Исследование StarForce

Пролог:

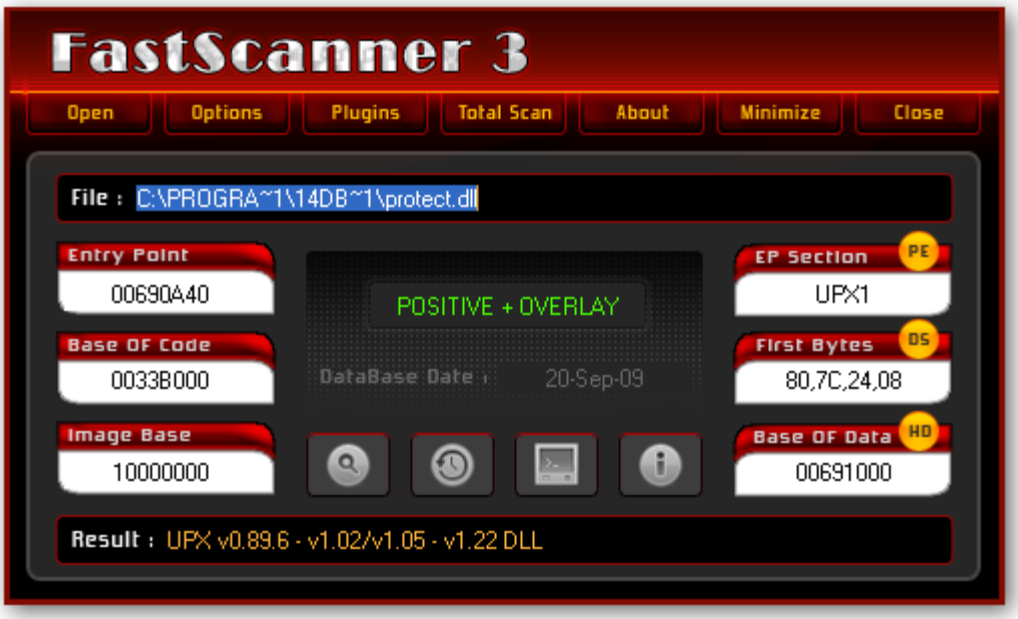
Всем приветы, спустя немного времени продолжаем тему по стару. Для начала хотелось бы немного пояснить, что писать я буду не о том как снять ту или иную версию стара в рекордные сроки, нет, мне это не интересно. Писать я буду скорее о некоторых нюансах, дырах и прочих прелестях. Я бы всё назвал это “записками”. В качестве подопытного кролика буду использовать “АнглоКУРС”. Ну что же, начнём.

Вот перечень софта, какой используются в статье:

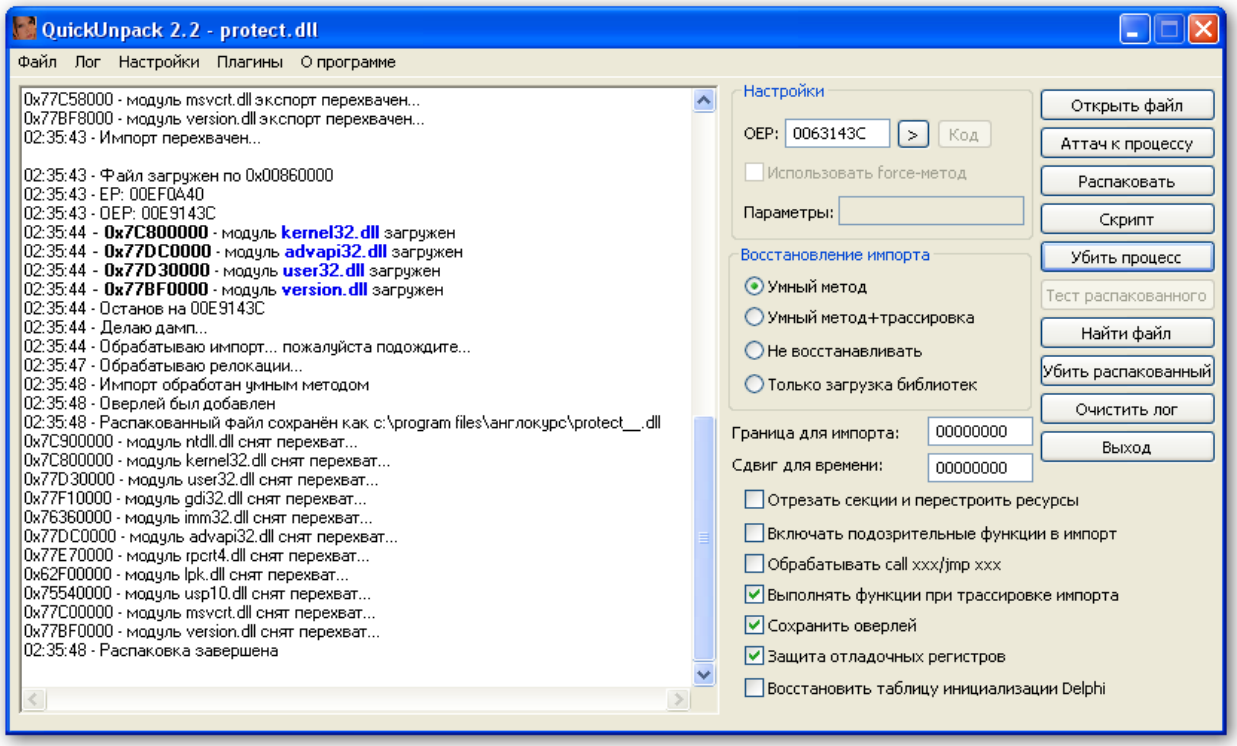
АнглоКУРС - http://cornel.biz/shop/download/Anglotrial.exe
AT4RE FastScaner - http://www.multiupload.com/AAKRHSIG33
Quick Unpack - http://qunpack.ahteam.org/?p=436
PeTools - http://petools.org.ru/petools.shtml
OllyDbg FOFF Team Edition 2.0 - http://tuts4you.com/request.php?1887

Распаковка библиотеки защиты “protect.dll” и убийство проверки CRC:

