

.net themida – reto 2007 lastdragon



Alejandro Torres
(torrescrack)

En este escrito muestro como cazar un password de una aplicación .net con una teoría que hice hace tiempo que era cazar los password de una app .net con ollydbg

Alejandro Torres
torrescrack.blogspot.com



<http://www.facebook.com/yo.torrescrack>



<https://twitter.com/TorresCrack248>



www.torrescrack.blogspot.com

INTRODUCCION

En esta ocasión hablaremos de un reto que encontré en la cueva de lastdragon, lo cual por cierto es un sitio el cual me entretuve leyendo varios artículos y entre algunas de esas cosas interesantes fue un reto de cracking (el único que encontré) y bueno el único que resolvió fue un buen reverser, no sé si recuerden a “dapaf” , lo cual comentaba el colega es un themida, eso fue lo que me llamo la atención y lo que me animo a escribir sobre este, ya que es un .net con themida y me gustan los retos, también que no había tutorial de cómo lo resolvió dapaf entonces aproveche y me puse manos a la obra lo cual me ayudo para recordar una teoría que escribí y publique hace un par de años en crackslatinos sobre cazar contraseñas de .net con ollydbg en unos sencillos pasos, de esa manera muestro que es posible cazar las contraseñas con el debugger aun sin desempacar el mismo, lo cual puede ayudarnos en caso de estar acorralados y no tener por donde entrar a la aplicación, dejare de aburrirlos y vamos por él.

La información que nos brindan del binario es:

Este reto consiste en obtener la clave privada la cual esta almacenada dentro del binario

Características propias del binario.

Fue escrito en C# y requiere Framework.NET 2.0 para ejecutarse en MS/Windows

El ejecutable es tipo mixto (IL y PE)

Como pista para el crackeo, la clave es un objeto String y es comparada en un if

[Descargar el binario del reto](#)

Bien nos comenta que es un .net y que hace una simple comparación “ IF “ por lo tanto es apto para usar nuestra teoría anterior y los que no la hayan leído les dejo el enlace...

<http://torrescrack.blogspot.mx/2012/12/teoria-2-alex-tecnica-para-cazar.html>

los que ya la conocen pues será más fácil y lo que no la conocen pues vamos a hacerlo ahora.

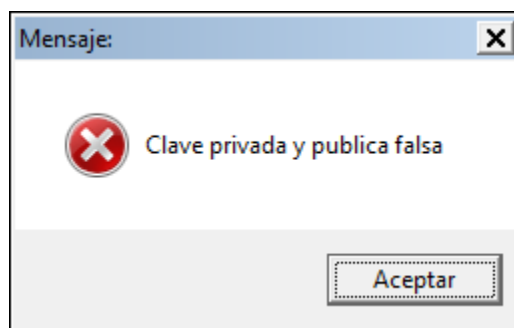
Usamos RDG packer protector para ver si esta empacado con lo que comentaba dapaf , asi que veamos:

Como vemos esta empacado con Themida

Seguimos y abriremos olly, por cierto creo que ya existen demasiados tutoriales sobre que Plugins usar, como configurarlos, etc, para que puedan correr nuestros binarios empacados con este tipo de packers tan poderosos y no ser detectados hasta hay algunas modificaciones de olly que andan por la red para poder bypassar los anti debugger de estos packers y para no extenderme tanto en este escrito solo mostrare como lograr cazar el password, si alguno tiene problemas en correrlo con el olly puede mandarme un correo a tora_248@hotmail.com y les mando el olly con los Plugins para que puedan correrlo, y no ser detectados, claro pero intenten investigar por su cuenta y leer, yo espero pronto poder escribir detalladamente sobre esto en otro escrito, mientras veamos como cazar un password .net packeado con themida sin tocar el packer, así que seguimos, cargamos nuestro olly debugger y caemos en el entrypoint, démosle “run” y seguimos hasta aparezca la ventana del crackme



