

بسمه تعالی



مرکز تخصصی آپا دانشگاه رازی

## تحلیل بد افزار با استفاده از نرم افزار Ida Pro

مرکز تخصصی آپا دانشگاه رازی

## فهرست

۳	..... فصل اول	۱
۴	..... ۱,۱ مقدمه	۱,۱
۴	..... ۱,۲ تحلیل ایستای بدافزار	۱,۲
۵	..... فصل دوم	۲
۶	..... ۲,۱ مقدمه	۲,۱
۶	..... ۲,۲ امکانات Ida Pro	۲,۲
۸	..... ۲,۲,۱ نوار راهنمای رنگی	۲,۲,۱
۹	..... ۲,۲,۲ نمایش گرافیکی جریان برنامه	۲,۲,۲
۱۱	..... ۲,۲,۳ پنجره‌های راهنما	۲,۲,۳
۱۱	..... پنجره رشته‌ها	•
۱۱	..... پنجره نام‌ها	•
۱۲	..... پنجره توابع	•

۱ فصل اول

مقدمه

مرکز تخصصی اپنا دانشگاه رازی

## ۱.۱ مقدمه

تحلیل بدافزار معمولاً به دو روش ایستا و پویا انجام می‌شود. در روش پویا، بدافزار اجرا شده و رفتارش تحلیل می‌شود. اما در روش ایستا، تحلیلگر فایل اجرایی بدافزار را با استفاده از تکنیک‌های مهندسی معکوس نرم‌افزار و بدون اجرای برنامه، تحلیل می‌کند. تمرکز این مقاله آموزشی صرفاً بر روی روش ایستا و معرفی مهم‌ترین ابزار مورد استفاده در آن، یعنی نرم‌افزار Ida Pro است. در ادامه به صورت مختصر به معرفی این برنامه و امکانات مفید آن در تحلیل بدافزار می‌پردازیم.

## ۱.۲ تحلیل ایستای بدافزار

در روش ایستای تحلیل بدافزار، تحلیلگر فقط کد برنامه را بررسی می‌کند و به هیچ وجه فایل مخرب را اجرا نمی‌کند. حال این سوال پیش می‌آید که آیا کد منبع (source code) برنامه‌ی مخرب همیشه در دسترس است؟ پاسخ این سوال منفی است. بدافزارهایی که شناسایی می‌شوند، غالباً به صورت فایل اجرایی بوده و دسترسی‌ای به کد آنها وجود ندارد. فلذا تحلیلگر ناچار است برای فهمیدن ساختار و نحوه‌ی کارکرد بدافزار، فایل اجرایی آن را تحلیل کند. از آنجایی که این فایل به زبان ماشین است و برای انسان قابل فهم نیست، لازم است از تکنیک‌هایی برای رسیدن به ساختار و کد اصلی برنامه استفاده شود. یکی از این تکنیک‌ها استفاده از برنامه‌هایی است که اطلاعاتی را در مورد ساختار فایل اجرایی برای ما فراهم می‌کنند. برای مثال نرم‌افزار Peid در مورد فایل‌های اجرایی ویندوز با فرمت PE (Portable Executable) اطلاعات خوبی فراهم می‌کند که دانستن آنها قبل از تحلیل کد برنامه بسیار مفید است. اینجا به جزئیات بیشتری در این مورد نمی‌پردازیم، اما یک تحلیلگر بدافزار لازم است که ساختار فایل‌های PE را به خوبی بشناسد.