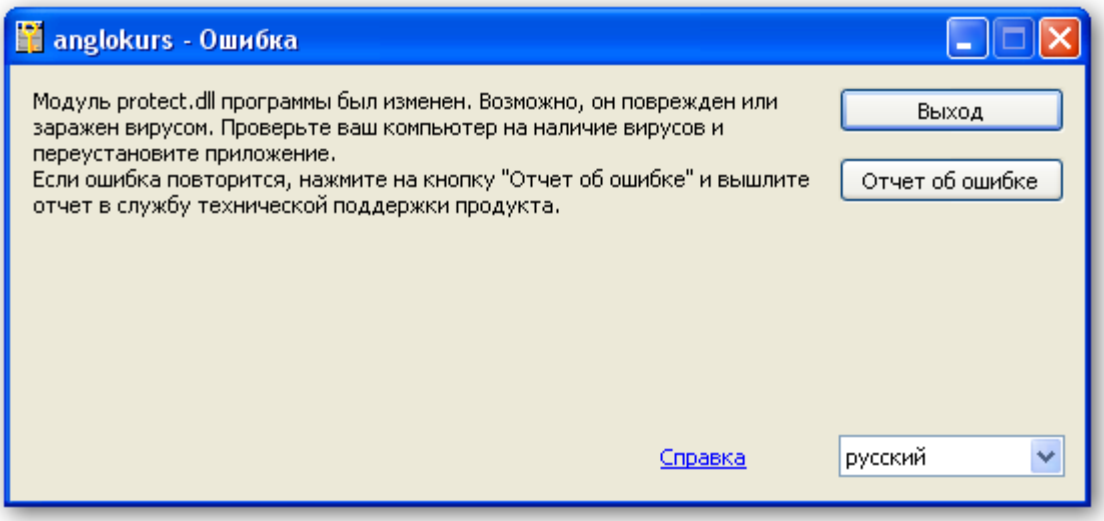
Забегая капельку вперёд хочу сразу сказать, что патчить проверку crc словно с горки скатиться - так же мгновенно и не принуждённо.
Если мы запихнём либу в какой-нибудь анализатор, ну например “PEiD” (лично я с него уже давно слез и поставил себе “AT4RE FastScanner 3”) то увидим что на ней с гордо поднятой головой сидит “UPX” (самый лучший протектор кстати :-).



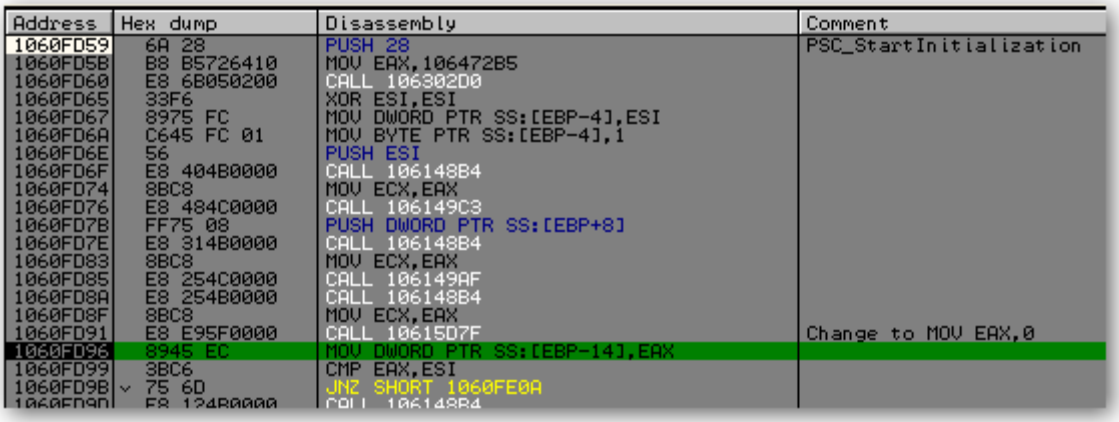
Можете распаковать его руками, я например, чтобы не тратить на это время лучше возьму “Quick Unpack”, который справится с этой задачей на ура, главное не потерять бы оверлей, для этого там есть соответствующая галка.



Окей. Анпакнули. Только вот незадача - если мы подменим оригинальную библу распакованной и запустим приложение...вылезет окно с уведомлением о том, что наш комп весь в вирях и троянах, что срочно необходимо переустановить программу. Будем исправлять такое недоразумение.



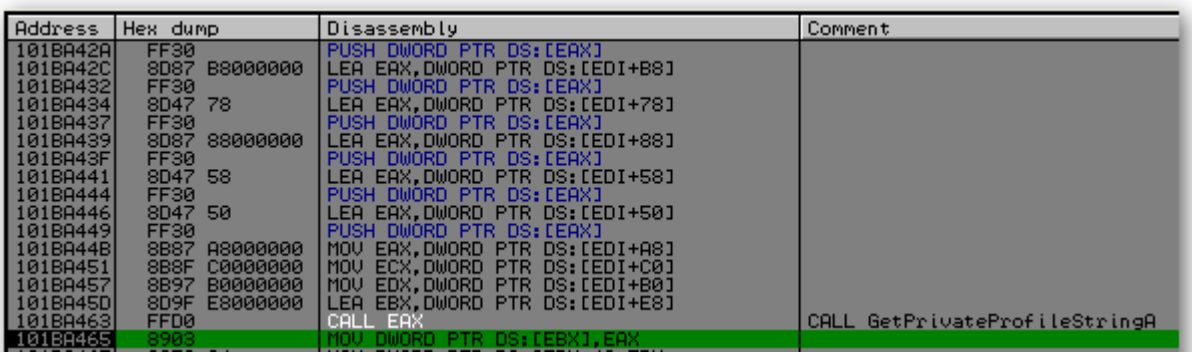
Откроем нашу библиотеку в “OllyDbg“ (думаю не надо объяснять как настраивать её). Теперь поставим бряку на экспортируемую функцию “PSC_StartInitialization” и отпустим либу в свободный полёт нажав “SHIFT-F9”.....Прервались однако. Теперь трассируем по F8 пока в eax не появится код ошибки, (а точнее после седьмого CALL’a) выглядит он примерно так: “E100????“. Теперь заменяем предыдущую инструкцию на “MOV EAX,0“ и сохраняем изменения.



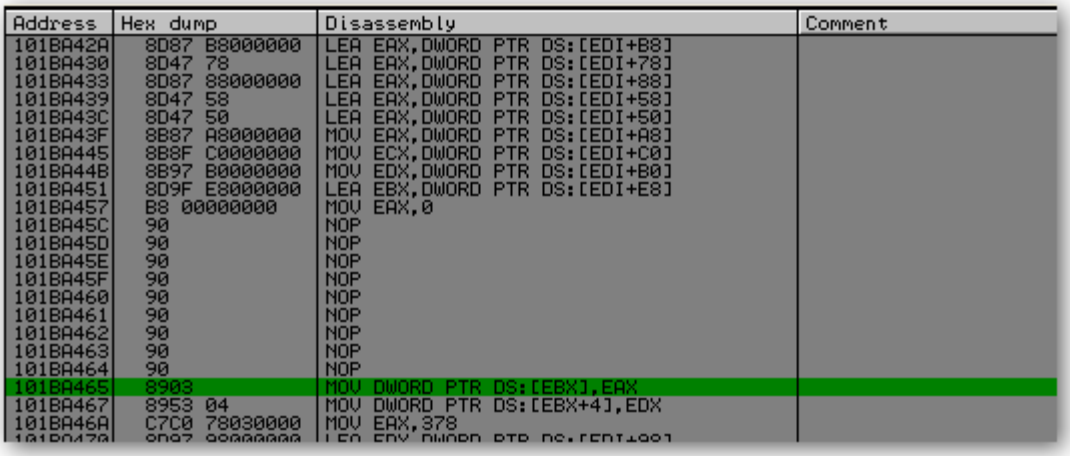
Запускаем нашу программу и видим что она вполне корректно работает и больше не ругается.

