

IDAPython ile Otomasyon

written by Mert SARICA | 3 October 2016

Her ne kadar güvenlik arařtırmalarında ve zararlı yazılım analizinde hata ayıklayıcı/disassembler olarak Immunity Debugger/OllyDbg araçlarını daha kullanışlı bulsam da, bu durum IDA hata ayıklayıcı/disassembler aracının gücünü ve yeteneklerini hem bireysel hem de kurumsal olarak göz ardı ettiğim anlamına gelmiyor. Özellikle analiz edilen programı kaynak koduna çevirme yeteneğİ, çok sayıda platform desteğİ, IDAPython eklentisi , Immunity Debugger'a göre her daim güncellenmesi ve geliştirilemeye devam edilmesi IDA aracını benim için vazgeçilmez kılıyor. Hem bu artılarından dolayı hem de bankacılık zararlı yazılımlarını yakından takip edip analiz eden bir banka için de bankanın lisanslı ticari araçları arasındaki yerini istikrarlı bir şekilde yıllardır korumaya devam ediyor. Bu yazıda örnek bir zararlı yazılım analizinde manuel olarak yapıldığı taktirde oldukça zaman alabilen bir işlemin IDAPython eklentisi ile nasıl hızlandırılabilceğine dikkat çekeceğim. Ayrıca bu yazı Pi Hediye Var #7 oyununun da çözüm yolunu içermektedir.

IDAPython, Python programlama dili ile hazırlamış olduğunuz betikleri (script), IDA API ve diğer Python modüllerinden de faydalanarak IDA üzerinde kullanmanızı sağlayan oldukça faydalı bir eklentidir. IDAPython eklentisi IDA Professional ve Starter ticari sürümleri ile birlikte yüklü olarak gelmekte, demo sürümünde ise harici olarak GitHub sayfası üzerinden yüklenmesi gerekmektedir.

IDAPython'ın kullanımı ile ilgili olarak internette çok sayıda olmasa da yeterli sayıda kaynak bulabilirsiniz. Bunlardan Ero Carrera'nın Introduction to IDAPython makalesini, ücretsiz The Beginner's Guide to IDAPython e-kitabını, Palo Alto'nun IDAPython yazı serilerini öncelikle okumanızı tavsiye edebilirim. IDA API kullanımı ile ilgili olarak da Hex-Rays'in IDAPython dokümanlarını inceleyebilirsiniz.

Bildiğiniz üzere zararlı yazılım analizine ilk olarak (çoğunlukla) statik analiz ile başlanmaktadır ve bu adımda zararlı yazılım üzerinde yer alan karakter dizileri (strings) GNU strings, Sysinternals strings vb. araçlarla incelenebilmektedir. Tabii sizin hangi adımlardan geçtiğiniz çok iyi bilen ve ileri seviye siber saldırılarda (APT) kullanılmak üzere zararlı yazılım geliştiren art niyetli kişiler buna karşın bu karakter dizilerinin çalışma

esnasında (runtime) çözülmesini (decode) sağlayan fonksiyonlar (string decoder) kullanmaktadırlar. Bu gibi bir durumla karşılaştığınızda örneğin zararlı yazılım üzerinde okunaklı olmayan (encoded string) 100 tane karakter dizisi var ise ilk iş olarak bu karakter dizilerini çözen ana fonksiyonu (string decoder) bulmak olmalıdır. Hata ayıklayıcıda (debugger) zararlı yazılımı çalıştırdıktan sonra çözme işlemini gerçekleştiren fonksiyonun başlangıcına ve sonuna kesme noktası (breakpoint) koyarak çözülen karakter dizilerini yazmaçlardan (register) anlık olarak elde edebilirsiniz. Teoride uygulanabilir olsa da pratikte her çözülen karakter dizisini bir kenara not almak veya programın akışında 5 karakter dizisi çözdükten sonra anti-vm, anti-debugger kontrollerine takıldığı için sonlanan hata ayıklayıcı bize zaman kaybettirebilir. Bu gibi durumlarda IDAPython eklentisi bizi bu çıkmazdan kurtabilir.

