Natural Language Processing for the Security Analyst

Daniel Severance

Natural Language Processing for the Security Analyst

GIAC (GMON) Gold Certification

Author: Daniel Severance, d.S3VERANCE@gmail.com
Advisor: *David Hoelzer*

Accepted: *5/5/2020*

Abstract

Data science is an emerging multidisciplinary field that offers multiple benefits to information security. Within this field, there is an inherent ability to do anomaly detection at scale. Recently there are increased efforts in applied data sciences in the field of information security and assurance, however there can be a high barrier to entry due to the mathematics required. Nonetheless, topics such as natural language processing can be and have been integrated into security toolsets successfully. These computational linguistic methods can effectively be used to empower analysis techniques. This paper examines the viability of applying these language techniques in security anomaly detection and the ability to integrate with existing security tools.

1. Introduction

Much like the field of information security, natural language processing has largely emerged in the last seventy years alongside the explosive growth of computational sciences and engineering. Security-related fields such as cryptography have history in works such as the Caesar substitution cipher and the "Manuscript for Deciphering Cryptographic Messages" by Al-Kindi. These same foundations in combinatorics, frequency analysis, and data classification underscore fundamental information science and text processing techniques (Broemeling, 2011).

The 1980s and 1990s gave rise to statistics-based natural language processing, which relied on statistical models to make probabilistic decisions within elementary machine learning models. As an interdisciplinary subject, the field of computational linguistics borrowed heavily from the existing domains of linguistics, mathematics, and statistics. With the facilitation of machine learning and artificial intelligence, modern natural language processing was born.

Classically, areas such as machine learning and data science have a high barrier to entry due to the educational background required as a prerequisite to domain knowledge. Natural language processing draws on many unique fields and disciplines. For this reason, no one person is expected to be an expert in all of these fields in addition to the security domain. Since linear algebra and multivariate calculus are fundamental to current machine learning approaches, all academic papers in this field assume a mastery of these topics as a prerequisite. This paper presents an applied approach covering highlights of relevant theory to offer a more direct application. Mathematical and statistical concepts are displayed by model and algorithm where applicable; however, the research principles focus on application to the security domain. While no formal training in linguistics is required to digest this paper, natural language rules are compared and contrasted to structured languages often found in the security realm.

Entry barriers withstanding, fields like natural language processing and text mining share many of a security analyst's existing skillsets and facilitate additional analysis techniques. Yet as security scales with big data and the digital age, static rules in security tools must scale too. Often, scaling static detections comes at an increased cost to the security engineers and analysts in the form of additional constraints to maintain.

Some data formats may assist in scalability. Text is fundamentally discrete in nature, such that its meaning is created through ordered combinatorial arrangements of symbolic units. The discrete nature of text facilitates many accessible mathematical analysis methods that can be borrowed from other domains. This paper attempts to research and outline natural language processing techniques and tools leveraging these methods and how they can be embedded into existing analysis processes in the security space.

2. Related Works

The natural language processing field contains numerous tool suites and research focuses. Due to the multidisciplinary nature of the field, toolsets may contain some specialization, and preference may differ within domains. The cited tools herein will not be used in-depth; however, it should be noted that in many cases, these tools contain functions and structures to assist in natural language processing natively.

At the time of publication, Python remains one of the most utilized languages in the natural language processing and data science space. Natural Language Toolkit (NLTK) is one of the best known and full-featured tools available. NLTK contains higher levels of flexibility at the cost of complexity through a rich feature set (Bird, 2019). Originally written by Matthew Honnibal, and now maintained by Explosion AI, SpaCy is one of the primary competitors to NLTK, boasting better performance at scale. SpaCy streamlines complexity by simplifying the implementation of NLP concepts. Reduced feature scope may impact some use-cases; however, object-based representation of data may assist in analysis techniques (Honnibal, 2020).

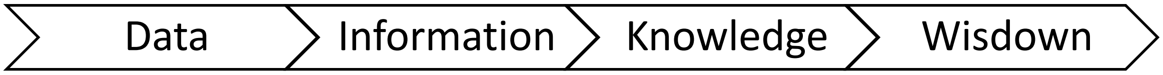
The SciPy stack facilitates further exploration in the mathematics and sciences involved in NLP . The SciPy stack, and by extension, the common python machine learning stack, consists primarily of libraries such as NumPy, SciPy, Matplotlib, Jupyter, and pandas. This Python stack can holistically process, analyze, model, and visualize data (Scientific computing tools, 2020). Each of these tools is worth research in the security realm under its own merit. Tools like Jupyter are being utilized in security tools such as the threat hunting elastic stack: HELK (Rodriguez, 2019).

Lastly, natural language processing has many overlaps with machine learning and artificial intelligence domains. TensorFlow remains one of the most popular machine learning platforms with flexible tools and libraries and an active community (Tensorflow, 2020) (Chollet, 2019). Developed by Facebook, PyTorch is a competitive alternative to Tensorflow (Paszke, 2020). Based on the SciPy stack, Scikit-learn balances the machine learning space. While less featured than the last two examples, Scikit-learn provides simple tools for data mining and analysis (Scientific computing tools, 2020).

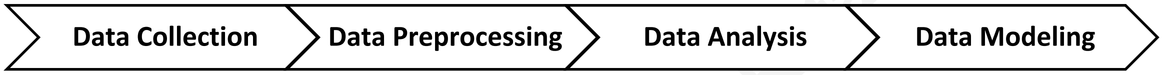
However, the application and documentation of natural language processing in the information security domain are still rather new. Analysis and application techniques are found in discrete applications such as phishing, spam detection, and domain generation algorithms. Markov models present usable techniques in both the creation and detection of phishing messages. An example application is the SNAP_R tool presented at BlackHat 2016 (Seymour & Tully, 2016). Utilizing conditional entropy, other tools such as freq by Mark Baggett offer efficient and accessible methods to measure the likelihood of domain names (Baggett, 2019). Expanded into engineering techniques, NLP is used extensively in information sciences for document classification (Brown & Charlebois, 2010).