Вечный триал:

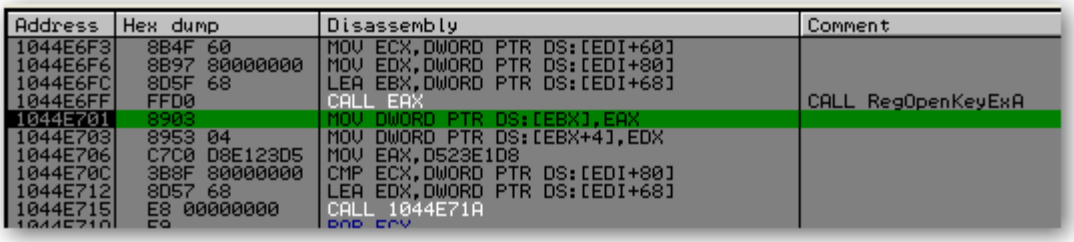
Всё чаще и чаще попадаются программы разработчики которых выбирают в роли защиты своего программного продукта “StarForce”. И всё чаще можно встретить полнофункциональный софт на котором висит ограничение по запускам или по времени работы. Давайте разберёмся как убрать эти ограничения.
Ну первое что хотелось бы заявить: работать мы будем только с “protect.dll”. Для начала нужно распаковать библиотеку (для удобства) и пропатчить проверку crc. Как это сделать вы уже знаете. Ну а теперь натравим ольку на нашу библу.
Немного посидев в отладчике было выяснено, что для хранения служебной информации старик использует реестр и файл desktop.ini который находится в документах пользователя. Давайте сначала разберёмся с файлом desktop.ini. Для этого в ольке поставим бряку на функу “GetPrivateProfileStringA” и нажмём “SHIFT-F9”. После того как мы брякнулись нажимаем “ALT-F9” (для возвращение в protect.dll). Немного поясню, сейчас нам нужно сделать так, чтобы эта функция вообще не вызывалась, а стар считал бы, что в файле “desktop.ini” ничего не записано, для этого нужно вернуть в eax ноль.



Итак. Мы сейчас стоим на следующей инструкции после “CALL EAX”, а выше идут передаваемые параметры для вызова функции. Занопим все PUSH’ы и упорядочим все инструкции по порядку так чтобы nop’ы ,были перед вызовом “GetPrivateProfileStringA”, затем занопим и его тоже и запишем инструкцию “MOV EAX,0“ (как показанно на рисунке).



Теперь сохраняем изменения и перезагружаем либу в отладчике нажав “CTRL+F2“.
Ну вот. половина дела сделана, теперь нам нужно разобраться с реестром. Нужный нам ключ (откуда стар читает инфу о триале) вызывается 4 раза: два раза до нага и 2 раза после нажатия на кнопку “Пробный период“. Т.е. нам нужно после вызова функции “RegOpenKeyExA“ записать в еах двоечку, (почему 2? ответ вы найдёте в гугле или в MSDN) при этом учесть, что после четырёх вызовов функции нам нужно вернуть всё на прежние места. Приступим.
Ставим бряку на “RegOpenKeyExA“ и отпускаем прогу нажав “SHIFT+F9”. Как только остановились жмём “ALT+F9” и попадаем в “protect.dll“

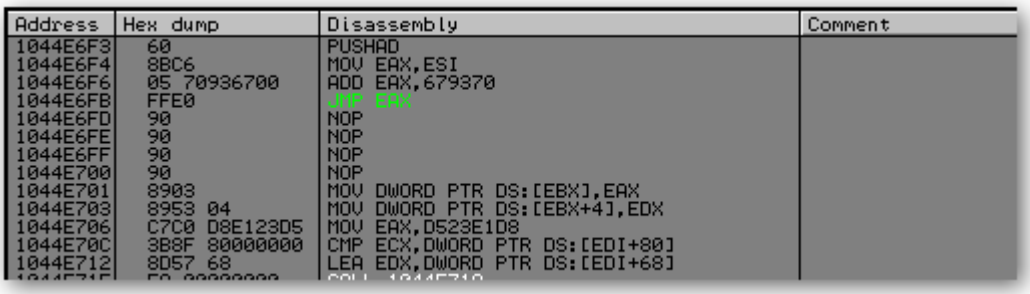


Теперь перед нами небольшая проблемка (но мы с ней справимся).
У нас нету места под патч, значит по идее нам надо поставить джамп в свободную область, но дело в том, что мы не сможем написать как обычно “JMP 1067A370“ потому как адреса будут менятся....Но тут нам приходит на помощь начальный адресс секции кода в регистре ESI, от этого и будем плясать.

Начиная с адресса “1044E6F3” запишем следующий код:

1044E6F3 PUSHAD //*Сохраняем регистры*
1044E6F4 MOV EAX,ESI *//Скопируем из регистра ESI в EAX значение "10001000"*
1044E6F6 ADD EAX,679370 *//Прибавим к EAX "679370"* (*10001000+679370"*=1067A370)
1044E6FB JMP EAX *//Прыгаем в свободное место*
1044E6FD NOP *//Очень сложный код, объяснению не поддаётся :-)*
1044E6FE NOP
1044E6FF NOP
1044E700 NOP

Должно получится как на картинке:



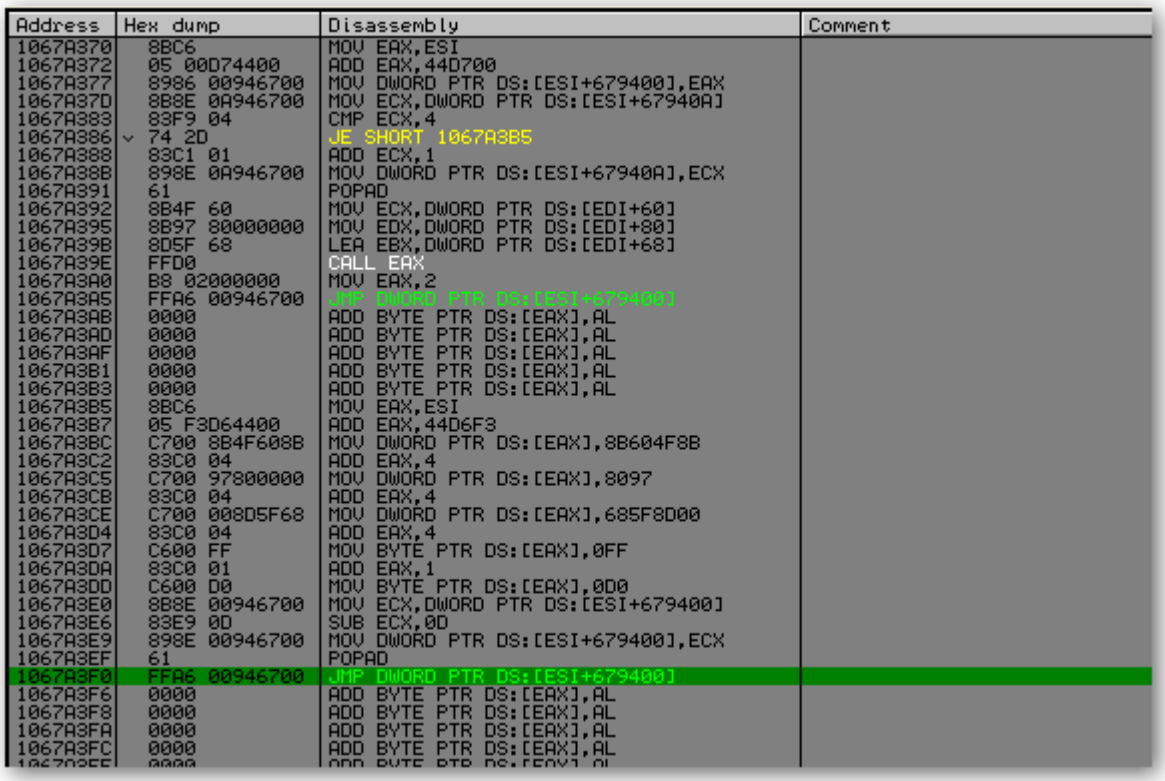
Address	Hex dump	Disassembly	Comment
1044E6F3	60	PUSHAD	
1044E6F4	8BC6	MOV EAX,ESI	
1044E6F6	05 70936700	ADD EAX,679370	
1044E6FB	FFE0	JMP EAX	
1044E6FD	90	NOP	
1044E6FE	90	NOP	
1044E6FF	90	NOP	
1044E700	90	NOP	
1044E701	8903	MOV DWORD PTR DS:[EBX],EAX	
1044E703	8953 04	MOV DWORD PTR DS:[EBX+4],EDX	
1044E706	C7C0 D8E123D5	MOV EAX,D523E1D8	
1044E70C	3B8F 80000000	CMP ECX,DWORD PTR DS:[EDI+80]	
1044E712	8D57 68	LEA EDX,DWORD PTR DS:[EDI+68]	

