[QuickNote] Techniques for decrypting BazarLoader strings

kienmanowar.wordpress.com/2022/02/24/quicknote-techniques-for-decrypting-bazarloader-strings/

February 24, 2022

1. Overview

Usually, to make it more difficult for analysts, malware authors will hide important strings and only decrypt these strings during runtime. The famous malwares like <u>Emotet</u>, <u>QakBot</u> or <u>TrickBot</u> often use the one or some functions to perform decrypting strings when needed.

However, on researching and analyzing some other malwares such as **Conti**, **BlackMatter** and **BazarLoader**, instead of using a separate function to decrypt strings, these malwares make it more difficult by saving the encrypted strings on the stack as stack strings. Then, strings are decrypted by XOR-ing with a key value (this value may not be fixed) or through quite complex computation. This technique consumes time of the analyst.

The images below are the pseudocode of the Conti and BlackMatter malware.

text:00038C5 mov [cbp:s28ttrog.dl:], 3b; 1 ⁻² 0.25 text:00038C5 mov [cbp:s28ttrog.dl:], 2b; 1 ⁻¹ 0.5 text:00038C5 mov [cbp:s28ttrog.dl:], 0]; 0 ⁻¹ 0.5 text:00038C5 mov [cbp:s28t	text:004038C5	mov [et	p+szRstrtmgr.dll], 3	27 57	zRsti	rtmgr.o	dll[0] = 3;			
cwt:00033C0 mov [cbp:s2strtng:.dl:], d0; ; 0' 0; cwt:00033C0 mov [cbp:s2strtng:.dl:], d0; ; 0' 0; ssRtrng:.dl:] ssRtrng:.dl:] cd; cwt:0003200 mov [cbp:s2strtng:.dl:], d0; ; 0' 0; ssRtrng:.dl:] ssRtrng:.dl:] cd; ssRtrng:.dl:] cd; sst::000355 mov d	text:004038C9	mov [et	p+szRstrtmgr.dll+1], 33h ; '3'	28 57		rtmgr.	dll[1] = 0×33;			
<pre>text:00030D1 mov [cbp:sRhtrmgr.dlt:3].20; if i sRhtrmgr.dlt[3] = 0:26; cxt:00030D1 mov [cbp:sRhtrmgr.dlt:3].20; if i sRhtrmgr.dlt[3] = 0:26; cxt:00030D1 mov [cbp:sRhtrmgr.dlt:3].20; if i sRhtrmgr.dlt[3] = 0:26; sRhtrmgr.dlt[3] = 0:26; sRhtrmgr.dlt[0] = 0:27; sRhtrmgr.dlt[0] = 0:27;</pre>	text:004038CD	mov [et	p+szRstrtmgr.dll+2], 40h ; '@'	29 53		rtmgr.	dll[2] = 0×40;			
<pre>text:0003050 mov [cbp+sRstrtgr.dlt+3], db; ie' 338strtgr.dlt(3] = 0+dq; sinstrtgr.dlt(3] =</pre>	text:004038D1	mov [et	p+szRstrtmgr.dll+3], 26h ; '&'	30 si		rtmgr.	dll[3] = 0×26;			
<pre>text:00038D9 mov [cbp+s2strtngr.dlt0], d0; 'd' szktrtngr.dlt0], d0</pre>	text:004038D5	mov [et	p+szRstrtmgr.dll+4], 40h ; '@'	31 s:		rtmgr.	dll[4] = 0×40; Conti	(ransomware)		
cwt:000038D0 mov [cbp+s2xttrug.dlt], 2b, 1ch cwt:00038D5 mov [cbp+s2xttrug.dlt], 2b, 1ch cwt:00036D5 mov [cbp+s2xttrug.dlt], 2b, 1ch cwt:00036D5 mov [cbp+s2xttrug.dlt], 2b,	text:004038D9	mov [et	p+szRstrtmgr.dll+5], 64h ; 'd'	32 si		rtmgr.	dll[5] = 0×64;			
<pre>text:0003351 mvv [cbpt:sz8trtmpr.dll+2], 26h; '+' cett:0003355 mvv [cbpt:sz8trtmpr.dll+2], 26h; '+' cett:0003355 mvv [cbpt:sz8trtmpr.dll+3], 6h; 'n' cett:0003355 mvv [cbpt:sz8trtmpr.dll+3], 6h; 'n' cett:0003355 mvv [cbpt:sz8trtmpr.dll+6m], 57h; '\' cett:0003355 mvv [cbpt:sz8trtmpr.dll-6m], 57h; '\' cett:0003355 mvv [cbpt:sz8trtmpr.dll] = cott; cotting, cll[[0] = 0×36; ' cett:0003355 mvv [cbpt:sz8trtmpr.dll] = cotting, cll[[0] = 0×36; ' cett:0003355 mvv [cbpt:sz8trtmpr.dll] = cotting, cll[[0] = 0×36; ' cett:0003355 mvv [cbpt:sz8trtmpr.dll] = cotting, cll[[0] = 0×36; ' cett:0003356 jnz short loc.403393</pre>	text:004038DD	mov [et	p+szRstrtmgr.dll+6], 16h	33 52		rtmgr.	dll[6] = 0×16;			
<pre>text:00038E5 mov [cbpt=zzRstrmpr.dlt], 2b; ; +'' [zzRstrmpr.dlt], 2b; ; +'' [zzRstrmpr.dlt], 2b v2b; [zzRstrmpr.dlt]</pre>	text:004038E1	mov [et	p+szRstrtmgr.dll+7], 26h ; '&'	34 s:		rtmgr.	dll[7] = 0×26;			
<pre>text:0003859 mov [cbp+s28tttmgr.dll+0], 5f; 'W' text:0003851 mov [cbp+s28tttmgr.dll+0], 5f; 'W' text:0003851 mov [cbp+s28tttmgr.dll+0], 5f; 'W' text:0003859 mov al, [cbp+s28ttmgr.dll+0], 5f; 'W' text:0003859 mov al, [cbp+s28ttmgr.dll] text:0003869 mov devcd ptr [cax:1], 0 text:0003869 mov devcd ptr [cax:1], 726978h ext:00049666 mov devcd ptr [cax:2], 1726978h ext:00049666 mov devcd ptr [cax:2], 1726978h ext:0009666 mov devcd ptr [cax:1], 1726978h ext:0009666 mov ecx, 8 ext:0009666 mov ecx, 8 ext:0009666 mov devcd ptr [cax:1], 1726978h ext:0009666 mov ecx, 8 ext:000966 ex</pre>	text:004038E5	mov [eb	p+szRstrtmgr.dll+8], 2Bh ; '+'	35 si		rtmgr.	dll[8] = 0×2B;			
<pre>text:00038ED</pre>	text:004038E9	mov [et	p+szRstrtmgr.dll+9], 6Eh ; 'n'	36 s:		rtmgr.	dll[9] = 0×6E;			
toxt:0003811 mov [cbp+s2Rstrtmgr.dll+08], g7h; 'v' 0.33 y2Rstrtmgr.dll[08] = 0×50; toxt:0003859 mov al, [cbp+s2Rstrtmgr.dll] 0.44 y2Rstrtmgr.dll[08] = 0×50; 1.50 for (j = 0; j < 0×0; +j)	text:004038ED	mov [eb	p+szRstrtmgr.dll+0Ah], 57h ; 'W'	37 s:		rtmgr.	dll[0×A] = 0×57;			
<pre>text:00038F5 mov [ebp+s:Retrtagr.dll+0ch], 50h; 'P' [0] #Retrtagr.dll(] = 0; j < 0; 0; 0; j = 0; j < 0; 0; j = 0; j < 0; 0; j = 0; j < 0; 0; j = 0; j = 0; j < 0; 0; j = 0; j = 0; j < 0; 0; j = 0; j = 0; j < 0; 0; j = 0; j = 0; j < 0; 0; j = 0; j = 0; j < 0; 0; j = 0; j = 0; j < 0; 0; j = 0; j = 0; j < 0; 0; j = 0; j = 0; j < 0; 0; j = 0; j = 0; j < 0; 0; j = 0; j = 0; j < 0; 0; j = 0; j = 0; j < 0; 0; j = 0; j = 0; j < 0; 0; j = 0; j = 0; j < 0; 0; j = 0; j = 0; j < 0; 0; j = 0; j = 0; j < 0; 0; j = 0; j = 0; j < 0; 0; j = 0</pre>	text:004038F1	mov [et	p+szRstrtmgr.dll+0Bh], 57h ; 'W'	38 s:		rtmgr.	dll[0×B] = 0×57;			
