові дані і згодом їх використовує.

Загалом NTLMv2 більш стійкий до атак із застосуванням «грубої сили», ніж NTLM, оскільки в протоколі застосовується 128-розрядний ключ шифрування. Відомо лише про дві програми злому паролів (одна з них –LC5 компанії Symantec), за допомогою яких вдавалося відкрити хеші паролів NTLMv2.

Компанія Microsoft прийняла Kerberos як протокол доменної автентифікації для доменів Windows 2000, а потім і AD. Kerberos – відкритий стандарт, придатний для взаємодії з сторонніми доменами (званими областями - realm - UNIX і Linux).

Кожен DC у доменах AD відіграє роль сервера розподілу (Kerberos Distribution Server, KDC) і може брати участь у процедурі автентифікації. Безпека підвищується завдяки наступним характеристикам Kerberos:

– взаємна автентифікація між клієнтом та сервером;

– надійний захист пароля, тому що Windows пересилає пароль тільки при початковому зверненні, а не в кожній події автентифікації та всі сеанси зв’язку шифруються;

– послідовність запит - відповідь з міткою часу не дозволяє зломщику використовувати перехоплений пароль після певного часу;

– серверний процес може звертатися до віддаленого ресурсу від імені користувача.

Наведемо короткий опис роботи Kerberos:

– після успішної звичайної автентифікації комп’ютер користувача запитує квиток безпеки із сервера Kerberos (DC) для майбутніх запитів автентифікації;

– сервер Kerberos видає запитувачу квиток для участі у майбутніх подіях автентифікації та авторизації без повторного пред’явлення початкових облікових даних автентифікації;

– коли потрібно звернутися до ресурсу сервера-учасника, він отримує інший квиток доступу від сервера Kerberos і пред’являє його серверу ресурсу для перевірки;

– початкові облікові дані автентифікації не передаються мережевими каналами в жодному з наступних сеансів автентифікації (до тих пір, поки не закінчиться термін дії квитка, виданого на етапі 2).

2.2 Основні засоби безпеки LSA при Kerberos authentication

LSA (Local Security Authority) – центральний компонент підсистеми безпеки в операційній системі Microsoft Windows. Local Security Authority (LSA) відповідає за управління інтерактивним входом в систему [43].

LSA – сервіс, як зазначалось вище, підтримує використання Kerberos протоколу при авторизації.

Нижче розглянемо термінологію основних понять нашого дослідження [20].

Kerberos – мережевий протокол автентифікації, який пропонує механізм взаємної автентифікації клієнта та сервера перед встановленням зв'язку між ними.

Клієнтський комп'ютер – комп'ютер, який потребує доступу до служби, яка підтримує автентифікацію Kerberos.

Сервісний комп'ютер – сервер/комп'ютер, на якому розміщується «Сервіс», який підтримує автентифікацію Kerberos KDC (Центр розповсюдження ключів).

SPN (ім'я учасника служби) – це ім'я служби, пов'язаної з обліковим записом користувача, в якому буде працювати служба. Ця прив'язка виконується в LDAP шляхом встановлення значення для атрибута "servicePrincipalName". SPN має формат типу SERVICE/hostname.

Ticket (квиток) – зашифрований пакет даних, який видається довіреним центром автентифікації KDC.

Зазвичай ці компоненти поставляються як єдина програма, яка запускається на центрі розподілу ключів (KDC містить базу даних паролів для користувачів). Сервер автентифікації виконує одну функцію: отримує запит, який містить ім'я клієнта, що запитує автентифікацію, і повертає йому зашифрований TGT. Потім користувач може використовувати цей TGT для запиту подальших мандатів на інші сервіси.

Коли користувач виконує первинну автентифікацію після успішного підтвердження його автентичності (див. рис. 2.2), KDC видає первинне посвідчення користувача для доступу до мережевих ресурсів - Ticket Granting Ticket (TGT).

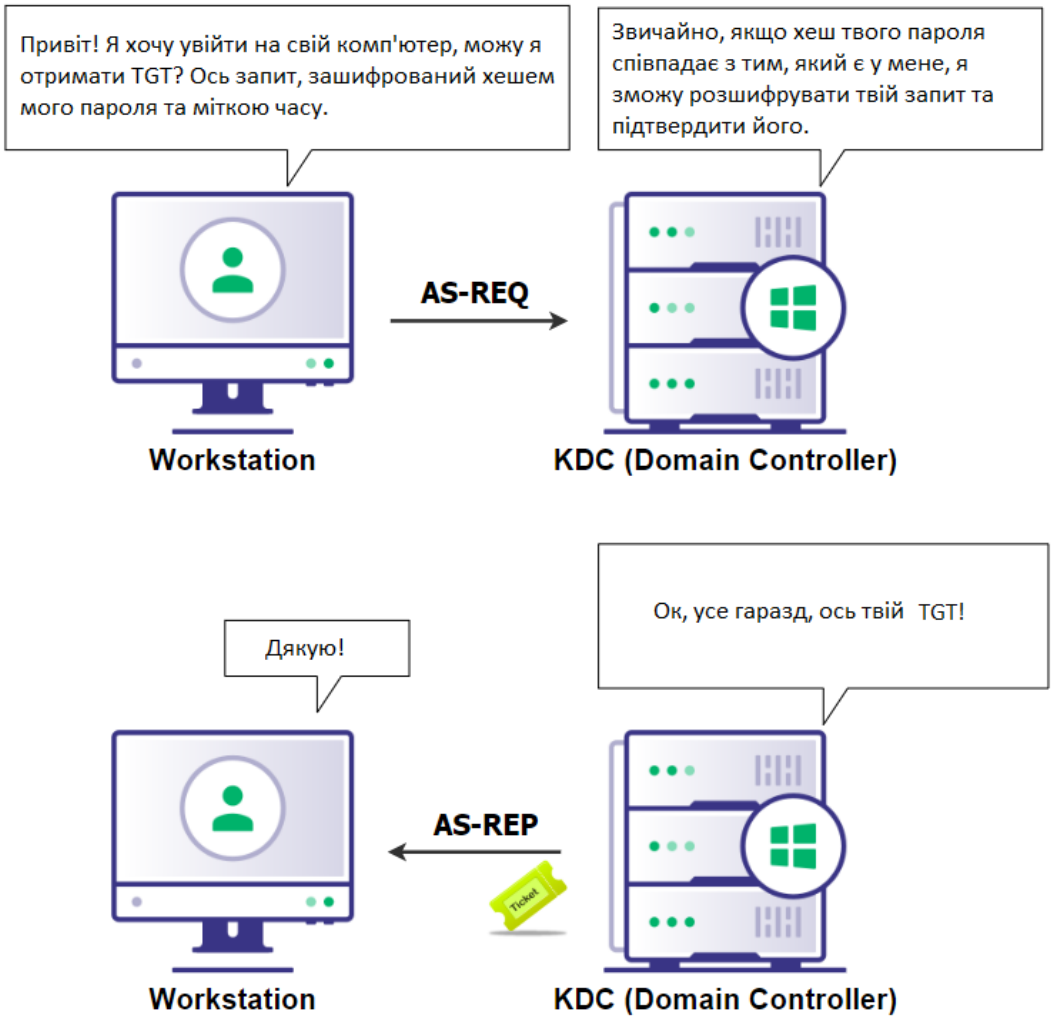


Рисунок 2.2 – Первинна автентифікація користувача

В даному випадку LSA відповідає за інші функції, пов’язані з безпекою:

– управління політикою аудиту на комп’ютері та реєстрація будь-яких подій, що генеруються контрольним монітором безпеки;

– керування локальною політикою безпеки на комп’ютері, наприклад, максимальна кількість дозволених спроб входу до системи та параметри блокування облікового запису;

– надає інтерактивні послуги автентифікації користувачів.

Після первинної автентифікації LSA надає користувачеві маркер доступу (див. рис. 2.3 та 2.4), який містить індивідуальний та груповий SID користувача та їх права, вони дозволяють користувачеві отримувати шлях до ресурсів, котрим має дозвіл.