А по адресу "1067A370" рисуем следующие вещи:

1067A370 MOV EAX,ESI *//Скопируем из регистра ESI в EAX значение "10001000"*
1067A372 ADD EAX,44D700 *//Прибавим к EAX "679370" (10001000+44D700=1044E700)*
1067A377 MOV DWORD PTR DS:[ESI+679400],EAX *//В EAX адрес позврата, запишем его по адресу "1067A400"*
1067A37D MOV ECX,DWORD PTR DS:[ESI+67940A] *//Типа счётчик :-)*
1067A383 CMP ECX,4 *//Если в ECX лежит четыре*
1067A386 JE SHORT 1067A3B5 *//Тогда прыг*
1067A388 ADD ECX,1 *//добавим в ECX единицу*
1067A38B MOV DWORD PTR DS:[ESI+67940A],ECX //Запишем по адресу *"1067A40A" сколько раз вызванна функция "RegOpenKeyExA".*
1067A391 POPAD *//Восстановим регистры*
//Тут оригинальные байты:
1067A392 MOV ECX,DWORD PTR DS:[EDI+60]
1067A395 MOV EDX,DWORD PTR DS:[EDI+80]
1067A39B LEA EBX,DWORD PTR DS:[EDI+68]
1067A39E CALL EAX *Вызываем "RegOpenKeyExA"*
1067A3A0 MOV EAX,2 *//Запишем в EAX парашу*
1067A3A5 JMP DWORD PTR DS:[ESI+679400] *//Возвращаемся*

//Если мы попали на адресс "1067A3B5" значит "RegOpenKeyExA" будет вызванна 5 раз
1067A3B5 MOV EAX,ESI *//Скопируем из регистра ESI в EAX значение "10001000"*
1067A3B7 ADD EAX,44D6F3 *//Получаем стартовый адресс для патча - "1044E6F3"*
//Возвращаем всё на свои места:
1067A3BC MOV DWORD PTR DS:[EAX],8B604F8B

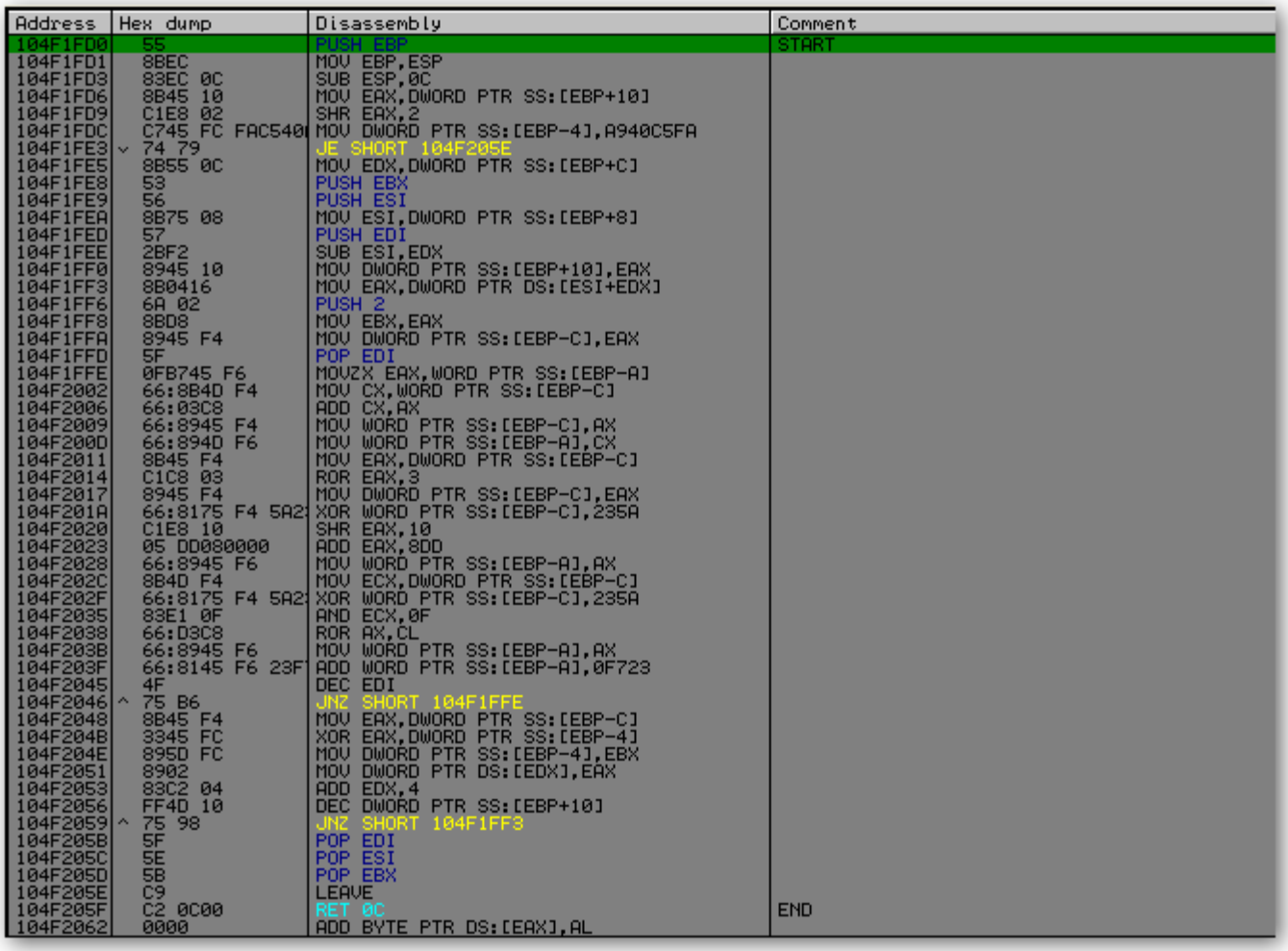
1067A3C2 ADD EAX,4
1067A3C5 MOV DWORD PTR DS:[EAX],8097
1067A3CB ADD EAX,4
1067A3CE MOV DWORD PTR DS:[EAX],685F8D00
1067A3D4 ADD EAX,4
1067A3D7 MOV BYTE PTR DS:[EAX],0FF
1067A3DA ADD EAX,1
1067A3DD MOV BYTE PTR DS:[EAX],0D0
1067A3E0 MOV ECX,DWORD PTR DS:[ESI+679400] *//Запишем в ECX адрес возврата*
1067A3E6 SUB ECX,0D *//Вычитаем, чтобы получилось "1044E6F3"*
1067A3E9 MOV DWORD PTR DS:[ESI+679400],ECX *//Записываем результат*
1067A3EF POPAD *//Восстановим регистры*
1067A3F0 JMP DWORD PTR DS:[ESI+679400] *//Возвращаемся на инструкцию "MOV ECX,DWORD PTR DS:[EDI+60]" которая теперь лежит по адресу "1044E6F3"*



