# DE-Cr1pt0r tool - The Cr1pt0r ransomware decompiled decryption routine

**e** resolverblog.blogspot.com/2019/03/de-cr1pt0r-tool-cr1pt0r-ransomware.html

AC.	File	Edit	Search	V	lew	Ana	lysis	100	Is 1	Wind	low	Hel	р								
FD	enc	FD	enc.out	tm	0																
(	Offse	et (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	OF	Decoded text		
	0000	1 F40	38	16	AI	C9	27	C3	DA	15	E9	71	2E	A2	92	23	3B	4 F	:.:É'ÃÚ.ég.¢š‡:0		
	00001	1F50	67	AS	2C	97	4A	E4	AA	CA	E2	4E	B8	CC	A3	1A	CO	8A	g",-JäªÊâN,Ì£,ÀŠ		
	0000	1F60	1D	CA	8C	2C	16	C8	CC	75	06	87	BB	09	FB	3A	F7	7D	.ÊŒ, .ÈÌu.‡».û:÷}		
	0000	1F70	ED	48	CD	F5	83	CC	В9	2C	01	D3	DD	39	02	FD	80	7F	íHÍðf̹,.ÓÝ9.ý€.		
	00001	1F80	91	34	E5	98	E7	82	Al	69	FB	E5	16	BC	9A	3A	59	80	'4å~ç,;iûå.4š:Y€		
	00001	1F90	42	3B	60	04	EC	5D	47	F7	7B	E5	67	16	64	4A	00	89	B;`.i]G÷{åg.dJ.%		
	0000	1FA0	70	BD	EE	53	4A	A8	30	5E	F9	BC	C9	FF	95	C7	F3	EB	p⊁îSJ″0^ù₩Éÿ•Çóë		
	0000	1FB0	6A	82	BD	11	D3	65	66	CB	B5	59	5C	EE	26	F2	8C	76	j,≒.Óef˵Y\î&òŒv		
	0000	1FC0	25	D3	03	AD	FO	A8 1	EE	65	3B	CC	A5	CE	8E	CC	E5	F7	<pre>%Óð"îe;̥ΎÌå÷</pre>		
	00001	1FD0	7A	A7	E6	B6	4E	58	B1	05	5E	OD	46	CF	<b>B</b> 3	C8	53	D9	z§æ¶NX±.^.FϰÈSÙ		
	0000	1FE0	EE	CO	F7	75	38	86	EC	13	6D	37	74	2F	7D	8C	87	25	îÀ÷u8†ì.m7t/}Œ‡%		
	0000	lFFO	48	94	16	6E	5D	A6	9D	DB	19	94	54	07	86	10	3C	29	H".n]¦.Û."T.†.<)		
	00002	2000	Bl	51	52	5E	86	12	5E	71	3D	9B	65	B5	1E	BO	59	40	±QR^t.^q=>eµ.°Y@		
	00002	2010	C7	91	AA	4A	CC	8B	7C	01	6C	B6	6C	A6	84	27	B3	92	Ç`*JÌ< .1¶1¦"'*'		
	00002	2020	DB	C6	34	D8	20	06	СС	30	Dl	CA	AO	95	16	5F	41	Bl	ÛÆ4Ø .ÌOÑÊ •A±		
	00002	2030	13	C6	BB	30	DD	B3 .	A2	5D	49	5D	85	C8	10	5B	2F	49	.Æ≫0ݰ¢]I]…È.[/I		
	00002	2040	3F	81	F3	60	04	47	87	Al	D8	F5	16	E3	EE	D3	CD	E7	?.ó`.G‡;Øõ.ãîÓÍç		
	00002	2050	76	51	5B	88	33	75.	AA	FE	CF	D2	10	ЗF	10	04	6E	20	vQ[^3uªþÏÒ.?n		
	00002	2060	Fl	41	19	4C	08	33	C6	C2	AF	90	56	44	E9	77	1A	60	ñA.L.3ÆÂ .VDéw.`	lenght 0x50 - CipherText	+
	00002	2070	8C	F4	3C	BO	1B	B3	0A	DO	9F	BO	EE	DF	22	93	E4	2B	Œô<°.'.ĐŸ°îß""ä+		
	00002	2080	FF	D7	24	9D	ΔD	FR	RC	FR	37	70	26	R4	20	31	DC	92	ÿ×\$û4aë: )´,1Ü,		
	00002	2090	49	9E	14	C.3	55	3D	E9	0.6	BF	D4	F!	ЗD	ЗF	78	63	ЗE	Iž.ÃU=é.¿Ôõ=?xc>	lenght 0x20 - Pubkey	+
	00002	20A0	A6	A7	99	C4	DC	F2	52	2D	90	21	C5	10	31	DE	6B	A3	¦§™ÄÜòR!Å.lÞk£	-	
	00002	20B0	EB	CF	06	TC	C5	CA	F8	18	43	C5	2F	5F	43	72	31	70	ëÏÅÊøøCÅ/ <mark>_Crlp</mark>	length 0xA	=
	00002	20C0	74	54	30	72	5F												tTOr	ion Ban over	

