Введение

Asprotect - это один из сложнейших пакеров для реверсера средней квалификации, так что для первого урока по распаковке защитных схем я выбрал именно эту защиту. Так же Вы выучите как восстановить IAT (Import Address Table), возвратить перемещенные байты и как использовать лог трассировки.

Данные о цели

Файл..... iconmaker.exe [download]
Разработчик... King Lee, Fred Xu, Steven Bai
Программное обеспечение .. OllyDbg, ImportREC + **Asprotect Plugin** [download], LordPE [download]

Adficio

При использовании Asprotect с программой происходят следующие изменения:

- Искажается OEP, EP определяется Asprotect'ом.
- Обфусцируются вызовы модулей (функции заменяются на JMP).
- Asprotect берет определенное количество байт с исходной программы и перемещает их.

Так что для распаковки нашей программы нам необходимо найти OEP, восстановить IAT и возвратить перемещенные байты (если таковые имеются; Asprotect не всегда перемещает байты). Но перед тем, как мы сможем приблизиться к защите через отладчик, нам необходимо настроить его немного иначе, чем обычно.

1 - a. Открываем Ollydbg
b. Идем в *Options > Debugging options > Exceptions*
c. Устанавливаем настройки как показано ниже:

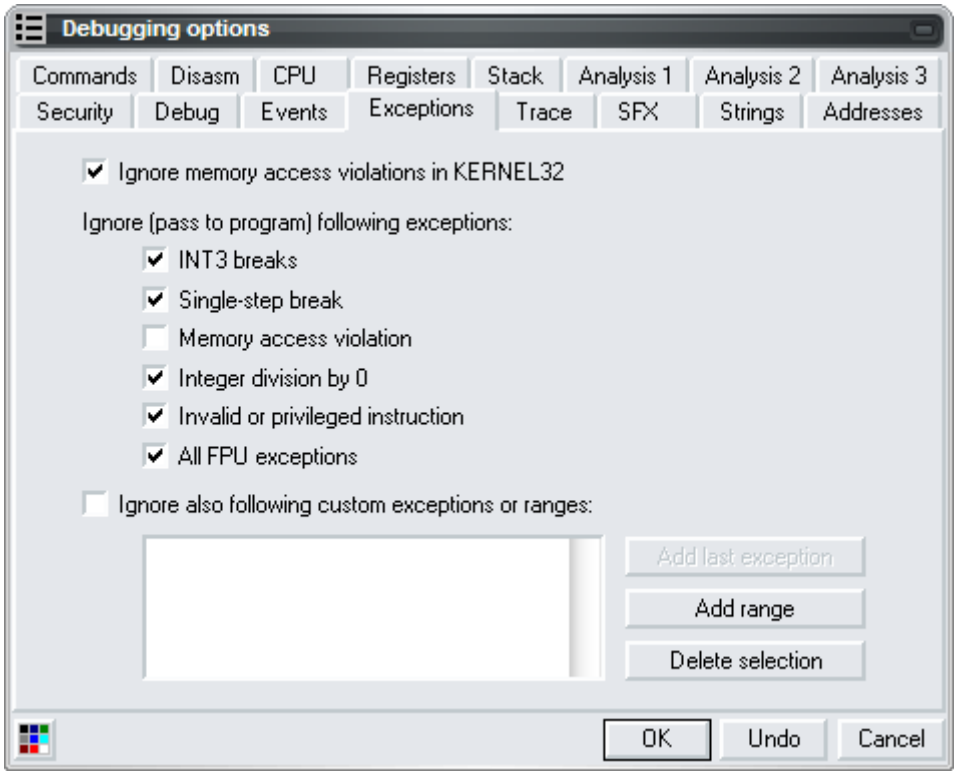


Рис. 1 Опции Ollydbg

2 - a. Перезапускаем Ollydbg
b. Идем в *File > Open* и загружаем программу
c. Нажимаем **No** в окне сообщения

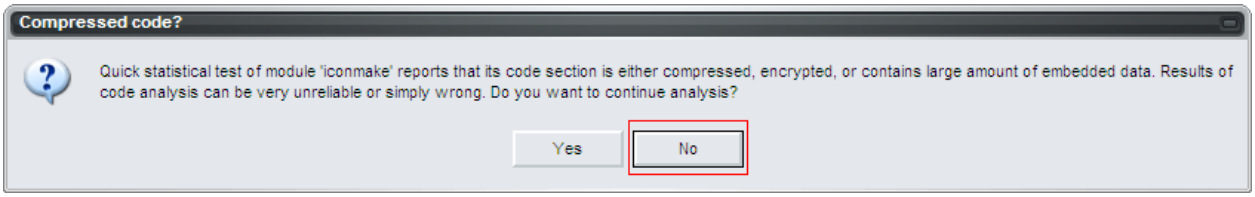


Рис. 2 Запрещаем анализ кода

Первый шаг - это нахождение OEP. Причиной смены конфигурации отладчика (задаем игнорирование большого количества исключений) является то, что Asprotect ловко использует исключения с целью запутать реверсера. Каждая программа, защищенная при помощи Asprotect, проходит через несколько (иногда до 30) исключений перед выполнением кода исходной программы. Нам необходимо точно определить, сколько исключений необходимо пройти перед тем, как программа выполнится, а потом пройти их еще раз, что бы на этот раз мы остановимся на последнем из них перед run-time :)

3 - a. Нажимаем [SHIFT] + [F9] (приблизительно 27 раз) пока не достигнем этого блока:

```
00A339EC  3100          XOR DWORD PTR DS:[EAX],EAX
00A339EE  64:8F05 000   POP DWORD PTR FS:[0]
00A339F5  58            POP EAX
00A339F6  833D B07EA3   CMP DWORD PTR DS:[A37EB0],0
00A339FD  74 14         JE SHORT 00A33A13
00A339FF  6A 0C         PUSH 0C
00A33A01  B9 B07EA300   MOV ECX,0A37EB0
00A33A06  8D45 F8       LEA EAX,DWORD PTR SS:[EBP-8]
00A33A09  BA 04000000   MOV EDX,4
00A33A0E  E8 2DD1FFFF   CALL 00A30B40
00A33A13  FF75 FC       PUSH DWORD PTR SS:[EBP-4]
00A33A16  FF75 F8       PUSH DWORD PTR SS:[EBP-8]
00A33A19  8B45 F4       MOV EAX,DWORD PTR SS:[EBP-C]
00A33A1C  8338 00       CMP DWORD PTR DS:[EAX],0
00A33A1F  74 02         JE SHORT 00A33A23
00A33A21  FF30          PUSH DWORD PTR DS:[EAX]
00A33A23  FF75 F0       PUSH DWORD PTR SS:[EBP-10]
00A33A26  FF75 EC       PUSH DWORD PTR SS:[EBP-14]
00A33A29  C3            RETN
```

Рис. 3 Последнее исключение перед run-time

4 - a. Ставим точку останова на первом **RETN** в **0xA33A29**
b. Нажимаем **Yes** в окне сообщения

