

Статья Деобфускация хеширования API DanaBot

 xss.is/threads/45278

Вы, наверное, уже догадались по названию названия, API-хеширование используется для обфускации двоичного файла, чтобы скрыть имена API от инструментов статического анализа, что мешает реверс инженеру понять функциональность вредоносного ПО. Первый способ получить представление о функциональных возможностях исполняемого файла - это более или менее подробно изучить функции и найти вызовы API. Если, например, функция `CreateFileW` вызывается в определенной подпрограмме, это, вероятно, означает, что перекрестные ссылки или сама подпрограмма реализуют некоторые функции обработки файлов. Это будет невозможно, если используется хеширование API.

Вместо прямого вызова функции каждый вызов API имеет соответствующую контрольную сумму/хэш. Может быть получено жестко запрограммированное хеш-значение, и для каждой библиотечной функции вычисляется контрольная сумма. Если вычисленное значение совпадает с хеш-значением, с которым мы его сравниваем, мы нашли нашу цель.

```
mov     edx, 507CA1h
mov     eax, dword_3856C0
call    ResolveFuncHash ; socket
mov     g_socket, eax

...

mov     [ebp+var_64], eax
fild   [ebp+var_64]
call    FistpCall
mov     [ebp+var_44], eax
push   0             ; _DWORD
push   1             ; _DWORD
push   2             ; _DWORD
call    g_socket
mov     [ebp+var_1C], eax
imul   eax, [ebp+var_54], 27Eh
mov     [ebp+var_50], eax
mov     eax, [ebp+var_24]
add    eax, [ebp+var_24]
mov     [ebp+var_50], eax
mov     eax, [ebp+var_28]
sub    [ebp+var_24], eax
lea    ecx, [ebp+var_2C]
lea    edx, [ebp+var_24]
lea    eax, [ebp+var_24]
call   sub_343944
mov     eax, [ebp+var_24]
add    eax, [ebp+var_24]
mov     [ebp+var_50], eax
cmp    [ebp+var_1C], 0
ja     short loc_34FB1C
```

Move checksum of function we want to call into EDX. Address of socket API call will be stored into EAX and persisted

Call socket API function from hardcoded address.

XSS.is

В этом случае реверс-инженер должен выбрать другой путь для анализа двоичного файла или его деобфускации. В этой статье блога будет рассказано, как банковский троян DanaBot реализует хеширование API, и, возможно, самый простой способ борьбы с этим. SHA256 двоичного файла, который я здесь анализирую, добавлен в конце этого сообщения в блоге.