<pre>text::00403959 mov al, [cbp+srxstrtmgr.dl]</pre>	text:004038F5	mov [et	p+szRstrtmgr.dll+0Ch], 50h ; 'P'	39 s:	zRsti	rtmgr.	dll[0×C] = 0×50;			
<pre>text:0040936C cmp [bp+var_16], 0 text:00409360 jnz short loc_409394 ext:00409360 jnz short loc_409394 ext:00409360 int loc_409394 ext:00409360 int loc_409363 loa eax, [ebp+wsz_TimesNewRomann] ext:00409669 mov dword ptr [eax], 1/2489FAA h_Rstrtmgr_dll = LoadLibraryA = (MROULE (</pre>	text:004038F9	mov al,	[ebp+szRstrtmgr.dll]	40 fo	or (j = Θ	; j < 0×D; ++j)			
<pre>text:00403900 jnz short loc_403934</pre>	text:004038FC	cmp [et	p+var_18], 0	• 41	szRs	strtmg	r.dll[j] = (0×27 * (0×50 - (unsignedi	<mark>nt8)</mark> szRstrtmgr.dl	l[j]) % 0×7F + 0×7F) % 0×7	7F;
<pre>text:00409063 lea eax, [ebp+msz_TimesHewRoman] ext:00409063 mov dword ptr [eax], 17089FACh ext:00409065 mov dword ptr [eax+1], 17049FBACh ext:0040906C mov dword ptr [eax+1], 1719FBAh ext:0040906C mov dword ptr [eax+10], 1709FPAh ext:0040906FB add eax, 4 ext:0040906FB add eax, 4 ext:0040906FB add eax, 4 ext:0040906FB add eax, 4 ext:00409076F jnz short loc_4096F5 ext:00409076F jnz short loc_4096F5 ext:00409076C mov ecx, eax ext:0040970C mov ecx, eax ext:0040970F mov ecx, eax ext:0040970F mov ecx, eax ext:0040970F mov ecx, [bop+vy] ext:0040970F mov ecx, eax ext:0040970F mov ecx, eax ext:0040970F mov ecx, [bop+vy] ext:0040970F mov ecx, eax ext:0040970F mov ecx, [bop+vy] ext:0040970F mov ecx, eax ex</pre>	text:00403900	jnz sho	ort loc_403934	● 42 L0	badLi	ibrary/	A = (HMODULE (stdcall *)(LPCSTR))f_dyna	amic_resolve_api_	funcs(0×F, func_kernel32_	.oadLibraryA,
ext:00409683 lea eax, [ebp+msz_TimesHewRoman] ext:00409663 mov dword ptr [eax], 17689FACh ext:00409665 mov dword ptr [eax+1], 17649F9Sh ext:00409660 mov dword ptr [eax+0], 17149F8Bh ext:00409660 mov dword ptr [eax+0], 17149F8Bh ext:00409660 mov dword ptr [eax+0], 17149F8Bh ext:00409660 mov dword ptr [eax+101], 1769F9Sh ext:00409662 mov dword ptr [eax+101], 1769F9Sh ext:00409662 mov dword ptr [eax+101], 1769F9Sh ext:00409665 mov dword ptr [eax+101], 1709F9Sh ext:00409666 mov ecx, 8 ext:004096675 mov ecx, 8 ext:004096675 xor dword ptr [eax], 17019FF8h ext:00409676 jnz short loc_409675 ext:00409676 yor short loc_409675	text:00403900			43 h.	Rstr	rtmgr_c	dll = LoadLibraryA(szRstrtmgr.dll);			
ext:00409683 lea exx, [bpt=msz.TimesNewRoman] ext:00409685 mov dword ptr [eax], 1769FACh ext:00409665 mov dword ptr [eax+1], 1749F85h ext:00409666 mov dword ptr [eax+1], 17219F88h ext:00409662 mov dword ptr [eax+10], 17219F88h ext:00409662 mov dword ptr [eax+10], 1769F95h ext:00409662 mov dword ptr [eax+10], 1769F95h ext:00409665 mov dword ptr [eax+10], 1769F95h ext:00409676 mov ext, 4 ext:00409676 add ext, 4 ext:00409676 add ext, 4 ext:00409676 jnz short loc_409675 ext:00409676 jnz short loc_409675 ext:00409676 jnz short loc_409675 ext:004099761 p										
ext:00409689 mov dword ptr [eax], 17699FACh 82 ext:00409685F mov dword ptr [eax+4], 17649F95h 83 ext:0040960C mov dword ptr [eax+4], 17649F86h 84 ext:0040960D mov dword ptr [eax+6], 17649F86h 85 ext:0040960D mov dword ptr [eax+10], 17649F86h 85 ext:0040960D mov dword ptr [eax+10], 17649F86h 85 ext:0040960D mov dword ptr [eax+10], 17649F86h 86 ext:0040960B mov dword ptr [eax+10], 17649F86h 85 ext:0040966B mov dword ptr [eax+10], 17649F86h 87 ext:0040966F mov dword ptr [eax+10], 17049F86h 87 ext:0040966F mov dword ptr [eax+10], 17049F86h 87 ext:004096F6 mov dword ptr [eax+10], 17049F86h 88 ext:004096F6 mov dword ptr [eax+10], 17049F86h 87 ext:004096F6 mov ecx, 8 92 do ext:004096F6 mov dword ptr [eax+10], 17019FF8h 92 do ext:004096F6F jzz short loc_4096F5	ext:004096B3	lea	eax. [ebp+wsz_TimesNewRoman]			81	}			
ext:0004096BF mov dword ptr [eax+0], 17219F8Bh 83 ext:0004096C5 mov dword ptr [eax+0C1], 17219F8Bh 84 ext:0004096C0 mov dword ptr [eax+0C1], 17219F8Bh 84 ext:0004096C0 mov dword ptr [eax+10], 17219F8Bh 86 ext:0004096C0 mov dword ptr [eax+10], 17219F8Bh 86 ext:0004096C2 mov dword ptr [eax+10], 17609F93h 86 ext:0004096C2 mov dword ptr [eax+10], 17609F93h 88 ext:0004096C6 mov dword ptr [eax+10], 17609F93h 88 ext:0004096C6 mov dword ptr [eax+10], 17609F93h 88 ext:0004096C6 mov dword ptr [eax+10], 17019F96h 89 ext:0004096C6 mov ecx, 8 90 ext:0004096C75 xor dword ptr [eax], 17019F78h 93 ext:0004096C75 xor dword ptr [eax], 17019F78h 94 ext:0004096C75 xor dword ptr [eax], 17019F78h 94 ext:0004096C75 xor dword ptr [eax], 17019F78h 95 ext:0004096C76 gat 95 97	ext:004096B9	mov	dword ptr [eax], 17689FACh			82	pwsz_TimesNewRoman = wsz_TimesNewF	Roman;		
