УДК 004.031

5.2.2. Математические, статистические и инструментальные методы экономики (физико-математические науки, экономические науки)

ПРОГРАММНЫЙ ИНТЕРФЕЙС МЕТОДОМ ПРЯМОГО ДОСТУПА К УСТРОЙСТВАМ ВВОДА RAW INPUT API

Минин Н. А.
студент факультета прикладной информатики

Минина Е. А.
доцент кафедры системного анализа и обработка информации

В статье рассмотрено применение интерфейса прямого доступа к устройствам ввода Raw Input API и реализация программного обеспечения для регистрации ввода с клавиатуры на его основе

Ключевые слова: ИНТЕРФЕЙС, ПРОГРАММА, ОПЕРАЦИОННАЯ СИСТЕМА, ПРЯМОЙ ДОСТУП, УСТРОЙСТВА, ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ

http://dx.doi.org/10.21515/1990-4665-201-039

UDC 004.031

5.2.2. Mathematical, statistical and instrumental methods of economics (physical and mathematical sciences, economic sciences)

THE PROGRAM INTERFACE IS A METHOD OF DIRECT ACCESS TO INPUT DEVICES RAW INPUT API

Minin N. A.
student of the Faculty of Applied Informatics

Minina E. A.
Associate Professor of the Department of System Analysis and Information Processing

The article considers the application of the interface for direct access to Raw Input API input devices and the implementation of software for registering keyboard input based on it

Keywords: INTERFACE, PROGRAM, OPERATING SYSTEM, DIRECT ACCESS, DEVICES, INFORMATION SECURITY

Программные средства для регистрации ввода с клавиатуры можно разделить на две подгруппы – работающие в режиме пользователя и работающие в режиме ядра. Те, которые работают в режиме ядра, имеют свои преимущества и недостатки: они сложно обнаруживаются программными средствами, однако их работа связана с созданием драйвера. Это вносит некоторые особенности в процесс развёртывания такого программного обеспечения на целевой системе: для установки драйвера нужны права администратора, в новых версиях операционной системы драйвер должен пройти сертификацию. Кроме того, создание драйвера требует высокой квалификации программиста, также они легко обходятся экранными клавиатурами. В целом, процесс создания программного обеспечения для регистрации ввода, работающего в режиме ядра, выходит за рамки этой статьи. В этой статье внимание уделено

созданию программного обеспечения, работающего в пользовательском режиме.

В исследовании рассмотрены некоторые варианты реализации программных закладок (кейлоггеров), работающих в режиме пользователя. Программная закладка – это внесенные в программное обеспечение функциональные объекты, которые при определенных условиях (входных данных) инициируют выполнение не описанных в документации функций, позволяющих осуществлять несанкционированные воздействия на информацию [4]. Присутствие программных закладок в информационных системах представляет собой серьезную потенциальную опасность. Внедрение и функционирование программной закладки в компьютере носит латентный характер [1].

Самым простым и часто используемым можно считать вариант реализации, устанавливающий ловушку (*хук, hook*), на сообщения от клавиатуры для всех потоков системы. Суть метода заключается в том, что кейлоггер устанавливает фильтрующую функцию-ловушку, посредством вызова функции SetWindowsHookEx(). Сама функция ловушка, как правило, находится в отдельной динамически загружаемой библиотеке. При выборке сообщений от клавиатуры из очереди сообщений потока, система вызовет установленную функцию-ловушку. К достоинствам такого метода стоит отнести простоту реализации. Но он обладает и массой недостатков. Во первых, по причине внедрения во все процессы, ожидающие сообщений от клавиатуры, его легко обнаружить. Во вторых, такой метод не сработает для всех процессов системы, если кейлоггер запущен от имени пользователя, обладающего малыми привилегиями. В третьих, при неправильной или не качественной реализации функции-ловушки, кейлоггер может создавать ощутимые для пользователя задержки при вводе. В четвёртых, перехват данных о вводе для

приложений, использующих более низкоуровневые методы, будет невозможен.