Konakladıkları otellerdeki Wi-Fi ağını kullanan üst düzey kurum çalışanlarının DarkHotel APT grubu tarafından uzun yıllardan beri hedef alındığı geçtiğimiz yıllarda haberlere yansımıştı. DarkHotel grubunun kullandığı zararlı yazılım ile benzerlikler taşıyan ve Pi Hediye Var #7 oyununa da konu olan Dubnium adındaki zararlı yazılım, Microsoft'un Threat Research & Response Blog'unda yer alan yazıda da belirtildiği üzere karakter dizilerini gizlemek için bir fonksiyon kullanıyordu.

Dubnium zararlı yazılımı ile ilişkili olan ve 0ac65c60ad6f23b2b2f208e5ab8be0372371e4b3 SHA1 hashine sahip olan sshkeypairgen.exe (Pi Hediye Var #7'de adı adobe-pdf-reader.exe olarak değiştirilmişti.) isimli zararlı yazılımı IDA ile incelediğimde, Microsoft'un yazısına konu olan karakter dizisi çözme fonksiyonuna (sub_1177036) benzer bir fonksiyonun bu yazılımda da olduğunu gördüm. sub_1177036 fonksiyonunu çağıran diğer (xrefs to) fonksiyonları listelediğimde sayının oldukça fazla olduğunu gördüm ki bu durum gerçekten karakter dizisi çözdüğüne dair bir işaret olabilirdi.

IDA - sshkeypairgen.exe C:\Users\Mert\Desktop\dubnium\sshkeypairgen.exe

File Edit Jump Search View Debugger Options Windows Help

No debugger

Library function Data Regular function Unexplored Instruction External symbol

Functions window

Function name

- sub_42B960
- sub_42BC40
- sub_42BC50
- sub_42BCA0
- sub_42BCD0
- sub_42BD30
- sub_42BD70
- sub_42BD90
- sub_42BDE0
- sub_42BEE0
- sub_42BF20
- sub_42BFD0
- sub_42C030
- sub_42C110
- sub_42C190
- sub_42C1E0

Graph overview

100.00% (-134,-20) (993,18) 00094166 00494D66: sub_494CC9+9D (Synchronized with Hex View-1)

Output window

Python 2.7.6 (default, Nov 10 2013, 19:24:18) [MSC v.1500 32 bit (Intel)]
IDAPython v1.7.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>

Using FLIRT signature: Microsoft VisualC 2-14/net runtime
Propagating type information...
4A61A9: propagate_stkargs: function is already typed
Function argument information has been propagated
The initial autoanalysis has been finished.

Python

AU: idle Down Disk: 20GB

IDA - sshkeypairgen.exe C:\Users\Mert\Desktop\dubnium\sshkeypairgen.exe

File Edit Jump Search View Debugger Options Windows Help

No debugger

Library function Data Regular function Unexplored Instruction External symbol

Functions window

Function name

- sub_42B960
- sub_42BC40
- sub_42BC50
- sub_42BCA0
- sub_42BCD0
- sub_42BD30
- sub_42BD70
- sub_42BD90
- sub_42BDE0
- sub_42BEE0
- sub_42BF20
- sub_42BFD0
- sub_42C030
- sub_42C110
- sub_42C190
- sub_42C1E0

Graph overview

100.00% (-134,1101) (665,3) 00094166 00494D66: sub_494CC9+9D (Synchronized with Hex View-1)

Output window

Python 2.7.6 (default, Nov 10 2013, 19:24:18) [MSC v.1500 32 bit (Intel)]
IDAPython v1.7.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>

Using FLIRT signature: Microsoft VisualC 2-14/net runtime
Propagating type information...
4A61A9: propagate_stkargs: function is already typed
Function argument information has been propagated
The initial autoanalysis has been finished.

