КОМП'ЮТЕРНІ ЗАСОБИ, МЕРЕЖІ ТА СИСТЕМИ

A. Kriachok, A. Shevchuk

APPLICATION OF DLP-SYSTEMS FOR DATA PROTECTION IN CORPORATION NETWORKS

It is analyzed the possibility of using the DLP-system architecture for data protection in corporation networks. The program for system events monitoring was developied.
Key words: DLP-system, data protect, program's system.

Проаналізовано можливість використання архітектури DLP-системи для захисту даних у корпоративних мережах. Розроблено програму для виконання моніторингу системних подій у додатках, визначених політиками інформаційної безпеки.
Ключові слова: DLP-система, захист даних, програмна система.

Проанализирована возможность использования архитектуры DLP-системы для защиты данных в корпоративных сетях. Разработана программа для выполнения мониторинга системных событий в приложениях, определенных политиками информационной безопасности.
Ключевые слова: DLP-система, защита данных, программная система.

УДК 004.4: 381.3

А.С. КРЯЧОК, А.В. ШЕВЧУК

О ПРИМЕНЕНИИ DLP-СИСТЕМ ДЛЯ ЗАЩИТЫ ДАННЫХ В КОРПОРАТИВНЫХ СЕТЯХ

Введение. Широкое применение вычислительных сетей увеличивает важность задачи защиты данных, поскольку существует вероятность несанкционированного использования или модификации информации. На всех этапах жизненного цикла информационной системы данные подвергаются определенным воздействиям. Основными причинами случайных воздействий на систему могут быть отказы и сбои аппаратуры, ошибки в программном обеспечении, ошибки в работе персонала, помехи в линиях связи и ряд других.

Несмотря на то, что современные операционные системы для персональных компьютеров имеют собственные подсистемы защиты [1], они не всегда способны в полной мере обеспечить защиту данных от внутренних нарушителей (работников, посетителей или конкурентов).

Современные программные системы защиты данных предназначены для блокировки портов и контроля доступа к защищенным данным. Основным назначением систем блокировки портов [2] является разграничение прав доступа пользователей к внешним и внутренним устройствам компьютеров в масштабах организации, а также контроль всех данных, записываемых на устройства. *IRM*-системы [3] могут применяться для контроля доступа пользователей к защищенным файлам. Эти системы работают по следующей схеме: защищаемый файл, помещается в специальный зашифрованный контейнер, в котором также сохраняются права доступа к этому файлу.

Для доступа к содержимому контейнера, пользователь должен пройти аутентификацию на *IRM*-сервере, получить ключи для расшифровки и иметь достаточные для доступа права.

Постановка задачи. Приведенные системы имеют определенные ограничения: невозможность контроля перемещения файлов, необходимость помещения информации в незащищенные контейнеры, требование отсутствия незашифрованных файлов на диске, участие пользователя в работе системы. Поэтому – актуальна задача создания систем защиты конфиденциальных данных от несанкционированной передачи и использования. Такие системы должны выполнять мониторинг системных событий, анализировать используемые данные, на предмет их конфиденциальности и, при некоторых условиях, выполнять действия определенные в политиках информационной безопасности.

