Proxyfeed - An Open Proxy RSS Auditing Tool

Michael J. Rossi

March 11, 2015

Contents

1 Introduction **4**
- 1.1 Our Purpose . . . 4
- 1.2 The Various Proxy Feeds . . . 4
- 1.3 Installation . . . 4
- 1.4 Literate Programming . . . 5

2 Test Framework **5**

3 Main Code Outline **5**
- 3.1 Broad Overview . . . 5
- 3.2 The Outline . . . 6

4 Extracting XML Feed Data **6**
- 4.1 Mapping XML to Python . . . 6
- 4.2 Extracting the Data . . . 7

5 Puling the RSS Feeds **8**

6 Filtering Country Specific Proxy Data **9**

7 Whois Lookups **10**
- 7.1 Getting Whois Descriptions . . . 10
- 7.2 Doing Mass Whois Lookups . . . 11

8 Removing Duplicate Entries **12**
- 8.1 An IP Predicate . . . 12
- 8.2 Actually Removing Duplicates . . . 13

9 Printing Results **14**

10 The Main Program **16**
- 10.1 Pulling All Proxies . . . 16
- 10.2 Pull Country Specific Proxies . . . 17
- 10.3 The __main__ Program . . . 17

11 To Do **19**

12 Updates **19**

13 Loose Ends **20**
- 13.1 Literate Code Comment . . . 20
- 13.2 License . . . 20
- 13.3 Test XML File . . . 21

14 Makefile **23**

15 Final Python Source Files **24**
- 15.1 proxyfeed.py . . . 24
- 15.2 tests.py . . . 28

16 Raw Nuweb File **31**

17 Index . . . **48**
17.1 Files . . . 48
17.2 Identifiers . . . 48
17.3 Fragments . . . 48

1 Introduction

1.1 Our Purpose

There is no darkness, nor shadow of death, where the workers of iniquity may hide themselves. – Job 34:22

Malicious hackers love open proxies. They love them so much, that they share them in various ways throughout the web so that other hackers and script-kiddes can use them to stay hidden.[1] One of the best places we've seen for up to date lists of open proxies are the various RSS feeds available online. The following program attempts to tap into the feeds so that we can better locate any open proxies being publicly advertised online. To do so, we'll write a small Python program to pull these various feeds, parse them, and then process them in various ways.

1.2 The Various Proxy Feeds

There are various sites out dedicated to open proxies which provide RSS feeds of the latest victims. The site proxyrss.com not only collects these various feeds into a central location, but it also provides an XML namespace specification to normalize the feeds. At this time, we have found the following feeds to be live, active, and conforming.

Proxy RSS Specification: http://www.proxyrss.com/specification.html.

1. http://www.proxz.com/proxylists.xml
2. http://www.freeproxylists.com/rss
3. http://www.xroxy.com/proxyrss.xml

Of these feeds, xroxy.com is the only one not fully conforming to the spec. Instead of the correct spec XML namespace, they instead have the namespace listed as http://www.proxyrss.com/content. At this time this link is 404'd, so I'm assuming that this was the old location and that xroxy.com has not updated their feed data. We bring this up now because this anomaly will come up when we begin to parse the feeds.

1.3 Installation

The code is written in Python 3. We're using the following two third party libraries (i.e. not included by default in the Python distribution).

- ipwhois
- termcolor *(for 31337 output)*

If you don't have these installed on our system, you'll need to install them using pip. For example:

```
$ pip install ipwhois
$ pip install termcolor
```

Otherwise, you just run it like any other Python program. For example:

```
$ python proxyfeed.py —h
```

Please also note that this is being written and tested on Kali Linux using the Anaconda Python distribution, specifically, Python 3.4.1, Anaconda 2.1.0 (64-bit). I will also try to test it on Windows 7, but it's an after thought.

[1] Assuming, of course, that the black hats didn't setup a malicious proxy to catch the script-kiddies. . . oh the tangled webs we weave.

1.4 Literate Programming

This is a literate program written using **nuweb** with Python. Literate programming is a discipline whereby the author writes the documentation and code as any other piece of prose that explains the entire program in whatever order or manner is best for the problem being tackled. By adhering to this discipline, the documentation and the code will not go out of sync. Moreover, it will be easy for anyone (including the author sometime in the future) to learn how the program works and why the author did what he did. For more information, please see **literateprogramming.com**.

Please note that for the Python code, we slightly pretty print the code in a mild CWEB style to make it easier on the eyes while still conforming to the "Zen of Python".[2] But we don't alter the syntax any like substituting == for ≡ or ! = with ≠. If for some bizarre reason you have trouble reading the pretty printed code, please consult the raw code later in this PDF.

The source code consists of only two files. A *Makefile* and the *web* file *proxyfeed.w* which generates everything else, including this PDF, proxyfeed.py, tests.py (unit tests), and even a test XML feed. This PDF documentation includes not only the "weaved" literate text, but also the "tangled" or produced source files along with the full nuweb "web" source. In other words, this PDF contains everything produced by *nuweb* in it–including the separate *Makefile*.

2 Test Framework

Along with our literate program, we also adhere to the discipline of testing. In this spirit, we'll setup our test framework which we'll fill in as we go. Although it won't be apparent from this, we actually write our tests first and then the functions to pass the tests. But in our literate text, we'll put the tests afterwards so that if you're just interested in learning what, how, and why the program does what it does, you can skip the tests. If you're not familiar with the "Testing Goat," and what it means, please visit **obeythetestinggoat.com**.

The test framework will be in it's own file. The only thing of note at this time is that *proxyfeed* will be our main program, so we'll import it now.

```
"tests.py" 5≡
    #!/usr/bin/env python3
    ⟨ Literate code header comment 20 ⟩
    import unittest
    from xml.etree.ElementTree import parse
    from proxyfeed import *

    class ProxyFeedTest(unittest.TestCase):
        ⟨ Obey the testing goat! 8, ... ⟩

    if __name__ == '__main__':
        unittest.main(warnings='ignore',failfast=True)
    ◇
```

Defines: `ProxyFeedTest` Never used, `setUp` Never used, `tearDown` Never used.

3 Main Code Outline

3.1 Broad Overview

As mentioned, the program will simply pull the RSS feed, parse the XML document, and then process the data. From the point of view of security, when it comes to processing the data, we'll want to pull the whois information from the IP addresses in order to determine who the proxy belongs to. We can, of course, do more processing later, but the main thing we're after is the ability to pull the data and see a list of proxies along with the whois description of the IP.

[2] Try running "import this" in your Python interpreter if you don't know what this means.

3.2 The Outline

Here is the outline which we'll fill in as we go along with our imports.

```
"proxyfeed.py" 6≡
    #!/usr/bin/env python3
    ⟨ Literate code header comment 20 ⟩
    from urllib.request import urlopen, Request
    from xml.etree.ElementTree import parse
    from copy import copy, deepcopy
    import datetime

    # 3rd party  libraries
    from ipwhois import IPWhois
    from termcolor import cprint
    ⟨ Main functions 7, ... ⟩
    ◇
```

4 Extracting XML Feed Data

4.1 Mapping XML to Python

According to the Proxy RSS Specification, we'll always have the following elements in a conforming proxy feed:

```
<prx:proxy>
<prx:ip>
<prx:port>
```

There are various optional elements, but the only two we're interested in at this time is the "country" of origin and the "timestamp" when the proxy was last verified. The timestamp is as follows.

```
<prx:check_timestamp>
```

As for countries, if we're trying to audit our organization in the US, we obviously don't care about the open proxies in North Korea[3], so the country will come in handy later. In the spec, there are two optional values for the country as follows:

```
<prx:country>
<prx:country_code>
```

The *country* is an unofficial string description while the *country_code* should satisfy ISO 3166-1 Alpha-2. In our experience at this time, only one feed uses the *country_code*'s while the rest use *country*.

We'll store all of this data into a Python dictionary which will have five string values and look like this.

```
proxy = {'ip':'192.168.1.1',
         'port':'3128',
         'country':'US',
         'timestamp':'01/01/1970',
         'whois':'Unknown'}
```

Please note that the country, timestamp, and whois do not have any standard format.

[3]Unless we're just so very ronery.

4.2 Extracting the Data

For our actual parse function, we'll take an already parsed XML document and the url to the XML namespace. Since there's problems with feeds using non-standard specs, the document we pass will need to be a *deepcopy*. Otherwise, we'll simply do our normal XML parsing with namespaces and add the results to a dictionary which we'll then add to a list of proxies.

The only wrinkle in the code is the problems with the "countries" and setting default fault tolerant values. We'll check for both types of "countries" and add a default value if for some reason we don't find any. You (apparently[4]) can't make your own *xml.etree.ElementTree.Element* and set the *text* value, so we'll make a copy of a mandatory XML field. Since the IP is mandatory, we'll copy it and then change the *text* value. We'll do the same for the timestamp.

Finally, it might be worth noting the use of *cprint* from the *termcolor* library which allows us to print in riveting ASCII colors.

⟨ *Main functions* 7 ⟩ ≡

```
def extract_proxy_data(doc, namespace_spec_url):
    """Extracts the data from an already parsed (also use deepcopy) XML doc. Returns a list of
dictionaries of five strings containing the 'ip', 'port', 'country', 'timestamp', and 'whois'."""
    namespaces = {'prx': namespace_spec_url}
    proxies = []
    try:
        for i in doc.iterfind('channel/item/prx:proxy', namespaces):
            ip = i.find('prx:ip', namespaces)
            port = i.find('prx:port', namespaces)
            timestamp = i.find('prx:check_timestamp', namespaces)
            if timestamp == None:  # make fault tolerant
                timestamp = copy(ip)
                timestamp.text = 'Unknown'
            country = i.find('prx:country', namespaces)
            if country == None:
                country = i.find('prx:country_code', namespaces)
            if country == None:
                country = copy(ip)
                country.text = 'Unknown'
            proxy = {'ip':ip.text,
                     'port':port.text,
                     'country':country.text,
                     'timestamp':timestamp.text,
                     'whois':'Unknown'}
            proxies.append(proxy)
    except Exception as e:
        cprint('[-] Error processing XML feed %s!' % doc, 'red')
        cprint('[-] Error: %s' % e, 'yellow')
        quit()
    return proxies
◇
```

Fragment defined by 7, 9a, 10a, 11a, 12a, 13ac, 14b, 15b, 16b, 17ab.
Fragment referenced in 6.
Defines: `extract_proxy_data` 8, 9a, 10b, 13b, 14a, 16a.

To test this, we'll snip down a real feed and use it in our tests. The XML file named *testfeed.xml* is included at the end of this document if you'd like to examine it. Otherwise, we'll just parse the file and check the results.

[4]Though I could be wrong here.

⟨ *Obey the testing goat!* 8 ⟩ ≡

```
def test_extract_proxy_data ( self ):
    with open('testfeed.xml', 'rt') as f:
        doc = parse(f)
        spec = 'http://www.proxyrss.com/specification.html'
        results = extract_proxy_data(doc, spec)
        self.assertEqual( results [0][ 'ip'], '183.223.173.237')
        self.assertEqual( results [0][ 'port'], '8123')
        self.assertEqual( results [0][ 'country'], 'China')
        self.assertEqual( results [0][ 'timestamp'], '03/6␣20:23:47')

        self.assertEqual( results [1][ 'ip'], '183.221.208.44')
        self.assertEqual( results [1][ 'port'], '8123')
        self.assertEqual( results [1][ 'country'], 'CN')
        self.assertEqual( results [1][ 'timestamp'], '03/6␣20:21:37')

        self.assertEqual( results [2][ 'ip'], '183.221.160.12')
        self.assertEqual( results [2][ 'port'], '8123')
        self.assertEqual( results [2][ 'country'], 'Unknown')
        self.assertEqual( results [2][ 'timestamp'], 'Unknown')
◇
```

Fragment defined by 8, 9b, 10b, 11b, 12b, 13b, 14a, 15a, 16a.
Fragment referenced in 5.
Defines: `test_extract_proxy_data` Never used.
Uses: `extract_proxy_data` 7.

