Министерство образования и науки Российской Федерации

Санкт-Петербургский политехнический университет Петра Великого

—

Институт прикладной математики и механики

**Кафедра «Информационная безопасность компьютерных систем»**

**ЛАБОРАТОРНАЯ РАБОТА №3**

**Планировщик задач Windows.**

по дисциплине «Безопасность современных информационных технологий»

Выполнил

студент гр. 33609/3 Палёный А.Э.

<*подпись*>

Руководитель Иванов Д.В.

<*подпись*>

Санкт-Петербург

2018

# ЦЕЛЬ РАБОТЫ

Изучить возможности встроенного в ОС Windows планировщика задач.

# ФОРМУЛИРОВКА ЗАДАНИЯ

Реализовать программу для управления планировщиком задач в ОС Windows, использующую возможности Windows API.

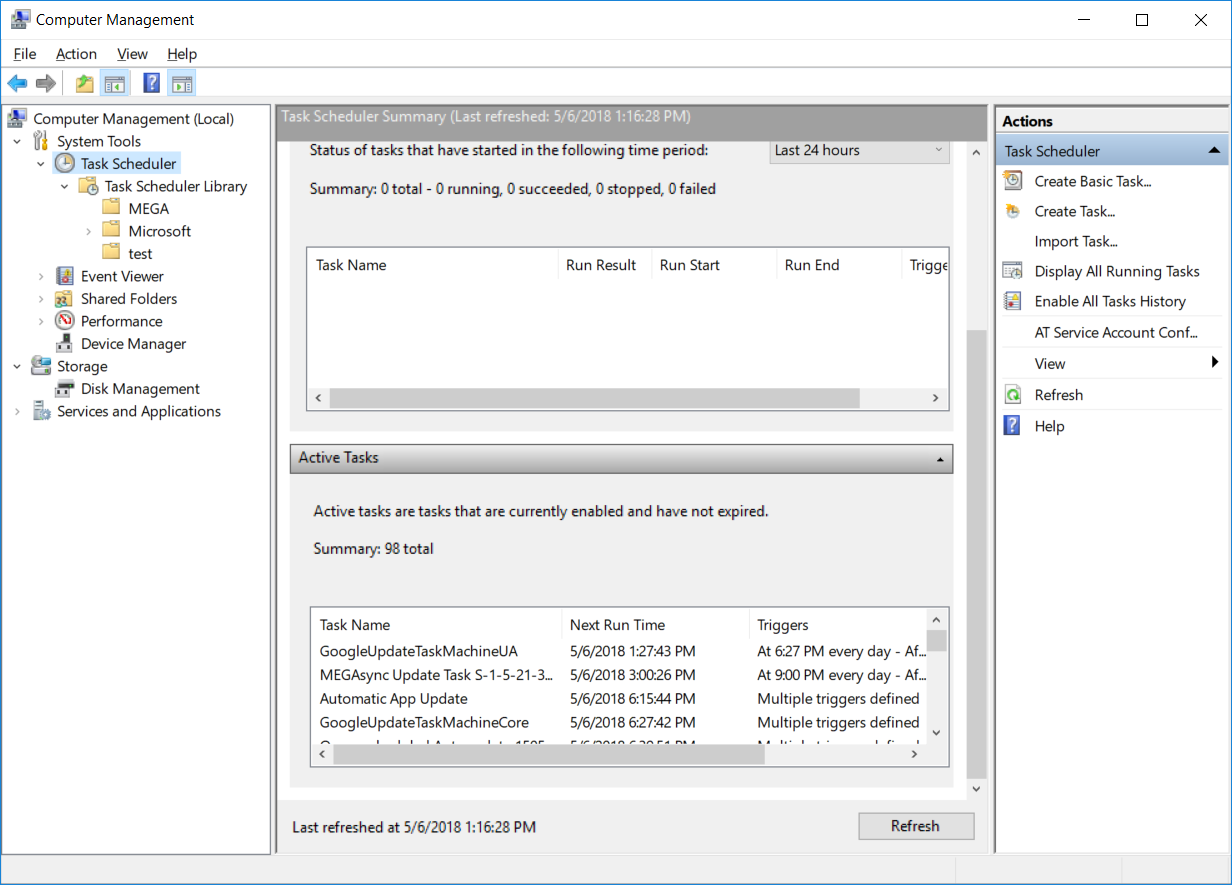
**Требования:**

* язык программирования: C/C++;
* разрабатываемая программа не должна использовать готовые обёртки над Windows API;
* программа должна уметь выводить список активных задач и их статус;
* создайте правило, уведомляющее пользователя о внесении изменений в механизмы безопасности Windows: в работу встроенного антивируса Windows Defender, изменении любых настроек брандмауэра;