Bien como de costumbre metamos algo en la casilla, algo como “torrescrack” y probemos si por primera vez en este crackme acertamos a la primera:



Como era de esperarse, un MessageBox, las cosas van a nuestro favor pues ya tenemos por donde cazarlo, directamente por el mensaje MessageBox

Ahora coloquemos un BreakPoint en “MessageBoxExW” y “MessageBoxExA” en alguna de ellas tiene que parar la ejecución y poder seguir con el método que les quiero volver a mostrar a quien no lo conoce, veamos:

75F0E987	8045 A4	LEA EAX, [EBP-5C]	
75F0E98A	50	PUSH EAX	
75F0E98B	E8 CFF9FFFF	CALL 75F0E98F	
75F0E98C	C9	LEAVE	
75F0E9C1	C2 0400	RETN 4	
75F0E9C4	90	NOP	
75F0E9C5	90	NOP	
75F0E9C6	90	NOP	
75F0E9C7	90	NOP	
75F0E9C8	90	NOP	
75F0E9C9	8BFF	MOV EDI, EDI	ID_X USER32, MessageBoxExA(hOwner, Text, Caption, Type, LanguageID)
75F0E9CB	55	PUSH EBP	
75F0E9CC	8BEC	MOV EBP, ESP	
75F0E9CE	6A FF	PUSH -1	
75F0E9D0	FF75 18	PUSH DWORD PTR SS:[EBP+18]	
75F0E9D3	FF75 14	PUSH DWORD PTR SS:[EBP+14]	
75F0E9D6	FF75 10	PUSH DWORD PTR SS:[EBP+10]	
75F0E9D9	FF75 0C	PUSH DWORD PTR SS:[EBP+0C]	
75F0E9DC	FF75 08	PUSH DWORD PTR SS:[EBP+08]	
75F0E9DF	E8 B1FDFFFF	CALL MessageBoxTimeoutA	
75F0E9E4	5D	POP EBP	
75F0E9E5	C2 1400	RETN 14	
75F0E9E8	90	NOP	
75F0E9E9	90	NOP	
75F0E9EA	90	NOP	
75F0E9EB	90	NOP	
75F0E9EC	90	NOP	
75F0E9ED	8BFF	MOV EDI, EDI	ID_X USER32, MessageBoxExW(hOwner, Text, Caption, Type, LanguageID)
75F0E9EF	55	PUSH EBP	
75F0E9F0	8BEC	MOV EBP, ESP	

Ya colocados los BreakPoint podemos seguir y metemos cualquier clave de nuevo y presionamos y tendrá que parar la ejecución en alguno de esos:

75F0E9E9	90	NOP	
75F0E9EA	90	NOP	
75F0E9EB	90	NOP	
75F0E9EC	90	NOP	
75F0E9ED	8BFF	MOV EDI, EDI	ID_X USER32, MessageBoxExW(hOwner, Text, Caption, Type, LanguageID)
75F0E9EF	55	PUSH EBP	
75F0E9F0	8BEC	MOV EBP, ESP	
75F0E9F2	6A FF	PUSH -1	
75F0E9F4	FF75 18	PUSH DWORD PTR SS:[EBP+18]	

0012EE58	00000000	!-#	*LanguageID = LANG_NEUTRAL
0012EE54	0012EE7C	!-#	
0012EE58	032C20F3	% , ♥	RETURN to 032C20F3
0012EE5C	000E0790	E-#	GUID ISequentialStream
0012EE60	03654FA8	¿0e#	UNICODE "Clave privada y publica falsa"
0012EE64	03654F04	¿0e#	UNICODE "Mensaje:"
0012EE68	00000010	¿0e#	
0012EE6C	F0A0C619	¿0e#	
0012EE70	00000000	¿0e#	
0012EE74	03654FE8	¿0e#	

Paramos hay y claramente podemos ver en el stack o pila la dirección de retorno y aquí empieza el método, recordemos después de pasar el “ret” caemos en una parte que usan todos los .net, es como un runtime que todos usan, hace tiempo iba escribir sobre esto pero no dispongo de mucho tiempo para extenderme tanto en un tema nuevo, no nos desviemos y veamos la dirección de retorno y la zona que les comento

CPU Stack

Address Value ASCII Comments

0012EE58 032C20F3 ó,? ; RETURN to 032C20F3

La zona que debemos tener siempre presente que nos indica que estamos en el runtime.

