энциклопедия антиотладочных приемов — локальное хранилище потока или TLS callbacks выпуск #05h

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

загадочная аббревиатура tls таит в себе намного больше секретов, чем это может показаться на первый взгляд. это мощнейшее оружие против отладчиков и дизассемблеров. в комбинации с упаковщиками tls превращается в гремучую термитную смесь термоядерного типа. мыщъх представляет исчерпывающее руководство по приготовлению и взлому tls.

введение

Что такое TLS и чем оно грозит хакерам? Начнем издалека. Популярные языки программирования (в том числе и Си) поддерживают статические и глобальные переменные, использование которых делает код потоконебезопасным. Все потоки разделяют один и тот же набор глобальных/статических переменных, порождая путаницу, неразбериху 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 предоставляет 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 и "прозрачным" для программы.

PE-формат поддерживает функции обратного вызова (TLS-callback), автоматически вызываемые системой _до_ передачи управления на точку входа, что позволяет, в частности, определить наличие отладчика или скрытно выполнить некоторые действия. Так же, системный загрузчик записывает TLS-индекс в заданную локацию — отличный способ неявной самомодификации программы, не отлавливаемой дизассемблерами и заводящей 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 прочтение не создается. Мыщъх надеется исправить этот дефект.

fundamentals

Прежде всего нам понадобиться спецификация на PE-формат, последнюю версию которого можно утянуть прямо из под загребущих лап Microsoft [1, a], представленную в XML формате. Тот же самый файл, только сконвертированный в MS Word 2000 мыщъх выложил на своем сервере [1, b].

TLS таблица описывается 9'ым (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 мыщъху операционными системами и здесь можно писать все 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 ни бывало, а вот проверка валидности размера TLS-таблицы может появится в любой момент, так что лучше не рисковать и писать здесь то, что нужно и не прикалываться.

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$, однако, это делается из чисто эстетических соображений. Системный загрузчик имя секции не проверяет, правда, некоторые упаковщики не обрабатывают TLS, расположенные вне .tls, но это уже их личные половые проблемы, тем более что ряд упаковщиков вообще не знает, что такое TLS.

Формат самой TLS-таблицы приведен ниже (см. таблицу 1):

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

Таблица 1 формат TLS-таблицы для PE32/PE32+ файлов

Функции обратного вызова вызываются системным загрузчиком при инициализации/терминации процесса, а так же при создании/завершении потока и имеют тот же самый прототип, что и DllMain (см. листинг 1):

```
typedef VOID (NTAPI *PIMAGE_TLS_CALLBACK) (
        PVOID DllHandle,        // дескриптор модуля
        DWORD Reason,           // причина вызова
        PVOID Reserved          // зарезервировано
);
```

Листинг 1 прототип функций обратного вызова

Двойное слово Reason принимает следующие значения, информируя функцию обратного вызова по какой причине она была вызвана (см. таблицу 2):

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

Таблица 2 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 и, чтобы не возникало путаницы, системный загрузчик при инициации модуля записывает по адресу Address of Index индекс данного модуля, т.е. реально 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. То есть, если наш файл не загружает никаких DLL, использующих TLS, индекс с высокой степенью вероятности будет равен нулю, хотя и без всяких гарантий. Как же тогда 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!

На этом теоретическую часть будет считать законченной и приступим к практическим занятиям.

ручное создание TLS

Для работы с TLS нам необходим компилятор и линкер, поддерживающий обозначенную технологию, и, хотя недостатка в таковых нет (хотя полноценной поддержки TLS как не было, так и нет), по любому это не хакерский путь, к тому же мы можем захотеть 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 посложнее, т. к. придется править таблицу перемещаемых элементов, но тут тоже есть свои хитрости и трюки, но сначала — EXE.

Пишем простую программу типа "hello, world!", компилируем ее и открываем полученный файл в HIEW'e. 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 и иным), где пишем следующую магическую последовательность: 10 61 40 00 | 20 61 40 00 | 30 61 40 00 | 60 61 40 00, которая на самом деле никакая не магическая. Первая пара двойных слов означает начало/конец блока данных локальной памяти потока, который может находится 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'e 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 единственный callback по адресу 00406190h, за которым следует ноль, указывающий, что других callback'ов здесь нет и не предвидится.

Что же касается самого callback'a, то, подогнав курсор к адресу 00406190h, легким нажатием ENTER'a мы переходим в режим ассемблера и 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'ом, может воспользоваться готовым файлом hello-tls.exe, прилагаемому к журналу, а так же выложенном на мыщъхином сервере [2].

```
.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   ♪@a@ ├
```

Листинг 2 TLS, созданный вручную (hex-дамп)

Остается только занести TLS в таблицу директорий. В HIEW'e это делается так (см. рис. 1): 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 (ну, размер можно брать любой).

