1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский политехнический университет Петра Великого
3. —
4. Институт прикладной математики и механики
5. **Кафедра «Информационная безопасность компьютерных систем»**

**ОТЧЕТ**

**по лабораторной работе № 1**

1. **«Исследование механизмов хранения конфиденциальной информации в прикладном ПО»**
2. по дисциплине «Теория обнаружения вторжений»
3. Выполнил
4. студент гр. 13646/1 Т.Н. Камалетдинов
6. Проверил
7. доцент, к.т.н. А.И. Печенкин
8. Санкт-Петербург
9. 2018

# Формулировка задания

Разработать программу, собирающую требуемую по варианту информацию и сохраняющую ее в файл или в ветку реестра.

Вариант 2. Пароли, сохраненные в веб-браузере Edge. Сохранение в файл.

# Теоретические сведения

В операционной системе Windows для хранения различной персональной информации пользователей и системы, например, паролей, используется Windows Vault. Данный механизм использует для хранения необходимой информации директории Credentials и Vault, путь к которым следующий: "<*имя\_пользователя*>\AppData\(Local | Roaming)\Microsoft\".

Папка Credentials, содержит файлы без расширений, в качестве имен которых выступают GUID. Каждый из файл содержит внутри себя «секретную» информацию.

Внутри папки Vault хранятся папки, в качестве имен которых выступают GUID. Внутри каждой такой папки хранятся файлы трех типов:

* Policy.vpol – содержит ключи AES, необходимые для расшифрования vcrd файлов. Данные ключи зашифрованы с помощью DPAPI.
* <GUID>.vsch – содержит схему: описатели структур, флаги и другую служебную информацию, необходимую для корректного отображения расшифрованной информации
* <GUID>.vcrd - запись Vault, в которой находятся исходные зашифрованные данные, относящиеся к определенной схеме и состоящие, как правило, из нескольких полей. Для шифрования исходных данных используется алгоритм AES, где ключ шифрования берется из файла Policy.vpol, в каждой записи при шифровании используется соль.

**2.2 DPAPI**

***2.2.1 Основные сведения***

Data Protection API (DPAPI) — криптографический [интерфейс программирования приложений](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9) в ОС семейства Windows. Основная цель — защита хранимых паролей и конфиденциальной информации на компьютере. Данный API содержит две доступные для вызова функции, позволяющие выполнять шифрование и расшифрование данных на системном уровне (то есть дополнительные библиотеки для этого не требуются) – достигается это за счет того, что DPAPI является частью CryptoAPI и входит в состав пакета MS Windows по умолчанию, а также взаимодействует с LSA.

CryptoAPI — интерфейс программирования приложений, который обеспечивает разработчиков Windows-приложений стандартным набором функций для работы с криптопровайдерами.

LSA или LSASS – Local Security Authority Subsystem Service – Сервис проверки подлинности локальной системы безопасности — сервис ОС Windows, отвечающий за авторизацию локальных пользователей отдельного компьютера.

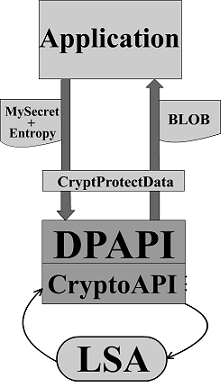


Рисунок 1 – Схема работы DPAPI

На рисунке выше можно увидеть схему работы DPAPI. Приложение обращается к ОС с помощью функции CryptProtectData(), задавая параметры Mysecret (открытый текст) и Entropy (энтропия, данный параметр не обязательный). При этом происходит следующее: CryptProtectData() делает локальный RPC-запрос к LSA, который использует внутренние функции DPAPI для выполнения операции зашифрования данных в собственном контексте. У самого LSA также есть функции системного уровня, позволяющие ему использовать DPAPI. Эти функции доступны только потокам, выполняющимся в контексте LSA – сделано это для того, чтобы данные, зашифрованные с помощью LSA-потоков, не могли быть расшифрованы потоками, использующими DPAPI, но при этом выполняющимися вне контекста LSA (например, пользовательское приложение не может выполнить расшифрование данных с использованием DPAPI, но в обход LSA). На выходе получаем BLOB – Binary Large OBject – своего рода контейнер, в котором содержатся в открытом виде данные, необходимые для расшифрования, а также сам шифртекст.