032C20CD	74 03	JE SHORT 032C20D2
032C20CF	83C0 0C	ADD EAX,0C
032C20D2	50	PUSH EAX
032C20D3	8B46 E4	MOV EAX,DWORD PTR DS:[ESI-1C]
032C20D6	85C0	TEST EAX,EAX
032C20D8	74 03	JE SHORT 032C20D0
032C20DA	83C0 0C	ADD EAX,0C
032C20DD	50	PUSH EAX
032C20DE	FF76 18	PUSH DWORD PTR DS:[ESI+18]
032C20E1	C643 08 00	MOV BYTE PTR DS:[EBX+8],0
032C20E5	F643 04 5F	TEST BYTE PTR DS:[EBX+4],5F
032C20E9	75 2B	JNE SHORT 032C2116
032C20EB	8B46 08	MOV EAX,DWORD PTR DS:[ESI+8]
032C20EE	8B40 14	MOV EAX,DWORD PTR DS:[EAX+14]
032C20F1	FF10	CALL DWORD PTR DS:[EAX]
032C20F3	C643 08 01	MOV BYTE PTR DS:[EBX+8],1
032C20F7	833D 94D37D64	CMP DWORD PTR DS:[mscorwks.647DD394],0
032C20FE	75 1D	JNE SHORT 032C2110
032C2100	8D65 F8	LEA ESP,[EBP-8]
032C2103	897B 0C	MOV DWORD PTR DS:[EBX+0C],EDI
032C2106	89EC	MOV ESP,EBP
032C2108	5D	POP EBP
032C2109	83C4 04	ADD ESP,4
032C210C	5F	POP EDI
032C210D	5E	POP ESI
032C210E	5B	POP EBX
032C210F	5D	POP EBP
032C2110	83C4 0C	ADD ESP,0C

Bien ahora lleguemos al “ret” y pasamos con f8 ese “ret” y caemos en una zona como la que mostrare y aquí el otro “tip”, donde nos guiaremos por el nombre de la sección donde estamos ubicados, lo cual si nos fijamos estaremos ubicados en la sección del “System_Windows_forms_ni”