3. Data Handling Ontology

Information sciences model the structural relationship of data, information, knowledge, and wisdom through the DIKW hierarchy (Rowley, 2007). The knowledge hierarchy closely mirrors the intelligence extraction process used in security and intelligence analysis. This transformation extracts information from raw data, and from this information, knowledge, or intelligence, is derived. Lastly, through the application of this knowledge, wisdom, or action, is derived. Within both natural language processing and security analysis, a rough form of the data handling ontological model is as follows:



Similar models extend the extraction practice into natural language processing and text mining. Focused primarily on the transformation of data and information, these models follow standard transformation methodologies. This transformation can largely be represented within the following four processes:



For the purpose of this paper, the data collection phase classically relies on security concepts and toolsets. The extraction ontology is applied to prevalent information formats and models such as the Open Systems Interconnection model (OSI), Transmission Control Protocol (TCP), and Syslog. It is imperative to note that outside the context of the text mining process, each security tool follows a similar data handling pipeline. During the analysis process, the practitioner may insert themselves into multiple stages of data handling. For information security, the arbitrary insertion holistically applies to the data handling process in three ways:

- *Analysts must treat all data as potentially hostile during processing*
- *Data may be sanitized, normalized, or otherwise processed by tools before ingestion*
- *Data may be duplicated at multiple levels of the OSI model or from multiple tools.*

This paper explores these three tenets in varying degrees through the preprocessing and analysis phases. Data collection has been assumed. Throughout preprocessing and analysis, the analyst can interpret data from multiple stages of the data flow and from multiple logical layers of a given protocol stack. In the context of security, the same authentication event may be present in centralized and distributed event logs, firewall logs, network flows, full packet captures, endpoint detection and response (EDR) process flows, and more. The languages presented to the analyst may include not only natural human languages, but constructed languages, encodings, and formats such as HyperText Markup Languages (HTML), Base64, and hexadecimal numerals as well. The knowledge ontology must be applied such that context is maintained for both the analyst and the parties interpreting the data ingested.

4. Data Preprocessing

4.1. Removing Noise

One of the most meaningful ways of extracting value is by removing noise. Within the text mining ontology, signal can be implicitly defined as the data relevant to the hypothesis. In generalized terms, the hypothesis is our supposition acting as our starting point for further investigation. These educated guesses provide the information to pivot the investigation alongside further evidence. Within security and intelligence analysis, the hypothesis is comprised of artifacts from the tactics, techniques, and procedures (TTPs) utilized by threat actors (Johnson et al., 2016). For models like the Pyramid of Pain, artifacts in the form of hashes and IP addresses can be used to identify malicious threat actors (Bianco, 2013). Given the previous definition of hypothesis, noise is any data that may introduce unknown or uncontrolled variability therein.

Language structure in the form of protocol is intimately linked to noise management. Within internet protocols, IPv6 and IPv4 use different delimiters in defining signal. It is crucial to retain the hypothesis during the noise removal process as the hypothesis determines which data are signal in which cases. Of the types of noise found within the data, external noise is often derived from the data carrier. Structured languages and protocols such as HTML and TCP encapsulate and codify communications per their defined standards. While more explicit than their natural counterparts, the structured languages present in information security have one major benefit over true natural languages: data classifiers are often built directly into protocols and standards. Analysts may leverage these standards to isolate targeted data in preprocessing.

Due to the consistent and predictable nature of these protocols, it is best to use designated parsers where applicable (Chifflier & Couprie, 2017). While due diligence should be given, these parsers typically handle standard specifications such as RFCs better than ad-hoc development. Within language theory, the concept of comprehension arises in the Chomsky hierarchy: a containment hierarchy for formal grammars (Chomsky, 1956). Simply stated, less complex grammars cannot recognize the language classes of more complex grammars. Those familiar with this concept or wishing to view mitigations directly may jump to below Figure 1.

Otherwise, within Chomsky's hierarchy, a Type-3 "Regular" grammar is that which is found in regular expressions. Present in formats such as raw text, or Comma Separated Value (CSV), Type-3 grammars are flat in architecture and contain up to two dimensions in their data structure, often visibly displayed in a tabular format. The planar structure constrains depth via concepts such as nesting or recursion. Formally, Type-3 grammars can be read using finite state automata, further discussed in section 5.3 Finite State Machines.

As the hierarchy is ascended, Type-2 "Context-free" grammars are those containing abstract syntax. In the computing realm, Type-2 grammars contain languages such as Extensible Markup Language (XML) and HTML. Unlike Type-3 grammars, Type-2 grammars may include dimensional depth through concepts like nesting or recursion. These languages are often displayed in terms of abstract syntax trees. Due to depth, regular parsers cannot process these languages directly; nesting instead requires a stack to process ingested data at the proper interval. While not covered under the extent of this paper, Type-2 grammars are formally read by Deterministic Pushdown Automata.

These rules come with some exceptions. In the case of Type-2 grammars, the language may be considered a Type-1 "Context-sensitive" grammar when applied to the external system, as languages such as HTML may be rendered or translated differently upon some external context. A conventional security scenario to facilitate conditional comprehension exists within the intrusion detection system. Provided network traffic with failed redundancy checks, the IDS must process the possible semantic result with and without the non-compliant payload, as the reconstruction process of the intended recipient is unknown (Ptacek & Newsham, 1998).

Conversely, Type-2 grammars may still be flat in structure. It is also possible that the semantic meaning of the signal does not change upon the removal of the syntax such as within the two HTML excerpts in Figure 1 below:

```
<div>
   <h1> First element </h1>
   <h1> Second element </h1>
   <h1> Third element </h1>
</div>
```

```
<h1> First element </h1>
<h1> Second element </h1>
<h1> Third element </h1>
```

Figure 1
Example Parity in HTML Nesting

In both of these cases, regular grammar may still successfully extract signal. Where possible, the analyst should frame data such that grammar and state dependency is removed (e.g., flattening the data). The case for flattening can be represented in HTML with the Beautiful Soup library originally created by Leonard Richardson (2019). Provided a known, limited subset of HTML, it is possible to strip tags through regular expressions in tools such as sed or egrep provided in most UNIX distros. Alternatively, methods such as the beautiful soup python library parse at the language level. These methods have been represented in Figure 2 below:

```
#!/usr/bin/bash
egrep -i ‘<[^>]*>’ html_doc
```

```
#!/usr/bin/python
from bs4 import BeautifulSoup
soup = BeautifulSoup(html_doc, 'html.parser')
print(soup.get_text())
```

Figure 2
Extraction of Text Data from HTML

Both methods are valid to the security analyst. While fidelity requirements and operational constraints may ultimately decide the target method, parsers often provide additional object-based structures that may assist in data extraction. The philosophy of optimizing interpretation extends to other familiar information security data formats. When applied to packet manipulation tools such as Scapy, originally created by Biondi et al., data can be implicitly structured, even at the byte level (2019).

