week 7 lab 2

Kevin Turkington

I. Reasoning

A. Regex

Immediately after reading the lab assignment I created a regex searcher and looked for all matches of the string google and nothing else within the URLs. This was because google is one of the top most used websites on the internet and it would make sense to trick users with variations of the url. After analyzing the results I found a majority of the malicious urls contain a variation of google with its popular extensions appended on to the string. Other than urls containing ”googleblogspot” where almost always a match for a malicious URL, So this was weighted heavily in my classifier.

Next I scanned for other fields for malicious patterns that could be regexifyed. I found that in most cases malware is hosted on personal servers without a domain attached to the host. However only half the time a host with a only ip host was malicious in the training set, so this field was rated low in terms of the threshold.

B. Schemas

Schema/protocol of the URL is extremely important in classification however after scanning some of the malicious URL schemas I found some of them where authority certified. Allowing them to receive an SSL layer to their packets and spoof legitimacy. So this received a negative score to the URL maliciousness threshold. Since there is only two protocols in the training and classification sets this threshold received a smaller weight.

C. Misc

Next I continued to mull over the results from malicious and non malicious urls within the training set and found a path that obnoxiously long urls tended to be malicious. While on average anything past 30 characters in length had a much greater chance of being malicious. However this was mainly an observation and this received a fairly low weight. Furthermore from the hints on the lab assignment I started looking for patterns to find how old domains are for malicious and non malicious urls. After some sort averaging I came to the conclusion anything that has been active for less than 6 months tend to be malicious and recieved a faily high weight for this. And finally I checked for alexa weights and heavily add points to the threshold for no ranking and removed point for being within the first million.

D. Classifier Weights

Overall my plan of attack for classifying urls where to add to my malicious threshold heavily at the sacrifice gaining more false positives. While removing from that score lightly for having positive patterns for non maliciousness. For example links that serve non malicious content like xml.

II. Code

```
#!/usr/bin/python

import json, sys, getopt, os, re

def usage():
  print("Usage: %s --file=[filename]" % sys.argv[0])
  sys.exit()

def main(argv):

  file=''

  myopts, args = getopt.getopt(sys.argv[1:], "", ["file="])

  for o, a in myopts:
    if o in ('-f, --file'):
      file=a
    else:
      usage()

  if len(file) == 0:
    usage()

  corpus = open(file)
  urldata = json.load(corpus, encoding="latin1")


  totalUrls = 0.0
  myMaliciousCount = 0.0
  validMalicious = 0.0
  thresholdMax = 500

  for record in urldata:
    threshold = 0
    totalUrls+=1.0
    malicousBit = 0
```

```
regexDl = re.search("[^(www\.)]google(docs|doc|drive|mail)*",record["
regexHostIp = re.match("^(\.[0-9][0-9]?[0-9])+$",record["host"]) #onl

if record["scheme"] == "https":
  threshold -= 300

ext = record["file_extension"]
if ext in ["zip","php"]:
  threshold += 300
elif ext == "exe":
  threshold += 1500
elif ext in ["aspx", "xml"]:
  threshold -= 200

if record["path_len"] > 30:
  threshold += 200

# if record["malicious_url"]:
#    validMalicious+=1.0

if regexDl:
  threshold += 1000

if regexHostIp:
  threshold += 300

domainAge = int(record["domain_age_days"])  #domain age less than hal
if domainAge < 180:
  threshold += 800

alexaRankNotExist = record["alexa_rank"] == None
if alexaRankNotExist:
  threshold += 1000
elif int(record["alexa_rank"]) < 1000000:
  threshold -= 200

if threshold > thresholdMax:
  myMaliciousCount += 1.0
  malicousBit = 1

#print (")%s \n \tmalicous: %s") % (record["url"],record["malicious_u
print ("%s,␣%s") %(record["url"], malicousBit)
```

```
    print "real malicous: %f" % (validMalicious / totalUrls)
    print "My malicous: %f" % (myMaliciousCount / totalUrls)
    corpus.close()

if __name__ == "__main__":
    main(sys.argv[1:])
```

III. Results