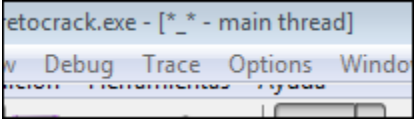
```
retocrack.exe - [*_* - main thread, module System_Windows_Forms_ni]
ew Debug Trace Options Windows Help

F31FD10 8840 B0 MOV ECX, DWORD PTR SS:[EBP-50]
F31FD13 E8 B06B9BFF CALL SEC068C8
F31FD18 8BC8 MOV ECX, EAX
F31FD1A E8 F1000000 CALL SF31FE10
F31FD1F 8945 CC MOV DWORD PTR SS:[EBP-34], EAX
F31FD22 C745 E0 000000 MOV DWORD PTR SS:[EBP-20], 0
F31FD29 C745 E4 FC0000 MOV DWORD PTR SS:[EBP-1C], 0FC
F31FD30 68 75FD315E PUSH OFFSET SF31FD75
F31FD35 EB 00 JMP SHORT SF31FD37
F31FD37 E8 70D8F2FF CALL SF24D5AC
F31FD3C 8840 D0 MOV ECX, DWORD PTR SS:[EBP-30]
F31FD3F E8 CC9AA0FF CALL SED29810
F31FD44 58 POP EAX
F31FD45 FFE0 JMP EAX
F31FD47 807D BC LEA EDI, [EBP-44]
F31FD4A 33C0 XOR EAX, EAX
F31FD4C AB STOS DWORD PTR ES:[EDI]
F31FD4D AB STOS DWORD PTR ES:[EDI]
F31FD4E FF75 D4 PUSH DWORD PTR SS:[EBP-2C]
F31FD51 804D BC LEA ECX, [EBP-44]
F31FD54 8B55 B4 MOV EDX, DWORD PTR SS:[EBP-4C]
F31FD57 E8 644798FF CALL SECA44C0
F31FD5C 8045 BC LEA EAX, [EBP-44]
F31FD5F FF70 04 PUSH DWORD PTR DS:[EAX+4]
F31FD62 FF30 PUSH DWORD PTR DS:[EAX]
F31FD64 6A 00 PUSH 0
F31FD66 33D2 XOR EDX, EDX
F31FD68 804A 07 LEA ECX, [EDX+7]
F31FD6B E8 309198FF CALL SECA8EA0
F31FD70 8B45 CC MOV EAX, DWORD PTR SS:[EBP-34]
F31FD73 EB 09 JMP SHORT SF31FD7E
F31FD75 C745 E4 000000 MOV DWORD PTR SS:[EBP-1C], 0
F31FD7C EB C9 JMP SHORT SF31FD47
F31FD7E 8B75 B8 MOV ESI, DWORD PTR SS:[EBP-48]
F31FD81 8B7D B8 MOV EDI, DWORD PTR SS:[EBP-78]
F31FD84 897E 0C MOV DWORD PTR DS:[ESI+0C], EDI
F31FD87 8065 F4 LEA ESP, [EBP-0C]
F31FD8A 5B POP EBX
F31FD8B 5E POP ESI
F31FD8C 5F POP EDI
F31FD8D 5D POP EBP
F31FD8E C2 1800 RETN 18
F31FD91 CC INT3
```

Ahora bien debemos de ver en el olly en el título de la ventana, el nombre de la sección donde estamos ubicados así que debemos pasar los return siguientes hasta ver algo como “Main Thread”, podemos ver claramente ese “retn” llegamos hay y lo seguimos pasando con f8 y cuando lo pasamos

seguimos en la misma sección, sigamos pasando más return hasta ver la zona que les comento.

Después de pasar este último return al parecer fue el siguiente de donde estábamos, podemos ver la zona que les comento y algo interesante ya que estamos ubicados en la sección donde ejecutan código los .net, veamos:



031F0600	85C0	TEST EAX,EAX
031F060F	74 1A	JE SHORT 031F062B
031F0611	6A 00	PUSH 0
031F0613	6A 40	PUSH 40
031F0615	8B0D 70206404	MOV ECX,DWORD PTR DS:[4642070]
031F061B	8B15 68206404	MOV EDX,DWORD PTR DS:[4642068]
031F0621	E8 7AF2125C	CALL SF31F8A0
031F0626	BF 01000000	MOV EDI,1
031F062B	85FF	TEST EDI,EDI
031F062D	75 15	JNE SHORT 031F0644
031F062F	6A 00	PUSH 0
031F0631	6A 10	PUSH 10
031F0633	8B0D 74206404	MOV ECX,DWORD PTR DS:[4642074]
031F0639	8B15 68206404	MOV EDX,DWORD PTR DS:[4642068]
031F063F	E8 5CF2125C	CALL SF31F8A0
031F0644	5E	POP ESI
031F0645	5F	POP EDI
031F0646	5D	POP EBP
031F0647	C2 0400	RETN 4

