Numbers go brrr

Mystiz

Disclaimer

- The slides are made today (Nov 28, 2022), so it might be confusing.
- Most photos are stolen from https://www.facebook.com/cucatcat/.
  If you don't know the *math*, you can still get use to the *meows* in CUHK.

Numbers go brrr (300 points, 3 solves)

Solved by…

- 🥇 PLUS (*invited team*)
- Project SEKAI (*invited team*)
- Black Banana (*open division*)

Numbers go brrr (300 points, 3 solves)

Challenge Summary.

We are given an attachment called `solve.py`, as given below:

```python
from pwn import *
from z3 import *
from operator import add, mul
from functools import reduce
from rich.progress import track
import itertools

# TODO: change this to the remote service
r = process('./chall')

for _ in track(range(100)):
    r.recvuntil('🔧 '.encode())

    m, s, p, q = map(int, r.recvline().decode().split())

    _s = Solver()
    xs = [Int(f'x_{i}') for i in range(m)]

    subss = [Int(f'ss_{i}') for i in range(m)]
    subps = [Int(f'ps_{i}') for i in range(m)]

    # The base conditions
    for i in range(1, m):
        _s.add(xs[i-1] <= xs[i])
    for i in range(0, m):
```

Numbers go brrr (300 points, 3 solves)

Challenge Summary

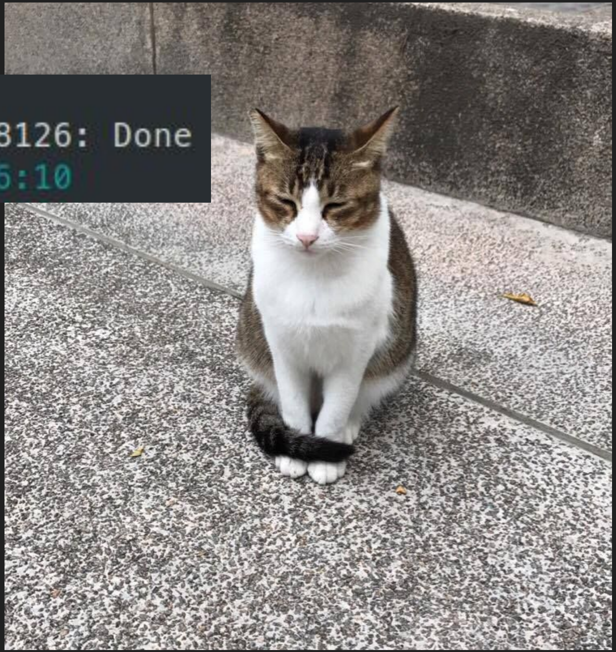
We are given an attac … w:


```
    # The base conditions
    for i in range(1, m):
        _s.add(xs[i-1] <= xs[i])
    for i in range(0, m):
```

Numbers go brrr (300 points, 3 solves)

Since it is the solution script, we can just run it against the remote service and get the flag… And you can get the flag in around six minutes!

```
❯ python3 solve.py
[+] Opening connection to chal.hkcert22.pwnable.hk on port 28126: Done
Working... -                                            3% 0:06:10
```




Thank you for listening!

Numbers go brrr (300 points, 3 solves)

~~Since it is the solution script, we can just run it against the remote service and get the flag…~~ In reality, the solution script is too slow to run.

```
❯ python3 solve.py
[+] Opening connection to chal.hkcert22.pwnable.hk on port 28126: Done
Working...                                                   7% -:--:--
Traceback (most recent call last):
  File "solve.py", line 12, in <module>
    r.recvuntil('🔧 '.encode())
  File "/home/mystiz/.local/lib/python3.8/site-packages/pwnlib/tubes/tube.py", l
ine 333, in recvuntil
    res = self.recv(timeout=self.timeout)
  File "/home/mystiz/.local/lib/python3.8/s
ine 105, in recv
    return self._recv(numb, timeout) or b''
  File "/home/mystiz/.local/lib/python3.8/s
ine 183, in _recv
```

Numbers go brrr (300 points, 3 solves)

