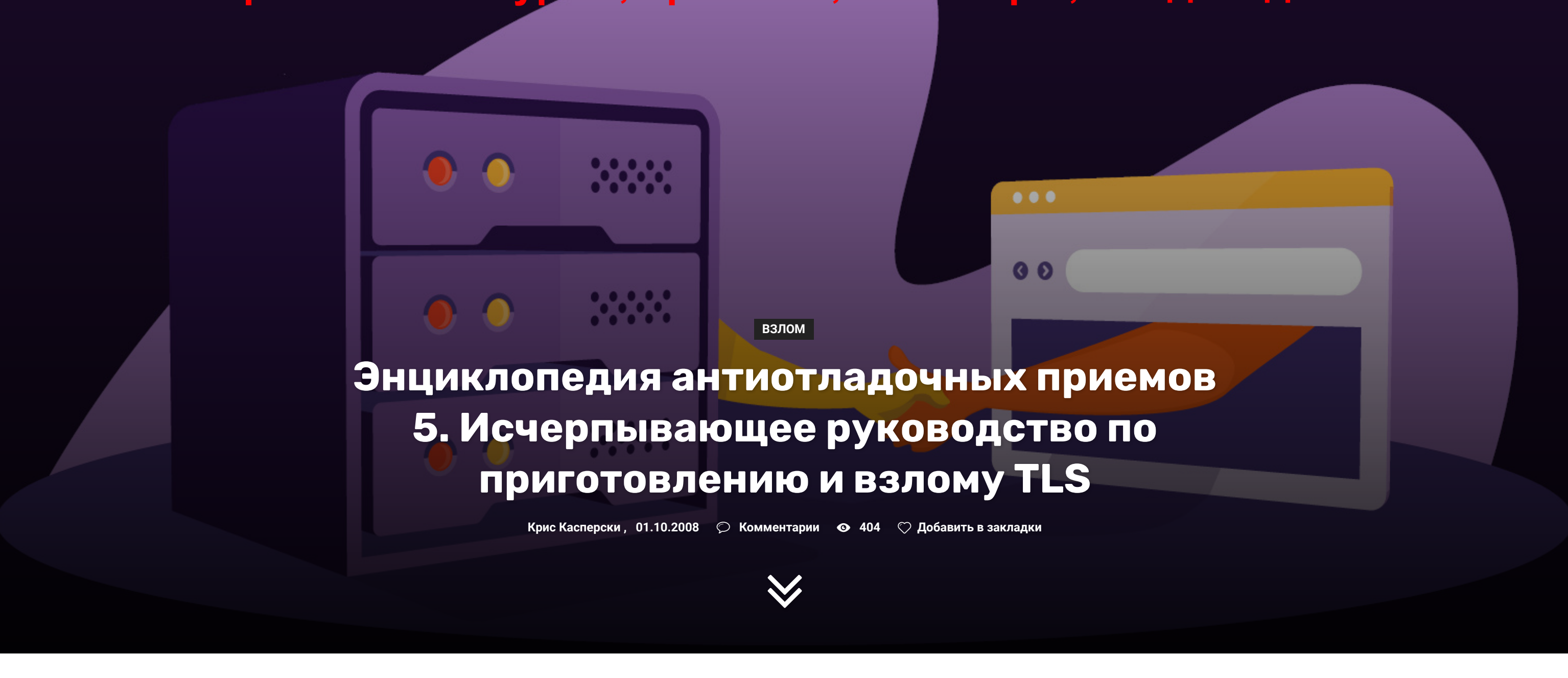


ВЗЛОМ

Энциклопедия антиотладочных приемов 5. Исчерпывающее руководство по приготовлению и взлому TLS

Крис Касперски , 01.10.2008 Комментарии 404 Добавить в закладки

Содержание статьи

Загадочная аббревиатура TLS таит в себе много секретов. Это — мощнейшее оружие против отладчиков и дизассемблеров. В комбинации с упаковщиками TLS превращается в гремучую смесь термоядерного типа.

Что же такое TLS и чем оно грозит хакерам? Начнем издалека. Популярные языки программирования (в том числе C) поддерживают статические и глобальные переменные, использование которых делает код потоконебезопасным. Все потоки разделяют один и тот же набор глобальных/статических переменных, порождая путаницу и хаос. Поток A положил в переменную foo значение X и только хотел прочитать его обратно, как внезапно пробудившийся поток B записал в foo значение Y, что оказалось для A полной неожиданностью.

Microsoft разработали специальный механизм, именуемый локальной памятью потока (Thread Local Storage, или сокращенно TLS), предоставляющий в распоряжение потоков индивидуальные наборы глобальных/статических переменных. TLS поддерживается как на уровне явно вызываемых API-функций (TLSAlloc, TLSFree, TLSSetValue, TLSGetValue), так и на уровне PE-формата, неявно обрабатываемого системным загрузчиком. PE-формат поддерживает функции обратного вызова (TLS-callback), автоматически вызываемые системой до передачи управления на точку входа. В частности, это позволяет определить наличие отладчика или скрытно выполнить некоторые действия. Системный загрузчик также записывает TLS-индекс в заданную локацию — отличный способ неявной самомодификации программы. Дизассемблерами она не отлавливается и заводит хакера в тупик.