ext:090096CG mov dword ptr [eax+0], 17219F88h 84 ext:090096CD mov dword ptr [eax+0], 17219F88h 85 ext:090096CD mov dword ptr [eax+0], 17219F88h 85 ext:090096CD mov dword ptr [eax+10], 17219F88h 85 ext:090096CD mov dword ptr [eax+10], 17219F88h 85 ext:090096CD mov dword ptr [eax+10], 17059FAAh 87 ext:090096CD mov dword ptr [eax+10], 17019F86h 88 ext:090096C5 mov dword ptr [eax+10], 17019F96h 88 ext:090096F6 mov ecx, 8 96 ext:090096F5 sort dword ptr [eax], 17019FF8h 92 ext:090096F5 sort dword ptr [eax], 17019FF8h 92 ext:090096F5 sort dword ptr [eax], 17019FF8h 92 ext:090096F5 sort dword ptr [eax], 17019FF8h 93 ext:090096F5 sort dword ptr [eax], 17019FF8h 94 ext:090096F6 dec ecx 95 ext:090096F6 gth; 'X' ; index 95 ext:090096F6 so	ext:004096BF	mov	dword ptr [eax+4], 17649F95h			83	wsz_TimesNewRoman[0] = 0×17689FAC			
ext:004096CD mov dword ptr [eax+0ch], 17640FB6h 0 ext:004096CD mov dword ptr [eax+10h], 17649FB6h 0 ext:004096CD mov dword ptr [eax+10h], 17649FB6h 0 ext:004096CD mov dword ptr [eax+10h], 17649FB6h 0 ext:004096CB mov ecx, 8 0 ext:004096CB xor dword ptr [eax+10h], 17019FFBh 0 ext:004096CB add eax, 4 0 ext:004096CFB add eax, 4 0 ext:00409705C	ext:004096C6	mov	dword ptr [eax+8], 17219F8Bh			84	wsz_TimesNewRoman[1] = 0×17649F95			
ext:000096001 mov dword ptr [eax+10h], 17219FBFh 86 ext:00009605 mov dword ptr [eax+10h], 17689FAAh 87 ext:00009605 mov dword ptr [eax+10h], 17689FAAh 87 ext:00009605 mov dword ptr [eax+10h], 17689FAAh 87 ext:00009605 mov dword ptr [eax+10h], 17609F95h 88 ext:00009605 mov dword ptr [eax+10h], 17019F96h 88 ext:00009606 mov dword ptr [eax+10h], 17019F96h 89 ext:000096676 mov ecx, 8 99 ext:000096676 ccx.00009675 cot 4000000000000000000000000000000000000	ext:004096CD	mov	dword ptr [eax+0Ch], 17649FB6			85	<pre>wsz_TimesNewRoman[2] = 0×17219F8B</pre>			
ext:0000960B mov dword ptr [eax+101], 17659FAAh 97 ext:00009662 mov dword ptr [eax+101], 17609F95h 98 ext:00009666 mov dword ptr [eax+101], 17019F96h 99 ext:00009666 mov ecx, 8 99 ext:000096676 mov dword ptr [eax+1, 1, 17019F78h 92 ext:000096676 good (ransommare) ext:000096676 good (ransommare) ext:00009676 good (for ext:00009676 good (good ext:00009676 good (good ext:00009676 good (good ext:000096767 good (good ext:00009676 good (good	ext:004096D4	mov	dword ptr [eax+10h], 17219F8FH			86	<pre>wsz_TimesNewRoman[3] = 0×17649FB6</pre>			
ext:0004096E2 mov dword ptr [eax+18h], 17609F95h 88 ext:0004096F3 mov dword ptr [eax+12h], 17019F96h 88 ext:0004096F3 mov ecx, 8 99 ext:0004096F6 mov ecx, 8 91 ext:0004096F6 ecx, 8 92 ext:0004096F5 cont = 1; cont = 1; cont = 2; cont = 4; 93 ext:0004096F5 add eax, 4 95 ext:0004096F6 dex, 4 95 ext:0004096F6 add eax, 4 95 ext:0004096F6 add eax, 4 95 ext:0004096F6 part 94 ext:0004096F6 add eax, 4 95 ext:0004096F6 part 94 ext:00040976F jnz short loc_4096F5 ext:000409701 push S8h; 'X' ; index 99 ext:00409703 push S8h; 'X' ; index 99 ext:00409706 call GetDeviceCaps 102 ext:00409706 mov ecx, eax 103 ext:00409706 mov ecx, eax 103	ext:004096DB	mov	dword ptr [eax+14h], 176E9FAAH			87	wsz_TimesNewRoman[4] = 0×17219F8F	Bla	ackMatter	
ext:000030559 mov dword ptr [eax+1ch], 17019F96h 0 ext:000030569 mov ecx, 8 0 ext:000030569 mov ecx, 8 0 ext:000030565 cont = 3; do (ext:000030565 xor dword ptr [eax], 17019FF8h 93 ext:000030565 xor dword ptr [eax], 17019FF8h 93 ext:000030565 xor dword ptr [eax], 17019FF8h 93 ext:000030565 add eax, 4 93 ext:000030567 jzz short loc_4096F5 95 ext:000030567 gzz jindex 95 ext:000030567 gzz jindex 96 ext:00003057 gzz jindex 96 ext:00003057 gzz jindex 97 ext:00003056 gzz jindex 96 ext:000030576 gzz jindex 96	ext:004096E2	mov	dword ptr [eax+18h], 17609F95h			88	wsz_TimesNewRoman[5] = 0×176E9FAA	(ra	nsomware)	
ext:004096F6 mov ecx, 8 ext:004096F6 ecx, 8 ext:004096F5 ecx, 4 ext:004096F5 icc, 4096F5 ext:004096F6 add ext:004096F6 icc, 4096F5 ext:00409761 push short loc, 4096F5 99 ext:00409761 push ext:00409762 push ext:00409763 push ext:00409766 icd ext:00409766 icd ext:00409766 icd ext:00409766 mov ext:00409766 mov ext:00409766 mov ext:00409766 mov ext:00409766 mov ext:00409766 mov </td <td>ext:004096E9</td> <td>mov</td> <td>dword ptr [eax+1Ch], 17019F96</td> <td></td> <td></td> <td>89</td> <td><pre>wsz_TimesNewRoman[6] = 0×17609F95</pre></td> <td></td> <td></td> <td></td>	ext:004096E9	mov	dword ptr [eax+1Ch], 17019F96			89	<pre>wsz_TimesNewRoman[6] = 0×17609F95</pre>			
ext:004096F6 91 cnt = 8; ext:004096F5 icc_4096F5: ; CODE XREF: sub ext:004096F5 xor dword ptr [eax], 17019FF8h ext:004096F5 add ext, 4 ext:004096F5 add ext, 4 ext:004096F5 dec ecx ext:004096F5 dec ecx ext:004096F5 dec ecx ext:004096F6 dec ecx ext:004096F6 dec ecx ext:004096F6 ext 99 ext:004096F6 ext 99 ext:00409701 push S8h; 'X' ; index 99 pixels = GetDeviceCaps(hdc, LOGPIXELSY); hFort = CreateFontW_0C inte(cnc '); ext:00409706 call GetDeviceCaps ext:00409706 inte(cnc '); inte(cnc '); ext:00409706 </td <td>ext:004096F0</td> <td>mov</td> <td></td> <td></td> <td></td> <td>90</td> <td><pre>wsz_TimesNewRoman[7] = 0×17019F96</pre></td> <td></td> <td></td> <td></td>	ext:004096F0	mov				90	<pre>wsz_TimesNewRoman[7] = 0×17019F96</pre>			