* создайте правило, уведомляющее пользователя, когда брандмауэр Windows блокирует ping-запрос с определённого IP-адреса;
* поддержка ОС Windows 7 – 10.

1. **ХОД РАБОТЫ**

Была создана программа для управления планировщиком задач в ОС Windows.

Одна из функций данной программы – вывод список активных задач и их статус. Для того, чтобы понять, как это сделать, было необходимо просмотреть список активных задач вручную. Чтобы это сделать необходимо открыть «Computer Management». Дальше, в появившемся окне перейти «System Tools» -> «Task Scheduler». В окне посередине будет изменено содержимое. Теперь необходимо выбрать «Active Tasks». Чуть ниже появятся все активные задачи в системе. На изображении 1 показаны часть активный задач в системе.



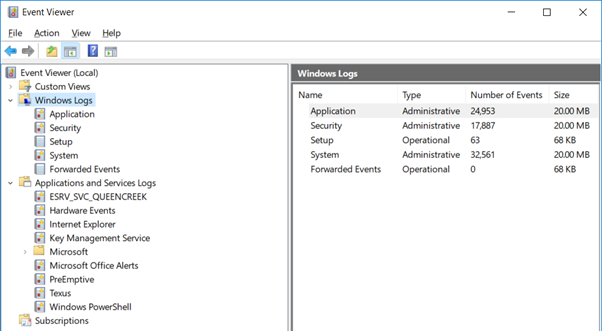
Изображение 1. Активные задачи в системе

Также, следует учесть, что задачи распределены по папкам. И для вывода всех задач в системе, необходимо пройтись по всем папкам и записать информацию о задачах.

Дальше, с помощь WinApi, был установлен специальный уровень безопасности для текущего процесса. Используя его, мы подключались к экземпляру объекта TaskScheduler. После это уже получали все свойства текущего объекта.