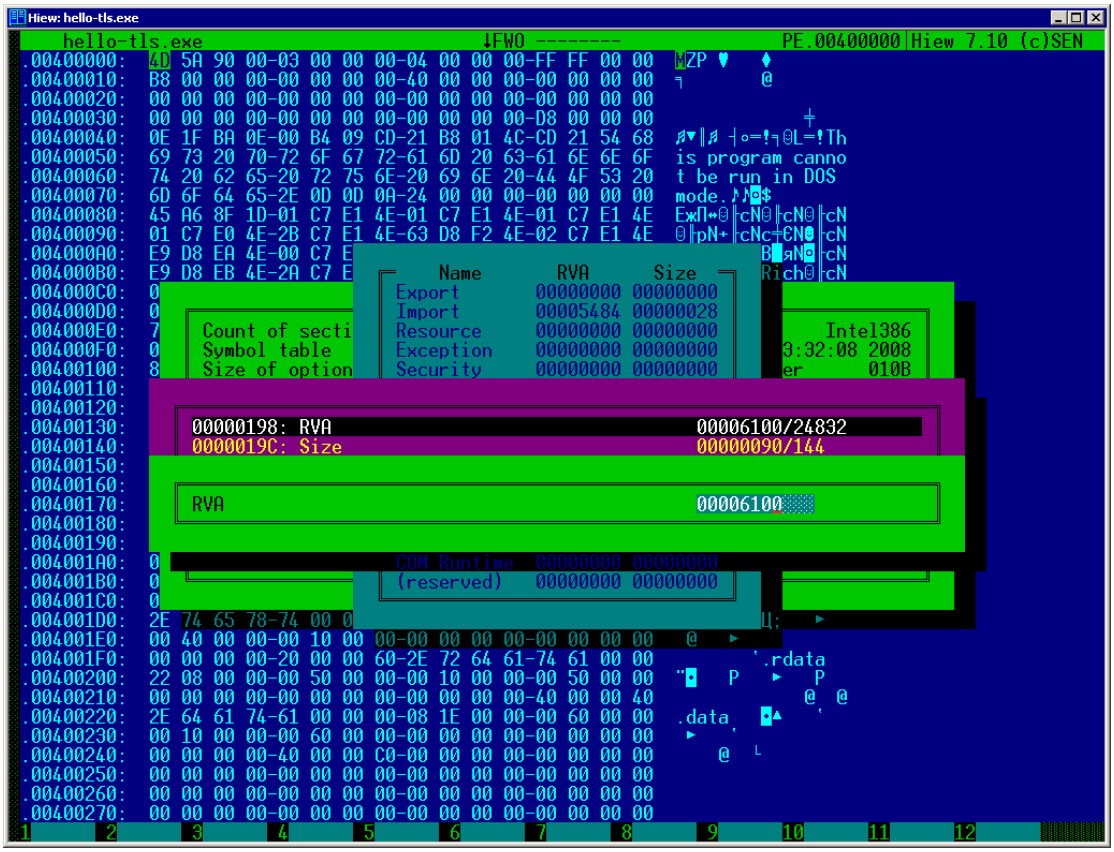

Рисунок 1 редактирование директории таблиц для "подключения" TLS

боевое крещение

Загружаем hello-tls.exe в отладчик (например, в Ольгу) и ходим по адресу 00406100h, где мы четко видим (см. рис. 2), что двойное слово 66666666h по адресу 00406130h мистическим образом обратилось в ноль, зато нулевое двойное слово по адресу 00406140h, уменьшившись на единицу, превратившись в FFFFFFFFh – результат записи индекса и вызова callback'а соответственно. Причем, это произошло _до_ того как мы успели выполнить хотя бы одну команду, стоя в точке входа.

