МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования
«Сибирский государственный аэрокосмический университет имени академика М.Ф. Решетнева»

Институт информатики и телекоммуникаций

Кафедра безопасности информационных технологий

КУРСОВАЯ РАБОТА (ПРОЕКТ)

по дисциплине: безопасность операционных систем

на тему: Двухфакторная аутентификация с помощью провайдера аутентификации ОС Microsoft Windows

Выполнил студент: группы БКБ12-01
очной формы обучения
Саядян Руслан Каренович

Руководитель: ст. преподаватель
Лубкин Иван Александрович

Дата сдачи: «_____»_____20___г.
Дата защиты: «____»_____20__г.
Оценка:_____ _____

Красноярск 2014 г.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

С давних времён перед людьми стояла довольно сложная задача — убедиться в достоверности важных сообщений и подлинности пользователей. Придумывались речевые пароли, сложные печати, методы аутентификации с применением механических устройств и т.д.

Аутентификация — это процедура, проверяющая, имеет ли пользователь с предъявленным идентификатором право на доступ к ресурсу. Проще говоря, проверка соответствия имени входа и пароля. Аутентификация выполняется программным модулем, находящимся непосредственно на компьютере. Сначала модуль запрашивает пароль пользователя, после чего пользователь вводит свою аутентификационную информацию, которая сравнивается с эталоном. На основании результатов этого сравнения пользователь считается опознанным или нет.

Ввиду все чаще случающихся атак на пользовательские аккаунты используется двухфакторная аутентификация.

Двухфакторная аутентификация — это метод аутентификации пользователя при помощи запроса аутентификационных данных двух разных типов, что обеспечивает двухслойную, а значит, более эффективную защиту аккаунта от несанкционированного проникновения. На практике это обычно выглядит так: первый рубеж — это логин и пароль, второй — специальный USB-ключ, носитель с уникальными данными, специальный код, приходящий по SMS или электронной почте или биометрические данные пользователя. В общем, суть подхода такова: чтобы куда-то попасть, нужно дважды подтвердить тот факт, что вы — это вы, причем при помощи двух «ключей», одним из которых вы владеете, а другой держите в памяти.

Данная курсовая работа актуальна, так как многие пользователи выбирают ненадежные пароли, либо хранят их в ненадежных местах (за пределами черепной коробки). Однако даже если пароль будет доступен посторонним субъектам, благодаря двухфакторной аутентификации они не смогут получить доступ к защищаемому объекту.

Целью данной курсовой работы является создания провайдера двухфакторной аутентификации Windows.

Для достижения поставленной цели необходимо решить следующие задачи:

1. изучить стандарт Component Object Model (COM) и интерфейс ICredentialProvider;
2. создать собственный провайдер двухфакторной аутентификации для ОС Microsoft Windows с использованием внешнего носителя.

ГЛАВА 1 ОБЪЕКТНАЯ МОДЕЛЬ КОМПОНЕНТОВ

1.1 Общие сведения

Провайдер аутентификации Windows представляет собой DLL, которая использует два COM-интерфейса, необходимых для его разработки: ICredentialProvider и ICredentialProviderCredential. Первая предоставляет функционал для перечисления учетных записей, вторая предоставляет функционал, необходимый определенной учетной записи для аутентификации. COM определяет модель и требования к этим интерфейсам, чтобы они могли взаимодействовать с другими объектами.

Модель компонентного объекта фирмы Microsoft является, как следует из её названия, моделью для проектирования и создания компонентных объектов. Модель определяет множество технических приемов, которые могут быть использованы разработчиком при создании независимых от языка программных модулей, в которых соблюдается определенный двоичный стандарт. Корпорация Microsoft обеспечивает реализацию модели COM во всех своих Windows-средах. В других операционных средах, таких как Macintosh и UNIX, технология COM также поддерживается, но не обязательно средствами фирмы Microsoft.