TLS используется в большом количестве протекторов, защит, вирусов, crackme и прочих программ, взлом которых описан в куче различных туторалов. Однако изложение обычно носит поверхностный характер — целостной картины после прочтения не создается. Попробуем это исправить.

WWW

Обязательно изучи спецификацию PE от Microsoft и слей «крякми» с моего сайта.

Требуется помощь читателей!

К сожалению, файл buckme-crackme.zip не был выложен на DVD «Хакера», а сайт Криса больше не доступен. Если у тебя сохранился этот файл, пожалуйста, пришли его в редакцию на адрес content@glc.ru.

FUNDAMENTALS

Прежде всего нам понадобится спецификация PE-формата, последнюю версию которого (представленную в виде XML) можно утянуть прямо из-под носа Microsoft. Тот же самый файл, только конвертированный в MS Word 2000, я выложил на своем сервере.

TLS-таблица описывается девятым (считая от нуля) четвертным словом в Optional Header Data Directories. Первое двойное слово хранит в себе RVA-адрес TLS-таблицы. Второе — ее размер, который игнорируется всеми известными мне операционными системами. Поэтому здесь можно писать что угодно, хоть 0, хоть FFFFFFFFh. Дизассемблерам это крышу не срывает, во всяком случае IDA-Pro, Olly и даже примитивный DUMPBIN работают как ни в чем не бывало. А вот проверка валидности размера TLS-таблицы может появиться в любой момент, так что лучше не прикалываться и писать то, что нужно.

TLS-таблица может находиться в любой секции с атрибутами [`IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ | IMAGE_SCN_MEM_WRITE`] (например, в секции данных). Некоторые линкеры помещают TLS-таблицу в специальную секцию .TLS или .TLS$ — это делается из чисто эстетических соображений. Системный загрузчик не проверяет имя секции. Правда, некоторые упаковщики не обрабатывают TLS, расположенные вне секции .TLS, но это уже их личные проблемы. Тем более что ряд упаковщиков, что такое TLS, не знает вообще.

смещение (PE32/PE32+)	размер (PE32/PE32+)	поле	описание
0	4/8	Raw Data Start VA	полный виртуальный адрес (VA, не RVA) первого байта локальной памяти потока, если PE-файл перемещаем, то данный VA-адрес должен быть обозначен в таблице фиксапов.
4/8	4/8	Raw Data End VA	полный виртуальный адрес последнего байта локальной памяти потока за вычетом заполняющих нулей (см. "Size of Zero Fill")
8/16	4/8	Address of Index	полный виртуальный адрес TLS-индекса, назначаемого системного загрузчиком и записываемого в заданную локацию, расположенной в любой области памяти, доступной на запись
12/24	4/8	Address of Callbacks	полный виртуальный адрес массива функций обратного вызова, завершаемого нулем;
16/32	4	Size of Zero Fill	количество нулевых байтов, которые системный загрузчик должен дописать к концу блока данных локальной памяти потока;
20/36	4	Characteristics	зарезервировано

Формат TLS-таблицы для файлов PE32/PE32+

Функции обратного вызова вызываются системным загрузчиком при инициализации/терминации процесса, а также при создании/завершении потока. Они имеют тот же самый прототип, что и DllMain:

Прототип функций обратного вызова

```
typedef VOID (NTAPI *PIMAGE_TLS_CALLBACK) (
    PVOID DllHandle,     // Дескриптор модуля
    DWORD Reason,        // Причина вызова
    PVOID Reserved       // Зарезервировано
);
```

Двойное слово Reason, информируя функцию обратного вызова, по какой причине она была вызвана, принимает следующие значения.

define	#	описание
DLL_PROCESS_ATTACH	1	сейчас будет запущен новый процесс
DLL_THREAD_ATTACH	2	сейчас будет запущен новый поток
DLL_THREAD_DETACH	3	поток сейчас будет завершен
DLL_PROCESS_DETACH	0	процесс сейчас будет завершен

Возможные значения параметра Reason

С функциями обратного вызова все понятно. Системный загрузчик просто вызывает их одну за другой, игнорируя возвращаемые значения и даже не требуя очистки аргументов из стека, — красота!

А вот с TLS-индексом все чуть-чуть сложнее. Двойное слово по адресу `FS:[2Ch]` указывает на TLS-массив, содержащий данные локальной памяти потока для всех модулей. Чтобы не возникало путаницы, системный загрузчик при инициации модуля записывает по адресу Address of Index индекс данного модуля. То есть локальная память потока находится по адресу `FS:[2Ch][index*4]`. Теоретически index может принимать любые значения, известные только одной операционной системе, но практически он равен нулю для первого модуля и увеличивается на единицу для всех последующих. Если наш файл не загружает никаких DLL, использующих TLS, индекс будет равен нулю (с высокой степенью вероятности, но без всяких гарантий). Как же его можно использовать на практике? Самое надежное — записать в секцию данных число типа 12345678h и натравить на него индекс. После инициализации приложения мы получим что-то «отличное от» — и дизассемблеры это не засекут!