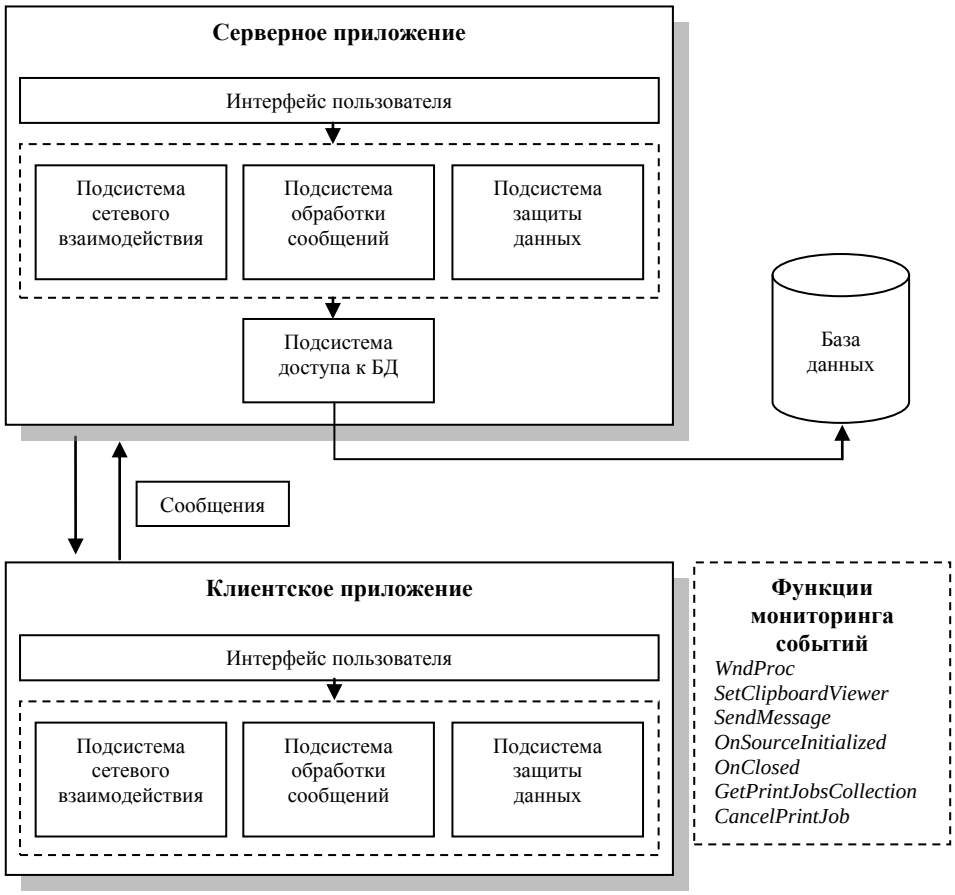

РИС. 1. Архитектура *DLP*-системы

Одним из инструментов защиты данных от несанкционированного использования и модификации являются системы предотвращения утечки данных, известные как *DLP*-системы [4], которые сочетают в себе технологии для защиты информации на протяжении ее жизненного цикла и минимальное влияние на работу пользователей. Основными функциями *DLP*-системы являются: централизованное управление, настройка политик безопасности, блокирование несанкционированных перемещений данных, создание цифровых отпечатков документов [5], формирование отчетов о работе системы.

Как правило, *DLP*-системы имеют клиент-серверную архитектуру (рис. 1) и выполняют следующие функции:

– создание политик информационной безопасности;
– поиск уязвимых данных на компьютерах пользователей;
– отслеживание данных в использовании и генерирование предупреждающих сообщений о нарушении политик безопасности;
– карантин сообщений электронной почты, отслеживание системных событий и блокировка перемещения файлов.

DLP-системы имеют и некоторые недостатки. Например, когда данные выходят за пределы корпоративной сети по локальным каналам (*USB, CD/DVD*, принтеры), то информация не попадает на уровень сетевого шлюза. Как следствие, приходится выполнять анализ данных на компьютере пользователя, что может привести к проблемам с эффективностью фильтрации и/или повышением нагрузки.

Реализация задачи мониторинга системных событий. Рассматриваемая в работе программная система построена на базе архитектуры *DLP*-системы. Покажем более подробно реализацию функций мониторинга [6] системных событий операционной системы *Windows* на примере платформы *.NET Framework* и языка программирования C#.

Для отслеживания операций буфера обмена (*Cut, Copy, Print Screen* и др.) необходимо перегрузить оконную процедуру приложения

```
private IntPtr WndProc(IntPtr hwnd, int msg, IntPtr wParam, IntPtr lParam, ref bool handled)
{
return IntPtr.Zero;
}
```

и создать окно просмотра буфера обмена, которое будет отражать текущее содержимое буфера и принимать сообщения при изменении его содержимого. Для создания окна просмотра буфера обмена приложение должно выполнить следующие операции:

- добавить окно к цепочке окон просмотра буфера обмена;
- обработать сообщение *WM_CHANGECBCHAIN*;
- обработать сообщение *WM_DRAWCLIPBOARD*;
- удалить окно из цепочки окон просмотра буфера обмена.

Для добавления окна в цепочку окон просмотра буфера обмена необходимо, чтобы окно добавило себя к цепочке окон просмотра буфера обмена, вызывая функцию *SetClipboardViewer* импортируемую из библиотеки *User*32.*dll*:

```
[DllImport("User32.dll")]
private static extern int SetClipboardViewer(int hWndNewViewer);
```

Значение, которое возвращает данная функция, является дескриптором следующего окна в цепочке. Окно должно следить за этим значением, например, с помощью сохранения его в переменной *m_nextClipboardViewer*. На следующем шаге необходимо перегрузить функцию *OnSourceInitialized*, которая вызывается при инициализации окна приложения, и, вызвав функцию *SetClipboardViewer*, добавить обработчик событий с помощью функции *AddHook*, который будет получать все сообщения. В результате, получим следующий код:

```
protected override void OnSourceInitialized(EventArgs e)
{
base.OnSourceInitialized(e);
m_nextClipboardViewer = (IntPtr)SetClipboardViewer((int)this.WindowHandle);
m_hwndSource = PresentationSource.FromVisual(this) as HwndSource;
m_hwndSource.AddHook(WndProc);
}
```