Одной из наиболее важных черт COM является ее способность предоставлять двоичный стандарт для программных компонентов. Этот двоичный стандарт обеспечивает средства, с помощью которых объекты и компоненты, разработанные на разных языках программирования разными поставщиками и работающие в различных операционных системах, могут взаимодействовать без каких-либо изменений в двоичном (исполняемом) коде. Это является основным достижением создателей COM и отвечает насущным потребностям сообщества разработчиков программ. Многоразовое использование программного обеспечения является одной из первоочередных задач при его разработке и обеспечивается составляющими его модулями, которые должны работать в разнообразных средах. Обычно программное обеспечение разрабатывается с использованием определенного

языка программирования, например C++, и может эффективно применяться только в том случае, если другие разработчики компонентов также применяют C++. Например, если мы разрабатываем C++ класс, предназначенный для манипулирования с данными, то необходимым условием его использования в других приложениях является их разработка на языке C++. Только C++ компиляторы могут распознать C++ классы. Фактически, поскольку средства C++ не поддерживают никакого стандартного способа адаптации вызовов C++ функций к новой программной среде, использование программного обеспечения в этой новой среде требует применения такого же (или аналогичного) инструментального средства для его обработки. Другими словами, использование класса в другой операционной среде требует обязательного переноса в эту среду исходного текста программы данного класса. Применение двоичного кода позволяет разработчику создавать программные компоненты, которые могут применяться без использования языков, средств и систем программирования, а только с помощью двоичных компонентов (например, DLL- или EXE - файлов). Эта возможность является для разработчиков очень привлекательной. Ведь теперь они могут выбирать наиболее удобный для себя язык и средство разработки компонентов, не заботясь о языке и средствах, которые будет использовать другой разработчик.

1.2 COM-объекты и интерфейсы

Одним из главных преимуществ разработки с помощью объектно-ориентированных языков, таких как C++ и Java, является возможность эффективной инкапсуляции внутренних функций и данных. Это осуществимо именно благодаря объектной ориентированности этих языков. В объекте скрыты способы его реализации, а "наружу" предоставляется только хорошо определенный интерфейс, позволяющий внешним клиентам эффективно использовать функциональные возможности объекта. Технология COM обеспечивает эти возможности также с помощью

определения стандартных способов реализации и предоставления интерфейсов COM-объекта.

Использование виртуальных функций в базовом классе является центральным моментом в проектировании COM-компонентов. Определение абстрактного класса порождает таблицу, содержащую только открытые методы (т.е. интерфейс) класса. Класс не содержит переменных-членов и функций реализации объекта. Его единственной задачей является порождение производного класса для виртуальной реализации методов интерфейса компонента. В технологии COM доступ к компонентам обеспечивается только с помощью указателей на виртуальные таблицы. Таким образом, прямой доступ к конкретным данным компонента становится невозможным.

При построении COM-компонента первым делом нужно реализовать интерфейс, который должны использовать все COM-компоненты: IUnknown. Интерфейс IUnknown выполняет две функции. Первая состоит в том, чтобы обеспечить стандартный способ запроса определенного интерфейса данного компонента его пользователем (клиентом). Эту возможность предоставляет метод QueryInterface. Вторая функция состоит в обеспечении способа управления временем жизни компонента извне. Интерфейс IUnknown предоставляет два метода (AddRef и Release), обеспечивающих управление временем жизни экземпляра компонента.

Метод QueryInterface обращается к идентификатору интерфейса (Interface Identifier — IID), который представляет собой 128-битовый уникальный идентификатор, и возвращает указатель на определенный интерфейс (например, IUnknown, IMath), предоставляемый COM-объектом. Указатель возвращается через второй параметр, который является указателем на указатель типа void.