Вторым по популярности методом реализации кейлоггеров, является циклический опрос состояния клавиатуры посредством функций GetAsyncKeyState() либо GetKeyState(), возвращающих массив, описывающий состояние клавиатуры. Анализируя массив, можно понять, какие клавиши были нажаты. Это предельно простой метод реализации, не требующий создания функции-ловушки и загружаемой динамической библиотеки. Достаточно лишь опрашивать клавиатуру с определённой частотой. Однако у этого метода есть серьёзные недостатки.␤Во-первых, в связи с тем, что опрос производится с определённой периодичностью, возможны пропуски. Во-вторых – велика возможность обнаружения программы. Достаточно производить мониторинг процессов, опрашивающих состояние клавиатуры с большой частотой. В третьих, такой метод может создавать большую нагрузку на процессор персонального компьютера, если программа производит опрос клавиатуры со слишком большой частотой.

Третьим методом реализации программного обеспечения для мониторинга ввода с клавиатуры, является метод внедрения в процесс и перехвата функций обработки сообщений GetMessage() и PeekMessage() из динамически загружаемой библиотеки user32.dll. Чаще всего применяется метод внедрения, называемый сплайсингом, либо подмена адресов функций в таблице импорта (IAT), перехват функции GetProcAddress(). Сам кейлоггер может реализовываться в виде динамически загружаемой библиотеки или посредством непосредственного внедрения кода в процесс. При вызове функции для извлечения сообщения из очереди сообщений, этот вызов на самом деле переходит к коду функции-перехватчика. Если сообщение является сообщением от клавиатуры, информация о сообщении извлекается из параметров сообщения, должным

образом обрабатывается и протоколируется кейлоггером. Это достаточно эффективный метод реализации, в виду относительной сложности реализации не часто применяющийся, и из-за этого и не так легко обнаруживающийся. От него не спасают и экранные клавиатуры. Как правило, для развёртывания такого кейлоггера могут потребоваться права администратора на системе, в виду внедрения кода во все процессы, ожидающие ввода с клавиатуры, что является недостатком. Просто заметить, что этот метод наследует много недостатков первого метода в виду близости двух этих методов: лёгкость обнаружения, невозможность перехвата во всех приложениях.

Ещё одним методом является использование модели прямого ввода (Raw Input API), который является предметом данного исследования. Единственным очевидным недостатком является, присущая всем методам реализации кейлоггеров пользовательского режима, относительная лёгкость обнаружения, так как приложение должно указать свою заинтересованность в сообщениях WM_INPUT – по умолчанию приложения не получают этого сообщения. Однако на практике, этот метод реализации пока не является достаточно распространённым, поэтому далеко не все средства обнаружения такого рода программного обеспечения способны обнаружить такой кейлоггер. Так же развёртывание такого кейлоггера не требует полномочий администратора на целевой системе, не требует создания кода, внедряющегося в другие процессы, такой кейлоггер не производит большой нагрузки на систему и не обходится экранными клавиатурами. В свете вышесказанного, он является предпочтительным методом реализации кейлоггера пользовательского режима.

Современные операционные системы корпорации Microsoft серии Windows обладают достаточно широким набором программных интерфейсов для доступа к устройствам ввода. Среди них изначально

родной для ОС Windows Windows Messaging, часть DirectX – DirectInput и, появившийся в Windows XP Raw Input API – программный интерфейс прямого доступа к устройствам ввода, который представляет наибольший интерес в вопросах информационной безопасности [2].

Программный интерфейс прямого доступа к устройствам ввода среди всех вышеперечисленных является самым низкоуровневым. Это позволяет ему иметь некоторые преимущества по сравнению с другими способами доступа к устройствам ввода, теряя при этом ряд удобств в работе. Другие способы доступа к устройствам ввода так или иначе основываются на нём, однако его низкоуровневая природа придаёт ему большую гибкость.