Ну всё, теперь сохраним всё что мы наворотили и попробуем запустить программу.....как мы видим всё получилось, и теперь триал будет длиться вечно, поздравляю! ;-)

Расшифровка секций:

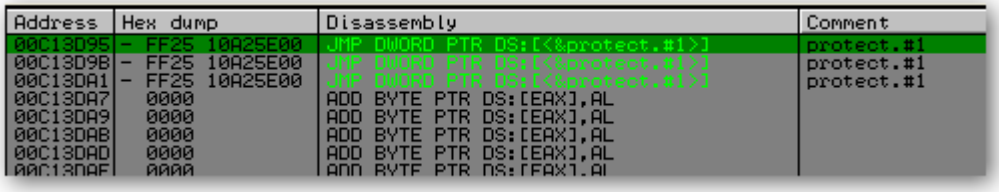
Вот захотелось мне узнать как стар расшифровывает секции, конечно наврятли это может пригодиться как мега полезная инфа, но можете например вместо “fixed exe” сделать анпакер под определённую программу - для мозга полезно бывает :-)
Сначала я думал что весь алгоритм завязан на ключе и без него никуда не попрёшь, но оказалось всё гораздо проще, хотя тут тоже есть некоторые сложности. Ну что, поехали!
Грузим распакованный “protect.dll” в ольку и ищем функцию расшифровки по следующей маске: “558BEC83EC0C8B4510C1E802C745FC“. Ну вот, попали на адресс: “104F1FD0“



Address	Hex dump	Disassembly	Comment
104F1FD0	55	PUSH EBP	START
104F1FD1	8BEC	MOV EBP,ESP	
104F1FD3	83EC 0C	SUB ESP,0C	
104F1FD6	8B45 10	MOV EAX,DWORD PTR SS:[EBP+10]	
104F1FD9	C1E8 02	SHR EAX,2	
104F1FDC	C745 FC FAC540	MOV DWORD PTR SS:[EBP-4],A940C5FA	
104F1FE3	74 79	JE SHORT 104F205E	
104F1FE5	8B55 0C	MOV EDX,DWORD PTR SS:[EBP+C]	
104F1FE8	53	PUSH EBX	
104F1FE9	56	PUSH ESI	
104F1FEA	8B75 08	MOV ESI,DWORD PTR SS:[EBP+8]	
104F1FED	57	PUSH EDI	
104F1FEE	2BF2	SUB ESI,EDX	
104F1FF0	8945 10	MOV DWORD PTR SS:[EBP+10],EAX	
104F1FF3	8B0416	MOV EAX,DWORD PTR DS:[ESI+EDX]	
104F1FF6	6A 02	PUSH 2	
104F1FF8	8BD8	MOV EBX,EAX	
104F1FFA	8945 F4	MOV DWORD PTR SS:[EBP-C],EAX	
104F1FFD	5F	POP EDI	
104F1FFE	0FB745 F6	MOVZX EAX,WORD PTR SS:[EBP-A]	
104F2002	66:8B4D F4	MOV CX,WORD PTR SS:[EBP-C]	
104F2006	66:03C8	ADD CX,AX	
104F2009	66:8945 F4	MOV WORD PTR SS:[EBP-C],AX	
104F200D	66:894D F6	MOV WORD PTR SS:[EBP-A],CX	
104F2011	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]	
104F2014	C1C8 03	ROR EAX,3	
104F2017	8945 F4	MOV DWORD PTR SS:[EBP-C],EAX	
104F201A	66:8175 F4 5A2	XOR WORD PTR SS:[EBP-C],235A	
104F2020	C1E8 10	SHR EAX,10	
104F2023	05 DD080000	ADD EAX,8DD	
104F2028	66:8945 F6	MOV WORD PTR SS:[EBP-A],AX	
104F202C	8B4D F4	MOV ECX,DWORD PTR SS:[EBP-C]	
104F202F	66:8175 F4 5A2	XOR WORD PTR SS:[EBP-C],235A	
104F2035	83E1 0F	AND ECX,0F	
104F2038	66:D3C8	ROR AX,CL	
104F203B	66:8945 F6	MOV WORD PTR SS:[EBP-A],AX	
104F203F	66:8145 F6 23F	ADD WORD PTR SS:[EBP-A],0F723	
104F2045	4F	DEC EDI	
104F2046	75 B6	JNZ SHORT 104F1FFE	
104F2048	8B45 F4	MOV EAX,DWORD PTR SS:[EBP-C]	
104F204B	3345 FC	XOR EAX,DWORD PTR SS:[EBP-4]	
104F204E	895D FC	MOV DWORD PTR SS:[EBP-4],EBX	
104F2051	8902	MOV DWORD PTR DS:[EDX],EAX	
104F2053	83C2 04	ADD EDX,4	
104F2056	FF4D 10	DEC DWORD PTR SS:[EBP+10]	
104F2059	75 98	JNZ SHORT 104F1FF3	
104F205B	5F	POP EDI	
104F205C	5E	POP ESI	
104F205D	5B	POP EBX	
104F205E	C9	LEAVE	
104F205F	C2 0C00	RET 0C	END
104F2062	0000	ADD BYTE PTR DS:[EAX],AL	

Вот собственно ОНО, код дешифруется блоками, в этом то и есть основная проблема о которой я говорил выше - узнать начало блока и его размер. этой функции передаётся три параметра: размер блока в байтах и стартовый адрес блока в RVA последний передаётся два раза. Ну адрес первого блока узнать не проблема, в основном (смотреть нужно) это всегда начальный адрес секции кода, почти всегда это: “00401000”. Давайте заглянем по этому адресу, но сначала скопипастим всю функцию в блокнот.
Закроем ольку и откроем файл “AngloKURS.exe“ в “PeTools“, чтобы библиотека не мешала нам при откладке заглянем в дирриктирию импорта и убьём там вызов

