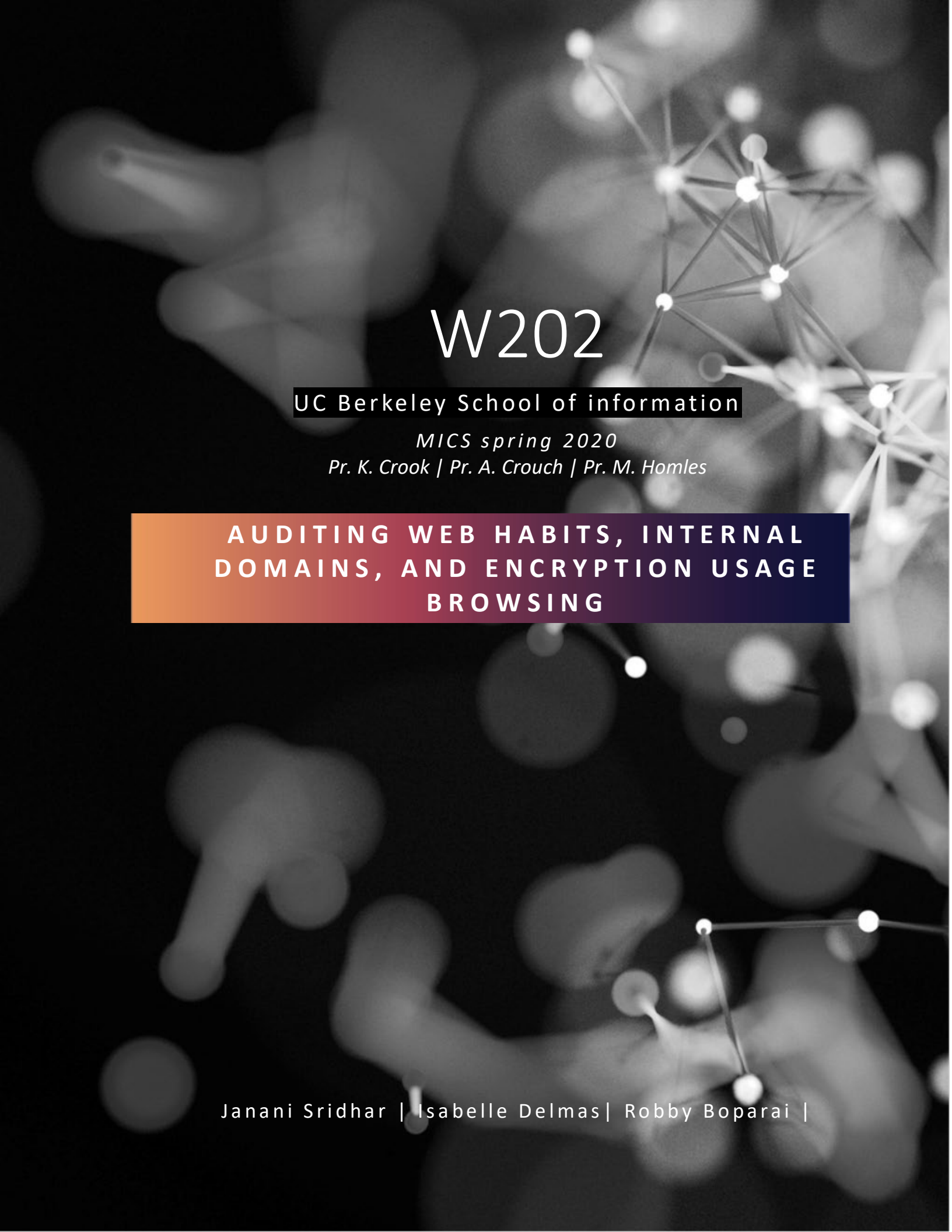


W202

UC Berkeley School of information

MICS spring 2020

Pr. K. Crook | Pr. A. Crouch | Pr. M. Homles

AUDITING WEB HABITS, INTERNAL DOMAINS, AND ENCRYPTION USAGE BROWSING

Janani Sridhar | Isabelle Delmas| Robby Boparai |

SUMMARY

In today's connected world, encryption is primary defense for companies to protect their intellectual property.
After weeks studying the mathematical theories behind symmetric and asymmetric encryption algorithms but also how flaws in algorithms and implementations can compromise even the strongest mathematical theories behind encryption, we set ourselves to audit the actual usage of encryption standards in employee's Internet and Intranet connection.

