сишные трюки
(1Eh выпуск)

крис касперски ака мыщъх, a.k.a. souriz, a.k.a. nezumi, no-email

борьба с хакерами на высоком сишном уровне (без ассемблерных вставок!) продолжается и сегодня мы рассмотрим технику обфускации указателей на данные и функции, продемонстрировав системно-независимые подходы — легкие в реализации, но устойчивые ко взлому

трюк #1 – обфускация указателей на данные

Дизассемблеры и отладчики поддерживают мощные механизмы реконструкции перекрестных ссылок, опутывающих всю программу, образуя своеобразный несущий каркас, на который уже навешивается все остальное. Перекрестные ссылки — это артерии, вены и автомагистрали, связывающие крошечные лоскуты кода воедино.

Допустим, у нас есть строка "wrong serial mumber" или "trial expired". Достаточно всего одного щелчка мыши, чтобы найти код, выводящий ее на экран, а следующий щелчок переносит нас в материнскую функцию, осуществляющую проверку серийного номера/срока действия программы. Чтобы воспрепятствовать анализу алгоритма достаточно ослепить механизм реконструкции перекрестных ссылок. Тогда программа распадется на ряд крошечных лоскутов, неизвестно каким образом связанных друг с другом.

Возьмем, к примеру, вариацию на тему "hello, world" (см. листинг 1):

```
char s1[] = "j.a.n.g.a.n b.e.r.u.m.a.h d.i t.e.p.i p.a.n.t.a.i j.i.k.a";
char s2[] = "do not bulild a house on near the beach if afraid of being hit by waves";

main() { MessageBox(0, s1, s2, MB_OK); }
```

Листинг 1 исходный текст незащищенной программы

А теперь посмотрим как выглядит ее дизассемблерный листинг, сгенерированный IDA-Pro (см. листинг 2):

```
.text:00401000 _main:                  ; CODE XREF: start+AF↓p
.text:00401000         push    0
.text:00401002         push    offset Caption ; "do not build a house on near the"...
.text:00401007         push    offset Text    ; "j.a.n.g.a.n   b.e.r.u.m.a.h d.i"...
.text:0040100C         push    0
.text:0040100E         call    ds:MessageBoxA
.text:00401014         retn
...
.data:00405030 ; char Text[]           ; DATA XREF: .text:00401007↑o
.data:00405030 Text    db 'j.a.n.g.a.n b.e.r.u.m.a.h d.i t.e.p.i p.a.n.t.a.i'
.data:00405030
.data:0040508C ; char Caption[]        ; DATA XREF: .text:00401002↑o
.data:0040508C Caption db 'do not bulild a house on near the beach if afraid of'
```

Листинг 2 дизассемблерный листинг незащищенной программы

Как мы видим, IDA-Pro автоматически реконструировала перекрестные ссылки на строки, упростив анализ программы до предела. Как этому помешать? Во-первых, мы должны предотвратить попадание незашифрованных указателей в код, сгенерированный компилятором. А, во-вторых — расшифровать указатели в манере, не поддерживаемой ни IDA-Pro, ни популярными отладчиками.

Первая фаза решается тривиально. Оптимизирующие компиляторы поддерживают ряд математических операций (типа сложения и вычитания), вычисляя их еще на стадии трансляции, в результате чего в код попадают зашифрованные указатели, что и демонстрируется в следующем примере (см. листинг 3):

```
#define _KEY_ 0x666999

main()
{
        char* p1 =  s1 + _KEY_;
```

```
        char* p2 =  s2 + _KEY_;

        MessageBox(0, p1 - _KEY_, p2 - _KEY_, MB_OK);
}
```

Листинг 3 очевидное, но не правильное решение

Компилируем файл, загружаем его в IDA-Pro и видим (см. листинг 4):

```
.text:00401000 _main:                  ; CODE XREF: start+AF↓p
.text:00401000         push    0
.text:00401002         push    offset Caption ; "do not bulild a house on near the"...
.text:00401007         push    offset Text    ; "j.a.n.g.a.n b.e.r.u.m.a.h d.i"...
.text:0040100C         push    0
.text:0040100E         call    ds:MessageBoxA
.text:00401014         retn
```

Листинг 4 оптимизирующие компиляторы стремятся выполнить автоматическую де-обфускацию указателей всегда, когда это только возможно

Вот так сюрприз!!! А где же наши зашифрованные указатели?! Программа какой была до обфускации такой и осталось!!! Оказывается, оптимизирующий компилятор, вычисливший значение "s1 + _KEY_" на стадии трансляции так же вычислил и значение "s1 – _KEY_", автоматически расшифровав указатель s1. Как запретить компилятору делать это? Причем, не какому-то одному отдельно взятому компилятору, а всем оптимизаторам сразу?

Очень просто! Достаточно раскрыть ANSI C и прочитать, что трансляторы не оптимизируют статические и глобальные переменные, следовательно, для достижения полученного результата, первый проход шифрования следует осуществлять с константой, а второй — с глобальной/статической переменной.