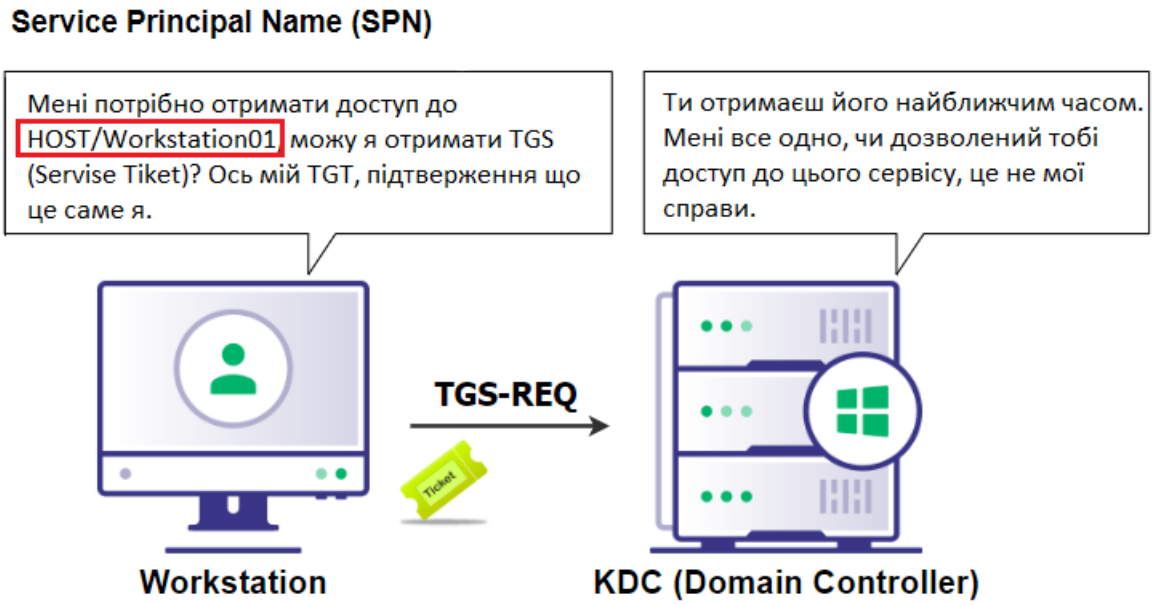


Рисунок 2.3 – Service Principal Name

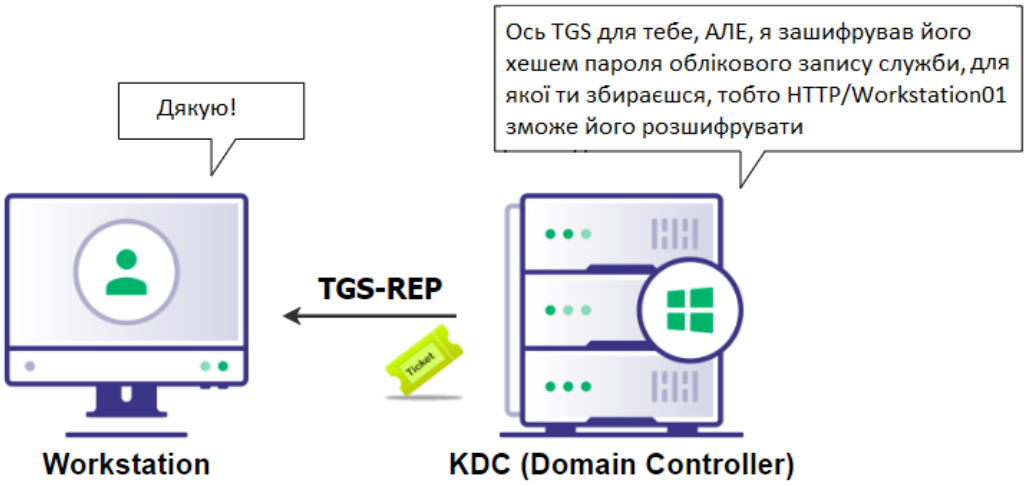


Рисунок 2.4 – Отримання TGS – квитка

3 ДОСЛІДЖЕННЯ KERBEROS АВТЕНТИФІКАЦІЇ В WINDOWS LSA-СЕРВІСАХ

3.1 Практичне застосування LSA API

Kerberos protocol – це мережевий протокол автентифікації, створений в Массачусетському технологічному інституті (MIT). Основною перевагою Kerberos protocol є те, що він реалізований так, що під час автентифікації не відбувається передача пароля у відкритому вигляді по мережі. Таким чином, виключається можливість перехоплення паролів по мережі. Представимо короткий список місць, де найбільш часто може знадобитися використання Kerberos:

– для імплементацій Single Sign-On (SSO) технологій;

– для установки криптованих з'єднань, клієнта з сервером;

– для автентифікації користувачів при login в Windows System;

– для автентифікації в Web Services.

Опишемо нижче простий варіант роботи Kerberos authentication:

– клієнт запитує, а потім отримує Ticket Granting Ticket (TGT ticket) від KDC;

– клієнт запитує, а потім отримує Ticket Granting Service's (TGS ticket) від KDC;

– клієнт відправляє Серверу, на якому він хоче зробити authentication TGS ticket;

– сервер, використовуючи ці два тікета здійснює authentication і створює new logon session, а також отримає user token handle;

– використовуючи отриманий user token handle, можна провести impersonation token, щоб отримати доступ до певних ресурсів сервера;

– після автентифікації у користувача і у сервера буде загальний session key, який можна використовувати для підписів повідомлень або для установки криптованного з’єднання;

– у Microsoft, для програмістів, яким потрібно використовувати Kerberos authentication, створена SSPI API, яка дозволяє зробити автентифікацію за допомогою Kerberos протоколу.

Завданням кваліфікаційної даної роботи є дослідження Lsa service (яке відомо підтримує Kerberos authentication [43]) та описати припущення щодо того, як реалізований Kerberos протокол в LSA service. Це зможе дати більш глибоке розуміння роботи Windows system, а також зуміє надати альтернативній спосіб (який не використовує SSPI API) реалізації Kerberos authentication в інших проєктах.

Альтернативний спосіб реалізації Kerberos authentication в Windows systems, можна здійснити використовуючи API для взаємодії с LSA service. LSA-service – це системний сервіс, за допомогою якого система здійснює створення Logon session на локальному комп’ютері. У LSA API, як і в SSPI API, велика частина Kerberos Protocol реалізована всередині API функції, а саме:

– установка з’єднання з KDC;
– переведення повідомлень KDC;
– прийом повідомлень з KDC;
– автентифікація користувача за допомогою TGT і TGS тікетів;
– та інше.

При цьому у програміста немає можливості безпосередньої взаємодії з LSA service. Програміст може взаємодіяти з LSA service тільки через API, що надається системою у вигляді бібліотеки secure32.dll.

Відомо, що відкритого коду LSA service не існує, тому на 100% не можна стверджувати як LSA service взаємодіє з KDC. Дослідивши LSA API (функцію LsaCallAuthenticationPackage та функцію LsaLogonUser), зроблено припущення, як це повинно виглядати (див. додаток А).

Таким чином, щоб підтвердити припущення пропонується реалізувати успішний виклик функції LsaLogonUser за допомогою Kerberos authentication, в LSA service API. Знаючи Kerberos Protocol та опис LSA service API, пропонується підтвердити схему роботи функції у разі, якщо запропонована програма справиться з поставленим завданням.

Для написання програми, яка використовує LSA API, та здійснює Kerberos authentication. потрібно розкрити наступні питання:

– загальний алгоритм дій при Kerberos authentication через LSA API, в Windows системах;

– алгоритм дій з LSA API, який необхідно виконати на Windows Клієнта;

– алгоритм дій з LSA API, які необхідно виконати на Windows Сервері;

– яке LSA API необхідно використовувати на Windows клієнта;

– яке LSA API використовувати на Windows сервері.

3.2 Проєкт з відкритим кодом LsaLogonUserTest

Однією з основних цілей, при написанні даної кваліфікаційної роботи, є надання можливості використовувати debug, для більш глибокого розуміння використання LSA API. Для вивчення Kerberos authentication через дебаг у читача, даної кваліфікаційної роботи, повинен бути PC, на якому стоїть Visual Studio 2015 року, так як LsaLogonUserTestпроєкт реалізований саме на VS - 2015. Якщо читач захоче продебажити реальну Kerberos authentication (НЕ unit-tests) у нього повинен бути доступ до Windows PC і до Windows Server PC. Обидві ці Windows ОС повинні знаходиться в одному домені. На Windows PC в дебазі можна вивчити отримання TGT і TGS - тікетів, а в Windows Server PC в дебазі можна вивчити як реалізувати автентифікацію.

Відзначається, що увесь код і діаграми класів, які зустрічаються в даній кваліфікаційній роботі, можна знайти в LsaLogonUserTest open source project.