Python

AU: idle Down Disk: 20GB

sub_494CC9 proc near

var_544= dword ptr -544h
var_524= dword ptr -524h
var_520= dword ptr -520h
var_51C= dword ptr -51Ch
String= byte ptr -518h
Dst= byte ptr -517h
var_410= byte ptr -410h
var_40F= byte ptr -40Fh
var_10= dword ptr -10h
var_C= dword ptr -0Ch
var_4= dword ptr -4

; FUNCTION CHUNK AT 004B2E58 SIZE 0000000B BYTES

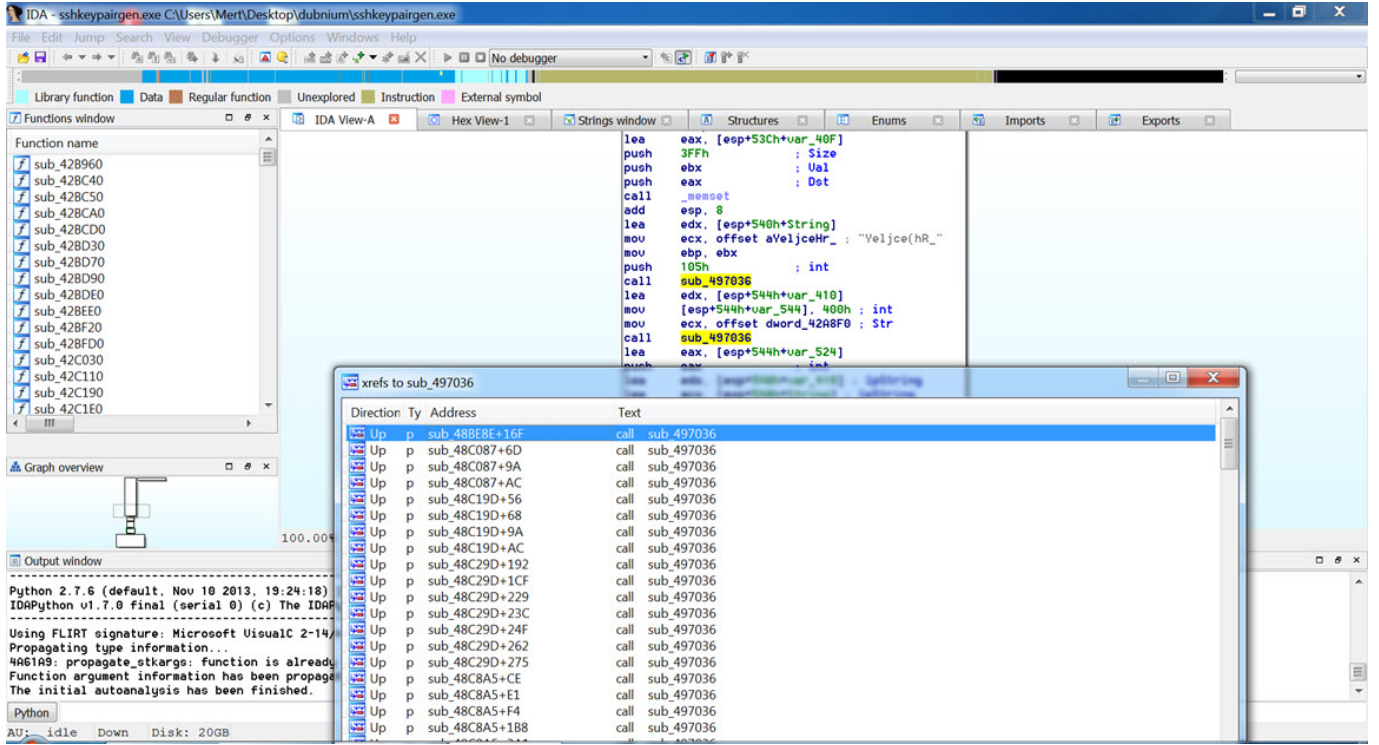
push 0FFFFFFFh
push offset SEH_494CC9
mov eax, large fs:0
push eax
sub esp, 51Ch
mov eax, security_cookie
xor eax, esp

