CrackMe 破解我 (212 points)

Solved by Student 2 - <@!668261593502580787>

Decompile the APK file with Bytecode Viewer. By using the FernFlower Decompiler on the file `/com/example/crackme/EnterPasswordActivity$1.class`, on line 36, it validates whether `var6[0] + var6[1] - var6[2] == 112 && var6[1] + var6[2] - var6[3] == 105` and so on, where `var6[n]` is n^{th} character of the supposed flag.

These sets of simultaneous equations can be easily solved by constructing a 26×26 matrix:

$$\begin{bmatrix} 1 & 1 & -1 & 0 & \dots & 0 & 112 \\ 0 & 1 & 1 & -1 & \dots & 0 & 105 \\ \vdots & & & \ddots & \vdots & & \dots \\ -1 & 0 & 0 & 0 & \dots & 1 & 72 \\ 1 & -1 & 0 & 0 & \dots & 1 & 122 \end{bmatrix}$$

and solving it with *numpy* in Python. After the calculation, the unknowns are expressed in an array of ascii character codes: (104,107,99,101,114,116,50,48,123,97,114,51,95,121,48,117,95,117,115,49,110,103,95,122,51,125), which can be converted into the flag using an ASCII converter to get the flag:

hkcert20{ar3_y0u_us1ng_z3}

Hacking the Mystic 解決ミステリー (482 points)

Solved by Student 1 (Captain) - <@!678782733782089748>

In line 1 of the `check()` function, it reads the value of each element in the predefined array of cells `[i1, i15, i26…]` - the location of such cells seems to be random. For every two characters in the array, they are grouped together to form `%{value1}{value2}`, which would later be interpreted as an URI component and decoded into ascii strings, giving the flag.

Since the flag generated is purely dependent on the user's input, there is not a way to reverse the flag. Hence, an online hexadecimal sudoku solver is used to find the correct solution.

However, when pressing the '*Check/resolvability*' button in the sudoku solver, it prompted that there may be several solutions to this sudoku, therefore we needed more information.

Since we know that the flag will be in the form of '`hkcert20{<flag>}`', we can input an extra 20 characters to the sudoku (2*10 as 2 values inputted return 1 character as output after decoding the URI component). Therefore, I encoded '`hkcert20{`' and '`}`' into URI components, which is '`%68%6b%63%65%72%74%32%30%7b`' and '`%7d`' respectively.

By putting each character into the relevant boxes in the solver, the correct solution could be found. We can confirm that this is the one and only solution by verifying it with the '*Check/resolvability*' button once again.

F	6	0	A	C	9	1	5	2	4	B	E	D	3	7	8
7	4	C	2	D	B	3	8	9	A	6	5	1	0	E	F
1	E	8	9	0	2	6	4	D	F	7	3	C	B	5	A
3	D	5	B	E	7	F	A	0	C	1	8	6	4	2	9
B	1	7	C	3	6	9	F	E	5	D	2	0	A	8	4
D	A	6	F	2	4	E	0	3	1	8	7	5	9	B	C
0	9	4	5	8	1	A	7	C	6	F	B	3	2	D	E
E	2	3	8	5	C	B	D	4	9	A	0	7	F	6	1
5	F	A	E	4	0	2	1	6	7	3	C	B	8	9	D
6	B	9	D	F	A	C	3	8	2	5	4	E	7	1	0
8	0	1	3	7	E	5	6	F	B	9	D	4	C	A	2
4	C	2	7	B	8	D	9	A	0	E	1	F	5	3	6
2	8	E	1	9	F	4	B	7	3	0	6	A	D	C	5
A	3	F	4	6	D	7	E	5	8	C	9	2	1	0	B
C	5	D	0	1	3	8	2	B	E	4	A	9	6	F	7
9	7	B	6	A	5	0	C	1	D	2	F	8	E	4	3

Black = inputted value, Red = calculated value.