Цей проєкт був реалізований саме під час дослідження LSA Service API, щоб за допомогою прикладу підтвердити припущення щодо можливої реалізації Kerberos протоколу в LSA, а також на практичному прикладі продемонструвати як реалізувати автентифікацію Kerberos використовуючи LSA API. За допомогою цього проєкту можна перевірити Kerberos authentication в реальній системі. Для цього потрібно виконати наступні дії:

– відкрити LsaLogonUserTest.sln файл в VS – 2015;

– забілдити два проєкти ServerApp та ClientApp. Якщо все вийде успішно, то з’являться два exe файли (ServerApp.exe та ClientApp.exe);

– скопіювати ServerApp.exe на Windows Server (наприклад Windows Server 2016);

– відкрити Command Line console та запустити ServerApp.exe з одним параметром -spn який буде мати вигляд: - ServerApp.exe -spn;

– скопіювати рядок, який отримується на виході. Цей рядок повинен мати наступний формат:

– host / <dnsHostName> <DnsDomainName> @ <dnsDomainName>;

– скопіювати ClientApp.exe на Windows Client (наприклад Windows 10), який знаходиться в тому ж домені, що і Windows Server;

– відкрити Command Line console та запустити ClientApp.exe з двома параметрами:

– перший параметр: -spn;

– другий параметр: рядок, отриманий з ServerApp.exe;

– виглядати це повинно наступним чином:

– "ClientApp.exe - spn host / <dnsHostName> <DnsDomainName> @ <dnsDomainName>";

– у результаті успішного запуску вищеперерахованими параметрами, поруч з ClientApp.exe з’являться два файли:

– krbtgtTicket.bin - це TGT ticket;

– serviceTicket.bin - це TGS ticket;

– скопіювавши отримані тікети (krbtgtTicket.bin і serviceTicket.bin) на Windows Server, потрібно розмістити їх поруч з ServerApp.exe;

– відкрити Command Line console та запустити ServerApp.exe з одним параметром -auth. А саме: - ServerApp.exe -auth;

– якщо тікети валідні, то отримаються наступні повідомлення "Success completed task of the authentication";

– якщо тікети не вірні (не валідні), то вспливає повідомлення, яке має наступний текст: "Failed task of the authentication".

Також в даному досліджені є unit-tests, які дозволяють в дебазі вивчити використання LSA API при Kerberos Authentication. У LsaLogonUserTest проєкті не реалізований наступний функціонал (замість цього функціоналу ми використовували Моки):

– зв’язок клієнта з сервером (відправлення повідомлень клієнту / сервера, отримання повідомлень з клієнта / сервера);

– парсинг повідомлень від клієнта / сервера;

– упаковка повідомлення в запит / відповідь.

Основною метою проєкту LsaLogonUserTest є:

– демонстрація коду в даній кваліфікаційній роботі;

– можливість вивчення Kerberos Authentication за допомогою дебага;

– надати заготовку для майбутніх проєктів пов’язаних з Kerberos Authentication.

Проєкт LsaLogonUserTest ліцензується під ліцензією MIT, тому його можна вільно використовувати в своєму комерційному або open source projects, наприклад, в якості заготовки для майбутніх проєктів.

Як зазначалося вище при описі Kerberos Protocol, на клієнті необхідно отримати два тікети TGT і TGS. Для отримання TGT ticket немає потреби в отриманні додаткової інформації, так як цей білет призначається саме клієнту. А ось для отримання TGS ticket потрібно передати Service Principal Name (SPN) на KDC, щоб він зміг зрозуміти, на який service планується здійснення

authentication і, відповідно, KDC мав би можливість зробити тікет, який зможе прочитати тільки цей service.

Таким чином, якщо не брати до уваги запити на KDC, для здійснення Kerberos Authentication, клієнт повинен послати два запити на сервер, на якому він хоче зробити authentication. Перший запит для отримання SPN і другий запит для автентифікації. Для того, щоб звернути ці два запити в один запит, прийнято було вибрати патерн Decorator. Діаграма класів клієнтської частини в LsaLogonUserTest представлена в Додатку Б.

Сервер, отримавши запит від клієнта на отримання SPN, формує його і відсилає його назад клієнтові. Формат SPN відповіді повинен відповідати формату, який використовується в Active Directory при реєстрації служби. Наприклад, в LsaLogonUserTest project, формат SPN виглядає так:

host / <dnsHostName>. <dnsDomainName> @ <dnsDomainName>.

Таким чином, алгоритм отримання SPN клієнтської частини LsaLogonUserTest project має вигляд:

– створюється об’єкт, який реалізує IServerRequest interface, за допомогою якого надалі посилається SPN запит на сервер;.

– створюється об’єкт KerbAuthRequestDecorator, в конструкторі якого ми передається об’єкт, створений в пункті вище;

– викликається метод SendRequest () у KerbAuthRequestDecorator об’єкта. У свою чергу KerbAuthRequestDecorator, викликає спочатку SendRequest () у spnRequest об’єкта, для відправки spn запиту. У тілі цього методу здійснюється відправка повідомлення на отримання SPN. Після отримання відповіді створюється об’єкт класу IServerResponse (с spn всередині);

– потім об’єкт KerbAuthRequestDecorator аналізує отриманий результат. Якщо spn відповідає spn формату, то можна рухаєтися далі, в іншому випадку викидається exception, який припиняє процес автентифікації з помилкою;

– якщо spn відповідає spn формату, то об’єкт KerbAuthRequestDecorator формує запит автентифікації.

Розглянемо код, в якому було створено об’єкт класу KerbAuthRequestDecorator.

```
TEST(KerbAuthRequestDecorator, success)
{
   int functionToFailing = 0;
   Secur32::Secur32WrapperPtr secur32Wrapper(new
MockSecur32Wrapper(functionToFailing));
   ServerRequestPtr spnRequest(new
MockServerRequest(Mock::RequestType::SPN_Request));
   SpnValidatorPtr spnValidator(new SpnValidator());
   ServerResponsePtr authResponse;
   KerbAuthRequestDecoratorPtr kerbAuthenticator;
   ASSERT_NO_THROW(kerbAuthenticator.reset(new
KerbAuthRequestDecorator(std::move(spnRequest),
                                                std::move(secur32Wrapper),
                                                std::move(spnValidator))));
   ASSERT_NO_THROW(authResponse = kerbAuthenticator->SendRequest());
   ASSERT_EQ(authResponse->GetStringDataFromResponse(L"auth"), L"Some
authentication response");
}
```

Якщо проаналізувати вище представлений код, то можна зауважити, що в конструкторі передається SPN Request.

В даному прикладі:

– ServerRequestPtr – це тип, визначений в IServerRequest.h файлі:

typedef std :: unique_ptr <IServerRequest> ServerRequestPtr;

– SpnValidatorPtr – це тип, визначений нами в ISpnValidator.h файлі (spn validator interface):

typedef std :: unique_ptr <ISpnValidator> SpnValidatorPtr;

– Secur32 :: Secur32WrapperPtr – це тип, визначений в ISecur32Wrapper.h. Основним завданням даного класу є надання доступу до системного API для взаємодії з LSA service:
typedef std :: unique_ptr <ISecur32Wrapper> Secur32WrapperPtr.

Відмічено, що об’єкти spnRequest і secur32Wrapper реалізовані в Мокі, так як в LsaLogonUserTest project не реалізована відправка і прийом повідомлень на сервер.

3.3 Отримання SPN з сервера та квитків TGT та TGS для автентифікації

На наступному етапі дослідження розглядалося, що відбувається в SendRequest методі класу AKerbAuthRequestDecorator, який є базовим класом для KerbAuthRequestDecorator :

```
ServerResponsePtr AKerbAuthRequestDecorator::SendRequest()
{
   IServerRequest* spnRequest = GetRequest();
   ServerResponsePtr spnResponse;
   TicketData serviceTicket;
   TicketData krbtgtTicket;
   const ULONG ticketFlags = GetKerberosTicketFlags();
   static const std::wstring krbtgtName = L"krbtgt";
   static const std::wstring SPN_KEY = L"SPN";
   spnResponse = spnRequest->SendRequest();
   const std::wstring servicePrincipalName = spnResponse-
>GetStringDataFromResponse(SPN_KEY);

   bool isSpnDataValid = m_spnValidator->Validate(servicePrincipalName);
   if (!isSpnDataValid)
```

```
  {
    throw KerberosException(
      KerberosException::ErrorType::INVALID_SPN_DATA,
      "AKerbAuthRequestDecorator::SendRequest: spn data is invalid"
    );
  }
}
```

Опис дій в коді:

– насамперед отримується покажчик на SPN Request;

– потім створюється smart pointer на IServerResponse (ServerResponsePtr). Цей smart pointer прийме у володіння об’єкт, після виклику spnRequest-> SendRequest ();

– далі створюються два порожніх std :: vector <unsigned char> (TicketData це typedef на цей std :: vector), об’єкти TicketData будуть заповнені пізніше;

– створюється ticketFlags і записується туди число 0x60A00000. Число 0x60A00000 є результатом виконання функції GetKerberosTicketFlags.

