Shad3-Keyg3n_M1#1

https://crackmes.one/crackme/5e66aea233c5d4439bb2dde8

Crackme writeup by @H0l3Bl4ck https://twitter.com/H0l3Bl4ck

crackmes.one user b1h0 https://crackmes.one/user/b1h0

Date: 10/abr/2020

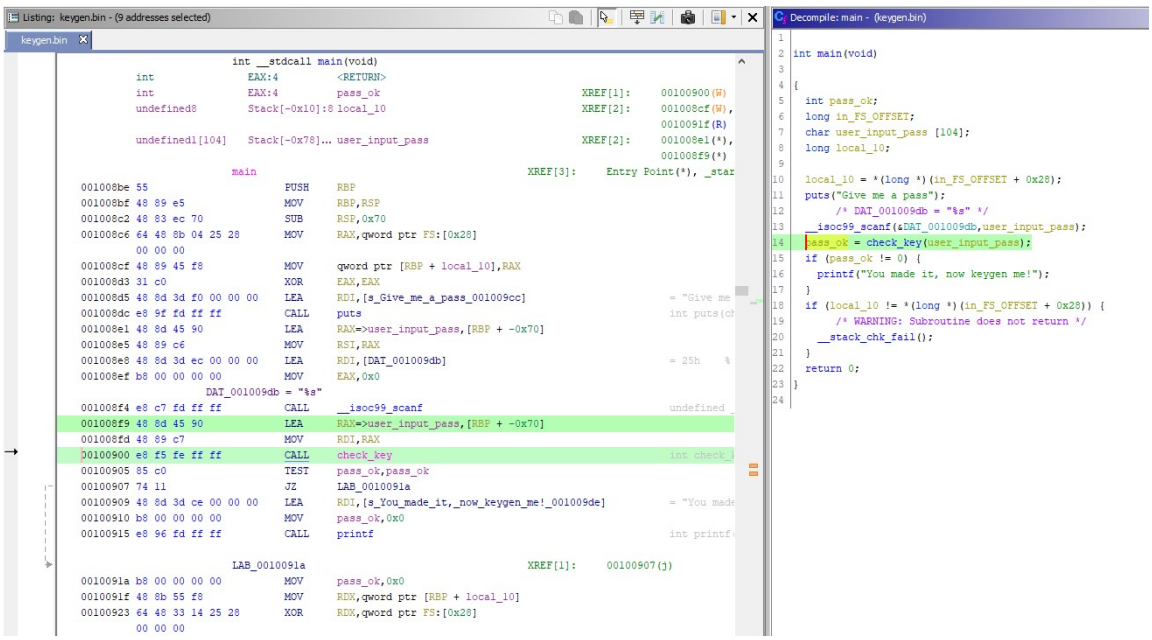
We have here a crackme that asks it to break and generate a keygen.

It claims to be cross-platform, but the binary is only in **ELF** format for Linux.

Let's start with the static analysis with Ghidra ...

