УДК 004

ТЕСТЕР ДЛЯ ОБУЧАЮЩИХСЯ: ВОЗМОЖНЫЕ ИЗЪЯНЫ В ЗАЩИТЕ И МЕРЫ ПО ИХ УСТРАНЕНИЮ

Бартосик Ф.М., Лимарева И.Г., Мурых Е.Л., Сайлаукызы Ж.С.
РГП «Карагандинский государственный технический университет», Караганда,
e-mail: bartoflex@yandex.ru, innalim21@mail.ru, murykh@mail.ru, s_k_zhuldiz@mail.ru

На примере программы-тестера для электронного тестирования рассмотрены наиболее возможные варианты взлома программы. Предложены соответствующие методы защиты, применение которых повышает достоверность результатов тестирования.

Ключевые слова: тестер; монопольный захват экрана тестером, Windows-функции, окна, варианты взлома, таймер, технология хуков, ассемблер, отладчик, ловушки

TESTER FOR TRAINED: POSSIBLE DEFECTS IN PROTECTION AND MEASURES ON THEIR ELIMINATION

Bartossik F.M., Limareva I.G. Murykh E.L., Sailaukyzy Z.S.
Karaganda State Technical University, Karaganda,
e-mail: bartoflex@yandex.ru, innalim21@mail.ru, murykh@mail.ru, s_k_zhuldiz@mail.ru

On an example of the program-tester for electronic testing the most possible variants of breaking of the program are considered. Corresponding methods of protection which application raises reliability of results of testing are offered.

Keywords: a tester, exclusive capture of the screen by a tester, Windows-functions, windows, variants of breaking, a timer, technology of hooks, the assembler, a debugger, breakpoints

В настоящее время сложно представить жизнь общества без использования электронно-вычислительных систем. Информационные технологии прочно вошли во все отрасли производства, науку, экономику. Такая же сфера общества как образование, на сегодняшний день совершенно не в состоянии обходиться без компьютера. Электронно-вычислительные машины применяются в учебном процессе при подготовке студентов практически на всех специальностях. В первую очередь – это проведение лекционных, семинарских и других видов занятий. Кроме того, в последние годы укрепилась практика оценки знаний у студентов с применением электронных средств. Программы-тестеры используются на экзаменах, при рубежном аттестационном контроле, и других срезах знаний. В большинстве случаев эти программы разрабатываются непосредственно IT-специалистами учебного заведения, и далеко не всегда являются надежными в плане безопасности. Для обеспечения объективности и достоверности результатов, полученных при тестировании, должна быть обеспечена реализация следующих мероприятий: защита ключей правильных ответов тестового материала от несанкционированного доступа к ним студентов; блокировка применения «электронных» шпаргалок; и в случае сетевого компьютерного тестирования – защита сервера от атак с целью подлога результатов тестирования.

Автоматизированные «электронные» шпаргалки

Термин «электронные» шпаргалки разделяется на два вида: банальное списывание с экрана монитора, и автоматизированная подстановка правильного ответа в окно тестера. В последнем случае очевидным является то, что у студентов на руках уже есть ключи правильных ответов, т.е. защита сохранения тестового материала была преодолена путем определенного мошенничества. И тогда автоматизированную подстановку применяют опять же в двух случаях: либо ввиду обычной лености, либо из-за невозможности обойти монопольный захват экрана тестером. Тем не менее, в таком тестере могут существовать недостатки, позволяющие беспрепятственно осуществлять доступ к дочерним окнам на форме, если применяется оконный тестер, либо к содержимому консоли, если применяется консольный тестер.

В дальнейшем будем ориентироваться на тестер, выполненный в виде оконного Windows-приложения, поскольку такое исполнение является наиболее популярным.