; START OF FUNCTION CHUNK FOR sub_494CC9

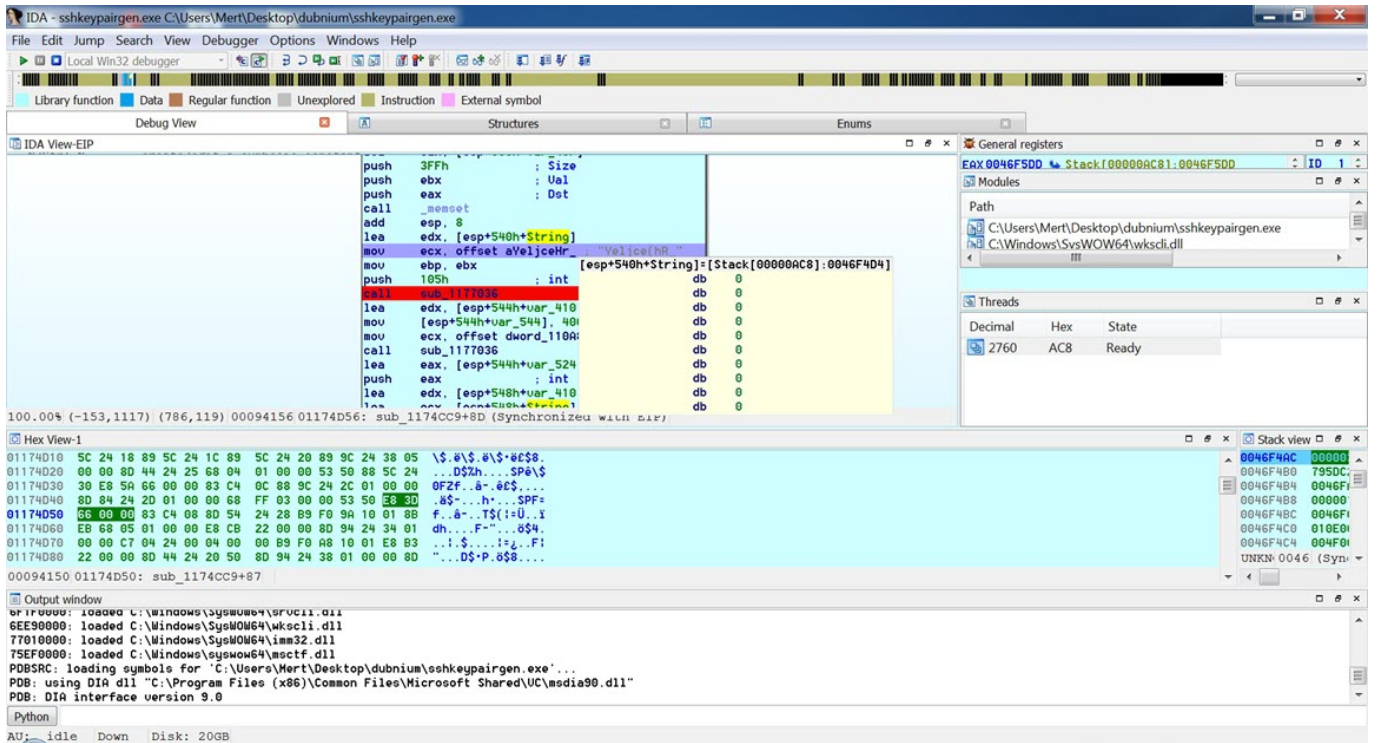
loc_4B2E58:
lea ecx, [ebp-524h]
jmp ?_Tidy0?Vector@PAXU?Allocator@PAXStd00std
; END OF FUNCTION CHUNK FOR sub_494CC9