5 Puling the RSS Feeds

Now that we can parse the feed, we'll pull the feed and use *extract_proxy_data* to do the dirty work. Some of the proxy feed sites apparently don't like Python (or curl and wget for that matter), so we'll need to change the "User-Agent" string so these sites think we're a "normal web browser" (though a highly generic and tight-lipped one). Otherwise, we simply pull the data via *Request*, parse it, make a deepcopy, and pass it to *extract_proxy_data*. If *proxies* comes back empty, we'll assume it's because it doesn't have the right XML namespace, and try again with the *content* URL.

⟨ *Main functions* 9a ⟩ ≡

```
def get_proxy_feed(url, verbose=False, agent='Mozilla/5.0'):
    """Given a url string, pull a proxy RSS feed conforming to the proxyrss.com specification and
return a list of proxy dictionaries. Some proxy feeds apparently don't like Python, so we'll use a
different user-agent."""
    request = Request(url, headers={'User-Agent': agent})
    try:
        u = urlopen(request)
    except Exception as e:
        cprint('[-] Error fetching feed %s!' % url, 'red')
        cprint('[-] Error: %s' % e, 'yellow')
        quit()
    doc = parse(u)
    spec = 'http://www.proxyrss.com/specification.html'
    proxies = extract_proxy_data(deepcopy(doc), spec)
    if len(proxies) == 0:  # if empty list, try the other XML spec
        spec = 'http://www.proxyrss.com/content'
        proxies = extract_proxy_data(deepcopy(doc), spec)
    if verbose:
        cprint('[+] Found %s total proxies.' % len(proxies), 'green')
    return proxies
◇
```

Fragment defined by 7, 9a, 10a, 11a, 12a, 13ac, 14b, 15b, 16b, 17ab.
Fragment referenced in 6.
Defines: `get_proxy_feed` 9b, 10a, 16b, 17a.
Uses: `extract_proxy_data` 7, `feeds` 17b.

Since we've already tested parsing, we'll just write a simple test to see that make sure we're pulling correctly by just checking that we get some results from a feed.

⟨ *Obey the testing goat!* 9b ⟩ ≡

```
def test_get_proxy_feed(self):
    feed = 'http://www.proxz.com/proxylists.xml'
    results = get_proxy_feed(feed)
    self.assertTrue(results[0]['ip'])
    self.assertTrue(results[0]['port'])
    self.assertTrue(results[0]['country'])
    self.assertTrue(results[0]['timestamp'])
    self.assertTrue(results[0]['country'])
◇
```

Fragment defined by 8, 9b, 10b, 11b, 12b, 13b, 14a, 15a, 16a.
Fragment referenced in 5.
Uses: `get_proxy_feed` 9a.

6 Filtering Country Specific Proxy Data

Since we're in the US and interested in auditing US entities, we'd like to be able to filter our proxy data down to just US proxies. But while we're writing a function to filter US only proxies, we can make it even more general by allowing us to filter by any country. This will require us to know both the appropriate *country_code* and *country* format. Otherwise, this is just a simple filtering function.

⟨ *Main functions* 10a ⟩ ≡

```
def filter_by_country ( proxy_list , country_code, country, verbose=False):
    """Remove any non−US entries from a proxy list generated by get_proxy_feed."""
    proxies = []
    for i in proxy_list :
        if i['country'].endswith(country_code) or i['country'].endswith(country):
            proxies.append(i)
    if verbose:
        cprint('[+]␣Found␣%s␣%s␣proxies.' %
               (len(proxies), country_code), 'green')
    return proxies
```
◇

Fragment defined by 7, 9a, 10a, 11a, 12a, 13ac, 14b, 15b, 16b, 17ab.
Fragment referenced in 6.
Defines: `filter_by_country` 10b, 17a.
Uses: `get_proxy_feed` 9a.

To test, we'll use our test file and filter out any non-Chinese proxies. Our test file has two Chinese entries in it, so this is straightforward. We'll then repeat the test for US proxies, which also has two entries.

⟨ *Obey the testing goat!* 10b ⟩ ≡

```
def test_filter_by_country ( self ):
    with open('testfeed.xml', 'rt') as f:
        doc = parse(f)
        spec = 'http://www.proxyrss.com/specification.html'
        results = extract_proxy_data(doc, spec)
        results = filter_by_country ( results , 'CN', 'China')
        self.assertEqual(len( results ), 2)
        results = extract_proxy_data(doc, spec)
        results = filter_by_country ( results , 'US', 'United␣States')
        self.assertEqual(len( results ), 2)
```
◇

Fragment defined by 8, 9b, 10b, 11b, 12b, 13b, 14a, 15a, 16a.
Fragment referenced in 5.
Uses: `extract_proxy_data` 7, `filter_by_country` 10a.

7 Whois Lookups

At this point, we're already capable of pulling and parsing open proxy feeds. But an IP and country information isn't sufficient to locate proxies associated with a specific organization. So we'll want to pull the whois information from the IP address. But rather then digging through a huge whois dump, we'll just get the "description" data for the IP. Luckily, Python already has the ipwhois library which can do this for us without having to roll our own.

7.1 Getting Whois Descriptions

We'll first write a little function to return the whois "descrption" field for an IP address.

⟨ *Main functions* 11a ⟩ ≡

```
def get_whois_description(ip):
    """Get's the 'description' field from a whois entry given an IP string."""
    obj = IPWhois(ip)
    result = obj.lookup()
    return result['nets'][0]['description']
◇
```

Fragment defined by 7, 9a, 10a, 11a, 12a, 13ac, 14b, 15b, 16b, 17ab.
Fragment referenced in 6.
Defines: `get_whois_description` 11b, 12a.

The test is pretty obvious. Please note that this IP was chosen at random and it just happened to be MS.

⟨ *Obey the testing goat!* 11b ⟩ ≡

```
def test_get_whois_description(self):
    results = get_whois_description('104.41.162.113')
    self.assertEqual(results, 'Microsoft␣Corporation')
◇
```

Fragment defined by 8, 9b, 10b, 11b, 12b, 13b, 14a, 15a, 16a.
Fragment referenced in 5.
Uses: `get_whois_description` 11a.

7.2 Doing Mass Whois Lookups

We now want a function to do our lookups on every IP in our proxy list. Our code will simply run through all the IPs, pull the whois description, and add them to the dictionary. In our tests, some organizations (I'm looking at you China!) use really long "descriptions" that span multiple lines. So we'll strip out any newline characters so the description is just a single line.

Note: In the future, I'd like to thread this to speed it up since we're at the mercy of whois servers.

⟨ *Main functions* 12a ⟩ ≡

```
def lookup_whois(proxies, verbose=True):
    """Performs a whois lookup for the 'description' returning a new list with the 'whois' added
to the dictionaries."""
    results = []
    count = 0
    for proxy in proxies:
        try:
            whois = get_whois_description(proxy['ip'])
            whois = whois.replace("\n", " ")  # strip newlines
            whois = whois.replace("\r", " ")
            proxy['whois'] = whois
            results.append(proxy)
            if verbose:
                count += 1
                if count % 5 == 0:
                    cprint('[+] Completed %s look ups.' % count,
                           'green')
        except Exception as e:
            cprint('[-] Error resolving whois for %s!' % proxy, 'red')
            cprint('[-] Error: %s' % e, 'yellow')
            cprint('[-] Skipping entry.', 'red')
            continue
    return results
```
◇

Fragment defined by 7, 9a, 10a, 11a, 12a, 13ac, 14b, 15b, 16b, 17ab.
Fragment referenced in 6.
Defines: `lookup_whois` 12b, 16b, 17a.
Uses: `get_whois_description` 11a.

To test, we'll manually build an entry so we don't have to do multiple lookups as we'd have to in out test feed.

⟨ *Obey the testing goat!* 12b ⟩ ≡

```
def test_lookup_whois(self):
    results = [{'ip':'104.41.162.113', 'port':'80'}]
    proxies = lookup_whois(results)
    self.assertEqual(results[0]['whois'], 'Microsoft Corporation')
```
◇

Fragment defined by 8, 9b, 10b, 11b, 12b, 13b, 14a, 15a, 16a.
Fragment referenced in 5.
Uses: `lookup_whois` 12a.

8 Removing Duplicate Entries

In our tests, we've noticed that our various proxy feeds usually share quite a bit of the same proxy data. This in turn, results in us having duplicate entries in our list of proxies. When it comes time to do whois lookups, we obviously don't want to make repeated lookups, so we'll want to remove these duplicate entries before we do our mass lookups.

8.1 An IP Predicate

Since our fancy data structure is just a hacked together list of dictionaries of strings, we'll need to create a predicate test to see if an IP is already in a result list. Once we have this, we can then easily remove the duplicates.

⟨ *Main functions* 13a ⟩ ≡

```
def is_ip_in_dict (ip, proxies):
    """Predicate to test if an IP address string is found in a proxy dictionary. We need this
since we're dealing with lists of dictionaries."""
    result = False
    for proxy in proxies:
        if ip == proxy['ip']:
            result = True
            break
    return result
```

◇

Fragment defined by 7, 9a, 10a, 11a, 12a, 13ac, 14b, 15b, 16b, 17ab.
Fragment referenced in 6.
Defines: `is_ip_in_dict` 13bc.

To test, we'll just verify that we find "183.221.160.12" in our list.

⟨ *Obey the testing goat!* 13b ⟩ ≡

```
def test_is_ip_in_dict (self):
    with open('testfeed.xml', 'rt') as f:
        doc = parse(f)
        spec = 'http://www.proxyrss.com/specification.html'
        results = extract_proxy_data(doc, spec)
        self.assertTrue( is_ip_in_dict ('183.221.160.12', results))
```

◇

Fragment defined by 8, 9b, 10b, 11b, 12b, 13b, 14a, 15a, 16a.
Fragment referenced in 5.
Uses: `extract_proxy_data` 7, `is_ip_in_dict` 13a.

8.2 Actually Removing Duplicates

With our predicate in hand, this is now trivial (though this might be even easier using comprehensions. . . maybe). I didn't think about them when I hacked this together, so this will have to do for now.

⟨ *Main functions* 13c ⟩ ≡

```
def remove_duplicates(proxies):
    """Removes any duplicate IPs from our proxies list of lists."""
    results = []
    for proxy in proxies:
        if results == []:
            results.append(proxy)
            continue
        elif not is_ip_in_dict (proxy['ip'], results):
            results.append(proxy)
    return results
```

◇

Fragment defined by 7, 9a, 10a, 11a, 12a, 13ac, 14b, 15b, 16b, 17ab.
Fragment referenced in 6.
Defines: `remove_duplicates` 14a, 16b, 17a.
Uses: `is_ip_in_dict` 13a.

In our test feed, "183.221.160.12" is duplicated. So we'll test the length before and after removing the duplicates.

⟨ *Obey the testing goat!* 14a ⟩ ≡

```
    def test_remove_duplicates( self ):
        with open('testfeed.xml', 'rt') as f:
            doc = parse(f)
            spec = 'http://www.proxyrss.com/specification.html'
            results = extract_proxy_data(doc, spec)
            self.assertEqual(len( results ), 6)
            results = remove_duplicates( results )
            self.assertEqual(len( results ), 5)
    ◇
```

Fragment defined by 8, 9b, 10b, 11b, 12b, 13b, 14a, 15a, 16a.
Fragment referenced in 5.
Uses: `extract_proxy_data` 7, `remove_duplicates` 13c.

9 Printing Results

At this point we have all the functionality we need to write a complete program to audit open proxy feeds. The last thing we need to do is to figure out how to output processed data. But before we go further, we'll want to fix the "timestamp" field. Unfortunately, some of our feeds do not normalize the timestamp, so we'll first create a function to return a "normalized" timestamp.

In some cases, we're given a timestamp string with no milliseconds, in some cases we'll get a format like this "08/19 13:54:28" with no year. So assuming they're both strings, we'll convert them so it will look like "'08-19-2015 at 13:54:28 UTC". Finally, in some cases, were not even given a time stamp at all. Of course all of this assumes that the times are given in UTC time to begin with. Also, since we're adding the year manually in some cases, we could run into a real time problem if we run this at midnight on new years, but for our purposes, it's fine.