Теоретическую часть будем считать законченной, приступим к практическим занятиям.

РУЧНОЕ СОЗДАНИЕ TLS

Для работы с TLS нам необходим компилятор и линкер, поддерживающий обозначенную технологию. Недостатка в таковых нет, хотя нет и полноценной поддержки TLS. Возможно, мы захотим прикрутить TLS к уже упакованной/запротекченной программе, следовательно, нам жизненно необходимо научиться создавать его руками. В случае EXE с убитыми фиксами это очень просто. С DLL уже будет сложнее, так как придется править таблицу перемещаемых элементов. Тут тоже есть свои хитрости и трюки… но сначала — EXE.

Пишем простую программу типа «Hello, world!». Компилируем ее и открываем полученный файл в HIEW. Идем в начало секции .data (по Enter переходим в hex-режим, F8 — для вызова PE-заголовка, F6 — Object Table, подводим курсор к .data и жмем Enter). Пропускаем инициализированные данные, подгоняя курсор к адресу .406100h (в другом случае адрес может быть иным), где и пишем такую магическую последовательность:

```
0 61 40 00 | 20 61 40 00 | 30 61 40 00 | 60 61 40 00
```

Ладно, на самом деле она никакая не магическая. Первая пара двойных слов означает начало/конец блока данных локальной памяти потока, который может находиться в любой области памяти, доступной на чтение. Третье двойное слово — адрес двойного слова, куда загрузчик запишет TLS-индекс. В нашем случае это 00406130h, где мы в HIEW ставим 66666666h (чтобы убедиться, что загрузчик действительно перезаписывает это значение). Последнее двойное слово — указатель на таблицу функций обратного вызова, расположенную по адресу 00406160h и содержащую указатель на единственный callback по адресу 00406190h, за которым следует ноль (указывающий, что других callback'ов здесь нет и не предвидится).

Что же касается самого callback'а, то, подогнав курсор к адресу 00406190h, легким нажатием Enter'а мы переходим в режим ассемблера. А тут пишем `DEC D,[00406140]`, Enter, RET. После чего сохраняем изменения по F9 и выходим, предварительно полюбовавшись на результат нашей работы (см. листинг 2). Ну а кому лень возиться с HIEW, может воспользоваться готовым файлом hello-TLS.exe.

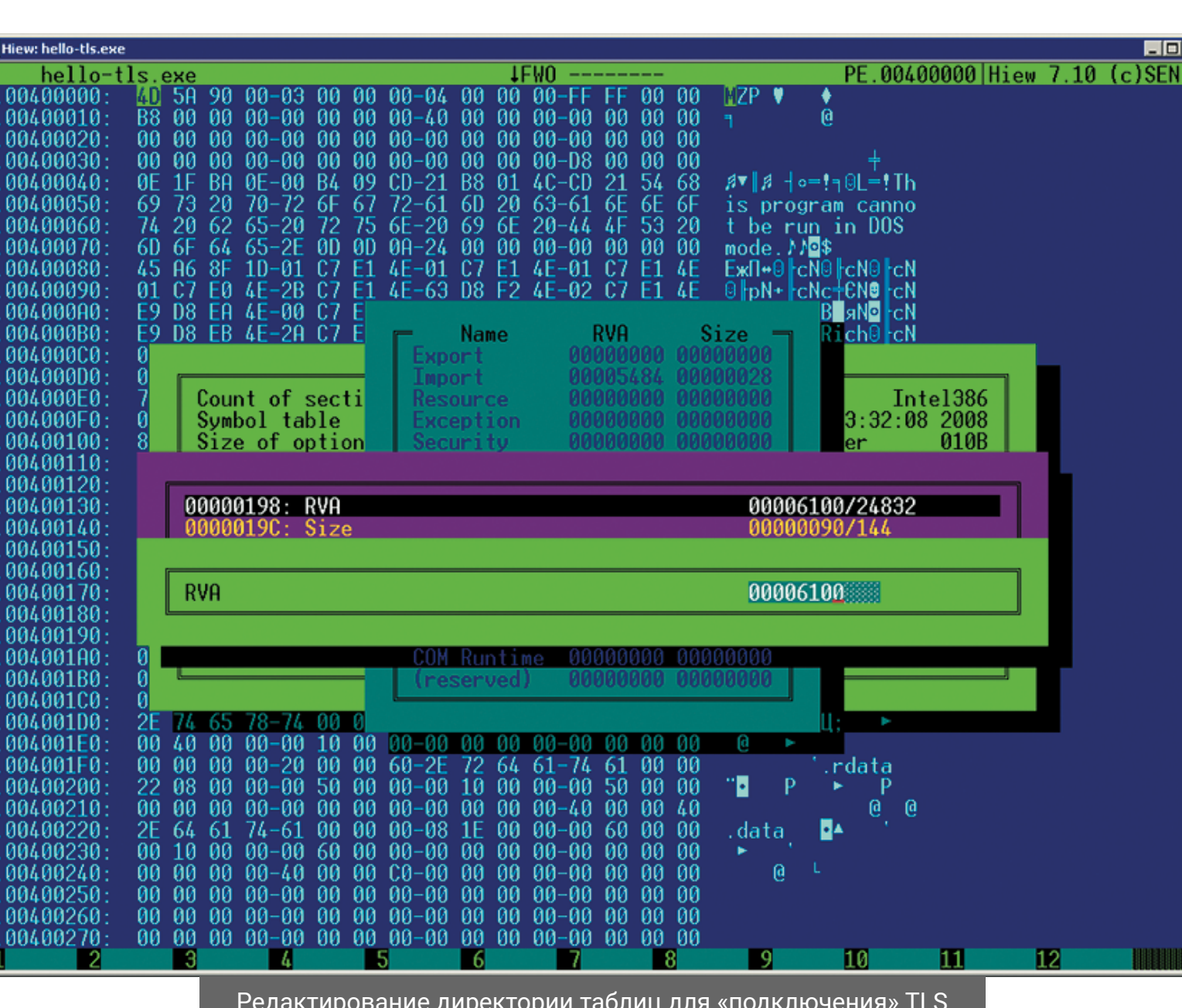
TLS, созданный вручную (hex-дамп)

```
.00406100: 10 61 40 00-20 61 40 00-30 61 40 00-60 61 40 00  >a@  a@ 0a@ `a@
.00406110: 00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
.00406120: 00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
.00406130: 66 66 66 66-00 00 00 00-00 00 00 00-00 00 00 00
.00406140: 00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
.00406150: 00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
.00406160: 90 61 40 00-00 00 00 00-00 00 00 00-00 00 00 00  Pa@
.00406170: 00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
.00406180: 00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
.00406190: FF 0D 40 61-40 00 C3 00-00 00 00 00-00 00 00 00   d@a@ +
```