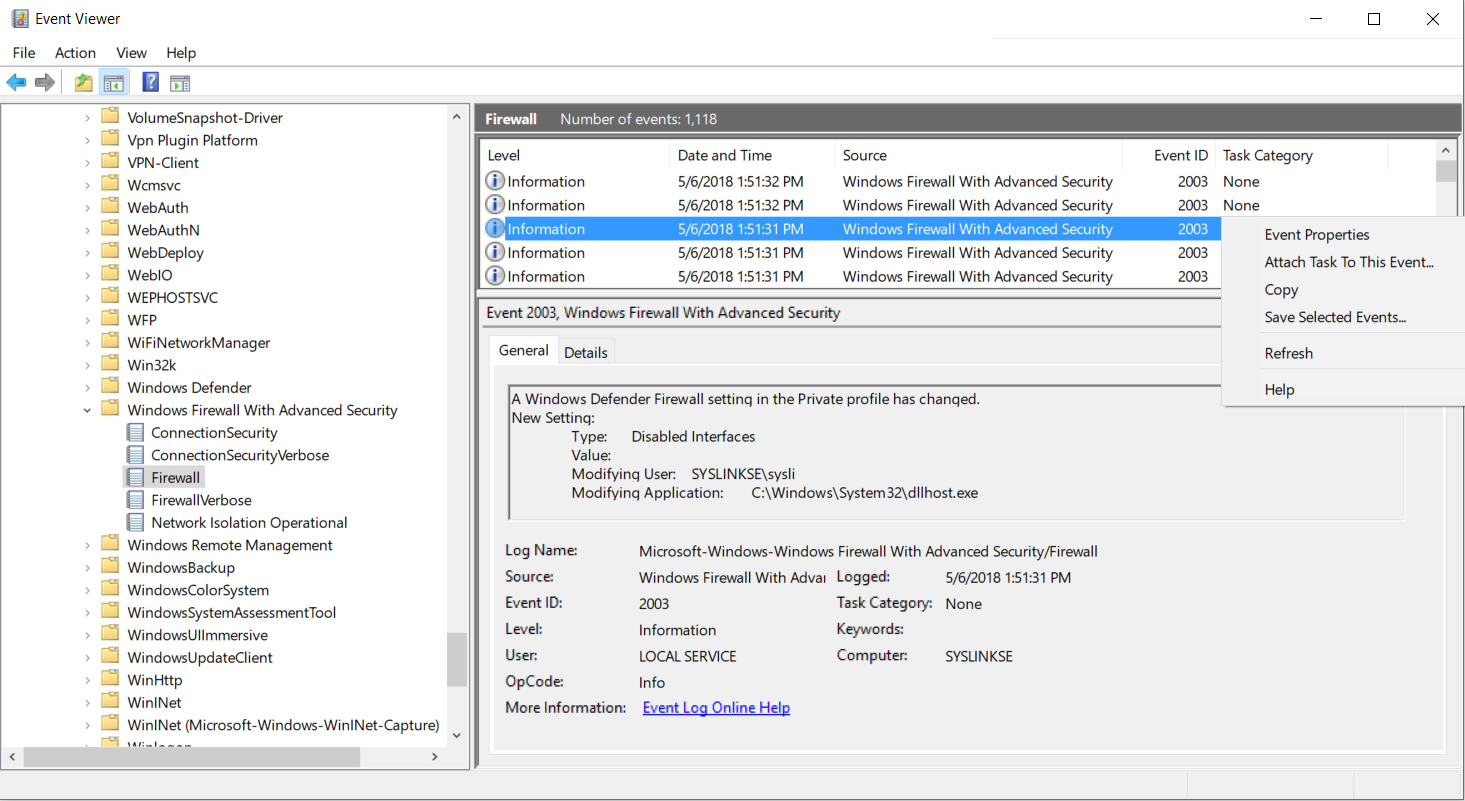
Дальше, необходимо было создавать правила, которые будут уведомлять пользователя. Для того, чтобы понять, как это работает, мы снова попробуем проделать все операции вручную.

Сначала заметим, что в системе есть программа «Event Viewer», которая будет выводить всю информацию обо всех событиях системы. На изображении 2 показан интерфейс данной программы.



