Highly concurrent Python for brute force and discovery

Marcus Hodges
Senior Software Security Engineer
Security Innovation

Goals

- Introduce concurrent programming techniques in python
- Teach how to build really fast tools for brute force and discovery
- Show a library I created to help simplify the mess

Disclaimer

- A variety of fast tools and frameworks already exist
- Varying degrees of complexity (Deferred Semaphores)
- Firewalls and sys admins are going to hate you

Motivations

Forceful Browsing

CITIBANK

How Hackers Stole 200,000+ Citi Accounts Just By Changing Numbers In The URL

By Ben Popken on June 14, 2011 3:00 PM

(Sebastian Anthony)

Details have emerged has to how hackers were able to steal over 200,000 Citi customer accounts, including names, credit card numbers, mailing addresses and email addresses. It turns out quite easily, in fact. All they had to do was log in as a customer and change around a few numbers into the browser's URL bar, NYT reports. Facepalm.

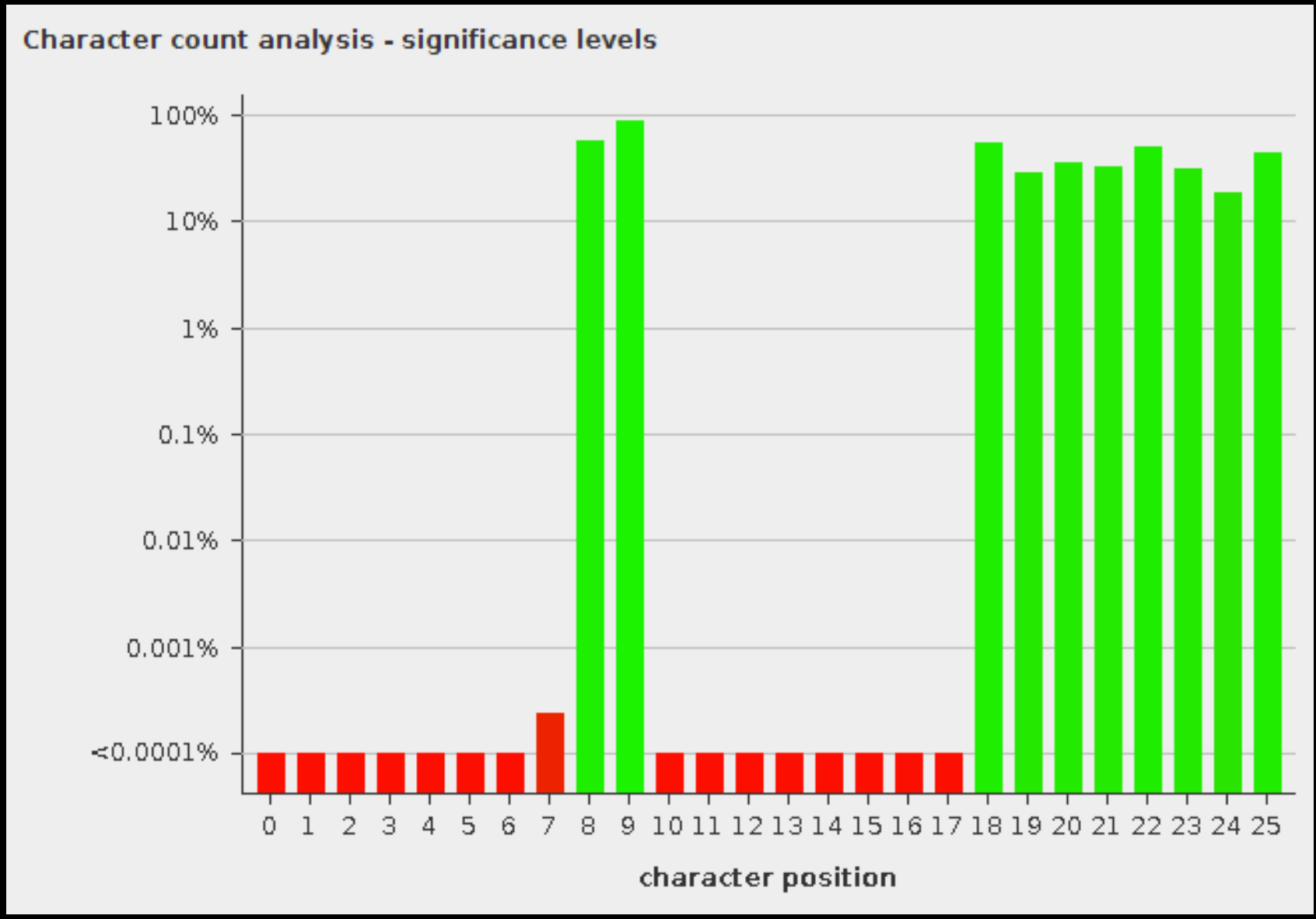
Basically after you logged into your account as a Citi customer, the URL contained a code identifying your account. All you had to do was change around the numbers and boom, you were in someone else's account.