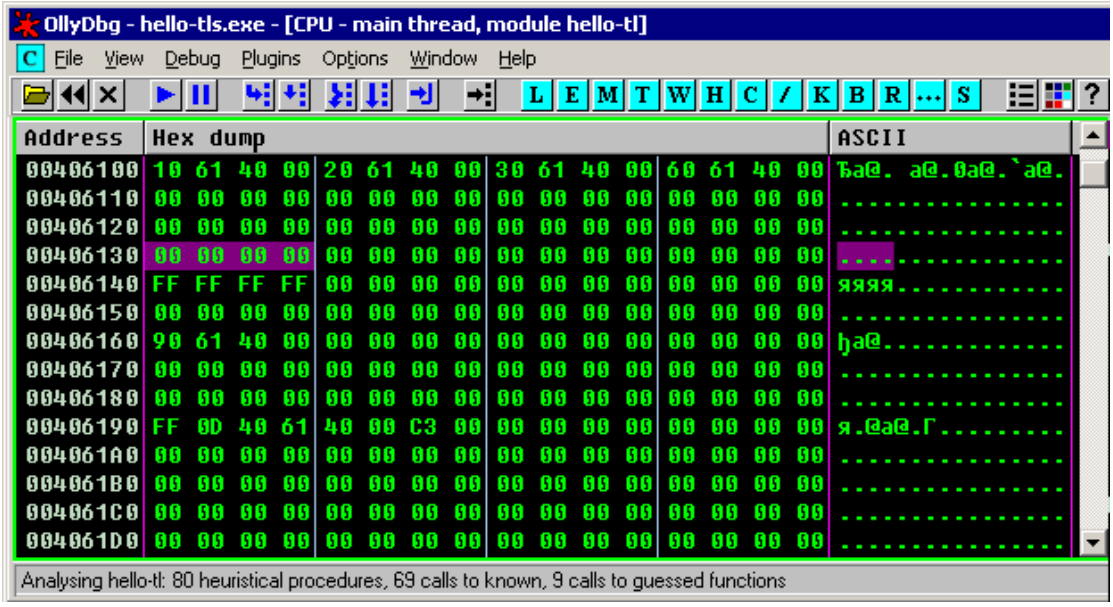

Рисунок 2 результат работы рукотворного TLS

как это ломают

Существует множество plug-in'ов для Ольги, автоматически стопящихся в начале TLS, но, во-первых, большинство из них не умеет обрабатывать более одного callback'а, а, во-вторых, мы — хакеры — должны готовы все делать своими руками, лапами и хвостом. Короче, зовем на помощь HIEW. Открываем файл и по <F10> зовет директорию таблиц, как уже описывалось выше. Видим там TLS, видим, 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 там. А там у нас 00406190h. Переходим по обозначенному адресу и жмем на ENTER, переключая HIEW в режим дизассемблера. Изучаем callback (см. рис. 3), попутно запоминая его адрес, который нам понадобиться чуть позже.

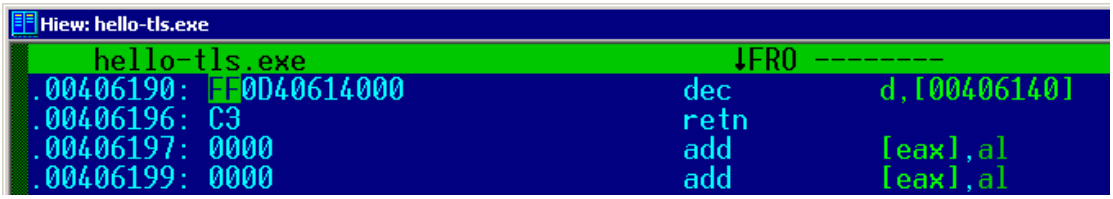

Рисунок 3 TLS функция обратного вызова в HIEW'e

OK, мы снова в Ольге. И снова TLS callback отработал еще до завершения загрузки файла в отладчик. Но сейчас мы знаем его адрес!!! Говорим <CTRL-G>, вводим "00406190h" (адрес callback'а) и устанавливаем аппаратную точку на исполнение. Перезапускаем отладчик по <CTRL-F2> и на этот раз Ольга останавливается в начале функции обратного вызова (см. рис. 4), трассируя которую 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).

Рисунок 4 Ольга, остановившаяся в начале функции обратного вызова

IDA-Pro автоматически отображает TLS-callback'и в списке точек входа (<CTRL-E>, см. рис. 5), а так же дешифрует TLS-таблицу в удобной для восприятия форме (см. листинг 3), так что на сложности взлома жаловаться не приходится. Главное помнить о TLS-индексе и о том, что он может использоваться для самомодификации.