INFO

Исключения, возникающие внутри TLS callback'ов, давятся системой на автомате. Но зато отлавливаются отладчиками (той же «Ольгой»). При этом хакер не понимает, как это может работать. Что тут думать — надо жать Shift-F9 для передачи управления на точку входа!

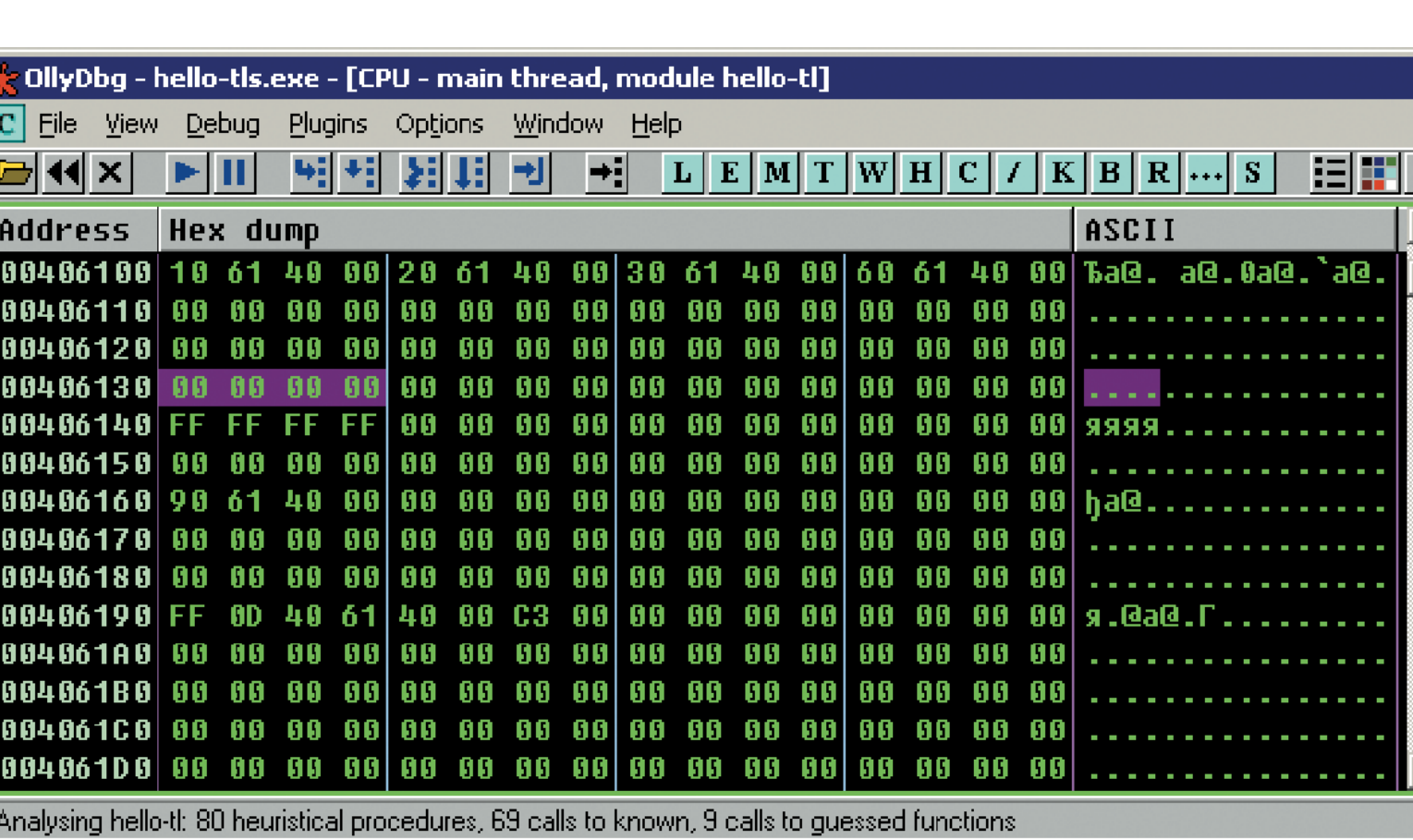
Остается только занести TLS в таблицу директорий. В HIEW это делается так: открываем файл, переходим в hex-режим, давим F8 для вызова PE-заголовка, а следом — F10 для вызова директории таблиц. Подгоняем курсор к TLS и редактируем его по F3, вводя RVA-адрес начала TLS-таблицы (в нашем случае — 6100h) и размер (можно брать любой).