⟨ *Main functions* 14b ⟩ ≡

```
    def convert_time( time_string ):
        """Normalizes a date/time string into a format like so: 08−19−2015 at 13:54:28 UTC"""
        format = "%m−%d−%Y at %H:%M:00 UTC"
        if time_string == "Unknown":
            now = datetime.datetime.now()
            x = "Unknown time, listing found at "
            x += now.strftime(format)
            return x
        elif "/" in time_string:
            tstr = time_string. split ()
            x = tstr [0]. replace("/", "−")
            now = datetime.datetime.now()
            x += "−" + str(now.year) + " at " + tstr[1] + " UTC"
            return x
        else:
            dt = datetime.datetime.utcfromtimestamp(int(time_string))
            return dt.strftime(format)
    ◇
```

Fragment defined by 7, 9a, 10a, 11a, 12a, 13ac, 14b, 15b, 16b, 17ab.
Fragment referenced in 6.
Defines: `convert_time` 15ab.

As always, we'll test all three cases.

⟨ *Obey the testing goat!* 15a ⟩ ≡

```
    def test_convert_time(self):
        self.assertTrue(convert_time("1439993032"), "08-19-2015 at 14:03:00 UTC")
        self.assertTrue(convert_time("08/19 13:54:28"), "08-19-2015 at 13:54:28 UTC")
        format = "%m-%d-%Y at %H:%M:00 UTC"
        now = datetime.datetime.now()
        self.assertTrue(convert_time("Unknown"),
                        "Unknown time, listing found at " + now.strftime(format))
```
◇

Fragment defined by 8, 9b, 10b, 11b, 12b, 13b, 14a, 15a, 16a.
Fragment referenced in 5.
Uses: `convert_time` 14b.

We'll now do two different versions, one for "human readable" output complete with snazzy ANSI colors, and one in boring "grep-able" format. For the grep-able, we'll separate each piece of info by the delimiter character '—' for easy integration with your standard *nix tools like *cut*.

⟨ *Main functions* 15b ⟩ ≡

```
    def print_results(results, print_country=True, human_readable=True):
        """Prints out the results in either a 'human readable format' or as a grep friendly ':'
    separated strings. We can also optionally skip printing the country in the results."""
        if human_readable:
            for i in results:
                #print("Human: " + i['timestamp'])
                time = convert_time(i['timestamp'])
                #print("Human: " + time)
                cprint(i['ip'] + '|' + i['port'] + '|' + time, 'blue', end="")
                cprint("\t => ", 'yellow', end="")
                if print_country:
                    cprint(i['country'], 'magenta', end="")
                    cprint(" => ", 'yellow', end="")
                cprint(i['whois'], 'green')
        else:  # non-human readable
            for i in results:
                #print("Nonhuman: ", i['timestamp'])
                time = convert_time(i['timestamp'])
                #print("Nonhuman: " + time)
                print(i['ip'] + '|' + i['port'] + '|' + time, end="")
                print("|", end="")
                if print_country:
                    print(i['country'], end="")
                    print("|", end="")
                print(i['whois'])
```
◇

Fragment defined by 7, 9a, 10a, 11a, 12a, 13ac, 14b, 15b, 16b, 17ab.
Fragment referenced in 6.
Defines: `print_results` 16ab, 17a.
Uses: `convert_time` 14b.

To test this side-effect "function"[5], well just check to see that it runs successfully.

[5] All apologies to any Haskell hackers reading this.

⟨ *Obey the testing goat!* 16a ⟩ ≡

```
def test_print_results(self):
    with open('testfeed.xml', 'rt') as f:
        doc = parse(f)
        spec = 'http://www.proxyrss.com/specification.html'
        results = extract_proxy_data(doc, spec)
        print_results(results)
        print_results(results, human_readable=False)
        print_results(results, print_country=False, human_readable=False)
```

◇

Fragment defined by 8, 9b, 10b, 11b, 12b, 13b, 14a, 15a, 16a.
Fragment referenced in 5.
Uses: `extract_proxy_data` 7, `print_results` 15b.

10 The Main Program

We're finally able to write our main command line program. As usual for these programs, we'll check the user option flags and pull the feeds as appropriate. To keep our *__main__* small, we're separate the functionality into two driver functions to do the lookups and print the results. We'll create one to pull all proxies and another to filter by country.

10.1 Pulling All Proxies

To pull all the proxies, we'll just get the feed, remove the duplicates, get the whois, and print the results. We set the default variables to non-verbose and non-human-readable since we're assuming that if you're going to pull the data from all proxy data, you'll want it quietly and with easily grep-able results.

⟨ *Main functions* 16b ⟩ ≡

```
def do_all(feeds, print_country=True, verbose=False,
           human_readable=False):
    """Look up and report the results for all proxies."""
    results = []
    for url in feeds:
        if verbose:
            print("[+] Fetching proxies from: %s" % url)
        results += get_proxy_feed(url, verbose)
    # remove the duplicate entries before doing slow whois
    results = remove_duplicates(results)
    if verbose:
        cprint('[+] Found %s unique proxies in feeds.' % len(results),
               'yellow')
        print('[+] Resolving whois descriptions: This can be slow...')
    final = lookup_whois(results, verbose)
    if verbose:
        cprint('[+] Total: %s proxies' % len(results), 'yellow')
    print_results(final, print_country, human_readable)
```

◇

Fragment defined by 7, 9a, 10a, 11a, 12a, 13ac, 14b, 15b, 16b, 17ab.
Fragment referenced in 6.
Defines: `do_all` 19.
Uses: `feeds` 17b, `get_proxy_feed` 9a, `lookup_whois` 12a, `print_results` 15b, `remove_duplicates` 13c.

10.2 Pull Country Specific Proxies

The only difference between this and *do_all* is that we also filter by country data. Note that we default to US given our bias.

⟨ *Main functions* 17a ⟩ ≡

```
    def do_country(feeds, print_country=False, verbose=False,
                   human_readable=False,
                   country_code='US', country='United States'):
        """Look up and report the results for country specific proxies."""
        results = []
        for url in feeds:
            if verbose:
                print("[+] Fetching proxies from: %s" % url)
            results += get_proxy_feed(url, verbose)
            results = filter_by_country(results, country_code, country, verbose)
        # remove the duplicate entries before doing slow whois
        results = remove_duplicates(results)
        if verbose:
            cprint('[+] Found %s unique proxies in feeds.' % len(results),
                   'yellow')
            print('[+] Resolving whois descriptions: This can be slow...')
        final = lookup_whois(results, verbose)
        if verbose:
            cprint('[+] Total: %s %s proxies' % (len(results), country_code), 'yellow')
        print_results(final, print_country, human_readable)
    ◇
```

Fragment defined by 7, 9a, 10a, 11a, 12a, 13ac, 14b, 15b, 16b, 17ab.
Fragment referenced in 6.
Defines: `do_country` 19.
Uses: `feeds` 17b, `filter_by_country` 10a, `get_proxy_feed` 9a, `lookup_whois` 12a, `print_results` 15b, `remove_duplicates` 13c.

10.3 The __main__ Program

Even though we've created our two helper "do" functions, since we're offering so many different options, this main function is going to be big. So we'll break it up into separate parts since most of this is just straight forward Python *argparse*. We'll setup our default feeds here and fill in the rest as we go.

⟨ *Main functions* 17b ⟩ ≡

```
    if __name__ == '__main__':
        feeds = ["http://www.proxz.com/proxylists.xml",
                 "http://www.freeproxylists.com/rss",
                 "http://www.xroxy.com/proxyrss.xml"]

        ⟨ Parse the command line args 18 ⟩
        ⟨ Run the program 19 ⟩
    ◇
```

Fragment defined by 7, 9a, 10a, 11a, 12a, 13ac, 14b, 15b, 16b, 17ab.
Fragment referenced in 6.
Defines: `feeds` 9a, 16b, 17a, 18, 19.

Next we parse the command line. This is standard *argparse*, so I'm not going to explain it, but rather refer the reader to the example usage as follows:

```
usage: proxyfeed.py [-h] [-c country] [-d] [-g] [-l] [-u] [-v]

Pull latest data from the open proxy RSS feeds. (file13@hushmail.me)

optional arguments:
  -h, --help  show this help message and exit
  -c country  filter by 1 country_code AND 1 country ie: -c "US" -c "United
              States"
  -d          do NOT print country in results
  -g          grep parseable results (delimiter='|')
  -l          print the default proxy feeds list
  -u          only report US proxies (implies -c)
  -v          verbose mode
```

As for why I chose these defaults, who knows? It seems like the right thing to do.

⟨*Parse the command line args* 18⟩ ≡

```
import argparse
parser = argparse.ArgumentParser(
    description="""Pull latest data from the open proxy RSS feeds.
    (file13@hushmail.me)""")
parser.add_argument('-c', dest='country_specific',
                    metavar='country',
                    action='append',
                    help='filter by 1 country_code AND 1 country ie: -c "US" -c "United 
States"')
parser.add_argument('-d', dest='print_country',
                    action='store_false',
                    help='do NOT print country in results')
parser.add_argument('-g', dest='human_readable',
                    action='store_false',
                    help="grep parseable results (delimiter='|')")
parser.add_argument('-l', dest='list_feeds',
                    action='store_true',
                    help='print the default proxy feeds list')
parser.add_argument('-u', dest='us_mode',
                    action='store_true',
                    help='only report US proxies (implies -c)')
parser.add_argument('-v', dest='verbose',
                    action='store_true',
                    help='verbose mode')
args = parser.parse_args()
◇
```

Fragment referenced in 17b.
Uses: `feeds` 17b.

And finally we run the program with the given flags. We have four options. One to print the proxy feeds, one to do US proxies, one for country specific proxies, and the default being do all the feeds.

⟨ *Run the program* 19 ⟩ ≡

```
    if args.list_feeds:
        print('Default open proxy RSS feeds:')
        for url in feeds:
            print(url)
        quit()
    if args.us_mode:
        do_country(feeds,
                   print_country=args.print_country,
                   verbose=args.verbose,
                   human_readable=args.human_readable)
    elif args.country_specific:
        if len(args.country_specific) != 2:
            parser.print_help()
        else:
            do_country(feeds,
                       print_country=args.print_country,
                       verbose=args.verbose,
                       human_readable=args.human_readable,
                       country_code=args.country_specific[0],
                       country=args.country_specific[1])
    else:
        do_all(feeds,
               print_country=args.print_country,
               verbose=args.verbose,
               human_readable=args.human_readable)
    ◇
```

Fragment referenced in 17b.
Uses: `do_all` 16b, `do_country` 17a, `feeds` 17b.

And that's it. Happy proxy hunting!

11 To Do

Here are our ideas for future expansion.

- Add to github.
- Add usage examples.
- Add threading to whois lookups.
- Add option to write to a file.
- Add option to read feeds from a file.

12 Updates

1. **August 19, 2015**: Added "convert_time" to normalize timestamps. It's not perfect, but it's because the data from the feeds is sloppy.

13 Loose Ends

13.1 Literate Code Comment

Since we ultimately want folks reading the source code to actually read the literate code instead of the raw source, we'll insert a comment at the beginning of each of our files that directs them to this file.

⟨ *Literate code header comment* 20 ⟩ ≡

```
######################################################################
# This program source is a part of a literate program and was produced
# by nuweb. Please see the accompanying literate program pdf for the
# full documentation.
#
# author: Michael J. Rossi
# contact: file13@hushmail.me
# literate pdf file: proxyfeed.pdf
######################################################################
⟨ BSD License 21 ⟩
◇
```

Fragment referenced in 5, 6.

13.2 License

We'll add our two clause "simplified" BSD license to each file.

⟨ *BSD License* 21 ⟩ ≡

```
######################################################################
# Copyright (c) 2015, Michael J. Rossi
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions
# are met:
#
# * Redistributions of source code must retain the above copyright
# notice, this list of conditions and the following disclaimer.
#
# * Redistributions in binary form must reproduce the above copyright
# notice, this list of conditions and the following disclaimer in the
# documentation and/or other materials provided with the distribution.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
# "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
# LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
# A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
# HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
# SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
# LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
# DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
# THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
# (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
# OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#
######################################################################
◇
```

Fragment referenced in 20.

13.3 Test XML File

Here's our test XML file.