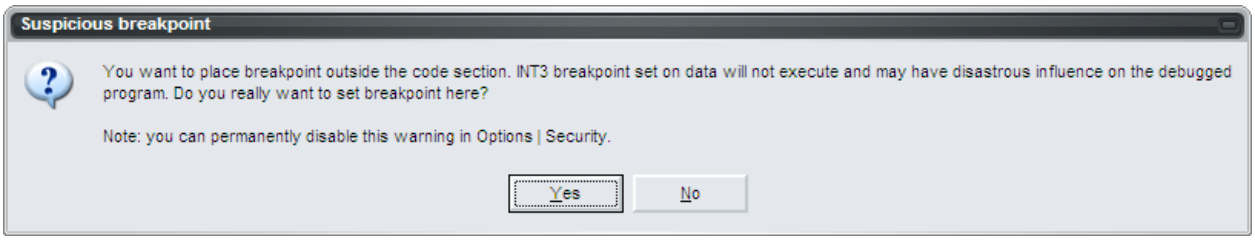


Рис. 4 Подозрительна точка останова

5 - a. Нажимаем [SHIFT] + [F9] для прохождения нашей точки останова

```
00A33A29  C3 RETN
```

Рис. 5 Успешная остановка на точке останова

Мы установили точку останова на первом **RETN** последнего исключения перед run-time и, благодаря этому, мы загрузили все команды, которые Asprotect выполняет глубоко в памяти. Так же благодаря этому, мы прошли все исключения, которые использует Asprotect перед распаковкой фактических инструкций программы. **Запомните**, следующая техника может быть использована во всех Ваших будущих руководствах по распаковке.

6 - a. Нажимаем [ALT] + [M]
b. Правый клик на **0x401000** и выбираем *Set memory breakpoint on access*

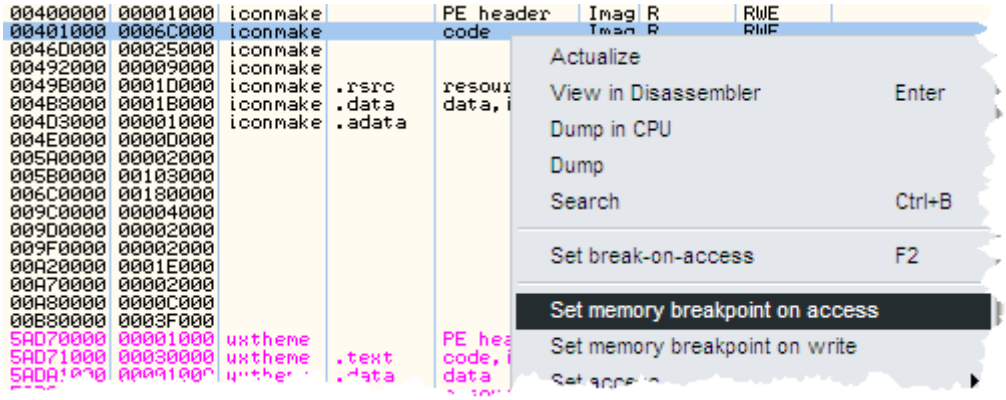


Рис. 6 Установка точки останова на секции **.code**

Установив другую точку останова (в этот раз - в начале секции **.code**) мы можем точно определить EP, так как в секции **.code** содержится код программы, так что ее первая инструкция, очевидно, и есть началом главной программы.

7 - a. Нажимаем [F9]

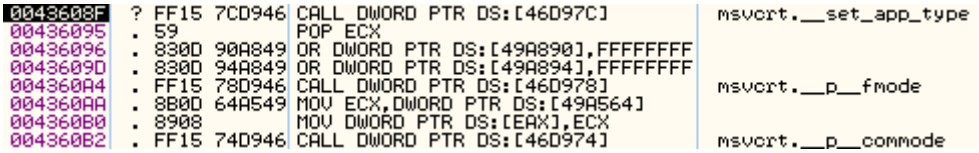


Рис. 7 EP

Этот раздел - это EP программы-цели (не OEP) недавно назначенный Asprotect'ом. " set_app_type" - это стандартная функция Microsoft Visual C++ 6.0, которая определяет тип приложения: консольное или оконное. Зачастую не важно значение этой функции, а важно само наличие ее. Сейчас мы знаем, что:

- EP программы - **0x43608F**

Что бы узнать, были ли перемещены некоторые байты программы, нам необходимо прокрутить вверх несколько строк и посмотреть на код. Если в программе с этим пакером были перемещены некоторые байты, то их можно найти как раз перед EP; это отличительная черта Asprotect'а этой версии :)

8 - a. Прокручиваем вверх до адреса **0x43605C**
b. Нажимаем [CTRL] + [A]

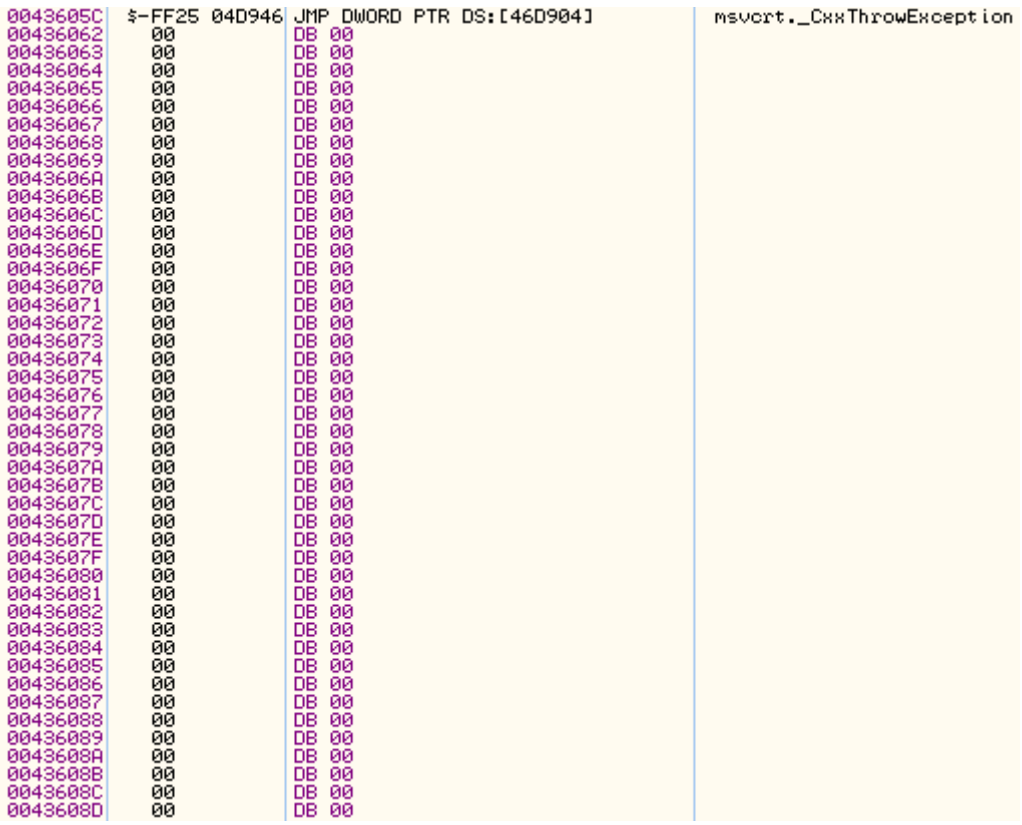


Рис. 8 Ячейки перемещенных байт

В указанном участке мы видим 44 ячейки перемещенных байт. Надо вернуть их назад :)

9 - a. Нажимаем [CTRL] + [F2]
b. Нажимаем **Yes** в первом окне сообщения
c. Нажимаем **No** во втором окне сообщения
d. Повторяем шаги 4 - 6