Редактирование директории таблиц для «подключения» TLS

БОЕВОЕ КРЕЩЕНИЕ

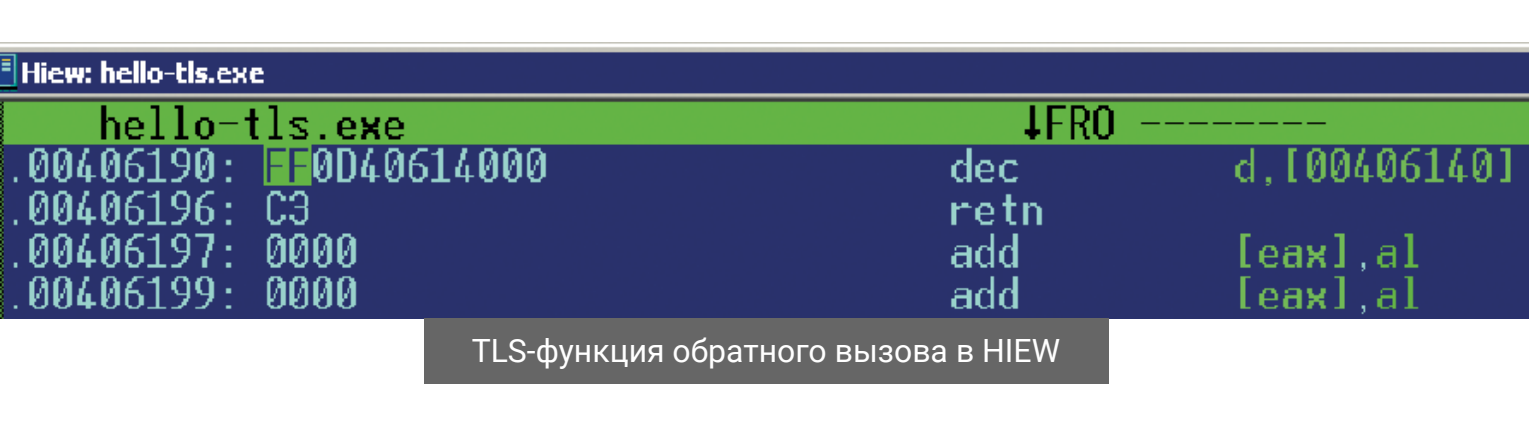
Загружаем hello-TLS.exe в отладчик (например, в «Ольгу») и ходим по адресу 00406100h. Мы четко видим, что двойное слово 66666666h по адресу 00406130h мистическим образом обратилось в ноль, зато нулевое двойное слово по адресу 00406140h, уменьшившись на единицу, превратилось в FFFFFFFFh — результат записи индекса и вызова callback'а, соответственно. Это произошло до того, как мы успели выполнить хотя бы одну команду, стоя в точке входа.