INTRODUCTION

ORIGINAL PROMPT

Write code to audit all the internal websites at a company. Read web logs from the proxy servers and audit all the websites employees have visited. Your audit can make sure they are using valid digital certificates and TLS 1.3 with appropriate symmetric and asymmetric protocols.

PROJECT PROPOSAL

We extended on the original prompt to create a more complete security tools to audit some security best practice in a company's intranet and the identify the security risks caused by employees' Internet browsing habits.

Assumption:

- We are given a list of URLs/IPs accessed by employees
- We are given the IPs/ports other internal resources to audit
- We will not evaluate resources requiring authentication

The tool was baptized sturdy-couscous because, why not?

Figure 1

PREPARATION

SETUP

- We generated our browsing history, exported as a csv file to use as initial list of URLs
- To ensure consistent environment on different machines, we used docker containers for running our tool: one container for the program itself and one container to run MongoDB used to store our data
- We used the set of 100 popular websites to trainer the Classifier and labeled them with 6 categories: media, education, government, IT/dev, commerce, news, other

TECHNOLOGY STACK

- Python 3.9 with:
  - SSL → obtain security context
  - BeautifulSoup → parsing HTML
  - NLTK → natural language processing
  - Pandas → statistical analysis
  - PyMongo → interaction with MongoDB
- Docker container and network:
- MongoDB
- Github
- DockerHub

SOFTWARE ARCHITECTURE

See Annex 1 for an architecture Diagram.
We split Sturdy Couscous into 5 pieces:

- a web parser to look for URL children (ie, ULRs in a page we accessed)
- a connection checker that checks TLS and certificate information
- a classifier to categorize the type of website
- and a Mongo DB Client to store our results.
- a printer to print a data analysis report based the data collected

IMPLEMENTATION

WEB PARSER

First, we wrote a basic web parser that returns two child links from a web page. We identified these links by using the BeautifulSoup (bs4) library in Python to look for <a> tags in the HTML content returned from GET requests. In order to prevent infinite crawling, we only looked for a site's immediate child links, and only returned two links.

CONNECTION CHECKER

We also wrote a connection checker to obtain the TLS versions supported for a website, validate the site's certificate, scan for open ports, and return a list of ciphers supported by the website. Originally, we had wanted to focus on ensuring that sites are using TLS v1.3. However, upon empirical testing, we found that many popular sites only support up to TLS v1.2, so we broadened our scope to include all the TLS versions supported. We did not include information on SSLv2/3 since the SSL library does not include support for anything lower than TLS v1.0.

We chose the common port based on network services that come or use to come pre-installed on network devices and servers and do not follow today's encryption best practices such as FTP (port 20, 21) which uses user credential for authentication but do not use SLL to encrypt network packet. We selected: 20, 21 (FTP) , 69(TFTP), 80 (HTTP), 123 (NTP), 8080(HTTPS, for webservers forgotten on an Intranet), 389 (LDAP without SSL).

CLASSIFIER

We wrote a Classifier to categories websites, which would allow to highlight security risk based on the kind of websites accessed from a company's network or company's computers. As a proof of concept, we chose social media, commerce, government, news, education, and IT.

Website classification is a fairly complex problem that has attracted research for at least 2 decades and could still be a subject of a full PhD thesis so rather than solving this complex puzzle we decide to create a classifier using on supervised learning and the static content of how a page renders (i.e., HTML). We used the implementation suggestions from Sara-Meshkizadeh published in the International Journal of Advancements in Computer Technology in 2010 (DOI: 10.4156)

Training:

- Label a set of known websites with the appropriate category
- Parse the website description and keywords from the HTML header and extract keywords
    - Split into words
    - Filter Punctuation
    - Remove stop words (common words of the English languages)
    - Stem Words (ie, eating → eat)
    - Remove capitalization
- Create weighted list of keywords for each category

Classification:

- Extract keywords from the more relevant HTML tags using Sara-Meshkizadeh's paper
    - Same data cleaning process as above
- Score each category "IT": ∑(weight of keywords present in IT training keywords)/num_os_keywords
- Check for .edu and .gov