Вигляд функції GetKerberosTicketFlags:

```
ULONG AKerbAuthRequestDecorator::GetKerberosTicketFlags()const
{  return KerberosTicketOptions::Forwardable |
KerberosTicketOptions::Forwarded |
        KerberosTicketOptions::Renewable |
KerberosTicketOptions::Pre_authent; }
```

Для отримання більш докладної інформації про те, які бувають прапори, що вони означають і як їх виставляти більш детально описано в 4768(S, F) [51]. А створений об’єкт ticketFlags буде використовуватися в наступних пунктах дослідження при отриманні квитків.

Створюються два рядки krbtgtName і SPN_KEY:

– krbtgtName – це ім’я тікета, яке необхідно використовувати при отриманні TGT ticket;

– SPN_KEY – це ключ, за допомогою якого з відповіді можна отримати SPN.

Використовуючи покажчик на SPN Request, посилається запит на сервер. Сервер обробляє цей запит. Формує SPN і відповідає клієнтові. Клієнт, отримавши відповідь від сервера, записує отримані дані в змінну spnReponse. Потрібно звернути увагу, що ніяких запитів на сервер в даному випадку не летить, так як spnRequest – це насправді Mock.

Для отримання SPN у вигляді рядка викликається метод GetStringDataFromResponse, передавши туди SPN_KEY для того, щоб отримати з відповіді саме SPN, а не щось інше.

Після отримання SPN у вигляді рядка, відбувається валідація на предмет відповідності певним форматом. Для цього викликається метод Validate у змінної m_spnValidator, при цьому, передавши туди SPN, отриманий від сервера. Для валідації SPN використовуються регулярні вирази (std :: regex і std :: cmatch).

Відбувається перевірка результату валідації. Якщо валідація пройшла успішно, можна рухатися далі до автентифікації, в іншому випадку викидається exception, який припиняє процес authentication, так як з невалідним SPN, це не має сенсу.

Отримавши SPN рядок, тим самим отримується вся необхідна інформація для отримання TGS ticket. Для отримання TGT ticket немає потреби в додатковій інформації. Ці тікети в подальшому спакуються в автентифікаційний запит до сервера.

Для того, щоб здійснити отримання TGT ticket і TGS ticket необхідно виконати наступні кроки:

– встановити з’єднання з LSA service, отримавши, після встановлення з’єднання, LSA HANDLE;

– отримати унікальний ідентифікатор автентифікаційного пакета (використовується при запиті тікетів);

– заповнити структуру KERB_RETRIEVE_TKT_REQUEST для отримання TGT ticket;

– отримати TGT ticket та розпарсити KERB_RETRIEVE_TKT_RESPONSE, вільнити пам’ять, займану KERB_RETRIEVE_TKT_RESPONSE;

– заповнити структуру KERB_RETRIEVE_TKT_REQUEST для отримання TGS ticket;

– отримати TGS ticket та розпарсити KERB_RETRIEVE_TKT_RESPONSE, вільнити пам’ять, займану KERB_RETRIEVE_TKT_RESPONSE;

– закрити з’єднання з LSA service, закривши LSA HANDLE;

– для отримання Keberos tickets в LsaLogonUserTest в проєкт був реалізований клас KerberosTicketsManger, основним методом даного класу є метод RequestTicketFromSystem. Метод RequestTicketFromSystem – приймає два параметри:

– перший параметр – це std :: vector, в який запишеться отриманий тікет;

– другий параметр – це рядок, в якій зазначено цільове ім’я тікета.

Значення цього рядка вказує на те, який тікет збираємося отримувати. Таким чином, в даному випадку для отримання TGT ticket, як другий параметр буде рядок "krbtgt".

Для отримання TGS ticket, як другий параметр буде SPN рядок, отримана раніше. Приклад коду наведено нижче:

```
ServerResponsePtr AKerbAuthRequestDecorator::SendRequest()
{
    IServerRequest* spnRequest = GetRequest();
    ServerResponsePtr spnResponse;
    TicketData serviceTicket;
    TicketData krbtgtTicket;
    const ULONG ticketFlags = GetKerberosTicketFlags();
```

```
    static const std::wstring krbtgtName = L"krbtgt";
    static const std::wstring SPN_KEY = L"SPN";
    spnResponse = spnRequest->SendRequest();
    const std::wstring servicePrincipalName = spnResponse-
>GetStringDataFromResponse(SPN_KEY);
    bool isSpnDataValid = m_spnValidator->Validate(servicePrincipalName);
    if (!isSpnDataValid)
    {
        throw KerberosException(
            KerberosException::ErrorType::INVALID_SPN_DATA,
            "AKerbAuthRequestDecorator::SendRequest: spn data is invalid"
        );
    }
    typedef std::unique_ptr<KerberosTicketsManger> KerberosTicketsMangerPtr;
    const KerberosTicketsMangerPtr kerbTicketsManger(new
KerberosTicketsManger(m_secur32Wrapper));
    kerbTicketsManger->SetTicketFlags(ticketFlags);
    kerbTicketsManger->RequestTicketFromSystem(serviceTicket,
servicePrincipalName);
    kerbTicketsManger-
>SetCacheOptions(KERB_RETRIEVE_TICKET_AS_KERB_CRED);
    kerbTicketsManger->RequestTicketFromSystem(krbtgtTicket, krbtgtName);
    const ServerRequestPtr authRequest = PackTicketsToRequest(serviceTicket,
krbtgtTicket);
    ServerResponsePtr authResponse = authRequest->SendRequest();
    return authResponse;
}
```

Аналіз вище описаного коду:

– перше, що відбувається на цьому етапі дослідження –це створення об’єкту класа KerberosTicketsManger;

– потім викликається SetTicketFlags (оскільки це не влаштовує за замовчуванням);

– отримується TGS ticket, викликавши метод RequestTicketFromSystem, передавши туди вектор, в якому буде записано результат виклику RequestTicketFromSystem, а також рядок SPN;

– встановлюється нове значення для cacheOptions, викликавши SetCacheOptions, оскільки старе значення не влаштовує;

– отримується TGT ticket, викликавши метод RequestTicketFromSystem та передавши туди вектор, у якому буде записаний результат виклику RequestTicketFromSystem і “krbtgt” рядок.

На цьому етапі відразу постає питання: чому спочатку отримується TGS ticket, а потім вже отримується TGT ticket, адже в документації Kerberos все виглядає навпаки: спочатку отримується TGT ticket, а потім на основі TGT ticket отримуємо TGS ticket. Насправді, при отриманні TGS ticket в API функції відбувається кілька операцій:

– отримання TGT ticket та запис його в кеш;

– отримання TGS ticket (використовуючи TGT з кеша);

– отриманий TGS ticket записується в out параметр API функції.

Таким чином, при отриманні TGS ticket, вже є в кеші TGT ticket. Тому коли йде другий запит на отримання TGT ticket, він не запитується з KDC, а береться з кеша.

Але повертаючись до функції SendRequest, можна детальніше розглянути, що відбувається на кожному етапі. Коли створюється об’єкт класу KerberosTicketsManger у конструкторі цього об’єкта, відбувається встановлення з’єднання з LSA service. Виглядає це в коді наступним чином:

```
void KerberosTicketsManger::InitializeUntrustedConnect()
{
   HANDLE hLsa = NULL;
   NTSTATUS ticketsStatus = m_secur32Wrapper-
>LsaConnectUntrusted(&hLsa);
```

```
    if (STATUS_SUCCESS != ticketsStatus)
    {
        throw KerberosException(
            KerberosException::ErrorType::FAILED_LSACONNECTUNTRUSTED,
            "KerberosTicketsManger::InitializeConnection: LsaConnectUntrusted
failed"
        );
    }
    m_hLsa = LsaHandlePtr(hLsa, GetLsaHandleDeleter());
    LSA_STRING lsaStrAuthPkg = {};
    lsaStrAuthPkg.Length =
static_cast<USHORT>(strlen(MICROSOFT_KERBEROS_NAME_A));
    lsaStrAuthPkg.MaximumLength =
static_cast<USHORT>(strlen(MICROSOFT_KERBEROS_NAME_A));
    lsaStrAuthPkg.Buffer = MICROSOFT_KERBEROS_NAME_A;
    ticketsStatus = m_secur32Wrapper-
>LsaLookupAuthenticationPackage(m_hLsa.get(),
                                                &lsaStrAuthPkg,
                                                &m_authPkgId);
    if (STATUS_SUCCESS != ticketsStatus)
    {
        throw KerberosException(
            KerberosException::ErrorType::FAILED_LSALOOKUPAUTHENTICATI
ONPACKAGE,
            "KerberosTicketsManger::InitializeConnection:
LsaLookupAuthenticationPackage failed"
        );
    }
}
```

Основні дії та функції вище представленого коду:

– LsaConnectUntrusted – встановлюється з’єднання з LSA service. Якщо все пройшло, успішно отримується LSA HANDLE;

– створюється smart pointer для LSA HANDLE – це поле класу m_hLsa, у деструкторі KerberosTicketsManger при руйнуванні об’єкта m_hLsa буде викликаний метод LsaDeregisterLogonProcess;

– створюється LSA_STRING, де макрос MICROSOFT_KERBEROS_NAME_A дорівнює рядку "Kerberos";

– викликається LsaLookupAuthenticationPackage. Якщо виклик функції закінчився успішно, на виході отримується унікальний ідентифікатор автентифікаційного пакета – це поле класу m_authPkgId.