Let's get to the real deal – read `solve.py`!

z3 is a package for satisfiability modulo theories (SMT) solver

```python
from pwn import *
from z3 import *
from operator import add, mul
from functools import reduce
from rich.progress import track
import itertools

# TODO: change this to the remote service
r = process('./chall')

for _ in track(range(100)):
    r.recvuntil('🔧 '.encode())

    m, s, p, q = map(int, r.recvline().decode().split())

    _s = Solver()
    xs = [Int(f'x_{i}') for i in range(m)]

    subss = [Int(f'ss_{i}') for i in range(m)]
```

Numbers go brrr (300 points, 3 solves)

```python
from pwn import *
from z3 import *
from operator import add, mul
from functools import reduce
from rich.progress import track
import itertools

# TODO: change this to the remote service
r = process('./chall')

for _ in track(range(100)):
    r.recvuntil('🔧 '.encode())

    m, s, p, q = map(int, r.recvline().decode().split())

    _s = Solver()
    xs = [Int(f'x_{i}') for i in range(m)]

    subss = [Int(f'ss_{i}') for i in range(m)]
```

the solve script connects to the remote service

Numbers go brrr (300 points, 3 solves)

```python
from pwn import *
from z3 import *
from operator import add, mul
from functools import reduce
from rich.progress import track
import itertools

# TODO: change this to the remote service
r = process('./chall')

for _ in track(range(100)):
    r.recvuntil('🔧 '.encode())

    m, s, p, q = map(int, r.recvline().decode().split())

    _s = Solver()
    xs = [Int(f'x_{i}') for i in range(m)]

    subss = [Int(f'ss_{i}') for i in range(m)]
```

there are 100 rounds...

in each round:

Numbers go brrr (300 points, 3 solves)

the program receives 4 variables from remote: m, s, p, q

```python
    m, s, p, q = map(int, r.recvline().decode().split())

    _s = Solver()
    xs = [Int(f'x_{i}') for i in range(m)]

    subss = [Int(f'ss_{i}') for i in range(m)]
    subps = [Int(f'ps_{i}') for i in range(m)]

    # The base conditions
    for i in range(1, m):
        _s.add(xs[i-1] <= xs[i])
    for i in range(0, m):
        _s.add(Not(xs[i] <= 0))
    for i in range(0, m):
        _s.add(xs[i] < q)
    for i, j in itertools.product(range(0, m), repeat=2):
        _s.add(Implies(i != j, xs[i] != xs[j]))
```

Numbers go brrr (300 points, 3 solves)

```
m, s, p, q = map(int, r.recvline().decode().split())

_s = Solver()
xs = [Int(f'x_{i}') for i in range(m)]

subss = [Int(f'ss_{i}') for i in range(m)]
subps = [Int(f'ps_{i}') for i in range(m)]

# The base conditions
for i in range(1, m):
    _s.add(xs[i-1] <= xs[i])
for i in range(0, m):
    _s.add(Not(xs[i] <= 0))
for i in range(0, m):
    _s.add(xs[i] < q)
for i, j in itertools.product(range(0, m), repeat=2):
    _s.add(Implies(i != j, xs[i] != xs[j]))
```

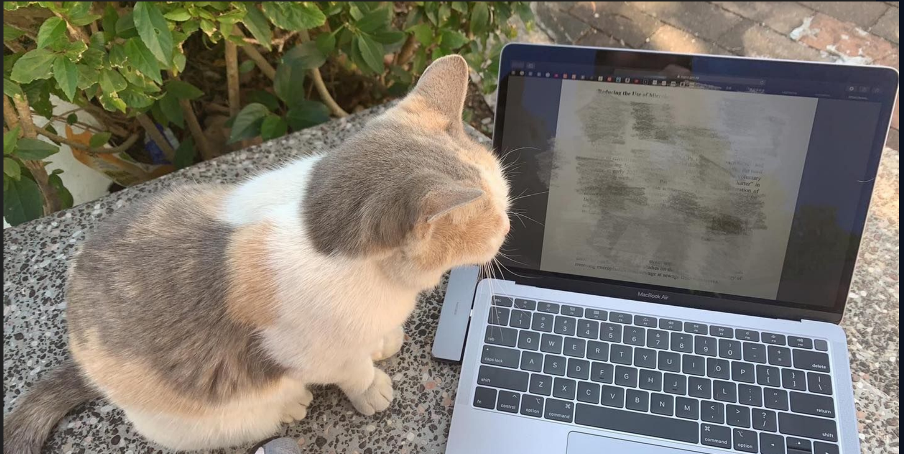
it defines 3m variables:

x_k, s_k, p_k for $k = 0, 1, \ldots, m-1$

Numbers go brrr (300 points, 3 solves)

Conditions (Part 1)

- $x_0 \leq x_1 \leq ... \leq x_{m-1}$

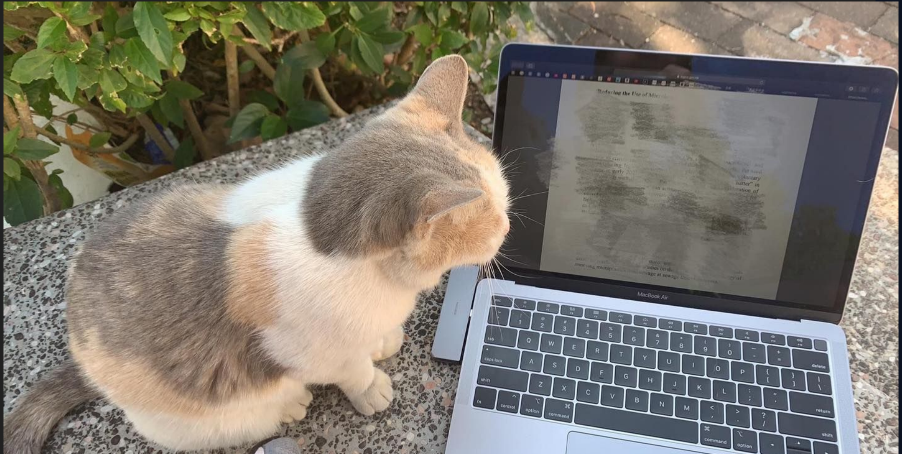


```python
    m, s, p, q = map(int, r.recvline().decode().split())

    _s = Solver()
    xs = [Int(f'x_{i}') for i in range(m)]

    subss = [Int(f'ss_{i}') for i in range(m)]
    subps = [Int(f'ps_{i}') for i in range(m)]

    # The base conditions
    for i in range(1, m):
        _s.add(xs[i-1] <= xs[i])
    for i in range(0, m):
        _s.add(Not(xs[i] <= 0))
    for i in range(0, m):
        _s.add(xs[i] < q)
    for i, j in itertools.product(range(0, m), repeat=2):
        _s.add(Implies(i != j, xs[i] != xs[j]))

    # The "s" and "p" requirements
    _s.add(subss[0] == xs[0])
    _s.add(subps[0] == xs[0])
    for i in range(m-1):
        _s.add(subss[i+1] == subss[i] + (i+2)*xs[i+1])
        _s.add(subps[i+1] == subps[i] * (i+2)*xs[i+1])
    _s.add(subss[m-1] % q == s)
```

Numbers go brrr (300 points, 3 solves)

Conditions (Part 1)