MONGODB

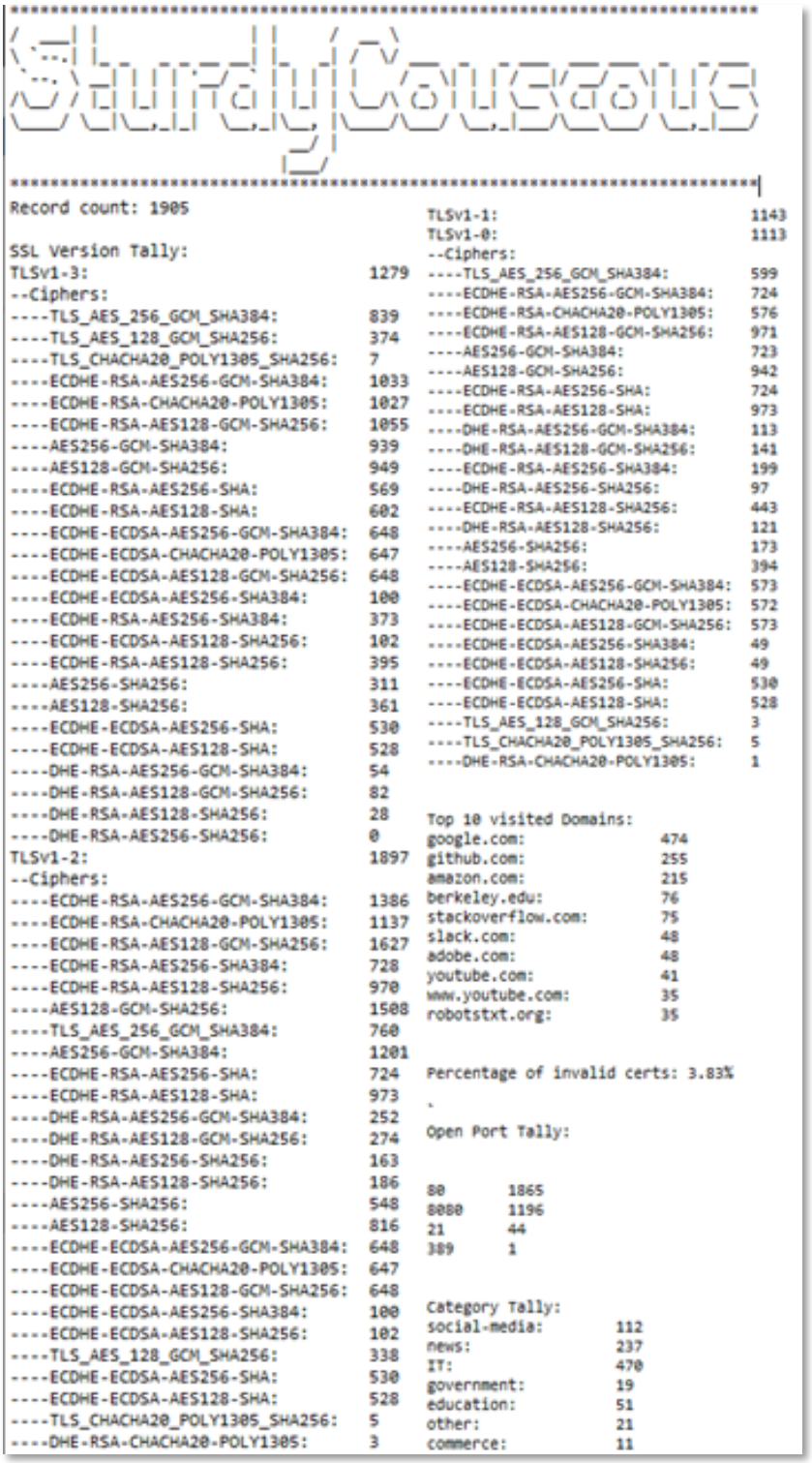
We wrote our findings to MongoDB. We chose a NoSQL database for flexibility and the statistical built-in methods. Our database maps a URL to its connection and classification information. After writing to MongoDB, we have a Printer to tally how often each TLS version was supported, what the most used cipher is, and which one of the common ports were left open.

PRINTER

Once the data has been inserted into Mongo, it must be read and aggregated for any useful analyses. The pymongo package does provide some support for basic statistical analysis, but some additional work was necessary in the aggregation of TLS version and cipher information. A Printer module is invoked at the end of a run to query the database and output organized data.