В качестве решения по исключению автоматизированной подстановки можно предложить следующее: не размещать вопрос и ответы теста в какие-либо дочерние окна, типа StaticText, Edit и т.п., откуда их легко можно получить в виде текста при помощи Windows-функций. Будет надежнее, если тест на форме будет отображаться как

растровое изображение – совсем не обязательно это должен быть рисунок, достаточно использования компонента Label или функции TextOut. В интерпретации разных языков программирования всегда найдутся подобные объекты и методы. Функцию TextOut можно вызывать и непосредственно как WinApi.

Монопольный захват экрана тестером

Теперь более подробно проанализируем монопольный захват экрана тестером. Для начала нужно определиться: нужно ли это? Если студенты будут пользоваться обыкновенными бумажными шпаргалками – тестер не сможет помешать. В тоже время «электронные» шпаргалки имеют существенное преимущество перед бумажными: быстрый поиск. Когда тестирование жестко ограничено по времени, студент физически не успеет просмотреть свои бумажные шпаргалки и ответить на все вопросы. Отметим, что если студент не сумел раздобыть ключи правильных ответов, то ситуация для него еще более «усугубляется». Поиск правильного ответа придется выполнять по лекционным и другим литературным источникам. Здесь преимущество «электронных» шпаргалок перед бумажными не подвергается сомнению.

Теперь в качестве примера рассмотрим тестер, который выполняет монопольный захват экрана, и на первый взгляд кажется, что со своей работой он справляется. Тестер выполнен в виде оконного Windows-приложения в среде Delphi. При запуске в процедуре FormShow, тестер устанавливает системный клавиатурный хук, используя Windows-функцию SetWindowsHookEx. Форма создается без заголовка и запускается в максимизированном состоянии, занимая при этом весь экран. В обработчике клавиатурного хука отслеживаются и блокируются сочетания клавиш, позволяющие свернуть окно, переключиться на другое приложение: это все сочетания с клавишей Windows, особенно, такие как <Windows+D>, <Windows+M>, <Windows+L> и т.п., <Alt+Tab>.

В дополнение к этому, форма должна иметь стиль TopMost для того, чтобы располагаться поверх других окон. Но в тоже время это не будет запрещать другим окнам со стилем TopMost располагаться впереди формы тестера. Это может быть, например, Диспетчер задач, вызываемый сочетанием клавиш <Ctrl+Alt+Del>, такое сочетание клавиш трудно блокировать. Полностью запретить появление других окон возможно, используя технологию все тех же хуков (хук WH_CBT с кодом HCBT_CREATEWND, хук WH_SHELL с кодом HSHELL_WINDOWACTIVATED). Однако далее будет показано, что защита, построенная на хуках, надежной не является. Данную проблему решим при помощи таймера, в процедуре которого через 100 мс, будет выполняться следующий код:

SetWindowPos(Application.Handle, hwnd_topmost, 0,0,0,0, SWP_NOMOVE or SWP_NOSIZE)

Теперь даже при вызове Диспетчера задач форма тестера будет переноситься на передний план. Мелькание Диспетчера задач может наблюдаться, но это уже не принципиально. Поскольку при старте форма запустилась в максимизированном состоянии и приняла размеры экрана, то определение параметров в функции SetWindowPos упрощается. Размеры и горизонтально-вертикальное положение формы на экране задавать не будем, путем введения флагов SWP_NOMOVE, SWP_NOSIZE.

Теперь можно приступать к анализу описанной защиты. Рассмотрим все возможные варианты сброса монопольного захвата экрана, и увидим, что в реальности такую защиту можно назвать очень слабой.

Атаки на монопольный режим тестера. Рекомендации по защите

Воздействие на основное окно. Студенту будет значительно удобнее, если он сможет приобрести тестер на руки для анализа его слабых мест, и уже в следующий раз прийти на экзамен во всеоружии. Вариант взлома данного тестера будет зависеть от профессионального уровня студента-хакера. В дальнейшем изложении будем придерживаться термина «хакер» без приставки «студент», поскольку сам студент хакером может и не быть, но такого человека найти может. Начнем от простого к сложному. Продвигаясь в своем рассуждении наверх по ступенькам профессионализма, варианты взлома будем перемежать предложениями соответствующей защиты.