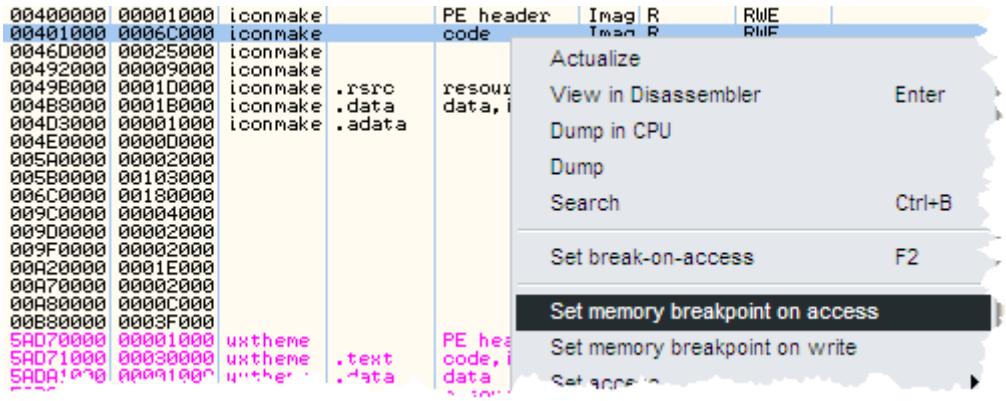


Рис. 9 (шаг 6)

Теперь, когда у нас BP указывает на секцию **.code**, мы запустим условную трассировку.

10 - a. Нажимаем [CTRL] + [T]
b. Устанавливаем настройки, как показано на рисунке, и жмем OK:

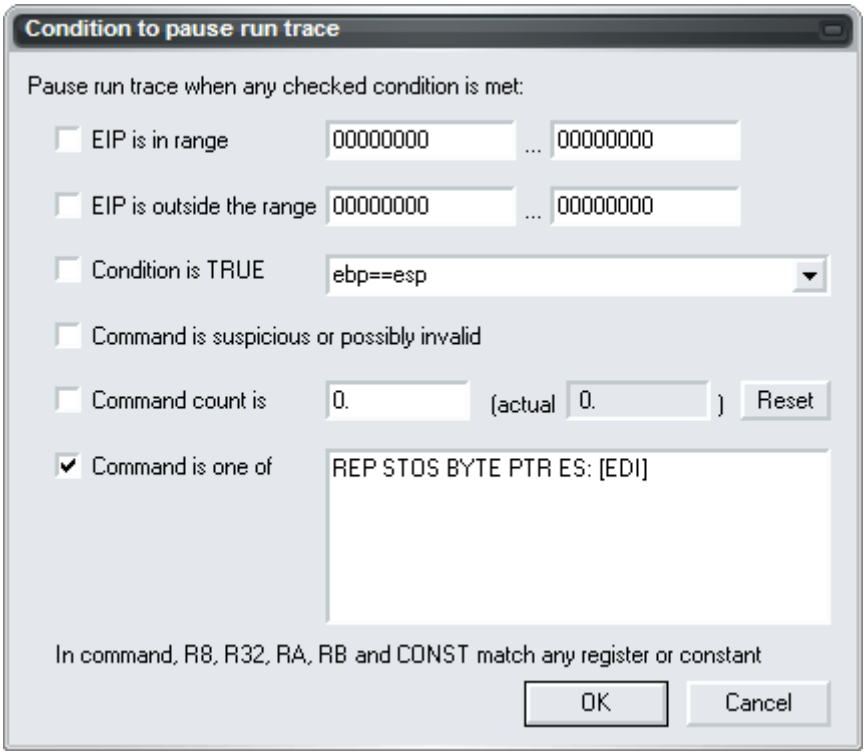


Рис. 10 Настройка условной трассировки

11 - a. Нажимаем [CTRL] + [F11] и ждем примерно ~40 секунд пока не получим:

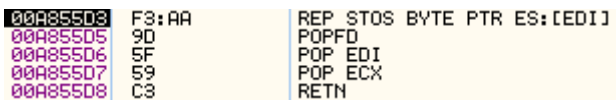


Рис. 11 Профит

Эти 5 инструкций являются "сигнатурой" этой версии Asprotect'a, так что достигнув этой точки, мы запускаем трассировку в наш назначенный раздел.

12 - a. Идем в *View > Run Trace*
b. Правый клик в любой точке (не забываем, что *Highlight register > EBP is selected*)

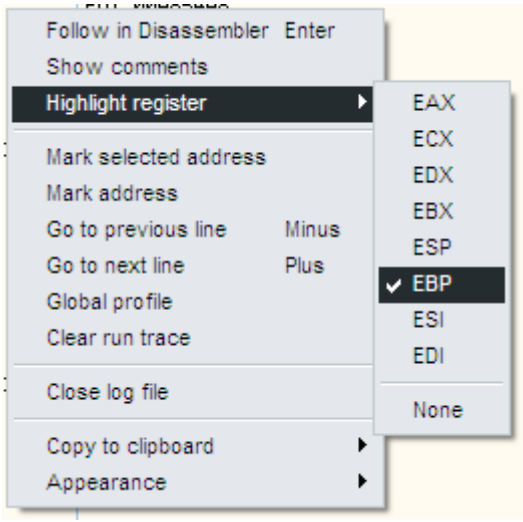


Рис. 12 Подсветка регистра EBP

13 - a. Прокручиваем вниз до конца лога трассировки

```
32. Main | 00A856C3 | PUSH EBP                                |
31. Main | 00A856C4 | MOV EBP,ESP                             | EBP=0012FFC0
30. Main | 00A856C6 | PUSH -1                                 |
29. Main | 00A856C8 | PUSH 471F88                             |
28. Main | 00A856CD | PUSH 43628A                             |
27. Main | 00A856D2 | MOV EAX,DWORD PTR FS:[0]                | EAX=0012FFE0
26. Main | 00A856D8 | JMP SHORT 00A856DC                      |
25. Main | 00A856DC | PUSH EAX                                |
24. Main | 00A856DD | MOV DWORD PTR FS:[0],ESP                |
23. Main | 00A856E4 | SUB ESP,68                              |
22. Main | 00A856E7 | JMP SHORT 00A856EB                      |
21. Main | 00A856EB | PUSH EBX                                |
20. Main | 00A856EC | JMP SHORT 00A856F0                      |
19. Main | 00A856F0 | PUSH ESI                                |
18. Main | 00A856F1 | JMP SHORT 00A856F5                      |
17. Main | 00A856F5 | PUSH EDI                                |
16. Main | 00A856F6 | MOV DWORD PTR SS:[EBP-18                |
15. Main | 00A856F9 | XOR EBX,EBX                             | EBX=00000000
14. Main | 00A856FB | MOV DWORD PTR SS:[EBP-4]                |
13. Main | 00A856FE | PUSH 2                                  |
```

Рис. 13 Перемещенные байты

Для того, что бы быть точным касательно трассировки: мы использовали условную трассировку для того, что бы достигнуть момента, когда Asprotect начинает "распаковку" программы в память.

Мы обнаружили перемещенные байты, которые находятся по адресу **0xA856C3**, однако Asprotect добавил немного мусорного кода. Но не стоит волноваться по этому поводу, нет ничего тяжелого в том, что бы убрать этот мусор из исходного кода, потому что мусор - это команды **JMP SHORT** ;)

Всякий раз, когда Asprotect перемещает байты из программы, их исходное начало это OEP (в нашем случае - **0x436062**).