So if the URL was something like citibank.com/user/12345, all you had to do was change it to citibank.com/user/123456 and you had access to all of their account information.

The hackers then used a simple script that automatically scraped all the account information, saved it, and then changed the numbers in the URL and repeated the process. Hundreds of thousands of times.

As someone who has been on the internet for a few years, this is a dead simple and common hack and Citi should have seen it and prevented against it. Seriously, this is kindergarten level stuff. Really, really stupid.

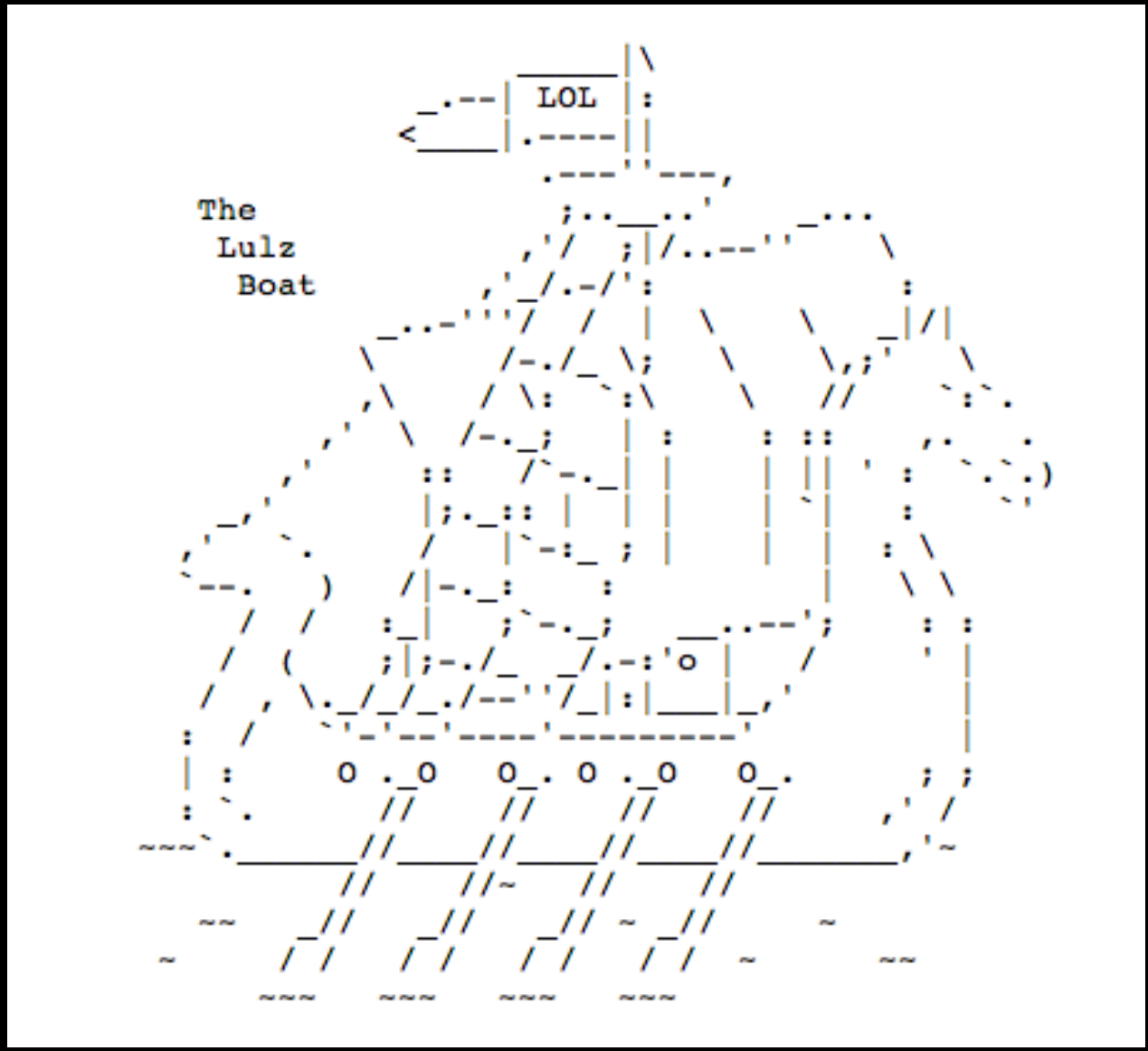
Token Entropy



Key-space

- Key-space is different for every vuln
- So you have to write or customize a tool to exploit every vuln

Dictionary Attacks



Discovery

Scanning the entire Internet for...

- default creds
- 0days
- SQLi

Performance

- My webserver can handle thousands of concurrent requests (C10K)
- Why cant my custom pentesting tools send thousands of concurrent requests

They can, here's how...

Concurrency

- Concurrent programming is concerned with the efficiency of non-deterministic programs
- aka multitasking
- Concurrency != Parrallism
- The entire stack has to be non-blocking to achieve concurrency

your code
network libraries
kernel

Need to make every later non-blocking...

Non-blocking sockets

```
from socket import socket
sock = socket()
sock.setblocking(0)
sock.connect((host,port))
sock.send(data)
```

Non-blocking sockets

- Ok, our socket doesnt block anymore...
- How do we know when it's
  - done connecting?
  - ready to read?
  - ready to write?

- Down the rabbit hole...

Into the Kernel

Epoll

- A Linux Kernel API for querying file handle status
- Epoll reports which files are ready for which events

Epoll

```
import select
epoll = select.epoll()
epoll.register(sock.fileno(), select.EPOLLOUT)
for fileno, event in epoll.poll(1):
    # read, write, or close for fileno
```

Consider a simple protocol

connect()
read(10)
write('helo')
read(10)
write('mail from')
read(10)
write('rcpt to')

The Catch

- Suppose epoll says fileno 0x42 is ready to write
- Which write() is fileno 0x42 ready for?

Keeping application state

- We need a way to preserve the application state associated with a file handle