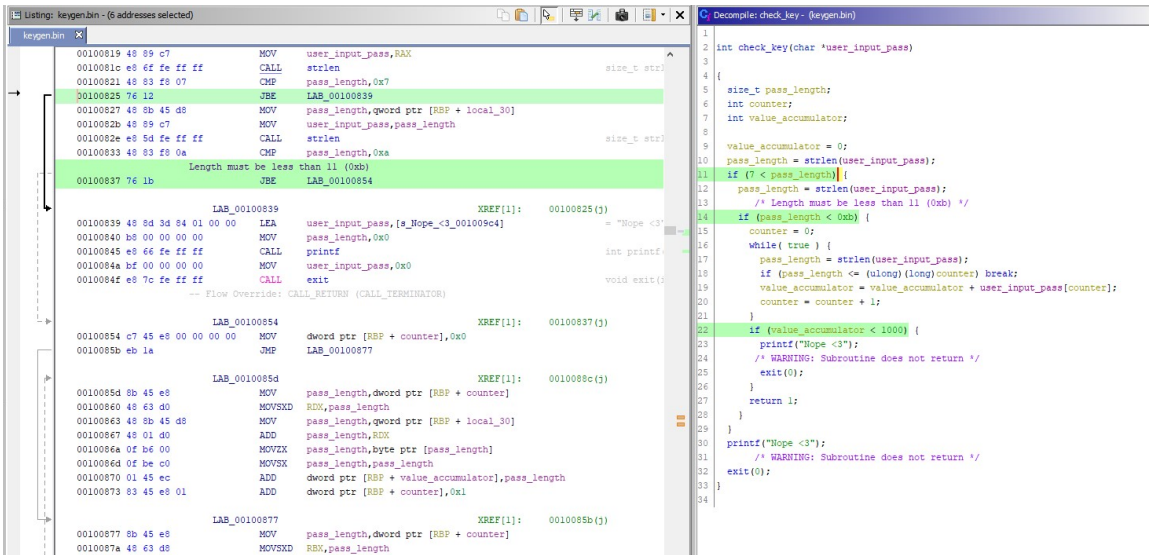
Ghidra's static analysis

After carrying out the initial analysis, looking for the **main()** function and substituting some variable names to make the code clearer and easier to understand, we have the following function.



We are asked to enter a password that will be stored in a variable that we will call **"user_input_pass"**, and then the **check_key()** function is called to check if the password is correct.

The function returns a value other than 0 if we have found the correct password.



Here we can observe 3 conditions (or 2 if we reduce the first two if in one with and condition):

1. The password length must be **greater than 7 and less than 11**. That is, it can be 8, 9 or 10 characters.

2. The sum of the **ASCII** values of all the digits of the password must be **equal to or greater than 1000**.

We also start from the condition, not written, that the password has easily visible or printable characters. So we could consider valid all characters from 33 (! After the blank space) to 126 (~).