ГЛАВА 2 ПОСТАВЩИК УЧЕТНЫХ ДАННЫХ

2.1 Архитектура входа в систему

В каждом сеансе, кроме нулевого, есть экземпляр процесса winlogon. В Приложении №1 показано, что новый процесс LogonUI зарегистрировал в системе и загрузил несколько поставщиков учетных данных. За визуализацию отвечает процесс LogonUI, который встроен в ОС: требуется, чтобы каждый поставщик учетных данных перечислял свои элементы пользовательского интерфейса. Например, в конкретной ситуации поставщик может указать процессу LogonUI, что ему нужны два поля ввода, два заголовка, флажок и растровое изображение. В свою очередь, процесс LogonUI отображает эти элементы от имени поставщика учетных данных.

В группе разработчиков корпорации Майкрософт, ответственной за архитектуру поставщика учетных данных, полагали, что сторонним разработчикам будет удобнее работать с моделью подключаемых модулей на базе технологии COM. Теперь рассмотрим пример программы, который поможет разобраться в интерфейсе credprov.

2.2 Поведение поставщика учетных данных во время выполнения

В Таблице 1 приведен список событий, которые происходят в процессе выполнения примера. Вызовам интерфейса ICredentialProvider (Приложение №2) предшествует обозначение «Provider::». Вызовам интерфейса ICredentialProviderCredential (Приложение №2) предшествует обозначение «Credential::».

Таблица 1 – последовательность вызовов поставщика учетных данных.

Событие	Описание
1. [The system boots] 2. [LogonUI.exe process is created] 3. [Credential provider DLLs are loaded] 4. Provider::CreateInstance	Winlogon запускает процесс LogonUI. После создания процесс LogonUI перечисляет все поставщики учетных данных, зарегистрированных в HKLM\Software\Microsoft\Windows\CurrentVersion\Authentication\Credential Providers. Каждая библиотека DLL поставщика загружается и получает вызов метода Provider::CreateInstance. В результате создается экземпляр объекта провайдера.
5. [User presses Ctrl+Alt+Del] 6. Provider::SetUsageScenario (CPUS_LOGON) 7. Credential::Initialize	Пользователь нажмет сочетание клавиш Ctrl+Alt+Delete, и каждый поставщик получит уведомление Provider::SetUsageScenario CPUS_LOGON. Это указывает поставщику на то, что пользователь хочет выполнить интерактивный вход в систему. Затем создается экземпляр объекта Credential, который связывается с текущим экземпляром Provider. Затем выполняется вызов метода Credential::Initialize.
8. Provider::Advise	Процесс LogonUI вызывает метод Provider::Advise для каждого загруженного поставщика. Целью этого вызова является предоставление поставщикам механизма для асинхронного уведомления процесса LogonUI о любом желаемом изменении в видимых элементах пользовательского интерфейса (ни одного из которых еще нет).
9. Provider::GetCredentialCount	Это указывает на количество учетных данных, которые поставщик хочет перечислить.
10. Provider::GetCredentialAt (dwIndex = 0)	В ответ поставщик возвращает указатель ICredentialProviderCredential на экземпляр учетных данных, который соответствует запрошенному индексу.

11. Provider::GetFieldDescriptorCount	Через это вызов поставщик возвращает максимальное количество элементов пользовательского интерфейса, которые можно найти в его учетных данных.
12. Provider::GetFieldDescriptorAt (dwIndex = 0) . . . 16. Provider::GetFieldDescriptorAt (dwIndex = 4) 17. Credential::GetBitmapValue (dwFieldID = 0; tile image) 18. Credential::GetStringValue (dwFieldID = 1; user name field) 19. Credential::GetFieldState (dwFieldID = 1; user name field) . . . 24. Credential::GetStringValue (dwFieldID = 4; domain name field) 25. Credential::GetFieldState (dwFieldID = 4; domain name field)	LogonUI вызывает метод Provider::GetFieldDescriptorAt. Это делается один раз для каждого элемента пользовательского интерфейса, и при каждом вызове возвращается тип элемента. Например, в ответ на вызов, соответствующий индексу растрового изображения, пример возвращает параметр CREDENTIAL_PROVIDER_FIELD_TYPE CPFT_TILE_IMAGE.
26. Credential::Advise	Этот вызов имеет ту же цель, что и выполненный ранее вызов метода Provider::Advise. Каждый элемент учетных данных может асинхронно уведомить процесс LogonUI о соответствующих изменениях, влияющих на состояние компонентов его пользовательского интерфейса.
27. Credential::GetSerialization	С точки зрения проверки подлинности пользователя наибольший интерес представляет этот вызов. В результате установки значения параметра *pbAutoLogonWithDefault метода GetCredentialCount равным ИСТИНА процессу LogonUI становится известно, что используемые по умолчанию учетные данные