ext:004096F5 ; CODE XREF: sub 92 do ext:004096F5 ic ; CODE XREF: sub 93 ext:004096F5 xor dword ptr [eax], 17019FF8h 93 ext:004096F5 add eax, 4 95 ext:004096F5 dec ecx 96 ext:004096F5 add eax, 4 95 ext:004096F5 dec ecx 96 ext:004096F5 jnz short loc_4096F5 97 ext:004096F6 jnz short loc_4096F5 97 ext:004096F6 jnz short loc_4096F5 97 ext:004096F6 jnz short loc_4096F5 97 ext:00409701 push Sh, 'X' ; index 99 ext:00409703 push [ebp+hdc] ; hdc 108 ext:00409766 rate 102 0 ext:00409766 mov ecx, eax 103 0 ext:00409765 mov ecx, eax 103 0 ext:00409766 mov ecx, eax 103 0 ext:00409706 mov ecx, eax	ext:004096F0					91	cnt = 8;			
ext:004096F5 loc_4096F5: ; CODE XREF: sub 93 { ext:004096F5 loc_4096F5 word ptr [eax], 17019FF8h 94 *pms2_TimesNewRoman ~= 0×17019FF8u; // Times New Roman ext:004096F5 add eax, 4 95 ext:004096F5 dec ecx 96 ext:004096F5 dec ecx 96 ext:004096F5 gec 96 ext:004096F6 dec ecx ext:004096F7 97 ext:00409701 push 58h; 'X' ext:00409702 push 58h; 'X' ext:00409703 push 58h; 'X' ext:00409706 call GetDeviceCaps ext:00409706 call GetDeviceCaps ext:00409706 mov ecx, eax ext:00409706 mov ecx, eax ext:00409706 mov ecx, eax ext:004097070 mov ecx, eax ext:00409706 mov ecx, eax ext:00409706 mov ecx, eax ext:00409706 mov ecx, eax ext:00409706 mov ecx, eax	ext:004096F5					92	do			
ext:004096F5 xor dword ptr [eax], 17019FF8h 94 +pmsz_TimesNewRoman ~= 0×17019FF8u; // Times New Roman ext:004096F5 add eax, 4 95 +tmsz_TimesNewRoman; cnt; ext:004096FF jnz short loc_4096F5 96 cnt; } ext:004096FF jnz short loc_4096F5 97 > ext:004096FF jnz short loc_4096F5 96 ext:004096FF jnz Shi; 'X' ; index 99 ext:00409706 call GebeviceCaps(hdc, LOCPIXELSV); ext:00409706 call GebeviceCaps 101 ext:00409706 mov ecx, eax 103 0, ext:00409707 mov ecx, eax 103 0, ext:00409706 mov eax, [ebp+vy] 104 9, ext:00409707 mov eax, [ebp+vy] 104 9,	ext:004096F5	loc_4096F5:	CODE	XREF: sub		93	{			
ext:0004096FB add eax, 4'' 95 ++pmsz_TimesNemReman; ext:004096FF dec ecx 96 cnt; ext:004096FF jnz short loc_4096F5 97 ext:0040970FF push S8h; 'X' ; index 99 ext:00409703 push S8h; 'X' ; index 99 ext:00409703 push [ebp+thc] ; hdc 100 ext:00409706 call GetDeviceCaps 101 ext:00409706 mov ecx, eax 102 ext:00409705 mov eax, [ebp+ty] 102 ext:00409705 mov eax, [ebp+ty] 104 ext:00409706 mov eax, [ebp+ty] 104	ext:004096F5	xor	dword ptr [eax], 17019FF8h			94	<pre>*pwsz_TimesNewRoman /= 0×17019FF</pre>	F8u;	// Times New Roman	
ext:00H096FF dec ecx ext:00H096FF jz short loc_4096F5 ext:00H096F short loc_4096F5 96 ext:00H09701 push 58h; 'X' ; index 99 ext:00H09703 push 58h; 'X' ; index 99 ext:00H09706 call GebDeviceCaps 101 ext:00H09706 call GetDeviceCaps 101 ext:00H09706 mov ecx, eax 103 0, ext:00H09706 mov ecx, [ebp+cy] 104 0, ext:00H09706 mov ecx, edx 105 FM BOLD.	ext:004096FB	add	eax, 4			95	<pre>++ pwsz_TimesNewRoman;</pre>			
ext:0004096FF jnz short loc_4096F5 97) ext:000409701 push 58h; 'X' ; index 99 nPixels = GetDeviceCaps(hdc, LOGPIXELSX); ext:000409703 push [ebp+hdc] ; hdc 100 hFont = CreateFontW_0(ext:000409706 call GetDeviceCaps 101 7 * (cy / (int64)nPixels), ext:000409706 mov eax, [ebp+ty] 102 0, ext:000409706 mov eax, [ebp+ty] 104 0, ext:004009706 mov eax, [ebp+ty] 105 FW ROLD.	ext:004096FE	dec				96	cnt;			
ext:0040936FF 98 mnile (Cnt ;) ext:004093701 push S8h; 'X' ; index 99 nPixels = GetDeviceCaps(hdc, LOGPIXELSY); ext:004093703 push [ebp+thdc] ; hdc 100 hFont = CreateFontW_0(ext:004093706 call GetDeviceCaps 101 7 * (cy / (_int64)nPixels), ext:004093706 mov ecx, eax 103 0, ext:004093706 mov ecx, [ebp+cy] 103 0, ext:004093706 mov ecx, [ebp+cy] 104 0, ext:004093706 mov ecx, [edx edx 105 FW BOLD.	ext:004096FF	jnz	short loc_4096F5			97	}			
ext:00409701 push SBh; 'X' ; index 99 nPixels = GetbeviceCaps(hdc, LOGPIXELSY); ext:00409703 push [ebp+hdc] ; hdc 000 hFont = CreateFontW_0(ext:00409706 call GetbeviceCaps 101 7 * (cy / _int64)nPixels), ext:00409706 mov ecx, eax 103 0, ext:00409705 mov eax, [ebp+vy] 104 0, ext:00409706 mov eax, [ebp+vy] 105 FW BOLD.	ext:004096FF					98	while (cht);			
ext:00409703 push [ebp+hdc] ; hdc ● 100 hFont = CreateFontW_0(ext:00409706 call GetDeviceCaps 101 7 * (cy / (int64)nPixels), ext:00409706 mov ecx. eax 102 0, ext:00409706 mov ecx. (eax 103 0, ext:00409706 mov ecx. (eax 103 0, ext:00409706 mov eax, [ebp+cy] 104 0, ext:00409706 mov edx. edx 105 FW ROLD.	ext:00409701	pust	n 58h;'X' ; inde	x		99	nPixels = GetDeviceCaps(hdc, LOGP	IXELSX);		
ext:00H09706 call GetDeviceCaps 101 7 * (cy / (int64))nPixels), ext:00H09706 mov ext. 02 0, ext:00H09706 mov eax, [ebp+cy] 103 0, ext:00H09706 mov eax, [ebp+cy] 104 0, ext:00H09706 mov eax, [ebp+cy] 105 FW ROLD.	ext:00409703	pust	n [ebp+hdc] ; hdc			100	hFont = CreateFontW_0(
ext:00409706 102 0, ext:0040970C mov ecx, eax 103 ext:0040970E mov eax, [ebp+cy] 104 ext:00409711 xor edx edx	ext:00409706	call	GetDeviceCaps			101	7 * (cv / (int64)nPixe	els),		
ext:0040970C mov ecx, eax 103 0, ext:0040970E mov eax, [Ebp+cy] 104 0, ext:0040970E mov eax, edx, edx 105 FW ROLD.	ext:00409706					102	0,			
ext:0040970E mov eax, [ebp+cy] 104 0, ext:00409711 xor edx. edx 115 EW BOLD.	ext:0040970C	mov	ecx. eax			103	θ.			
ext:00409711 xor edx edx 105 FW BOLD	ext:0040970E	mov	eax, [ebp+cv]			104	Θ.			
	ext:00409711	xor	edx, edx			105				