Потім, як зазначалося вище, викликаються два методи SetTicketFlags та RequestTicketFromSystem. У функції SetTicketFlags на цьому етапі змінюється тільки поле класу, а ось метод RequestTicketFromSystem реалізується інакше.

Метод RequestTicketFromSystem має вигдяд:

```
void KerberosTicketsManger::RequestTicketFromSystem(TicketData& vecTicket,
                                                    const std::wstring& tgtName)const
{
   KerbRetrieveTktRequest kerbRetrieveTktRequest(tgtName);
   ULONG responseLen = static_cast<ULONG>(-1);
   NTSTATUS protocolStatus = STATUS_ACCESS_DENIED;
   KERB_RETRIEVE_TKT_RESPONSE* pResp = NULL;
   kerbRetrieveTktRequest.SetTicketFlags(m_ticketFlags);
   kerbRetrieveTktRequest.SetCacheOptions(m_cacheOptions);
   NTSTATUS ticketsStatus = m_secur32Wrapper-
>LsaCallAuthenticationPackage(
      m_hLsa.get(),
      m_authPkgId,
      reinterpret_cast<PVOID>(kerbRetrieveTktRequest.GetRetrieveTktRequest())
      kerbRetrieveTktRequest.Length(),
      reinterpret_cast<PVOID*>(&pResp),
      &responseLen,
```

```
    &protocolStatus
  );
  typedef std::unique_ptr<KERB_RETRIEVE_TKT_RESPONSE,
LsaBufferDeleter> LsaBufferDeleterPtr;
  const LsaBufferDeleterPtr ticketPtr(pResp, GetLsaBufferDeleter());
  if (STATUS_SUCCESS != ticketsStatus)
  {
    throw KerberosException(
      KerberosException::ErrorType::FAILED_LSACALLAUTHENTICATIO
NPACKAGE,
      "KerberosTicketsManger::RequestTicketFromSystem:
LsaCallAuthenticationPackage
       failed");
  }
```

Як видно з прикладу, метод не дуже складний. Основна робота в методі відбувається при викликанні функції LsaCallAuthenticationPackage. Якщо функція була успішною, на виході отримується покажчик на KERB_RETRIEVE_TKT_RESPONSE структуру. Потім, використовуючи поле Ticket.EncodedTicket цієї структури, отримуються дані для тикету, що запитується клієнтом. Розглянемо параметри, які передаються до функції LsaCallAuthenticationPackage і звідки вони беруться.

Першим параметром є LSA HANDLE - m_hLsa.get(). Цей параметр ми отримуємо під час виклику LsaConnectUntrusted.

Другий параметр – це унікальний ідентифікатор автентифікаційного пакета – m_authPkgId. Цей параметр отримується під час виклику LsaLookupAuthenticationPackage API функції.

Третім параметром йде покажчик на структуру KERB_RETRIEVE_TKT_REQUEST. Як заповнюється ця структура показано нижче, єдине, що потрібно відзначити, такі значення як ім’я Kerberos ticket (krbtgt string or spn string), ticketFlags, cachedOptions беруть участь у заповненнях структури. Загалом те, які дані будуть у Ticket.EncodedTicket

(TGT ticket data або TGS ticket data), залежить від того, якими даними заповнюється KERB_RETRIEVE_TKT_REQUEST в структуру.

Четвертий параметр – це розмір KERB_RETRIEVE_TKT_REQUEST структури.

П’ятий параметр – це покажчик на покажчик KERB_RETRIEVE_TKT_RESPONSE структури. При успішному виконанні функції LsaCallAuthenticationPackage виділиться пам’ять під KERB_RETRIEVE_TKT_RESPONSE структуру і заповнить її відповідними даними. На цьому етапі потрібно подбати про звільнення пам’яті, яку займає ця структура. Для цього викликається функція LsaFreeReturnBuffer API. Виклик методу LsaFreeReturnBuffer досягається за рахунок створення smart pointer на покажчик KERB_RETRIEVE_TKT_RESPONSE структури.

Шостий параметр – це покажчик на unsigned long, в який буде записаний розмір отриманої KERB_RETRIEVE_TKT_RESPONSE структури - responseLen.

Сьомий параметр – це NTSTATUS, який вказує стан завершення пакета автентифікації.

Щоб до кінця розкрити тему отримання TGT та TGS тікетів, залишається продемонструвати, як відбувається створення та заповнення структури KERB_RETRIEVE_TKT_REQUEST. Для того, щоб створити структуру KERB_RETRIEVE_TKT_REQUEST і заповнити її відповідним чином був реалізований KerbRetrieveTktRequest клас. KerbRetrieveTktRequest клас – є обгорткою на структуру KERB_RETRIEVE_TKT_REQUEST. Основним методом даного класу є метод GetRetrieveTktRequest, який повертає нам покажчик на структуру KERB_RETRIEVE_TKT_REQUEST.

Код функції GetRetrieveTktRequest:

```
KERB_RETRIEVE_TKT_REQUEST*
KerbRetrieveTktRequest::GetRetrieveTktRequest()
{
   KERB_RETRIEVE_TKT_REQUEST* ret = NULL;
```

```
    ret =
reinterpret_cast<KERB_RETRIEVE_TKT_REQUEST*>(&m_retrieveTktRequest
Data[0]);
    LUID luidLogonId;
    memset(&luidLogonId, 0, sizeof(LUID));
    SecHandle hSec;
    memset(&hSec, 0, sizeof(SecHandle));
    ret->MessageType =
KERB_PROTOCOL_MESSAGE_TYPE::KerbRetrieveEncodedTicketMessage;
    ret->LogonId = luidLogonId;
    ret->TargetName.Length = static_cast<USHORT>(m_targetName.length() *
sizeof(wchar_t));
    ret->TargetName.MaximumLength =
static_cast<USHORT>(m_targetName.length() * sizeof(wchar_t));
    ret->TargetName.Buffer = (wchar_t*)(ret + 1);
    memcpy(ret->TargetName.Buffer, m_targetName.c_str(), ret-
>TargetName.Length);
    ret->TicketFlags = m_ticketFlags;
    ret->CacheOptions = m_cacheOptions;
    ret->EncryptionType = 0;
    ret->CredentialsHandle = hSec;
    return ret;   }
```

3.4 Розробка класів для проведення автентифікації ServerLib

У серверній частині планувалось виконати дві операції:

– під час отримання запиту на SPN, потрібно створити його та відправити клієнту;

– при отриманні запиту автентифікації, витягнути із запиту TGT ticket та TGS ticket. Провести автентифікацію. Надіслати клієнту результат автентифікації.

Для організації обробки запитів було обрано паттерн Strategy в LsaLogonUserTest. Діаграма класів Серверної частини у LsaLogonUserTest project показана в додатку В.

Варто відзначити , що у ASpnResponseStrategy при отриманні рядка наступного формату «host/<DnsHostname>.<DnsDomain>@<DnsDomain>» використовується лише одна system API функція – це GetComputerNameExW (щоб отримати і <DnsHostname> DnsDomain>).

Розглянемо докладніше, як працює KerbAuthStrategy.