Aquí tenemos algo interesante, porque? porque vemos que estamos en la sección donde ejecuta código y acabamos de salir de la llamada que hizo mostrar el “MessageBox” , en los .net no veremos llamadas a las apis como estamos acostumbrados a ver pero necesita los parámetros antes de entrar a las rutinas, si vemos arriba antes de entrar al “call” de la dirección 0x31f063f, esta como guardando unos punteros los registros para después usarlos dentro del “call” veamos que contienen esas direcciones de memoria que sospecho son las strings que usara el MessageBox, veamos el contenido que guarda en “ecx”,

Address	Hex dump	ASCII
03654F9C	70 0B 52 63 1E 00 00 00	p3Rc▲
03654FA4	1D 00 00 00 43 00 6C 00	# C l
03654FAC	61 00 76 00 65 00 20 00	a v e
03654FB4	70 00 72 00 69 00 76 00	p r i v
03654FBC	61 00 64 00 61 00 20 00	a d a
03654FC4	79 00 20 00 70 00 75 00	y p u
03654FCC	62 00 6C 00 69 00 63 00	b l i c
03654FD4	61 00 20 00 66 00 61 00	a f a
03654FDC	6C 00 73 00 61 00 00 00	l s a
03654FE4	00 00 00 00 9C C5 14 64	stnd
03654FEC	00 00 00 00 00 00 00 00	

Como dije entonces está guardando lo que mostrara en el mensaje “Clave Privada y Publica Falsa”, por lo tanto vamos, analicemos la situación y es tan sencillo, pues más arriba vemos la misma situación, guarda datos en los registros “ecx” y “edx” y después una llamada y bueno sería lógico creer que es el mensaje de “el password es correcto!”, no nos quedemos con la duda y veamos:

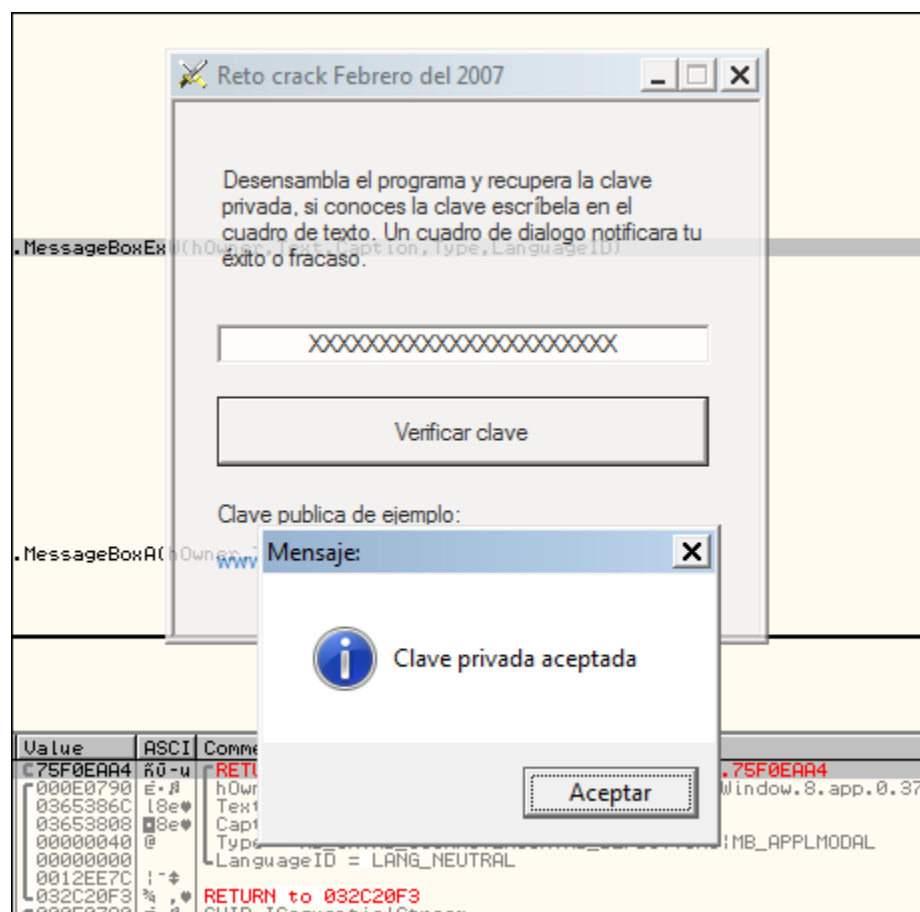
31F05F8	8B01	MOV EAX,DWORD PTR DS:[ECX]
31F05FA	FF90 64010000	CALL DWORD PTR DS:[EAX+164]
31F0600	8B15 6C206404	MOV EDX,DWORD PTR DS:[464206C]
31F0606	8BC8	MOV ECX,EAX
31F0608	E8 73162E60	CALL 634D1C80
31F060D	85C0	TEST EAX,EAX
31F060F	74 1A	JE SHORT 031F062B
31F0611	6A 00	PUSH 0
31F0613	6A 40	PUSH 40
31F0615	8B0D 70206404	MOV ECX,DWORD PTR DS:[4642070]
31F061B	8B15 68206404	MOV EDX,DWORD PTR DS:[4642068]
31F0621	E8 7AF2125C	CALL 5F31F8A0
31F0626	BF 01000000	MOV EDI,1
31F062B	85FF	TEST EDI,EDI
31F062D	75 15	JNE SHORT 031F0644
31F062F	6A 00	PUSH 0
31F0634	20 10	PUSH 10