"testfeed.xml" 22≡
```
<?xml version="1.0" encoding="iso−8859−1"?>
<rss xmlns:content="http://purl.org/rss/1.0/modules/content/"
     xmlns:prx="http://www.proxyrss.com/specification.html"
     version="2.0">
<channel>
    <title>Proxz.com : free, fresh, and fast</title>
    <link>http://www.proxz.com/</link>
    <description>Supplying you with proxies since 2003</description>
    <image>
        <title>Proxz.com : free, fresh, and fast</title>
        <url>http://www.proxz.com/images/88x31.gif</url>
        <link>http://www.proxz.com/</link>
    </image>
    <language>en−us</language>
    <lastBuildDate>Fri, 06 Mar 2015 15:41:26 EST</lastBuildDate>
    <ttl>60</ttl>
    <managingEditor>info@proxz.com</managingEditor>
    <generator>Proxy RSS Generator v0.1</generator>
    <item>
        <title>High anonymous proxies on 2015/03/6 20:41:26</title>
        <link>http://www.proxz.com/proxy_list_high_anonymous_0.html</link>
        <description>1339 proxies are in the database at the moment.</description>
        <guid>http://www.proxz.com/proxy_list_high_anonymous_0.html</guid>
        <prx:proxy><prx:ip>183.223.173.237</prx:ip><prx:port>8123</prx:port><prx:type>
Anonymous</prx:type><prx:country>China</prx:country><prx:check_timestamp>03/6 20:23:47</
prx:check_timestamp></prx:proxy>
        <prx:proxy><prx:ip>183.221.208.44</prx:ip><prx:port>8123</prx:port><prx:type>
Anonymous</prx:type><prx:country_code>CN</prx:country_code><prx:check_timestamp>03/6
20:21:37</prx:check_timestamp></prx:proxy>
        <prx:proxy><prx:ip>183.221.160.12</prx:ip><prx:port>8123</prx:port><prx:type>
Anonymous</prx:type></prx:proxy>
        <prx:proxy><prx:ip>32.21.19.4</prx:ip><prx:port>8080</prx:port><prx:type>
Anonymous</prx:type><prx:country_code>US</prx:country_code><prx:check_timestamp>03/6
20:21:37</prx:check_timestamp></prx:proxy>
        <prx:proxy><prx:ip>241.2.28.99</prx:ip><prx:port>8080</prx:port><prx:type>
Anonymous</prx:type><prx:country>United States</prx:country><prx:check_timestamp>03/6
20:21:37</prx:check_timestamp></prx:proxy>
        <prx:proxy><prx:ip>183.221.160.12</prx:ip><prx:port>8123</prx:port><prx:type>
Anonymous</prx:type><prx:check_timestamp>03/6 20:21:34</prx:check_timestamp></prx:proxy>
</item>
</channel>
</rss>
```
◇

14 Makefile

Here's the raw makefile for this program.

```
# This must be the first this in Makefile.common
TOP := $(dir $(lastword $(MAKEFILE_LIST)))

# Main web file
WEB = proxyfeed
# Main procedure
MAIN = $(WEB)
TESTS = tests
PY = python3
#PY = /home/file13/anaconda3/bin/python
all: tangle basic weave

tangle:
        nuweb -rl $(WEB).w
        chmod +x $(MAIN).py
        chmod +x $(TESTS).py

weave: tangle
        pdflatex '\scrollmode \input $(WEB).tex'
        nuweb -rl $(WEB).w
        pdflatex '\scrollmode \input $(WEB).tex'

release: tangle weave

basic:
        $(PY) $(TESTS).py
#       $(PY) $(MAIN).py

clean: clean-doc

clean-doc:
        rm -f *.log *.aux *.tex *.out *.dvi *.toc

fresh: clean
        rm -rf *.pdf *.py *.xml __pycache__

dist: tangle basic clean
        rm -rf __pycache__
```

15 Final Python Source Files

Here is the actual weaved Python source code. Other then the line numbers (which we've added), this is the raw Python source code.

15.1 proxyfeed.py

```
#!/usr/bin/env python3

########################################################################
# This program source is a part of a literate program and was produced
# by nuweb. Please see the accompanying literate program pdf for the
# full documentation.
#
# author: Michael J. Rossi
# contact: file13@hushmail.me
# literate pdf file: proxyfeed.pdf
########################################################################

########################################################################
# Copyright (c) 2015, Michael J. Rossi
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions
# are met:
#
# * Redistributions of source code must retain the above copyright
# notice, this list of conditions and the following disclaimer.
#
# * Redistributions in binary form must reproduce the above copyright
# notice, this list of conditions and the following disclaimer in the
# documentation and/or other materials provided with the distribution.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
# "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
# LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
# A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
# HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
# SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
# LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
# DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
# THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
# (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
# OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#
########################################################################


from urllib.request import urlopen, Request
from xml.etree.ElementTree import parse
from copy import copy, deepcopy
import datetime

# 3rd party libraries
from ipwhois import IPWhois
from termcolor import cprint

def extract_proxy_data(doc, namespace_spec_url):
    """Extracts the data from an already parsed (also use deepcopy) XML doc. Returns a list of
    dictionaries of five strings containing the 'ip', 'port', 'country', 'timestamp', and 'whois
    '."""
    namespaces = {'prx': namespace_spec_url}
    proxies = []
    try:
        for i in doc.iterfind('channel/item/prx:proxy', namespaces):
            ip = i.find('prx:ip', namespaces)
            port = i.find('prx:port', namespaces)
            timestamp = i.find('prx:check_timestamp', namespaces)
            if timestamp == None:  # make fault tolerant
                timestamp = copy(ip)
                timestamp.text = 'Unknown'
            country = i.find('prx:country', namespaces)
            if country == None:
                country = i.find('prx:country_code', namespaces)
            if country == None:
                country = copy(ip)
                country.text = 'Unknown'
            proxy = {'ip':ip.text,
                     'port':port.text,
                     'country':country.text,
                     'timestamp':timestamp.text,
                     'whois':'Unknown'}
            proxies.append(proxy)
    except Exception as e:
        cprint('[-] Error processing XML feed %s!' % doc, 'red')
        cprint('[-] Error: %s' % e, 'yellow')
        quit()
    return proxies

def get_proxy_feed(url, verbose=False, agent='Mozilla/5.0'):
    """Given a url string, pull a proxy RSS feed conforming to the proxyrss.com specification and
      return a list of proxy dictionaries. Some proxy feeds apparently don't like Python, so we'll
      use a different user-agent."""
    request = Request(url, headers={'User-Agent': agent})
    try:
        u = urlopen(request)
    except Exception as e:
        cprint('[-] Error fetching feed %s!' % url, 'red')
        cprint('[-] Error: %s' % e, 'yellow')
        quit()
    doc = parse(u)
    spec = 'http://www.proxyrss.com/specification.html'
    proxies = extract_proxy_data(deepcopy(doc), spec)
    if len(proxies) == 0:  # if empty list, try the other XML spec
        spec = 'http://www.proxyrss.com/content'
        proxies = extract_proxy_data(deepcopy(doc), spec)
    if verbose:
        cprint('[+] Found %s total proxies.' % len(proxies), 'green')
    return proxies

def filter_by_country(proxy_list, country_code, country, verbose=False):
    """Remove any non-US entries from a proxy list generated by get_proxy_feed."""
    proxies = []
    for i in proxy_list:
        if i['country'].endswith(country_code) or i['country'].endswith(country):
            proxies.append(i)
    if verbose:
        cprint('[+] Found %s %s proxies.' %
               (len(proxies), country_code), 'green')
    return proxies

def get_whois_description(ip):
    """Get's the 'description' field from a whois entry given an IP string."""
    obj = IPWhois(ip)
    result = obj.lookup()
    return result['nets'][0]['description']

def lookup_whois(proxies, verbose=True):
    """Performs a whois lookup for the 'description' returning a new list with the 'whois' added
    to the dictionaries."""
    results = []
    count = 0
    for proxy in proxies:
        try:
            whois = get_whois_description(proxy['ip'])
            whois = whois.replace("\n", " ")  # strip newlines
            whois = whois.replace("\r", " ")
            proxy['whois'] = whois
            results.append(proxy)
            if verbose:
                count += 1
                if count % 5 == 0:
                    cprint('[+] Completed %s look ups.' % count,
                           'green')
        except Exception as e:
            cprint('[-] Error resolving whois for %s!' % proxy, 'red')
            cprint('[-] Error: %s' % e, 'yellow')
            cprint('[-] Skipping entry.', 'red')
            continue
    return results

def is_ip_in_dict(ip, proxies):
    """Predicate to test if an IP address string is found in a proxy dictionary. We need this
    since we're dealing with lists of dictionaries."""
    result = False
    for proxy in proxies:
        if ip == proxy['ip']:
            result = True
            break
    return result

def remove_duplicates(proxies):
    """Removes any duplicate IPs from our proxies list of lists."""
    results = []
    for proxy in proxies:
        if results == []:
            results.append(proxy)
            continue
        elif not is_ip_in_dict(proxy['ip'], results):
            results.append(proxy)
    return results

def convert_time(time_string):
    """Normalizes a date/time string into a format like so: 08-19-2015 at 13:54:28 UTC"""
    format = "%m-%d-%Y at %H:%M:00 UTC"
    if time_string == "Unknown":
        now = datetime.datetime.now()
        x = "Unknown time, listing found at "
        x += now.strftime(format)
        return x
    elif "/" in time_string:
        tstr = time_string.split()
        x = tstr[0].replace("/", "-")
        now = datetime.datetime.now()
        x += "-" + str(now.year) + " at " + tstr[1] + " UTC"
        return x
    else:
        dt = datetime.datetime.utcfromtimestamp(int(time_string))
        return dt.strftime(format)

def print_results(results, print_country=True, human_readable=True):
    """Prints out the results in either a 'human readable format' or as a grep friendly ':'
    separated strings. We can also optionally skip printing the country in the results."""
    if human_readable:
        for i in results:
            #print("Human: " + i['timestamp'])
            time = convert_time(i['timestamp'])
            #print("Human: " + time)
            cprint(i['ip'] + '|' + i['port'] + '|' + time, 'blue', end="")
            cprint("\t => ", 'yellow', end="")
            if print_country:
                cprint(i['country'], 'magenta', end="")
                cprint(" => ", 'yellow', end="")
            cprint(i['whois'], 'green')
    else:  # non-human readable
        for i in results:
            #print("Nonhuman: ", i['timestamp'])
            time = convert_time(i['timestamp'])
            #print("Nonhuman: " + time)
            print(i['ip'] + '|' + i['port'] + '|' + time, end="")
            print("|", end="")
            if print_country:
                print(i['country'], end="")
                print("|", end="")
            print(i['whois'])

def do_all(feeds, print_country=True, verbose=False,
           human_readable=False):
    """Look up and report the results for all proxies."""
    results = []
    for url in feeds:
        if verbose:
            print("[+] Fetching proxies from: %s" % url)
        results += get_proxy_feed(url, verbose)
    # remove the duplicate entries before doing slow whois
    results = remove_duplicates(results)
    if verbose:
        cprint('[+] Found %s unique proxies in feeds.' % len(results),
               'yellow')
        print('[+] Resolving whois descriptions: This can be slow...')
    final = lookup_whois(results, verbose)
    if verbose:
        cprint('[+] Total: %s proxies' % len(results), 'yellow')
    print_results(final, print_country, human_readable)

def do_country(feeds, print_country=False, verbose=False,
               human_readable=False,
               country_code='US', country='United States'):
    """Look up and report the results for country specific proxies."""
    results = []
    for url in feeds:
        if verbose:
            print("[+] Fetching proxies from: %s" % url)
        results += get_proxy_feed(url, verbose)
        results = filter_by_country(results, country_code, country, verbose)
    # remove the duplicate entries before doing slow whois
    results = remove_duplicates(results)
    if verbose:
        cprint('[+] Found %s unique proxies in feeds.' % len(results),
               'yellow')
        print('[+] Resolving whois descriptions: This can be slow...')
    final = lookup_whois(results, verbose)
    if verbose:
        cprint('[+] Total: %s %s proxies' % (len(results), country_code), 'yellow')
    print_results(final, print_country, human_readable)

if __name__ == '__main__':
    feeds = ["http://www.proxz.com/proxylists.xml",
             "http://www.freeproxylists.com/rss",
             "http://www.xroxy.com/proxyrss.xml"]


    import argparse
    parser = argparse.ArgumentParser(
        description="""Pull latest data from the open proxy RSS feeds.
        (file13@hushmail.me)""")
    parser.add_argument('-c', dest='country_specific',
                        metavar='country',
                        action='append',
                        help='filter by 1 country_code AND 1 country ie: -c "US" -c "United States"')
    parser.add_argument('-d', dest='print_country',
                        action='store_false',
                        help='do NOT print country in results')
    parser.add_argument('-g', dest='human_readable',
                        action='store_false',
                        help="grep parseable results (delimiter='|')")
    parser.add_argument('-l', dest='list_feeds',
                        action='store_true',
                        help='print the default proxy feeds list')
    parser.add_argument('-u', dest='us_mode',
                        action='store_true',
                        help='only report US proxies (implies -c)')
    parser.add_argument('-v', dest='verbose',
                        action='store_true',
                        help='verbose mode')
    args = parser.parse_args()


    if args.list_feeds:
        print('Default open proxy RSS feeds:')
        for url in feeds:
            print(url)
        quit()
    if args.us_mode:
        do_country(feeds,
                   print_country=args.print_country,
                   verbose=args.verbose,
                   human_readable=args.human_readable)
    elif args.country_specific:
        if len(args.country_specific) != 2:
            parser.print_help()
        else:
            do_country(feeds,
                       print_country=args.print_country,
                       verbose=args.verbose,
                       human_readable=args.human_readable,
                       country_code=args.country_specific[0],
                       country=args.country_specific[1])
    else:
        do_all(feeds,
               print_country=args.print_country,
               verbose=args.verbose,
               human_readable=args.human_readable)
```