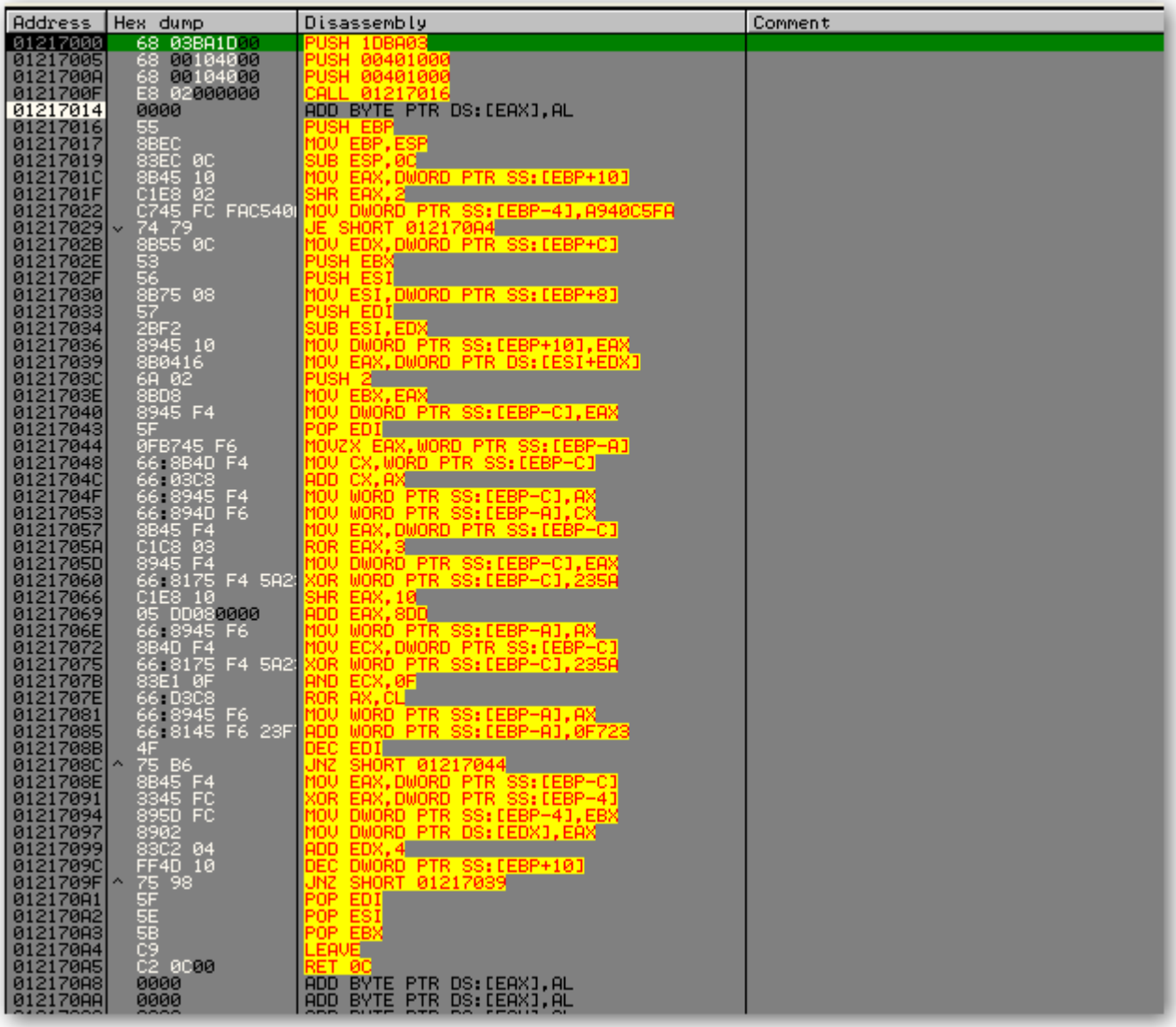
“protect.dll“. Теперь засунем прогу в ольку и перейдём по адресу “00401000” Что мы видим? Да ничего внятного, покриптованный код. Теперь нам нужно определить размер блока, для этого перейдём на EP и поссмотрим что из себя представляют переходники в вм. Как мы видим все переходы можно найти с помощью такой маски: “FF25????5E00“.



Ок. Возвращаемся назад на адресс “00401000“ И ищем переходники по маске которая указанна выше. Зачем? Да всё очень просто, блоки разделяются двумя способами: Нулями и переходниками в вм, чистым кодом (т.е. кодом не покриптованным - такого у нас нет, так как пролистав всю секцию мы не найдём ни одной валидной инструкции). Ищем и.....ничего не находим. Вери гуд! Значит блок у нас один в этой секции - это нам облегчает задачу. Теперь пролистаем в самый низ. Конец блока находится по адресу “005DCA03“, т.е. размер блока равен “1DBA03“ (005DCA03-00401000=1DBA03). Давайте же её наконец-то расшифруем, для этого нам нужно записать код декриптора в свободное место. таковое имеется в секции “.ps4” (1000 байт нам больше чем предостаточно). Открываем “Memory Map”, смотрим адрес нужной нам секции и переходим туда, а точнее на адрес “01217000“. Запишем туда вот такой код:

```
//Передаём параметры функции
01217000    PUSH 1DBA03 // Размер блока
01217005    PUSH 00401000 // Начальный адрес блока
0121700A    PUSH 00401000 // Начальный адрес блока
0121700F    CALL 01217016 // Вызываем функцию
 // Собственно сама функа (мы её копипастили в блокнот)
01217016    PUSH EBP
01217017    MOV EBP,ESP
01217019    SUB ESP,0C
0121701C    MOV EAX,DWORD PTR SS:[EBP+10]
0121701F    SHR EAX,2
01217022   MOV DWORD PTR SS:[EBP-4],A940C5FA
01217029    JE SHORT 012170A4
0121702B    MOV EDX,DWORD PTR SS:[EBP+C]
0121702E    PUSH EBX
0121702F    PUSH ESI
01217030    MOV ESI,DWORD PTR SS:[EBP+8]
01217033    PUSH EDI
01217034    SUB ESI,EDX
01217036    MOV DWORD PTR SS:[EBP+10],EAX
01217039    MOV EAX,DWORD PTR DS:[ESI+EDX]
```

```
0121703C   PUSH 2
0121703E   MOV EBX,EAX
01217040   MOV DWORD PTR SS:[EBP-C],EAX
01217043   POP EDI
01217044   MOVZX EAX,WORD PTR SS:[EBP-A]
01217048   MOV CX,WORD PTR SS:[EBP-C]
0121704C   ADD CX,AX
0121704F   MOV WORD PTR SS:[EBP-C],AX
01217053   MOV WORD PTR SS:[EBP-A],CX
01217057   MOV EAX,DWORD PTR SS:[EBP-C]
0121705A   ROR EAX,3
0121705D   MOV DWORD PTR SS:[EBP-C],EAX
01217060   XOR WORD PTR SS:[EBP-C],235A
01217066   SHR EAX,10
01217069   ADD EAX,8DD
0121706E   MOV WORD PTR SS:[EBP-A],AX
01217072   MOV ECX,DWORD PTR SS:[EBP-C]
01217075   XOR WORD PTR SS:[EBP-C],235A
0121707B   AND ECX,0F
0121707E   ROR AX,CL
01217081   MOV WORD PTR SS:[EBP-A],AX
01217085   ADD WORD PTR SS:[EBP-A],0F723
0121708B   DEC EDI
0121708C   JNZ SHORT 01217044
0121708E   MOV EAX,DWORD PTR SS:[EBP-C]
01217091   XOR EAX,DWORD PTR SS:[EBP-4]
01217094   MOV DWORD PTR SS:[EBP-4],EBX
01217097   MOV DWORD PTR DS:[EDX],EAX
01217099   ADD EDX,4
0121709C   DEC DWORD PTR SS:[EBP+10]
0121709F   JNZ SHORT 01217039
012170A1   POP EDI
012170A2   POP ESI
012170A3   POP EBX
012170A4   LEAVE
012170A5   RET 0C
```