Результат работы рукотворного TLS

КАК ЭТО ЛОМАЮТ

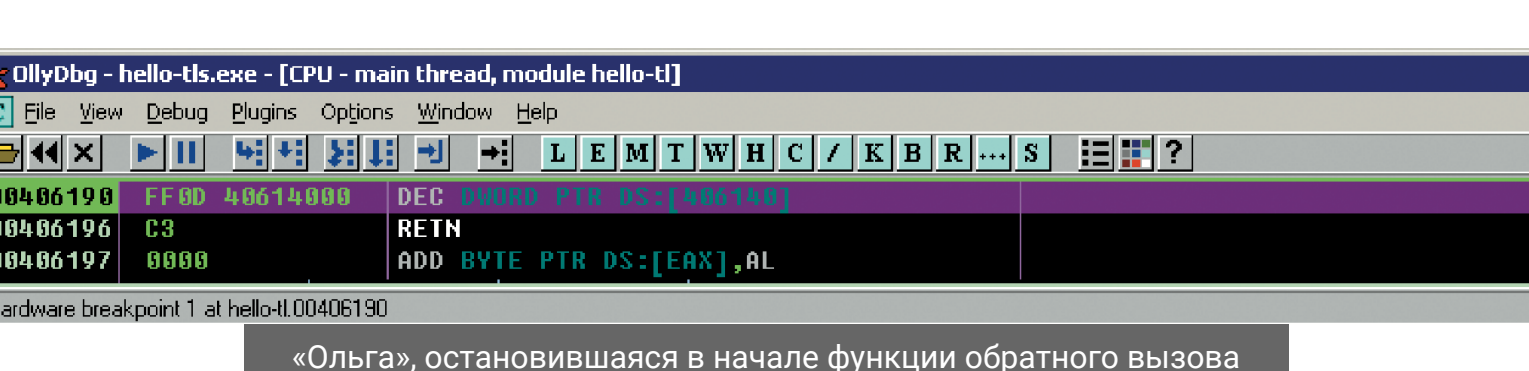
Существует множество плагинов для «Ольги», автоматически стопящихся в начале TLS. Но, во-первых, большинство из них не умеют обрабатывать больше одного callback'а, а во-вторых, мы, хакеры, должны все делать своими руками. Короче, зовем на помощь HIEW. Открываем файл и по F10 зовем директорию таблиц, как уже описывалось выше. Видим там TLS и что RVA-адрес не равен нулю. Ага! Значит, тут есть TLS! Подгоняем курсор к строке TLS и переносимся туда по Enter'у. Четвертое (считая от одного) двойное слово — указатель на таблицу функций обратного вызова. Смотрим, что у нас здесь. А здесь у нас 00406190h. Переходим по обозначенному адресу и жмем Enter, переключая HIEW в режим дизассемблера. Изучаем callback, попутно запоминая его адрес. Он нам понадобится чуть позже.



TLS-функция обратного вызова в HIEW

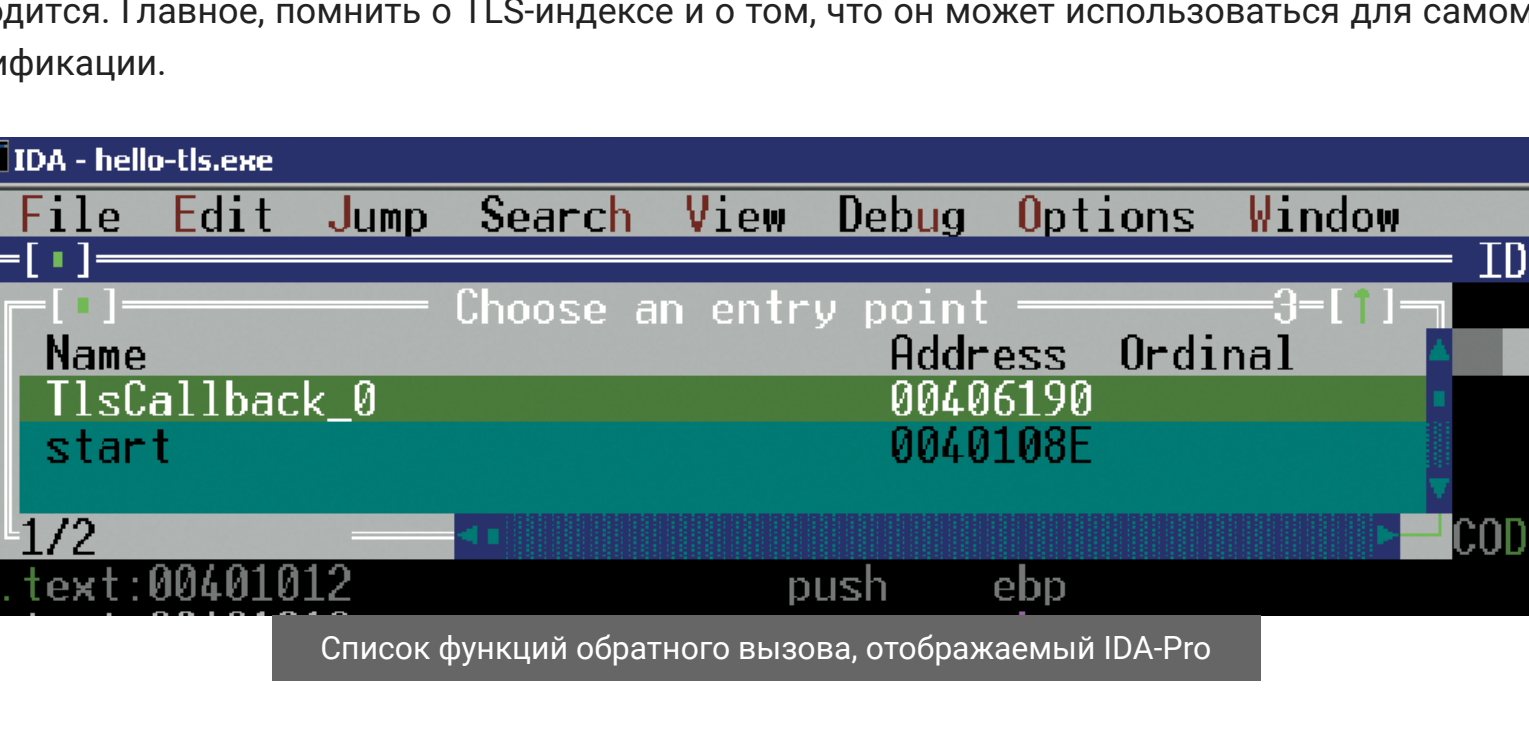
Мы снова в «Ольге». И снова TLS callback отработал еще до завершения загрузки файла в отладчик. Но сейчас-то мы знаем его адрес! Давим Ctrl-G, вводим 00406190h (адрес callback'а) и устанавливаем аппаратную точку на исполнение. Теперь перезапускаем отладчик по Ctrl-F2. На этот раз «Ольга» останавливается в начале функции обратного вызова (см. следующий рисунок). Трассируя ее, мы доходим до RET, попадая в недра NTDLL.DLL, но F9 выносит нас в точку входа (а если не выносит — ставим туда бряк).