15.2 tests.py

```
#!/usr/bin/env python3

######################################################################
# This program source is a part of a literate program and was produced
# by nuweb. Please see the accompanying literate program pdf for the
# full documentation.
#
# author: Michael J. Rossi
# contact: file13@hushmail.me
# literate pdf file: proxyfeed.pdf
######################################################################

######################################################################
# Copyright (c) 2015, Michael J. Rossi
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions
# are met:
#
```

```
# * Redistributions of source code must retain the above copyright
# notice, this list of conditions and the following disclaimer.
#
# * Redistributions in binary form must reproduce the above copyright
# notice, this list of conditions and the following disclaimer in the
# documentation and/or other materials provided with the distribution.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
# "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
# LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
# A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
# HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
# SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
# LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
# DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
# THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
# (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
# OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#
########################################################################


import unittest
from xml.etree.ElementTree import parse
from proxyfeed import *

class ProxyFeedTest(unittest.TestCase):

    def test_extract_proxy_data(self):
        with open('testfeed.xml', 'rt') as f:
            doc = parse(f)
            spec = 'http://www.proxyrss.com/specification.html'
            results = extract_proxy_data(doc, spec)
            self.assertEqual(results[0]['ip'], '183.223.173.237')
            self.assertEqual(results[0]['port'], '8123')
            self.assertEqual(results[0]['country'], 'China')
            self.assertEqual(results[0]['timestamp'], '03/6 20:23:47')

            self.assertEqual(results[1]['ip'], '183.221.208.44')
            self.assertEqual(results[1]['port'], '8123')
            self.assertEqual(results[1]['country'], 'CN')
            self.assertEqual(results[1]['timestamp'], '03/6 20:21:37')

            self.assertEqual(results[2]['ip'], '183.221.160.12')
            self.assertEqual(results[2]['port'], '8123')
            self.assertEqual(results[2]['country'], 'Unknown')
            self.assertEqual(results[2]['timestamp'], 'Unknown')

    def test_get_proxy_feed(self):
        feed = 'http://www.proxz.com/proxylists.xml'
        results = get_proxy_feed(feed)
        self.assertTrue(results[0]['ip'])
        self.assertTrue(results[0]['port'])
        self.assertTrue(results[0]['country'])
        self.assertTrue(results[0]['timestamp'])
        self.assertTrue(results[0]['country'])

    def test_filter_by_country(self):
        with open('testfeed.xml', 'rt') as f:
            doc = parse(f)
            spec = 'http://www.proxyrss.com/specification.html'
            results = extract_proxy_data(doc, spec)
            results = filter_by_country(results, 'CN', 'China')
            self.assertEqual(len(results), 2)
            results = extract_proxy_data(doc, spec)
            results = filter_by_country(results, 'US', 'United States')
            self.assertEqual(len(results), 2)

    def test_get_whois_description(self):
        results = get_whois_description('104.41.162.113')
        self.assertEqual(results, 'Microsoft Corporation')

    def test_lookup_whois(self):
        results = [{'ip':'104.41.162.113', 'port':'80'}]
        proxies = lookup_whois(results)
        self.assertEqual(results[0]['whois'], 'Microsoft Corporation')

    def test_is_ip_in_dict(self):
        with open('testfeed.xml', 'rt') as f:
            doc = parse(f)
            spec = 'http://www.proxyrss.com/specification.html'
            results = extract_proxy_data(doc, spec)
            self.assertTrue(is_ip_in_dict('183.221.160.12', results))

    def test_remove_duplicates(self):
        with open('testfeed.xml', 'rt') as f:
            doc = parse(f)
            spec = 'http://www.proxyrss.com/specification.html'
            results = extract_proxy_data(doc, spec)
            self.assertEqual(len(results), 6)
            results = remove_duplicates(results)
            self.assertEqual(len(results), 5)

    def test_convert_time(self):
        self.assertTrue(convert_time("1439993032"), "08-19-2015 at 14:03:00 UTC")
        self.assertTrue(convert_time("08/19 13:54:28"), "08-19-2015 at 13:54:28 UTC")
        format = "%m-%d-%Y at %H:%M:00 UTC"
        now = datetime.datetime.now()
        self.assertTrue(convert_time("Unknown"),
                        "Unknown time, listing found at " + now.strftime(format))

    def test_print_results(self):
        with open('testfeed.xml', 'rt') as f:
            doc = parse(f)
            spec = 'http://www.proxyrss.com/specification.html'
            results = extract_proxy_data(doc, spec)
            print_results(results)
            print_results(results, human_readable=False)
            print_results(results, print_country=False, human_readable=False)


if __name__ == '__main__':
    unittest.main(warnings='ignore',failfast=True)
```