Первое что приходит в голову хакеру, проверить воздействие Windows-функций на окно тестера. Единственное препятствие, которое хакеру нужно преодолеть – это догадаться, каким способом запустить свою программу в то время, когда весь экран занимает окно тестера. Из того, что является наиболее простым для реализации – это компонент операционной системы «Назначенные задания» или применение таймера в своей программе. Последний вариант удобнее: достаточно запустить свою программу, затем тестер, и дождаться срабатывания таймера. Еще более удобным будет использование Windows-функции

RegisterHotKey. С ее помощью можно назначить пару удобных для себя сочетаний клавиш, и предварительно запущенную программу активизировать, либо переводить в пассивный режим.

После этого, в данной программе нужно получить Handle окна тестера. Если окно тестера имеет достаточно уникальное название, то Handle получить можно, используя Windows-функцию FindWindow. В любом случае можно воспользоваться Windows-функцией GetForegroundWindow, возвращающей Handle окна, имеющего приоритет ввода. Под последним термином нужно понимать такое окно, с которым в текущий момент работает пользователь; т.е. окно, получившее клавиатурный фокус. Не следует путать приоритет ввода с активным окном. Каждое приложение среди своих форм имеет одну активную, и именно она получает приоритет ввода, когда пользователь переключается на это приложение. Напомним, что программа получает Handle окна после того, как был запущен тестер, и соответственно, он и имеет приоритет ввода.

Имея Handle окна с ним можно выполнять различные манипуляции. Это уже будет зависеть от способностей и фантазий хакера. Например, применить Windows-функцию ShowWindow с параметром SW_SHOWMINIMIZED, либо SW_HIDE – сворачивая, либо скрывая окно тестера. А затем эту же Windows-функцию с параметром SW_SHOWMAXIMIZED, восстанавливающую окно тестера на весь экран. Также можно воспользоваться Windows-функцией MoveWindow, например в таком виде, располагающей окно тестера в левой половине экрана:

MoveWindow(GetForegroundWindow, 0,0, Screen.WorkAreaWidth shr 1, Screen.WorkAreaHeight, true);

Этот же код, но за вычетом “shr 1”, восстановит окно таймера на весь экран.

Почему же такой довольно простой взлом оказался возможным? Потому что разработчики тестера, в параметрах Windows-функции SetWindowPos поленились указать размеры и горизонтально-вертикальное положение формы на экране. Ведь тестер запустился в максимизированном состоянии, а изменение разрешения экрана в период выполнения работы не предполагается. У хакера же все получилось достаточно просто. Даже, если он не знает, как победить окно верхнего уровня и передвинуть такое окно в Z-порядке (под этим понимается расположение окон по вектору, смотрящему на нас с экрана), то ему никто не помешал просто передвинуть окно в сторону.

Поэтому для нормальной защиты в Windows-функции SetWindowPos не целесообразно использовать флаги SWP_NOMOVE, SWP_NOSIZE, а нужно указывать конкретные значения по размерам экрана. Кроме того, необходимо добавить флаг SWP_SHOWWINDOW для того, чтобы возобновлять видимость формы. И поскольку эта функция не умеет восстанавливать окно из свернутого состояния, то рядом с ней в процедуре таймера нужно будет добавить:

if IsIconic(handle) then ShowWindow (handle, SW_RESTORE);

либо:

if not IsZoomed(handle) then ShowWindow (handle, SW_SHOWMAXIMIZED);

Windows-функция SetWindowPos не единственная, которая необходима в процедуре таймера. Когда окно тестера находится на самом верхнем уровне, это еще не значит, что оно одновременно имеет приоритет ввода. Подобную ситуацию видел почти каждый пользователь своими глазами: когда Диспетчер Задач находится на переднем плане, это нисколько не мешает работать с окнами, которые позади него.