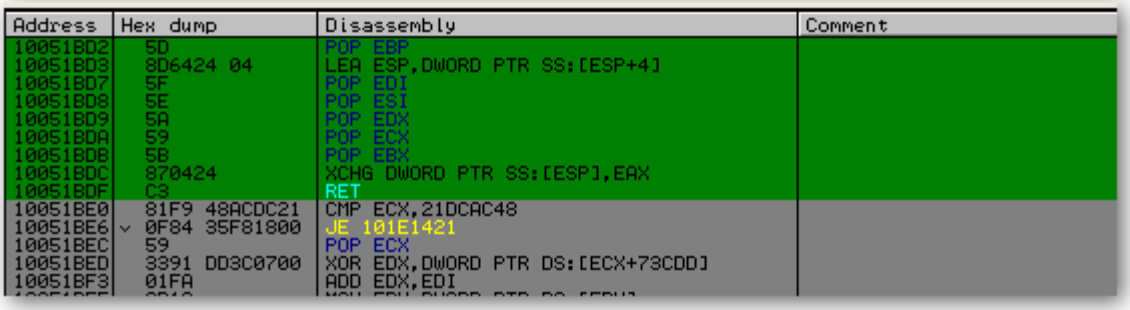


```
Address   Hex dump          Disassembly                      Comment
01217000    68 03BA1D00     PUSH 1DBA03
01217005    68 00104000     PUSH 00401000
0121700A    68 00104000     PUSH 00401000
0121700F    E8 02000000     CALL 01217016
01217014    0000            ADD BYTE PTR DS:[EAX],AL
01217016    55              PUSH EBP
01217017    8BEC            MOV EBP,ESP
01217019    83EC 0C         SUB ESP,0C
0121701C    8B45 10         MOV EAX,DWORD PTR SS:[EBP+10]
0121701F    C1E8 02         SHR EAX,2
01217022    C745 FC FAC540  MOV DWORD PTR SS:[EBP-4],A940C5FA
01217029  v 74 79           JE SHORT 012170A4
0121702B    8B55 0C         MOV EDX,DWORD PTR SS:[EBP+C]
0121702E    53              PUSH EBX
0121702F    56              PUSH ESI
01217030    8B75 08         MOV ESI,DWORD PTR SS:[EBP+8]
01217033    57              PUSH EDI
01217034    2BF2            SUB ESI,EDX
01217036    8945 10         MOV DWORD PTR SS:[EBP+10],EAX
01217039    8B0416          MOV EAX,DWORD PTR DS:[ESI+EDX]
0121703C    6A 02           PUSH 2
0121703E    8BD8            MOV EBX,EAX
01217040    8945 F4         MOV DWORD PTR SS:[EBP-C],EAX
01217043    5F              POP EDI
01217044    0FB745 F6       MOVZX EAX,WORD PTR SS:[EBP-A]
01217048    66:8B4D F4      MOV CX,WORD PTR SS:[EBP-C]
0121704C    66:03C8         ADD CX,AX
0121704F    66:8945 F4      MOV WORD PTR SS:[EBP-C],AX
01217053    66:894D F6      MOV WORD PTR SS:[EBP-A],CX
01217057    8B45 F4         MOV EAX,DWORD PTR SS:[EBP-C]
0121705A    C1C8 03         ROR EAX,3
0121705D    8945 F4         MOV DWORD PTR SS:[EBP-C],EAX
01217060    66:8175 F4 5A2  XOR WORD PTR SS:[EBP-C],235A
01217066    C1E8 10         SHR EAX,10
01217069    05 DD080000     ADD EAX,8DD
0121706E    66:8945 F6      MOV WORD PTR SS:[EBP-A],AX
01217072    8B4D F4         MOV ECX,DWORD PTR SS:[EBP-C]
01217075    66:8175 F4 5A2  XOR WORD PTR SS:[EBP-C],235A
0121707B    83E1 0F         AND ECX,0F
0121707E    66:D3C8         ROR AX,CL
01217081    66:8945 F6      MOV WORD PTR SS:[EBP-A],AX
01217085    66:8145 F6 23F  ADD WORD PTR SS:[EBP-A],0F723
0121708B    4F              DEC EDI
0121708C  ^ 75 B6           JNZ SHORT 01217044
0121708E    8B45 F4         MOV EAX,DWORD PTR SS:[EBP-C]
01217091    3345 FC         XOR EAX,DWORD PTR SS:[EBP-4]
01217094    895D FC         MOV DWORD PTR SS:[EBP-4],EBX
01217097    8902            MOV DWORD PTR DS:[EDX],EAX
01217099    83C2 04         ADD EDX,4
0121709C    FF4D 10         DEC DWORD PTR SS:[EBP+10]
0121709F  ^ 75 98           JNZ SHORT 01217039
012170A1    5F              POP EDI
012170A2    5E              POP ESI
012170A3    5B              POP EBX
012170A4    C9              LEAVE
012170A5    C2 0C00         RET 0C
012170A8    0000            ADD BYTE PTR DS:[EAX],AL
012170AA    0000            ADD BYTE PTR DS:[EAX],AL
```

Теперь зайдём в “Memory Map“ и на секцию кода посставим полный доступ (правая кнопка мышки на секции > “Set access“ > “Full access“), на всякий случай, мало ли что :-)
Далее сменим EIP на начало нашего кода, т.е. на адрес “01217000” и посставим бряку сюда: “01217014”. Всё, теперь отпускаем прогу нажав “F9”. После того как прервёмся перейдём по адресу “00401000“ и щёлкнув правой кнопкой мыши выбирем “Analysis“ > “Analyse code“ ну или просто нажмём “CTRL+A“ (для наглядности). После всех этих махинаций мы увидим, что код стал более и разборчивым и внятным.
Вот собственно и всё, думаю для начала хватит вам для изучения. В принципе я уже дописываю скрипт которых расшифровывает секции, находит блоки и т.д. и т.п., вообщем делает всё на автомате, но он ещё сырой-сырой и многое там нужно переписать, как доделаю обязательно выложу :-)

Переходники и импорт (скрипт):