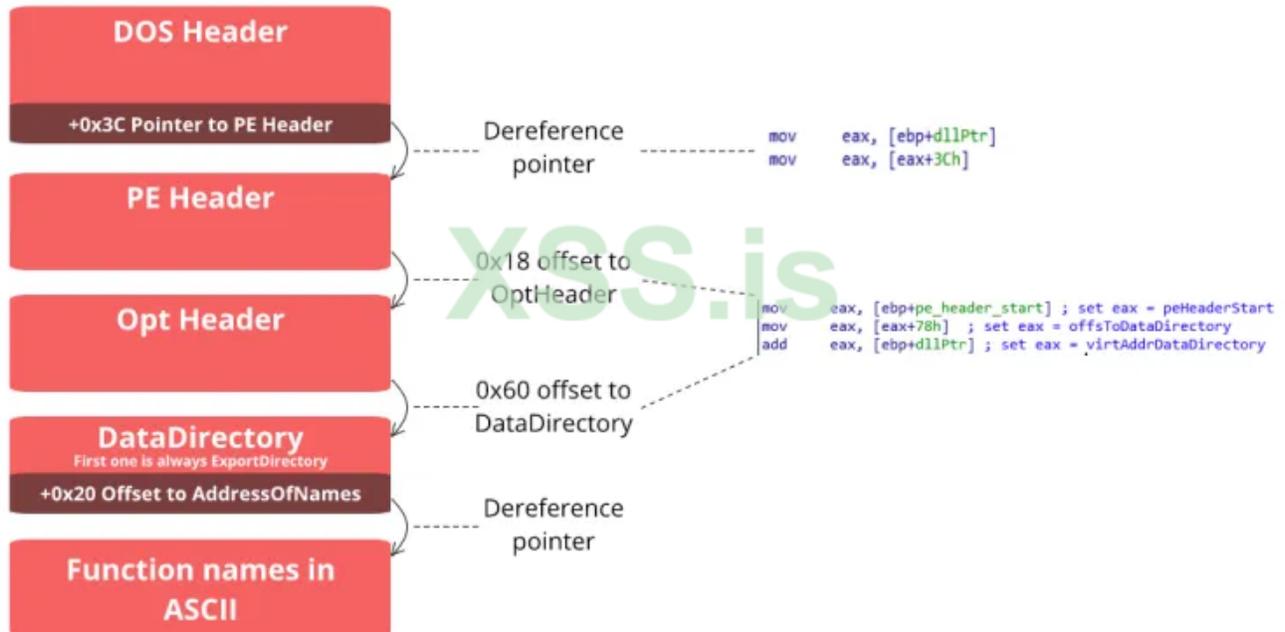
Глубокое погружение в DanaBot

Сам DanaBot является банковским трояном, существует как минимум с 2018 года и впервые был обнаружен ESET[1]. Стоит отметить, что он реализует большинство своих функций в плагинах, которые загружаются с сервера C2. Я сосредоточусь на деобфускации хеширования API на первом этапе DanaBot, DLL, которая дропается и сохраняется в системе, используемой для загрузки дополнительных плагинов.