14 - a. Запишем каждую инструкцию с **0xA856C3** по **0xA856FE** в блокнот пропуская **JMP SHORT**

```
32. Main | 00A856C3 | PUSH EBP                                |
31. Main | 00A856C4 | MOV EBP,ESP                             | EBP=0012FFC0
30. Main | 00A856C6 | PUSH -1                                 |
29. Main | 00A856C8 | PUSH 471F88                             |
28. Main | 00A856CD | PUSH 43628A                             |
27. Main | 00A856D2 | MOV EAX,DWORD PTR FS:[0]                | EAX=0012FFE0
26. Main | 00A856D8 |                                         |
25. Main | 00A856DC | PUSH EAX                                |
24. Main | 00A856DD | MOV DWORD PTR FS:[0],ESP                |
23. Main | 00A856E4 | SUB ESP,68                              |
22. Main | 00A856E7 |                                         |
21. Main | 00A856EB | PUSH EBX                                |
20. Main | 00A856EC |                                         |
19. Main | 00A856F0 | PUSH ESI                                |
18. Main | 00A856F1 |                                         |
17. Main | 00A856F5 | PUSH EDI                                |
16. Main | 00A856F6 | MOV DWORD PTR SS:[EBP-18                |
15. Main | 00A856F9 | XOR EBX,EBX                             | EBX=00000000
14. Main | 00A856FB | MOV DWORD PTR SS:[EBP-4]                |
13. Main | 00A856FE | PUSH 2                                  |
```

Рис. 14 Перемещенные байты

Посмотрим на несколько линий под этой секцией (**0xA85708**), где можно увидеть вызов, который заносит текущий EP в стек. Ячейки перемещенного кода начинаются с **0x436062**, а EP находится по адресу **0x43608F**. Для начала изменим точку перехода jump'a на наш OEP (который указывает на ячейки с перемещенными байтами).

15 - a. Выбираем **0xA85708** в Run Trace dialog и нажимаем [ENTER]
b. Двойной клик на выбранной инструкции по адресу **0xA85708** и пишем:

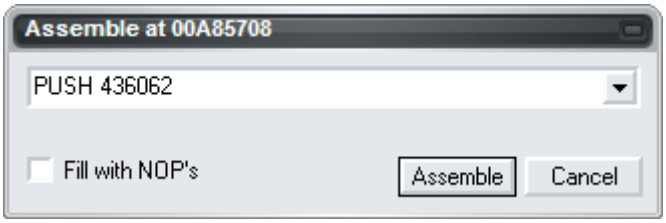


Рис. 15 Восстановление OEP

Хотя мы и определили перемещенные байты и восстановили OEP, мы еще не готовы к восстановлению перемещенных байт. Для начала необходимо восстановить IAT (спасибо dqtln за указание на это!). Теперь мы собираемся сделать дамп программы.

16 - a. Повторяем шаг 9
b. Повторяем шаги 7 и 8
c. Правый клик на **0x436062** и выбираем *New origin here*

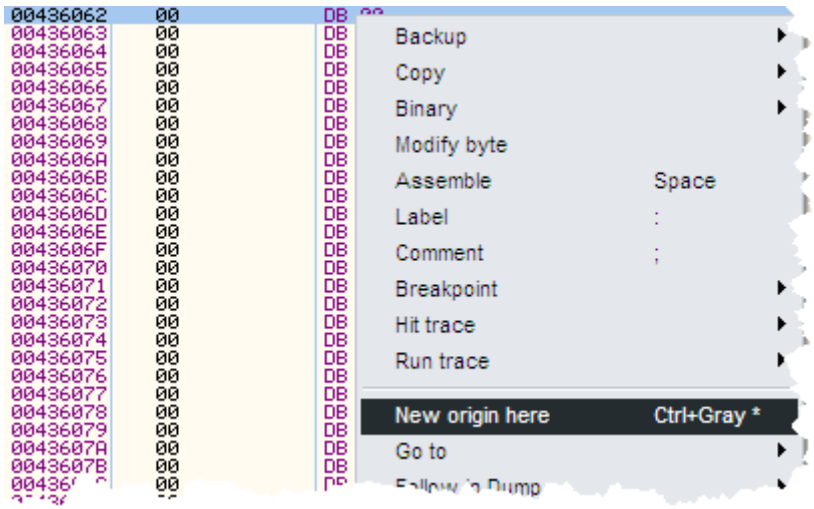


Рис. 16 Восстановление OEP

17 - a. Идем в *Plugins > OllyDump > Dump debugged process*
b. Ставим настройки как показано ниже (затем нажмите **Dump** и сохраните файл как "dump.exe"):

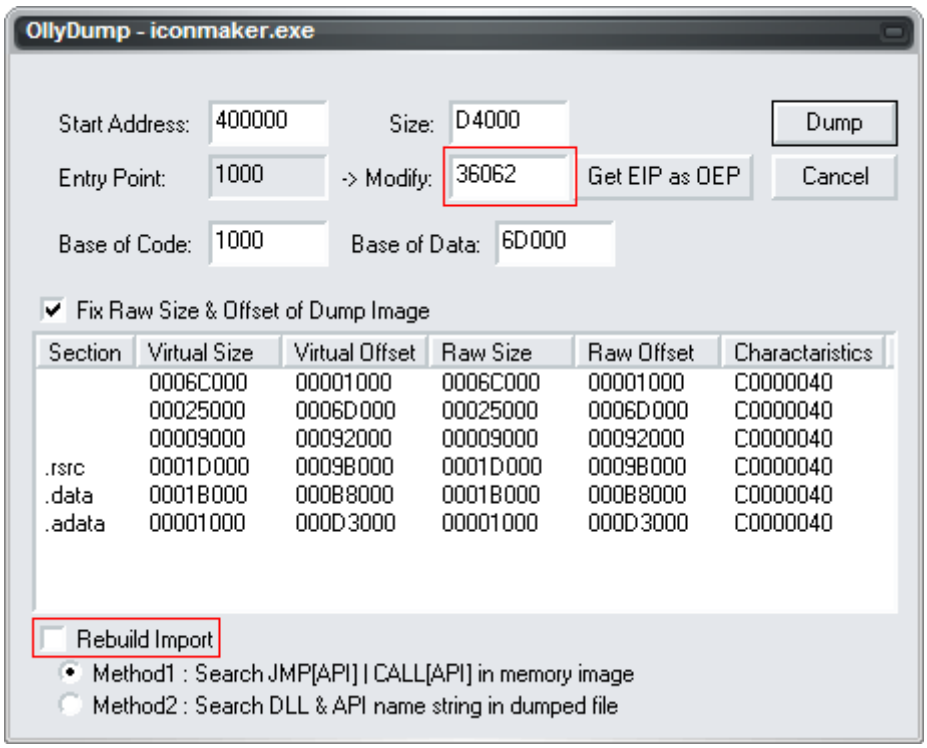


Рис. 17 Делаем дамп программы

18 - a. Сворачиваем ollydbg и открываем imprec and и выбираем нашу программу из комбобокса
b. В поле "OEP" задаем значение "00036062"
c. Нажимаем **AutoSearch**
d. Нажимаем **OK** в окне сообщения