Для возвращения себе приоритета ввода форма нашего тестера должна будет вызывать в таймере Windows-функцию SetForegroundWindow. Но вызовом одной только этой функции единственное чего процесс добьется – это мигание ярлыка на Панели Задач. Чтобы заимствовать приоритет ввода у другого процесса, к нему нужно прикрепиться через Windows-функцию AttachThreadInput, воспользоваться функцией SetForegroundWindow, и отсоединится все также через функцию AttachThreadInput.

Создание дочернего окна

Перейдем к следующему варианту взлома. Как известно, для того чтобы создать дочернее окно на своей форме, нужно вызвать Windows-функцию CreateWindowExA, указав в качестве родителя окно формы. Но далеко не всем приходит в голову указать в качестве родителя Handle чужой формы-окна. Если хакеру это придет в голову, то на форму нашего тестера, он без проблем может поместить свое дочернее окно в виде многострочного редактора. Затем через Windows-функцию SendMessageA сообщением WM_SETTEXT, наполнить дочернее окно текстом со шпаргалками. И периодически, через зарегистрированные клавиши, показывать и скрывать его.

Как отследить данную ситуацию? На сообщения WM_CREATE, WM_NCCREATE рассчитывать не приходится, т.к. они отправляются создаваемому окну, а не родителю. При создании дочернего окна форме придет сообщение WM_PARENTNOTIFY.

Но только в том случае, если хакер не догадается создавать окно с расширенным стилем WS_EX_NOPARENTNOTIFY. В этом случае форме ничего не придет. Самое оптимальное решение, это организовать еще один таймер, с интервалом 100–200 мс. В обработчике этого таймера при помощи Windows-функции EnumChildWindows идет перечисление дочерних окон на форме. Через Windows-функцию GetWindowThreadProcessId получаем ID процесса, породившего дочернее окно, и сравниваем с ID своего процесса. Чтобы не перегружать таймер, последнее значение через Windows-функцию GetCurrentProcessId можно получить еще при старте тестера с сохранением в переменную. В результате, если дочернее окно порождено чужим процессом можно смело выполнять завершение работы тестера.

Блокировка хуков

Вернемся к технологии хуков, которой воспользовались в нашем тестере. Существует ли у хакера возможность каким-либо образом снять блокировку клавиатурного хука, и вообще любого? Все достаточно просто, поскольку эта технология предполагает порядочность программиста. Суть в том, что хуки одного и того же типа выстраиваются в цепочку. Вновь добавленный хук операционной системой ставится в начале цепочки, и после своих действий должен передавать эстафету следующему хуку. Поэтому, в конце обработчика любого хука настоятельно рекомендуется вызывать Windows-функцию CallNextHookEx, иначе другие хуки управления не получат. Хакер преследует другую цель. Никакого управления он никуда передавать не будет, в конце обработчика своего хука, такого же типа, напишет: Result:=0. Теперь после старта тестера, хакер активизирует свой хук, например, при помощи таймера, и клавиатурные блокировки тестера перестанут функционировать.

Помешать в этом хакеру возможно только административным способом. Например, поместив тестер (или его ярлык) в автозагрузку, блокировать вовремя тестирования запуск любых программ за исключением тестера. Однако в этом случае возникают другие проблемы: тестер должен закрываться какой-то комбинацией клавиш, которую можно подобрать или подглядеть. Если закрытие тестера совсем не предусмотрено, то для того, чтобы убирать из его автозагрузки, неопытный инженер может выполнять вход под именем другого пользователя. Хотя лишний раз демонстрировать пароли других пользователей нежелательно. Кроме того, при запуске в безопасном режиме автозагрузка не работает. А пользователь может сделать именно такой запуск. Проблемы могут возникнуть и с антивирусными программами. Тестирование может пересекаться с другими процедурами учебного процесса, и такой надоедливый тестер быстро утомит обслуживающий персонал.

Обсуждение всех возможностей и недостатков автозапуска – это тема отдельной статьи. В нашем случае при совместном и правильном применении Windows-функций SetWindowPos и SetForegroundWindow, использование хуков совсем не требуется, и проблема пропадает сама собой.