## Hello Everybody,

after so many articles(1 - 2 - 3) about my research on this Cr1ptor ransomware finally there is a tiny way to decrypt your files.

## SPOILER ALERT:

This is a very early alpha release, is destined to programmers not directly to the victims. Calm down, this will not be quick and/or easy at all but there is only a theorical chance. Probably you'll need few months, years, your son's life of computational work to brute the key. **This is not a solution**.

Let's start from the beginning:

as I wrote in the last article I got chance to have a pair of valid keys to run some tests on my Raspberry PI VM.

Before to talk about the source code, I need you to focus on the encrypted files's structure:

le ter :	Search 1	liew	Acu	hp: As	Ter	de l	Wier	lew.	Hel	p.								
- 2 -	N. ANDA	p)																
Carlo 2 and	00 01	02	03	04	65	04	0T	-	09	08.	08	80	00	02	17	Decoded text		
0017100	38.14	2.5	0.0	27	63	DG.	15	2.9	13	28	3.2	53	23	38	47	7.18-10.eg.+8410		
007100	47.51	20	17	45	E 4	2.0.	CA	2.2	42	20	00	2.7	18.	00	15	9",-28*248,24.A3		
00750	10 0	80	20	1.4	69	65	76	9.6	61	55	99	15	34.	17	7.0	. Bl Rin. to . 8111		
017220	ED 11	CD.	35	0.5	0C	0.9	2¢	0.1	DO	0D	21	62	370	00	7.8	Lati/11, .020.94.		
01260	93.01	85	54	87	82	23	65	rn.	85	14	80	54	14	.59		16175,1101,142-700		
01790	42 23	63	04	RC.	12	4.7	87	78	2.5	97	14	65	42.	00	11	81,1710+1981-01-2		
NOTAO.	70.30	EE	-53	53	23	н	58	12	EC.	09	r:	55		$\mathbf{r}$	20	\$4157.0-8MB3+098		
03780	42, 12	80	11	25	6.5	44	03	85	1.9	30	11	26	72	8C	74	3.4.0effat11480v		
ert.	25.01	.03	MD.	10	h1	e.e	45	35		A5	1	15		85	1.1	400 1e:1#1214-		
00100	78.30	EX	204	400	10	21.2	03	NK.	00	4.6	9	8.3	-	1.3	29	seriors.".FI*ESS		
SIFEO	RE CO	177	75	20	15	EC.	1.9	60	22	24	27	22	10	07	25	LANGETL. MTL/ 1204		
carro	-	14	12	10	2.6	10	28	2.9	11	2.4	97		10	80	29	#".6[[.U."T.4.<]		
022000	81 51	122	10	24	Ξ.	25	71	22	20	12	22	25	22	22	40	POK-1, -Q* HDD8		
62010	67.71	2,0	10	-	-	14	24	50	22	22		22	11		24	C. CONTRACT.		
02010	00 01	24	- 22	22	-	200	22	20	-	20	22		25	-1		LEAD .LOSE . AL		
		-	-	-	11			22		10			-	-		The state of the s		
0.0000	6-1	-12	-17	÷.	÷	÷	**	-	-12	12	44	÷	-	-2	-	with here and a second		
	12.0		10	-	-		12	1.0	10	14	-				100	as a self order .	lenght 0x50 - CipherText	
62070	10.11	1.0	-	18	10	64	100	10	10	2	22	33	-	2.4	28	3317.1.0771677At	teright only expires terit	
0.000	G		-	100			-	54	1	-	-	-	1	-		140 (Aug.) 11.17		
04463	10.00		114	4.4			10		1.1		-	17				18. Link, different	Jamethe Oy 20 - Dublease	-
02080	24.3	1.5	24	10	11	11	10	10	11		10	11		10		14-0.008- 15.1PM	But over a Lange of	
02080	-	- 11	-		-	-	-	-	-			-	-	- 11	11	et. Steels light	to the second	
0.000	114 114	- 11	-	1.1	_	_	-	-	-	-	-	-	-	-	-	No. of Lot of Lo	length 0xA	

Basically this ransomware append after encryption 0x7A bytes. This is important because of this:

This is how the Decryption routine looks initially (where I made some gusses..):



Studying the code from here is completely INSANE......well, I did it anyway (tens hours of hard work) helped by the <u>libsodium documentation</u> I figured out the exact pseudo code of the decompilation routine built in the ransomware and what do it exactly do. After a few IDA corrections here is where I landed:



Most of functions now looks familiar, especially those concerning libsodium and files manipulation.