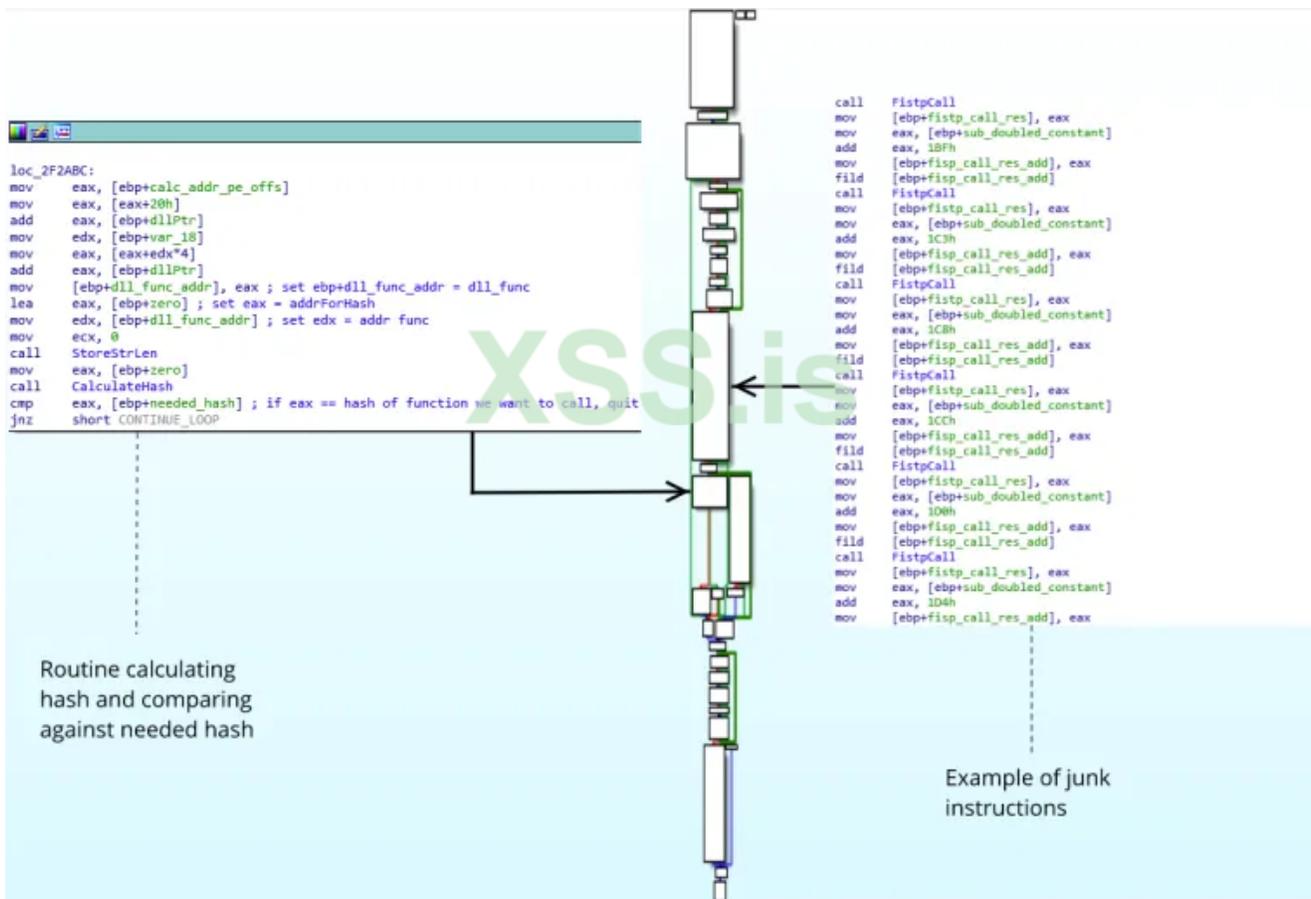
Реверсинг процедуры ResolvFuncHash

В начале функции регистр EAX хранит указатель на заголовок DOS динамически связанной библиотеки, который содержит функцию, которую двоичный файл хочет вызвать. Соответствующий хэш еще неизвестной функции API хранится в регистре EDX. Подпрограмма также содержит кучу ненужных инструкций, обфусцирующий фактический вариант использования этой функции.

Хэш вычисляется исключительно из имени функции, поэтому первым шагом является получение указателя на все имена функций целевой библиотеки. Каждая DLL содержит таблицу со всеми экспортируемыми функциями, которые загружаются в память. Этот Каталог Экспорта всегда является первой записью в массиве Каталога Данных. Формат файла PE и его заголовки содержат достаточно информации, чтобы добраться до указанного каталога путем анализа структур заголовков:



На рисунке ниже вы можете увидеть пример упомянутых нежелательных инструкций, а также критический блок, который сравнивает вычисленный хэш с контрольной суммой функции, которую мы хотим вызвать. Подпрограмма выполняет итерацию по всем именам функций в каталоге экспорта и вычисляет хэш. Цикл прерывается, когда вычисленный хэш совпадает со значением, которое хранится в регистре EDX с начала этой процедуры.



Реверсинг алгоритма хеширования

Алгоритм хеширования довольно простой и не содержит ничего особенного. Нежелательные инструкции и непрозрачные предикаты усложняют процесс реверсинга этой процедуры.

Алгоритм берет n-й и stringLength-n-1-й символ имени функции и сохраняет их, а также версии с заглавной буквы в памяти, в результате чего получается 4 символа. Каждый из этих символов подвергается операции XOR с длиной строки. Наконец, они умножаются, и значения складываются каждый раз при запуске цикла, в результате чего получается хеш-значение.