16 Raw Nuweb File

Here is the raw nuweb file that produced this for folks curious as to what this looks like or in the strange event where this PDF is all that remains of the program and someone wanted to reconstruct it. Otherwise, you'll want to ignore this.

```
% This is a nuweb literate program: proxyfeed.w
% To tangle/weave, run:
% nuweb -rl proxyfeed.w

\documentclass[10pt]{article}
%\documentclass[a4paper]{report}
%\documentclass[9pt]{extarticle}

\usepackage{latexsym}
\usepackage{amsfonts}
\usepackage{amssymb}
\usepackage{courier}

% Code styles
\usepackage{ascii}
\usepackage{listings}

% Use this style for plain lstlisting Python.
\lstdefinestyle{Mono Python}{
  extendedchars=true,keepspaces=true,language=Python,
  basicstyle=\footnotesize\ttfamily,
  morecomment=[s]{"""}{"""},
  breaklines=true,
  % To create more custom keywords--for example, typedefs, add them
  % to morekeywords.
  % morekeywords={Thing},
}

% Use this style for CWEB style code formatting.
% Feel free to change any of this to suit personal preference.
\lstdefinestyle{CWEB Python}{
  extendedchars=true,keepspaces=true,language=Python,
  basicstyle=\normalsize\rmfamily,
  identifierstyle=\itshape,
  commentstyle=\normalsize\rmfamily,
  stringstyle=\normalsize\asciifamily,
  %stringstyle=\footnotesize\ttfamily,
  showstringspaces=true,
  morecomment=[s]{"""}{"""},
  breaklines=true,
  % To create more custom keywords--for example, typedefs, add them
  % to morekeywords.
  % morekeywords={Thing},
  literate=*
  % General CWEB Style
  {+}{{$+$}}1
  {-}{{$-$}}1
  {*}{{$*$}}1
  {(}{{$($}}1 {)}{{$)$}}1
  {<}{{$<$}}1 {>}{{$>$}}1
  %{<=}{{$\leq$}}1 {>=}{{$\geq$}}1
  %{!}{{\kern.3em{$\lnot$}}}2
  %{!}{{$\lnot$}}2
  %{!=}{{$\not=$}}2
  %{!=}{{$\not\equiv$}}2
  %{&&}{{$\land$}}2
  %{||}{{$\lor$}}2
  %{\%}{{\footnotesize{$\%$}}}1
  %{|}{{$\kern3pt|$}}1
  %{^}{{$\oplus$}}1
  %{~}{{$\sim$}}1
  %{<<}{{$\ll$}}1
  %{>>}{{$\gg$}}1
  %{++}{{$+\!+$}}1
  %{——}{{$−\kern−.5pt−$}}1
  %{=}{{$\gets$}}1
  %{==}{{$\equiv$}}1
  %{NULL}{{$\Lambda$}}1
  %{−>}{{\tiny\raisebox{.6ex}{$\!\rightarrow$}}}1
}

%% To change the code style if you don’t want fancy CWEB style,
%% change it here.
\lstset{style=CWEB Python}
%\lstset{style=Mono Python}

% "Example" style for lstlistings, turns off anything fancy
% for when you want to show examples
\lstdefinestyle{example}{
  language=,
  style=,
  breaklines,
  keepspaces=true,
  identifierstyle=\normalsize\ttfamily,
  frame=single,
  extendedchars=true,
  literate=*
  %{−}{{−}}1 % listing package screws up, puts minus sign for dash
  %{−}{{$−$}}1
  %{−}{{——}}1 % this interfers with our morekeywords
}

\lstdefinestyle{rawcode}{
  language=,
  style=,
  frame=,
  extendedchars=true,
  keepspaces=true,
  breaklines,
  basicstyle=\footnotesize\ttfamily,
}

% taken from nuweb
\usepackage{color}
\definecolor{linkcolor}{rgb}{0, 0, 0.7}
%\usepackage[plainpages=false,pdftex,colorlinks,backref]{hyperref}

\usepackage[%
backref,%
raiselinks,%
pdfhighlight=/O,%
pagebackref,%
hyperfigures,%
%breaklinks,%
%plainpages=false,
colorlinks,%
pdfpagemode=UseNone,%
pdfstartview=FitBH,%
linkcolor={linkcolor},%
anchorcolor={linkcolor},%
citecolor={linkcolor},%
filecolor={linkcolor},%
menucolor={linkcolor},%
urlcolor={linkcolor}%
]{hyperref}

%\usepackage[hidelinks]{hyperref} %hide ugly links
\usepackage[margin=1in]{geometry} %smaller margin
\usepackage{fancyvrb} % for our includetextfile
\usepackage{multicol} % for three columns
```

```
% Used to include raw text files.
\newcommand{\includetextfile}[1]{
  \lstinputlisting[
    style=rawcode,
  ]{#1}
}

% Used to include raw text files with line numbering.
\newcommand{\includenumberedtextfile}[1]{
  \lstinputlisting[
    style=rawcode,
    numbers=left,
  ]{#1}
}

\begin{document}
%\hoffset 0in

\title{Proxyfeed -- An Open Proxy RSS Auditing Tool}
\date{March 11, 2015}
\author{Michael J. Rossi}

\maketitle
\newpage
%\section{Table of Contents}
\tableofcontents
\newpage

\section{Introduction}
\subsection{Our Purpose}
{\it There is no darkness, nor shadow of death, where the workers of
  iniquity may hide themselves. -- Job 34:22}\\
\\
Malicious hackers love open proxies. They love them so much, that they
share them in various ways throughout the web so that other hackers
and script-kiddes can use them to stay hidden.\footnote{Assuming, of course,
that the black hats didn't setup a malicious proxy to catch the
script-kiddies{\ldots}oh the tangled webs we weave.}
One of the best places
we've seen for up to date lists of open proxies are the various RSS
feeds available online. The following program attempts to tap into the
feeds so that we can better locate any open proxies being
publicly advertised online. To do so, we'll write a small Python
program to pull these various feeds, parse them, and then process them
in various ways.

\subsection{The Various Proxy Feeds}
There are various sites out dedicated to open proxies which
provide RSS feeds of the latest victims. The site
\href{http://www.proxyrss.com/}{proxyrss.com} not only collects
these various feeds into a central location, but it also provides an
XML namespace specification to normalize the feeds. At this time,
we have found the following feeds to be live, active, and conforming.
\\

\noindent{\bf{Proxy RSS Specification:}
  \href{http://www.proxyrss.com/specification.html}
       {http://www.proxyrss.com/specification.html}.}

\begin{enumerate}
\item \href{http://www.proxz.com/proxylists.xml}
  {http://www.proxz.com/proxylists.xml}
\item \href{http://www.freeproxylists.com/rss}
  {http://www.freeproxylists.com/rss}
\item \href{http://www.xroxy.com/proxyrss.xml}
  {http://www.xroxy.com/proxyrss.xml}
\end{enumerate}
```

```
Of these feeds,
\href{http://www.xroxy.com/proxyrss.xml}{xroxy.com} is the only one
not fully
conforming to the spec. Instead of the correct spec XML namespace,
they instead have the namespace listed as
\href{http://www.proxyrss.com/content}{http://www.proxyrss.com/content}.
At this time this link is 404’d, so I’m assuming that this was the old
location and that
\href{http://www.xroxy.com/proxyrss.xml}{xroxy.com}
has not updated their feed
data. We bring this up now because this anomaly will come
up when we begin to parse the feeds.

\subsection{Installation}
The code is written in Python 3. We’re using the following two third
party libraries (i.e. not included by default in the Python
distribution).

\begin{itemize}
\item ipwhois
\item termcolor {\it (for 31337 output)}
\end{itemize}

If you don’t have
these installed on our system, you’ll need to install them using
pip. For example:

\begin{lstlisting}[style=example]
$ pip install ipwhois
$ pip install termcolor
\end{lstlisting}

Otherwise, you just run it like any other Python program. For example:

\begin{lstlisting}[style=example]
$ python proxyfeed.py —h
\end{lstlisting}

Please also note that this is being written and tested on Kali Linux
using the \href{https://store.continuum.io/cshop/anaconda/}{Anaconda}
Python distribution, specifically, Python 3.4.1, Anaconda 2.1.0
(64—bit). I will also try to test it on Windows 7, but it’s
an after thought.

\subsection{Literate Programming}
This is a literate program written using
\href{http://nuweb.sourceforge.net/}{\bf nuweb} with Python. Literate
programming is a discipline whereby the author writes the documentation
and code as any other piece of prose that explains the entire program
in whatever order or manner is best for the problem being tackled. By
adhering to this discipline, the documentation and the code will not
go out of sync. Moreover, it will be easy for anyone (including the
author sometime in the future) to learn how the program works and why
the author did what he did. For more information, please see
\href{http://http://literateprogramming.com/} {\bf
  literateprogramming.com}.

Please note that for the Python code, we slightly pretty print the
code in a mild
\href{http://www—cs—faculty.stanford.edu/˜uno/cweb.html}{CWEB}
style to make it easier on the eyes while
still conforming to the ‘‘Zen of Python’’.\footnote{Try running
  ‘‘import this’’ in your Python interpreter if you don’t know what
  this means.}
But we don’t alter the
syntax any
like substituting $==$ for $\equiv$ or $!=$ with $\not=$.
```

```
If for some bizarre reason you have trouble reading the pretty printed
code, please consult the raw code later in this PDF.

The source code consists of only two files. A {\it Makefile} and the
{\it web} file {\it proxyfeed.w} which generates everything else,
including this PDF, proxyfeed.py, tests.py (unit tests), and even a
test XML feed. This PDF documentation includes not only the ``weaved''
literate text,
but also the ``tangled'' or produced source files along with the full
nuweb ``web'' source. In other words, this PDF contains everything
produced by {\it nuweb} in it--including the separate {\it Makefile}.

\section{Test Framework}
Along with our literate program, we also adhere to the discipline of
testing. In this spirit, we'll setup our test framework which we'll
fill in as we go. Although it
won't be apparent from this, we actually write our tests first and
then the functions to pass the tests. But in our literate text, we'll
put the tests afterwards so that if you're just interested in learning
what, how, and why the program does what it does, you can skip the
tests. If you're not familiar with the ``Testing Goat,'' and what it
means, please visit
\href{http://www.obeythetestinggoat.com/}{\bf obeythetestinggoat.com}.

The test framework will be in it's own file. The only thing of note at
this time is that {\it proxyfeed} will be our main program, so we'll
import it now.

@o tests.py @{#!/usr/bin/env python3
@<Literate code header...@>
import unittest
from xml.etree.ElementTree import parse
from proxyfeed import *

class ProxyFeedTest(unittest.TestCase):
    @<Obey the testing goat!@>

if __name__ == '__main__':
    unittest.main(warnings='ignore',failfast=True)
@| ProxyFeedTest setUp tearDown @}

\section{Main Code Outline}
\subsection{Broad Overview}
As mentioned, the program will simply pull the RSS feed, parse the XML
document, and then process the data. From the point of view of
security, when it comes to processing the data, we'll want to pull the
whois information from the IP addresses in order to determine who the
proxy belongs to. We can, of course, do more processing later, but the
main thing we're after is the ability to pull the data and see a list
of proxies along with the whois description of the IP.

\subsection{The Outline}
Here is the outline which we'll fill in as we go along with our
imports.

@o proxyfeed.py @{#!/usr/bin/env python3
@<Literate code header...@>
from urllib.request import urlopen, Request
from xml.etree.ElementTree import parse
from copy import copy, deepcopy
import datetime

# 3rd party libraries
from ipwhois import IPWhois
from termcolor import cprint
@<Main functions@>
@|@}
```

```
\section{Extracting XML Feed Data}
\subsection{Mapping XML to Python}
According to the
\href{http://www.proxyrss.com/specification.html}{Proxy RSS
  Specification},
we’ll always have the following elements in a conforming proxy feed:

\begin{lstlisting}[language=XML,style=]
  <prx:proxy>
  <prx:ip>
  <prx:port>
\end{lstlisting}

There are various optional elements, but the only two we’re
interested in at this time is the ‘‘country’’ of origin and the
‘‘timestamp’’ when the proxy was last verified. The timestamp is as
follows.

\begin{lstlisting}[language=XML,style=]
  <prx:check_timestamp>
\end{lstlisting}

As for countries, if we’re
trying to audit our organization in the US, we obviously don’t care
about the open proxies in North Korea\footnote{Unless we’re just so
  very ronery.}, so the country will come in handy later. In the spec,
there are two optional values for the country as follows:

\begin{lstlisting}[language=XML,style=]
  <prx:country>
  <prx:country_code>
\end{lstlisting}

The {\it country} is an unofficial string description while the
{\it country\_code} should satisfy
\href{https://en.wikipedia.org/wiki/ISO_3166—1_alpha—2}
     {ISO 3166—1 Alpha—2.}
In our experience at this time, only one feed uses the
{\it country\_code}’s while the rest use {\it country}.

We’ll store all of this data into a Python dictionary which will
have five string values and look like this.

\begin{lstlisting}[language=Python,style=]
  proxy = {’ip’:’192.168.1.1’,
           ’port’:’3128’,
           ’country’:’US’,
           ’timestamp’:’01/01/1970’,
           ’whois’:’Unknown’}
\end{lstlisting}

Please note that the country, timestamp, and whois do not have any
standard format.

\subsection{Extracting the Data}
For our actual parse function, we’ll take an already parsed XML
document and the url to the XML namespace.
Since there’s problems with feeds using non—standard specs, the
document we pass will need to be a {\it deepcopy}.
Otherwise, we’ll simply do our normal XML parsing with
namespaces and add the results to a dictionary which we’ll then add to
a list of proxies.

The only wrinkle in the code is the problems with the ‘‘countries’’
and setting default fault tolerant values. We’ll check for both types
of ‘‘countries’’ and add a default value if
for some reason we don’t find any. You (apparently\footnote{Though I
  could be wrong here.})
```

```
can’t make your own {\it xml.etree.ElementTree.Element} and set the
{\it text} value, so we’ll make a copy of a mandatory XML field.
Since the IP is mandatory, we’ll copy it and then change the
{\it text} value. We’ll do the same for the timestamp.

Finally, it might be worth noting the use of {\it cprint}
from the {\it termcolor} library which allows us to print in riveting
ASCII colors.

@d Main functions @{
def extract_proxy_data(doc, namespace_spec_url):
    """Extracts the data from an already parsed (also use deepcopy) XML doc. Returns a list of
    dictionaries of five strings containing the ’ip’, ’port’, ’country’, ’timestamp’, and ’whois
    ’."""
    namespaces = {’prx’: namespace_spec_url}
    proxies = []
    try:
        for i in doc.iterfind(’channel/item/prx:proxy’, namespaces):
            ip = i.find(’prx:ip’, namespaces)
            port = i.find(’prx:port’, namespaces)
            timestamp = i.find(’prx:check_timestamp’, namespaces)
            if timestamp == None:  # make fault tolerant
                timestamp = copy(ip)
                timestamp.text = ’Unknown’
            country = i.find(’prx:country’, namespaces)
            if country == None:
                country = i.find(’prx:country_code’, namespaces)
            if country == None:
                country = copy(ip)
                country.text = ’Unknown’
            proxy = {’ip’:ip.text,
                     ’port’:port.text,
                     ’country’:country.text,
                     ’timestamp’:timestamp.text,
                     ’whois’:’Unknown’}
            proxies.append(proxy)
    except Exception as e:
        cprint(’[—] Error processing XML feed %s!’ % doc, ’red’)
        cprint(’[—] Error: %s’ % e, ’yellow’)
        quit()
    return proxies
@| extract_proxy_data @}

To test this, we’ll snip down a real feed and use it in our tests. The
XML file named {\it testfeed.xml} is included at the end of this
document if you’d like to examine it. Otherwise, we’ll just parse the
file and check the results.

@d Obey the... @{
def test_extract_proxy_data(self):
    with open(’testfeed.xml’, ’rt’) as f:
        doc = parse(f)
        spec = ’http://www.proxyrss.com/specification.html’
        results = extract_proxy_data(doc, spec)
        self.assertEqual(results[0][’ip’], ’183.223.173.237’)
        self.assertEqual(results[0][’port’], ’8123’)
        self.assertEqual(results[0][’country’], ’China’)
        self.assertEqual(results[0][’timestamp’], ’03/6 20:23:47’)

        self.assertEqual(results[1][’ip’], ’183.221.208.44’)
        self.assertEqual(results[1][’port’], ’8123’)
        self.assertEqual(results[1][’country’], ’CN’)
        self.assertEqual(results[1][’timestamp’], ’03/6 20:21:37’)

        self.assertEqual(results[2][’ip’], ’183.221.160.12’)
        self.assertEqual(results[2][’port’], ’8123’)
        self.assertEqual(results[2][’country’], ’Unknown’)
        self.assertEqual(results[2][’timestamp’], ’Unknown’)
```

```
@| test_extract_proxy_data @}

\section{Puling the RSS Feeds}
Now that we can parse the feed, we’ll pull the feed and use
{\it extract\_proxy\_data} to do the dirty work. Some of the proxy
feed sites apparently don’t like Python (or curl and
wget for that matter), so we’ll need to change the ‘‘User—Agent’’
string so these sites think we’re a ‘‘normal web
browser’’ (though a highly generic and tight—lipped one). Otherwise, we
simply pull the data via {\it Request}, parse it, make a deepcopy, and
pass it to {\it extract\_proxy\_data}. If {\it proxies} comes back
empty, we’ll assume it’s because it doesn’t have the right XML
namespace, and try again with the {\it content} URL.

@d Main functions @{
def get_proxy_feed(url, verbose=False, agent=’Mozilla/5.0’):
    """Given a url string, pull a proxy RSS feed conforming to the proxyrss.com specification and
     return a list of proxy dictionaries. Some proxy feeds apparently don’t like Python, so we’ll
     use a different user—agent."""
    request = Request(url, headers={’User—Agent’: agent})
    try:
        u = urlopen(request)
    except Exception as e:
        cprint(’[—] Error fetching feed %s!’ % url, ’red’)
        cprint(’[—] Error: %s’ % e, ’yellow’)
        quit()
    doc = parse(u)
    spec = ’http://www.proxyrss.com/specification.html’
    proxies = extract_proxy_data(deepcopy(doc), spec)
    if len(proxies) == 0:  # if empty list, try the other XML spec
        spec = ’http://www.proxyrss.com/content’
        proxies = extract_proxy_data(deepcopy(doc), spec)
    if verbose:
        cprint(’[+] Found %s total proxies.’ % len(proxies), ’green’)
    return proxies
@| get_proxy_feed @}

Since we’ve already tested parsing, we’ll just write a simple test to
see that make sure we’re pulling correctly by just checking that we
get some results from a feed.

@d Obey the... @{
def test_get_proxy_feed(self):
    feed = ’http://www.proxz.com/proxylists.xml’
    results = get_proxy_feed(feed)
    self.assertTrue(results[0][’ip’])
    self.assertTrue(results[0][’port’])
    self.assertTrue(results[0][’country’])
    self.assertTrue(results[0][’timestamp’])
    self.assertTrue(results[0][’country’])
@|@}

\section{Filtering Country Specific Proxy Data}
Since we’re in the US and interested in auditing US entities, we’d like
to be able to filter our proxy data down to just US proxies. But while
we’re writing a function to filter US only proxies, we can make it
even more general by allowing us to filter by any country. This will
require us to know both the appropriate {\it country\_code} and
{\it country} format. Otherwise, this is just a simple filtering
function.

@d Main functions @{
def filter_by_country(proxy_list, country_code, country, verbose=False):
    """Remove any non—US entries from a proxy list generated by get_proxy_feed."""
    proxies = []
    for i in proxy_list:
        if i[’country’].endswith(country_code) or i[’country’].endswith(country):
            proxies.append(i)
    if verbose:
        cprint('[+] Found %s %s proxies.' %
               (len(proxies), country_code), 'green')
    return proxies
@| filter_by_country @}

To test, we'll use our test file and filter out any non—Chinese
proxies. Our test file has two Chinese entries in it, so this is
straightforward. We'll then repeat the test for US proxies, which also
has two entries.

@d Obey the... @{
def test_filter_by_country(self):
    with open('testfeed.xml', 'rt') as f:
        doc = parse(f)
        spec = 'http://www.proxyrss.com/specification.html'
        results = extract_proxy_data(doc, spec)
        results = filter_by_country(results, 'CN', 'China')
        self.assertEqual(len(results), 2)
        results = extract_proxy_data(doc, spec)
        results = filter_by_country(results, 'US', 'United States')
        self.assertEqual(len(results), 2)
@|@}

\section{Whois Lookups}
At this point, we're already capable of pulling and parsing open proxy
feeds. But an IP and country information isn't sufficient to locate
proxies associated with a specific organization. So we'll want to pull
the whois
information from the IP address. But rather then digging through a
huge whois dump, we'll just get the ``description'' data for the
IP. Luckily, Python already has the
\href{https://pypi.python.org/pypi/ipwhois}{ipwhois} library which
can do this for us without having to roll our own.

\subsection{Getting Whois Descriptions}
We'll first write a little function to return the whois ``descrption''
field for an IP address.

@d Main functions @{
def get_whois_description(ip):
    """Get's the 'description' field from a whois entry given an IP string."""
    obj = IPWhois(ip)
    result = obj.lookup()
    return result['nets'][0]['description']
@| get_whois_description @}

The test is pretty obvious. Please note that this IP was chosen at
random and it just happened to be MS.

@d Obey the... @{
def test_get_whois_description(self):
    results = get_whois_description('104.41.162.113')
    self.assertEqual(results, 'Microsoft Corporation')
@|@}

\subsection{Doing Mass Whois Lookups}
We now want a function to do our lookups on every IP in our proxy
list. Our code will simply run through all the
IPs, pull the whois description, and add them to the dictionary.
In our tests, some organizations (I'm looking at
you China!) use really long ``descriptions'' that span multiple
lines. So we'll strip out any newline characters so the description is
just a single line.\\
\\
{\noindent\bf Note:} In the future, I'd like to thread this to speed it
up since we're at the mercy of whois servers.
```

```
@d Main functions @{
def lookup_whois(proxies, verbose=True):
    """Performs a whois lookup for the ’description’ returning a new list with the ’whois’ added
    to the dictionaries."""
    results = []
    count = 0
    for proxy in proxies:
        try:
            whois = get_whois_description(proxy[’ip’])
            whois = whois.replace("\n", " ")  # strip newlines
            whois = whois.replace("\r", " ")
            proxy[’whois’] = whois
            results.append(proxy)
            if verbose:
                count += 1
                if count % 5 == 0:
                    cprint(’[+] Completed %s look ups.’ % count,
                           ’green’)
        except Exception as e:
            cprint(’[-] Error resolving whois for %s!’ % proxy, ’red’)
            cprint(’[-] Error: %s’ % e, ’yellow’)
            cprint(’[-] Skipping entry.’, ’red’)
            continue
    return results
@| lookup_whois @}

To test, we’ll manually build an entry so we don’t have to do multiple
lookups as we’d have to in out test feed.

@d Obey the... @{
def test_lookup_whois(self):
    results = [{’ip’:’104.41.162.113’, ’port’:’80’}]
    proxies = lookup_whois(results)
    self.assertEqual(results[0][’whois’], ’Microsoft Corporation’)
@|@}

\section{Removing Duplicate Entries}
In our tests, we’ve noticed that our various proxy feeds usually share
quite a
bit of the same proxy data. This in turn, results in us having
duplicate entries in our list of proxies. When it comes time to do
whois lookups, we obviously don’t want
to make repeated lookups, so we’ll want to remove these duplicate
entries before we do our mass lookups.

\subsection{An IP Predicate}
Since our fancy data structure is just a hacked together list of
dictionaries of strings, we’ll need to create a predicate test to see
if an IP is already in a result list. Once we have this, we can then
easily remove the duplicates.

@d Main functions @{
def is_ip_in_dict(ip, proxies):
    """Predicate to test if an IP address string is found in a proxy dictionary. We need this
    since we’re dealing with lists of dictionaries."""
    result = False
    for proxy in proxies:
        if ip == proxy[’ip’]:
            result = True
            break
    return result
@| is_ip_in_dict @}

To test, we’ll just verify that we find ‘‘183.221.160.12’’ in our
list.

@d Obey the... @{
def test_is_ip_in_dict(self):
    with open('testfeed.xml', 'rt') as f:
        doc = parse(f)
        spec = 'http://www.proxyrss.com/specification.html'
        results = extract_proxy_data(doc, spec)
        self.assertTrue(is_ip_in_dict('183.221.160.12', results))
@|@}

\subsection{Actually Removing Duplicates}
With our predicate in hand, this is now trivial (though this might be
even easier using
comprehensions{\ldots}maybe). I didn't think about them when I hacked
this together, so this will have to do for now.

@d Main functions @{
def remove_duplicates(proxies):
    """Removes any duplicate IPs from our proxies list of lists."""
    results = []
    for proxy in proxies:
        if results == []:
            results.append(proxy)
            continue
        elif not is_ip_in_dict(proxy['ip'], results):
            results.append(proxy)
    return results
@| remove_duplicates @}

In our test feed, ``183.221.160.12'' is duplicated. So we'll test the
length before and after removing the duplicates.

@d Obey the... @{
def test_remove_duplicates(self):
    with open('testfeed.xml', 'rt') as f:
        doc = parse(f)
        spec = 'http://www.proxyrss.com/specification.html'
        results = extract_proxy_data(doc, spec)
        self.assertEqual(len(results), 6)
        results = remove_duplicates(results)
        self.assertEqual(len(results), 5)
@|@}

\section{Printing Results}
At this point we have all the functionality we need to write a
complete program to audit open proxy feeds. The last thing we need
to do is to figure out how to output processed data. But before we go
further, we'll want to fix the ``timestamp'' field. Unfortunately, some
of our feeds do not normalize the timestamp, so we'll first create a
function to return a ``normalized'' timestamp.

In some cases, we're given a timestamp string with no milliseconds, in
some cases we'll get  a format like this ``08/19 13:54:28'' with no
year. So assuming they're both strings, we'll convert them so it will
look like ``'08—19—2015 at 13:54:28 UTC''. Finally, in some cases,
were not even given a time stamp at all. Of course all of this
assumes that the times are given in UTC time to begin with. Also,
since we're adding the year manually in some cases, we could run into
a real time problem if we run this at midnight on new years, but for
our purposes, it's fine.

@d Main functions @{
def convert_time(time_string):
    """Normalizes a date/time string into a format like so: 08—19—2015 at 13:54:28 UTC"""
    format = "%m—%d—%Y at %H:%M:00 UTC"
    if time_string == "Unknown":
        now = datetime.datetime.now()
        x = "Unknown time, listing found at "
        x += now.strftime(format)
        return x
    elif "/" in time_string:
        tstr = time_string.split()
        x = tstr[0].replace("/", "-")
        now = datetime.datetime.now()
        x += "-" + str(now.year) + " at " + tstr[1] + " UTC"
        return x
    else:
        dt = datetime.datetime.utcfromtimestamp(int(time_string))
        return dt.strftime(format)
@| convert_time @}
```