- $x_0 \leq x_1 \leq ... \leq x_{m-1}$
- $x_0 > 0, x_1 > 0, ..., x_{m-1} > 0$

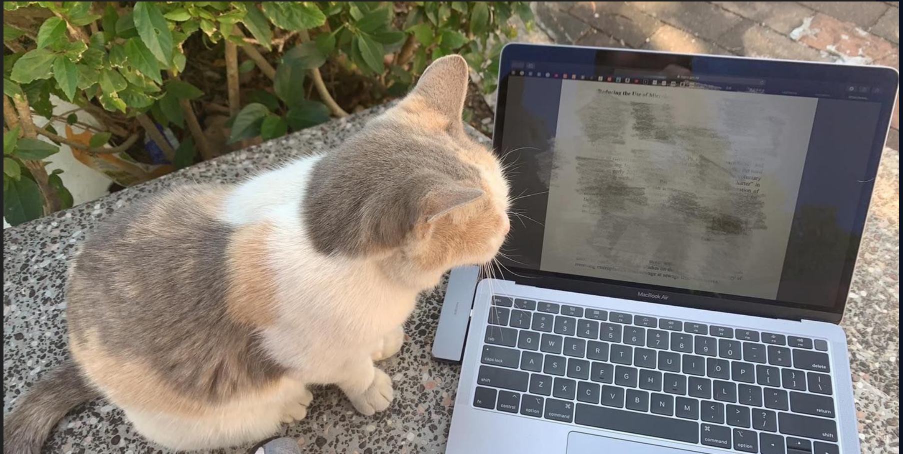


```python
    m, s, p, q = map(int, r.recvline().decode().split())

    _s = Solver()
    xs = [Int(f'x_{i}') for i in range(m)]

    subss = [Int(f'ss_{i}') for i in range(m)]
    subps = [Int(f'ps_{i}') for i in range(m)]

    # The base conditions
    for i in range(1, m):
        _s.add(xs[i-1] <= xs[i])
    for i in range(0, m):
        _s.add(Not(xs[i] <= 0))
    for i in range(0, m):
        _s.add(xs[i] < q)
    for i, j in itertools.product(range(0, m), repeat=2):
        _s.add(Implies(i != j, xs[i] != xs[j]))

    # The "s" and "p" requirements
    _s.add(subss[0] == xs[0])
    _s.add(subps[0] == xs[0])
    for i in range(m-1):
        _s.add(subss[i+1] == subss[i] + (i+2)*xs[i+1])
        _s.add(subps[i+1] == subps[i] * (i+2)*xs[i+1])
    _s.add(subss[m-1] % q == s)
```

Numbers go brrr (300 points, 3 solves)

Conditions (Part 1)

- $x_0 \leq x_1 \leq \ldots \leq x_{m-1}$
- $x_0 > 0, x_1 > 0, \ldots, x_{m-1} > 0$
- $x_0 < q, x_1 < q, \ldots, x_{m-1} < q$

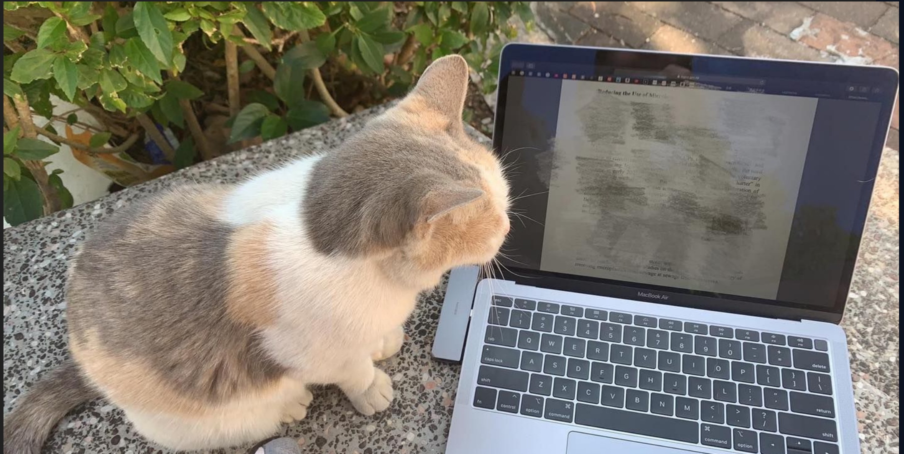


```
m, s, p, q = map(int, r.recvline().decode().split())

_s = Solver()
xs = [Int(f'x_{i}') for i in range(m)]

subss = [Int(f'ss_{i}') for i in range(m)]
subps = [Int(f'ps_{i}') for i in range(m)]

# The base conditions
for i in range(1, m):
    _s.add(xs[i-1] <= xs[i])
for i in range(0, m):
    _s.add(Not(xs[i] <= 0))
for i in range(0, m):
    _s.add(xs[i] < q)
for i, j in itertools.product(range(0, m), repeat=2):
    _s.add(Implies(i != j, xs[i] != xs[j]))

# The "s" and "p" requirements
_s.add(subss[0] == xs[0])
_s.add(subps[0] == xs[0])
for i in range(m-1):
    _s.add(subss[i+1] == subss[i] + (i+2)*xs[i+1])
    _s.add(subps[i+1] == subps[i] * (i+2)*xs[i+1])
_s.add(subss[m-1] % q == s)
```