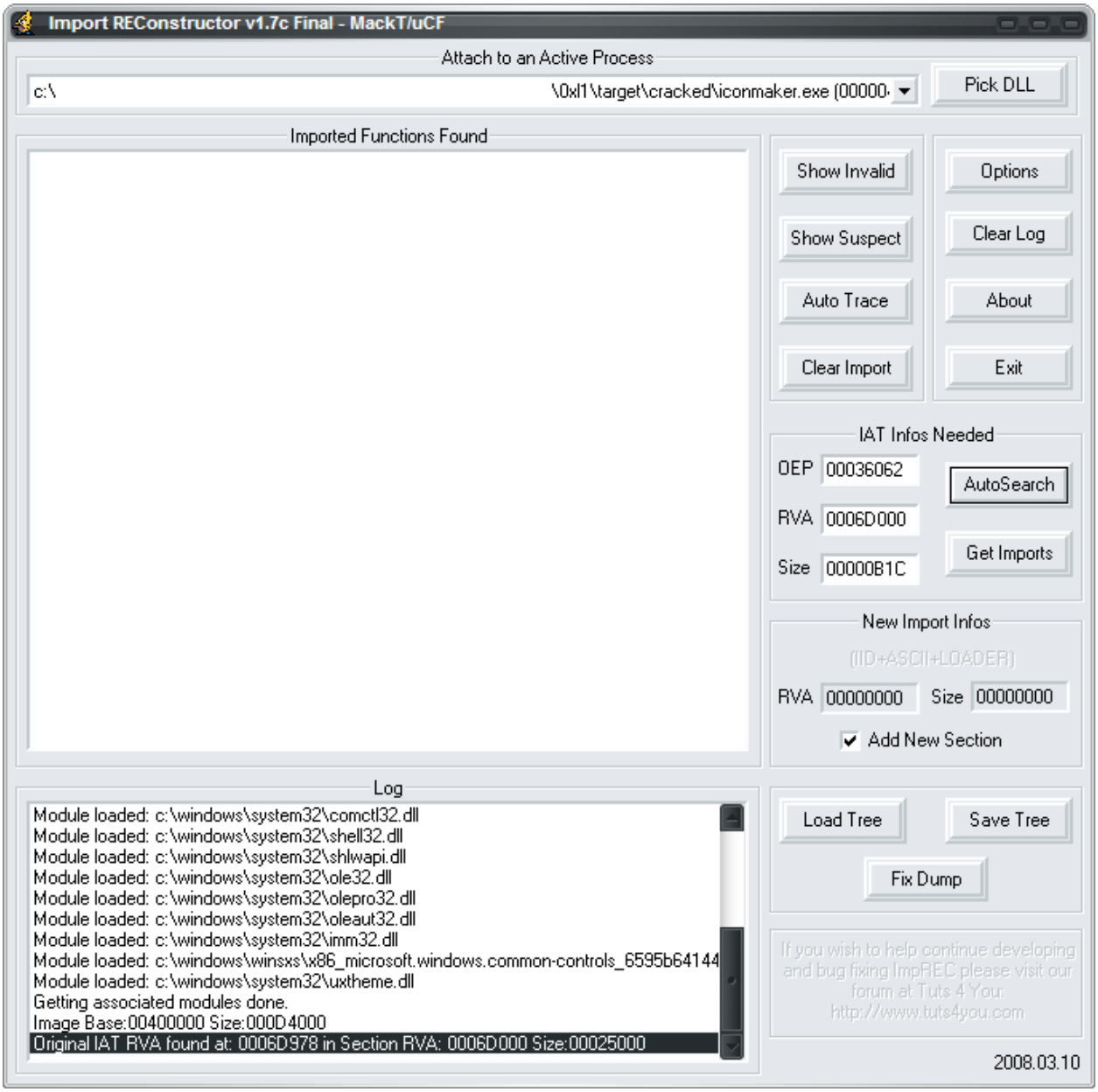


Рис. 18 Подготовка imprec

19 - a. В поле "Size" задаем значение "00001000"
b. Нажимаем **Get Imports**

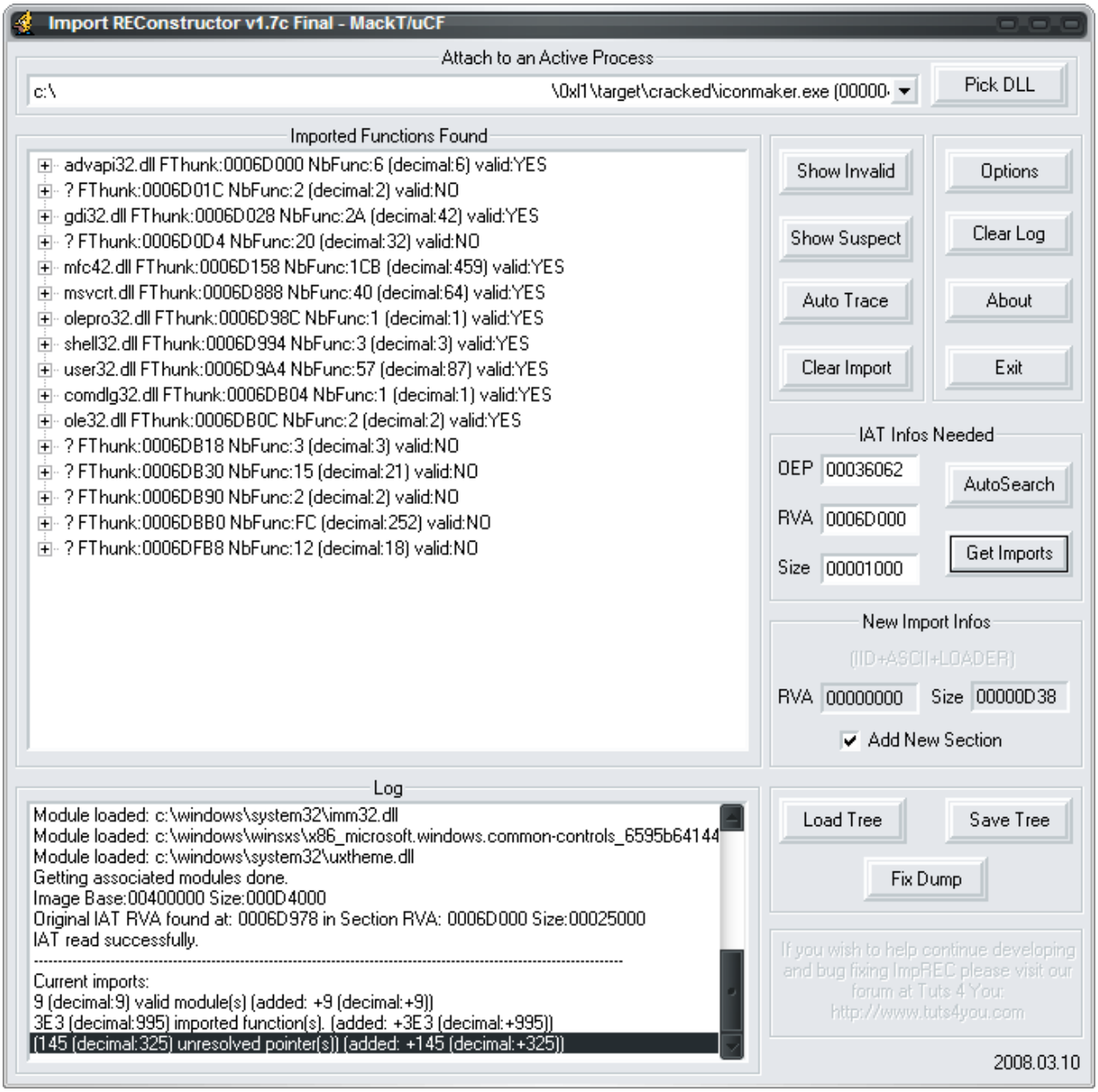


Рис. 19 Загрузка импортов

20 - a. Нажимаем **Show Invalid**

b. Правый клик на отмеченных элементах и выбираем *Trace Level1 (Disasm)*

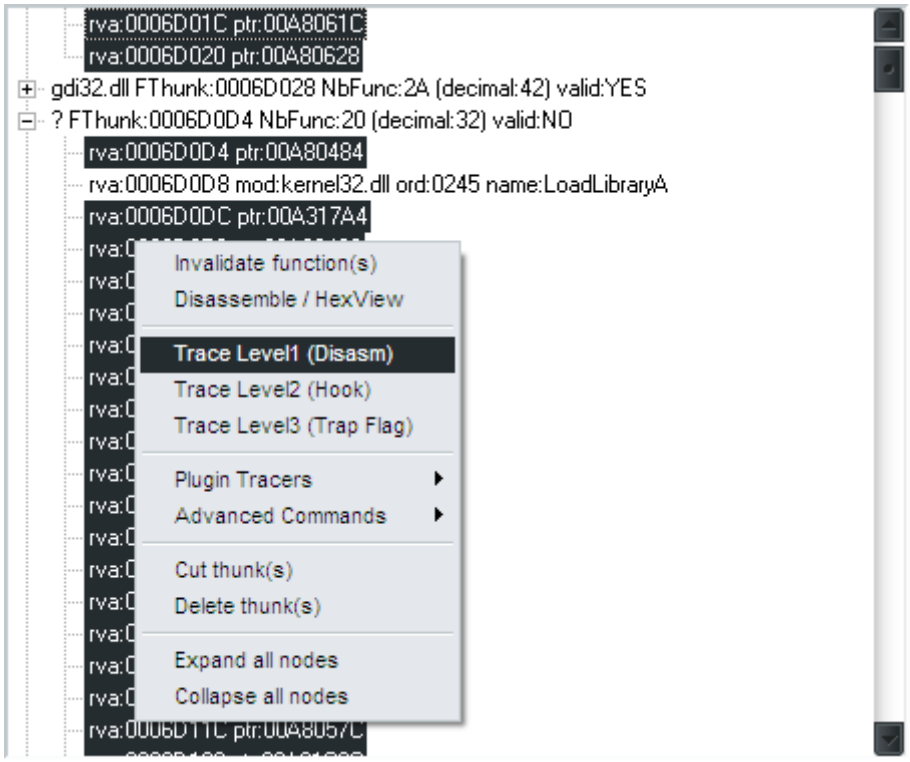


Рис. 20 Восстановление импорта Level1

21 - a. Нажмите **Show Invalid** (еще раз)
b. Правый клик на отмеченных элементах и нажимаем *Plugin Tracers > ASProtect 1.23 RC4*