	уже должны содержать достаточно информации для проверки подлинности пользователя (даже если на экране не показан ни один элемент пользовательского интерфейса, а пользователь, в свою очередь, ничего не ввел). В этом случае для получения имени пользователя, пароля и необязательного имени домена вызывается подпрограмма Credential::GetSerialization. Поставщик учетных данных готовит возвращаемое значение для этой подпрограммы путем преобразования этих трех элементов в формат, пригодный для протокола Kerberos. После сериализации учетных данных поставщик сообщает процессу LogonUI через выходной параметр типа CREDENTIAL_PROVIDER_GET_SERIALIZATION _RESPONSE, что возвращается полный набор учетных данных. Это отслеживается по значению параметра CPGSR_RETURN_CREDENTIAL_FINISHED.
28. Credential::UnAdvise 29. Provider::UnAdvise	LogonUI вызывает методы Credential::UnAdvise и Provider::UnAdvise. Этим объекты предупреждаются о том, что уведомления на их соответствующие интерфейсы событий (Events) не принимаются.
30. [The WinLogon process calls LogonUser]	После вызова метода GetSerialization упакованные учетные данные передаются от процесса LogonUI процессу winlogon, который, в свою очередь, с помощью вызова метода LogonUser передает их Local Security Authority.
31. Credential::Advise 32. Credential::ReportResult (ntsStatus = 0) 33. Credential::UnAdvise	Перед тем, как учетные данные получат код состояния, полученный от метода LogonUser, они снова передаются на интерфейс обратного вызова для внесения изменений в элементы пользовательского интерфейса. Результат попытки проверки подлинности возвращается в учетные данные с помощью подпрограммы Credential::ReportResult.

ГЛАВА 3 РЕАЛИЗАЦИЯ И ИСПОЛЬЗОВАНИЕ СОБСТВЕННОГО ПРОВАЙДЕРА АУТЕНТИФИКАЦИИ

3.1 Требования

Провайдер аутентификации должен предоставлять возможность аутентификации с использованием пароля. Вторым фактором аутентификации служит внешний носитель с файлом, содержащим уникальный для данного пользователя ключ.

Также требуется реализовать централизованное безопасное хранение необходимых данных для аутентификации и возможность удобного их редактирования с использованием графического интерфейса.

3.2 Реализация

Стандартный провайдер аутентификации Windows удовлетворяет практически всем требованиям, однако не позволяет использовать двухфакторную аутентификацию с использованием внешнего носителя. Реализовывать функционал существующего провайдера не имеет смысла, поэтому была создана «обертка» стандартного провайдера.

Провайдеры аутентификации являются COM-объектами, значит их можно создавать и использовать как любой другой COM-объект. В нашем провайдере мы используем CLSID_PasswordCredentialProvider и направляем все вызовы через наш провайдер.

В SetUsageScenario (Приложение №3) создается экземпляр стандартного провайдера и запрашивается интерфейс для него. Затем указатель на него присваивается свойству нашего провайдера _pWrappedProvider. Далее провайдер направляет все вызовы _pWrappedProvider, вызывая его методы.