The second form of noise within the raw data originates from internal sources. The payload data itself contains noise. In the case of structured languages and data, analysts can utilize explicit classifiers to remove noise. Given protocol and standard definitions such as byte offsets in TCP, or key/value pairing in syslog formats like Common Event Format (CEF) and Log Event Extended Format (LEEF), signal may be reliably extracted. The concept of noise as variability arises in this segment. As discussed further within section 5.2 Entropy, classifiers may not necessarily contribute to noise. As the uniformity of a format approaches a value of one, the amount of variability, or randomness, inherent in that value approaches zero. In the example of key/value pairs, the uniformity of the same key provides no additional information about the values without external contextual information.

The concept of uniformity extends mathematically to other concepts in noise reduction. By specification RFC 793, TCP reserves 8 bits to represent the control flags in the protocol (Postel, 1981). Defined later within standard RFC 3168, Explicit Congestion Notification (ECN) provides additional control flags to TCP in the form of ECN-Echo (ECE) and Congestion Window Reduced (CWR) (Ramakrishnan, Floyd, & Black, 2001). However, in the context of the TCP three-way handshake, the congestion notification flags may provide variability in state while providing the analyst with no additional semantic information. The intrusion analysis and network security spaces often handle variability at low layers in the form of bit-masking and bitwise operation. Berkeley Packet Filter (BPF) facilitates offsets and bitwise operations to support the masking methodology (McCanne & Jacobson, 1993). Located at the 13th byte offset, the TCP header presents the control flags in big-endian format. Masking utilizes the bitwise AND operation, which results in a binary 0 regardless of input. Analysts can apply bitwise operations to flags such that only the signal remains.

	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	
input	0	0	1	1	1	0	0	0	= 0x38
mask	0	0	0	1	0	0	0	1	= 0x11
result	0	0	0	1	0	0	0	0	= 0x10

Figure 3
Bitwise Masking

In the above Figure 3, the analyst attempts to review a packet containing the URG, ACK, and PSH flags. When limiting the scope to FIN and ACK flags, the hexadecimal mask 0x11 can be created from the representation of the two flags. A bitwise AND operation results in true only if the input and mask are true, resulting in the 0x10 above with only the initial ACK qualifying. BPF holistically represents this filter as **tcp[13] & 0x11** consisting of both byte offset for tcp flags and the target bitmask.

Lastly, analysts can still filter arbitrary noise through regular expressions in known cases. Provided the list [C,E,U,A,P,R,S,F] consisting of all control flags, regular expressions may identify all strings containing ‘A’ and ‘F’ to provide parity with the previous scenario. Tools such as Scapy maintain these flags within the object (Biondi et al., 2019). Within the context of data flow, data structures and their requirements may differ. Raw text contains no validation or structure for data formats like internet protocol addresses, while certain asset management and SIEM platforms may accommodate address validation or normalization. Analysts can leverage structural rules alongside confidence requirements to alter detection patterns accordingly.

Analysts may manage time-complexity tradeoffs through variations in confidence requirements such as those that may present through the two IPv4 detection patterns below. These non-exhaustive methods differentiate by accuracy in bounding numeral digits from 000-255 and 000-999, respectively:

```
^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$
^(?:[0-9]{1,3}\.){3}[0-9]{1,3}$
```

One last common form of noise is present in the form of true noise or variability. Unexpected variability is often derived from external corruption and improper error handling. True noise is mitigated through the same data handling and transmission channels. Analysts may encounter these forms through other mechanisms such as sampling. The analyst can determine confidence intervals through the algorithms and attributes of the sampling process defined within the source system. Cisco NetFlow Version 9 Exporter defined in RFC3954 is a good reference for this sampling. Specification outlines SAMPLING_INTERVAL and SAMPLING_ALGORITHM attributes, which each define their respective states (Cisco Systems Netflow, 2004). Here the sampling interval is established in the form of $\frac{1}{n}$ such that one out of every *n* packets is sampled from the population. Conventionally, the accuracy of the sample increases as the value of *n* approaches 1. This accuracy is explained through inversion (Haddadi et al., 2008). Additionally, the sampling algorithms, such as the deterministic and random sampling provided in Netflow Version 9, impact accuracy dependent upon factors such as the homogeneity of the population network. Analysts can mitigate sample variability through collection at alternative stages in the data flow, or via reconfiguration of the data collection process through the removal of sampling.

Lastly, there is perceived noise that is true signal. Unlike the original definition under signals processing, abstraction in data sciences provides for noise through variability within the signal itself of a data set. Extraction of internal noise, or signal variability, has semantic meaning to both the analyst interpreting the data as well as the computer processing the data. Within the information security space, the concept of unintentional interpretation arises in null byte injection (Auger, 2009).

This active exploitation bypass technique attempts to bypass filters through the addition of null-byte characters within user-supplied data. Exploitation is derived from C/C++ style languages where the null-byte represents string termination, leading the application to stop processing the string immediately, which can lead to unusual behavior. Sanitization methods attempt to prevent the ingestion of these characters, or at a minimum, halt the processing of a string termination literal (Wittenstein, 2015).

While encoding methods like those used in a Uniform Resource Locator (URL) render the NUL character in a human-readable manner, some data collectors may retain the original or interpreted meaning as a null-byte. Casting to human-readable characters or in other ways removing unexpected noise otherwise changes the semantic meaning of the message, thereby removing the true contextual security of the event.

It is vital to note that each stage in the data flow may interpret or process data in a different manner. Noise removal and data preprocessing as a whole must maintain interpretability for both the human and the machine to retain the semantic meaning of the event.

4.2. Text Standardization and Normalization

After noise removal, natural language processing and text mining stacks perform standardization and normalization on the ingested data. Normalization comes in many forms. On the preventative side of information security, processes such as parameterization, input sanitization, or validation are typically found in processing user-supplied input. Extraction and normalization in forms such as abstract syntax trees have been shown successful by Xie and Aiken (2006). These same processes occur within the natural language processing realm.

Extended from the extraction ontology, normalization borrows heavily from the original data flows of the source security toolsets. The analyst can normalize data not only through peer data sets but across logical layers and stages in the data flow process. Structured data, such as internet protocol addresses, often share common normalization processes, unlike most unstructured data sets. However, structured sanitization processes may not extend into sanitization techniques injected by the security tools within the data flow. Obfuscations such as the substitutions found in HyperText Transfer Protocol (HTTP) links in the form of HXXP and [.] dot-notations are not natively normalized.