This article uses the BazarLoader samples as an example to demonstrate how to decrypt strings with:

- Automate resolving with IDAPython script.
- Emulate code with IDA uEmu plugin.
- Debugging with IDA Bochs plugin.

2. BazarLoader samples

BazarLoader was first discovered in April 2020. The malware loader has been continuously evolving, allowing attackers to install additional malware, often used for ransomware attacks, dropping Cobalt Strike, and stealing sensitive data. The common assumption is that the distribution and post-exploitation activities of the loader are akin to the Trickbot malware.

These samples are all 64-bit Windows executable.

- Unpacked sample 1: <u>cc522400b3fed1d2c4dcca16666ddcff</u>
- Unpacked sample 2: <u>63c4bb3f1044f36632ce1759b62296dc</u>

3. Decrypt strings

3.1. Using IDApython script

Analyzing the first sample of BazarLoader, we will see that it uses the same stack strings decryption technique as in BlackMatter ransomware:



To decrypt these strings, you can use **<u>x64dbg</u>** to debug or extract the above values and use **<u>CyberChef</u>** to perform the following:

Recipe		2 🖬 🗊	Input
Swap endianness		⊘ 11	0x3B8D8A75 0x67CC837B
Data format Hex	Word length (bytes) 4	Pad incomplete words	0x39938B30 0x55FFEF1E
From Hex		0 11	
Delimiter			Output
Auto			kernel32.dll
XOR		⊗ II	
^{Key} Øx1eefff55		HEX 🕶	
^{Scheme} Standard		Null preserving	

However, debugging with **x64dbg** or using **CyberChef** as above will take more time, to make static analysis easier, I will use **IDAPython** script to decrypt the strings. The code I use is as follows:

```
import idc, idaapi, struct
    def bazarloader_decrypt_str(start_ea, end_ea, xor_key_ea):
         enc_buf = []
        xor_key = []
         xor_key_val = struct.pack("<I", idc.get_operand_value(xor_key_ea, 1))</pre>
         for i in bytearray(xor_key_val): xor_key.append(i)
         if start_ea is None or end_ea is None or xor_key_ea is None:
             print ('Not enough information to decrypt')
             return
         curr = start_ea
         while curr <= end_ea:</pre>
             enc_val = struct.pack("<I", idc.get_operand_value(curr, 1))</pre>
             for i in bytearray(enc_val): enc_buf.append(i)
             curr = idc.next_head(curr)
         for i in range(len(enc_buf)):
             enc_buf[i] ^= (xor_key[i % len(xor_key)]) & 0xFF
         return ''.join([chr(c) for c in enc_buf])
27
```

Load this script into IDA, providing the relevant addresses to perform the decryption:

0000018000A622	lea	esi, [rdi+1]	In [2]:
0000018000A62E	mov	edx, 55FFEF1Eh	xor_key addr
0000018000A630	mov	<pre>cs:byte_18001D330, sil</pre>	in [2]:
0000018000A637	mov	[rbp+57h+var 40], dil	start addr
0000018000A63B	mov	<pre>[rbp+57h+var_3C], 3B8D8A75h</pre>	in [2]:
0000018000A642	mov	[rbp+57h+var_3C+4], 67CC837Bh	
0000018000A649	mov	[rbp+57h+var_3C+8], 39938B30h	10 131: Driht(bazarloader_decrypt_str(b×0000000180004638, 0×0000000180004649, 0×0000000180004628))
0000018000A650	litov	<pre>[rbp+57h+var_3C+0Ch], edx</pre>	kernet32.dtt
0000018000A653	mov	eax, [rbp+57h+var_3C]	
9000018000A656	mov	al, [rbp+57h+var_40]	in L4 end addr
0000018000A659	test	al, al	
•• 0000018000A65B	jnz	short loc_18000A672	decrypted string

Finally, by using the above script, the analysis process will be much more convenient:



3.2. Using uEmu plugin

In the second sample of BazarLoader, the code that decrypt the stack strings is similar to the Conti ransomware and quite complicated:

•	:000000020414E986	sub	rsp. 68h	7	char v13; // [rsp+5Bh] [rbp-2Dh]	
•	:000000020414E98A	mov	rax, 1001443E45754F03h	8		
•	:000000020414E994	mov	<pre>[rsp+88h+enc_buf], rax</pre>	9	*enc_buf = 0×1D01443E45754F03i64;	
•	:000000020414E999	mov	r10, rcx		$enc_buf[0\times 12] = 0\times 6F;$	
•	:000000020414E99C	lea	rdi, [rsp+88h+dec_buf]	11	v13 = 0;	
•	:000000020414E9A1	mov	r13, r8	0 12	*&enc_buf[8] = 0×6C4F1D4433230E4Fi64;	
•	:000000020414E9A4	mov	[rsp+88h+var_41], 6Fh ; 'o'	0 13	*&enc_buf[0×10] = 0×3422;	
•	:000000020414E9A9	lea	<pre>rsi, [rsp+88h+enc_buf]</pre>	• 14	<pre>qmemcpy(dec_buf, enc_buf, sizeof(dec_buf));</pre>	
•	:000000020414E9AE	mov	r8, r9	0 15	for (i = 0164; i \neq 19; ++i)	
•	:000000020414E9B1	mov	r12d, edx	16	{	
•	:000000020414E9B4	mov	ecx, 13h	• 17	<pre>dec_buf[i] = ((0×FFFFFFE9 * (dec_buf[i] - 0×6F)) % 0×7F + 0×7F) % 0</pre>	×7F;
•	:000000020414E9B9	mov	[rsp+88h+var_2D], 0	18	}	
•	:000000020414E9BE	mov	rax, 6C4F1D4433230E4Fh	0 19	v1 = sub_20414AB1A(a1, dec_buf);	
•	:000000020414E9C8	mov	r9d, 7Fh	0 20	<pre>return (v1)(dwFlags, lpModuleName, phModule);</pre>	
•	:000000020414E9CE	mov	[rsp+88h+var_4B], rax	0 21		
•	:000000020414E9D3	mov	[rsp+88h+var_43], 3422h			
•	:000000020414E9DA	rep m	ovsb			
τ:	0000002021414F2DA				8	
• t:0	00000020414F41C	sub	rsp, 58h		9 <u>і = мікш</u> :	
• t:0	00000020414F420	xor	r8d, r8d		<pre>10 *enc_buf = 0×6B70702306676B73i64;</pre>	
t:0	00000020414F423	mov			<pre>11 *&enc_buf[8] = 0×2C6B743E700B6703i64;</pre>	
t:0	00000020414F429	mov	rax, 6B70702306676B73h		<pre> 12 *dec_buf = *enc_buf; </pre>	
t:e	00000020414F433	mov	<pre>qword ptr [rsp+58h+enc_buf], rax</pre>		v6 = 0;	
t:0	00000020414F438	mov	rax, 2C6B743E700B6703h		14 *&dec buf[0×10] = 0×752C:	
t:0	00000020414F442	mov	<pre>qword ptr [rsp+58h+enc_buf+8], ra</pre>	ax 🛛	15 <mark>do</mark>	
• t:0	00000020414F447	movups	<pre>xmm0, xmmword ptr [rsp+58h+enc_bu</pre>	lt]	16 {	
• t:0	00000020414F44C	mov	<pre>word ptr [rsp+58h+enc_buf+10h], 7</pre>	752Ch	<pre> • 17 dec_buf[i] = (0×1C * (dec_buf[i] - 0×75) % 0×7F + 0×7F) % 0× </pre>	:7F;
t:0	00000020414F453	movups	<pre>xmmword ptr [rsp+58h+dec_buf], xm</pre>	ım⊖	● 18 ++i;	
t:0	00000020414F458	mov	<pre>[rsp+58h+var_E], 0</pre>		19 }	
• t:0	00000020414F45D	mov	<pre>word ptr [rsp+58h+dec_buf+10h], 7</pre>	752Ch	• 20 while ($i \neq 18$);	
t:0	00000020414F45D				<pre> 21 v2 = sub_20414AB1A(a1, dec_buf);</pre>	
t:0	00000020414F464				● 22 return v2();	2
t:0	00000020414F464 loc_20	9414F464:	; CODE XF	REF: sub_2	2 23}	

With the code as shown in the figure, the implementation of using IDApython script will be difficult and not feasible. The most suitable solution for this case is to use an emulator to emulate the code. Here, I will use <u>uEmu</u>, a tiny cute emulator plugin for IDA based on unicorn engine.

Very easy to emulate the decoding code with uEmu:

First, set a breakpoint at the address after the string has been decrypted.