В прошлой статье (которая была на ксакепе) я основное внимание уделил переходникам и написал даже для этого скрипт, только вот незадача - а как быть с импортом? Оказалось всё тоже очень просто, достаточно произвести поиск по маске: “5D8D6424045F5E5A595B870424C3” в библиотеке “protect.dll” и мы попадём в чудесное место, я бы сказал даже магическое :-)



Address	Hex dump	Disassembly	Comment
10051BD2	5D	POP EBP	
10051BD3	8D6424 04	LEA ESP,DWORD PTR SS:[ESP+4]	
10051BD7	5F	POP EDI	
10051BD8	5E	POP ESI	
10051BD9	5A	POP EDX	
10051BDA	59	POP ECX	
10051BDB	5B	POP EBX	
10051BDC	870424	XCHG DWORD PTR SS:[ESP],EAX	
10051BDF	C3	RET	
10051BE0	81F9 48ACDC21	CMP ECX,21DCAC48	
10051BE6	0F84 35F81800	JE 101E1421	
10051BEC	59	POP ECX	
10051BED	3391 DD3C0700	XOR EDX,DWORD PTR DS:[ECX+73CDD]	
10051BF3	01FA	ADD EDX,EDI	

Если после вызова переходника посставить бряку на адрес “10051BDF“ (в моём случае), т.е. на “RET” и после того как прервёмся нажать два раза “F8” то мы попадём на адрес возврата.Вот и весь секрет. Для этого я написал скрипт который я приведу ниже с подробными коментариями. Скрипт пройдётся по табличке импорта и восстановит всё что ему попадётся на глаза, но его можно легко заточить и под переходники (это вам задание). Собственно сам скриптец:

```
//Объявляем переменные
var Amount
var Modulebase
var EndCopyIat
var Ep
var IatStart
var IatEnd
var NewIAT
var DllBase
var Reg_Eax
var Reg_Ecx
var Reg_Edx
var Reg_Ebx
var Reg_Esp
var Reg_Ebp
var Reg_Esi
var Reg_Edi
var SizeIat
var Temp
var TempAlloc
```

```
@start:
//Сохраняем регистры
mov Reg_Eax, eax
mov Reg_Ecx, ecx
mov Reg_Edx, edx
mov Reg_Ebx, ebx
mov Reg_Esp, esp
mov Reg_Ebp, ebp
mov Reg_Esi, esi
mov Reg_Edi, edi
mov Ep, eip //Сохраняем OEP в переменную ep

ask "Введите ModuleBase библиотеки protect.dll" //Просим ввести ModuleBase
cmp $RESULT, 0 //Если не введено
je @exit //Тогда уходим
mov DllBase,$RESULT //Записываем результат в переменную DllBase
ask "Введите начало IAT" //Показываем диалог для ввода начала таблички импорта
cmp $RESULT, 0 //Если не введено
je @exit //Тогда уходим
mov IatStart,$RESULT //Записываем результат в переменную IatStart
ask "Введите конец IAT" //Показываем диалог для ввода конца таблички импорта
cmp $RESULT, 0 //Если не введено
je @exit //Тогда уходим
mov IatEnd, $RESULT //Записываем результат в переменную IatEnd
mov SizeIat,IatEnd-IatStart //Вычисляем размер таблички

@alloc:
//Выделяем память
alloc 1000
mov TempAlloc,$RESULT //Пишем результат в переменную tempalloc
mov NewIAT,TempAlloc //Дублируем результат записав его в переменную NewIAT
mov [TempAlloc],[IatStart],SizeIat //Копируем табличку в выделенную нами память
mov EndCopyIat,TempAlloc+SizeIat //Вычисляем последний адрес в нашей новой табличке

@search:
cmp TempAlloc, EndCopyIat //Если переменные равны
je @copy //Тогда уходим
mov eip, [TempAlloc] //Берём адрес из таблички и ставим его на eip
gmi eip, modulebase //Определяем ModuleBase
cmp DllBase, $RESULT //Если равно значит мы в protect.dll
je @run //Тогда прыгаем

mov eip, Ep //Переходим на OEP
```

```
@add:
add TempAlloc, 4 //Прибавим к текущему адрессу в табличке 4
jmp @search //Ищем дальше

@run:
gmi eip, modulebase //Определяем ModuleBase
mov Modulebase,$RESULT //Записываем результат в переменную modulebase
mov Temp, Modulebase //Записываем в переменную temp modulebase
findmem #5D8D6424045F5E5A595B870424C3#, Temp //Ищем "магическую" последовательность байт
cmp $RESULT, 0 //Если не нашли
je @exit //Тогда уходим
bp $RESULT + 0D //Ставим бряку на ret
erun //Отпускаем прогу
sti //Выходим из подпрограммы
mov [TempAlloc], [esp] //Кладём в табличку новый адресс функи
add Amount, 1 //Добавляем в переменную amount единицу (для отчётности)
bc //Снимаем все бряки
mov eip, Ep //Переходим на OEP
jmp @add //Следующий цикл

@copy:
mov [IatStart],[NewIAT],SizeIat //Копируем восстановленную табличку на место оригинальной

@exit:
free TempAlloc //Освобождаем выделенную нами ранее память
//Восстанавливаем регистры
mov eax, Reg_Eax
mov ecx, Reg_Ecx
mov edx, Reg_Edx
mov ebx, Reg_Ebx
mov esp, Reg_Esp
mov ebp, Reg_Ebp
mov esi, Reg_Esi
mov edi, Reg_Edi
ITOA Amount, 10. //Кодируем 16-ричное, в 10-ную систему исчисления
eval "Восстановленно функций - {$RESULT} штук."
msg $RESULT //Показываем сообщение о том скока фунок восстановили
ret //Завершаем работу скрипта
```

Хотелось бы немного пояснить, что от вас требуется ввести “ModuleBase” библиотеки “protect.dll”, а также начало и конец таблички импорта. Ну и запускать его разумеется нужно стоя на “OEP”.

Эпилог:

Вот в принципе и всё, что я хотел вам рассказать, думаю для начала будет достаточно ;-) В прошлой статье я писал, что сделаю разбор сталкера, но жизнь такая штука хитрая - решения и желания меняются очень часто.....Так что сорри если что-то не так :-) Надеюсь, что очень скоро порадую вас очередным материалом по данной защите.

P.S.: Глупая школота, не пишите мне в аську, на мыло и в личку, я не буду вам давать скрипты и свои наработки (всё что я хотел - я итак расскажу) и уж тем более не буду снимать вам стар. И ещё хотелось бы пояснить, что всё что написано в статье никаким образом не является плагиатом, поэтому огромная просьба понять, что если я в конце всего написанного мною выше благодарю кого-то - это ещё не значит что я у них что-то спиздил (простите за выражение).

Выражаю свою благодарность компании “Protection Technology” за создание прекрасного продукта. А так же: “**CRACKL@B.RU”**, “**UNPACK.CN**”, “**AT4RE**”, “**TSRH**”, “**SND**”, “**REVENGE CREW**”, “**FFF**”, “**RES**”, “**TRUE**”, “**ICU**”, “**AORE**“..... Ну и естественно всем,всем, всем моим единомышленникам - спасибо за помощь, советы и поддержку. До новых встреч!

Tutorial writted by Yasuichi Kitamura