بعد از شناسایی اولیه‌ی فایل، نوبت به تحلیل کد برنامه می‌رسد. همانطور که می‌دانیم، وقتی برنامه‌ای با یکی از زبان‌های سطح بالا نوشته می‌شود، برای اجرا روی کامپیوتر، لازم است توسط کامپایلر یا مفسر به زبان ماشین ترجمه شود و در واقع به دستورالعمل‌های ساده‌ای که برای CPU قابل فهم باشند، تبدیل شود. این دستورالعمل‌ها (opcode) معادل دستورات زبان اسمبلی هستند. به این معنا که می‌توان به راحتی آنها را به دستورات معادل اسمبلی برگرداند. به این کار دیس اسمبل کردن می‌گویند که باعث می‌شود کد پیچیده‌ی زبان ماشین برای انسان قابل فهم (human readable) شود و بتوان آن را تحلیل کرد. برای این کار ابزارهای مختلفی وجود دارد. حتی ویندوز هم این امکان را فراهم می‌کند. برای دیس اسمبل کردن یک برنامه در ویندوز می‌توان از دستور dumpbin در خط فرمان ویندوز استفاده کرد. جزئیات دقیق‌تر در این باره را می‌توانید در [MSDN](#) مشاهده کنید.

۲ فصل دوم

معرفی نرم افزار Ida Pro

مرکز تخصصی اپنا دانشگاه رازی

## ۲.۱ مقدمه

در بین دیس اسمبرهای موجود، قدرتمندترین و محبوبترین آنها نرم افزار Ida Pro است. این برنامه در بین تحلیلگران بدافزار، مهندسی معکوس و حتی کرکرها محبوب است. علت آن هم امکانات بسیار خوبی است که این برنامه برای تحلیل کدهای اسمبلی فراهم می کند و در هیچ نرم افزار مشابه دیگری یافت نمی شود. Ida Pro در تحلیل بدافزار، مهندسی معکوس، یافتن آسیب پذیری برنامه ها، کرک نرم افزارها و هر کار دیگری که نیاز به تحلیل یک برنامه اجرایی بدون داشتن کد منبع آن است؛ استفاده می شود.

## ۲.۲ امکانات Ida Pro

همانطور که قبلا اشاره شد، اسمبلی یک زبان سطح پایین و معادل دستورات زبان ماشین است. با وجود اینکه کدهای آن برای انسان قابل فهم است، اما درک ساختار یک برنامه ی پیچیده نوشته شده به این زبان، کار راحتی نیست. حتی ساده ترین بدافزارها هم می توانند چند صد و یا حتی چند هزار خط کد اسمبلی داشته باشند. کدهایی که ترکیبی از داده، دستورالعمل، توابع کتابخانه ای، فراخوانی های سیستمی، کدهای اضافه شده توسط کامپایلر و ... هستند. به همین دلیل ابزار دیس اسمبلر مورد استفاده ی تحلیلگر بسیار مهم است. Ida با امکاناتی که فراهم می کند، بخشی از این مشکلات را حل کرده است. از جمله امکانات این نرم افزار می توان به موارد زیر اشاره کرد:

- ✓ داشتن رابط گرافیکی قدرتمند با امکانات بشمار نظیر چندین پنجره که اطلاعات مفیدی فراهم می کنند.
- ✓ نمایش گرافیکی جریان برنامه با ابزارهایی مثل گراف و فلوجارت
- ✓ شناسایی بسیاری از کتابخانه های فراخوانی شده و نوع داده ها
- ✓ راحتتر کردن تحلیل با امکاناتی نظیر اضافه کردن کامنت، تغییر نام متغیرها و توابع و ...
- ✓ جدا کردن section های برنامه از هم و همینطور تشخیص کد کاربر از سایر کدهای اضافه شده به برنامه
- ✓ امکان دیباگ برنامه و تحلیل پویای بدافزار به همراه نمایش محتوای سگمنت ها، پشته و ...
- ✓ امکان پرش (jump) در بین کدها و جستجوی قدرتمند
- ✓ پشتیبانی از ۶۴ بیت
- ✓ امکان افزودن پلاگین به برنامه

در ادامه برای آشنایی با امکانات و نحوه ی کار با این نرم افزار، از یک مثال ساده استفاده کرده و برخی از ابزارهای مفید Ida را معرفی می کنیم.