Аналогичным образом работают и другие отладчики (в частности, Soft-Ice).



«Ольга», остановившаяся в начале функции обратного вызова

IDA-Pro автоматически отображает TLS-callback'и в списке точек входа (Ctrl-E), а также дешифрует TLS-таблицу в удобной форме. Так что на сложности взлома жаловаться не приходится. Главное, помнить о TLS-индексе и о том, что он может использоваться для самомодификации.



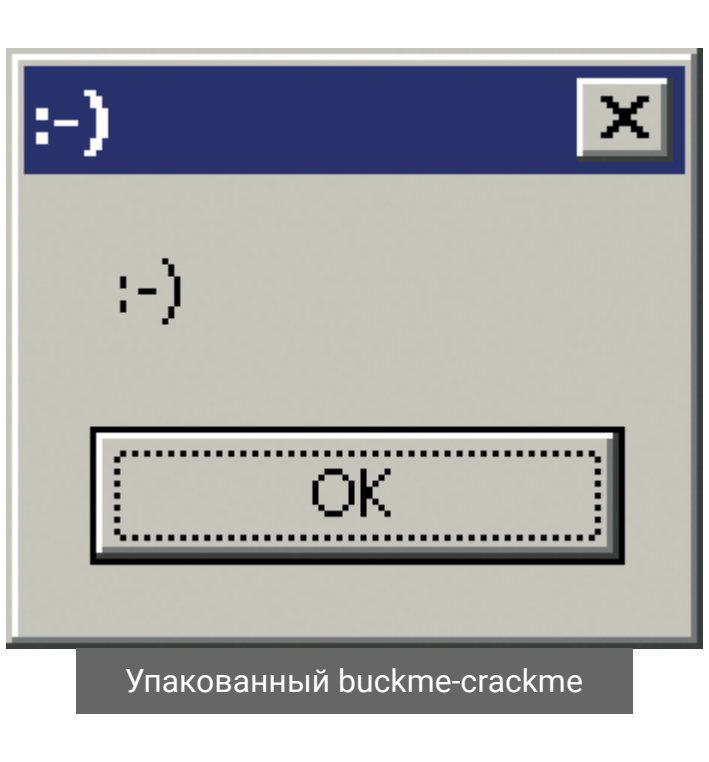
Список функций обратного вызова, отображаемый IDA-Pro

TLS-таблица декодирования IDA-Pro

```
.data:00406100 TLSDirectory      dd offset TLSSizeOfZeroFill
.data:00406104 TLSEnd_ptr        dd offset TLSEnd
.data:00406108 TLSIndex_ptr      dd offset TLSIndex
.data:0040610C TLSCallbacks_ptr dd offset TLSCallbacks
.data:00406110 TLSSizeOfZeroFill   dd 0
.data:00406114 TLSCharacteristics  dd 0
```

BUCKME-CRACKME — РЕАЛЬНЫЙ ХАРДКОР

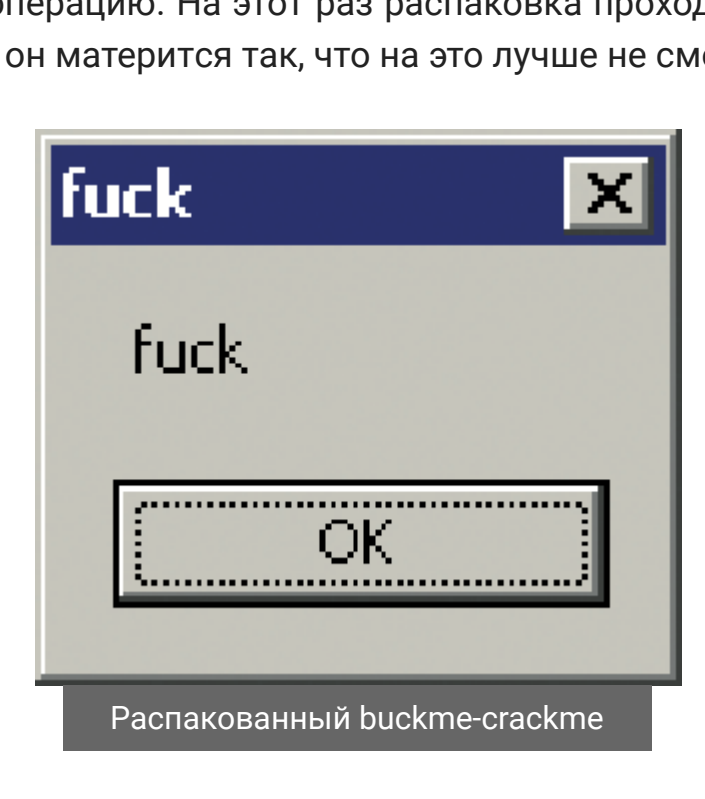
Разобравшись с основами TLS, попробуем заломать buckme-crackme. Заломать — в смысле распаковать, а упакован он UPX, что легко определить как с помощью PEiD/PE-TOOLS, так и визуальным просмотром файла в HIEW — по названиям секций UPX0, UPX1, UPX2. Запускаем упакованный файл на выполнение и видим ухмыляющуюся рожицу в диалоговом окне.