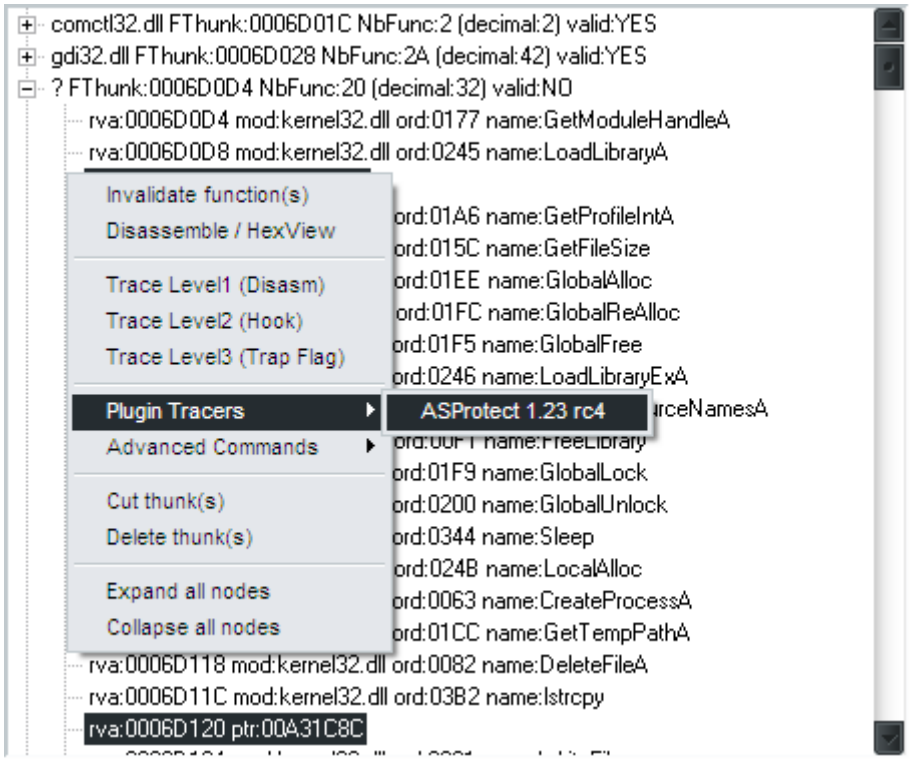


Рис. 21 Автотрассировка при помощи плагина

В логе видно немалое количество ошибок и только несколько успешных результатов трассировки, но не стоит волноваться: так и должно быть :) Остальные thunks в файле могут быть безболезненно удалены, потому что они всего лишь мусор, который Asprotect добавил в программу.

22 - a. Нажимаем **Show Invalid** (...очередной раз)
b. Правый клик на одном из отмеченных элементов и выбираем *Delete thunk(s)*