کد زیر یک رشته را از کاربر دریافت می‌کند. آن را با کلمه‌ی عبور مقایسه کرده و به کاربر پاسخ می‌دهد:

```
int _tmain(int argc, _TCHAR* argv[])
{
#define password "ApaRazi"
int aa=11;
char pwd[100];
printf("Please enter the password:\n");
scanf("%s", pwd);
if ( strcmp(pwd, password) == 0 )
printf("Congratulation!\n");
else
printf("Wrong password");
return 0;
}
```

این برنامه را با MS visual studio کامپایل کرده و سپس فایل اجرایی (.exe) آن را با Ida Pro دیس اسمبل کرده‌ایم که در تصویر ۱ قابل مشاهده است. این تصویر پنجره‌ی IDA View را نشان می‌دهد که شامل کدهای اسمبلی است. پنجره‌ی دیگری هم به نام HEX View وجود دارد که کدها را به زبان ماشین و به صورت هگز نمایش می‌دهد.

سمت چپ تصویر، section برنامه و آدرس حافظه را نشان می‌دهد. سمت راست هم کد اسمبلی برنامه است. در ابتدای باز کردن فایل اجرایی، Ida نقطه‌ی ورود برنامه (entry point) را به ما نشان می‌دهد که همانطور که در تصویر پیداست، تابع main نیست. بلکه رویه‌ی start است که قبل از main اجرا می‌شود.

```
.text:0041106E ; ===== S U B R O U T I N E =====
.text:0041106E
.text:0041106E ; Attributes: thunk
.text:0041106E
.text:0041106E public start
.text:0041106E start proc near
.text:0041106E jmp start_0
.text:0041106E start endp
.text:0041106E
.text:00411073 ; ===== S U B R O U T I N E =====
.text:00411073
.text:00411073 ; Attributes: thunk
.text:00411073 sub_411073 proc near ; DATA XREF: sub_4118C0+C↓o
.text:00411073 ; sub_4120D0+C↓o ...
.text:00411073 jmp sub_412EA0
.text:00411073 sub_411073 endp
.text:00411073
.text:00411078 ; [00000005 BYTES: COLLAPSED FUNCTION j_lock. PRESS CTRL-NUMPAD+ TO EXPAND]
.text:0041107D ; -----
.text:0041107D jmp loc_413534
.text:00411082
```

تصویر ۱: رویه‌ی start در پنجره‌ی IDA View

تصویر ۲ بخشی از تابع main برنامه را نشان می‌دهد. Ida Pro نمی‌تواند نام توابع را برگرداند اما این امکان را به ما می‌دهد که بعد از شناسایی تابع، نام دلخواه‌مان را روی آن بگذاریم. در این مثال با شناسایی main نام آن را تغییر داده‌ایم (با کلید n).

```
.text:004113A0 ; ===== SUBROUTINE =====
.text:004113A0
.text:004113A0 ; Attributes: bp-based frame
.text:004113A0
.text:004113A0 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:004113A0 main          proc near          ; CODE XREF: j_main↑j
.text:004113A0
.text:004113A0 var_13C      = byte ptr -13Ch
.text:004113A0 Str1        = byte ptr -78h
.text:004113A0 var_C       = dword ptr -0Ch
.text:004113A0 var_4       = dword ptr -4
.text:004113A0 argc        = dword ptr 8
.text:004113A0 argv        = dword ptr 0Ch
.text:004113A0 envp        = dword ptr 10h
.text:004113A0
```

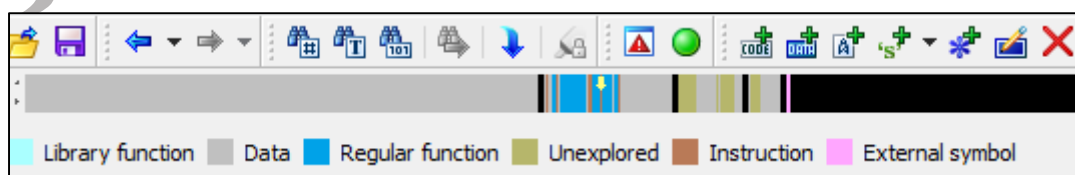
تصویر ۲: تابع main در پنجره‌ی IDA View

عبارت CODE XREF که با فلش آبی به آن اشاره شده، نشان می‌دهد که کدام قسمت برنامه تابع فعلی را فراخوانی کرده و یا به آن پرش می‌کند. این امکان در پیگیری جریان برنامه برای تحلیل کدها بسیار مفید است. در ادامه به برخی از امکانات مفید Ida در تحلیل کدها می‌پردازیم.