RESULTS

Output shown below:



```
Record count: 1905

SSL Version Tally:
TLSv1-3:                                    1279
--Ciphers:
----TLS_AES_256_GCM_SHA384:                 839
----TLS_AES_128_GCM_SHA256:                 374
----TLS_CHACHA20_POLY1305_SHA256:           7
----ECDHE-RSA-AES256-GCM-SHA384:            1033
----ECDHE-RSA-CHACHA20-POLY1305:            1027
----ECDHE-RSA-AES128-GCM-SHA256:            1055
----AES256-GCM-SHA384:                      939
----AES128-GCM-SHA256:                      949
----ECDHE-RSA-AES256-SHA:                   569
----ECDHE-RSA-AES128-SHA:                   602
----ECDHE-ECDSA-AES256-GCM-SHA384:          648
----ECDHE-ECDSA-CHACHA20-POLY1305:          647
----ECDHE-ECDSA-AES128-GCM-SHA256:          648
----ECDHE-ECDSA-AES256-SHA384:              100
----ECDHE-RSA-AES256-SHA384:                373
----ECDHE-ECDSA-AES128-SHA256:              102
----ECDHE-RSA-AES128-SHA256:                395
----AES256-SHA256:                          311
----AES128-SHA256:                          361
----ECDHE-ECDSA-AES256-SHA:                 530
----ECDHE-ECDSA-AES128-SHA:                 528
----DHE-RSA-AES256-GCM-SHA384:              54
----DHE-RSA-AES128-GCM-SHA256:              82
----DHE-RSA-AES128-SHA256:                  28
----DHE-RSA-AES256-SHA256:                  0
TLSv1-2:                                    1897
--Ciphers:
----ECDHE-RSA-AES256-GCM-SHA384:            1386
----ECDHE-RSA-CHACHA20-POLY1305:            1137
----ECDHE-RSA-AES128-GCM-SHA256:            1627
----ECDHE-RSA-AES256-SHA384:                728
----ECDHE-RSA-AES128-SHA256:                970
----AES128-GCM-SHA256:                      1508
----TLS_AES_256_GCM_SHA384:                 760
----AES256-GCM-SHA384:                      1201
----ECDHE-RSA-AES256-SHA:                   724
----ECDHE-RSA-AES128-SHA:                   973
----DHE-RSA-AES256-GCM-SHA384:              252
----DHE-RSA-AES128-GCM-SHA256:              274
----DHE-RSA-AES256-SHA256:                  163
----DHE-RSA-AES128-SHA256:                  186
----AES256-SHA256:                          548
----AES128-SHA256:                          816
----ECDHE-ECDSA-AES256-GCM-SHA384:          648
----ECDHE-ECDSA-CHACHA20-POLY1305:          647
----ECDHE-ECDSA-AES128-GCM-SHA256:          648
----ECDHE-ECDSA-AES256-SHA384:              100
----ECDHE-ECDSA-AES128-SHA256:              102
----TLS_AES_128_GCM_SHA256:                 338
----ECDHE-ECDSA-AES256-SHA:                 530
----ECDHE-ECDSA-AES128-SHA:                 528
----TLS_CHACHA20_POLY1305_SHA256:           5
----DHE-RSA-CHACHA20-POLY1305:              3
TLSv1-1:                                    1143
TLSv1-0:                                    1113
--Ciphers:
----TLS_AES_256_GCM_SHA384:                 599
----ECDHE-RSA-AES256-GCM-SHA384:            724
----ECDHE-RSA-CHACHA20-POLY1305:            576
----ECDHE-RSA-AES128-GCM-SHA256:            971
----AES256-GCM-SHA384:                      723
----AES128-GCM-SHA256:                      942
----ECDHE-RSA-AES256-SHA:                   724
----ECDHE-RSA-AES128-SHA:                   973
----DHE-RSA-AES256-GCM-SHA384:              113
----DHE-RSA-AES128-GCM-SHA256:              141
----ECDHE-RSA-AES256-SHA384:                199
----DHE-RSA-AES256-SHA256:                  97
----ECDHE-RSA-AES128-SHA256:                443
----DHE-RSA-AES128-SHA256:                  121
----AES256-SHA256:                          173
----AES128-SHA256:                          394
----ECDHE-ECDSA-AES256-GCM-SHA384:          573
----ECDHE-ECDSA-CHACHA20-POLY1305:          572
----ECDHE-ECDSA-AES128-GCM-SHA256:          573
----ECDHE-ECDSA-AES256-SHA384:              49
----ECDHE-ECDSA-AES128-SHA256:              49
----ECDHE-ECDSA-AES256-SHA:                 530
----ECDHE-ECDSA-AES128-SHA:                 528
----TLS_AES_128_GCM_SHA256:                 3
----TLS_CHACHA20_POLY1305_SHA256:           5
----DHE-RSA-CHACHA20-POLY1305:              1


Top 10 visited Domains:
google.com:                 474
github.com:                 255
amazon.com:                 215
berkeley.edu:               76
stackoverflow.com:          75
slack.com:                  48
adobe.com:                  48
youtube.com:                41
www.youtube.com:            35
robotstxt.org:              35


Percentage of invalid certs: 3.83%

.

Open Port Tally:


80      1865
8080    1196
21      44
389     1


Category Tally:
social-media:       112
news:               237
IT:                 470
government:         19
education:          51
other:              21
commerce:           11
```