Starting from here, I've ported the code into a C application to reproduce the decryption.Once figured out which kind of encryption the ransomware adopted, I've started to write a C program and from the libsodium documentation there was something interesting:



Well, this looks similar to our ransomware implementation, except for the fact that he's doing some manipulation on the top of the pseudo code, in fact this code example is not sufficient. A sealed box implementation seems to anticipate the code we seen:

## Sealed boxes

### Example



Good. We now have so many pieces of the puzzle. Its time to put them togheter. What do we need to decrypt the files?

Take a closer look at the "crypto\_box\_seal\_open" function. Do you remember the encrypted structure? CIPHERTEXT\_LEN, from the bottom of file is 0x50. We have it. recipient\_pk, from the bottom of the file and is 0x20. We have it. recipient\_sk, from the end of.....No. Unfortunaly we haven't the secret key. The result decrypted array is then used to decrypt the rest of the file more or less as decribed on the libsodium documentation secret-key\_cryptography -> "Stream encryption/file encryption" on github. To procede with

```
crypto_secretstream_xchacha20poly1305_init_pull(&st, header, key) != 0)
```

we need the header and the key. The header is actually stored into the encrypted file as the same as the example shows. So we have it.

the key....the key is the "decrypted" crypto\_box\_seal\_open resoult! We have it.

Since the fact I had a working keypair, I had everything I need to run some tests with the good old DEV-C++ IDE.

Once set up the code, I found a very strange behaviour of libsodium which brings me to a correct decryption with 3 different private keys!(!?!?!?!?) O.o

110					
118					
176	signed they wellinger	EDD-IN-MANY	MANAGAMANA AND A MANAGAMAN	UNAMA ANALAN	
110	Underforce the 12byte	Any. Tranks to Gel	alle from DC Mprogramming # /	reamanda	
179 a 140	transfor \$(4) - (min)	11-To	BullCOCHTMASSARDS .	Bulassoerrishikasa ,	write service internal interna
1423 A	111(1) = 4:				
345	111223 - 41				
347	valat - et				
110 000(2) 151	(Brecipient_sk)115.40 skrsp(Bracipient_sk)	11,4536400000			
153 Acarp) 158	offenciation_sACIS_AC interceptionstations_sAC	11, sisse#(3[2]])) 21, 0, 4),			
156 demision 137 - repres 156	(Brealighest_sh(1), hit) externet(brealighest_sh(	Transia (DDD)			
110 memcaj 358 rverm 361	(Brecipiert_0, 4119), administration_abut	1111			
all prints	wayfrecipient (8, 32	li rystal, cipterbest,	Bride, recipient pi, recipient		
204 <b>11200</b>	tinfords minister	ofmitplet, w. 12	SUPPLIT-SUPPLIED		
208					m
158 -	Labola Colore C. D	(1) IQ or a UCNISA, N	adden (Arroll)		
179	] while (1)) - + unk	144 Manual Providence (			
and a generation	iting [\$ feel feath @	Case			
-	Casting .	The later of the	Send command to GDB privacipient,	400	
Doce	ne Loone	These and the years	dan history value		
er, Deta fun	See States	Interestion	Durild .		
		- W			
			toward bollogy and		
// 8405	sectors/_st[0]	- 0400, 08 643		eremenfally? 0.v	
// Kuis	sautossi_st(0)	+ 0xDC, OR 6x30	to property the ends of the description ends of the	eremental15° 0.v	
// Kats	sestposs_sk(c)	- DaDC, GE Call	t - paragraphi t - parag	nonesfally: 0.v	1400 - 1417 - 1418 - 141 <sup>4</sup>
// Mass	sunipassi_sk[0] 6 char secipiess_	- 1400, 08 043		0440 - 0474 - 0474 - 0450 - 0474 - 0474 - 0450 - 0475 - 0475 -	5eC0 , 5eB7 , 5e28 , 5eC0 , 5e59 , 5e28 , 5e28 , 5e27 , 5e49 , 5e27 , 6e37 , 6e84 , 6e79 , 5e27 , 6e37 , 6e84 ,

Is due to a libsodium bug?! IDK!