Raw Input API позволяет, например, обрабатывать в своём приложении несколько клавиатур и мышей как различные устройства. Очевидно, что интерфейс прямого доступа составляет конкуренцию DirectInput в среде разработчиков игровых и мультимедийных приложений, но в данном исследовании нас этот вопрос не интересует. Нас интересует то, что этот интерфейс, работает надёжно и достаточно слабо блокируется другими приложениями. Если в программе в каком-либо виде используется регистрация ввода с клавиатуры (модель Windows Messaging), то, после запуска приложения, использующего способ ввода DirectInput, наша программа перестанет регистрировать ввод, поступающий в это приложение. С программным интерфейсом прямого доступа Raw Input API такого не произойдёт в следствие его более низкоуровневой природы.

Рассмотрим модель работы программного интерфейса прямого доступа к устройствам ввода подробнее.

Ранее мышь и клавиатура генерировали данные о вводе. Операционная система скрывала данные непосредственно поступающие от устройства. Например, клавиатура при нажатии клавиши генерирует

специфичный для устройства скан-код клавиши, но приложение получало вместо него виртуальный код клавиши, стандартизированный в рамках операционной системы. Скрывая специфичную для устройства информацию, операционная система не давала возможности приложениям поддерживать новые типы устройств ввода и все их возможности. В такой ситуации приложению, для получения ввода с неподдерживаемых устройств, приходилось выполнять много лишней работы – открывать устройство, периодически опрашивать его, очень часто требовалось реализовать специализированный драйвер [3]. Интерфейс прямого доступа к устройствам ввода предоставляет программный интерфейс для простого доступа к низкоуровневым данным от устройств, включая клавиатуру и мышь.

Модель программного интерфейса прямого доступа к устройствам ввода отличается от изначальной модели доступа к данным от устройств ввода в Windows. В изначальной модели приложение получало независимую от конкретного устройства ввода информацию с помощью таких сообщений, как WM_CHAR, WM_MOUSEMOVE и WM_APPCOMMAND. Для получения прямого доступа к устройству, приложение должно зарегистрировать устройства, от которых оно хочет получать информацию о вводе. Так же, приложение, которое получает информацию от устройств ввода напрямую, должно обрабатывать сообщение WM_INPUT.

Программный интерфейс прямого доступа к устройствам ввода имеет некоторые важные преимущества:

1) приложение не должно получать доступ или обнаруживать устройство;

2) приложение получает информацию напрямую от устройства и обрабатывает полученные данные как ему угодно;

3) приложение может различить конкретное устройство ввода, даже если оно получает ввод от нескольких устройств одного типа; например, обработать по-разному две одновременно подключенные компьютерные мыши;

4) приложение контролирует поступление данных от конкретного типа устройств или конкретного устройства;

5) новые устройства ввода могут без проблем использоваться сразу после своего появления, не дожидаясь пока ввод от них будет стандартизирован в сообщениях операционной системы.

Для получения прямого доступа к устройствам ввода приложение должно зарегистрировать устройства, от которых оно хочет получать информацию [3].

Для регистрации устройств, приложение должно создать массив из структур RAWINPUTDEVICE. Таким образом приложение укажет типы устройств, от которых оно хочет получать данные о вводе. Приложение должно вызвать функцию RegisterRawInputDevices(), чтобы зарегистрировать устройства, от которых оно хочет получать данные о вводе.

Следует отметить, что приложение может зарегистрировать устройство, которое не подключено к системе. Когда оно всё же будет подключено, приложение будет получать от него данные. Чтобы получить список устройств на конкретной системе, приложение должно вызвать функцию GetRawInputDeviceList(), после чего оно получит массив структур RAWINPUTDEVICELIST, в этой структуре есть дескриптор (hDevice), описывающий конкретное устройство. Используя полученный дескриптор и функцию GetRawInputDeviceInfo() можно получить информацию о конкретном устройстве ввода.

Используя член структуры RAWINPUTDEVICE dwFlags, приложение может указать, как именно оно хочет получать информацию от устройств ввода.

Приложение получает информацию от устройств ввода обрабатывая сообщение WM_INPUT. Приложение получает это сообщение, даже работая в фоновом режиме. Для нас это один из ключевых моментов.

Есть два метода получения информации от устройств ввода напрямую – не буферизированный (или стандартный) и буферизированный.