Numbers go brrr (300 points, 3 solves)

Conditions (Part 1)

- $x_0 \leq x_1 \leq ... \leq x_{m-1}$
- $x_0 > 0, x_1 > 0, ..., x_{m-1} > 0$
- $x_0 < q, x_1 < q, ..., x_{m-1} < q$
- $x_i \neq x_j$ when $i \neq j$



```python
m, s, p, q = map(int, r.recvline().decode().split())

_s = Solver()
xs = [Int(f'x_{i}') for i in range(m)]

subss = [Int(f'ss_{i}') for i in range(m)]
subps = [Int(f'ps_{i}') for i in range(m)]

# The base conditions
for i in range(1, m):
    _s.add(xs[i-1] <= xs[i])
for i in range(0, m):
    _s.add(Not(xs[i] <= 0))
for i in range(0, m):
    _s.add(xs[i] < q)
for i, j in itertools.product(range(0, m), repeat=2):
    _s.add(Implies(i != j, xs[i] != xs[j]))

# The "s" and "p" requirements
_s.add(subss[0] == xs[0])
_s.add(subps[0] == xs[0])
for i in range(m-1):
    _s.add(subss[i+1] == subss[i] + (i+2)*xs[i+1])
    _s.add(subps[i+1] == subps[i] * (i+2)*xs[i+1])
_s.add(subss[m-1] % q == s)
```

Numbers go brrr (300 points, 3 solves)

Conditions (Part 1)

- $x_0 \leq x_1 \leq ... \leq x_{m-1}$
- $x_0 > 0, x_1 > 0, ..., x_{m-1} > 0$
- $x_0 < q, x_1 < q, ..., x_{m-1} < q$
- $x_i \neq x_j$ when $i \neq j$

Numbers go brrr (300 points, 3 solves)

Conditions (Part 1)

- $\boxed{x_0 \leq x_1 \leq ... \leq x_{m-1}}$
- $\boxed{x_0 > 0, x_1 > 0, ..., x_{m-1} > 0}$
- $x_0 < q, x_1 < q, ..., x_{m-1} < q$
- $x_i \neq x_j$ when $i \neq j$

Numbers go brrr (300 points, 3 solves)

Conditions (Part 1)

- $0 < x_0 \leq x_1 \leq x_2 \leq \ldots \leq x_{m-1}$
- $x_0 < q, x_1 < q, \ldots, x_{m-1} < q$
- $x_i \neq x_j$ when $i \neq j$

Numbers go brrr (300 points, 3 solves)

Conditions (Part 1)

- $\boxed{0 < x_0 \le x_1 \le x_2 \le \ldots \le x_{m-1}}$
- $\boxed{x_0 < q, x_1 < q, \ldots, x_{m-1} < q}$
- $x_i \neq x_j$ when $i \neq j$

Numbers go brrr (300 points, 3 solves)

Conditions (Part 1)

- $0 < x_0 \leq x_1 \leq x_2 \leq ... \leq x_{m-1} < q$
- $x_i \neq x_j$ when $i \neq j$

Numbers go brrr (300 points, 3 solves)

Conditions (Part 1)

- $0 < x_0 \leq x_1 \leq x_2 \leq ... \leq x_{m-1} < q$
- $x_i \neq x_j$ when $i \neq j$

Numbers go brrr (300 points, 3 solves)

Conditions (Part 1)

- $0 < x_0 < x_1 < x_2 < ... < x_{m-1} < q$

Numbers go brrr (300 points, 3 solves)

