nullhat 2025: notThat

@elesquina

The jail reads one line of Python code and enforces a maximum length of 50 characters. It also rejects some substrings such as `import`, and it blocks characters like underscores and quotes. After the filter, it sets `sys.stdout` to a string, which makes normal output unusable, but exceptions still print to stderr. The process ends after one call to `chall()`, but inside the executed code we can call `chall()` again, so we can send many short inputs.

The useful observation is that global state survives between these nested calls. The module `string` is already imported at the top level, so it is a perfect scratchpad because we can attach attributes to it and reuse them later. This lets us build forbidden tokens without typing forbidden characters, for example `chr(95)` gives the underscore, and concatenation can create strings like `__builtins__` and `__import__`.

I split the exploit into multiple < 50-character inputs. Each input stores one intermediate value in `string.*`, then immediately calls `chall()` to get the next input evaluated in the same global context. At the end I use `glob` to find the randomized flag filename `flag-*` and I leak it by raising an assertion with the file contents as the message, because the traceback is printed even when stdout is broken.

The exact sequence of inputs is the following, with each line entered as one separate prompt:

```
string.u=chr(95);chall()
string.d=string.u*2;chall()
string.k=string.d+'builtins'+string.d;chall()
string.B=globals().get(string.k);chall()
string.m=string.d+('im'+'port')+string.d;chall()
string.I=getattr(string.B,string.m);chall()
string.G=string.I('glob');chall()
string.F=next(string.G.iglob('flag-*'));chall()
assert 0,open(string.F).read()
```