Отдельно рассмотрим использование стандартного CredentialProviderCredential, провайдера учетных данных для одной конкретной учетной записи, в нашем собственном провайдере. В GetCredentialCount создается как экземпляр стандартного поставщика, так и нашего, после чего вызывается метод нашего поставщика Initialize

(Приложение №4). В аргументах передается указатель на стандартный провайдер. В последствии наш поставщик CSampleCredential вызывает его методы.

Основная особенность заключается в CSampleCredential::GetSerialization (Приложение №5). Именно здесь реализуется поиск накопителя, файла и его содержимого, проверка соответствия пользователя.

Сначала вызывается функция GetKey, которая с помощью вызова FindFirstVolume и FindNextVolume получает список подключенных внешних носителей. На каждом диске проверяется наличие файла с уникальной последовательностью. Если файл найден, ключ возвращается через указатель, переданный через аргументы.

В случае успешного завершения GetKey, данные из файла хешируются и хеш сравнивается со значением из реестра. Ключем реестра является имя пользователя, запрашиваемое у стандартного обертываемого провайдера вызовом GetStringValue. Если значения сверток совпадают, пароль проверяется стандартным провайдером аутентификации Windows, вызывая его GetSerialization, это и есть причина использования обертки вместо создания нового провайдера «с нуля». Иначе выводится сообщение об ошибке и GetSerialization не вызывается.

Вопрос безопасного хранения решается использованием реестра Windows. В «HKEY_LOCAL_MACHINE\SOFTWARE\USBCredProv\Users» хранятся хеш-свертки ключей и имена пользователей.

Для редактирования этой ветки реестра была создана программа CredentialManager (Приложение №6).

3.3 Установка и использование

Для установки созданного провайдера аутентификации и добавления пользователя необходимо:

– скопировать dll файл нашего провайдера в «\Windows\System32»;

– запустить файл Register.reg с помощью regedit.exe, чтобы добавить имя провайдера и уникальный GUID в реестр;

– с помощью CredentialManager.exe добавить пользователя, ввести ключ и выбрать носитель, куда будет сохранен файл ключа.

ЗАКЛЮЧЕНИЕ

Двухфакторная аутентификация является довольно надежным способом аутентификации.

ОС Microsoft Windows (начиная с Vista) предоставляет возможность создания собственных провайдеров аутентификации с использованием объектной модели компонентов и стандартных интерфейсов. Благодаря этому создание провайдера двухфакторной аутентификации не является сложной задачей.

В данной курсовой работе была рассмотрена объектная модель компонентов, а также интерфейсы ICredentialProvider и ICredentialProviderCredential. Поставленные задачи выполнены путем создания «обертки» для стандартного провайдера аутентификации.

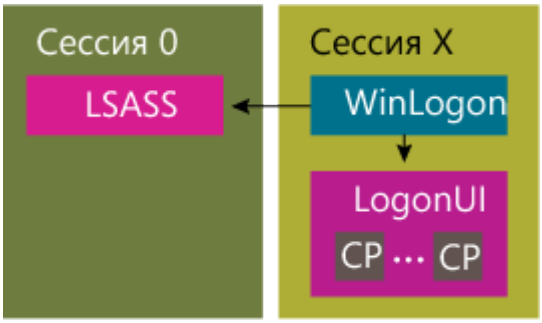
БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Введение в COM [электронный ресурс]: URL http://rsdn.ru/article/com/introcom.xml

2 Создание специальных возможностей входа с помощью поставщиков учетных данных для Windows Vista [электронный ресурс]: URL http://msdn.microsoft.com/ru-ru/magazine/cc163489.aspx