I hope some of you knows (and tells me) the reason of such behaviour, by the way victims does not have the private key and this strange behaviour of libsodium motivated me to implement a brute force routine into the code (to MAYBE find a working decryption sk with humanly acceptable timing).



And here is the result (skipping the brute force routine)!



(and yes, the original file was filled by a junky 0xAA  $\cong$ )Here is the main source (I do not share the encryption routine to avoid a Cr1pt0r x86 porting)

```
#include
#include
#include
#include
#include
#include
#include
#include
#define UINT64_MAX (18446744073709551615ULL)
#define CHUNK_SIZE 4096
#define crypto_stream_chacha20_ietf_KEYBYTES 32U
void rvereseArray(unsigned char *arr, int start, int end)
{
    while (start < end)</pre>
    {
        unsigned char temp = arr[start];
        arr[start] = arr[end];
        arr[end] = temp;
        start++;
        end--;
    }
}
void printArray(unsigned char arr[], int size)
{
  int i;
  for (i=0; i < size; i++)</pre>
    printf("%02x ", arr[i]);
  printf("\n");
}
static int
decrypt(const char *target_file, const char *source_file, const unsigned char
key[crypto_secretstream_xchacha20poly1305_KEYBYTES])
{
    unsigned char buf_in[CHUNK_SIZE + crypto_secretstream_xchacha20poly1305_ABYTES];
    unsigned char buf_out[CHUNK_SIZE];
    unsigned char header[crypto_secretstream_xchacha20poly1305_HEADERBYTES];
    crypto_secretstream_xchacha20poly1305_state st;
    FILE
                  *fp_t, *fp_s, *fp_s1;
    unsigned long long out_len;
    size_t
                   rlen;
    int
                   eof;
                   ret = -1;
    int
    unsigned char tag = 0 \times 0;
#define MESSAGE (const unsigned char *) "Message"
#define MESSAGE_LEN 15
#define CIPHERTEXT_LEN (crypto_box_SEALBYTES + MESSAGE_LEN)
```

```
unsigned char recipient_pk[crypto_box_PUBLICKEYBYTES];
//My pk
//unsigned char recipient_pk[crypto_box_PUBLICKEYBYTES];={0x3D , 0x3F , 0x78 , 0x63 ,
0x3E , 0xA6 , 0xA7 , 0x99 , 0xC4 , 0xDC , 0xF2 , 0x52 , 0x2D , 0x90 , 0x21 , 0xC5 ,
0x10 , 0x31 , 0xDE , 0x6B , 0xA3 , 0xEB ,
             //0xCF , 0x06 , 0x1C , 0xC5 , 0xCA , 0xF8 , 0xF8 , 0x43 , 0xC5 ,
0x2F};//; /* Bob's public key */
11
//recipient_sk decrypt the files also with recipient_sk[0]=0xDB (the original byte)
than OxDD and also OxDC
// unsigned char recipient_sk[crypto_box_SECRETKEYBYTES]={ 0xDB , 0xA2 , 0xD4 , 0x74
, 0xC0 , 0xB7 , 0x2B , 0x62 , 0x0E , 0xCD , 0xC8 , 0x7F , 0x43 , 0xEA , 0xAB , 0x2E ,
0x24 ,
//0x65 , 0x00 , 0x91 , 0x74 , 0xDC , 0x03 , 0xB4 , 0x22 , 0xC8 , 0x48 , 0x30 , 0x1F ,
0x19 , 0xDD , 0x78 }; //; /* Bob's secret key */
unsigned char recipient_sk[32]=
unsigned char ciphertext[80];
//var50h
long filelen:
fp_s1 = fopen("enc", "rb");
  if(fp_s1==0){ printf("Encrypted files not found"); return 1;}
   fseek(fp_s1, 0xFFFFFD6, SEEK_END);
   //filelen = ftell(fp_s1);
  fread(recipient_pk, 1, 0x20, fp_s1);
  fseek(fp_s1, 0xFFFFF86, SEEK_END);
  fread(ciphertext, 1, 0x50, fp_s1);
//because the new key is 32dec bytes long
unsigned char decrypted[32];
//brute start
//Bruteforce the 32byte key. Thanks to GeDaMo from IRC ##programming @ freenode
11
uint64_t i[4] = { 0, 0,0,0 };
// uint64_t i[4] = {0xDBA2D474C0B72B62 , 0x0ECDC87F43EAAB2E ,
0x2465009174DC03B4 , 0x22C848301F19DD78 };
     int exit=0;
 printf("DE-Cr1pt0r Tool By RE-Solver @solver_re:\r\n Bruteforcing \r\n ");
  do {
  i[1] = 0;
  do {
  i[2] = 0;
 do {
  i[3] = 0;
     do {
```

```
memcpy(&recipient_sk[23],&i[3],sizeof(i[3]));
rvereseArray(&recipient_sk[23],0,8);