## ۲.۲.۱ نوار راهنمای رنگی

برای تشخیص نوع کدها می‌توان از نوار راهنمای Ida استفاده کرد. در هر کجای برنامه که باشیم، این نوار رنگی به ما کمک می‌کند نوع کدها را از هم تشخیص دهیم. همانطور که در تصویر ۳ مشاهده می‌شود، با رفتن به قسمت‌های مختلف کد، فلش زرد رنگ روی نوار راهنما حرکت کرده و رنگ متناظر با آن را نشان می‌دهد. هر کدام از رنگ‌ها معرف کد خاصی است:

- ✓ آبی روشن : توابع کتابخانه‌ای
- ✓ قرمز: کدهای ایجاد شده توسط کامپایلر
- ✓ آب تیره : کدهای کاربر
- ✓ صورتی : کدهای وارد شده به برنامه (imported)
- ✓ خاکستری : داده‌های تعریف شده
- ✓ قهوه ای : داده‌های تعریف نشده



تصویر ۳: نوار راهنمای رنگی



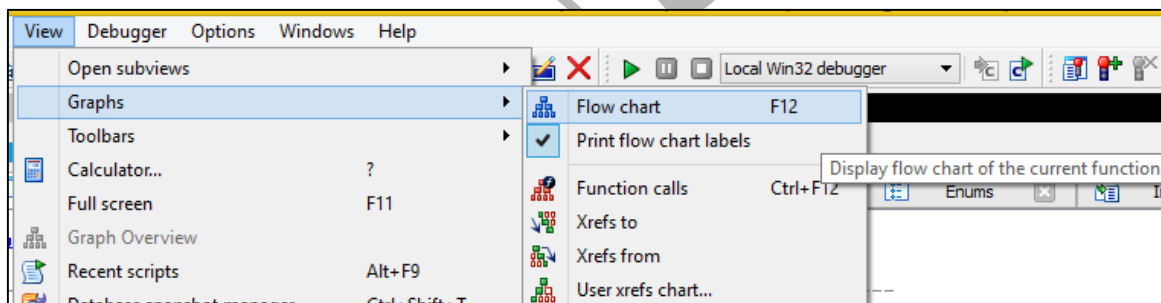
این رنگ‌بندی‌ها در آدرس‌دهی سمت چپ دستورات هم قابل مشاهده است. برای مثال در تصویر ۴ زیر از رنگ قهوه‌ای برای نمایش داده‌های تعریف نشده و رنگ خاکستری برای داده‌های تعریف شده استفاده شده است.

```
.rdata:00415DB3 db 0
.rdata:00415DB4 db 0
.rdata:00415DB5 db 0
.rdata:00415DB6 db 0
.rdata:00415DB7 db 0
.rdata:00415DB8 ; CHAR aStackAreaAro_0[]
.rdata:00415DB8 aStackAreaAro_0 db 'Stack area around
.rdata:00415DB8
.rdata:00415DB8 db 'rupted',0Ah,0
.rdata:00415E01 align 10h
.rdata:00415E10 aSSSS db '%s%s%s%s',0
```