From this, by inputting the relevant values into each cell and pressing the 'Check' button, it outputted the following flag:

hkcert20{@Rows&Columns_sumss_to_0x78}

F	6			C	9						E	D		7	8
7	4	C	2	D		3	8			6	5	1	0		F
		8		0	2		4		F	7			B	5	A
3		5		E	7	F			C				4	2	
B	1	7	C	3	6	9	F		5	D			A	8	4
D		6	F					3			7	5		B	C
0			5		1	A	7		6		B	3	2		
		3	8	5					9		0	7	F	6	
5	F		E	4		2		6	7	3			8		
6		9	D	F	A	C	3		2	5		E	7		
8			3	7		5	6				D		C		
4			7							E				3	6
2		E					B	7		0	6				
	3	F		6		7	E				9	2	1		B
			0			8			E		A		6	F	7
9	7						C			2				4	3

Welcome to Hacking the Mystic! Normal Sudoku rules apply.

Check

The flag you got is: hkcert20{@Rows&Columns_sumss_to_0x78}

Note 1: Since the cells that are white are not read by the program, to save time it was not inputted as it does not affect the result. ~~Not because we're lazy~~

Note 2: that the references of each cell starts from the top left corner as `i0`, then increments horizontally across until the top right (`i15`), then moves onto the next row (`i16`) and so on.

777 (492 points)

Solved by Student 2 - <@!668261593502580787>

By reading through `777.py` and to understand how it works, it:

- → 34:37 - Returns the exact timestamp of the server, precision up to the nearest second
- → 35:14 - Requires the user to input a valid HKID
- → 41: 9 - Concatenates the value of `current_time % 100` and the `hkid` as the variable 'seed'
- → 42:42 - Encrypts the 'seed' with SHA3-256
- → 44:21 - If the first 3 bytes match `\x81\x24\x58`, print the `flag`, otherwise exit.

The general idea is to figure out the correct HKID in a very short period of time, such that the timestamp in Step 1 and Step 3 are the same. This could be achieved by brute-forcing the HKID prior to the request, and waiting until a very specific time.

As an example, I selected a time at which I decided to perform the request - at 1604777000. By iterating through all possible combinations of hashes:

Time % 100	HKID	sha3-256 hash
000	A0000003	\x1e\x4f\xcb\x7f\xc1\xe0\x4f\x31\x6b\x11\xf5\x52\xeb\x5a\x15\x80\x7a\xb3\xba\xd4\x4c\x07\xb2\x6d\x2a\xa2\x36\x52\x30\x31\x24\xf1
000	A0000011	\x28\x29\x83\x9b\x19\xd3\xac\x52\x60\x1d\xc3\x28\xac\x2d\x13\xa6\x4b\x71\x47\xe2\x17\x8d\xfb\x6c\x93\xab\x8b\xf4\x31\x1a\x9a\xe6
...	...	...
000	H5226750	**\x81\x24\x58**\xd0\xc5\xec\x3b\x3e\x47\xca\xc2\xdb\x60\x8f\xeb\x23\x13\xc9\xb8\xeb\x78\xe4\x68\xd3\xd2\x9d\x5e\x99\x65\xcb\x66\x70

From the above, as long as we enter in the HKID of `H5226750` within a short period of time, for which the server timestamp ends with `000`, it will return the flag:

hkcert20{gum_tin_tin_hei_but_gaai_moe_sum_ching_m_hui_jyu}

Below is the Python script used to find the correct HKID for the specified time:

```python
import hashlib

t = '000' # current_time % 100

# calculates the check digit of the hkid
def get_new(s):
    s1 = [int(s2) for s2 in s[1:]]
    rem = (324 + dct[s[0]]*8 + s1[0]*7 + s1[1]*6 + s1[2])*5 + s1[3]*4 + s1[4]*3 + s1[5]*2) % 11
    if not rem:
        return f'{s}0'
    elif rem == 1:
        return f'{s}A'
    else:
        return f'{s}{11-rem}'

# loops through all combinations of HKID, breaks if found.
def get_match():
    for char in dct:
        for i in range(999999):
            seed = f"{t}{addCheckDigit(f'{char}{i:06}')}"
            b = hashlib.sha3_256(seed.encode('ascii')).digest()
            if b[:3] == b'\x81\x24\x58':
                return f"{seed}: {''.join('\\x{:02x}'.format(l) for l in b)}"
    return ''

print(get_match())
```

Where dct is a dictionary that maps 'A': 10, 'B': 11, and so on.
HKIDs with two-alphabets at the start will be omitted and not checked to minimise the time needed to brute force.