	.text:000000020414E9DC .text:000000020414E9E1 .text:000000020414E9E4 .text:000000020414E9E7 .text:000000020414E9E8 .text:000000020414E9E8 .text:000000020414E9EE .text:000000020414E9EF .text:000000020414E9F2 .text:000000020414E9F9 .text:000000020414E9F9 .text:000000020414E9F9	movsx sub imul cdq idiv lea cdq idiv mov inc cmp jnz	<pre>eax, [rsp+rcx+88h+dec_buf] eax, 6Fh ; 'o' eax, -23 r9d eax, [rdx+7Fh] r9d [rsp+rcx+88h+dec_buf], dl rcx rcx, 13h short loc_20414E9DC</pre>
	text:0000000000000000		
	.text:000000020414E9FF	lea	rdx, [rsp+88h+dec_buf]
•	.text:000000020414EA04	mov	rcx, r10
•	.text:000000020414EA07	mov	[rsp+88h+var_60], r8

Go to the beginning of the function and select the starting address of the function, then start uEmu. The **CPU Context Edit** window will appear, click **OK** to continue. uEmu will now initialize the emulator. Check the CPU context to see if the address of the **EIP/RIP** register is pointed at the beginning of the function:

.text:000000020414E9	980 var_4B = qword ptr -4Bh	👔 uEmu CPU Context Edit	– 🗆 X
.text:00000020414E9	980 var_43 = word ptr -43h	Desister Velue	Sn,
.text:00000020414E9	080 var_41 = byte ptr -41h	Register Value	
.text:00000020414E9	080 dec_buf = byte ptr -40h	rax 0x0	3E4
.text:00000020414E9	980 var_2D = byte ptr -2Dh	rbx 0x0	1
.text:00000020414E9	980 	rcx 0x0	
.text:00000020414E9	980 push r13	rdx UxU	IF:
.text:00000020414E9	982 push r12	rsi UxU	(3)
.text:00000020414E9	984 push rdi	rai UXU	
.text:00000020414E9	985 push rsi	rop UxU	
.text:00000020414E9	986 sub rsp, 68h	-9 0x0	
.text:00000020414E9	98A mov rax, 1D01443E4575	-0 0-0	FI
.text:00000020414E9	994 mov [rsp+88h+enc_buf]	r10 0v0	
.text:00000020414E9	999 mov r10, rcx	r11 0v0	1
.text:00000020414E9	99C lea rdi, [rsp+88h+dec	r12 0v0	
.text:00000020414E9	A1 mov r13, r8	r13 0x0	
.text:00000020414E9	A4 mov [rsp+88h+var_41],	r14 0x0	
.text:00000020414E9	0A9 lea rsi, [rsp+88h+enc	r15 0x0	
.text:00000020414E9	PAE mov r8, r9	rip 0x0	
.text:00000020414E9	9B1 mov r12d, edx	sp 0x0)41
.text:00000020414E9	9B4 mov ecx, 13h		
.text:00000020414E9	BB9 mov [rsp+88h+var 2D].		
.text:00000020414E9	BE mov rax, 6C4F1D443323		
.text:00000020414E9	9C8 mov r9d, 7Fh	OK Cancel Search	h Help
.text:00000020414E9	PCE mov [rsp+88h+var_4B].	Line 1 of 18	
/iew-A		< 1 Pseudocode-A	uEmu CPU Context
.text::00000000004140080 enc buf = aword (ptr -53h	CPU context at [0×20414E	980: push r13]
.text::000000000004140980 var 4B = gword	ptr -4Bh		
.text::00000000000414E980 var_43 = word p	tr -43h	rax: 0×0000000000000000	r9: 0×00000000000000
.text:000000000004140980 var_41 = byte p	tr -41h	rbx: 0×000000000000000	r10: 0×000000000000000
.text::0000000000004140980 dec_buf = byte p	tr -40h	rcx: 0×0000000000000000	r11: 0×00000000000000
.text::000000000004140980 var_2D = byte p	tr -2Dh	rdx: 0×0000000000000000	r12: 0×00000000000000
1 ext: 08080800000105988		rsi: 0×0000000000000000	r13: 0×00000000000000
.text:00000020414E980 push	r13	rdi: 0×00000000000000000	r14: 0×000000000000000
• .text:000000020414E982 push	r12	rbp: 0×0000000000000000	r15: 0×00000000000000
.text:00000020414E984 push		rsp: 0×0000000000000000	rip: 0×00000020414E980
<pre>.text:00000020414E985 push</pre>		r8: 0×0000000000000000	sp: 0×0000000000000
• .text:00000020414E986 sub	rsp, 68h		
• .text:00000020414E98A mov	rax, 1D01443E45754F03h		
<pre>.text:000000020414E994 mov</pre>	<pre>[rsp+88h+enc_buf], rax</pre>		
.text:00000020414E9999 mov	r10, rcx		
.text:00000020414E99C lea	rdi, [rsp+88h+dec_buf]		
.text:00000020414E9A1 mov	r13, r8		
.text:00000020414E9A4 mov	[rsp+88h+var_41], 6Fh ; 'o'		2
.text:000000020414E9A9 lea	rsi, [rsp+88h+enc_buf]		
.text:00000020414E9AE mov	r8, r9		
.text:00000020414E9B1 mov	r12d, edx		
.text:000000020414E9B4 mov	ecx, 13h		
.text:000000020414E9B9 mov	[rsp+88h+var_2D], 0		
.text:000000020414E9BE mov	rax, 6C4F1D4433230E4Fh		
.text:000000020414E9C8 mov	r9d, 7Fh	0	
.text:000000020414E9CE mov	[rsp+88h+var_4B], rax		
.text:00000020414E9D3 mov	[rsp+88h+var_43], 3422h	La uEmu Control	
.text:00000020414E9DA rep movs	b		
.text:00000020414E9DA			
.text:000000020414E9DC			Start Run Sten Ston
			Start Ran Step Stop

Then, through **uEmu Control**, you can trace the code by **Step** or **Run** to emulates instructions until breakpoint is reached. During execution, uEmu will ask about unmapped memory, select **No** to continue.

View-	A			□ @ ×		Pseudocode-A	×	uEmu CPU Context, Output window	×	
	.text:000000020414E9DC	loc_20414E9DC:		; CODE XREF: s^	[uEmu Ci	PU Context				
•	.text:000000020414E9DC	movsx	eax, [rsp+rcx+88h+dec_b	uf]	CPU c	ontext at [0×20414E	9FF: lea rdx.	[rsp+88h+dec_buf]]		
•	.text:000000020414E9E1	sub	eax, <mark>6Fh</mark> ; 'o'							
•	.text:000000020414E9E4	imul	eax, -23		rax:		r9: 0×0000	00000000007F		
•	.text:000000020414E9E7	cdq			rbx:	0×00000000000000000	r10: 0×0000	000000000000		
•	.text:000000020414E9E8	idiv	r9d		rcx:		r11: 0×0000	000000000000		
•	.text:000000020414E9EB	lea	eax, [rdx+7Fh]		rdx:	0×0000000000000000	r12: 0×0000	00000000000		
•	.text:000000020414E9EE	cdq			rsi:		r13: 0×0000	00000000000		
•	.text:000000020414E9EF	idiv	r9d		rdi:		r14: 0×0000	00000000000		
	.text:000000020414E9F2	mov	<pre>[rsp+rcx+88h+dec_buf],</pre>	dl	rbp:	0×0000000000000000	r15: 0×0000	00000000000		
•	.text:000000020414E9F6	inc			rsp:		rip: 0×0000			
•	.text:000000020414E9F9	cmp	rcx, 13h		r8:	0×0000000000000000	sp: 0×0000		3	
u 2	.text:000000020414E9FD	jnz	short loc_20414E9DC							
	.text:000000020414E9FD									
	.text:000000020414E9FF	lea	rdx, [rsp+88h+dec_buf]							
	.text:000000020414EA04	mov	rcx, r10							
•	.text:000000020414EA07	mov	[rsp+88h+var_60], r8		9					
•	.text:000000020414EA0C	call	sub_20414AB1A		<					
	.text:000000020414EA0C				Output	window				
	.text:000000020414EA11	mov	r8, [rsp+88h+var_60]		[uEmu]	: cpy [204168000:20	04168FFF]			
	.text:000000020414EA16	mov	rdx, r13		[uEmu]	: Emulation started				
	.text:000000020414EA19	mov	ecx, r12d		Flushi	ng butters, please w	altok			
	.text:000000020414EA1C	add	rsp, 68h		[uEmu]	: ! <m> Missing memo</m>	ry at Øxttttt	TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT	s, data value = 0x0	
	.text:000000020414EA20	рор			[ucmu]	: map [FFFFFFFFFFFFF	ot 0x20414E0	FFFFFFFFFJ -> [FFFFFFFFFFF	- hufl	1
	.text:000000020414EA21	рор	rdi		[uumu]	. breakpoint reached	at 0x20414L9	rr . iea nux, [nsp+880+ue		_
	.text:00000020414EA22	рор	r12		IDC					
	.text:000000020414EA24	pop	r13		[uEmu Ci	ontrol				
	.text:00000020414EA26	jmp	rax							
	.text:00000020414EA26									
	.text:00000020414EA26	sub_20414E980 @	indp							
	.text:000000020414EA26									
	. text. 000000020414EA26									
	. text. 000000020414EA29	acgn_20414EA29		; DATA XREF: .						
	text.000000020414EA29	align 2								
	text:000000020414EA2A						S	Start Run Sten	Stop	
	. text:000000020414EA2A	,	== SUBROUTINE =					bup		