All forms of normalization must preserve the semantic context; however, contextual requirements may differ between the preservation requirements for machines in the data flow and for the analyst. Sanitization of web traffic before and after the injection of a Web Application Firewall (WAF) changes the semantic meaning of the data for both the direct receptor as well as the indirect analyst (OWASP, 2008). Rendering commands as a string object may change the semantic meaning to the receptor, while semantics are upheld for the analyst. Data collectors generally gain precision at the cost of variability and normalization as the logical abstraction and distance to the source is reduced.

This concept extends for data formats such as time. Standards such as epoch time facilitate high adaptability in indirect normalization at the cost of direct human interpretation. Social and geographic considerations may impact normalizations, as time and date, time zone, and daylight savings are conceptually abstracted. In preserving the hypothesis, normalization techniques should facilitate the identification of trends like seasonality in time series found in statistical analysis (Davey & Flores, 1993).

Protocol definitions and limitations such as the use of American Standard Code for Information Interchange (ASCII) in the Domain Name System (DNS) protocol inherently limit the native representation of character sets outside of American English. An internationalized domain name (IDN) facilitates the encoding of language-specific scripts in a Unicode format, which is transcribed for DNS in punycode (Batayneh, 2011). The discrepancy in representation has led to vulnerabilities such as IDN homograph attacks in web navigation (Davis & Suignard, 2014).

Analysts must make further considerations for encoding techniques in the normalization process. Formats such as URL encoding and Base64 are prevalent in multiple protocol stacks. Data can be encoded in multiple formats given a single source. Representation may also include statically cast content such as (char)71. Substitution may also occur semantically within forms such as the PowerShell ".replace()" function. The semantic meaning may also be modified to either the data receptor or the analyst through the insertion of inert data or deletion through methods such as substrings (Hendler, Kels, & Rubin, 2018). If these operations appear familiar, it is because these actions are regular obfuscation techniques. Deobfuscation methods can be duplicated within the normalization process.

Analysts can classify these obfuscation and deobfuscation operations through their underlying actions. In natural language processing, these operations are expressed in the concept of edit distance. These methods have been successfully applied to anomaly detection in areas such as system calls (Quan, Jinlin, Wei, & Mingjun, 2012). Outside of language disparity, greedy regular expressions have shown moderate success in translating flattened structures, which may extend to forced normalization of formats such as Base64 (Frisch & Cardelli, 2004). While Base64 decoders are greedy on their own, the analyst still needs to validate that the decoded result is valuable to them. The limitations of regular grammars discussed for noise management apply to normalization as well. Analysts may use regular expressions iteratively for known subsets such as statically cast characters, however, nested substitutions like the .replace() function require flattening. Instead, interpretation can be used to retain meaning.

In-language attributes such as capitalization and punctuation may differ in semantic meaning even provided the same alphabet as well. Ordinal data relies on positioning. Normalizing duplicate characters may remove the contextual formatting in delimited files for spaces, tabs, commas, and other delimiters. Differences in case sensitivity between languages like PowerShell and Python impact normalization requirements such as those shown in Figure 4 below:

```
#!/usr/bin/python
var = 1
vAr = 0
print(var) # 1
print(vAr) # 0
```

```
#!/usr/bin/powershell
$var = 1
$vAr = 0
Write-Host $var # 0
Write-Host $vAr # 0
```

Figure 4
Discrepancy in Case Sensitivity Amongst Languages

One of the most prevalent forms of text normalization is casting. Constrained encoding enforces uniformity within a known subset of characters. While casting may occur in formats such as character case, encoding standards such as ASCII and Unicode are the most commonplace. Backwards compatibility is only guaranteed in a segment of encoding standards, such as ASCII character representations in Unicode. It is essential to note that Unicode is only an abstracted encoding standard, consisting of code points for discrete symbolic characters. UTF-8, UTF-16, and other encoding formats represent these Unicode characters, facilitating their translation, or coding, through a discrete mapping (Whistler, 2020).

However, due to encoding discrepancies, equality of these characters is not guaranteed because languages may not retain equality before normalization. Analysts can observe disparity in equality in the case of UTF-8 and UTF-16, both Unicode formats. Unicode may represent the character “ë” as the single code point U+00EB or as the combination of two code points U+0065 and U+308 for the e and the COMBINING DIAERESIS. These discrepancies are outlined in Figure 5 below:

```
#!/usr/bin/python
# Unicode as single code point
'\u00eb' # 'ë'
# Unicode as combination of code points
'\u0065\u0308' # 'ë'
# Numeric evaluation of codepoints
'\u00eb' == '\u0065\u0308'
# False
```

```
#!/usr/bin/python
# Decode hexadecimal UTF-8 encoding
b'\xc3\xab'.decode("utf-8") # 'ë'
# Decode hexadecimal UTF-16 encoding
b'\xff\xfe\xeb\x00'.decode("utf-16") # 'ë'
# Numeric evaluation of encodings
b'\xc3\xab' == b'\xff\xfe\xeb\x00' # False
# Symbolic evaluation of expression
b'\xc3\xab'.decode("utf-8") ==
b'\xff\xfe\xeb\x00'.decode("utf-16")
# True
```

Figure 5
Discrepancies in Evaluation for Unicode Formats

Universal normalizations exist to facilitate process repetition. There are four standard forms which normalize the strings into canonical forms. The most common is Normalization Form Canonical Composition (NFC), which first decomposes all characters, and then all sequences are recomposed in a standardized order (Whistler, 2020). Consistency in the normalization process is required to ensure the sample is treated consistently. Maintaining compliance with RFCs and standards in personally developed code may create additional challenges and considerations.

Security applies additional context to the normalization process. Deletion and Substitution operations commonly disrupt semantic meaning in normalization. Note that the composition plays a strict role in the combination of code points in deletion, as seen below in Figure 6:

```
#!/usr/bin/python
# DELETION
'ëalertë()'.encode("utf-8") # b'\xc3\xabalert\xc3\xab()'
b'\xc3\xabalert\xc3\xab()'.decode("ascii","ignore") # 'alert()'
# SUBSTITUTION
import unicodedata
text = "alërt()"
unicodedata.normalize('NFKD',text).encode('ascii','ignore') # b'alert()'
```

Figure 6
Security Considerations for Unicode Normalization

When viewing a tool like Mark Baggett's *freq*, it is necessary to retain context of the design (Baggett, 2019). It is a powerful tool, but it is framed around ASCII and casts in 'latin1' character set. Casting to 'latin1' completely disregards Asiatic character sets such as Hanzi and Hangul found within Unicode. However, as a purpose-built tool for Domain Generation Algorithms, *freq* builds upon the inherent limitations of DNS, and facilitates other languages through Punycode, through a subset of ASCII defined as the Letter-Digit-Hyphen (LDH) subset.