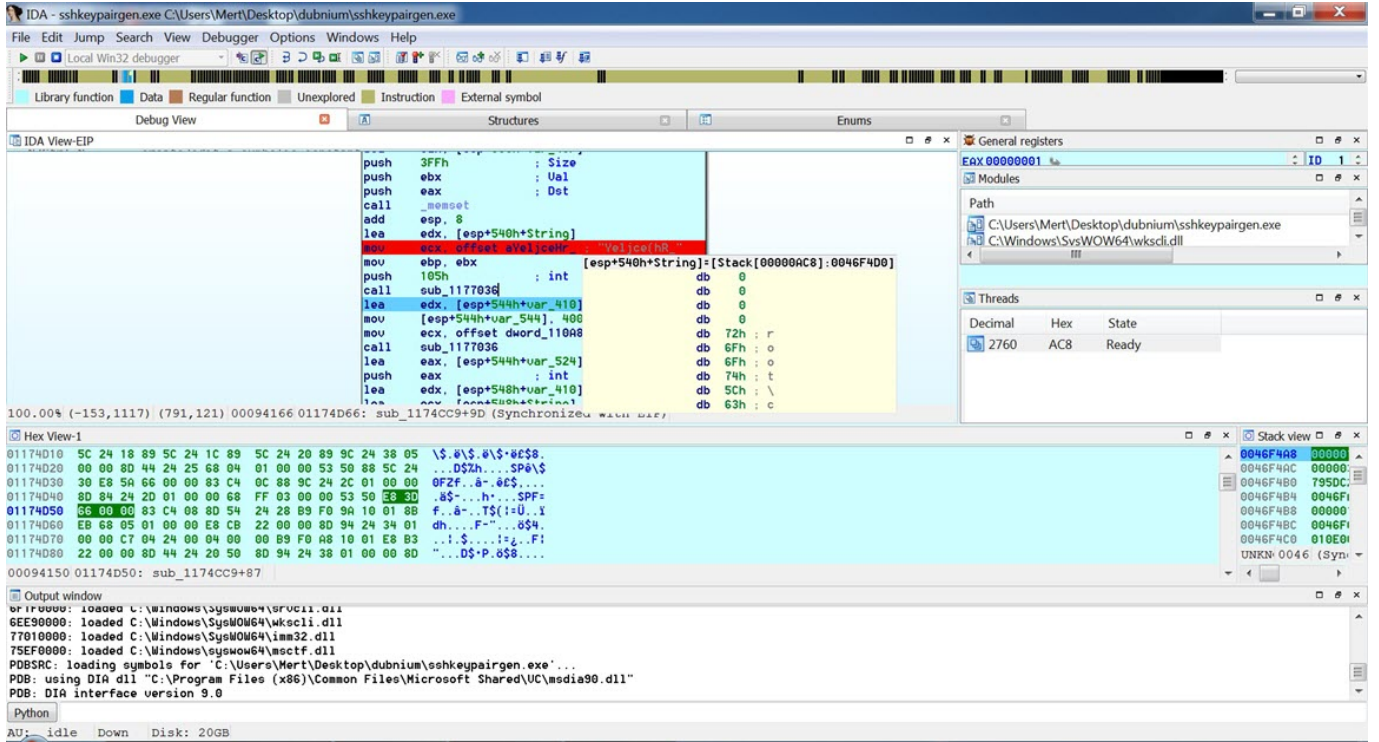
lea eax, [esp+53Ch+var_40F]
push 3FFh ; Size
push ebx ; Val
push eax ; Dst
call _memset
add esp, 8
lea edx, [esp+540h+String]
mov ecx, offset aVeljceHr_ ; "Veljce(HR_"
mov ebp, ebx
push 105h ; int
call sub_497036
lea edx, [esp+544h+var_410]
mov [esp+544h+var_544], 400h ; int
mov ecx, offset dword_42A8F0 ; Str
call sub_497036
lea eax, [esp+544h+var_524]
push eax ; int
lea edx, [esp+548h+var_410] ; lpString
lea ecx, [esp+548h+String] ; lpString
call sub_49953B
mov esi, [esp+548h+var_524]
add esp, 0Ch
test eax, eax
jz short loc_494D84

mov eax, [esp+53Ch+var_520]