ПРИЛОЖЕНИЕ №1

Архитектура входа в систему



ПРИЛОЖЕНИЕ №2

Интерфейсы ICredentialProvider и ICredentialProviderCredential

```
ICredentialProvider : public IUnknown
{
    HRESULT STDMETHODCALLTYPE SetUsageScenario(
        /* [in] */ CREDENTIAL_PROVIDER_USAGE_SCENARIO cpus,
        /* [in] */ DWORD dwFlags);

    HRESULT STDMETHODCALLTYPE SetSerialization(
        /* [in] */ const CREDENTIAL_PROVIDER_CREDENTIAL_SERIALIZATION
            *pcpcs);

    HRESULT STDMETHODCALLTYPE Advise(
        /* [in] */ ICredentialProviderEvents *pcpe,
        /* [in] */ UINT_PTR upAdviseContext);

    HRESULT STDMETHODCALLTYPE UnAdvise( void);

    HRESULT STDMETHODCALLTYPE GetFieldDescriptorCount(
        /* [out] */ DWORD *pdwCount);

    HRESULT STDMETHODCALLTYPE GetFieldDescriptorAt(
        /* [in] */ DWORD dwIndex,
        /* [out] */ CREDENTIAL_PROVIDER_FIELD_DESCRIPTOR **ppcpfd);

    HRESULT STDMETHODCALLTYPE GetCredentialCount(
        /* [out] */ DWORD *pdwCount,
        /* [out] */ DWORD *pdwDefault,
        /* [out] */ BOOL *pbAutoLogonWithDefault);

    HRESULT STDMETHODCALLTYPE GetCredentialAt(
        /* [in] */ DWORD dwIndex,
        /* [out] */ ICredentialProviderCredential **ppcpc);
};
```

```
ICredentialProviderCredential : public IUnknown
{
    HRESULT STDMETHODCALLTYPE Advise(
        /* [in] */ ICredentialProviderCredentialEvents *pcpce);

    HRESULT STDMETHODCALLTYPE UnAdvise( void);

    HRESULT STDMETHODCALLTYPE SetSelected(
        /* [out] */ BOOL *pbAutoLogon);

    HRESULT STDMETHODCALLTYPE SetDeselected( void);

    HRESULT STDMETHODCALLTYPE GetFieldState(
        /* [in] */ DWORD dwFieldID,
        /* [out] */ CREDENTIAL_PROVIDER_FIELD_STATE *pcpfs,
        /* [out] */ CREDENTIAL_PROVIDER_FIELD_INTERACTIVE_STATE *pcpfis);

    HRESULT STDMETHODCALLTYPE GetStringValue(
        /* [in] */ DWORD dwFieldID,
        /* [string][out] */ LPWSTR *ppsz);

    HRESULT STDMETHODCALLTYPE GetBitmapValue(
```

```
        /* [in] */ DWORD dwFieldID,
        /* [out] */ HBITMAP *phbmp);

    HRESULT STDMETHODCALLTYPE GetCheckboxValue(
        /* [in] */ DWORD dwFieldID,
        /* [out] */ BOOL *pbChecked,
        /* [string][out] */ LPWSTR *ppszLabel);

    HRESULT STDMETHODCALLTYPE GetSubmitButtonValue(
        /* [in] */ DWORD dwFieldID,
        /* [out] */ DWORD *pdwAdjacentTo);

    HRESULT STDMETHODCALLTYPE GetComboBoxValueCount(
        /* [in] */ DWORD dwFieldID,
        /* [out] */ DWORD *pcItems,
        /* [out] */ DWORD *pdwSelectedItem);

    HRESULT STDMETHODCALLTYPE GetComboBoxValueAt(
        /* [in] */ DWORD dwFieldID,
        DWORD dwItem,
        /* [string][out] */ LPWSTR *ppszItem);

    HRESULT STDMETHODCALLTYPE SetStringValue(
        /* [in] */ DWORD dwFieldID,
        /* [string][in] */ LPCWSTR psz);

    HRESULT STDMETHODCALLTYPE SetCheckboxValue(
        /* [in] */ DWORD dwFieldID,
        /* [in] */ BOOL bChecked);

    HRESULT STDMETHODCALLTYPE SetComboBoxSelectedValue(
        /* [in] */ DWORD dwFieldID,
        /* [in] */ DWORD dwSelectedItem);

    HRESULT STDMETHODCALLTYPE CommandLinkClicked(
        /* [in] */ DWORD dwFieldID);

    HRESULT STDMETHODCALLTYPE GetSerialization(
        /* [out] */ CREDENTIAL_PROVIDER_GET_SERIALIZATION_RESPONSE
                    *pcpgsr,
        /* [out] */ CREDENTIAL_PROVIDER_CREDENTIAL_SERIALIZATION *pcpcs,
        /* [out] */ LPWSTR *ppszOptionalStatusText,
        /* [out] */ CREDENTIAL_PROVIDER_STATUS_ICON
                    *pcpsiOptionalStatusIcon);

    HRESULT STDMETHODCALLTYPE ReportResult(
        /* [in] */ NTSTATUS ntsStatus,
        /* [in] */ NTSTATUS ntsSubstatus,
        /* [out] */ LPWSTR *ppszOptionalStatusText,
        /* [out] */ CREDENTIAL_PROVIDER_STATUS_ICON
*pcpsiOptionalStatusIcon);
};
```