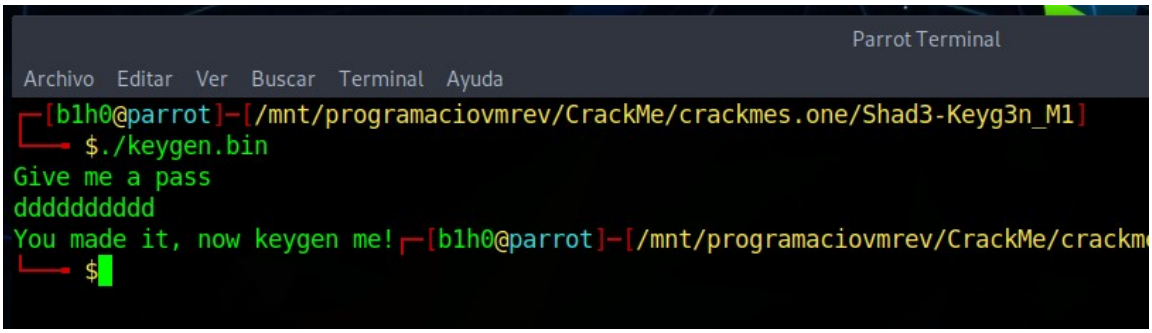
ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]	48	30	110000	60	0	96	60	1100000	140	`
1	1	1	1	[START OF HEADING]	49	31	110001	61	1	97	61	1100001	141	a
2	2	10	2	[START OF TEXT]	50	32	110010	62	2	98	62	1100010	142	b
3	3	11	3	[END OF TEXT]	51	33	110011	63	3	99	63	1100011	143	c
4	4	100	4	[END OF TRANSMISSION]	52	34	110100	64	4	100	64	1100100	144	d
5	5	101	5	[ENQUIRY]	53	35	110101	65	5	101	65	1100101	145	e
6	6	110	6	[ACKNOWLEDGE]	54	36	110110	66	6	102	66	1100110	146	f
7	7	111	7	[BELL]	55	37	110111	67	7	103	67	1100111	147	g
8	8	1000	10	[BACKSPACE]	56	38	111000	70	8	104	68	1101000	150	h
9	9	1001	11	[HORIZONTAL TAB]	57	39	111001	71	9	105	69	1101001	151	i
10	A	1010	12	[LINE FEED]	58	3A	111010	72	:	106	6A	1101010	152	j
11	B	1011	13	[VERTICAL TAB]	59	3B	111011	73	;	107	6B	1101011	153	k
12	C	1100	14	[FORM FEED]	60	3C	111100	74	<	108	6C	1101100	154	l
13	D	1101	15	[CARRIAGE RETURN]	61	3D	111101	75	=	109	6D	1101101	155	m
14	E	1110	16	[SHIFT OUT]	62	3E	111110	76	>	110	6E	1101110	156	n
15	F	1111	17	[SHIFT IN]	63	3F	111111	77	?	111	6F	1101111	157	o
16	10	10000	20	[DATA LINK ESCAPE]	64	40	1000000	100	@	112	70	1110000	160	p
17	11	10001	21	[DEVICE CONTROL 1]	65	41	1000001	101	A	113	71	1110001	161	q
18	12	10010	22	[DEVICE CONTROL 2]	66	42	1000010	102	B	114	72	1110010	162	r
19	13	10011	23	[DEVICE CONTROL 3]	67	43	1000011	103	C	115	73	1110011	163	s
20	14	10100	24	[DEVICE CONTROL 4]	68	44	1000100	104	D	116	74	1110100	164	t
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]	69	45	1000101	105	E	117	75	1110101	165	u
22	16	10110	26	[SYNCHRONOUS IDLE]	70	46	1000110	106	F	118	76	1110110	166	v
23	17	10111	27	[ENG OF TRANS. BLOCK]	71	47	1000111	107	G	119	77	1110111	167	w
24	18	11000	30	[CANCEL]	72	48	1001000	110	H	120	78	1111000	170	x
25	19	11001	31	[END OF MEDIUM]	73	49	1001001	111	I	121	79	1111001	171	y
26	1A	11010	32	[SUBSTITUTE]	74	4A	1001010	112	J	122	7A	1111010	172	z
27	1B	11011	33	[ESCAPE]	75	4B	1001011	113	K	123	7B	1111011	173	{
28	1C	11100	34	[FILE SEPARATOR]	76	4C	1001100	114	L	124	7C	1111100	174	|
29	1D	11101	35	[GROUP SEPARATOR]	77	4D	1001101	115	M	125	7D	1111101	175	}
30	1E	11110	36	[RECORD SEPARATOR]	78	4E	1001110	116	N	126	7E	1111110	176	~
31	1F	11111	37	[UNIT SEPARATOR]	79	4F	1001111	117	O	127	7F	1111111	177	[DEL]
32	20	100000	40	[SPACE]	80	50	1010000	120	P					
33	21	100001	41	!	81	51	1010001	121	Q					
34	22	100010	42	"	82	52	1010010	122	R					
35	23	100011	43	#	83	53	1010011	123	S					
36	24	100100	44	$	84	54	1010100	124	T					
37	25	100101	45	%	85	55	1010101	125	U					
38	26	100110	46	&	86	56	1010110	126	V					
39	27	100111	47	'	87	57	1010111	127	W					
40	28	101000	50	(	88	58	1011000	130	X					
41	29	101001	51	)	89	59	1011001	131	Y					
42	2A	101010	52	*	90	5A	1011010	132	Z					
43	2B	101011	53	+	91	5B	1011011	133	[					
44	2C	101100	54	,	92	5C	1011100	134	\					
45	2D	101101	55	-	93	5D	1011101	135	]					
46	2E	101110	56	.	94	5E	1011110	136	^					
47	2F	101111	57	/	95	5F	1011111	137	_					

Thus, based on these premises, we can quickly calculate a valid password. For example, any character with an ASCII value equal to or greater than 100 (lowercase letter d) and repeated 10 times, will give us a valid password.