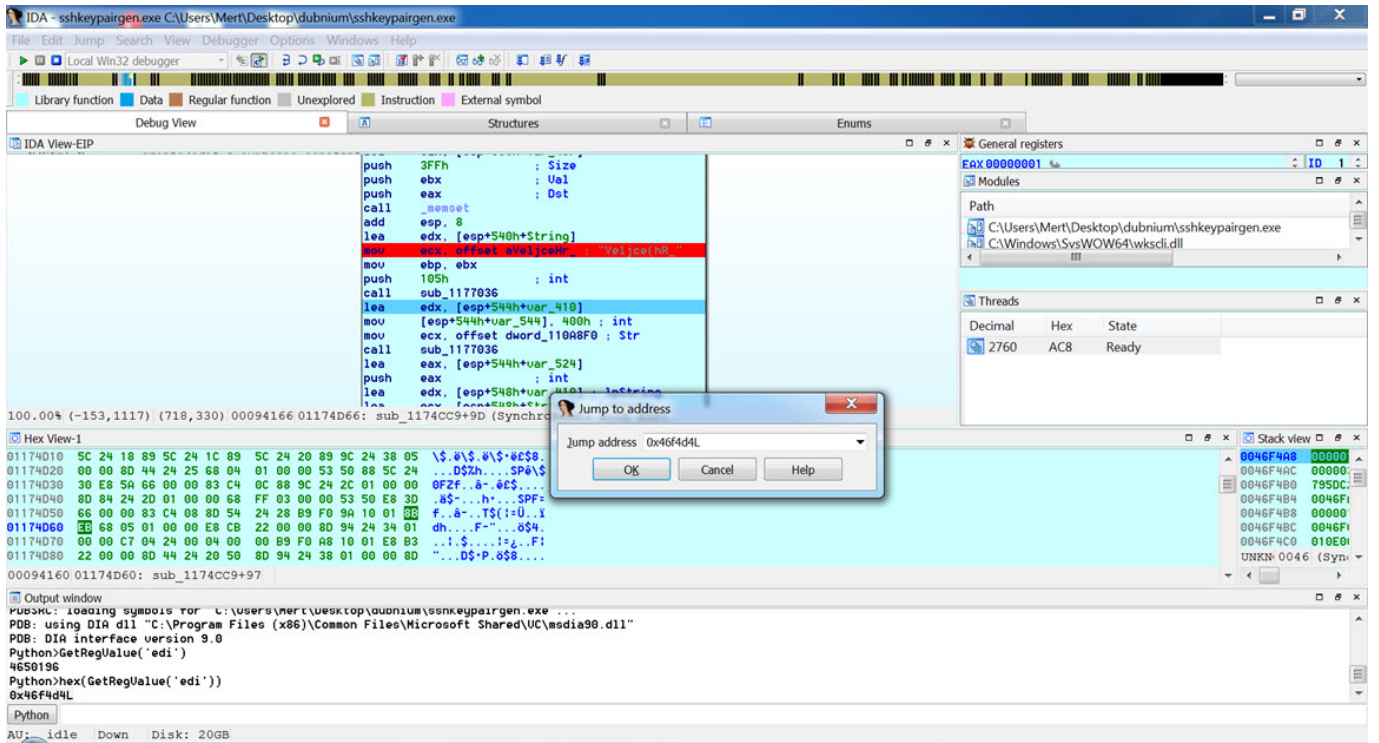


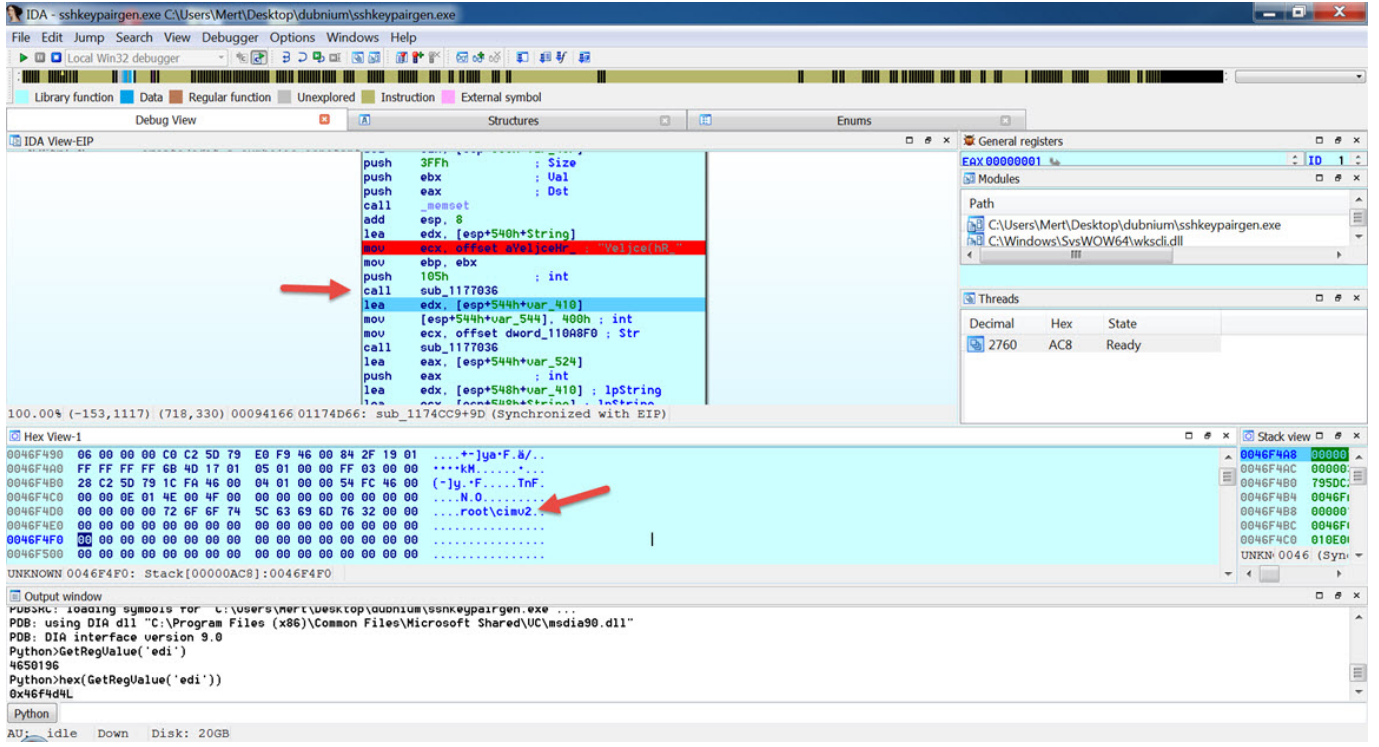
Doğrulama adına sub_1177036 fonksiyonunu çağıran fonksiyonlardan sub_1174CC9 fonksiyonu üzerinde ilerlemeye karar verdim. Karakter dizisini çözen fonksiyonun öncesine (ecx yazmacına Yeljce(hR_ değerinin kopyalandığı) ve sub_1177036 fonksiyonunun