Блок обработки сообщения *WM_CHANGECBCHAIN* может быть реализован следующим образом. Окно просмотра буфера обмена принимает сообщение *WM_CHANGECBCHAIN* в то время, когда другое окно удаляется из цепочки окон просмотра буфера. Иными словами, это сообщение должно быть передано в следующее окно в цепочке, с помощью функции *SendMessage*, импортируемой из библиотеки *User*32.*dll*. Реализация этого блока может иметь такой вид:

```
[DllImport("User32.dll", CharSet = CharSet.Auto)]
private static extern int SendMessage(IntPtr hwnd, int wMsg, IntPtr wParam,
IntPtr lParam);
case WM_CHANGECBCHAIN:
{
if (wParam == m_nextClipboardViewer)
{
m_nextClipboardViewer = lParam;
}
else
{
SendMessage(m_nextClipboardViewer, msg, wParam, lParam);
}
break;
}
```

Обработка сообщения *WM_DRAWCLIPBOARD* предупреждает окно просмотра буфера обмена об изменении содержимого буфера. При обработке данного сообщения необходимо определить тип операции буфера обмена и отправить сообщение следующему окну в цепочке окон просмотра буфера.

Для удаления окна из цепочки окон просмотра буфера обмена, необходимо вызвать функцию *ChangeClipboardChain*, импортируемую из библиотеки *User*32*.dll*. Выполнение данной части кода требует перегрузки функции *OnClosed*, как показано далее:

```
protected override void OnClosed(EventArgs e)
{
base.OnClosed(e);
ChangeClipboardChain(this.WindowHandle, m_nextClipboardViewer);
if (m_hwndSource != null)
{
m_hwndSource.RemoveHook(WndProc);
}
}
```

После создания окна просмотра буфера обмена у приложения появляется возможность отслеживать операции с буфером и блокировать их, вызывая статический метод *Clear* класса *Clipboard*.

Для блокировки возможности печати документов приложение должно отслеживать очередь принтера и отменять ее. Реализовать эти функции можно с помощью технологии *WMI*, являющейся одной из базовых технологий для централизованного управления и отслеживания работы различных частей компьютерной инфраструктуры под управлением платформы *Windows*. Важной особенностью *WMI* является то, что объекты, которые хранятся в нем, соответствуют динамическим ресурсам, поэтому параметры таких объектов не сохраняются постоянно, а создаются по запросу потребителя данных. Хранилище свойств объектов *WMI* называется репозиторием и расположено в системной папке операционной системы *Windows*:

*%SystemRoot%\System*32*\WBEM\Repository\FS*

Поскольку *WMI* построен по объектно-ориентированному принципу, то все данные операционной системы представлены в виде объектов, их свойств и методов. Для обращения к объектам *WMI* используется специфический язык запросов *WQL*, являющийся одной из разновидностей *SQL*. Основное его отличие от *ANSI SQL* состоит в невозможности изменения данных, т. е. посредством *WQL* возможна только выборка данных с помощью команды *SELECT*.

Получить доступ к очереди принтера можно через *WMI* класс *Win*32*_PrintJob*, представляющий работу принтера. После выполнения запроса к классу *Win*32*_PrintJob*, получаем список работ принтера, происходящих в настоящее время. Пример функции *GetPrintJobsCollection*, получающей список работ принтера, приведен далее:

```
public StringCollection GetPrintJobsCollection(string printerName)
{
StringCollection printJobCollection = new StringCollection();
string searchQuery = "SELECT * FROM Win32_PrintJob";
```

```
ManagementObjectSearcher searchPrintJobs = new ManagementObject-
Searcher(searchQuery);
ManagementObjectCollection prntJobCollection = searchPrintJobs.Get();
foreach(ManagementObject prntJob in prntJobCollection)
{
string jobName = prntJob.Properties["Name"].Value.ToString();
char[] splitArr = new char[] {','};
string jobPrinterName = jobName.Split(splitArr)[0];
string documentName = prntJob.Properties["Document"].Value.ToString();
if (jobPrinterName == printerName)
{
printJobCollection.Add(documentName);
}
}
return printJobCollection;
}
```