Изображение 2. Списки событий в программе Event Viewer

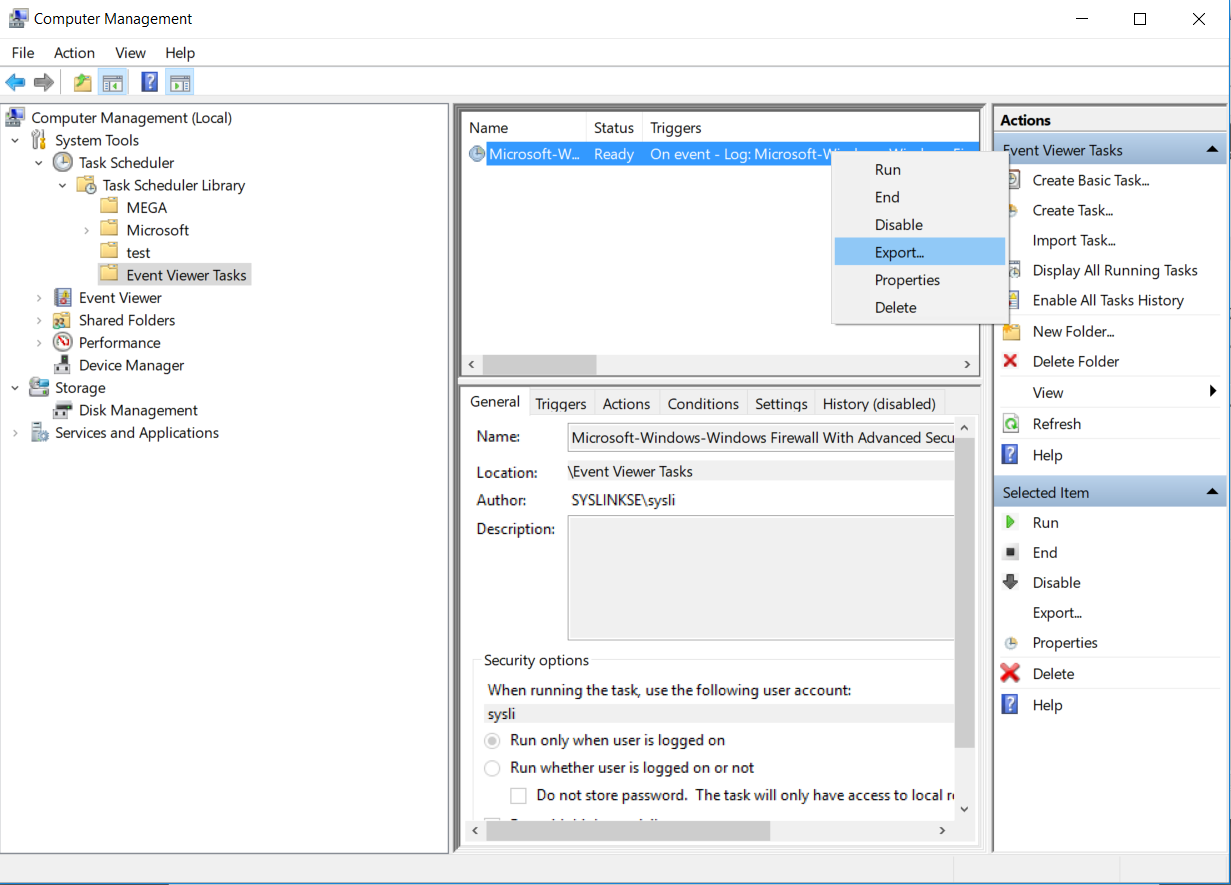
Заметим, также, что у программы есть 4 подпапки. Из них нам будут интересны только две – «Windows Logs» - журналы событий системы и «Application and Services Logs» - журналы событий приложений и сервисов. В «Application and Services Logs» перейдём по такому пути: «Microsoft» -> «Windows» -> «Windows Firewall With Advanced Security» -> «Firewall». Здесь будут отображены события брандмауэра Windows. Включая и отключая брандмауэр Windows, мы добиваемся того, чтобы появлялись события связанные с изменением состояния брандмауэра, как на изображении 3.



Изображение 3. События брандмауэра Windows

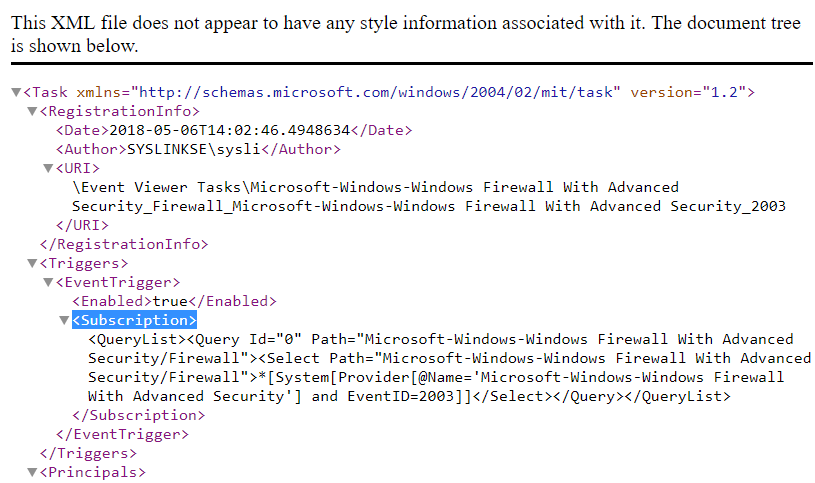
Нетрудно заметить, что события имеют Id 2003. Этот Id мы будем использовать при установке правила. Кликаем правой кнопкой мыши по событию и выбираем пункт «Attach Task To This Event». Кликаем 3 раза на кнопку «Next». Дальше выбираем какую программу запускать. Можно, например, запустить калькулятор по пути «C:\Windows\System32\calc.exe». Дальше жмём «Next» и после этого «Finish».