Lastly, when normalizing data, precision is also a valid criterion in the alternative direction. It is possible that precision may obfuscate the measured values pertaining to the hypothesis. In the information security field, this is often the result of flattened nested trees. Analysts may reduce Object Identifiers (OIDs), subdomains, and other formats to such a level that only the hypothesis remains.

4.3. Tokenization and Lexical Analysis

Lexical analysis, or tokenization, is the process of converting a sequence of characters into a sequence of tokens which are units of an ascribed meaning. It is at the lexical analysis stage that the analyst continues by explicitly reviewing the syntax of the input data. Tokenization is one of the core tenets in text mining and NLP.

For natural language, in something like the English language, tokenization presents as a sentence, comprised of multiple words, split by spaces, and contains symbols such as punctuation. Tokenization inherently relies on the success of the previous processes. In English, splitting on symbolic units derives characters from words, spaces split words from sentences, while punctuation often splits sentences into paragraphs. Stripping punctuation or otherwise removing spaces and other content impacts the tokenization process.

Tokenization is merely the act of segmenting data into its atomic form. Structured language can mirror natural language in many ways. In the same way a novel tells the story of who did what, and where, and how they did it at some time, a properly constructed syslog message can tell the same story. The same parts of speech and tagging the analyst derives in NLP apply here as well. Better yet, security classifiers are often explicitly annotated.

Analysts may engage in tokenization with varying levels of complexity. It is possible to natively split strings or process with regular expressions in many languages. Natural language processing toolsets may provide additional assistance. Please note that natural language elements such as contractions are handled natively in many NLP tools; however, these tools may be explicitly tuned for natural languages.

```
#!/usr/bin/python
domain_name = "mail.test.example.com"
english_phrase = "I can't do this."

#Native splitting techniques
[domain_name] # [' mail.test.example.com ']
domain_name.split(".") # ['mail', 'test', 'example', 'com']
list(domain_name) # ['m', 'a', 'i', 'l', '.', 't', 'e', 's', 't', '.', 'e',
'x', 'a', 'm', 'p', 'l', 'e', '.', 'c', 'o', 'm']

# Regular expressions
import re
tokens = re.findall("\w+",domain_name) # ['mail', 'test', 'example', 'com']
tokens = re.findall(".",domain_name) # ['m', 'a', 'i', 'l', '.', 't', 'e',
's', 't', '.', 'e', 'x', 'a', 'm', 'p', 'l', 'e', '.', 'c', 'o', 'm']

# Tokenizers
import nltk
nltk.word_tokenize(domain_name) # ['mail.test.example.com']
nltk.word_tokenize(english_phrase) # ['I', 'ca', "n't", 'do', 'this', '.']
```

Figure 7
Varied Tokenization Methods in Python

Tokenization is one of the most vital parts of analysis. The process must mirror the hypothesis. Given the fully qualified domain name (FQDN) “mail.test.example.com”, an analyst can derive atoms as the FQDN, the list of domain elements split on “.”, or the list of all characters split by symbol. This is displayed in the figure above. The FQDN may provide a unique count of visitors, but the FQDN does not natively count the intersection of Top Level Domains (TLDs) present. Tokenization at the subdomain presents the native segmentation of the structure. The context may now answer questions such as “How many requests are made to the test mail server compared to the production mail server?” or “Are there any calls to mail servers outside example.com?” Note that it is typically easier to deconstruct a larger object than to reconstruct a set of smaller ones.

In summation, tokenization facilitates the comparison between the tokens generated. Splitting on symbolic characters yields the ability to compare between the letters themselves. Semantic atomization also extends to security data structures such as TCP flags. Control flags may provide more value in context than in isolation, such as the combination of SYN and ACK flags.

Note that analysts may execute information security-styled tokenization alongside NLP tokenizations. If the need to privatize data occurs, the analyst may do so with features such as pseudonymization and data anonymization. The security tokenization process assists with data classification but may also assist in removing biases that the analyst may inject. Tokenization may assist in regulation such as the European Union General Data Privacy Regulation (GDPR) wherein ‘personal data’ is pseudonymized such that it can no longer be attributed to a single data subject without additional data (Biskup & Flegel, 2000).

4.4. Vocabulary Indexing

Once tokenized, a selection of atomic elements is formed. One of the most universal models in natural language processing is bag-of-words (Harris, 1954). The definitive bag of words model discards all context, containing only a listing of atomic elements. Analysts can vectorize the bagged content by collecting the count of the occurrences for the individual atoms. For interactive shells such as bash, vectorization can occur through establishing the count of each unique token contained in the data. Note that the vectorization process relies on particular data formats. The trim command accomplishes tokenization through substituting spaces with newline characters, which are then processed with the sort command.

```
#!/usr/bin/bash
echo "Mary had a little lamb. Little lamb, little lamb." > not_normalized
echo "mary had a little lamb little lamb little lamb" > normalized
```

```
tr ' ' '\n' < not_normalized | sort | uniq -c    tr ' ' '\n' < normalized | sort | uniq -c
      1 a                                              1 a
      1 had                                            1 had
      1 lamb,                                          3 lamb
      2 lamb.                                          3 little
      2 little                                         1 mary
      1 Little
      1 Mary
```

Figure 8
Tokenization as a Feature of Normalization

The vocabulary indexing completed in Figure 8 above with sort and uniq is text vectorization. In addition to shell commands, scripting languages like Python provide alternative methods. While the analyst may use native data structures, libraries may simplify logical requirements. An example is shown in Figure 9 below. Lastly, like the parsers discussed during noise removal, natural language processing libraries contain vocabulary indexing and vectorization methods to complete the required task.

```python
#!/usr/bin/python
text = "mary had a little lamb little lamb little lamb"
tokens = text.split(" ")
# with native dictionary
counter = dict(zip(tokens,[tokens.count(i) for i in tokens])) #
print(counter) # {'little': 3, 'mary': 1, 'had': 1, 'a': 1, 'lamb': 3}
# using collections library
from collections import Counter
Counter(tokens) # Counter({'little': 3, 'lamb': 3, 'mary': 1, 'had': 1, 'a': 1})
```

Figure 9
Varied Text Vectorization Methods in Python

One of the most meaningful attributes to security analysts is context. This tenet holds true for natural language processing as well. A foundational feature of NLP is term frequency. There are several methods for calculating a term frequency; the most elementary definition being the raw count of the term. Other methods, such as the number of times the word appears in the document over the total number of words appearing in a document, attempt to retain some context of the word to the sample document itself. Analysts may utilize term frequency alongside *log* to dampen the outliers and obtain a more human-readable output. This log function holds tremendous value to analysts because exponential becomes linear. Shown below is an example of traffic for a small network which roughly matches $f(x) = 2^x$. Long-tail analysis may assist in identifying anomalous traffic; however, features of this tail are muted due to the head. This same data mapped to a logarithmic scale on the y-axis may begin to show other patterns.