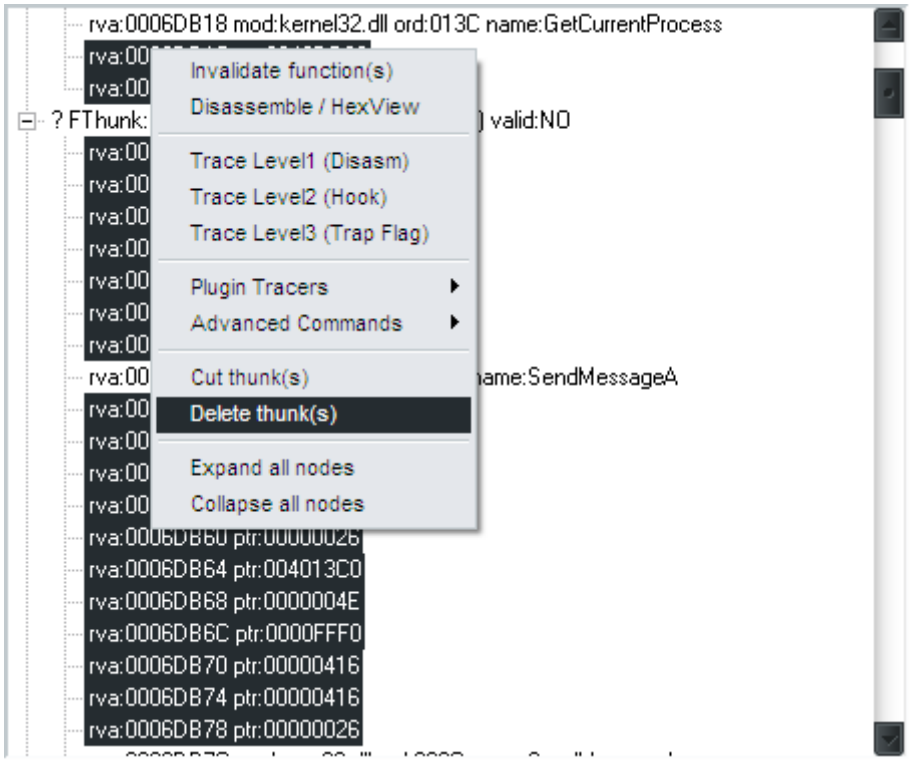


Рис. 22 Убираем мусорные thunks

23 - a. Нажимаем **Fix Dump**
b. Выбираем дамп ("dump.exe") и нажимаем **Open**

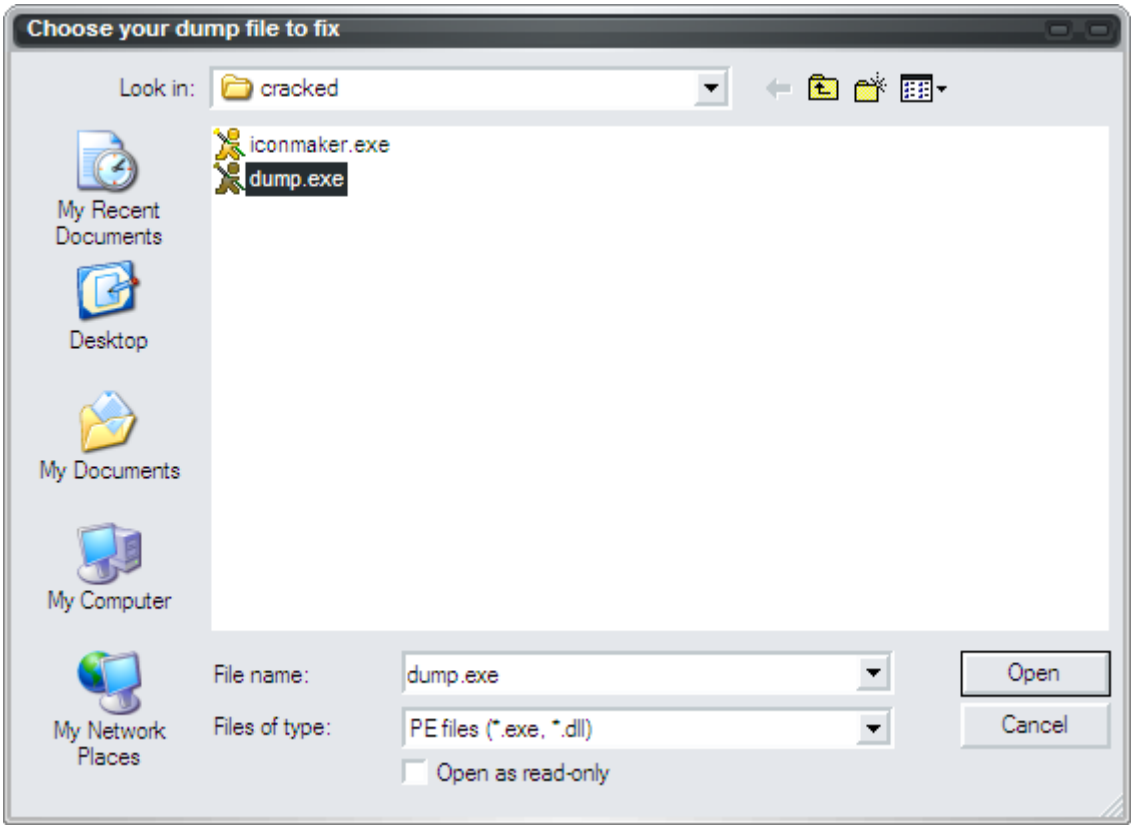


Рис. 23 Сохраняем новый дамп

Imprec сохранил наш новый дамп как "dump_.exe" и теперь мы собираемся добавить перемещенные байты в него для того, что бы снова сделать его рабочим.

24- a. Закрываем ollydbg и открываем "dump_.exe" в заново открытом ollydbg
b. Выбираем все команды начиная с **0x436062** по **0x43608C**, правый клик и выбираем *Assemble*
c. Восстанавливаем все инструкции из перемещенных байт
d. Нажимаем [CTRL] + [A]

```
00436062  $ 55           PUSH EBP
00436063  . 8BEC         MOV EBP,ESP
00436065  . 6A FF        PUSH -1
00436067  . 68 881F4700  PUSH dump_.00471F88
0043606C  . 68 8A624300  PUSH <JMP.&msvcrt._except_handler3>
00436071  . 64:A1 0000000(MOV EAX,DWORD PTR FS:[0]
00436077  . 50           PUSH EAX
00436078  . 64:8925 00000(MOV DWORD PTR FS:[0],ESP
0043607F  . 83EC 68      SUB ESP,68
00436082  . 53           PUSH EBX
00436083  . 56           PUSH ESI
00436084  . 57           PUSH EDI
00436085  . 8965 E8      MOV DWORD PTR SS:[EBP-18],ESP
00436088  . 33DB         XOR EBX,EBX
0043608A  . 895D FC      MOV DWORD PTR SS:[EBP-4],EBX
0043608D  . 6A 02        PUSH 2
```

Рис. 24 Восстановление перемещенных байт

25 - a. Правый клик в любой точке и выбираем *Copy to executable > All modifications*
b. Нажимаем **Copy All**
c. Нажимаем "x" в верхнем правом углу и жмем **Yes**
d. Выбираем дамп ("dump_.exe") и нажимаем **Save**