تصویر ۴: رنگ‌بندی آدرس‌های حافظه

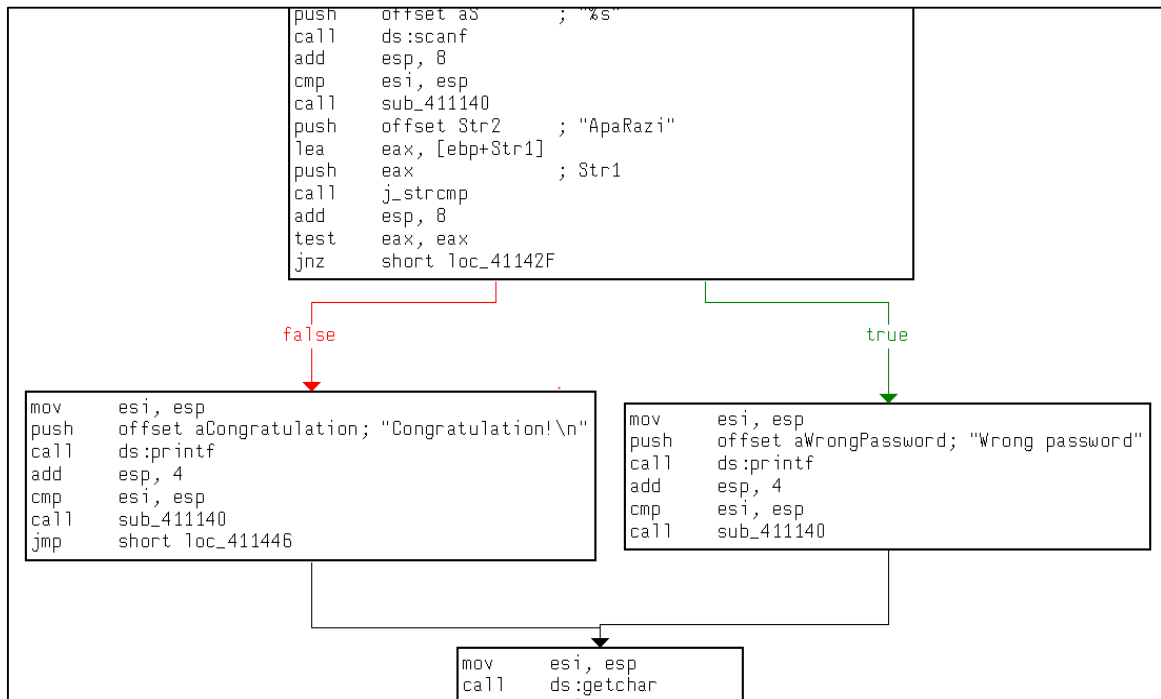
## ۲.۲.۲ نمایش گرافیکی جریان برنامه

یکی دیگر از امکانات مفید Ida، نمایش گرافیکی جریان برنامه است. همانطور که در تصویر ۵ قابل مشاهده است، از منوی view می‌توان به این ابزار دسترسی داشت.



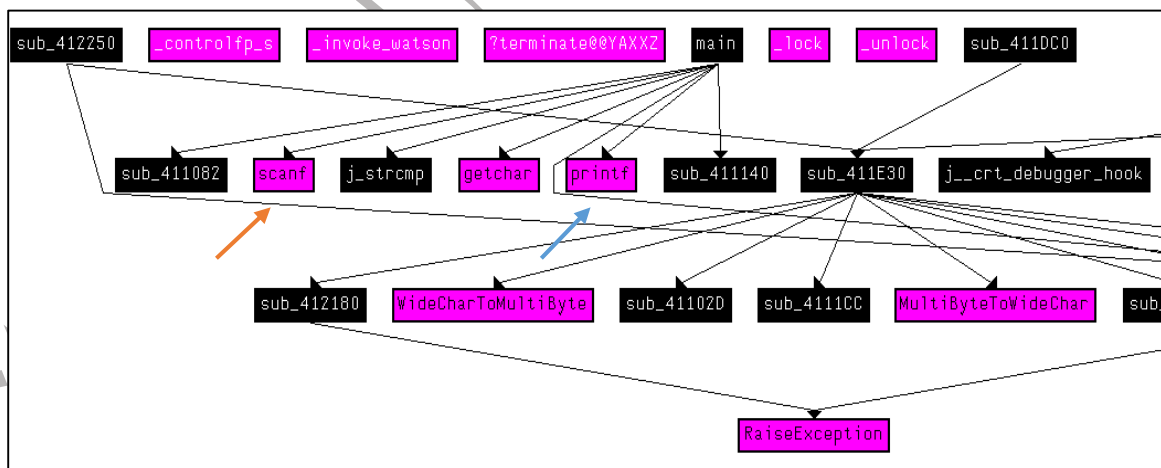
تصویر ۵: دسترسی به ابزارهای گرافیکی از طریق منوی View