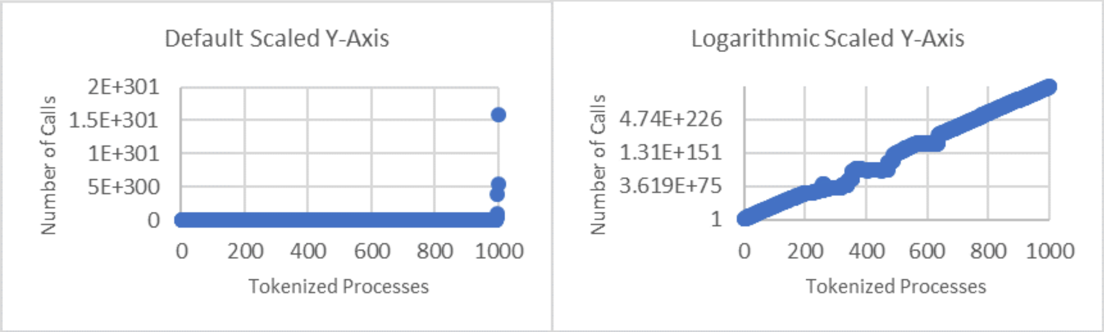


Figure 10
Visual Impact of Y-Axis Scaling

Analysts may obtain more value if the data is viewed as an overall pattern. Data may present common distributions at scale. One of the most common distributions in languages is Zipf's distribution, a power-law. Similar patterns in natural language may exist in other populations, such as URLs visited across a corporate domain (Adamic & Huberman, 2002). These patterns become significant in baselining behavior (Jensen, 2007). Note that there is a distinct difference in power-law distributions and exponential distributions. While it can be meaningful to know the math behind these distributions, it is possible to derive some context alone from which the data appears. This sentiment is extended if further data sets are accessible. Neighbor documents could be other syslog messages, peer group access reports, or similar data sets from another time period.

The latter context is between these peer groups. IDF is the inverse document frequency. The IDF is often presented as the log of the number of documents over the number of documents in which the atom appears (Sparck Jones, 1972). Presented together, the intersection of these techniques creates the TF-IDF: the product of these two values. The TF-IDF can be written as:

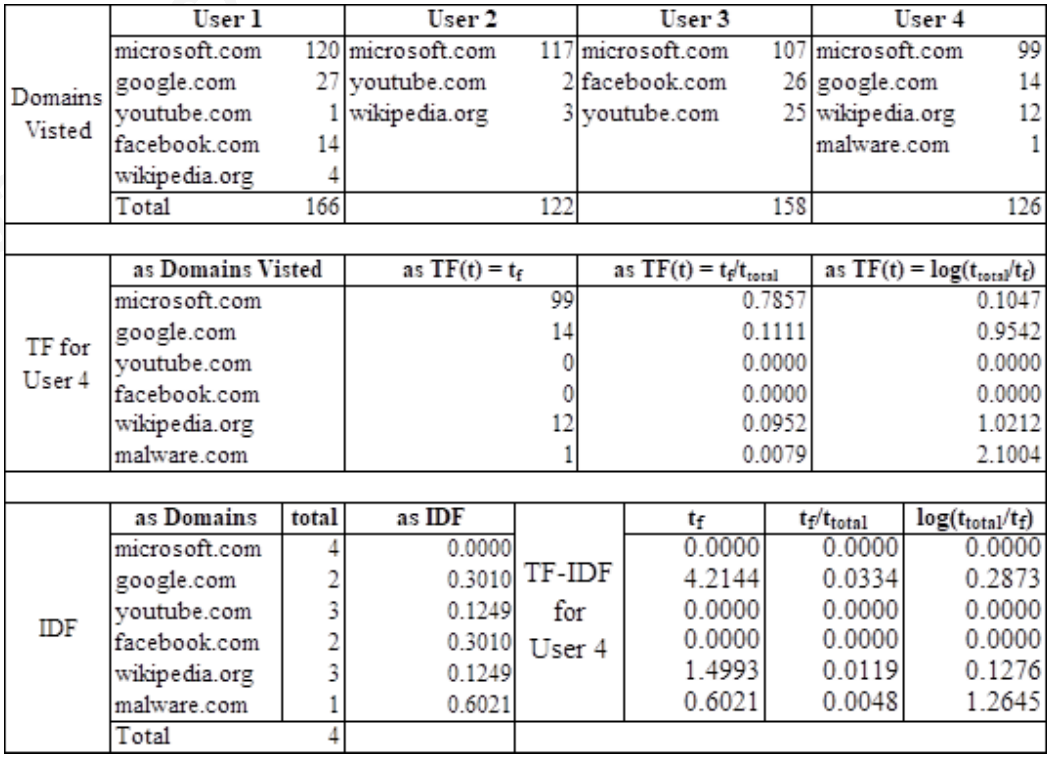
$$TF(t) = \frac{The\ number\ of\ times\ term\ t\ appears\ in\ a\ document}{total\ number\ of\ terms\ in\ a\ document}$$

$$IDF(t) = \log \frac{total\ number\ of\ documents}{number\ of\ documents\ containing\ term\ t}$$

$$TFIDF(t) = TF(t) \cdot IDF(t)$$

Given ordinary sentences, words such as articles disappear while potentially significant words such as proper names remain. Similar to techniques such as long-tail analysis, the unique content remains. The inversion of anomalous terms is in large part due to the log function within the IDF. As a term appears in more documents, the fraction approaches the value of one, where the log(1) = 0. Phrased otherwise, the more frequently the word occurs amongst all documents, the less impact that word provides (Robertson, 2004). Note that TF is unique per document, whereas IDF is shared for all documents given that term. The above TF equation also looks for terms used more frequently in that document. The provided definition in the equation above is counter-intuitive for low and slow attack methods.