As always, we'll test all three cases.

```
@d Obey the... @{
def test_convert_time(self):
    self.assertTrue(convert_time("1439993032"), "08-19-2015 at 14:03:00 UTC")
    self.assertTrue(convert_time("08/19 13:54:28"), "08-19-2015 at 13:54:28 UTC")
    format = "%m-%d-%Y at %H:%M:00 UTC"
    now = datetime.datetime.now()
    self.assertTrue(convert_time("Unknown"),
                    "Unknown time, listing found at " + now.strftime(format))
 @|@}
```

We'll now do two
different versions, one for ``human readable'' output complete with
snazzy ANSI colors, and one in boring ``grep-able'' format. For the
grep-able, we'll separate each piece of info by the delimiter
character `|' for easy integration with your standard *nix tools like
{\it cut}.

```
@d Main functions @{
def print_results(results, print_country=True, human_readable=True):
    """Prints out the results in either a 'human readable format' or as a grep friendly ':'
    separated strings. We can also optionally skip printing the country in the results."""
    if human_readable:
        for i in results:
            #print("Human: " + i['timestamp'])
            time = convert_time(i['timestamp'])
            #print("Human: " + time)
            cprint(i['ip'] + '|' + i['port'] + '|' + time, 'blue', end="")
            cprint("\t => ", 'yellow', end="")
            if print_country:
                cprint(i['country'], 'magenta', end="")
                cprint(" => ", 'yellow', end="")
            cprint(i['whois'], 'green')
    else:  # non-human readable
        for i in results:
            #print("Nonhuman: ", i['timestamp'])
            time = convert_time(i['timestamp'])
            #print("Nonhuman: " + time)
            print(i['ip'] + '|' + i['port'] + '|' + time, end="")
            print("|", end="")
            if print_country:
                print(i['country'], end="")
                print("|", end="")
            print(i['whois'])
@| print_results @}
```