Після того, як сервер виявляє, що це запит автентифікації він, відповідно, витягує із запиту два тікети (TGT і TGS) і створює об’єкт класу, успадкований від AkerbAuthStrategy. Самої логіки, яка розпізнає тип запиту та створення відповідної стратегію дій побачити неможливо. Але дізнатися, як це приблизно має виглядати можна проаналізувавши unit-test:

```
TEST(TestKerbAuthStrategy, success)
{
   TicketData serviceTicket = CreateKerbTicketFromString("Some server ticket");
   TicketData krbtgtTicket = CreateKerbTicketFromString("Some krbtgt ticket");
   ClientResponsePtr response;
   KerbAuthStrategyPtr kerbAuthStrategy(new KerbAuthStrategy(serviceTicket,
krbtgtTicket, 0));
   ASSERT_NO_THROW(response = kerbAuthStrategy->CreateResponse());
   const std::string& authenticationResult = kerbAuthStrategy-
>GetAuthenticationResult();
   ASSERT_EQ(authenticationResult, "success authentication");
}
```

На даному етапі створюється об’єкт класу KerbAuthStrategy, а потім у нього викликається метод CreateResponse. Сам метод CreateResponse теж досить простий і виглядає наступним чином:

```
ClientResponsePtr AKerbAuthStrategy::CreateResponse()
{
   ClientResponsePtr response;
   try
   {
      KerbAuthenticatorPtr kerbAuthenticator = GetKerbAuthenticator();
      m_hToken = TokenHandlePtr(kerbAuthenticator-
>Authenticate(GetServiceTicket(),
                                          GetKrbtgtTicket()), &::CloseHandle);
      response = PackSpnToResponse("success authentication");
   }
   catch (const KerbException::KerberosException& /*ex*/)
   {
      //TO DO: write to the log file here.
      response = PackSpnToResponse("failed authentication");
      m_hToken.reset(nullptr);
   }
   return response;
 }
```

Тут створюється об’єкт класу KerbAuthenticator. Як і клієнта, в конструкторі цього класу відбувається установка з’єднання з LSA service. Логіка встановлення з’єднання така сама, як і в клієнті. Після встановлення з’єднання з LSA service у поля класу KerbAuthenticator також записуються LSA HANDLE (m_hLsa) та унікальний ідентифікатор автентифікаційного пакета (m_authPkgId).

Потім в об’єкті (KerbAuthenticator) викликається метод Authenticate. В якості параметрів метод Authenticate приймає TGT ticket та TGS ticket.

У методі Authenticate відбувається вся основна робота:

```
HANDLE AKerbAuthenticator::Authenticate(const TicketData& serviceTicket,
const TicketData& krbtgtTicket)
{
   const LsaLogonUserDataManagerPtr lsaLogonUserDataManager(new
LsaLogonUserDataManager(
                                                GetSystemApiWrapper(),
                                                GetOriginName(),
                                                GetSourceModuleIdentifier()));
   ULONG kerbTicketLogonLen = 0;
   KERB_TICKET_LOGON * pKerbTicketLogon = lsaLogonUserDataManager-
>GetKerbTicketLogon(serviceTicket,
                                                    krbtgtTicket,
                                                    kerbTicketLogonLen);
   LSA_STRING lsastrOriginName = {};
   lsaLogonUserDataManager->InitLsaOrigin(lsastrOriginName);
   TOKEN_SOURCE sourceContext = {};
   lsaLogonUserDataManager->InitTokenSource(sourceContext);
   PLUID logonId = lsaLogonUserDataManager->GetDefaultLogonId();
   PQUOTA_LIMITS quotaLimits = lsaLogonUserDataManager-
>GetDefaultQuotaLimits();
   PKERB_TICKET_PROFILE profileBuffer = NULL;
   HANDLE hToken = nullptr;
   ULONG profileBufferLen = -1;
   NTSTATUS subStatus = -1;
   NTSTATUS status = m_secur32Wrapper->LsaLogonUser(m_hLsa.get(),
                                        &lsastrOriginName,
                                        GetLogonType(),
                                        m_authPkgId,
                                        pKerbTicketLogon,
```

```
                                    kerbTicketLogonLen,
                                    NULL, // additional local groups
                                    &sourceContext,
                                    reinterpret_cast<PVOID*>(&profileBuffer),
                                    &profileBufferLen,
                                    logonId,
                                    &hToken,
                                    quotaLimits,
                                    &subStatus);
   typedef std::unique_ptr<KERB_TICKET_PROFILE, LsaBufferDeleter>
LsaProfileBufferPtr;
   const LsaProfileBufferPtr profileBufferPtr(profileBuffer,
GetLsaBufferDeleter());
   if (status != STATUS_SUCCESS)
   {
      throw KerberosException(
         KerberosException::ErrorType::FAILED_LSALOGONUSER,
         "AKerbAuthenticator::Authenticate: LsaLogonUser failed");
   }
   return hToken;
}
```

Якщо проаналізувати цей код, то можна побачити, що основна робота відбувається під час виклику системної функції LsaLogonUser. Тобто LsaLogonUser приймає багато параметрів. Розглянемо, що кожен із них означає і звідки береться.

Першим параметром є LSA HANDLE - m_hLsa.get(). Цей параметр отримується під час виклику LsaConnectUntrusted

Другий параметр – це рядок – lsastrOriginName. Цей параметр ідентифікує того, хто намагається залогінитись. У кожного origin name може бути свій. Наприклад, у LsaLogonUserTest проєкті origin name = "exmln".

Далі йде значення SECURITY_LOGON_TYPE enum, яке визначає тип logon request. У проєкті LsaLogonUserTest – це значення дорівнює SECURITY_LOGON_TYPE::Network.

Потім йде параметр m_authPkgId. Цей параметр передає унікальний ідентифікатор автентифікаційного пакета. Цей параметр отримується під час виклику LsaLookupAuthenticationPackage.

П’ятим параметром йде покажчик на структуру KERB_TICKET_LOGON - pKerbTicketLogon. Для створення та наповнення цієї структури був реалізований клас LsaLogonUserDataManager. Для отримання покажчика на KERB_TICKET_LOGON структуру необхідно викликати GetKerbTicketLogon у класу LsaLogonUserDataManager. Ось як виглядає код GetKerbTicketLogon функції:

```
KERB_TICKET_LOGON* LsaLogonUserDataManager::GetKerbTicketLogon(
                                        const TicketData& serviceTicket,
                                        const TicketData& krbtgtTicket,
                                        ULONG& length)
{   length = sizeof(KERB_TICKET_LOGON) +
                static_cast<ULONG>(serviceTicket.size()) +
                static_cast<ULONG>(krbtgtTicket.size());
  m_kerbTicketLogonData.resize(length, 0);
  KERB_TICKET_LOGON* pKerbTicketLogon = NULL;
       pKerbTicketLogon =
      reinterpret_cast<KERB_TICKET_LOGON*>(&m_kerbTicketLogonData[0
      ]);
  pKerbTicketLogon->MessageType = KerbTicketLogon;
  pKerbTicketLogon->Flags = m_kerbTicketLogonFlag;
  pKerbTicketLogon->ServiceTicketLength =
      static_cast<ULONG>(serviceTicket.size());
  pKerbTicketLogon->TicketGrantingTicketLength =
      static_cast<ULONG>(krbtgtTicket.size());
```

```
   pKerbTicketLogon->ServiceTicket =
        reinterpret_cast<PUCHAR>(pKerbTicketLogon + 1);
   memcpy(pKerbTicketLogon->ServiceTicket, &serviceTicket[0],
        serviceTicket.size());
   pKerbTicketLogon->TicketGrantingTicket = pKerbTicketLogon->ServiceTicket
        + serviceTicket.size();
   memcpy(pKerbTicketLogon->TicketGrantingTicket, &krbtgtTicket[0],
        krbtgtTicket.size());
   return pKerbTicketLogon; }
```

З коду, TGT і TGS тікети копіюються у структуру, а в поля ServiceTicket і TicketGarantingTicket записуються відповідні покажчики.

Потім йде розмір KERB_TICKET_LOGON структури – kerbTicketLogonLen.

Сьомим параметром йде покажчик на TOKEN_GROUPS. У цьому параметрі необхідно надсилати додаткові ідентифікатори груп. У проєкті LsaLogonUserTest цей параметр дорівнює NULL.

