**Арифметика на службе защиты**

Попалась мне тут как-то одна программа, которая требовала, как это обычно бывает в таких случаях, пароля:



Дело было на выходных, а потому настроения плотно в чем-либо рыться не было. Казалось бы, удобный случай размять пальцы в течении нескольких часов, тем более, что я всегда любил проги с текстовым режимом (как еще говорят, "досовские", хотя от доса зачастую в них ничего нет).

Запускаю td + программа... Сейчас только найти буфер, и все. Перехватить int 16 и клавишные ф-ции int 21 из td как-то сразу не получилось, потому выхожу из td и запускаю маленького резидента, схватившего эти вектора.

Хоккей. На досе резидент ничего не сказал (или я не все ф-ции ему указал), а вот по int 16 все адреса перечислил. Захожу опять в td, уже с бумажкой с этими адресами, ставлю break, но что-то мне сразу вот это не понравилось:

int 16h

jmp far ...

Через несколько секунд я понимаю, что прога работает в pm. С td особенно с pm не поработаешь. Что делать? По-честному, не хотелось воевать с прогой, которую кто-то, особенно не задумываясь, слинковал с эктсендером на чем-нибудь типа clarion'а, да и время жаль. Что ж, придется как всегда.

В softice все стало значительно проще. Написал в поле "Пароль" экзотику типа ABC123 и дал команду поиска в памяти:

s 0030:0000 L FFFFFFFF 'ABC123' ; У меня win 98

Нашлось все, только вот уже тогда для меня было тревожным рингом то, что совсем не один раз... Ладно, ставлю на найденных ребят bpm, ловлю программу на проверке пароля:

:1710:

mov bx,ds

les di,[...] ; Адрес введенного пароля -> ABC123

lds si,[...] ; Адрес эталонного пароля

repz cmpsb

Да, вот и все. В отладчике найти пароль оказалось довольно просто. Однако я уже истратил около 1,5 часа (резидент готовил) и захотелось большего, захотелось сделать "напоминатель" пароля.

Сначала я опытным путем установил, что пароль зависит от введенного INN и кода (на рисунке выше он равен 072163), более ничего на него не влияет. Несколько вариантов полученных "паролей" подтвердили мою мысль, что пароль всегда включает в себя только цифры. Вообще, пароли должны быть только из цифр. Так проще программисту. Так проще... неважно. В общем, передо мной лежало нечто довольно приятное ... что-то вроде crack me.

Итак, приступим. Стандартные шаги: вбиваю INN, ищу в памяти, собираюсь ставить на найденный буфер bpm и следить за процессом. Но что это? Этих буферов много. Их очень много! Если шарить в памяти после прохождения поля "Пароль", то их не менее 28! (Вы уж извините, что не смог до конца досчитать). В одних лишь байтах softice'а где-то в старших адресах C0XX, где обитают vxd-модуля, их гордое число - один!

Контрольных точек только четыре. Ладно, все же попробуем... Ставлю на первые четыре найденных буфера. Понеслись! Эх... Вижу, как программа копирует один в другой, другой в третий ... Да еще, зараза, норовит их в стеке разместить, поэтому после каждого освобождения под другие нужды возникают холостые брейки. Особенно обидно их видеть после команд типа push или call.

Нет, это не выход. Это слишком трудоемкий путь. Может быть, пароль? Нет, пароль - это хорошо. Но он используется только для сравнения. Как насчет значения из поля "Код"? Проделываю все те же шаги, что и при поиске обращений к ИНН. Все то же самое - куча пересылок.

Обратный реверсинг? Найти буфер с образованным паролем, и следить, как его получают? Делаю опять поиск по памяти, но уже с правильным кодом. Есть! Но опять - много.

Но тут мне немного повезло. Ввел я ИНН=1234567894, чтобы было легче искать. Отслеживая буфера, содержащие верный код, нахожу вот такой:

59497190000000000000..0 ; 16-ть цифр, для ИНН=1234567894 и кода 855907

Похоже на отформатированное большое число. Пароль тоже состоит из чисел. "Истина где-то рядом" (c) Фокс Малдер. Целенаправленно уже ищу того, кто их сделал и нахожу вот такую подпрограмму:

:6D32:

push es

mov ax,ds

...

mov cx,4

rep movsw ; ds:si -> 0,0,0,C0,45,B2,56,41...

...

...

:6DD5:

mov cx,10h

xor al,al

push cx,bx,dx,bp,si,ax

shl ah,1

rcl si,1

rcl bp,1

rcl dx,1

rcl bx,1

rcl al,1

shl ah,1

rcl si,1

rcl bp,1

rcl dx,1

rcl bx,1

rcl al,1

pop cx

add ah,ch

pop cx

add si,cx

...

shl ah,1

rcl si,1

rcl bp,1

rcl dx,1

rcl bx,1

rcl al,1

add al,30h

stosb ; Формирует как раз "594971900000..."

pop cx

loop ...

Честно говоря, я выдрал ее ;) Я выдрал ее из экзешника, выдрал и отладил, научился подавать на вход по 8-мь байт в ds:si и получать вот такие вот "числа". Вот какие данные получает эта процедура, и вот что она выдает всякий раз, когда рассчитывает пароль (всего ~32 раза):

ds:si -> 0,0,0, 0,86,58,31,41 result="1136774000..."

ds:si -> 0,0,0,80,DF,67,40,41 result="2150335000..."

ds:si -> 0,0,0,80,9F,45,4D,41 result="3836735000..."

...

ds:si -> 0,0,0,C0,41,55,52,41 result="4805895000..."

ds:si -> 0,0,0,C0,45,B2,56,41 result="5949719000..."