ПРИЛОЖЕНИЕ №3

Метод SetUsageScenario

```
HRESULT CSampleProvider::SetUsageScenario(
    CREDENTIAL_PROVIDER_USAGE_SCENARIO cpus,
    DWORD dwFlags
    )
{
    HRESULT hr;
    IUnknown *pUnknown = NULL;
    hr = ::CoCreateInstance(CLSID_PasswordCredentialProvider, NULL,
CLSCTX_ALL, IID_PPV_ARGS(&pUnknown));
    if (SUCCEEDED(hr))
    {
        hr = pUnknown->QueryInterface(IID_PPV_ARGS(&(_pWrappedProvider)));
        if (SUCCEEDED(hr))
        {
            hr = _pWrappedProvider->SetUsageScenario(cpus, dwFlags);
        }
    }
    if (FAILED(hr))
    {
        if (_pWrappedProvider != NULL)
        {
            _pWrappedProvider->Release();
            _pWrappedProvider = NULL;
        }
    }

    return hr;
}
```

ПРИЛОЖЕНИЕ №4

Метод Initialize

```
HRESULT CSampleCredential::Initialize(
    const CREDENTIAL_PROVIDER_FIELD_DESCRIPTOR* rgcpfd,
    const FIELD_STATE_PAIR* rgfsp,
    ICredentialProviderCredential *pWrappedCredential,
    DWORD dwWrappedDescriptorCount
    )
{
    HRESULT hr = S_OK;

    if (_pWrappedCredential != NULL)
    {
        _pWrappedCredential->Release();
    }
    _pWrappedCredential = pWrappedCredential;
    _pWrappedCredential->AddRef();

    _dwWrappedDescriptorCount = dwWrappedDescriptorCount;

    for (DWORD i = 0; SUCCEEDED(hr) && i <
ARRAYSIZE(_rgCredProvFieldDescriptors); i++)
    {
        _rgFieldStatePairs[i] = rgfsp[i];
        hr = FieldDescriptorCopy(rgcpfd[i], &_rgCredProvFieldDescriptors[i]);
    }

    if (SUCCEEDED(hr))
    {
        hr = SHStrDupW(L"I Work In:",
&_rgFieldStrings[SFI_I_WORK_IN_STATIC]);
    }
    if (SUCCEEDED(hr))
    {
        hr = SHStrDupW(L"Database", &_rgFieldStrings[SFI_DATABASE_COMBOBOX]);
    }

    return hr;
}
```