Отмена работы принтера осуществляется путем вызова метода *Delete* для объекта из списка текущих работ. Пример функции *CancelPrintJob*, отменяющей работу принтера, показан далее:

```
public bool CancelPrintJob(string printerName, int printJobId)
{
bool isActionPerformed = false;
string searchQuery = "SELECT * FROM Win32_PrintJob";
ManagementObjectSearcher searchPrintJobs = new
ManagementObjectSearcher(searchQuery);
ManagementObjectCollection prntJobCollection = searchPrintJobs.Get();
foreach(ManagementObject prntJob in prntJobCollection)
{
string jobName = prntJob.Properties["Name"].Value.ToString();
char[] splitArr = new char[] {','};
string jobPrinterName = jobName.Split(splitArr)[0];
int jobId = Convert.ToInt32(jobName.Split(splitArr)[1]);
string documentName = prntJob.Properties["Document"].Value.ToString();
if (jobPrinterName == printerName)
{
prntJob.Delete();
isActionPerformed = true;
break;
}
}
return isActionPerformed;
}
```

Описание программного продукта. На основе выбранной архитектуры (рис. 1) и предложенного программного алгоритма разработана клиент-серверная *DLP*-система «*DLP security package*» [7], сочетающая в себе набор технологий предотвращения утечки данных по локальным и сетевым каналам.

В серверном приложении формируются политики безопасности, в которых определяется имя исполняемого файла приложения и действия, применяемых при выполнении операций с буфером обмена. Клиентское приложение работает в фоновом режиме, выполняя мониторинг системных событий. При возникновении определенного события, информация о нем передается серверному приложению, которое, согласно политикам безопасности, принимает решение относительно действий по данному событию.

Результаты работы программного продукта. С использованием разработанного программного обеспечения был проведен ряд экспериментов. На рис. 2 показана экранная форма с результатами работы программы «*DLP security package*» на примере приложения *MS Word* 2007, для которого в политиках безопасности определено блокирование операции копирования данных в буфер обмена.

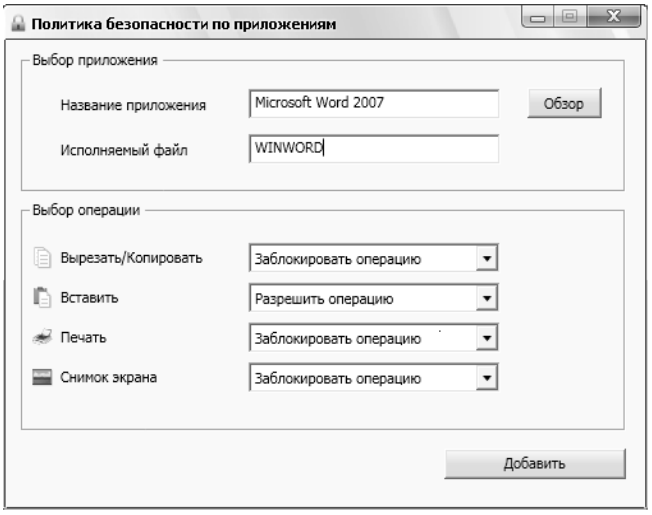

РИС. 2. Создание политики безопасности для приложения *MS Word* 2007

При попытке пользователя выполнить операцию копирования текста, данное событие будет отслежено (выполняется обработка сообщения *WM_DRAWCLIPBOARD* в оконной процедуре *WndProc*) и заблокировано (с помощью вызова статического метода *Clear* класса *Clipboard*), а на экране появится окно с сообщением о блокировании данной операции согласно политике безопасности. На рис. 3 показана реакция системы на данную операцию.

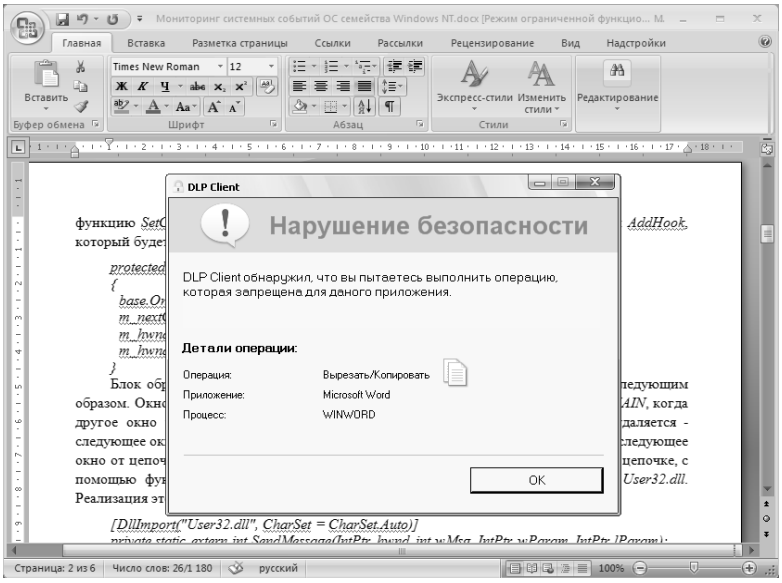

РИС. 3. Сообщение о блокировании операции копирования текста

Результаты работы предлагаемой программы на примере приложения *Notepad*, для которого в политиках безопасности определено блокирование операции печати документов, показаны на рис. 4.