Press **Step**, to trace over the **lea** command. Then using uEmu's **Show Memory Range** feature, enter the address of the **rdx** register and select **Add**. The result will be similar to the following:

View-A					□ ₽ ×		Pseudocode-A	×	uEmu CPU Context, uEmu Memo	ry [], Output window 🛛
.text:00000020	414E9DC Loc	20414E9DC:		: CODE	XREF: s^	E uEmu CPU Context				
.text:000000020	414E9DC	movsx ea	x. [rsp+rcx+88h+	dec bufl			at [0x20/11		ncy n10]	
.text:00000020	414E9E1	sub ea	x, 6Fh ; 'o'			CFO CONCEXC	at [0/2041	46404. 100	ICX, 110 J	
.text:00000020	414E9E4	imul ea	ix, −23			ray: AxAAAAA	000000000000000000000000000000000000000	r9 • A X	99999999999999975	
.text:000000020	414E9E7	cda				rby: 0x00000	000000000000000000000000000000000000000	r10: 0x	000000000000007	
.text:000000020	414E9E8	idi 🕱 Show	Memory Range		×	rcx: 0x00000	000000000000000000000000000000000000000	r11: 0x	000000000000000000000000000000000000000	
.text:000000020	414E9EB	lea				rdy: Bresses		r12: 0x	000000000000000000000000000000000000000	
.text:000000020	414E9EE	cda specity st	art address and size of new me	emory range,		rsi: Oxeses	FFFFFFFFFFCO	r13: 0x	00000000000000000	4
.text:00000020	414E9EF	idi Address:	0xFFFFFFFFFFFFC0	Size: 16	~	rdi: OXEEEE	EEEEEEEEE	r1/1: 0x	000000000000000000000000000000000000000	
.text:00000020	414E9F2	MOV Comments			~	rbp: 0x00000		r15: 0x	000000000000000000000000000000000000000	
.text:00000020	414E9F6	inc				rsp: 0x55555	CCCCCCCCCC78	rip: Ox	00000000000000000	
.text:00000020	414E9F9	CMD	Add	Cancel		13p: 0x00000		sp: 0x	0000000204142804	
.text:000000020	414E9FD	inz sr	ort Loc_20414E9D	C		10. 0.00000		sþ. 0^	000000000000000000000000000000000000000	
.text:000000020	414E9FD					5				
.text:00000020	414E9FF	lea ro	x. [rsp+88h+dec	bufl						
.text:00000020	414EA04	mov ro	x. r10			Le utmu Memory []				
A View-A				🗆 🗗 × 🔤		Pseudocode-A	×	uEmu CPU Conte	xt, uEmu Memory [], Output windo	w 🗵
.text:000000020414E9DC loc	_20414E9DC:		; COD	E XREF: s^ [uEmu CPU Con	text				
.text:00000020414E9DC	movsx	eax, [rsp+rcx	+88h+dec_buf]		CPU conte	xt at [0x2041	4FA04: mov 1	rcx, r10]		
.text:000000020414E9E1	sub	eax, 6Fh ; 'o								
.text:000000020414E9E4	imul	eax, -23			rax: 0x00	000000000000000000000000000000000000000	r9: 0X(0007F	
.text:000000020414E9E7	cdq				rbx: 0x00	000000000000000000000000000000000000000	r10: 0X	000000000000000000000000000000000000000	00000	
.text:000000020414E9E8	idiv	r9d			rcx: 0x00	0000000000000013	r11: 0X	000000000000000000000000000000000000000	00000	
.text:000000020414E9EB	lea	eax, [rdx+7Fh]		rdy: axE	FEFFFFFFFFFFFFF	r12 0x(99999	
.text:000000020414E9EE	cdq				rsi: OxE	FEELEFEEEECO	r13 0x		99999	
.text:000000020414E9EF	idiv	r9d			rdi: 0XE	FEEL FEEFEFED3	r14: 0x		00000	
.text:000000020414E9F2	mov	[rsp+rcx+88h+	dec_buf], dl		rbn: 0x0	0000 0000000000000000000000000000000000	r15: 0x0	000000000000000000000000000000000000000	00000	
.text:000000020414E9F6	inc	rcx			rsp: 0xE	FFF FFFFFFFFFF78	rin: ex	0000002041	4FA04	
.text:000000020414E9F9	cmp	rcx, 13h			r8: 0×0	0000 0000000000000000000000000000000000	sp: 0x(00000000000	0FF78	
.text:000000020414E9FD	jnz	short loc_204	14E9DC							
.text:000000020414E9FD				5	5					
<pre>.text:000000020414E9FF</pre>	lea	rdx, [rsp+88h	+dec_buf]			ra 📕				
• .text:00000020414EA04	mov	rcx, r10		42	J domu memory			1 1 6 3 3		
.text:000000020414EA07	mov	[rsp+88h+var	60], r8		Memory at	L 0×FFFFFFFFFF	FFFFFC0: 32	byte(s)]		
.text:000000020414EA0C	call	sub_20414AB1A								
.text:000000020414EA0C						0 1 2	3 4 5 6	789A	BCDEF	
.text:00000020414EA11	mov	r8, [rsp+88h+	var_60]	F	FFFFFFFF	FFFFC0: 47 65	74 4D 6F 64	75 6C 65 4	8 61 6E 64 6C 65 45	GetModuleHandleE
.text:00000020414EA16	mov	rdx, r13		F	FFFFFFFF	FFFFD0: 78 57	00 00 00 00	00 00 00 0	0 00 00 00 00 00 00	×W
.text:00000020414EA19	mov	ecx, r12d						deen	unted string	
.text:00000020414EA1C	add	rsp, 68h						ueci	ypeed sering	
.text:000000020414EA20	рор	rsi			0					
<pre>.text:00000020414EA16 .text:000000020414EA19 .text:000000020414EA12 .text:000000020414EA1C .text:000000020414EA20</pre>	mov mov add pop	rdx, r13 ecx, r12d rsp, 68h rsi		F		FFFFD0: 78 57	00 00 00 00	decr	0 00 00 00 00 00 00 00 ypted string	×W



IDA Bochs debugger plugin allows malware researchers to debug malicious code in a safe/emulated environment. For more information please visit [<u>1</u>], [<u>2</u>].

In order to debug the code that decrypt the string, we configure the Bochs plugin to work in **IDB mode**. This mode is used to debug code snippets by simply selecting the code from the database.