To test this side-effect ``function''\footnote{All apologies to any
  Haskell hackers reading this.}, well just check to see that it runs
successfully.

```
@d Obey the... @{
def test_print_results(self):
    with open('testfeed.xml', 'rt') as f:
        doc = parse(f)
        spec = 'http://www.proxyrss.com/specification.html'
        results = extract_proxy_data(doc, spec)
        print_results(results)
        print_results(results, human_readable=False)
        print_results(results, print_country=False, human_readable=False)
@|@}

\section{The Main Program}
We’re finally able to write our main command line program. As usual
for these programs, we’ll check the user option flags and pull the
feeds as appropriate. To keep our {\it \_\_main\_\_} small, we’re
separate the functionality into two driver functions to do the lookups
and print the results. We’ll create one to pull all proxies and
another to filter by country.

\subsection{Pulling All Proxies}
To pull all the proxies, we’ll just get the feed, remove the
duplicates, get the whois, and print the results. We set the default
variables to non—verbose and non—human—readable since we’re assuming
that if you’re going to pull the data from all proxy data, you’ll want
it quietly and with easily grep—able results.

@d Main functions @{
def do_all(feeds, print_country=True, verbose=False,
           human_readable=False):
    """Look up and report the results for all proxies."""
    results = []
    for url in feeds:
        if verbose:
            print("[+] Fetching proxies from: %s" % url)
        results += get_proxy_feed(url, verbose)
    # remove the duplicate entries before doing slow whois
    results = remove_duplicates(results)
    if verbose:
        cprint(’[+] Found %s unique proxies in feeds.’ % len(results),
               ’yellow’)
        print(’[+] Resolving whois descriptions: This can be slow...’)
    final = lookup_whois(results, verbose)
    if verbose:
        cprint(’[+] Total: %s proxies’ % len(results), ’yellow’)
    print_results(final, print_country, human_readable)
@| do_all @}

\subsection{Pull Country Specific Proxies}
The only difference between this and {\it do\_all} is that we also
filter by country data. Note that we default to US given our bias.

@d Main functions @{
def do_country(feeds, print_country=False, verbose=False,
               human_readable=False,
               country_code=’US’, country=’United States’):
    """Look up and report the results for country specific proxies."""
    results = []
    for url in feeds:
        if verbose:
            print("[+] Fetching proxies from: %s" % url)
        results += get_proxy_feed(url, verbose)
        results = filter_by_country(results, country_code, country, verbose)
    # remove the duplicate entries before doing slow whois
    results = remove_duplicates(results)
    if verbose:
        cprint(’[+] Found %s unique proxies in feeds.’ % len(results),
               ’yellow’)
        print(’[+] Resolving whois descriptions: This can be slow...’)
    final = lookup_whois(results, verbose)
    if verbose:
        cprint(’[+] Total: %s %s proxies’ % (len(results), country_code), ’yellow’)
    print_results(final, print_country, human_readable)
@| do_country @}

\subsection{The \_\_main\_\_ Program}
```

Even though we’ve created our two helper ‘‘do’’ functions, since we’re
offering so many different options, this main function is going to be
big. So we’ll break it up into separate parts since most of this is
just straight forward Python {\it argparse}. We’ll setup our default
feeds here and fill in the rest as we go.

```
@d Main functions @{
if __name__ == '__main__':
    feeds = ["http://www.proxz.com/proxylists.xml",
             "http://www.freeproxylists.com/rss",
             "http://www.xroxy.com/proxyrss.xml"]

    @< Parse the command line args@>
    @< Run the program@>
@| feeds @}
```

Next we parse the command line. This is standard {\it argparse}, so
I’m not going to explain it, but rather refer the reader to the
example usage as follows:

```
\newpage
\begin{lstlisting}[style=example]
usage: proxyfeed.py [—h] [—c country] [—d] [—g] [—l] [—u] [—v]

Pull latest data from the open proxy RSS feeds. (file13@@hushmail.me)

optional arguments:
  —h, ——help  show this help message and exit
  —c country  filter by 1 country_code AND 1 country ie: —c "US" —c "United
              States"
  —d          do NOT print country in results
  —g          grep parseable results (delimiter='|')
  —l          print the default proxy feeds list
  —u          only report US proxies (implies —c)
  —v          verbose mode
\end{lstlisting}
```

As for why I chose these defaults, who knows? It seems like the right
thing to do.

```
@d Parse the command... @{
import argparse
parser = argparse.ArgumentParser(
    description="""Pull latest data from the open proxy RSS feeds.
    (file13@@hushmail.me)""")
parser.add_argument('—c', dest='country_specific',
                    metavar='country',
                    action='append',
                    help='filter by 1 country_code AND 1 country ie: —c "US" —c "United States"')
parser.add_argument('—d', dest='print_country',
                    action='store_false',
                    help='do NOT print country in results')
parser.add_argument('—g', dest='human_readable',
                    action='store_false',
                    help="grep parseable results (delimiter='|')")
parser.add_argument('—l', dest='list_feeds',
                    action='store_true',
                    help='print the default proxy feeds list')
parser.add_argument('—u', dest='us_mode',
                    action='store_true',
                    help='only report US proxies (implies —c)')
parser.add_argument('—v', dest='verbose',
                    action='store_true',
                    help='verbose mode')
args = parser.parse_args()
@|@}
```

And finally we run the program with the given flags. We have four

```
options. One to print the proxy feeds, one to do US proxies, one for
country specific proxies, and the default being do all the feeds.

@d Run the program @{
if args.list_feeds:
    print('Default open proxy RSS feeds:')
    for url in feeds:
        print(url)
    quit()
if args.us_mode:
    do_country(feeds,
               print_country=args.print_country,
               verbose=args.verbose,
               human_readable=args.human_readable)
elif args.country_specific:
    if len(args.country_specific) != 2:
        parser.print_help()
    else:
        do_country(feeds,
                   print_country=args.print_country,
                   verbose=args.verbose,
                   human_readable=args.human_readable,
                   country_code=args.country_specific[0],
                   country=args.country_specific[1])
else:
    do_all(feeds,
           print_country=args.print_country,
           verbose=args.verbose,
           human_readable=args.human_readable)
@|@}

And that's it. Happy proxy hunting!

\section{To Do}
Here are our ideas for future expansion.

\begin{itemize}
\item Add to github.
\item Add usage examples.
\item Add threading to whois lookups.
\item Add option to write to a file.
\item Add option to read feeds from a file.
\end{itemize}

\section{Updates}
\begin{enumerate}
\item {\bf August 19, 2015}: Added ``convert\_time'' to normalize
  timestamps. It's not perfect, but it's because the data from the
  feeds is sloppy.
\end{enumerate}

\section{Loose Ends}
\subsection{Literate Code Comment}
Since we ultimately want folks reading the source code to actually
read the literate code instead of the raw source, we'll insert a
comment at the beginning of each of our files that directs them to
this file.

% since we're done with any formatted code, set to raw
\lstset{style=rawcode}
@d Literate code header comment @{
#######################################################################
# This program source is a part of a literate program and was produced
# by nuweb. Please see the accompanying literate program pdf for the
# full documentation.
#
# author: Michael J. Rossi
# contact: file13@@hushmail.me
# literate pdf file: proxyfeed.pdf
######################################################################
@<BSD License@>
@|@}

\subsection{License}
We'll add our two clause ``simplified'' BSD license to each file.

@d BSD License @{
######################################################################
# Copyright (c) 2015, Michael J. Rossi
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions
# are met:
#
# * Redistributions of source code must retain the above copyright
# notice, this list of conditions and the following disclaimer.
#
# * Redistributions in binary form must reproduce the above copyright
# notice, this list of conditions and the following disclaimer in the
# documentation and/or other materials provided with the distribution.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
# "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
# LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
# A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
# HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
# SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
# LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
# DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
# THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
# (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
# OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#
######################################################################
@|@}

\subsection{Test XML File}
Here's our test XML file.

@o testfeed.xml @{<?xml version="1.0" encoding="iso-8859-1"?>
<rss xmlns:content="http://purl.org/rss/1.0/modules/content/"
     xmlns:prx="http://www.proxyrss.com/specification.html"
     version="2.0">
<channel>
    <title>Proxz.com : free, fresh, and fast</title>
    <link>http://www.proxz.com/</link>
    <description>Supplying you with proxies since 2003</description>
    <image>
        <title>Proxz.com : free, fresh, and fast</title>
        <url>http://www.proxz.com/images/88x31.gif</url>
        <link>http://www.proxz.com/</link>
    </image>
    <language>en-us</language>
    <lastBuildDate>Fri, 06 Mar 2015 15:41:26 EST</lastBuildDate>
    <ttl>60</ttl>
    <managingEditor>info@@proxz.com</managingEditor>
    <generator>Proxy RSS Generator v0.1</generator>
    <item>
        <title>High anonymous proxies on 2015/03/6 20:41:26</title>
        <link>http://www.proxz.com/proxy_list_high_anonymous_0.html</link>
        <description>1339 proxies are in the database at the moment.</description>
        <guid>http://www.proxz.com/proxy_list_high_anonymous_0.html</guid>
        <prx:proxy><prx:ip>183.223.173.237</prx:ip><prx:port>8123</prx:port><prx:type>Anonymous</
    prx:type><prx:country>China</prx:country><prx:check_timestamp>03/6 20:23:47</prx:
    check_timestamp></prx:proxy>
```

```
        <prx:proxy><prx:ip>183.221.208.44</prx:ip><prx:port>8123</prx:port><prx:type>Anonymous</
    prx:type><prx:country_code>CN</prx:country_code><prx:check_timestamp>03/6 20:21:37</prx:
    check_timestamp></prx:proxy>
        <prx:proxy><prx:ip>183.221.160.12</prx:ip><prx:port>8123</prx:port><prx:type>Anonymous</
    prx:type></prx:proxy>
        <prx:proxy><prx:ip>32.21.19.4</prx:ip><prx:port>8080</prx:port><prx:type>Anonymous</prx:
    type><prx:country_code>US</prx:country_code><prx:check_timestamp>03/6 20:21:37</prx:
    check_timestamp></prx:proxy>
        <prx:proxy><prx:ip>241.2.28.99</prx:ip><prx:port>8080</prx:port><prx:type>Anonymous</prx:
    type><prx:country>United States</prx:country><prx:check_timestamp>03/6 20:21:37</prx:
    check_timestamp></prx:proxy>
        <prx:proxy><prx:ip>183.221.160.12</prx:ip><prx:port>8123</prx:port><prx:type>Anonymous</
    prx:type><prx:check_timestamp>03/6 20:21:34</prx:check_timestamp></prx:proxy>
</item>
</channel>
</rss>
@|@}


\newpage
\section{Makefile}
Here's the raw makefile for this program.
\includetextfile{Makefile}

\newpage
\section{Final Python Source Files}
Here is the actual weaved Python source code.
Other then the line numbers
(which we've added), this is the raw Python source code.

\subsection{proxyfeed.py}
\includenumberedtextfile{proxyfeed.py}

\subsection{tests.py}
\includenumberedtextfile{tests.py}

\newpage
\section{Raw Nuweb File}
Here is the raw nuweb file that produced this for folks curious as to
what this looks like or in the strange event where this PDF is all
that remains of the program and someone wanted to reconstruct it.
Otherwise, you'll want to ignore this.
\includetextfile{proxyfeed.w}

\newpage
\section{Index} \label{indices}
{\bf Nuweb} automatically creates three sets of indices: an index
of file
names, an index of fragment names, and an index of user—specified
identifiers. An index entry includes the name of the entry, where it
was defined, and where it was referenced.

\subsection{Files}

@f

\subsection{Identifiers}

@u

\subsection{Fragments}

@m

\end{document}
```

17 Index

Nuweb automatically creates three sets of indices: an index of file names, an index of fragment names, and an index of user-specified identifiers. An index entry includes the name of the entry, where it was defined, and where it was referenced.

17.1 Files

`"proxyfeed.py"` Defined by 6.
`"testfeed.xml"` Defined by 22.
`"tests.py"` Defined by 5.

17.2 Identifiers

`convert_time`: 14b, 15ab.
`do_all`: 16b, 19.
`do_country`: 17a, 19.
`extract_proxy_data`: 7, 8, 9a, 10b, 13b, 14a, 16a.
`feeds`: 9a, 16b, 17a, 17b, 18, 19.
`filter_by_country`: 10a, 10b, 17a.
`get_proxy_feed`: 9a, 9b, 10a, 16b, 17a.
`get_whois_description`: 11a, 11b, 12a.
`is_ip_in_dict`: 13a, 13bc.
`lookup_whois`: 12a, 12b, 16b, 17a.
`print_results`: 15b, 16ab, 17a.
`ProxyFeedTest`: 5.
`remove_duplicates`: 13c, 14a, 16b, 17a.
`test_extract_proxy_data`: 8.

17.3 Fragments

⟨BSD License 21⟩ Referenced in 20.
⟨Literate code header comment 20⟩ Referenced in 5, 6.
⟨Main functions 7, 9a, 10a, 11a, 12a, 13ac, 14b, 15b, 16b, 17ab⟩ Referenced in 6.
⟨Obey the testing goat! 8, 9b, 10b, 11b, 12b, 13b, 14a, 15a, 16a⟩ Referenced in 5.
⟨Parse the command line args 18⟩ Referenced in 17b.
⟨Run the program 19⟩ Referenced in 17b.