تصویر ۶ فلوجارت تابع main را نشان می‌دهد. این ابزار خوانایی و درک کدها را بسیار بالا می‌برد.



تصویر ۶: فلوجارت تابع main برنامه

تصویر ۷ گراف فراخوانی های توابع را نشان می دهد. همانطور که در تصویر پیداست، با این ابزار به راحتی می توان توابع برنامه و فراخوانی های سیستمی را شناسایی کرد. هم چنین می توان تابع main را در برنامه پیدا کرد. در این مثال، main توابعی مثل printf یا scanf را فراخوانی کرده است.



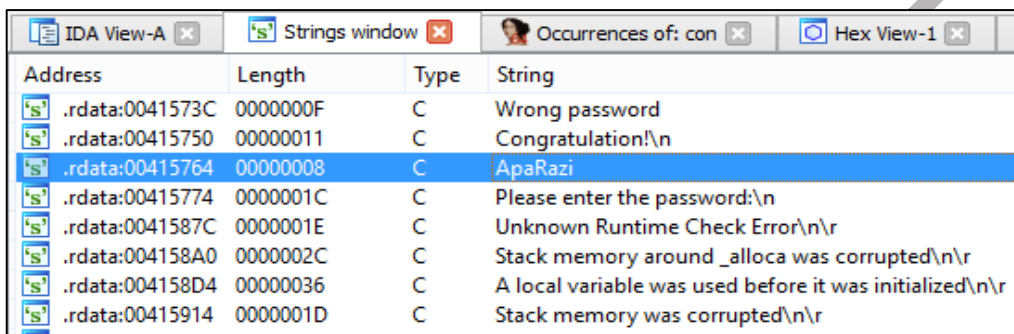
تصویر ۷: گراف فراخوانی های توابع

## ۲.۲.۳ پنجره‌های راهنما

یکی دیگر از امکانات مهم Ida Pro، پنجره‌های مفید آن است که اطلاعات بسیار خوبی برای تحلیلگر فراهم می‌کند. در ادامه تعدادی از آن‌ها را معرفی می‌کنیم.

### • پنجره رشته‌ها

این پنجره تمام رشته‌های به‌کاررفته در برنامه را نمایش می‌دهد. همانطور که در تصویر ۸ پیداست، حتی پسورد برنامه را هم می‌توان مشاهده کرد.