ПРИЛОЖЕНИЕ №5

Метод GetSerialization

```
DWORD GetKey(void ** key, DWORD * keySize)
{
	char fileName[] = ":\\Users\\Ruslan\\cred.txt";
	char volName[MAX_PATH] = "";
	char volPathName[(MAX_PATH + 1) * sizeof(char)] = "";
	DWORD retSize = NULL;
	char * dir = NULL;
	HANDLE hFile = NULL;
	HANDLE hList = FindFirstVolume(volName, MAX_PATH);
	if (hList == INVALID_HANDLE_VALUE)
		return ERROR;
	do
	{
		if (GetVolumePathNamesForVolumeName(volName, volPathName,
(MAX_PATH + 1) * sizeof(char), &retSize))
		{
			dir = new char[sizeof fileName + 1];
			strncpy(dir, volPathName, 1);
			strncpy((dir + 1), fileName, sizeof fileName);
			hFile = CreateFile(dir, GENERIC_READ, NULL, NULL,
OPEN_EXISTING, NULL, NULL);
			delete[] dir;
			if (hFile != INVALID_HANDLE_VALUE)
			{
				*keySize = 0;
				*keySize = GetFileSize(hFile, keySize);

				*key = new BYTE[*keySize];

				ReadFile(hFile, *key, *keySize, keySize, NULL);

				CloseHandle(hFile);
				return 1;
			}
		}
	} while (FindNextVolume(hList, volName, MAX_PATH));
	return ERROR_NOT_FOUND;
}

HRESULT CSampleCredential::GetSerialization(
    CREDENTIAL_PROVIDER_GET_SERIALIZATION_RESPONSE* pcpgsr,
    CREDENTIAL_PROVIDER_CREDENTIAL_SERIALIZATION* pcpcs,
    PWSTR* ppwszOptionalStatusText,
    CREDENTIAL_PROVIDER_STATUS_ICON* pcpsiOptionalStatusIcon
    )
{
    HRESULT hr = E_UNEXPECTED;
	  DWORD keySize = 0;
	  void * key = 0;
	  DWORD res = GetKey(&key, &keySize);
	  DWORD dummy = 0, checkSum = 0;
	  HKEY hRegKey = 0;
	  if (res == ERROR)
	  {
		MessageBox(NULL, TEXT("Unexpected error."), TEXT("Error"), 0);
		return hr;
	  }
	  else if(res == ERROR_NOT_FOUND)
	  {
```

```
            MessageBox(NULL, TEXT("Cannot find the specified file."),
TEXT("Error"), 0);
            return hr;
      }
      CheckSumMappedFile(key, keySize, &dummy, &checkSum);
      checkSum = 206;
      std::stringstream ss;
      ss << keySize;
      MessageBox(NULL, ss.str().c_str(), TEXT("key"), 0);
      ss << checkSum;
      MessageBox(NULL, ss.str().c_str(), TEXT("checkSum"), 0);
      if (RegOpenKey(HKEY_LOCAL_MACHINE, "SOFTWARE\\USBCredProv\\Users",
&hRegKey) != ERROR_SUCCESS)
      {
            MessageBox(NULL, TEXT("Cannot open registry key."), TEXT("Error"),
0);
            return hr;
      }

      PWSTR userName = 0;
      _pWrappedCredential->GetStringValue(0, &userName);
      DWORD size = sizeof DWORD;
      BYTE * regCheckSum = new BYTE[size];
      if ((RegQueryValueExW(hRegKey, userName, NULL, NULL, regCheckSum,
&size)) != ERROR_SUCCESS)
      {
            MessageBox(NULL, TEXT("Cannot get key value."), TEXT("Error"), 0);
            return hr;
      }


      if ((DWORD)*regCheckSum == checkSum)
      {
            if (_pWrappedCredential != NULL)
                  {
                        hr = _pWrappedCredential->GetSerialization(pcpgsr,
pcpcs, ppwszOptionalStatusText, pcpsiOptionalStatusIcon);
                  }
      }
      else
      {
            MessageBox(NULL, TEXT("Incorrect key."), TEXT("Error"), 0);
            return hr;
      }
      return hr;

}
```

ПРИЛОЖЕНИЕ №6

CredentialManager.exe