Conditions (Part 2)

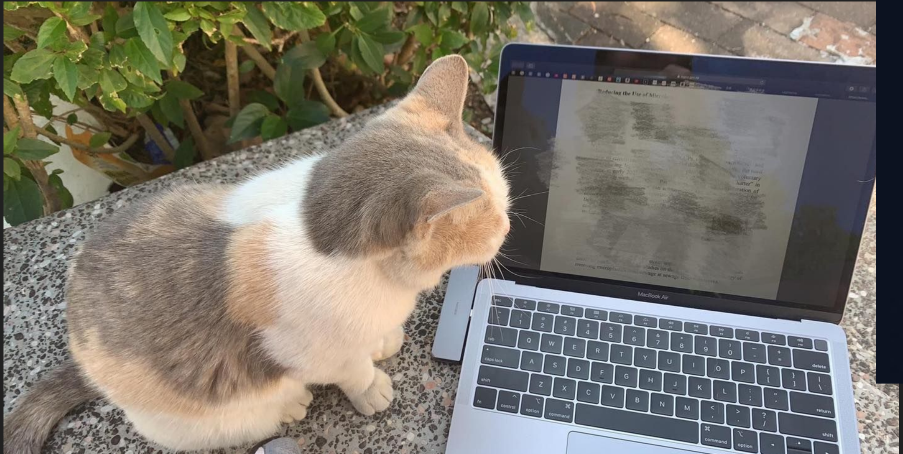
- $s_0 = x_0, p_0 = x_0$

```python
# The "s" and "p" requirements
_s.add(subss[0] == xs[0])
_s.add(subps[0] == xs[0])
for i in range(m-1):
    _s.add(subss[i+1] == subss[i] + (i+2)*xs[i+1])
    _s.add(subps[i+1] == subps[i] * (i+2)*xs[i+1])
_s.add(subss[m-1] % q == s)
_s.add(subps[m-1] % q == p)

assert _s.check() == sat
md = _s.model()
x0s = [md.evaluate(xs[i]) for i in range(m)]
r.sendlineafter('🥺 '.encode(), ' '.join(map(str, x0s
```

Numbers go brrr (300 points, 3 solves)

Conditions (Part 2)

- $s_0 = x_0, p_0 = x_0$
- $s_{i+1} = s_i + (i+2) \cdot x_{i+1}$ for $i = 0, 1, \ldots, m-2$
- $p_{i+1} = p_i \cdot (i+2) \cdot x_{i+1}$ for $i = 0, 1, \ldots, m-2$

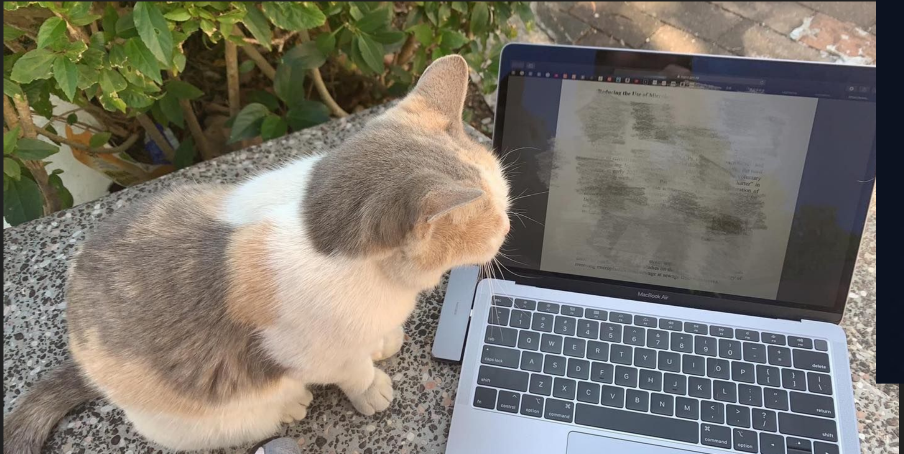


```python
# The "s" and "p" requirements
_s.add(subss[0] == xs[0])
_s.add(subps[0] == xs[0])
for i in range(m-1):
    _s.add(subss[i+1] == subss[i] + (i+2)*xs[i+1])
    _s.add(subps[i+1] == subps[i] * (i+2)*xs[i+1])
_s.add(subss[m-1] % q == s)
_s.add(subps[m-1] % q == p)

assert _s.check() == sat
md = _s.model()
x0s = [md.evaluate(xs[i]) for i in range(m)]
r.sendlineafter('🥺 '.encode(), ' '.join(map(str, x0s
```