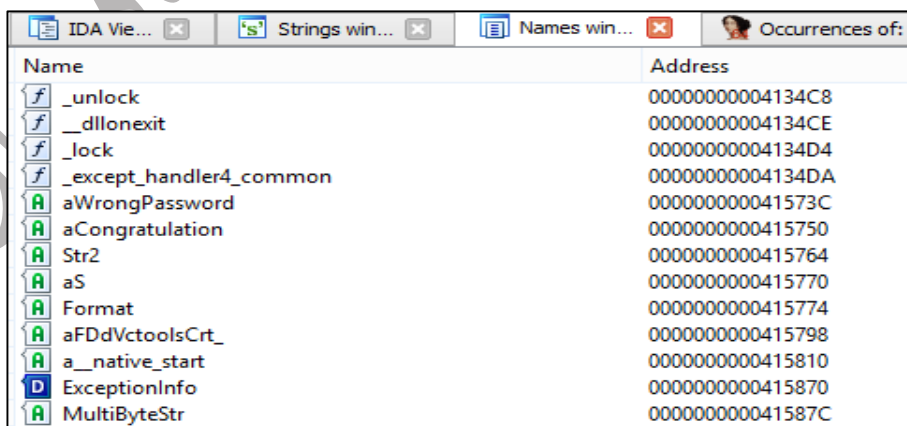


Address	Length	Type	String
.rdata:0041573C	0000000F	C	Wrong password
.rdata:00415750	00000011	C	Congratulation!\n
.rdata:00415764	00000008	C	ApaRazi
.rdata:00415774	0000001C	C	Please enter the password:\n
.rdata:0041587C	0000001E	C	Unknown Runtime Check Error!\n\r
.rdata:004158A0	0000002C	C	Stack memory around _alloca was corrupted\n\r
.rdata:004158D4	00000036	C	A local variable was used before it was initialized\n\r
.rdata:00415914	0000001D	C	Stack memory was corrupted\n\r

تصویر ۸: پنجره رشته‌ها

### • پنجره نام‌ها

این پنجره نام‌های موجود در برنامه و آدرس آن‌ها را نمایش می‌دهد. در تصویر ۹ تعدادی از این نام‌ها قابل مشاهده است. علائمی که کنار نام‌ها وجود دارد، نشان دهنده‌ی نوع آن‌ها هستند. برای مثال، f نشانگر توابع (function)، A رشته‌ها و D داده‌های تعریف شده‌است.

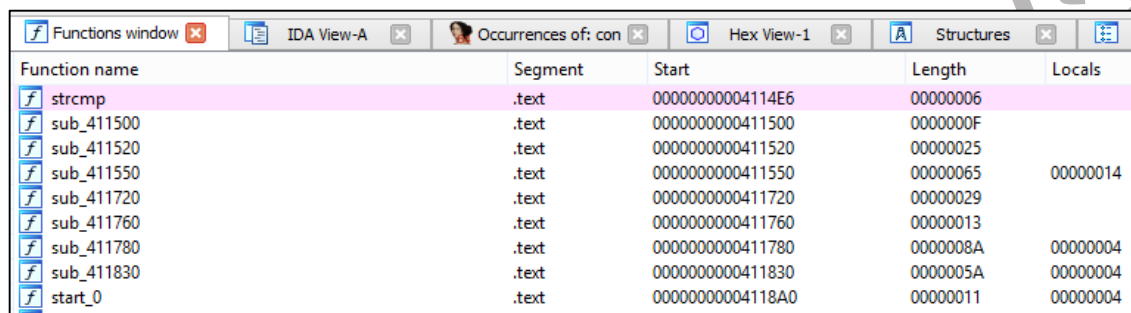


Name	Address
f _unlock	0000000004134C8
f _dllonexit	0000000004134CE
f _lock	0000000004134D4
f _except_handler4_common	0000000004134DA
A aWrongPassword	00000000041573C
A aCongratulation	000000000415750
A Str2	000000000415764
A aS	000000000415770
A Format	000000000415774
A aFDdVctoolsCrt_	000000000415798
A a_native_start	000000000415810
D ExceptionInfo	000000000415870
A MultiByteStr	00000000041587C

• تصویر ۹: پنجره نام‌ها

## • پنجره توابع

این پنجره، نام توابع به کاررفته در برنامه دیس اسمبل شده را لیست کرده و اطلاعاتی در مورد آدرس شروع، طول و آرگومان‌های آنها فراهم می‌کند. در تصویر ۱۰ نام برخی از توابع موجود در برنامه قابل مشاهده است.



Function name	Segment	Start	Length	Locals
strcmp	.text	0000000004114E6	00000006	
sub_411500	.text	000000000411500	0000000F	
sub_411520	.text	000000000411520	00000025	
sub_411550	.text	000000000411550	00000065	00000014
sub_411720	.text	000000000411720	00000029	
sub_411760	.text	000000000411760	00000013	
sub_411780	.text	000000000411780	0000008A	00000004
sub_411830	.text	000000000411830	0000005A	00000004
start_0	.text	0000000004118A0	00000011	00000004

تصویر ۱۰: پنجره توابع