çağrıldığı (call sub_1177036) komutun sonrasına (lea edx, [esp+554h+var_410]) kesme noktası koyduktan sonra hata ayıklama (debug) adıma geçtim.





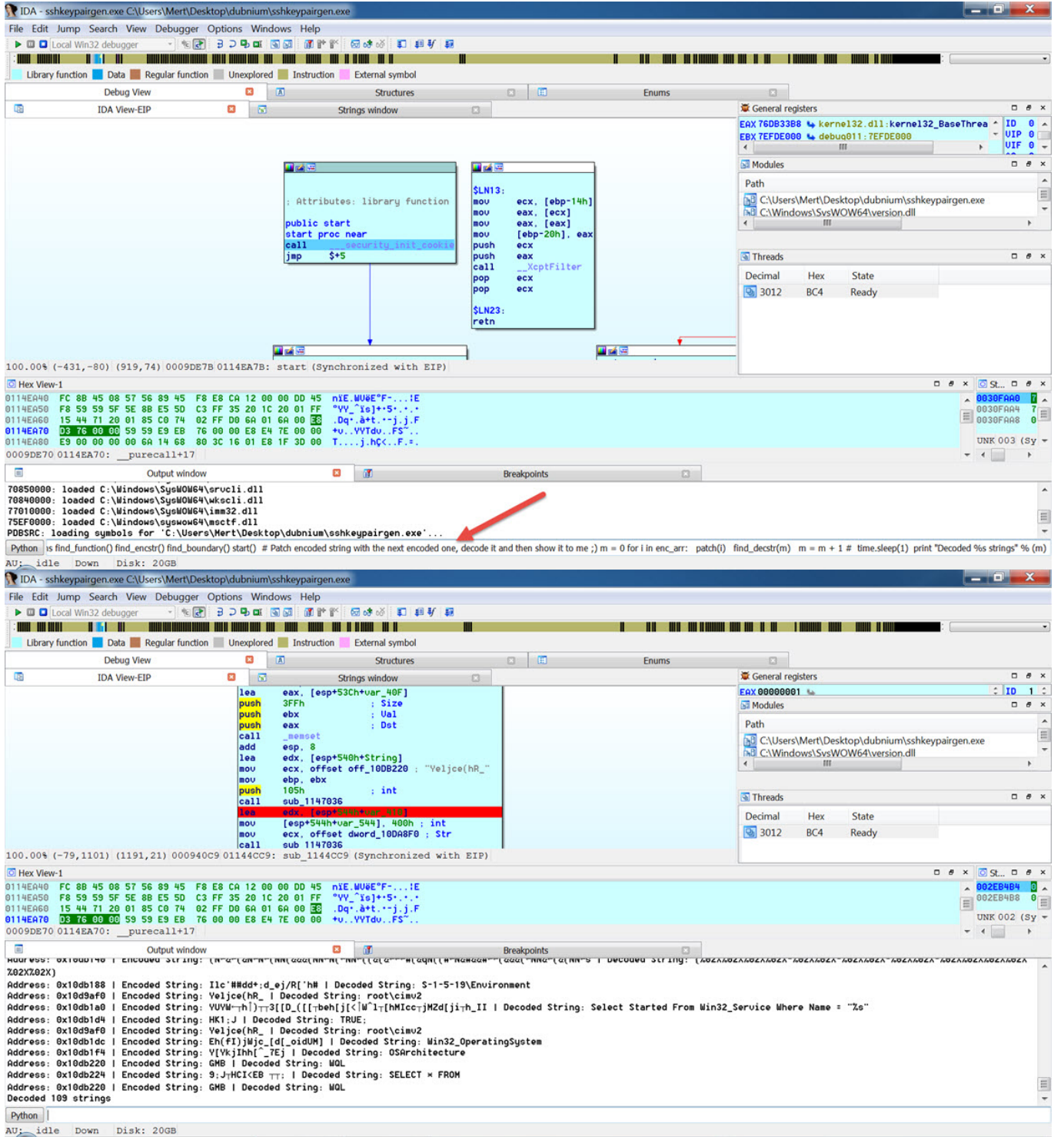
sub_1177036 fonksiyonu çağırıldıktan sonra çözülen karakter dizisinin (string) EDI yazmacında (register) yer aldığını gördüm.