Дальше, находим созданное нами правило, как на изображении 4.



Изображение 4. Созданное нами правило

Дальше кликаем правой кнопкой мыши по созданному правилу и выбираем «export». Сохраняем файл и открываем его в браузере, например, Google Chrome. На изображении 5 в разделе «<Subscription>» изображена, необходимая нам строка, которую мы будем использовать в нашей программе при регистрации правила в Task Scheduler.



Изображение 5. Внутренний вид правила Task Scheduler

Дальнейшие действия были выполнены аналогичным образом. Следует отметить то, что для того чтобы получать события в Event Viewer об отбрасывании пакета, необходимо дополнительно изменить политику безопасности системы. Для англоязычной версии Windows необходимо в командной строке выполнить:

auditpol /set /subcategory:"Filtering Platform Packet Drop" /success:enable /failure:enable

Также, события об отбрасывании пакета будут отображены по пути: «Windows Logs» -> «Security» в «Event Viewer».

Чтобы вывести дополнительную информацию о событии можно использовать такую ссылку: <https://social.technet.microsoft.com/Forums/scriptcenter/en-US/34cc504f-1d9e-40de-a119-b3991e20fb6c/referencing-an-event-that-triggered-a-task?forum=ITCG>

1. **Ответы на контрольные вопросы**

*- Опишите, зачем нужен планировщик задач ОС Windows и какие есть альтернативы данному инструменту.*

Планировщик заданий - это компонент Windows, который может автоматически запускать задачи в определённое время или в ответ на конкретное событие. Например, задачи могут запускаться при запуске компьютера или при входе пользователя в систему. Альтернативы: Windows Shutdown Assistant, Freebyte Task Scheduler, System Scheduler, VisualCron, Advanced Task Scheduler.

*- Какие триггеры существуют во встроенном планировщике задач ОС Windows?*

Триггер события (триггер событий) - запускает задачу при возникновении определённого системного события.

Триггер времени (срабатывание по времени) – запускает задачу в определённую дату и время.

Ежедневный триггер (таймер календаря по времени) – запускает задачу в определённое время по ежедневному графику. Например, задача начинается в 8:00 утра каждый день или через день.

Еженедельный триггер (таймер календаря по времени) – запускает задачу в определённое время по недельному графику. Например, задача начинается в 8:00 утра в определённый день недели каждую неделю или в определённый день недели каждую другую неделю.

Ежемесячный триггер (таймер календаря по времени) – запускает задачу в определённое время по ежемесячному расписанию. Например, задача начинается в 8:00 в определённые дни месяца в определённые месяцы.

Ежемесячный триггер дня (таймер календаря по времени) – начинает задание в определённое время по ежемесячному расписанию. Например, задача начинается в 8:00 в определённые дни недели, недели месяца и месяцы года.

Триггер бездействия (инициируемый событиями триггер) – запускает задачу, когда компьютер переходит в состояние ожидания.

Триггер регистрации (инициируемый событиями триггер) – запускает задачу, когда задача зарегистрирована или обновлена.

Триггер загрузки системы (триггер, основанный на событиях) – запускает задачу при загрузке системы.

Триггер входа в систему (триггер, основанный на событиях) – запускает задачу при входе пользователя в систему.

Триггер изменения состояния сеанса (триггер, основанный на событиях) - запускает задачу при изменении состояния сеанса сервера терминалов.

*- Какие ограничения накладываются на задачи?*

Задачи выполняется в контексте конкретного пользователя который вошёл в систему. Существуют ограничения на количество задач:

1000 – максимальное количество задач, разрешённых для очереди в диспетчере сеансов.

1000 - максимальное количество экземпляров задачи, которые могут быть в состоянии RUNNING с повышенными привилегиями в любой момент времени.

1000 - максимальное количество экземпляров задачи, которые могут быть в состоянии RUNNING с обычными привилегиями в любой момент времени.

*- Перечислите типы задач, которые могут быть запущены с помощью планировщика.*

TASK\_ACTION\_EXEC – Это действие выполняет операцию командной строки. Например, действие может запускать скрипт, запускать исполняемый файл или, если указано имя документа, найти связанное приложение и запустить приложение с документом.