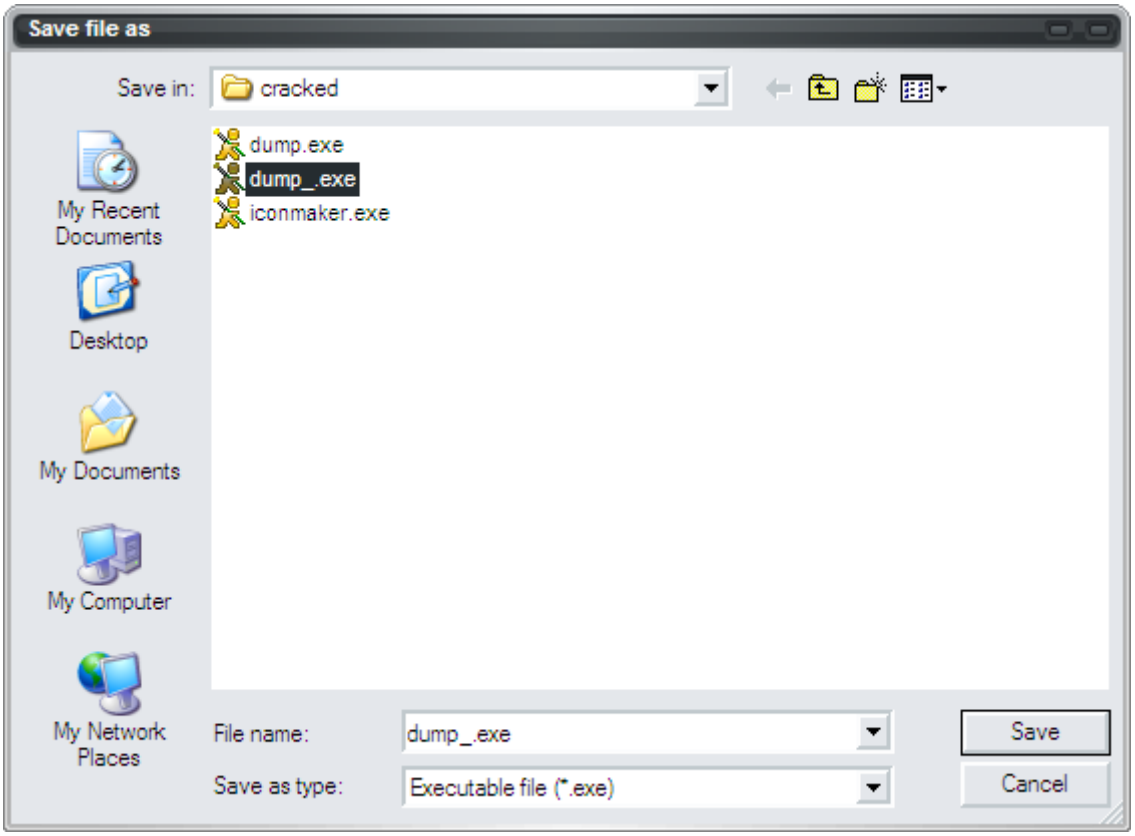


Рис. 25 Сохраняем распакованную программу

26 - a. Открываем LordPE и нажимаем **Rebuild PE**
b. Выбираем "dump_.exe" и жмем **Open**
c. Нажимаем **OK**

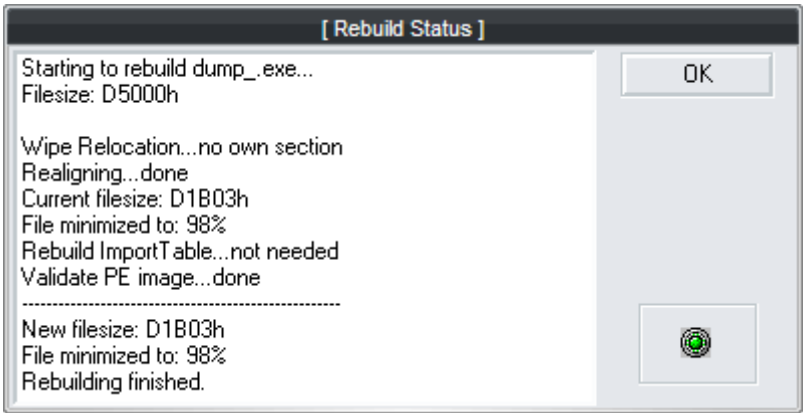
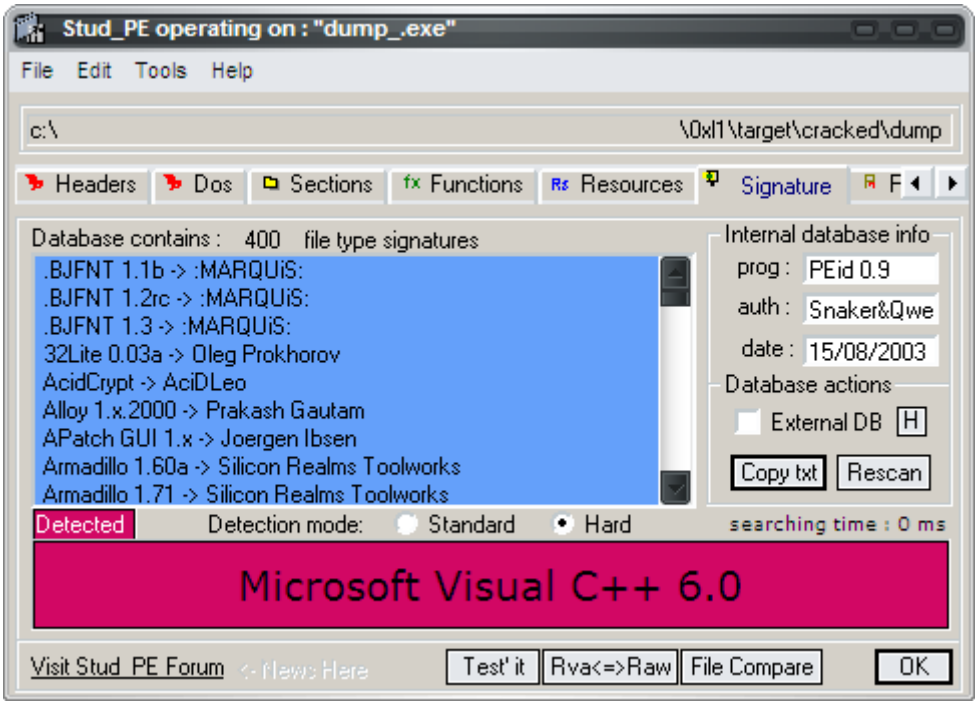
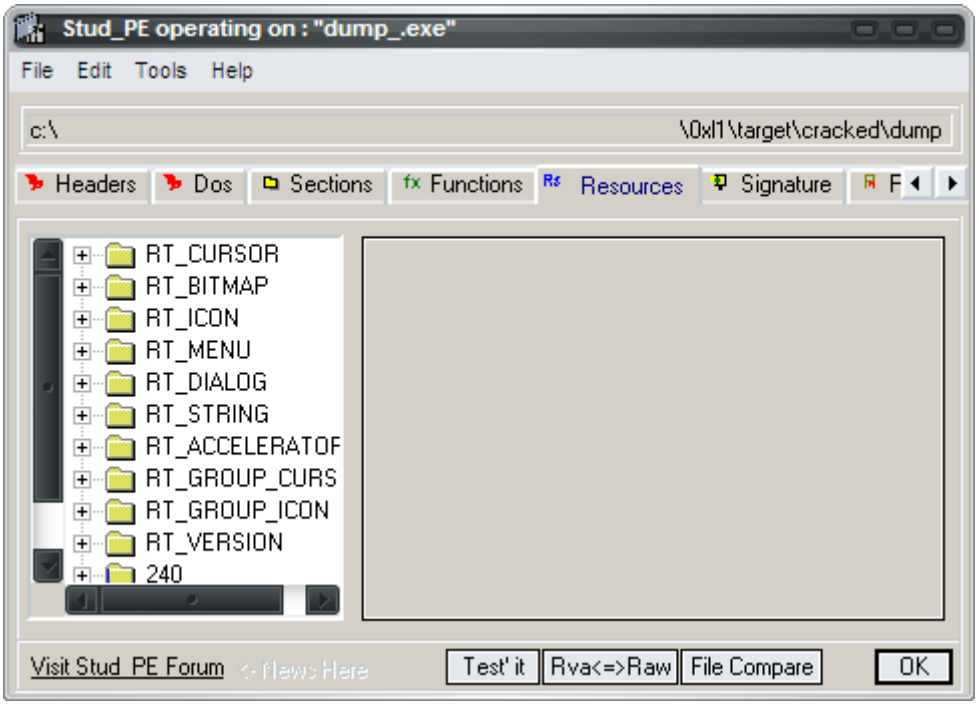


Рис. 26 Сохраняем распакованную программу

Сейчас мы загрузили программу в детектор пакеров и видим, что все отлично восстановилось :)





Neffarion
https://www.0xneff.eu/

Перевел **Svet**