Для расшифрования используется функция CryptUnprotectData(). При расшифровании данная схема выполняется в обратную сторону.

***2.2.2 Мастер-ключ***

Мастер-ключ пользователя – это двоичный файл (BLOB), содержащий зашифрованные данные, которые используются для создания первичного ключа шифрования во всех объектах DPAPI. Поскольку мастер-ключ шифрует конфиденциальные данные пользователя, сам он нуждается в серьезной защите. В качестве исходного материала для защиты мастер-ключа используется пароль пользователя ОС Windows, а точнее кэш пароля. Если пароль не установлен, берется null.

Процесс создания и шифрования мастер-ключа:

* Сначала вызывается API функция RtlGenRandom, возвращающая 64 псевдо-случайно сгенерированных байт. Это и есть исходный материал мастер-ключа, с помощью которого будут шифроваться объекты DPAPI.
* Исходный материал мастер-ключа необходимо защитить. Для этого происходит шифрование с использованием пароля на вход пользователя, описателя безопасности (SID) и еще 16 байт случайных данных (т.н. «соль»). Процесс шифрования мастер-ключа состоит из двух этапов: сначала с помощью функции PBKDF2 (криптографического стандарта, описанного в PKCS #5), из пароля, SID и соли получается ключ. Затем этот ключ используется для получения симметричного ключа шифрования мастер-ключа. Разные версии Windows, защищают мастер-ключ по-разному. Если в Windows XP и Windows 2003 по умолчанию применяется алгоритм – 3DES, то начиная с Windows Vista – AES256.
* Мастер-ключу присваивается уникальное имя – GUID. Каждый объект DPAPI хранит этот уникальный идентификатор. GUID мастер-ключа является его «привязкой» к объекту DPAPI.

Сам Мастер-ключ не используется для защиты данных. Вместо этого используется симметричный сессионный ключ, сформированный из:

* Мастер-ключа
* Случайного набора данных
* (Опционально) дополнительной энтропии

Именно этим ключом неявно пользуются приложения при вызове функций CryptProtectData() и CryptUnprotectData() DPAPI. Сессионный ключ никогда не хранится – вместо этого, DPAPI записывает информацию о случайном наборе данных, использованном в ходе шифрования, в Data Blob. Когда приложение предоставляет DPAPI сформированный им ранее Data Blob, DPAPI использует сохраненные случайные данные для восстановления ключа и извлечения зашифрованного открытого текста.

Мастер-ключ имеет срок хранения (установленное значение составляет 3 месяца), по истечении которого генерируется новый Мастер-ключ.

***2.2.3 Credential History***

Предположим произошла смена пароля пользователя. С точки зрения алгоритма – мастер-ключ стал недосягаем. В связи с этим системе придется перешифровать мастер-ключ. Данный процесс трудоемок, мастер ключей может быть много, и перешифровывать их надо все.

В связи с этим была появилась Credential History – история паролей пользователя. Все предыдущие пароли хранятся в виде хэшей в специальном файле-контейнере, который называется CREDHIST и находится в каталоге %APPDATA%/Microsoft/Protect.

Слоты с хэшами пользователя хранятся последовательно, друг за другом. Каждый хэш пароля зашифрован предыдущим хэшем, а первый хэш – текущим паролем пользователя. Поэтому, для расшифрования всей цепочки необходимо знать текущий пароль пользователя