Законченный (в смысле окончательный) пример реализации приведен ниже (см. листинг 5):

```
main()
{
        char* p1 = s1 + _KEY_;
        char* p2 = s2 + _KEY_;
        static  _key_ = _KEY_;

        MessageBox(0, p1 - _key_, p2 - _key_, MB_OK);
}
```

Листинг 5 реально работающая обфускация указателей

Программа усложнилась незначительно, зато результат превзошел все ожидания:

```
.text:00401000 _main   proc near              ; CODE XREF: start+AF↓p
.text:00401000         mov     eax, dword_4050D8      ; _key_
.text:00401005         mov     ecx, 0A6BA25h          ; указатель на s1 (зашифрованный)
.text:0040100A         mov     edx, 0A6B9C9h          ; указатель на s2 (зашифрованный)
.text:0040100F         sub     ecx, eax
.text:00401011         push    0                      ; uType
.text:00401013         sub     edx, eax
.text:00401015         push    ecx                    ; lpCaption
.text:00401016         push    edx                    ; lpText
.text:00401017         push    0                      ; hWnd
.text:00401019         call    ds:MessageBoxA
.text:0040101F         retn
.text:0040101F _main   endp
…
.data:00405030         db  6Ah ; j                    ; начало строки s1
.data:00405031         db  2Eh ; .
.data:00405032         db  61h ; a
.data:00405033         db  2Eh ; .
.data:00405034         db  6Eh ; n
.data:00405035         db  2Eh ; .
.data:00405036         db  67h ; g
...
.data:0040508C         db  64h ; d                    ; начало строки s2
.data:0040508D         db  6Fh ; o
.data:0040508E         db  20h
.data:0040508F         db  6Eh ; n
.data:00405090         db  6Fh ; o
.data:00405091         db  74h ; t
```

Листинг 6 дизассемблерный код программы, шифрующей указатели на данные

IDA-Pro не только не реконструировала перекрестные ссылки, но и распознала указатели на s1 и s2, оставив их в зашифрованном виде и хотя расшифровать значение указатель вполне возможно (достаточно проанализировать дизассемблерный код), на это уходит время и кроме того, все средства для постройки графов тушатся на корню. И все это достигается без применения ассемблерных вставок и прочих нестандартных извращений.

трюк #2 – обфускация указателей на функции

Зашифровать указатели на функции намного сложнее, поскольку, оптимизаторы не поддерживают математических преобразований над ними, а не поддерживают они их потому, что их не поддерживает стандарт, позволяющий законными средствами получить указатель на функцию, но сужающий меню доступных действий только до присвоения нуля, что, естественно, не входит в наши планы.

Запрет на математические преобразования легко обходится кастингом. В частности, 32-битные операционные системы (Windows 9x/NT, Linux, FreeBSD) используют плоскую модель адресного пространства и 32-битные указатели на код, с которыми можно оперировать так же, как и с целочисленным типом DWORD (unsigned int). В других случаях разрядность указателя может отличаться от обозначенной, более того, он вообще может представлять собой сложную структуру состоящую из селектора и смещения, а потому кастинг это уже хак! Но этот хак работает!!! Главное — вынести физический тип указателя на код в отдельный define, зависящий от платформы.

Кастинг снимает защиту на математические операции с указателями на функции, но все преобразования выполняются на стадии выполнения программы, а вовсе не на стадии компиляции, в результате чего, указатели "благополучно" переживают оптимизацию, попадая в машинный код целевого файла, где их распознает IDA-Pro вместе с отладчиками.

Проблема кажется неразрешимой, но… кто нам мешает доработать откомпилированный код уже после трансляции, зашифровав указатели непосредственно в двоичном файле, выполняя расшифровку уже в самой программе?! Вся проблема в том — как найти указатели в откомпилированном коде? Самое простое — загнать указатели в структуру, предваренную специальным маркером — текстовой строкой или константой с уникальным содержимым, после чего нам остается только найти этот маркер в программе и зашифровать следующие за ним указатели, что можно сделать как вручную в HIEW'e, там и автоматически с помощью несложной программы.

Но довольно слов, побольше дела, в смысле кода наглядных примеров (см. листинг 7):

```
#define p DWORD
#define _KEY_ 0x66666666

baz(char* s1, char* s2){ MessageBox(0, s1, s2, MB_OK); }

struct FF
{
        p am;   // <<-- marker
        p f1;   // <<-- list of func. pointers
}       ff = { 0xEFBEADDE, (p) &baz};

main()
{

        char* p1 = s1 + _KEY_;
        char* p2 = s2 + _KEY_;
        static  _key_ = _KEY_;

        int (*foo)(char*, char*);
        foo = (int (*)(char*, char*)) (ff.f1 ^ pk);

        foo((char*) p1 - pk, (char*) p2 + pk);
}
```

Листинг 7 обфускация указателей на функции

После компиляции программы мы должны найти в исполняемом файле "магическую" последовательность 0xDEADBEEF, наложив на следующее за ней двойное слово ключ шифрования 0x66666666 по XOR. Убедившись, что все выполнено правильно и программа работает, а не падает, загружаем ее в дизассемблер (см. листинг 8):

```
.text:00401020 _main   proc near                    ; CODE XREF: start+AF↓p
.text:00401020         mov     eax, dword_405050    ; _key_
.text:00401025         mov     edx, 66A6B696h       ; указатель на s1 (зашифрованный)
.text:0040102A         sub     edx, eax
.text:0040102C         lea     ecx, [eax-6626162Eh] ; указатель на s2 (зашифрованный)
.text:00401032         push    ecx
.text:00401033         mov     ecx, dword_40504C    ; ff
.text:00401039         push    edx
.text:0040103A         xor     eax, ecx
.text:0040103C         call    eax                  ; foo
.text:0040103E         add     esp, 8
.text:00401041         retn
...
.data:0040504C dword_40504C   dd      66267666h     ; зашифрованный указатель на foo
```

Листинг 8 убийственный результат обфускации указателей на код и данные — полный хаос!

Теперь сам черт не разберет что это за код и какого он делает! Да, конечно, при прогоне программы под отладчиком (или плагином эмулятором для IDA-Pro) хакер узнает значение регистра EAX, определив какая функция тут вызывается. Но… наглядность дизассемблерного листинга необратимо утеряна. Механизмы реконструкции потока управления тихо курят в сторонке, высаживая хакера на измену и увеличивая время анализа программы на порядок-другой.