TASK\_ACTION\_COM\_HANDLER – Это действие вызывает обработчик. Это действие можно использовать, только если для свойства совместимости задачи установлено значение TASK\_COMPATIBILITY\_V2.

TASK\_ACTION\_SEND\_EMAIL – Это действие отправляет сообщение электронной почты. Это действие разрешено только для версий Windows 7 и ниже. В более старших версиях данное действие запрещено.

TASK\_ACTION\_SHOW\_MESSAGE – Это действие показывает окно сообщения. Это действие разрешено только для версий Windows 7 и ниже. В более старших версиях данное действие запрещено.

*- Опишите механизмы безопасности Windows, которые задействованы при регистрации новых задач.*

Чтобы получить право регистрировать задачи в системе необходимо пользователю состоять в группе Administrators.

1. **Вывод**

В результате была создана программа, позволяющая работать с Task Scheduler. Также был изучен Task Scheduler в OC Windows. Были получены основные навыки работы с ним.

1. **Исходный программы**

HRESULT \_\_CRTDECL AddTrigger(ITriggerCollection \*pTriggerCollection, wchar\_t \*TriggerId, wchar\_t\* policy, size\_t ValQueries, ...)

{

HRESULT hr;

va\_list queryData;

va\_start(queryData, ValQueries);

ITrigger \*pTrigger = NULL;

hr = pTriggerCollection->Create(TASK\_TRIGGER\_EVENT, &pTrigger);

if (FAILED(hr))

{

printf("\nCannot create the trigger: %x", hr);

return S\_FALSE;

}

IEventTrigger \*pEventTrigger = NULL;

hr = pTrigger->QueryInterface(

IID\_IEventTrigger, (void\*\*)&pEventTrigger);

pTrigger->Release();

if (FAILED(hr))

{

printf("\nQueryInterface call on IEventTrigger failed: %x", hr);

return S\_FALSE;

}

hr = pEventTrigger->put\_Id(\_bstr\_t(TriggerId));

if (FAILED(hr))

printf("\nCannot put trigger ID: %x", hr);

hr = pEventTrigger->put\_Subscription(

policy);

if (FAILED(hr))

{

printf("\nCannot set the event trigger: %x", hr);

pEventTrigger->Release();

return S\_FALSE;

}

ITaskNamedValueCollection \*pNamedValueQueries = NULL;

hr = pEventTrigger->get\_ValueQueries(&pNamedValueQueries);

if (FAILED(hr))

{

printf("\nCannot put the event collection: %x", hr);

pEventTrigger->Release();

return S\_FALSE;

}

for (size\_t i = 0; i < ValQueries; i++)

{

ITaskNamedValuePair\* pNamedValuePair = NULL;

wchar\_t \*tmpA = va\_arg(queryData, wchar\_t\*);

wchar\_t \*tmpB = va\_arg(queryData, wchar\_t\*);

hr = pNamedValueQueries->Create(

\_bstr\_t(tmpA), \_bstr\_t(tmpB), &pNamedValuePair);

pNamedValuePair->Release();

if (FAILED(hr))

{

printf("\nCannot create name value pair: %x", hr);

pNamedValueQueries->Release();

pEventTrigger->Release();

return S\_FALSE;

}

}

pNamedValueQueries->Release();

pEventTrigger->Release();

return S\_OK;

}

void RegisterFirewallTrigger()

{

HRESULT hr = CoInitializeEx(NULL, COINIT\_MULTITHREADED);

if (FAILED(hr))

{

printf("\nCoInitializeEx failed: %x", hr);

return;

}

hr = CoInitializeSecurity(

NULL,

-1,

NULL,

NULL,

RPC\_C\_AUTHN\_LEVEL\_PKT,

RPC\_C\_IMP\_LEVEL\_IMPERSONATE,

NULL,

0,

NULL);

if (FAILED(hr))

{

printf("\nCoInitializeSecurity failed: %x", hr);

CoUninitialize();

return;

}

LPCWSTR wszTaskName = L"Event\_Firewall\_Trigger";