Отключение таймера и снятие стиля TopMost у основного окна

В следующем варианте обсудим, как хакер может отменить стиль TopMost у окна тестера, для того чтобы оно перестало быть переднеплановым. Само по себе это не сложно, достаточно в программе хакера применить такой код:

```
SetWindowPos (GetForegroundWindow,
hwnd_NoTopMost,0,0,0,0, SWP_NOMOVE
+ SWP_NOSIZE);
```

Однако в тестере специально и используется таймер, в обработчике которого вызывается эта же Windows-функция, но с «зеркальным» параметром: hwnd_TopMost. Перед хакером возникает следующий вопрос: «возможно ли остановить таймер в другом приложении»? В нашей статье уже пришло время ответить девизом: «для хакера нет ничего невозможного»!

Компонент Timer компилятор Delphi реализует через обычный таймер, имеющий кратность 15,6 мс. Но создавая таймер через Windows-функцию SetTimer, компилятор Delphi закрепляет его не окном формы, а создает для него отдельное окно с пустым именем и названием класса «TPUtilWindow». И так для каждого таймера отдельное окно, или другими словами: у каждого такого окна только один таймер. На этом и можно построить следующий алгоритм. Используя Windows-функцию EnumWindows, просматриваем все окна в системе и читаем название класса через Windows-функцию GetClassName. Повстречав окно класса «TPUtilWindow» через цепочку Windows-функций: GetWindowThreadProcessId, OpenProcess, GetModuleFileNameEx получаем путь к приложению, использующему таймер. Если это интересующий нас тестер, то остается вызвать Windows-функцию KillTimer. После этого беспрепятственно можно отменить стиль TopMost у окна тестера.

Защита от остановки таймера очень проста, и в тоже время очень надежна. До-

статочно будет воспользоваться булевой переменной, которая в обработчике таймера будет принимать одно значение, а в процедуре навигации по тестам – другое. В процедуре навигации по этой переменной можно отслеживать: «живой» таймер или нет.

Взлом кода тестера

Настало время затронуть самую трудную ситуацию с точки зрения защиты – взлом с использованием дизассемблера и бинарного редактора. Требует как от хакера, так и от разработчика тестера (да и вообще любой другой программы) знания языка программирования ассемблера и понимания работы программы изнутри на уровне системы. Тема достаточно большая, и применение различных вариантов защиты зависит от глубины знаний программиста, и даже в некотором смысле всем известного «авось» – авось хакер не такой умный. Защитить код со 100-процентной надежностью невозможно, если за дело взялся хакер умеющий читать код изнутри.

Например, что может сделать такой хакер с нашим тестером, причем достаточно быстро? Установив в отладчике ловушку в старших виртуальных адресах на Windows-функцию SetWindowPos, либо SetForegroundWindow, в стеке он увидит адрес возврата в обработчик таймера. Остается закомментировать вызов этих функций серией инструкций ассемблера NOP.

Можно сделать еще больше – это для очень ленивых. Найти обработчик кнопки, выводящий на экран результат о тестировании. И организовать там вывод: “количество правильных ответов=100 %”. Эта задача несколько сложнее. Если в предыдущем случае хакер предполагал название Windows-функций, обеспечивающих монопольный захват экрана, и их вызов непосредственно из обработчика таймера, то здесь нужно хорошо знать, какие Windows-функции, и из каких методов Delphi вызываются. Иерархия вложенности и объем методов довольно громоздкие. Однако не следует изобретать велосипед. Такие программы как EMS Source Rescuer, умеют работать с форматами исполнимых файлов, создаваемых в средах разработки фирмы Borland. Т.е. адрес нужного нам обработчика, эта программа покажет.

Что можно здесь предложить в качестве защиты? Например, не будет лишним, воспользоваться пробелом в программе EMS Source Rescuer. Если перенести объявление обработчика события в секцию PRIVATE, либо PUBLIC, данная программа не показывает адрес таких обработчиков. Или можно ввести хакера в заблуждение, наведя его на «ложный след». Т.е. пусть он увидит адрес обработчика кнопки с помощью этой программы, но это будет не настоящий обработчик. В глубине процедуры FormCreate сделаем переопределение обработчика кнопки на другую процедуру. Хакер потеряет много времени и нервов пока догадается в чем дело.