Після цього йде покажчик структуру TOKEN_SOURCE – sourceContext. Цей параметр ідентифікує вихідний модуль. Для її створення необхідний рядок, що ідентифікує вихідний модуль. Цей рядок, як і originName, у кожного може бути свій. Наприклад, у LsaLogonUserTest проєкті m_srcModuleIdentifies = "exm". Приклад коду, який наповнює TOKEN_SOURCE у проєкті LsaLogonUserTest:

```
void LsaLogonUserDataManager::InitTokenSource(TOKEN_SOURCE&
        srcContext)const
{
   memset(&srcContext, 0, sizeof(TOKEN_SOURCE));
   const char* tmp = m_srcModuleIdentifies.c_str();
   strncpy_s(srcContext.SourceName,
                TOKEN_SOURCE_LENGTH,
```

```
                m_srcModuleIdentifies.c_str(),
                TOKEN_SOURCE_LENGTH - 1);
  if (!m_systemWrapper-
      >AllocateLocallyUniqueId(&srcContext.SourceIdentifier))
  {
     DWORD error = ::GetLastError();
     // TO DO: Write to logs the error code;
     throw KerbServerException(
                    KerbServerException::FAILED_INIT_TOKENSOURCE,
                    "LsaLogonUserDataManager::LsaLogonUserDataManager:
       Failed on allocation the
                     local unique Id");
  }
      }
```

Відзначається, що розмір рядка m_srcModuleIdentifies не повинен бути більшим за 8 байт. Так як AllocateLocallyUniqueId – це система API.

Дев’ятий параметр – це покажчик на покажчик структури KERB_TICKET_PROFILE - profileBuffer. Це out параметр, який є результатом роботи функції LsaLogonUser. При виході з функції Authenticate необхідно звільнити пам’ять займаної цією структурою. Досягається це створенням profileBufferPtr smart pointer. У деструкторі profileBufferPtr буде викликаний LsaFreeReturnBuffer system API.

Потім передається покажчик на unsigned long, в який буде записано розмір KERB_TICKET_PROFILE структури – profileBufferLen.

Одинадцятим параметром йде покажчик на LUID структуру – logonId. Цей параметр є параметром out і в цю структуру записується унікальний ідентифікатор Logon session.

Потім знову йде out параметр, який є покажчиком на HANDLE - hToken. У цей hToken буде записуватись новий user token, створений для цієї сесії. Цей параметр повертається з Authenticate. Також звертається увага, що після того,

як буде використаний HANDLE, він більше не потрібен, для усунення витоків необхідно викликати функцію CloseHandle API. У проєкті LsaLogonUserTest це досягається створенням smart pointer на HANDLE, в деструкторі якого буде викликатися CloseHandle API функція.

Тринадцятий параметр також out параметр – quotaLimits. Цей параметр буде заповнено, коли hToken дорівнюватиме основному token. В цьому випадку в цьому параметрі будуть перебувати process quota limits.

Останнім параметром йде NTSTATUS, і цей параметр є додатковим результатом виконання LsaLogonUser – subStatus. Це значення встановлюється лише в тому випадку, якщо дані облікового запису користувача є дійсними, але вхід до системи відхиляється. При цьому цей параметр міститиме додаткову інформацію, що вказує на причину відмови. Цей параметр може приймати одне з таких значень:

- STATUS_INVALID_LOGON_HOURS;
- STATUS_INVALID_WORKSTATION;
- STATUS_PASSWORD_EXPIRED;
- STATUS_ACCOUNT_DISABLED.

ВИСНОВКИ

В даній кваліфікаційній роботі було здійснено дослідження Kerberos автентифікації в Windows LSA-сервісах. Досягнута мета роботи, а саме: досліджена можливість використання LSA Service при логіні користувача за допомогою Kerberos протоколу. Завдяки чому були отримані більш глибокі знання в роботі системи Windows, а також описано припущення щодо того, як реалізований Kerberos authentication в LSA Service при логіні користувача в систему. Таким чином, дана кваліфікаційна робота розрахована на людей, які знайомі з Kerberos Protocol та їм цікава реалізація Kerberos Authentication в Windows системах, альтернативним способом не використовуючи SSPI.

Представлене дослідження буде корисно тим, хто бажає на практичному прикладі побачити як використовувати LSA service API. Даний спосіб, Kerberos authentication, можливо використовувати, наприклад в Credential Provider фічах або для входу в систему без введення пароля.

Отже, зручним та простішим способом для здійснення kerberos authentication, найкраще залишається використовувати SSPI API. Але, якщо є потреба в написанні credential provider, в якому необхідно здійснити logon в систему за допомогою kerberos tickets, то дане дослідження буде дуже корисним.

Представлене дослідження має відкритий доступ та вбачає посилання на open source проєкт, який може бути використаний як заготівля для інших майбутніх проєктів.

Результат роботи продемонстрований за посиланням на LsaLogonUserTest project: https://gitlab.com/InnaDanchuk/lsalogonusertest

UML архітектура LsaLogonUserTest проєкту: https://drive.google.com/file/d/1UmgelOBKzQ04RAa24pBVnUp01jpO7hfK/view?usp=sharing

ПЕРЕЛІК ПОСИЛАНЬ

1. Асимметричное шифрование. URL : https://encyclopedia. kaspersky.ru/ glossary/asymmetric-encryption/ (дата звернення 15.11.21).

2. Босуэлл Д., Фаучер Т. Читаемый код или программирование как искусство. Питер. 2012. 208 с.

3. Буза М. Архитектура Компьютеров. Москва. 2015. 253 с.

4. Яенко В. В. Введение в Криптографию. Москва. 2012. 348 с.

5. Вельшенбаха М . Криптография на С и С++ Москва. 2004. 443 с.

6. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Д. Паттерны объектно-ориентированного проектирования. Санкт-Петербург. 2001.368 с.

7. Гамма Е., Хелм Р., Джлнсон Р., Влиссидес Д. Приёмы объектно-ориентированного проектирования. Санкт-Перербург. 2021. 367 с.

8. Гради Буч. Введение в UML от создателей языка. Питер. 2006.

9. Идентификация (информационные системы). URL : https://ru.wikipedia.org/wiki/%D0%98%D0%B4%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B) (дата звернення 15.11.21).

10. Коробко И.В. Справочник системного администратора по Windows, 2009 .

11. Кулямин В. В. Методы верификации программного обеспечения. *Работа, прошедшая по конкурсу обзорно-аналитических статей по направлению "Информационно-телекоммуникационные системы"*, 2008.

12. Леоненков А. Самоучитель UML 2, Питер. 2007.

13. Лафоре Р. Объектно-ориентированное программирование в С++. Питер. 2004.

14. Прата С. Язык программирования С++. Лекции и упражнения. 6-е изд.Издательство Диалектика. 2021. 928 с.

15. Побегайло А. Системное программирование в Windows, Москва. 2006.

16. Полежаев В., Андрианов А., Хританков А. Проектирование на UML. Сборник задач. Издательские решения. 2017. 270 с.

17. Процедура аутентификации Windows. URL : https://www.osp.ru /winitpro/2005/05/177754 (дата звернення 15.11.21).

18. Рамбо Дж. UML 2.0 Обьектно-ориентированное моделирование и разработка. 2-е издание.Питер. 2007. 544 с.

19. Ростовцев А .О времени жизни персонального и открытого ключа. 2021. 281 с.

20. Разбираем атаки на Kerberos с помощью Rubeus. Часть 1. URL : https://habr.com/ru/company/tomhunter/blog/507140/ (дата звернення 15.11.21).

21. Саттер А. А Современное проектирование на С++. Андрей Александреску. Москва. Издательский дом «Вильямс». 2006. 336 с.

22. Саттер А. А. Стандарты программирования на С++. Серия "C++ In-Depth". Москва. Издательский дом «Вильямс».2006. 282 с.

23. Седжвик Р.Средний уровень владения С++.Алгоритмы на С++.

24. Сервер аутентификации Kerberos. Информационная безопасность. URL : https://www.jetinfo.ru/server-autentifikaczii-kerberos/(дата звернення 15.11.21).

25. Сервер аутентифікації Kerberos. URL : http://um.co.ua/2/2-7/2-72620.html (дата звернення 20.11.21).

26. Сервер проверки подлинности локальной системы безопасности. URL : https://ru.wikipedia.org (дата звернення 10.11.21).

27. Сравнение промышленных СОВ: ISIM vs. KICS/ URL : https://habr.com/ru/company/jetinfosystems/blog/450956/ (дата звернення 10.11.21).

28. Страуструп Б. Программирование: принципы и практика с использованием C++. 2-е издание. 2021. 1328 с.

29. Фримен Э., Сьера К., Бейст Б. Паттерны проектирования. 2021.640 с.

30. Хориков В. Принципы юнит-тестирования. Питер. 2022. 320 с.

31. Чиртик А. Программирование на C++. Трбки и эффекты, 2010. 350 с.

32. Шилт Г. C++ для начинающих. Шаг за шагом. Москва. 2013.620 с.