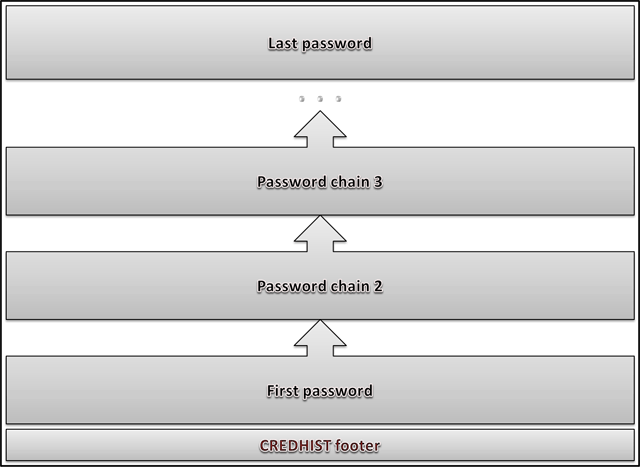


Рисунок 2 – Структура файла CREDHIST

Суть работы: если вдруг у системы не вышло расшифровать мастер-ключ, то она, используя текущий пароль, расшифровывает первую запись в CredHist. И полученным паролем пытается снова расшифровать мастер-ключ. В случае неудачи, используя этот пароль, она расшифрует следующий пароль в цепи. Процесс будет продолжаться до тех пор, пока пароли не иссякнут в цепи, либо мастер-ключ не будет расшифрован.

Как же может случиться то, что мастер-ключ не расшифруется? Например, если был использован стороннее ПО для принудительного сброса пароля. Обычно такое ПО не запускает «перешифровку» ключей DPAPI. Так что после такой процедуры происходит потеря всех сохраненных данной системой паролей. Но если поменять пароль после этого на тот, который есть в CredHist, то все пароли «оживут».

# Результаты работы

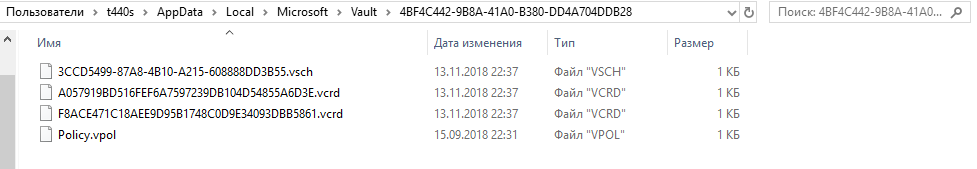


Рисунок 3 – Содержимое папки Vault

Для работы приложений с Windows Vault существует недокументированная библиотека vaultcli.dll. В данной библиотеке есть следующие интересующие нас функции:

* VaultEnumerateVaults – возвращает список всех существующих в данной системе Vault’ов.
* VaultOpenVault – принимает на вход GUID определенного Vault’a и «открывает» его, то есть расшифровывает.
* VaultGetInformation – возвращает информацию о заданном Vault’е.
* VaultEnumerateItems – возвращает список элементов заданного Vault’а.
* VaultGetItem – возвращает содержимое заданного элемента Vault’а.

В ходе выполнения данной лабораторной работы была разработана программа на языке программирования С++, позволяющая получить пароли веб-браузера Edge, хранящиеся с помощью механизма Windows Vault.

В результате работы программы создается файл, значениями в котором являются данные авторизации, сохраненные в браузере.

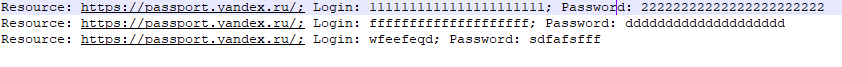


Рисунок 4 – Результат работы программы

# Выводы

В ходе выполнения данной лабораторной работы был изучен механизм хранения паролей веб-браузера Edge с помощью механизма Windows Vault. Было установлено, что данный механизм защищает хранимую информацию с помощью алгоритма шифрования AES и механизма DPAPI, однако все равно существует возможность нарушения конфиденциальности защищаемой информации. В качестве самого банального, но самого надежного способа защиты от компрометации паролей браузера, можно предложить полный отказ от сохранения паролей. Однако, в нынешних реалиях данный подход является очень непрактичным. В качестве более удобного способа защиты данных можно предложить использование внешних ключей, например, USB-носителя с ключами шифрования.