Как наиболее действенное предложение, можно посоветовать подсчет контрольной суммы важных с точки зрения защиты процедур. Такими у нас являются обработчики таймеров, кнопок: вывода результатов тестирования и навигации по тестам. И поместить код, выполняющий подсчет и сверяющий его с контрольной суммой – в процедуре навигации. Напомним, что у нас есть булева переменная, принимающая в обработчике таймера одно значение, и в процедуре навигации – другое. Отладчики позволяют ставить ловушку, срабатывающую при обращении к ячейке памяти (переменной). Обработчик таймера не большой и хакер наверняка увидит эту странную «бесхозную» переменную. Воспользовавшись ловушкой на эту переменную, он выйдет на соответствующее место в обработчике навигации: и может там изменить код. Вот поэтому код процедуры навигации тоже должен быть внесен в контрольную сумму. Механизм подсчета контрольной суммы несложен. Защищаемые процедуры расположим в модуле подряд, установим указатель на начало первой процедуры. pTmr: ^byte; pTmr:= @EnumProc;

Подсчет выполняем, пока не встретим два подряд идущих байта «$5D; $C3».Что соответствует ассемблерной мнемонике «pop ebp; ret». Либо это могут быть байты «$5B; $C3», т.е. «pop ebx; ret». Нужно в каждом конкретном случае смотреть в отладчике, какими байтами заканчивается последняя защищаемая процедура. Кроме того, необходимо проконтролировать, чтобы эта комбинация байт в другом месте этих процедур не встречалась, чего нельзя сказать об одиночном байте “$C3”.

Нельзя пропустить в нашей защите еще один очень важный момент. После того, как тестер обнаруживает, что его взломали, он должен завершить свою работу, например через Windows-функцию ExitProcess. И опять же хакер, поставив на нее ловушку в старших виртуальных адресах, в стеке получает место, где эта функция вызывается. Эту проблему можно решить, если уходить на эту функцию с использованием языка ассемблера. Поскольку возврата из этой функции уже не будет, то отсутствие адреса возврата, ошибки не вызовет.

asm jmp dword ptr [$00467338] end;

Число в примере условно. Смотреть его надо в отладчике, причем на финальной стадии написания кода. Обозначает собой адрес в секции импорта программы. В данную ячейку памяти при загрузке программы помещается адрес Windows-функции ExitProcess. Сам этот адрес зависит от версии операционной системы. А секция импорта во всех случаях всегда будет находиться на своем месте. Вызов Windows-функции ExitProcess необходимо где-то еще написать в программе обычным образом (в данном случае неиспользуемый вызов), для того чтобы компилятор подготовил место для этой функции в секции импорта.

Заключение

Подводя итоги нужно отметить, что рассмотрены далеко не все варианты взлома и соответствующей защиты. Основная цель статьи продемонстрировать всем, кто пользуется информационными технологиями в учебном процессе то что, во-первых, всегда существует возможность взлома любого программного продукта – и проявлять халатность недопустимо. Во-вторых, всегда найдутся контрмеры, позволяющие свести возникшую ситуацию в правовое поле.

Список литературы

1. Рихтер Дж. P558 Windows для профессионалов: создание эффективных Win32 приложений с учетом специфики 64-разрядной версии Windows / Пер. с англ – 4-е изд. – СПб.; Питер; М.: Издательско-торговый дом «Русская Редакция», 2001. – 752 с.; ил.

2. Пирогов В.Ю. ASSEMBLER. Учебный курс. – М.: Издатель Молгачева СВ., Издательство Нолилж, 2001. – 848 с., ил.

3. Пирогов В. Ю. Ассемблер для Windows. – М.: Издатель Молгачева С.В., 2002. – 552 с., ил.