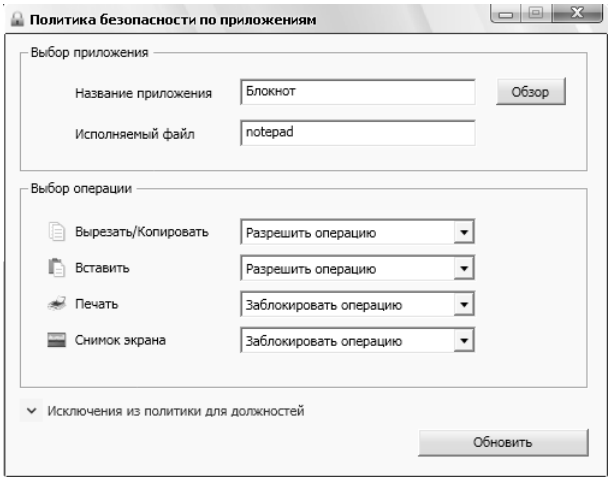

РИС. 4. Создание политики безопасности для приложения *Notepad*

На рис. 5 показано диалоговое окно с сообщением о блокировании печати документа, согласно политике безопасности, при попытке пользователя выполнить данную операцию.

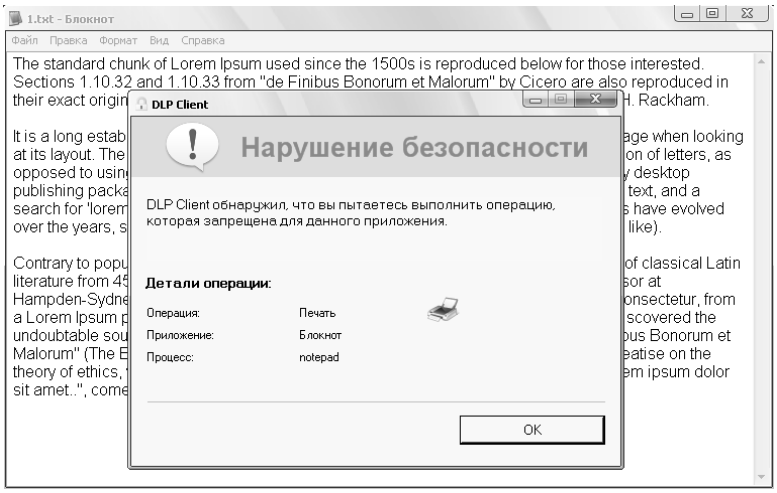

РИС. 5. Сообщение о блокировании операции печати документа

Выводы. Таким образом, анализ известных систем контроля данных показал определенные ограничения в их применении. Учитывая это, предложено при проектировании систем защиты конфиденциальных данных в корпоративных сетях использовать современную архитектуру *DLP*-системы. На базе этой архитектуры и разработанного оригинального алгоритма создана новая программа «*DLP security package*» для выполнения мониторинга системных событий *Windows* в приложениях, определенных политиками информационной безопасности. Проведенные эксперименты показали высокую эффективность (широкий круг решаемых задач и низкая ресурсоемкость) данного подхода при решении задач ограничения несанкционированного использования и модификации данных.

1. *Леонтьев Б.К.* Microsoft Windows XP SP3. Скрытые возможности. – М.: НТ пресс, 2006. – 240 с.
2. *Система* защиты портов, http://www.securit.ru/products/info/zlock/purpose
3. *Зенкин Д.* DLP и IRM: конкуренты или союзники? http://www.it-world.ru/upload/iblock/7e8/55188.pdf
4. *Martin L.* Understanding DLP, http://www.infosectoday.com/Articles/DLP/Understanding_DLP.htm
5. *Schleimer S.* Winnowing: Local Algorithms for Document Fingerprinting. – 8 p.
6. *Румянцев П.* Азбука программирования в Win 32 API. – М.: Радио и связь, 2000. – 257 с.
7. *Крячок О.С., Шевчук О.В.* Комп'ютерна програма «DLP security package», Свідоцтво про реєстрацію авторського права на твір № 38296 // Міністерство освіти і науки України. Державна служба інтелектуальної власності; 11.05.2011.

Получено 12.07.2011