Example: **ddddddddd**

Let's check it ...



Based on this, we can easily make a valid password generator.

Password generator

We are not going to complicate much. You just have to do a function that adds the ASCII values of a string and prints the generated strings that return a value of 1000 or higher.

For this we generate the sequence of digits character by character within the range that we need and we make 10 nested loops, one for each character, and we calculate if the password is valid for the combinations of 8, 9 and 10 characters.

Since the sum has to be 1000, for 10 characters the minimum character would be the letter 'd', but since we can get to the 126th character, we can subtract 26 positions and start from the 74th character that is the 'J' to find the maximum of valid passwords.

Here is the keygen source code, and you can download it Windows binary executable from this link:

```
/*
 * Shad3-keygen-M1.c
 * Password generator for Shad3-Keyg3n-M1
 * Author: Gabriel Marti
 * Twitter: @H0l3Bl4ck
 */
#include <stdio.h>
#include <string.h>

/* returns ASCII value sum of pass if length are between 8 and 10 */
int key_value(char *current_pass) {
        int pass_length;
        int counter;
        int value_accumulator;

        value_accumulator = 0;
        pass_length = strlen(current_pass);
        if (pass_length > 7 && pass_length < 11 ) {
          counter = 0;
          while(counter < pass_length) {
                value_accumulator += current_pass[counter];
                counter++;
          }
        }
        return value_accumulator;
}


/* Print password if it's valid */
void show_valid_pass( char *pass, int good_pass ) {
        int result;
        result = key_value(pass);
        if ( result >= good_pass) {
                printf("Valid Password %s value( %d )\n", pass, result);
        }
}


int main(int argc, char **argv)
```

```
{
        int good_pass = 1000;
        char c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, first, last;
        char pass[11];

        pass[10] = '\0';
        printf("Searching possible valid passwords ...\n\n");

        first = 'J';
        last = '~';

        for (c1 = first; c1 <= last; c1++) {
                for (c2 = first; c2 <= last; c2++) {
                        for (c3 = first; c3 <= last; c3++) {
                                for (c4 = first; c4 <= last; c4++) {
                                        for (c5 = first; c5 <= last; c5+
                                                for (c6 = first; c6 <= l
                                                        for (c7 = first;
                                                                for (c8 
                                                                }
                                                        }
                                                }
                                        }
                                }
                        }
                }
        }
        return 0;
}
```

Obviously, as there are many combinations, the result can take a long time.