Python:

```

def get_hash(funcname):
    """Calculate the hash value for function name. Return hash value as integer"""
    strlen = len(funcname)
    # if the length is even, we encounter a different behaviour
    i = 0
    hashv = 0x0
    while i < strlen:
        if i == (strlen - 1):
            ch1 = funcname[0]
        else:
            ch1 = funcname[strlen - 2 - i]
        # init first character and capitalize it
        ch = funcname[i]
        uc_ch = ch.capitalize()
        # Capitalize the second character
        uc_ch1 = ch1.capitalize()
        # Calculate all XOR values
        xor_ch = ord(ch) ^ strlen
        xor_uc_ch = ord(uc_ch) ^ strlen
        xor_ch1 = ord(ch1) ^ strlen
        xor_uc_ch1 = ord(uc_ch1) ^ strlen
        # do the multiplication and XOR again with upper case character1
        hashv += ((xor_ch * xor_ch1) * xor_uc_ch)
        hashv = hashv ^ xor_uc_ch1
        i += 1
    return hashv

```

Сценарий python для вычисления хэша для заданного имени функции также загружен на мою страницу github и доступен всем желающим. Я также загрузил текстовый файл с хешами для экспортируемых функций часто используемых DLL.

Деобфускация с помощью комментирования

Итак, теперь, когда мы взломали алгоритм, мы хотим обновить наш дизассемблируемый код, чтобы знать, какое хеш-значение представляет какую функцию. Как я уже упоминал, мы хотим сосредоточиться на простоте. Самый простой способ - вычислить хеш-значения для экспортируемых функций часто используемых DLL и записать их в файл.

```
Terminal - zorro@zorro-VirtualBox: ~/Projects/Malwareandstuff/DanabotDeobfuscation
File Edit View Terminal Tabs Help
vininet.dll--0x3a9300--GopherFindFirstFileA--218--0x11a8d16
vininet.dll--0x3a9300--GopherFindFirstFileW--219--0x1171666
vininet.dll--0x3a9320--GopherGetAttributeA--220--0xfa4dfd
vininet.dll--0x3a9320--GopherGetAttributeW--221--0xf79ccd
vininet.dll--0x3a9340--GopherGetLocatorTypeA--222--0x10cf987
vininet.dll--0x3a9340--GopherGetLocatorTypeW--223--0x10998af
vininet.dll--0x3a9360--GopherOpenFileA--224--0xa71d4f
vininet.dll--0x3a9360--GopherOpenFileW--225--0xa8efaf
vininet.dll--0x2d3c70--HttpAddRequestHeadersA--226--0x112cad9
vininet.dll--0x2d59c0--HttpAddRequestHeadersW--227--0x10dfef9
vininet.dll--0x3b1680--HttpCheckDavCompliance--228--0x14b628e
vininet.dll--0x2fb3c0--HttpCloseDependencyHandle--229--0x1805105
vininet.dll--0x30f670--HttpDuplicateDependencyHandle--230--0x1c59b46
vininet.dll--0x31a520--HttpEndRequestA--231--0xda8c61
vininet.dll--0x31cbd0--HttpEndRequestW--232--0xdc58cd
vininet.dll--0x28e190--HttpGetServerCredentials--233--0x1697253
vininet.dll--0x391340--HttpGetTunnelSocket--234--0x1061559
vininet.dll--0x31a2a0--HttpIndicatePageLoadComplete--235--0x1b30529
vininet.dll--0x29dee0--HttpIsHostHstsEnabled--236--0x10c0653
vininet.dll--0x294390--HttpOpenDependencyHandle--237--0x17cae8d
vininet.dll--0x3b1fc0--HttpOpenRequestA--238--0xb9bd0d
vininet.dll--0x2c4b60--HttpOpenRequestW--239--0xb7b28d
vininet.dll--0x391ea0--HttpPushClose--240--0xafc7f0
vininet.dll--0x391f10--HttpPushEnable--241--0xbf702b
vininet.dll--0x391f90--HttpPushWait--242--0xab8fe8
vininet.dll--0x2cdd80--HttpQueryInfoA--243--0xbd2507
vininet.dll--0x2cbd80--HttpQueryInfoW--244--0xbf0e7
vininet.dll--0x31ae40--HttpSendRequestA--245--0xb74b43
vininet.dll--0x31caa0--HttpSendRequestExA--246--0xd62b76
vininet.dll--0x3196b0--HttpSendRequestExW--247--0xd33f56
vininet.dll--0x2d86a0--HttpSendRequestW--248--0xb540c3
vininet.dll--0x3c0d30--HttpWebSocketClose--249--0xf28d47
vininet.dll--0x3c11f0--HttpWebSocketCompleteUpgrade--250--0x1a252d0
vininet.dll--0x3c0e40--HttpWebSocketQueryCloseStatus--251--0x192f9a1
vininet.dll--0x3c15c0--HttpWebSocketReceive--252--0x102795a
```

