Ks1q2's solves

Screaming Crying Throwing up
Flag: flag{edabfbafedcbbfbadcafbdaefdadfaac}
Solution: Just look up the xkcd cipher, it's in the description

CTF101
Flag flag{3b74fc0628299870edabc5072b25cf78}
Solution: web vulnerable to os command injection, use ls -l to view the directory and go from there

Zero Ex Six One
Flag: flag{c50d82c0a25f3e644d0702b41dbd085a}
Solution: It's a XOR encryption, decrypt it using the hexadecimal key '61', or use an online decrypter.

Free Range Packet
Flag: flag{b5be72ab7e0254c056ffb57a0db124ce}
Solution: Pay attention to the packet details.

ClickityClack
Flag: flag{a3ce310e9a0dc53bc030847192e2f585}
Solution: Apply this filter **((usb.device_address == 2) && (usb.transfer_type == 0x01)) && (usb.data_len == 8)** and use tshark to extract them into a readable txt file. Use the ctf-usb-keyboard-parser to parse the file and get the flag.

String Me Along
Flag:flag{850de1a29ab50b6e5ad958334b68d5bf}
Solution: Run the command **'strings ./string-me-along'**, find the code, then run the executable through **'./string-me-along'**. Input the code to get the flag

An Offset
Flag:flag{c54315482531c11a76aeaa828e43807c}
Solution: Run the command **'ltrace ./an-offset'**, it should reveal the input the parameter is expecting, use it, get the code.

A Powerful Shell
Flag: flag{45d23c1f6789badc1234567890123456}
Solution: The command **'files ./challenge.ps1'**, reveals it's a ASCII file with with very long lines (489), with CRLF line terminators. Running the command **'strings ./challenge.ps1'** reveals an encoded string in base 64, once decoded it says:

```
$decoded =
[System.Convert]::FromBase64String('ZmxhZ3s0NWQyM2MxZjY3ODliYWRjMTIzNDU2Nzg5MD
EyMzQ1Nn0=')
```

```
$flag = [System.Text.Encoding]::UTF8.GetString($decoded)

# Only show flag if specific environment variable is set
if ($env:MAGIC_KEY -eq 'Sup3rS3cr3t!') {
	Write-Output $flag
} else {
	Write-Output "Nice try! But you need the magic key!"
}
```

The $decoded variable is the flag, you could either decode it again, using an online decoder with an UTF-8 charset, or you could try and run the program in windows, using the magic key to get the flag's output.

Math for Me

Flag: flag{h556cdd`=ag.c53664:45569368391gc}

Solution: You will need Ghidra or any software that can decompile the executable. Open up Ghidra and import the executable. There you'll want to let Ghidra analyze the code first before you do anything else. On the toolbar, click Windows > Symbol Tree. From there you can search up the available functions. Look at `main()` first, from there you can see it calls the `check_number()` function. Double click the check_number() function and you should be able to see its contents like so:

```
bool check_number(int param_1)

{
  return (param_1 * 5 + 4) / 2 == 0x34;
}
```

I've highlighted the hexadecimal memory address in red. Hover over it in Ghidra with your mouse, you should see its value of 52. Time to do some maths to get the magic number, after that, run the program and input the number and you should get the flag.