Bunu tespit ettikten sonra IDAPython ile manuel olarak gerçekleştirdiğim adımları otomatik olarak gerçekleştirecek bir betik hazırlamaya karar verdim. Betik temel olarak şu adımları gerçekleştiriyor;

1. Gizlenmiş karakter dizisini çözen sub_1177036 fonksiyonunun çağrıldığı sub_1174CC9 fonksiyonun girişine kesme noktası (breakpoint) koyuyor.
2. Yeljce(hR_ gizlenmiş karakter dizisini ECX yazmacına kopyalayan komutun üzerine kesme noktası (breakpoint) koyuyor. (mov ecx, offset Yeljce(hR_)
3. Karakter dizisini çözen fonksiyona (sub_1177036) iletilen tüm gizlenmiş (encoded) karakter dizilerini teker teker tespit ediyor.
4. Döngü içine girerek sub_1174CC9 fonksiyonunda yer alan ve Yeljce(hR_ gizlenmiş karakter dizisinin yer aldığı adresi (offset Yeljce(hR_), tespit ettiği diğer bir gizlenmiş karakter dizileri ile teker teker değiştiriyor ve karakter dizisini çözen fonksiyona iletiyor.
5. Gizlenmiş karakter dizisi fonksiyon içinde çözüldükten sonra EDI yazmacından çözülmüş karakter dizisini okuyor ve sub_1174CC9 fonksiyonunun başına dönüyor ve aynı adımlardan tekrar geçiyor.



Sonuç olarak bu örnekten yola çıkacak olursak, IDAPython eklentisi ile güvenlik araştırmasında, zararlı yazılım analizinde zaman alabilecek süreçleri hızlandırabilir, çalışmanız için değer üretebilecek çıktılara kısa bir sürede ulaşabilirsiniz.

Bir sonraki yazıda görüşmek dileğiyle herkese güvenli günler dilerim.

Not: GitHub sayfam üzerinde hazırlamış olduğum Dubnium String Decoder isimli betiği bulabilir ve yazı boyunca anlattıklarımın kısa bir özetini kayıt etmiş olduğum aşağıdaki olduğum videoda izleyebilirsiniz.