Упакованный buckme-crackme

Берем UPX и пишем `$UPX -d buck-me.exe`… Получаем: `upx: buck-me.exe: IOException: buck-me.exe: Permission denied` — как это так? С какого вдруг перепугу доступ отвергнут? Атрибута Read-Only у файла нет. Правда, на запись в файл у нас атрибуты были. А теперь нет. Куда же они подевались? Все просто. После нажатия OK программа не завершилась, и процесс продолжил болтаться в памяти. А как известно, доступ к запущенным файлам заботливо блокируется системой.

Лезем в диспетчер задач (или в FAR) и сносим процесс buck-me.exe к чертовой матери, после чего пробуем повторить операцию. На этот раз распаковка проходит успешно, но при запуске распакованного файла он матерится так, что на это лучше не смотреть.



Распакованный buckme-crackme

Короче, накрылась наша распаковка медным тазом. Поведение распакованной программы радикально изменилось. Значит, где-то есть проверка на наличие упаковщика. Чтобы понять где, смотрим распакованный код, который предельно прост.

Распакованный код buckme-crackme

```
.00401000: 8B442404      mov eax,[esp][04]
.00401004: 8B1500304000 mov edx,[00403000]
.0040100A: 6A00       push     000
.0040100C: 6800304000        push     000403000 ;'buck'
.00401011: 33D0       xor edx,eax
.00401013: 6800304000        push     000403000 ;'buck'
.00401018: 6A00       push     000
.0040101A: 891500304000 mov [00403000],edx
.00401020: FF1508204000 call     MessageBoxA ;USER32
.00401026: 6A00       push     000
.00401028: FF1500204000 call     ExitProcess ;KERNEL32
.0040102E: C3            retn
```

Ничего не понятно! Во-первых, в файле начисто отсутствует строка `:-)`, зато есть `buck`. Вместо `buck` мы получаем `fuck`, а все потому, что `buck` ссорится с аргументом, переданным программе, — который при выполнении из шелла равен 04h, а при запуске под отладчиком — 00h. Так программа еще и отладчик детектит? Здорово! Но все же куда девалась наша улыбающаяся рожа?

Упакованный вариант, видимо, вызывал функцию `start`, передавая ей такой аргумент, который при наложении на `buck` выдавал `:-)`. Проделав обратную операцию, мы восстановим исходный аргумент — 6B4A5858h. Интересно, кто бы его мог заслать в стек? Уж точно не UPX!

Извлекаем оригинальный exe из архива и загружаем его в HIEW. Втыкаем в директорию таблиц. Видим, что там есть TLS. Рысцой переключаемся на распакованную версию. TLS как турбиной сдуло. Что хоть в TLS было? Зовем на помощь IDA-Pro или HIEW.

Утерянный при распаковке TLS callback

```
.00406160: 6858584A6B        push     06B4A5858
.00406165: 50            push     eax
.00406166: 33C0       xor eax,eax
.00406168: 40            inc eax
.00406169: 39442410      cmp [esp][10],eax
.0040616D: 75FE       jne .00040616D  ---^ (1)
.0040616F: E96CEFFFFF       jmp .0004050E0  ---^ (2)
```

Ага, вот он — уже знакомый нам аргумент 6B4A5858h, засылаемый в стек. После чего callback проверяет значение параметра Reason, и если он не `DLL_PROCESS_ATTACH`, то циклит программу. В противном же случае передает управление на точку входа, давая отработать UPX. Тот распаковывает программу и зовет start, оставляя на стеке 6B4A5858h. А вот при статической распаковке UPX не сохраняет TLS, поскольку TLS был наложен руками на уже упакованный UPX'ом файл.

Подобный трюк использовался в конкурсе, проводимом F-Secure. Большое количество участников, видя знакомый UPX, распаковывало его на автомате, теряя TLS callback, а вместе с ним и часть функциональности.

Вывод: перед распаковкой всегда смотри TLS!

Дурим антивирусы

Секреты TLS на этом не заканчиваются. Они способны на такие трюки, что просто дух захватывает. Некоторые вирусы внедряются исключительно путем модифицирования всего четырех байт — указателя на TLS-таблицу, расположенную в памяти (в одной из системных DLL), где находится указатель на команду передачи управления на shell-код.

Конечно, подобная техника внедрения работает только на той версии операционной системы, под которую она заточена, но антивирусы таких вирусов не обнаруживают. Или вообще не обращают внимания на изменение directory table.