memcpy(&recipient_sk[15],&i[2],sizeof(i[2]));
rvereseArray(&recipient_sk[15],0,8);
memcpy(&recipient_sk[7],&i[1],sizeof(i[1]));
rvereseArray(&recipient_sk[7],0,8);
memcpy(&recipient_sk,&i[0],sizeof(i[0]));
rvereseArray(recipient_sk,0,7);
//printArray(recipient_sk, 32);
if(crypto_box_seal_open(decrypted, ciphertext, 0x50u, recipient_pk, recipient_sk) ==
0)
{printf("Found: ");printArray(recipient_sk, 32);exit=1;break;}
           } while (i[3]++ < UINT64_MAX&&exit==0);</pre>
        } while (i[2]++ < UINT64_MAX&&exit==0);</pre>
   } while (i[1]++ < UINT64_MAX&&exit==0);</pre>
 } while (i[0]++ < UINT64_MAX&&exit==0);</pre>
//END brute
//from the decompilated program this was the original routine, because of the
bruteforce is added as a comment now
/*
if (crypto_box_seal_open(decrypted, ciphertext, 0x50u, recipient_pk, recipient_sk) !=
0) {
    // message corrupted or not intended for this recipient
     printf ("message corrupted or not intended for this recipient %s", decrypted);}
*/
    fp_s = fopen(source_file, "rb");
    fp_t = fopen(target_file, "wb");
    if(fp_s==0 || fp_t==0){printf("Encrypted files not found");
 goto ret;}
    fread(header, 1, 0x18, fp_s);
    if (crypto_secretstream_xchacha20poly1305_init_pull(&st, header, decrypted) != 0)
{
        goto ret; /* incomplete header */
    }
    do {
        rlen = fread(buf_in, 1, 0x1011, fp_s);
        eof = feof(fp_s);
        //tag = 0x0;
        int value=crypto_secretstream_xchacha20poly1305_pull(&st, buf_out, &out_len,
&tag, buf_in, rlen, NULL,0);
        if (value != 0) {
            goto ret; /* corrupted chunk */
        }
        if (tag == 3 && ! eof) { //crypto_secretstream_xchacha20poly1305_TAG_FINAL ->
3
            goto ret; /* premature end (end of file reached before the end of the
stream) */
```

```
}
        fwrite(buf_out, 1, (size_t) out_len, fp_t);
    } while (! eof);
    ret = 0;
ret:
    fclose(fp_t);
    fclose(fp_s);
    fclose(fp_s1);
    return ret;
}
int
main(void)
{
    unsigned char key[crypto_secretstream_xchacha20poly1305_KEYBYTES];
    if (sodium_init() != 0) {
        return 1;
    }
    crypto_secretstream_xchacha20poly1305_keygen(key);
    if (decrypt("enc.outtmp", "enc.tmp", 0x0) != 0) {
     printf("Something goes wrong.");
        return 1;
    }
    printf("Decrypted! RE Solver");
    return 0;
}
```

IDE: <u>DEV-C++</u> libsodium library: <u>libsodium-1.0.17-mingw.tar.gz</u> Remember to link the library into the Project/Project Options

Compiled tool: https://www.sendspace.com/file/275c70 sha256: 4066fa0d402a8458f7784e89ba979929ee1d7efd761b3cabe9705784aa8af865

usage: Copy an encrypted file into the same folder of the tool and rename it as enc (with no extensions). Copy the same encrypted file and rename it as enc.tmp and strip the last 0x7A from the end of the file. If you're lucky within some weeks you'll have the key printed on the console and the encrypted.outtmp decrypted file created on the same folder dir.

Next step: create a file named privkey and write the hex key (with no spaces) into a text file and put it in the Cr1pt0r folder. From the same folder, rename the file pubkey as pubkey\_backup and turn on your D-Link nas again.

Note: My GF is waiting me since days, she has been so patient. A special Thanks to her. I'm sorry but I do not support the tool usage or others kind of requests. Since the fact that code is released under GPL, everyone can compile, improve, modify the code. (And I hope it happens).

Follow me on Twitter @solver\_re Hire me! Job offers are welcome. Cheers, RE Solver