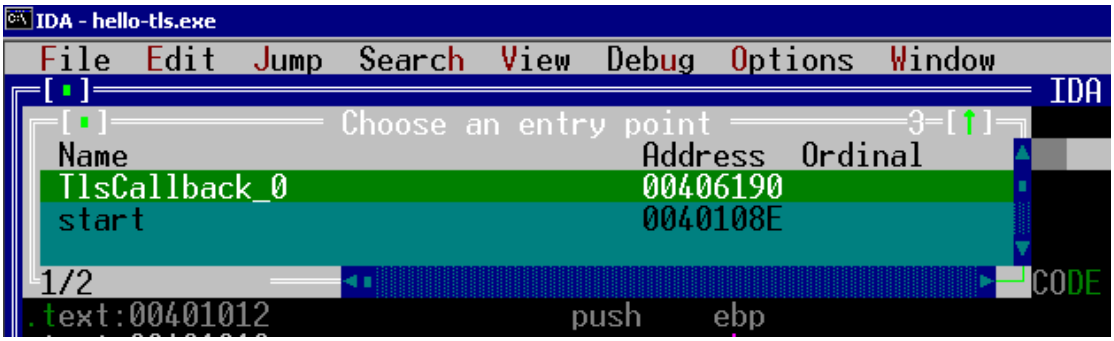

Рисунок 5 список функций обратного вызова, отображаемый IDA-Pro

```
.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
```

Листинг 3 TLS-таблица декодирования IDA-Pro

buckme-crackme — реальный хардкор

Теперь, разобравшись с основами TLS, попробуем заломать buckme-crackme [3]. Заломать в смысле распаковать, а упакован он UPX'ом, что легко определить как с помощью PEiD/PE-TOOLS, так и визуальным просмотром файла в HIEW'е по названием секций UPX0, UPX1, UPX2.

Запускаем упакованный файл на выполнение и видим ухмыляющуюся рожицу в диалоговом окне (см. рис. 6).

Рисунок 6 buckme-crackme (упакованный)

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

Материмся, лезем в "Диспетчер Задач" (или в FAR) и сносим процесс "buck-me.exe" к чертовой матери, после чего повторяем операцию вновь. На этот раз распаковка проходит успешно, но… при запуске распакованного файла он матерится так, что на это лучше не смотреть (см. рис. 7).

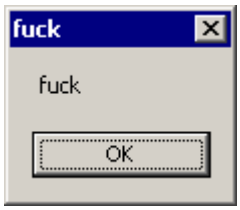

Рисунок 7 buckme-crackme (распакованный)

Короче, накрылась наша распаковка. Медным Тазом. Поведение распакованной программы изменилось. Причем весьма радикально. Значит, где-то есть проверка на наличие упаковщика. Но где?! Смотрим распакованный код, который предельно прост (см. листинг 4):

```
.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
```

Листинг 4 распакованный код buckme-crackme

Ничего не понятно! Во-первых, в файле начисто отсутствует строка ":-)", зато есть "buck", только вместо "buck" мы получаем "fuck", а все потому, что "buck" ксориться 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 рожа?!

Видимо, упакованный вариант 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. А что там хоть в TLS было?! Зовем на помощь IDA-Pro или HIEW (см. листинг 5).

```
.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)
```

Листинг 5 утерянный при распаковке TLS callback

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'у, который 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 частности, в конкурсе проводимом F-Secure. Большое количество участников, видя знакомый UPX, распаковывало его на автомате, теряя TLS callback, а вместе с ним и часть функционала.

Вывод: перед распаковкой всегда смотреть TLS.

заключение

Секреты TLS на этом не заканчиваются, а только начинаются. Они способны на такие трюки, что просто дух захватывает. В частности, некоторые вирусы внедряются исключительно путем модифицирования всего 4х байт — указателя на TLS таблицу, расположенную в памяти (в одной из системных DLL), где находится указатель на команду передачи управления на shell-код, так же находящийся в системных DLL. Конечно, подобная техника внедрения работает только на той версии операционной системы под которую она заточена, но антивирусы таких вирусов не обнаруживают, а то и вообще не обращают внимание на изменение directory table.

>>> врезка: а знаете ли вы что…

Исключения, возникающие внутри TLS callback'ов давятся системой на автомате, зато отлавливаются отладчиками, той же Ольгой к примеру и хакер ни хвоста не может понять как это _вообще_ может работать, хотя что тут думать — давить Shift-F9 для передачи управления на точку входа!

>>> ссылки к статье

1. **Microsoft Portable Executable and Common Object File Format Specification**:
   a. http://www.microsoft.com/whdc/system/platform/firmware/PECOFF.mspx;
   b. http://nezumi.org.ru/souriz/hack/pecoff_v8.zip;
2. **hello-TLS**:
   a. http://nezumi.org.ru/souriz/hack/pecoff_v8.zip;
3. **buckme-crackme**:
   a. http://nezumi.org.ru/buckme-crackme.zip;