Я думал, что нашел ключевой код, который итерациями формирует пароль! Когда имеешь дело с криптографией, привыкаешь, что сдвиги да and-ы в огромных количествах просто так не встречаются. Первое число как-то получается из INN и Код'а - слепляется по-хитрому или что-то вроде того, думал я.

Ищу уже ссылки на парметр, передаваемый этой процедуре (в ds:si). Искал сначала внимательно, потом не очень, потом просто тупо переписывал адреса буферов (опять очень много копирований). Но все же я не прошел мимо вот этих двух процедур:

fld real8 ptr [bp+6]

fmul real8 ptr [bp+0E]

fstp real8 ptr ds:[...]

и:

fld real8 ptr [bp+6]

fadd real8 ptr [bp+0E]

fstp real8 ptr ds:[...]

Причем эти процедуры вызываются кратное число раз коду, формирующему эти длинные числа. Уже теплее. Смотрю, что делают эти парни. И вижу, что при очередном вызове, после fadd, сопроцессор выгружает те самые входные байты, которые подаются на вход процедуре создания длинных чисел... Это была просто процедура форматирования real-чисел!

Мда... Вот что значит быть избалованным сопроцессором. Если бы не эта прога, то так бы и не узнал, как выглядят числа в формате с плавающей точкой.

Теперь уже внимательнее смотрю, что же мы там умножаем и делим. Оказывается, циклически вызывается по два умножения, затем - сложение. А вот что умножается и складывается:

Шаг

1. 1x1 = 1 1x2401 = 2401 1+2401 = 2402

2. 2x2 = 4 4x2500 = 10000 2402+10000 = 12402

3. 3x3 = 9 9x2601 = 23409 23409+15986 = 35811

4. 4x4 = 16 16x2704 = 43264 43264+35811 = 79075

...

И так 42 раза. На последем как раз получается искомый код для ИНН="1234567894", только с "лишней" пятеркой впереди:

Шаг

42. 42x42 = 1746 1746x2916 = 5091336 5091336+805895 = 5949719

...

"Лишняя" пятерка сразу показывает, что не с всеми шагами все "чисто". А именно - например, на последнем шаге складывается произведение 1746x2916 не с результатом предидущего сложения (4805895), а с его частью (805895). Теперь выясняется примерный алгоритм:

SUM=1

LOOP I=1 TO 42

SUM+= I^2 x S[i]

IF I = STEP\_LEVEL THEN ... ; Откусить старшие разряды

END LOOP

; Где S[i] - добавка на каждом I-м шаге

Вот зачем нужен был код форматирования нецелых чисел! Программист не нашел ничего лучшего, чем переформатировать real-число в строку и контекстным поиском выяснять, не нужно ли отбросить старшие разряды! :)

Остается выяснить, что такое эти S[i]. Вот их список:

Шаг S[i]

1. 2401

2. 2500

3. 2601

4. 2704

...

Что же это за числа? Мы возводим индекс в квадрат, нет ли и тут чего-то похожего? Действительно:

Шаг S[i]

1. 2401 = 49^2

2. 2500 = 50^2

3. 2601 = 51^2

4. 2704 = 52^2

...

Ага, понятно, что это за числа! Да это же байты ИНН: "12345...". А что дальше? ИНН длиной 10 символов, первые 10 шагов понятны. Но далее S[i] становятся загадочными:

Шаг S[i]

11. 16641 = 129^2

12. 25600 = 160^2

13. 29241 = 171^2

14. 25600 = 160^2

...

Такие S[i] следуют какое-то время, но затем становится опять понятно их происхождение: шаги 31-36 добавляют к этой "контрольной сумме" цифры поля "код".

Тут я понял, что дело близко к развязке. Но не понял, что настолько. Первая версия - промежуточные S[i] - какие-то магические константы. Приготовился даже поискать возможный их формирователь. Для начала решил все же поискать их в файле... Да, да, они там были...

Осталось привести код remainder'а. Писать его на асм, как я обычно делаю, было как-то не прикольно. Так что извините за пас. Но и в нем я все же обошелся без real-чисел. В структуре, объявленной в программе, поясняется, зачем нужны были добавочные байты (S[i]).

{$A-}

{$N+}

{$S-}

{$D-}

const

{INN: string[11] = '0000000000';}

INN: string[11] = '7575757575'; { result must be 359683 }

TypeSetup: string[11] = 'Преферанс';

Code: string[7] = '014194';

Version: string[7] = 'Вер 46';

type

tExtPassData =

record

INN: array[1..10] of byte; { ИНН - вместе с ключом }

TypeSetup: array[1..10] of byte; { Тип конфигурации: Преферанс, etc }

Fillers: array[1..10] of byte; { Пробелы }

Code: array[1..6] of byte; { Код }

Version: array[1..6] of byte; { Версия программы: Вер 46 и т.п. }

end;

var

sum,i,PassByte: longint;

ExtPassData: tExtPassData;

begin

writeln; writeln('Preferance password remainder started!'); writeln;

{ Ask for INN & Code }

write('Enter your INN:'); readln(INN);

write('Enter your Code:'); readln(Code);

{ Make extended "password" }

fillchar(ExtPassData,sizeof(tExtPassData),' ');

move(INN[1],ExtPassData.INN,length(INN));

move(TypeSetup[1],ExtPassData.TypeSetup,length(TypeSetup));

move(Code[1],ExtPassData.Code,length(Code));

move(Version[1],ExtPassData.Version,length(Version));

{ Create control sum }

PassByte:=0;

sum:=1;

for i:=1 to sizeof(tExtPassData) do

begin

move(ExtPassData.INN[i],PassByte,1);

inc(sum,(i\*i)\*(PassByte\*PassByte));

if i > sizeof(ExtPassData.INN) then

while sum > 1000000 do dec(sum,1000000);

end;

writeln('Your password for full version is "',sum,'". Enjoy it! ;)')