These principles also apply to web traffic and other security data sets. Universal traffic such as web identity providers contributes little to anomaly detection, while unique values remain. By substituting terms for unique domains visited, and documents for unique users, TF-IDF can assist in extracting anomalous activity. An example of TF-IDF application is shown in Figure 11 below:



	User 1		User 2		User 3		User 4	
Domains Visted	microsoft.com	120	microsoft.com	117	microsoft.com	107	microsoft.com	99
	google.com	27	youtube.com	2	facebook.com	26	google.com	14
	youtube.com	1	wikipedia.org	3	youtube.com	25	wikipedia.org	12
	facebook.com	14					malware.com	1
	wikipedia.org	4						
	Total	166		122		158		126

	as Domains Visted	as TF(t) = t_f	as TF(t) = t_f/t_{total}	as TF(t) = $\log(t_{total}/t_f)$
TF for User 4	microsoft.com	99	0.7857	0.1047
	google.com	14	0.1111	0.9542
	youtube.com	0	0.0000	0.0000
	facebook.com	0	0.0000	0.0000
	wikipedia.org	12	0.0952	1.0212
	malware.com	1	0.0079	2.1004

	as Domains	total	as IDF		t_f	t_f/t_{total}	$\log(t_{total}/t_f)$
IDF	microsoft.com	4	0.0000	TF-IDF for User 4	0.0000	0.0000	0.0000
	google.com	2	0.3010		4.2144	0.0334	0.2873
	youtube.com	3	0.1249		0.0000	0.0000	0.0000
	facebook.com	2	0.3010		0.0000	0.0000	0.0000
	wikipedia.org	3	0.1249		1.4993	0.0119	0.1276
	malware.com	1	0.6021		0.6021	0.0048	1.2645
	Total	4					

Figure 11
TF-IDF Matrix for Sample Users

The document comparison practice extends to traffic for one user over the course of four days, and other formats as well. Active Directory (AD) Organizational Unit (OU), job role, and other classifiers inherently organize data subsets. Given File Access Monitoring (FAM) logs, each term *t* may present as a unique file ID accessed, while document *d* is the list of files accessed in a discrete period. When used within the context of file access amongst a peer group for IDF, security researchers may identify reconnaissance through improbable access.

There are weaknesses to the TF-IDF technique. As framed above, the IDF depends largely on the log function. The log function relates roughly to the distribution and the power-law discussed previously due to its inverse exponential nature. Distributions may impact the effectiveness of this approach. Nevertheless, there are methods to improve the relevance of the TF-IDF technique (Wu et al., 2008).

However, it is also vital to retain security through context. The example cited above is presented at the domain level. Security analysts would not identify excessive traffic within the payload, or traffic tunneled through an existing domain at the domain level. Alternative tokenization or numeric quantifiers such as bytes in/out ratios may be better suited for those abstractions.

Lastly, analysts must consider normalization and data flow. Within the above example, obfuscation and covert channel techniques such as domain fronting may present themselves at certain stages in data collection (Fifield et al., 2015).

At this point, the analyst should have a firm grasp on the preprocessing steps found within natural language processing. The combination of noise management and text standardization phases can be applied directly back into the SIEM, IDS, and other detection tools. The tokenization and indexing presented herein facilitate the quantitative comparison of feature sets within the qualitative data itself. TF-IDF and other indexing tools present a quantitative method to quickly determine both relationships and anomalous behaviors within a heterogeneous data set.

5. Data Analysis and Modeling

5.1. Probability and Behavior

The techniques described in the previous sections are empowered dramatically in the context of probability. Broached indirectly in the topic of TF-IDF, probability defines the likelihood of an event's occurrence given some context, such as a term appearing in all documents. For TF-IDF, frequent behaviors are excluded as the more shared a behavior is, the less information the behavior provides about a given context.

Many security tools, such as the SIEM, are designed to function within static thresholds. A familiar example would be fifty incorrect password attempts over thirty seconds. But a question arises as to how those numbers are derived. As more users populate the network, the number of failures may increase naturally. Static rules often scale growth through the introduction of additional information to limit the natural scale of growth itself. Methods may isolate to a context that does not grow (e.g., from a single source) or, define and whitelist elements that do not meet the desired criteria (e.g., limit to interactive logins to facilitate automation).

Probability merely quantifies these relationships. Absent knowledge of intent, ratios, such as the ones above, assist the analyst in categorizing the normalcy of behavior within a given context.

An extension of this philosophy lies in Bayesian statistics. In statistics and probability theory, Bayes' Theorem describes the probability, or likelihood, of an event occurring based on prior knowledge of possible conditions (Pearl, 1988). Bayes' theorem is comprised of a few elemental forms of probability:

Marginal Probability is the likelihood of an event irrespective of the outcomes for other random variables. This is represented simply as P(A)

Joint Probability is the likelihood of two or more simultaneous events. This is often written as the occurrence of two or more events, such as P(A and B) or P(A,B)

Conditional probability is the likelihood of one or more events given the occurrence of another event. This is often provided as two events A and B and presented as P(A given B) or P(A|B).

The **Product Rule** represents the relationship between the joint probability and conditional probability. The product rule states that the probability both A and B will occur is equal to the product of the two terms: the probability that event B occurs, and the probability that A occurs given that B has already occurred. The product rule is written as:

$$P(A \cap B) = P(A|B) \cdot P(B)$$

From the product rule, security analysts can infer two relationships that strongly impact event relatedness where:

$$A \subseteq B \textit{ implies } P(A|B) = \frac{P(A)}{P(B)}$$
$$B \subseteq A \textit{ implies } P(A|B) = 1$$

These two relationships describe environments where events A and B are subsets of each other, such as in contiguous events. In the case that all events B occur within the context of A, then the likelihood of its conditionality is 100%. Security analysts often work with sequences in the form of language, protocols, processes, and transactions. Conditional probability offers another frame to evaluate these events. Bayes' Theorem presents conditional probability as:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Within the security realm, analysts may use Bayes' Theorem to represent arbitrary sequences or collections of events. Given some regulatory database, the analyst is asked to determine the likelihood of an HR employee making a query that contains regulatory data. Logs show that 5% of all queries to this database request regulatory data. Meanwhile, analysis shows that 15% of total requests are made from users in the HR organizational unit. After counting, the analyst determines that 75% of all queries requesting regulatory data are tied to users in the HR organizational unit. Bayes' Theorem presents the following:

$$P(HR\ OU|make\ regQuery) = \frac{P(regQuery|from\ HR\ OU)\ \cdot P(regQuery)}{P(HR\ OU)}$$
$$= \frac{0.75\ \cdot 0.05}{0.15} = 0.25$$

Within the above scenario, there is a 25% chance that the database query from an HR employee would request some form of regulatory data. However, Bayes' Theorem isn't bound to a single directionality. The original hypothesis is focused upon P(A|B) for HR employees making regulatory queries. It is equally mathematically valid to identify the probability of a regulatory query coming from the HR team when both the marginal probabilities and P(B|A) is known. Bayes' Theorem can be used to indirectly measure the conditional probability of events that do not directly present within the data. The use of this technique should always reduce the effort of an analyst. The hypothesis, P(A|B), should be an unknown test, given known events for A and B.