Possible options:

- Threads
- Callbacks
- Or ...

Coroutines

- A relatively obscure language feature
- Originated as an assembly language technique in 1963
- Feature of Python since 2.5 (PEP-342)

Coroutines

- A coroutine is a generalization of a function
- can send values to the calling code
- can suspend execution and preserve function state
- calling code can inject values into local variables (not parameters)

Python Coroutines

- In Python, all of this is done with the “**yield**” statement
- Similar syntax to Generators but conceptually distinct

Python Coroutines

```
def foo():
    x = yield "What makes you think she's a witch?"
    print x
    y = yield "A newt?"
    print y

f = foo()
print f.send(None)
print f.send("She turned me into a newt!")
print f.send("I got better.")
```

Python Coroutines

```
In [1]: f = foo()
In [2]: print f.send(None)
        What makes you think she's a witch?
In [3]: print f.send("She turned me into a newt!")
        She turned me into a newt!
        A newt?
In [4]: print f.send("I got better.")
        I got better.
```

The Big Idea

- Coroutines are an alternative to threads
- yield can be used as a system call for Python

yield 0x80 <==> int 0x80

Killer Joke

"Wenn ist das Nunstück git und
Slotermeyer? Ja! Beiherhund das
Oder die Flipperwaldt gersput!"

Putting everything together

An exercise for the reader...

- Create a new networking library that
- Uses yield as a Python systemcall
- To suspend and resume coroutines
- From within an epoll event loop

For-loop

```
import socket

def pwn(count):
    for x in range(count):
        msg = "GET /account/%s HTTP/1.1 ..." % x
        sock.connet((host, port))
        sock.write(msg)
        response = sock.read(1024)
        sock.close()

pwn(count)
```

Blackmamba

```
from blackmamba import *

def pwn(x):
    msg = "GET /account/%s HTTP/1.1 ..." % x
    yield connect(host, port)
    yield write(msg)
    response = yield read(1024)
    yield close()

def gen(count):
    for i in range(count):
        yield pwn(i)

run(gen(count))
```

Blackmamba

1000 connections completed in 3.501 seconds
(285.638 per sec)

Enjoy :)

Thank you

Questions?

Marcus Hodges
0xmeta@gmail.com
rootfoo.org