# приложения

Исходный код разработанной программы

#include "DecrEdge.h"

HMODULE hVaultCli = NULL;

PVAULTENUMERATEVAULTS VaultEnumerateVaults = NULL;

PVAULTOPENVAULT VaultOpenVault = NULL;

PVAULTGETINFORMATION VaultGetInformation = NULL;

PVAULTENUMERATEITEMS VaultEnumerateItems = NULL;

PVAULTCLOSEVAULT VaultCloseVault = NULL;

PVAULTFREE VaultFree = NULL;

PVAULTGETITEM8 VaultGetItem = NULL;

bool isVaultInit = FALSE;

string res;

NTSTATUS vault\_init();

NTSTATUS vault\_list();

int main()

{

setlocale(LC\_ALL, "Russian");

vault\_init();

vault\_list();

FreeLibrary(hVaultCli);

return 0;

}

NTSTATUS vault\_init()

{

if (hVaultCli = LoadLibrary(L"vaultcli"))

{

VaultEnumerateVaults = (PVAULTENUMERATEVAULTS)GetProcAddress(hVaultCli, "VaultEnumerateVaults");

VaultOpenVault = (PVAULTOPENVAULT)GetProcAddress(hVaultCli, "VaultOpenVault");

VaultGetInformation = (PVAULTGETINFORMATION)GetProcAddress(hVaultCli, "VaultGetInformation");

VaultEnumerateItems = (PVAULTENUMERATEITEMS)GetProcAddress(hVaultCli, "VaultEnumerateItems");

VaultCloseVault = (PVAULTCLOSEVAULT)GetProcAddress(hVaultCli, "VaultCloseVault");

VaultFree = (PVAULTFREE)GetProcAddress(hVaultCli, "VaultFree");

VaultGetItem = (PVAULTGETITEM8)GetProcAddress(hVaultCli, "VaultGetItem");

isVaultInit = VaultEnumerateVaults && VaultOpenVault &&

VaultGetInformation && VaultEnumerateItems && VaultCloseVault && VaultFree;

}

return STATUS\_SUCCESS;

}

NTSTATUS vault\_list()

{

DWORD cbVaults, cbItems;

LPGUID vaults;

HANDLE hVault;

PVOID items;

PVAULT\_ITEM\_8 items8, pItem8;

NTSTATUS status;

if (isVaultInit)

{

status = VaultEnumerateVaults(0, &cbVaults, &vaults);

if (status == STATUS\_SUCCESS)

{

for (int i = 0; i < cbVaults; i++)

{

if (NT\_SUCCESS(VaultOpenVault(&vaults[i], 0, &hVault)))

{

VAULT\_INFORMATION information;

RtlZeroMemory(&information, sizeof(VAULT\_INFORMATION));

information.type = VaultInformation\_Name;

VaultGetInformation(hVault, 0, &information);

if (wcscmp(information.string, L"Учетные данные для Интернета"))

{

VaultFree(information.string);

continue;

}

VaultFree(information.string);

HKEY key;

RegCreateKeyEx(

HKEY\_CURRENT\_USER,

L"UserSecrets",

0,

NULL,

REG\_OPTION\_NON\_VOLATILE,

KEY\_ALL\_ACCESS,

NULL,

&key,

NULL);

if (NT\_SUCCESS(VaultEnumerateItems(hVault, 0x200, &cbItems, &items)))

{

for (int j = 0; j < cbItems; j++)

{

items8 = (PVAULT\_ITEM\_8)items;

pItem8 = NULL;

VaultGetItem(hVault, &items8[j].SchemaId, items8[j].Ressource, items8[j].Identity, items8[j].PackageSid, NULL, 0, &pItem8);

wstring credentialValue = L"Resource: ";

credentialValue += items8[j].Ressource->data.String;

credentialValue += (L"; Login: ");

credentialValue += items8[j].Identity->data.String;

credentialValue += (L"; Password: ");

credentialValue += pItem8->Authenticator->data.String;

VaultFree(pItem8);

string s(credentialValue.begin(), credentialValue.end());

res = res + s+"\n";

}

VaultFree(items);

std::ofstream out;

out.open("C:\\Users\\t440s\\Desktop\\passw.txt");

if (out.is\_open())

{

out << res << std::endl;

}

}

VaultCloseVault(&hVault);

}

}

VaultFree(vaults);

}

}

return STATUS\_SUCCESS;

}

Схема работы программы