ITaskService \*pService = NULL;

hr = CoCreateInstance(CLSID\_TaskScheduler,

NULL,

CLSCTX\_INPROC\_SERVER,

IID\_ITaskService,

(void\*\*)&pService);

if (FAILED(hr))

{

printf("Failed to create an instance of ITaskService: %x", hr);

CoUninitialize();

return;

}

hr = pService->Connect(\_variant\_t(), \_variant\_t(),

\_variant\_t(), \_variant\_t());

if (FAILED(hr))

{

printf("ITaskService::Connect failed: %x", hr);

pService->Release();

CoUninitialize();

return;

}

ITaskFolder \*pRootFolder = NULL;

hr = pService->GetFolder(\_bstr\_t(L"\\test"), &pRootFolder);

if (FAILED(hr))

{

printf("Cannot get Root Folder pointer: %x", hr);

pService->Release();

CoUninitialize();

return;

}

pRootFolder->DeleteTask(\_bstr\_t(L"Event\_Firewall\_Trigger"), 0);

ITaskDefinition \*pTask = NULL;

hr = pService->NewTask(0, &pTask);

pService->Release();

if (FAILED(hr))

{

printf("Failed to create a task definition: %x", hr);

pRootFolder->Release();

CoUninitialize();

return;

}

IRegistrationInfo \*pRegInfo = NULL;

hr = pTask->get\_RegistrationInfo(&pRegInfo);

if (FAILED(hr))

{

printf("\nCannot get identification pointer: %x", hr);

pRootFolder->Release();

pTask->Release();

CoUninitialize();

return;

}

hr = pRegInfo->put\_Author(L"Alexey Palonyj");

pRegInfo->Release();

if (FAILED(hr))

{

printf("\nCannot put identification info: %x", hr);

pRootFolder->Release();

pTask->Release();

CoUninitialize();

return;

}

ITaskSettings \*pSettings = NULL;

hr = pTask->get\_Settings(&pSettings);

if (FAILED(hr))

{

printf("\nCannot get settings pointer: %x", hr);

pRootFolder->Release();

pTask->Release();

CoUninitialize();

return;

}

hr = pSettings->put\_StartWhenAvailable(VARIANT\_TRUE);

pSettings->Release();

if (FAILED(hr))

{

printf("\nCannot put setting info: %x", hr);

pRootFolder->Release();

pTask->Release();

CoUninitialize();

return;

}

ITriggerCollection \*pTriggerCollection = NULL;

hr = pTask->get\_Triggers(&pTriggerCollection);

if (FAILED(hr))

{

printf("\nCannot get trigger collection: %x", hr);

pRootFolder->Release();

pTask->Release();

CoUninitialize();

return;

}

hr = AddTrigger(pTriggerCollection, L"FirewallTrigger",

L"<QueryList><Query Id='0'><Select Path='Microsoft-Windows-Windows Firewall With Advanced Security/Firewall'>"

"\*[System[Provider[@Name='Microsoft-Windows-Windows Firewall With Advanced Security'] and EventID=2003]]"

"</Select></Query></QueryList>", 10,

L"eventTimeCreated", L"Event/System/TimeCreated/@SystemTime",

L"eventDataSourceAddress", L"Event/EventData/Data[@Name='SourceAddress']",

L"eventDataProfiles", L"Event/EventData/Data[@Name='Profiles']",

L"eventDataSettingType", L"Event/EventData/Data[@Name='SettingType']",

L"eventDataSettingValueSize", L"Event/EventData/Data[@Name='SettingValueSize']",

L"eventDataSettingValue", L"Event/EventData/Data[@Name='SettingValue']",

L"eventDataSettingValueString", L"Event/EventData/Data[@Name='SettingValueString']",

L"eventDataOrigin", L"Event/EventData/Data[@Name='Origin']",

L"eventDataModifyingUser", L"Event/EventData/Data[@Name='ModifyingUser']",

L"eventDataModifyingApplication", L"Event/EventData/Data[@Name='ModifyingApplication']");

pTriggerCollection->Release();

if (FAILED(hr))

{

pRootFolder->Release();

pTask->Release();

CoUninitialize();

return;

}

IActionCollection \*pActionCollection = NULL;

hr = pTask->get\_Actions(&pActionCollection);

if (FAILED(hr))