The aggregation was performed over 1,905 records, tallying not only the TLS versions used by each URL hit, but the ciphers supported by each version. T

One noteworthy aspect of this data is the evident backwards compatibility of SSL versions for the majority of most of the sites that we scanned. Nearly every host negotiated TLSv1.2, but most are also able to negotiate over deprecated versions of SSL.

Another fascinating aspect of this information is the inconsistency in cipher suite availability across hosts. Given that cryptographic libraries are generally very standardized across platforms, we could potentially infer information or correlations between different cryptographic libraries used for TLS negotiation.

Additionally, we tallied the top visited domains from our own search histories, and conducted a portscan across every host. Notably, it seems that 44 hosts had the standard FTP port open, and one had port 389 open, which is traditionally used for LDAP.

Lastly, we leveraged Natural Language Toolkit (NTLK) in Python to preemptively create a training dataset that the core scanner uses to classify content based on linguistic patterns.

FURTHER WORK

IMPLEMENTATION

The following are improvement that we would have added in a POC or final product given more time and human resources.

- IPv4/6 range scanning
- Differentiating the Internet URL/IPs from the Intranet URLs/IPs to adapt checks to context
- Classifier:
    - using more features
        - keywords in the URL domain, subdomain, or directory
        - keywords in the HTML response to get request
        - Similarities to domain/url already evaluated or from training set
        - Expanding URLs symbols (nyt-->NewYorkTimes)
        - Considering frequent nominal groups as keywords (ie, New York Times)
        - Match nominal groups to abbreviations or the same group without spaces (ie New York Time <-> nyt <-> newyorktimes)
        - Use children's features to improve classification
    - Accuracy:
        - Larger training set
        - Keep 10% of training set to validate result accuracy
- DNS caching: most internet services today run some flavor of a UNIX operating system, which does not have native DNS caching abilities. Adding DNS caching to our tool could drastically improve performance as some domains are visited frequently. Unbound, for example, would have been an option for a python program.

TAKE AWAYS

More than how the usage of the current encryption standards, we learned some valuable lessons that we will take with for future projects:

- Threading in docker is a bad idea
- Port scanning is a costly operation
- Use a VPN – During our first attempt at scaling our program, Isabelle's IP got blacklisted either by her ISP or relay between her and the global Internet network, which caused her to have an Internet outage lasting a few hours on Tuesday. **Always use a VPN when doing web scraping or programmatically setting up many connections**.
- Do not attempt PhD worth problems in a 5 weeks side project while working full time and being confined with kids/dogs/parents that expect you to have a life

SOURCES

NLP data processing:

- https://machinelearningmastery.com/clean-text-machine-learning-python/

Webpage Classification:

- Web Page Classification based on Compound of Using HTML Features & URL Features and Features of Sibling Pages" by Sara-Meshkizadeh and Dr.Amir Masoud-Rahmani (International Journal of Advancements in Computing Technology Volume 2, Number 4, October 2010 | doi: 10.4156/ijact.vol2.issue4.4

DNS caching:

- https://nlnetlabs.nl/documentation/unbound/howto-setup/

APPENDIX 1

ARCHITECTURE DIAGRAM