33. Шнайер Б. Прикладная криптография 2012. 610 с.

34. Яроменко І. М., Орлик О. В. Організація комп'ютерної безпеки та захисту інформації. *Одеський національний економічний університет. Інформаційні технології в економіці та управлінні*. Одеса. 2021. С.122-129.

35. Authentication. URL : https://en.wikipedia.org/wiki/Authentication (дата звернення 15.11.21).

36. B. W. Boehm. Software Engineering; R&D Trends and Defense Needs. In R. Wegner, ed. Research. Directions in Software Technology. Cambridge, MA:MIT Press, 1979.

37. Freeman S. Growing Object-Oriented Software, Guided by Tests 1st Edition. 2007.

38. IEEE 610.12-1990 Standard Glossary of Software Engineering Terminology, Corrected Edition. IEEE, February 1991.

39. Internet Security Glossary, Version 2. URL : https://datatracker.ietf.org/doc/html/rfc4949 (дата звернення 15.11.21).

40. ISO/IEC 12207 Systems and software engineering - Software life cycle processes. Geneva, Switzerland: ISO, 2008.

41. Kent Beck . Test Driven Development: By Example 1st Edition. URL : https://www.amazon.com/dp/0321146530 (дата звернення 15.11.21).

42. Koskela L. Test Driven: TDD and Acceptance TDD for Java Developers. Manning. 2007

43. Local Security Authority (LSA). URL : https://networkencyclopedia.com/local-security-authority-lsa/ (дата звернення 15.11.21).

44. Local Security Authority. URL : https://networkencyclopedia.com/local-security-authority-lsa/ (дата звернення 15.11.21).

45. Logon and Authentication Technologies. URL : https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc780455(v=ws.10)?redirectedfrom=MSDN (дата звернення 15.11.21).

46. LsaLogonUser function (ntsecapi.h). URL : https://docs.microsoft.com/en-us/windows/win32/api/ntsecapi/nf-ntsecapi-lsalogonuser (дата звернення 15.11.21).

47. Meszaros G. xUnit Test Patterns. 2007.

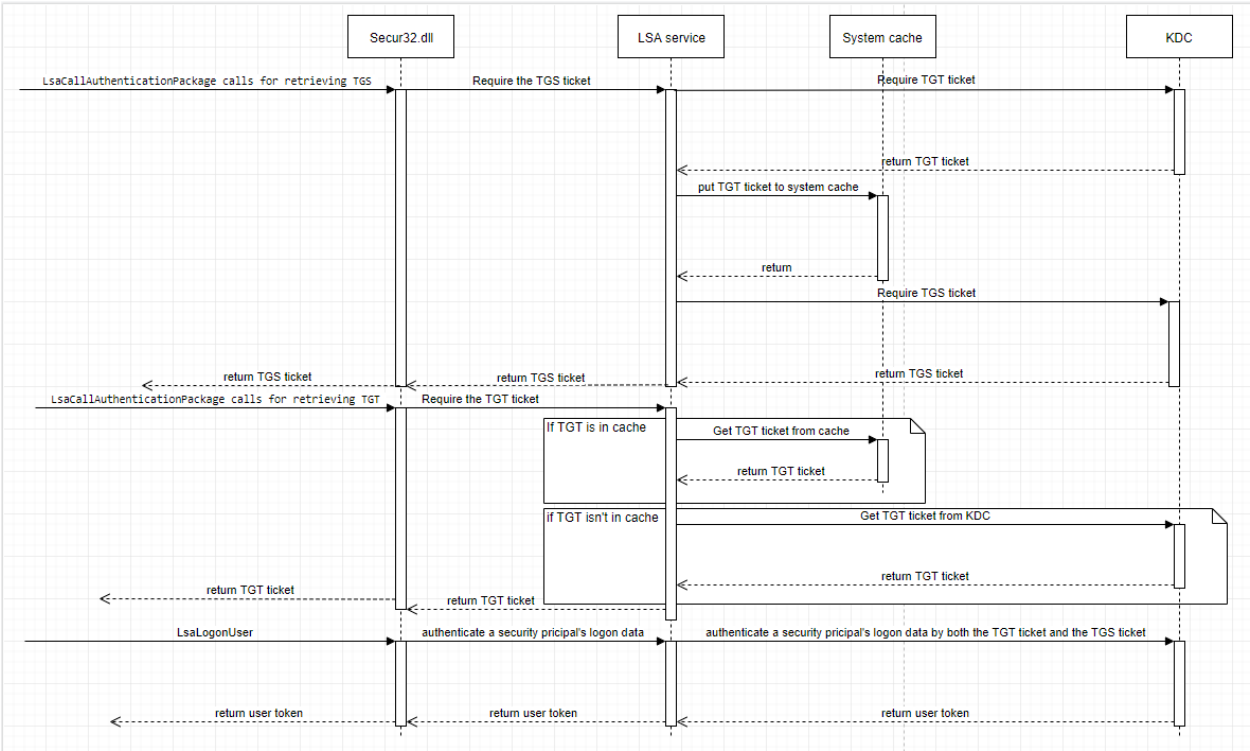
48. Roy Osherove . The Art of Unit Testing2021.

49. Winlogon and Credential Providers. URL : https://docs.microsoft.com/en-us/windows/win32/secauthn/winlogon-and-credential-providers?redirectedfrom=MSDN (дата звернення 15.11.21).

50. 4768(S, F): A Kerberos authentication ticket (TGT) was requested. URL : https://docs.microsoft.com/en-us/windows/security/threat-protection/auditing/event-4768(дата звернення 15.11.21).

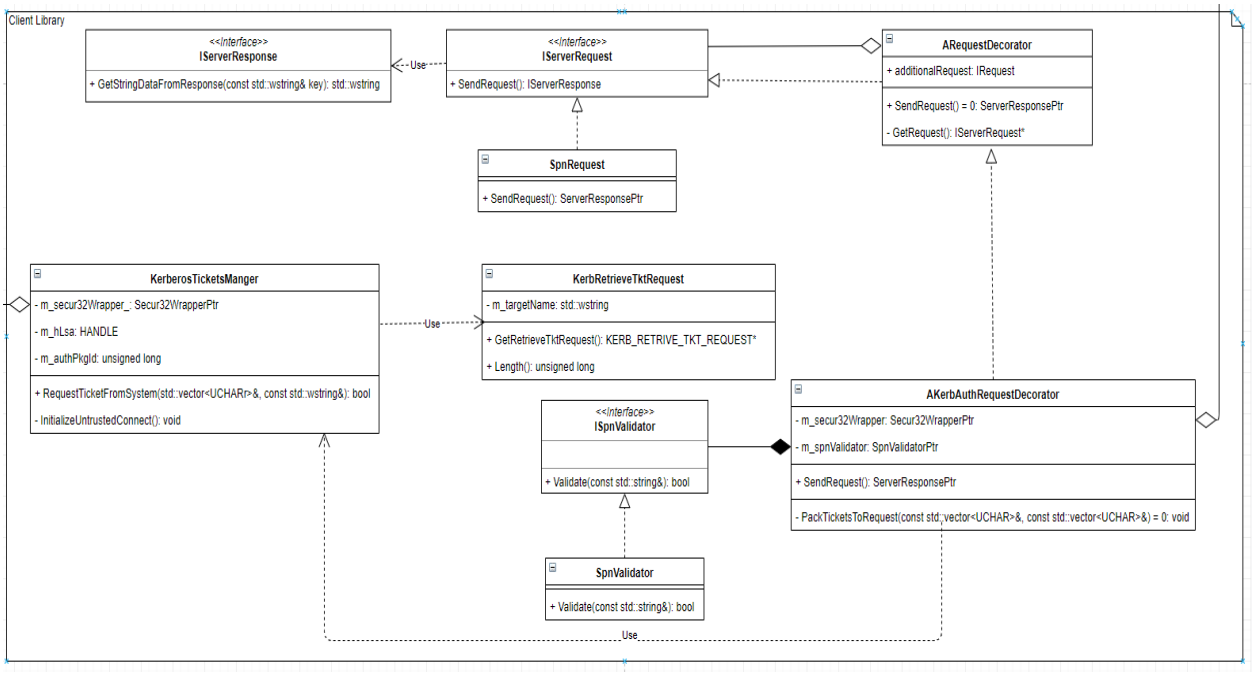
ДОДАТОК А

Діаграма використання LSA API при автентифікації



ДОДАТОК Б

Діаграма класів клієнтської частини в LsaLogonUserTest project



ДОДАТОК В

Діаграма класів серверної частини у LsaLogonUserTest