Numbers go brrr (300 points, 3 solves)

Conditions (Part 2)

- $s_0 = x_0, p_0 = x_0$
- $s_{i+1} = s_i + (i+2) \cdot x_{i+1}$ for $i = 0, 1, ..., m-2$
- $p_{i+1} = p_i \cdot (i+2) \cdot x_{i+1}$ for $i = 0, 1, ..., m-2$
- $s_{m-1} \bmod q = s, p_{m-1} \bmod q = p$



```python
# The "s" and "p" requirements
_s.add(subss[0] == xs[0])
_s.add(subps[0] == xs[0])
for i in range(m-1):
    _s.add(subss[i+1] == subss[i] + (i+2)*xs[i+1])
    _s.add(subps[i+1] == subps[i] * (i+2)*xs[i+1])
_s.add(subss[m-1] % q == s)
_s.add(subps[m-1] % q == p)

assert _s.check() == sat
md = _s.model()
x0s = [md.evaluate(xs[i]) for i in range(m)]
r.sendlineafter('🥺 '.encode(), ' '.join(map(str, x0s
```

Numbers go brrr (300 points, 3 solves)

Conditions (Part 2)

- $\sum_{k=1}^{m} k \cdot x_{k-1} \equiv s \pmod{q}$
- $\prod_{k=1}^{m} k \cdot x_{k-1} \equiv p \pmod{q}$

Numbers go brrr (300 points, 3 solves)

Objective.

Find $(x_0, x_1, ..., x_{m-1})$ such that $0 < x_0 < x_1 < x_2 < ... < x_{m-1} < q$,

$$\sum_{k=1}^{m} k \cdot x_{k-1} \equiv s \pmod q \quad \text{and} \quad \prod_{k=1}^{m} k \cdot x_{k-1} \equiv p \pmod q$$

Numbers go brrr (300 points, 3 solves)

How?

The system has 2 equations with *m* unknowns

⇒ The system is probably underdetermined!

We can assume $x_0 = 1, x_1 = 2, ..., x_{m-3} = m - 2$ for simplicity.

There are two unknowns left, $m - 2 < x_{m-2} < x_{m-1} < q$.

Numbers go brrr (300 points, 3 solves)

Pointers.

After the reduction, it essentially asks us to solve a quadratic equation under a prime modulo.

If you want the math behind… Go take MATH3080 (*Number Theory*), or read from "Quadratic equation solutions modulo prime p" on *StackExchange*.

Numbers go brrr (300 points, 3 solves)

Pointers.

If you want to skip the math part… Use *sagemath*: In particular,

1. Define a polynomial ring by `PolynomialRing(Zmod(p))`,
2. Define the quadratic equation `p`, and
3. Solve it with `p.roots()`.

Numbers go brrr (300 points, 3 solves)

Final words.

An official writeup on this challenge (and the rest I made) will be available on mystiz.hk… Likely later this week.

Some advertisements.

[blackb6a] Mystiz 2022/11/14 00:10
btw, 如果有興趣想同我哋一齊玩 ctf 嘅話可以 dm 我哋
我哋會遲啲物色啲新手向嘅 ctf 同大家一齊玩

Mystiz ✔✔ 今天 02:14
Let's kick-start our first project idea in the channel: Minecraft [redacted] Ultra-Hardcore PVP rounds
We'll be regularly hosting (when the infrastructure is ready) UHC rounds which all of us compete and discuss how to make improvements.
At the same time, the hosting server will be making patches to make cheating harder in each round.
This involves game reverse engineering, so it is probably exciting!


Thank you for listening!
(For real)