Bochs configuration ×										
BOCHS <u>R</u> C Operation mode ○ Disk image ○ IDB ○ PE										
 Delete image files upon session end Use virtual breakpoints when protected mode is enabled 64-bit emulation 										
Startup stack size (in KB) 64 ~ Maximum memory (in KB) 7168 ~ OK Cancel										

Next, select the position or code snippets to debug, then press **F9** to start debugging:

.text:000000020414BB84		·	1int64fastcall sub_20414BB84(int64 a1,int64 a2,int64 a3, unsigned i
.text:00000020414BB84			2 {
.text:00000020414BB84	sub_20414BB84 proc near	; CODE XREF:	3int64 i; // rcx
.text:00000020414BB84		; StopWorkei	<pre>4int64 (fastcall *v8)(int64,int64, _QWORD); // rax</pre>
.text:00000020414BB84			5 _BYTE enc_buf[19]; // [rsp+35h] [rbp-53h]
.text:00000020414BB84	var_5C = dword ptr -5Ch		6
.text:00000020414BB84	enc_buf = qword ptr -53h		7 char v13; // [rsp+5Bh] [rbp-2Dh]
.text:00000020414BB84	var_4B = qword ptr -4Bh		8
.text:00000020414BB84	var_43 = word ptr -43h		9 *enc_buf = 0×162A674742444D5Fi64;
.text:000000020414BB84	var_41 = byte ptr -41h		10 enc_buf[0×12] = 0×24;
.text:00000020414BB84	dec_buf = byte ptr -40h		● 11 v13 = 0;
.text:00000020414BB84	<pre>var_2D = byte ptr -2Dh</pre>		12 *&enc_buf[8] = 0×7B36284D1664794Di64;
.text:000000020414BB84			13 *&enc_buf[0×10] = 0×3C4D;
• .text:000000020414BB84 000	push r13		14 qmemcpy(dec_buf, enc_buf, sizeof(dec_buf));
.text.000000020414BB88 008	push 112		• 15 for (i = 0i64; i \neq 0×13; ++i)
.text:000000020414BB88 010	push rdi		16 {
.text:000000020414BB89 018	push rsi		17 dec_buf[i] = ((0×FFFFFD4 * (dec_buf[i] - 0×24)) % 0×7F + 0×7F) % 0×7F;
• .text:000000020414BB8A 020	sub rsp, 68h		18 }
.text:00000020414BB8E 088	mov rax, 162	A674742444D5Fh	19 v8 = sub_20414AB1A(a1, dec_buf);
.text:00000020414BB98 088	mov [rsp+88h	<pre>+enc_buf], rax</pre>	20 return v8(a2, a3, a4);
.text:00000020414BB9D 088	mov r10, rcx		• 21}
.text:00000020414BBA0 088	lea rdi.[rs	p+88h+dec_buf]	
 ✓ ✓	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1		🗅 Local Bodn((x64) dobugger 🕐 😢 👔 🕆 🌮 🌾 🌩 🍠 😝 👷 😡
Library function 🧮 R	egular function 📕 Instruction 📒 Data 📕	Unexplored 📕 External symbol 📕 Lumina functio	on
	Debug View		Structures 🗵 🔃 Enums
IDA View-RIP			🗆 🗗 🗙 🛐 deREferencing - Registers
.text:0000 .text:0000 .text:0000 .text:0000	00020414BB84 var_41 00020414BB84 dec_bu 00020414BB84 var_2D 00020414BB84 000020414BB84	= byte ptr -41h f = byte ptr -40h = byte ptr -2Dh	
• text:0000	000020414BB86 008	push r12	
• text:0000	000204148888 010	nush rdi	
• text:0000	000204148800 010	nush rsi	
• text:0000	000204148884 020	sub rsp 68h	
• toxt:0000	00020414BB8F 088	m_{0V} rax 162467//7//2/////	DSEh
• text:0000	000204140002 000	mov [rsp+88b+enc buf]	ray
• toxt:0000	000204148890 088	mov proprositenc_bury,	
. text. 0000		los ndi Exertentes t	

From here you can trace the code as usual or simply set a breakpoint at the address after finished decryting the string and press **F9**. The resulting at **rdx** register will point to the decrypted string as follows:

i 📂 🖥	← - → - ♠ ♠ ♠ ♣ ♣ ♪ ≶	🧧 🖾 🥥 🛛 📩 💼	at st • 🖈 🖬 🗙	🕨 🔲 🔲 Local Bochs(x64) debugge	r 🔻 🔁 🕻	2	* 🛛 🕯	19 4				
Libr	🗧 Library function 📙 Regular function 📕 Instruction 📕 Data 📕 Linexplored 📕 External symbol 📕 Limina function											
	Debug View			Structures			×	Ħ	Enums			
IDA	/iew-RIP				8 × [deREferenc	cing - Registers					
	.text:000000020414BBE0	loc_20414BBE0:		; CODE XREF	: s^ *R/	X 0000	000000000	0001				
e 👘	.text:000000020414BBE0 088	movsx	eax, [rsp+rc	x+88h+dec_buf]	*R	BX FFFF	C0000000	0000				
	.text:000000020414BBE5 088	sub	eax, 24h ; '	\$'	*R(<u>x 0000</u>	000000000	0013		_		
	.text:000000020414BBE8 088	imul	eax, -2Ch ;	'Ô'	*RI	DX 0000	000020417	8F58)		
	.text:000000020414BBEB 088	cdq			AR	0000	000020417	8F6B	-> 000000000000000			
	.text:000000020414BBEC 088	idiv	r9d		×RS	0000	000020417	8F58)		
•	.text:000000020414BBEF 088	lea	eax, [rdx+7F	h]	*R8	0000	0000000000	0000				
•	.text:000000020414BBF2 088	cdq			*R9	0000	0000000000	007F				
•	.text:000000020414BBF3 088	idiv	r9d		*R1	LO 0000	000020414	BB84	(.text ! sub_20414BB84)	→ push r13		
•	.text:000000020414BBF6 088	mov	[rsp+rcx+88h	+dec_buf], dl 🖊	*R1	0000	0000000000	0000				
•	.text:000000020414BBFA 088	inc	rcx		*R1	0000	0000000000	0001				
•	.text:000000020414BBFD 088	cmp	rcx, 13h		*R1	0000	0000000000	0000				
. i.e	.text:000000020414BC01 088	jnz	short loc_20	414BBE0	*R1	6000	000000000	0000				
	.text:000000020414BC01				*R1	5 0000	000000000	0000				
•	text:000000020414BC03 088	lea	rdx, [rsp+88	h+dec_buf]	*R	3P 0000	000000000	0000				
RIP	.text:000000020414BC08 088	mov	rcx, r10		*R	P 0000	000020417	8F10	→ 0000000000000000000			
•	.text:000000020414BC0B 088	mov	[rsp+88h+var	_5C], r8d	*R	CP 0000	900020414	BC08	(.text ! sub_20414BB84+8	4) \rightarrow mov rcx, r10		
•	.text:000000020414BC10 088	call	sub_20414AB1	A	*E	L 0000	000000000	0046				
	.text:000000020414BC10											
•	.text:000000020414BC15 088	mov	r8d, [rsp+88	h+var_5C]								
•	.text:000000020414BC1A 088	mov	rdx, r13									
•	.text:000000020414BC1D 088	mov	rcx, r12		3							
•	.text:000000020414BC20 088	add	rsp, 68h		9	Threads						
•	.text:000000020414BC24 020	рор	rsi		Dec	imal	Hex Stat	te	Name			
•	.text:000000020414BC25 018	рор	rdi		9	4884	1314 Rea	dv	sub 20414BB84			
								1	-			
	0000B008 00000020414BC08: sub_204	14BB84+84			~							
	<				>							
Õ	Hex View-1 🛛	Hex View-2	🛛 🖸 He	x View-3 🛛 🖸	Hex View-4	E	X		l deREferenci	ng - Stack		
00000	00204178F40 00 00 00 00 00	5F 4D 44 42 42	7 67 2A 16 4D	79 64 MDBGg*.Myd					^ 00:0000	0000000204178F10 ->		
00000	00204178F50 16 4D 28 36 7B	4D 3C 24 47 65	5 74 4D 6F 64	75 6C .M(6{M<\$GetModul					01:0008	0000000204178F18 ->		
00000	00204178F60 65 46 69 6C 65	4E 61 6D 65 53	7 00 00 00 00	00 00 eFileNameW					02:0010	0000000204178F20 ->		
00000	00204178F70 00 00 00 00 00	00 00 00 CF 4:	1 00 00 00 CO	FF FFÏAÀÿÿ					03:0018	0000000204178F28 ->		

4. References

End.

m4n0w4r