{

printf("\nCannot get Task collection pointer: %x", hr);

pRootFolder->Release();

pTask->Release();

CoUninitialize();

return;

}

IAction \*pAction = NULL;

hr = pActionCollection->Create(TASK\_ACTION\_EXEC, &pAction);

pActionCollection->Release();

if (FAILED(hr))

{

printf("\nCannot create the action: %x", hr);

pRootFolder->Release();

pTask->Release();

CoUninitialize();

return;

}

IExecAction \*pExecAction = NULL;

hr = pAction->QueryInterface(

IID\_IExecAction, (void\*\*)&pExecAction);

pAction->Release();

if (FAILED(hr))

{

printf("\nQueryInterface call failed on IExecAction: %x", hr);

pRootFolder->Release();

pTask->Release();

CoUninitialize();

return;

}

hr = pExecAction->put\_Path(\_bstr\_t(L"C:\\Users\\sysli\\Documents\\study\_files\\SMIT\\3\\scripts\\FirewallDataChanging.bat"));

if (FAILED(hr))

{

printf("\nCannot add path for executable action: %x", hr);

pRootFolder->Release();

pTask->Release();

CoUninitialize();

return;

}

hr = pExecAction->put\_Arguments(\_bstr\_t(L"\"$(eventDataProfiles)\" "

L"\"$(eventDataSettingType)\" "

L"\"$(eventDataSettingValueSize)\" "

L"\"$(eventDataSettingValue)\" "

L"\"$(eventDataSettingValueString)\" "

L"\"$(eventDataOrigin)\" "

L"\"$(eventDataModifyingUser)\" "

L"\"$(eventDataModifyingApplication)\" "

L"\"$(eventTimeCreated)\""

));

if (FAILED(hr))

{

printf("\nCannot add path for executable action: %x", hr);

pRootFolder->Release();

pTask->Release();

CoUninitialize();

return;

}

pExecAction->Release();

CREDUI\_INFO cui;

TCHAR pszName[CREDUI\_MAX\_USERNAME\_LENGTH] = L"";

TCHAR pszPwd[CREDUI\_MAX\_PASSWORD\_LENGTH] = L"";

BOOL fSave;

DWORD dwErr;

cui.cbSize = sizeof(CREDUI\_INFO);

cui.hwndParent = NULL;

cui.pszMessageText = TEXT("Account information for task registration:");

cui.pszCaptionText = TEXT("Enter Account Information for Task Registration");

cui.hbmBanner = NULL;

fSave = FALSE;

dwErr = CredUIPromptForCredentials(

&cui,

TEXT(""),

NULL,

0,

pszName,

CREDUI\_MAX\_USERNAME\_LENGTH,

pszPwd,

CREDUI\_MAX\_PASSWORD\_LENGTH,

&fSave,

CREDUI\_FLAGS\_GENERIC\_CREDENTIALS |

CREDUI\_FLAGS\_ALWAYS\_SHOW\_UI |

CREDUI\_FLAGS\_DO\_NOT\_PERSIST);

if (dwErr)

{

CoUninitialize();

return;

}

IRegisteredTask \*pRegisteredTask = NULL;

hr = pRootFolder->RegisterTaskDefinition(

\_bstr\_t(wszTaskName),

pTask,

TASK\_CREATE\_OR\_UPDATE,

\_variant\_t(\_bstr\_t(pszName)),

\_variant\_t(\_bstr\_t(pszPwd)),

TASK\_LOGON\_INTERACTIVE\_TOKEN,

\_variant\_t(L""),

&pRegisteredTask);

if (FAILED(hr))

{

printf("\nError saving the Task : %x", hr);

pRootFolder->Release();

pTask->Release();

SecureZeroMemory(pszName, sizeof(pszName));

SecureZeroMemory(pszPwd, sizeof(pszPwd));

CoUninitialize();

return;

}

printf("\n Success! Task succesfully registered. ");

pRootFolder->Release();

pTask->Release();

pRegisteredTask->Release();

SecureZeroMemory(pszName, sizeof(pszName));

SecureZeroMemory(pszPwd, sizeof(pszPwd));

CoUninitialize();

return;

}