С помощью этого файла мы можем написать сценарий IdaPython, чтобы прокомментировать имя библиотечной функции рядом с вызовом хеширования Ari. К счастью, функция хеширования Ari всегда вызывается по одному и тому же шаблону:

- Переместите желаемое значение хеш-функции в регистр EDX
- Переместить DWORD в регистр EAX

Сначала мы получаем все ссылки на функцию хеширования Ari. Каждый XRef будет содержать адрес, по которому вызывается функция хеширования Ari, что означает, что как минимум в 5 предыдущих инструкциях мы найдем упомянутый шаблон.

Таким образом, мы будем извлекать предыдущую инструкцию до тех пор, пока не извлечем желаемое значение хеш-функции, которое будет помещено в EDX. Наконец, мы можем использовать это немедленно, чтобы извлечь соответствующую функцию arі из значений хэша, которые мы сгенерировали ранее, и прокомментировать имя функции рядом с адресом Xref.

Python:

```

def add_comment(addr, hashv, api_table):
    """Write a comment at addr with the matching api function.Return True if a corresponding
    api hash was found."""
    # remove the "h" at the end of the string
    hashv = hex(int(hashv[:-1], 16))
    keys = api_table.keys()
    if hashv in keys:
        apifunc = api_table[hashv]
        print "Found ApiFunction = %s. Adding comment." % (apifunc,)
        idc.MakeComm(addr, apifunc)
        comment_added = True
    else:
        print "Api function for hash = %s not found" % (hashv,)
        comment_added = False
    return comment_added

def main():
    """Main"""
    f = open(
        "C:\\Users\\luffy\\Desktop\\Danabot\\05-07-2020\\Utils\\danabot_hash_table.txt", "r")
    lines = f.readlines()
    f.close()
    api_table = get_api_table(lines)
    i = 0
    ii = 0
    for xref in idutils.XrefsTo(0x2f2858):
        i += 1
        currentaddr = xref.frm
        addr_minus = currentaddr - 0x10
        while currentaddr >= addr_minus:
            currentaddr = PrevHead(currentaddr)
            is_mov = GetMnem(currentaddr) == "mov"
            if is_mov:
                dst_is_edx = GetOpnd(currentaddr, 0) == "edx"
                # needs to be edx register to match pattern
                if dst_is_edx:
                    src = GetOpnd(currentaddr, 1)
                    # immediate always ends with 'h' in IDA
                    if src.endswith("h"):
                        add_comment(xref.frm, src, api_table)
                        ii += 1
        print "Total xrefs found %d" % (i,)
        print "Total api hash functions deobfuscated %d" % (ii,)

if __name__ == '__main__':
    main()

```

Заклучение

Как реверс-инженеры, мы, вероятно, продолжим сталкиваться с Api Hashing по-разному. Надеюсь, я смог показать вам какой-нибудь быстрый и грязный метод или дать вам хотя бы некоторые основы того, как победить эту технику обфускации. Я также надеюсь, что в следующий раз, когда blue team товарищу придется проанализировать DanaBot, эта статья может оказаться для него полезной и сэкономит ему время на реверс инжиниринг этого банковского трояна.

- Dropper =
e444e98ee06dcoe26cae8aa57aocddab7b05odb22d3002bd2boda47d4fd5d78c
- DLL = cde01a2eeb558545c57d5c71c75e9a3b70d71ea6bbeda790a0b871fcb1b76f49

Источник: <https://malwareandstuff.com/deobfuscating-danabots-api-hashing/>

Автор перевода: yashechka

Переведено специально для <https://xss.is>