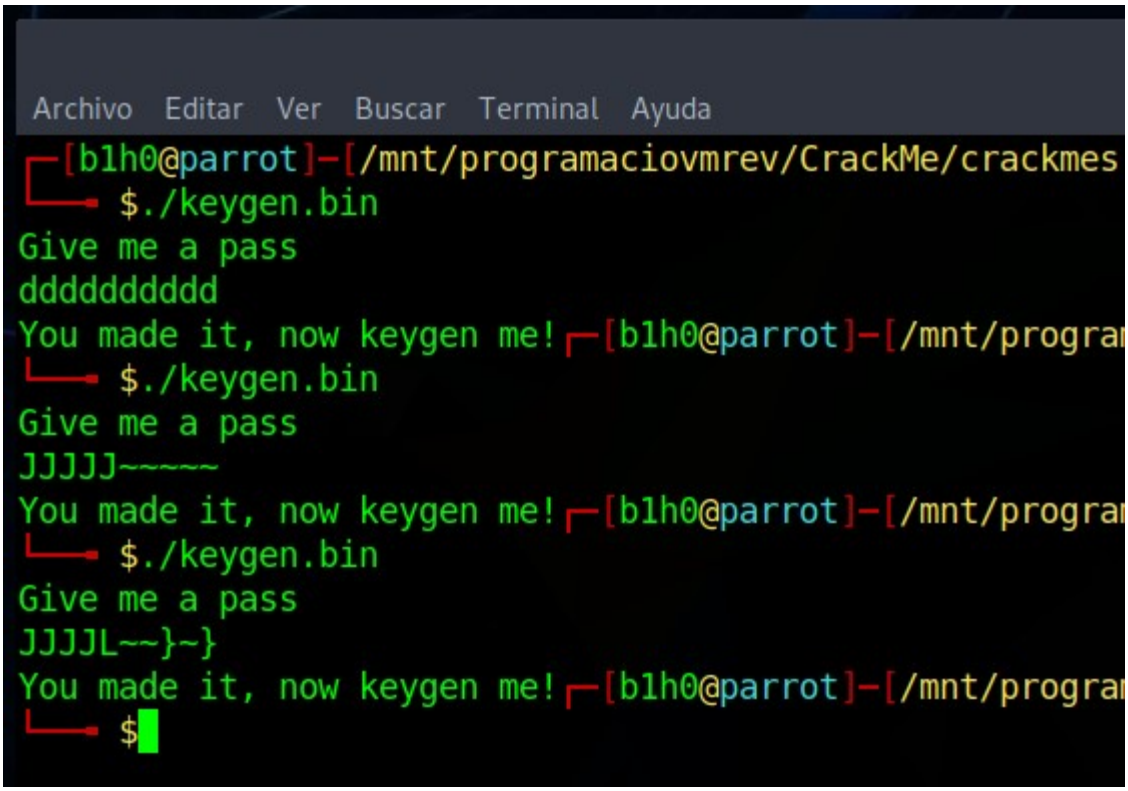
Here we have a part of the passwords that it begins to generate.

```
Símbolo del sistema

C:\DATA\WorkFiles\Workspace1\Shad3-Keyg3n\Release>Shad3-Keyg3n.exe
Searching possible valid passwords ...

Valid Password JJJJJ~~~~~ value( 1000 )
Valid Password JJJJK}~~~~ value( 1000 )
Valid Password JJJJK~}~~~ value( 1000 )
Valid Password JJJJK~~}~~ value( 1000 )
Valid Password JJJJK~~~}~ value( 1000 )
Valid Password JJJJK~~~~} value( 1000 )
Valid Password JJJJK~~~~~ value( 1001 )
Valid Password JJJJL|~~~~ value( 1000 )
Valid Password JJJJL}}~~~ value( 1000 )
Valid Password JJJJL}~}~~ value( 1000 )
Valid Password JJJJL}~~}~ value( 1000 )
Valid Password JJJJL}~~~} value( 1000 )
Valid Password JJJJL}~~~~ value( 1001 )
Valid Password JJJJL~|~~~ value( 1000 )
Valid Password JJJJL~}}~~ value( 1000 )
Valid Password JJJJL~}~}~ value( 1000 )
Valid Password JJJJL~}~~} value( 1000 )
Valid Password JJJJL~}~~~ value( 1001 )
Valid Password JJJJL~~|~~ value( 1000 )
Valid Password JJJJL~~}}~ value( 1000 )
Valid Password JJJJL~~}~} value( 1000 )
Valid Password JJJJL~~}~~ value( 1001 )
Valid Password JJJJL~~~|~ value( 1000 )
Valid Password JJJJL~~~}} value( 1000 )
Valid Password JJJJL~~~}~ value( 1001 )
Valid Password JJJJL~~~~| value( 1000 )
Valid Password JJJJL~~~~} value( 1001 )
Valid Password JJJJL~~~~~ value( 1002 )
^C
C:\DATA\WorkFiles\Workspace1\Shad3-Keyg3n\Release>_
```

And here is the result of trying a couple of these.



That's all folks!