dress	Hex dump	ASCII
654F5C	70 0B 52 63 17 00 00 00	p3Rc▲
654F64	16 00 00 00 43 00 6C 00	# C l
654F6C	61 00 76 00 65 00 20 00	a v e
654F74	70 00 72 00 69 00 76 00	p r i v
654F7C	61 00 64 00 61 00 20 00	a d a
654F84	61 00 63 00 65 00 70 00	a c e p
654F8C	74 00 61 00 64 00 61 00	t a d a
654F94	00 00 00 00 00 00 00 80	ç

Bien entonces como vemos es sencillo cazar el password, pues en un poco más arriba hay un “test eax, eax” y un salto condicional por lo tanto más arriba en la llamada de arriba está haciendo una comparación, y como mencione anteriormente arriba de la llamada está guardando algunos punteros que dan al offset donde está ubicado algún parámetro que usara dentro de esa llamada, veamos:

Address	Hex dump	ASCII
03654F1C	70 0B 52 63 17 00 00 00	p0Rc
03654F24	16 00 00 00 61 00 71 00	u i l a q
03654F2C	75 00 69 00 6C 00 65 00	s 0 7 a
03654F34	73 00 30 00 37 00 61 00	r g a m
03654F3C	72 00 67 00 61 00 6D 00	e n o n
03654F44	65 00 6E 00 6F 00 6E 00	2 0 0 8
03654F4C	32 00 30 00 30 00 38 00	
03654F54	00 00 00 00 00 00 00 00	
03654F5C	70 0B 52 63 17 00 00 00	p0Rc
03654F64	16 00 00 00 61 00 71 00	u i l a q

Bien aquí tenemos algo como “aquiles07argamenon2008”, como estoy seguro que eso te manda al mensaje correcto entonces cerramos todo y probemos con dicho password y vemos como nos va:



Listo quedo resuelto y con esto concluimos este reto algo sencillo utilizando la teoría que les comente, lo cual ni con el themida resulto ser tan complejo, igual este tutorial no trataba de como desempacar la aplicación, esto seria todo, nos vemos en el próximo escrito, por cierto estoy trabajando en un análisis de malware que estoy seguro gustara mucho ya que es demasiado extenso, saludos

Alejandro Torres (torrescrack)

Twitter: <https://twitter.com/TorresCrack248>

e-mail: tora_248@hotmail.com