Не буферизированный подходит для устройств ввода, которые генерируют не много данных о вводе (например, клавиатура). Приложение может получить не более одной структуры RAWINPUT за раз. При обработке сообщения WM_INPUT приложение должно вызывать функцию GetRawInputData() используя дескриптор на структуру RAWINPUT полученный из сообщения. Именно этот способ нас и интересует.

При буферизированном вводе приложение получает массив структур RAWINPUT за раз. Приложение должно вызвать функцию GetRawInputBuffer() для получения массива структур. Макрос NEXTRAWINPUTBLOCK должен использоваться для обхода массива.

Для обработки информации нужна подробная информация об устройстве ввода. Приложение может получить подробную информацию используя функцию GetRawInputDeviceInfo() и дескриптор устройства. Дескриптор может быть получен из сообщения WM_INPUT либо из члена структуры RAWINPUTHEADER hDevice.

В заключение следует отметить, что после краткого анализа функционировании кейлоггеров, необходимо применение их на практике, что будет следующим шагом исследования с использованием программного интерфейса прямого доступа к устройствам ввода (Raw Input API), преимущества которого рассмотрены выше.

Литература

1. Баранов А. И. Программные интерфейсы для работы с устройствами ввода: Учеб. пособие / А. И. Баранов. – Москва: Техника, 2018. – 320 с.

2. Дежинин П. Н., Хорошилов А. В., Тележников В. Ю. Формирование методологии разработки безопасного системного программного обеспечения на примере операционных систем // Тр. ИСП РАН. 2021. Т. 33. № 5. С. 25-40.

3. Караев А. В. Актуальность и особенности внедрения ИТ-сервисов с применением облачных технологий / А. В. Караев, Д. О. Емельянов, Т. П. Барановская // Информационное общество: современное состояние и перспективы развития : сборник материалов XIII международного форума, Краснодар, 13–18 июля 2020 года. – Краснодар: Кубанский государственный аграрный университет имени И.Т. Трубилина, 2020. – С. 387-390.

4. Петров А. А. Анализ основных технических характеристик систем активной защиты в сетях общего пользования / А. А. Петров, Е. А. Минина // Современная экономика: проблемы и решения. – 2022. – № 10(154). – С. 34-46.

5. Mathematical modeling of corporate network tolerance troubleshooting methods / A. A. Petrov, D. N. Savinskaya, E. A. Minina, L. K. Dunskaya // Modern Economics: Problems and Solutions. – 2020. – No. 12(132). – P. 35-45.

References

1. Baranov A. I. Programmny`e interfejsy` dlya raboty` s ustrojstvami vvoda: Ucheb. posobie / A. I. Baranov. – Moskva: Texnika, 2018. – 320 s.

2. Dezhinin P. N., Xoroshilov A. V., Telezhnikov V. Yu. Formirovanie metodologii razrabotki bezopasnogo sistemnogo programmnogo obespecheniya na primere operacionny`x sistem // Tr. ISP RAN. 2021. T. 33. № 5. S. 25-40.

3. Karaev A. V. Aktual`nost` i osobennosti vnedreniya IT-servisov s primeneniem oblachny`x texnologij / A. V. Karaev, D. O. Emel`yanov, T. P. Baranovskaya // Informacionnoe obshhestvo: sovremennoe sostoyanie i perspektivy` razvitiya : sbornik materialov XIII mezhdunarodnogo foruma, Krasnodar, 13–18 iyulya 2020 goda. – Krasnodar: Kubanskij gosudarstvenny`j agrarny`j universitet imeni I.T. Trubilina, 2020. – S. 387-390.

4. Petrov A. A. Analiz osnovny`x texnicheskix xarakteristik sistem aktivnoj zashhity` v setyax obshhego pol`zovaniya / A. A. Petrov, E. A. Minina // Sovremennaya e`konomika: problemy` i resheniya. – 2022. – № 10(154). – S. 34-46.

5. Mathematical modeling of corporate network tolerance troubleshooting methods / A. A. Petrov, D. N. Savinskaya, E. A. Minina, L. K. Dunskaya // Modern Economics: Problems and Solutions. – 2020. – No. 12(132). – P. 35-45.