Naïve Bayes spam filtering is a direct application of this technique. This spam filtering introduces two additional concepts to Bayes' Theorem. Firstly, the probability is represented in terms of the product rule depicted earlier in this section. Secondly, the denominator adds the concept of the complement, shown here as B^c.

$$P(B|A) = \frac{P(B \cap A)}{P(A)} = \frac{P(B \cap A)}{P(B \cap A) + P(B^c \cap A)}$$

$$P(Spam|Word) = \frac{P(Word|Spam) \cdot P(Spam)}{(P(Word|Spam) \cdot P(Spam)) + (P(Word|Ham) \cdot P(Ham))}$$

In written language, the equation presents a binary classification for the test and condition, which presents as the familiar 2x2 contingency table consisting of true and false positives and negatives used within the security space. Derived above, spam filters present probability in terms of binary classification as the likelihood of content being spam is directly related to the likelihood of true positives in relation to all positives.

Bayes' Theorem can be applied in both the simplified and extended forms across a number of security events. While the concept can be directly applied to endpoint detection and response process trees, such as in finding the likelihood of svchost spawning an arbitrary process, the concept can also be supplied across neighbor groups such as subnets and organizational units in the same manner as TF-IDF (Jensen, 2007).

5.2. Entropy

Probability can be further developed when conceptualizing the amount of information contained within a symbol or event. Discussed previously in the context of key-value pairs, information is naturally defined through differences. Where all keys are “username”, there is nothing besides the value of the username to differentiate between the pairs. It would be possible to remove “username=” from all Key/Value pairs without changing the semantic meaning of the values.

First introduced in the paper “A Mathematical Theory of Communication” by Claude Shannon (1948), information entropy is a fundamental quantity in information theory that attributes the average level of information, or uncertainty, the entire unit contains. Entropy is dependent upon the total amount of information that can be conveyed. The informational maximum is directly dependent upon the data preprocessing phase that occurred previously. Given a character set of the ASCII table, there is a decimal count of 128 units, from 0-127. However, the max character count is reduced with the removal of non-printable characters, such as NUL (#0). Alternatively, the removal of capitalization reduces the total number of printable alphabetical characters from fifty-two to twenty-six. A single character is possible of representing one of twenty-six possible states. This system is equivalent to a base-26 numeral system.

Reducing the system down to the elemental bit used in computing simplifies the concept of entropy by means of probability. The bit presents a base-2 numeral system where a bit can symbolize ON or OFF in a binary state. A combination of two bits allows for four total states, while eight bits, or one byte, can represent 256, or 2^8 different states. By extension, a four-letter word in the lower-case alphabet of 26 characters is capable of representing 456,976, or 26^4 possible states. These permutations are the foundation of password security when discussing the likelihood of a guessing attack (Wheeler, 2016). In reality, only a subset of these permutations is used for sociological or linguistic reasons. This factor facilitates wordlists, such as those used by password cracking tools.

Abstracted further, variation in entropy applies directly to the data collection flow. The same unit of information may be presented in a binary or hexadecimal format or abstracted into a string format through further encodings such as UTF-8 or ASCII. The same symbolic unit may have differing entropy dependent upon the size of the character set. Hexadecimal utilizes the base-16 alphabet of 0-F while ASCII utilizes 7 bits or an effective byte. Given the hexadecimal of '5A' equivalent to 'Z', it is still possible to get an accurate count, so long as tokenization and normalization have ensured semantic meaning of the byte is intact. Covert channels utilizing concepts such as the least significant bit can exist at all levels (Gupta, Goyal, & Bhushan, 2012). The closer the data collection is to the wire, the more available this entropy. While done at the cost of abstraction, analysts may find that requisite data is not available at an abstracted level.

Entropy gets even more profitable when viewed as a relation to itself. Within information theory, conditional entropy quantifies the amount of information needed to describe the result of some random variable *Y*, given that the value of another random variable *X* has occurred. The stated definition hearkens back to conditional probability. In information theory, this is described as "The entropy of Y conditioned on X" and is written as H(Y|X).

This probability can be related back to entropy itself. The likelihood of pulling the letter 'a' out of the alphabet is 1/26. By extension, the likelihood of the letter 'a' following 'a' can be described as two independent probabilities such that: $\left(\frac{1}{26}\right) \cdot \left(\frac{1}{26}\right) = \left(\frac{1}{676}\right) \sim 0.0014$ The amount of entropy, or possible information, contained in this character sequence can be ascribed alongside its probability.

$$Shannon\ Entropy\ of\ an\ alphabet = -\sum_{i=1}^{N} p_i \log_2 p_i$$

Context deepens further when provided known probabilities from a given data set, such as letter frequencies in the English language. The entropy of an individual character symbol doesn't represent information density as a whole. Within language, symbols are not truly independent. By this factor, the information contained within these sequences may differ from the product of its alphabet. Mark Baggett utilizes these contextual frequencies within the *freq* script (2019). While natural entropy provides an objective view of a string given the entire alphabet available, *freq* applies apriori knowledge of how frequently characters appear together. This applies to the concept of conditional entropy.

Another extension of this concept falls under the chain rule: an extension of the product rule. The joint probability can be expressed in terms of the conditional probability such that:

$$\begin{aligned} \mathrm{P(X,Y,Z)} &= \mathrm{P(X|Y,Z)P(Y,Z)} \\ &= P(X|Y,Z)P(Y|Z)P(Z) \end{aligned}$$

$$P(X_1, X_2, X_3, \dots X_n) = \cdots$$

The chain rule concept can be applied across an entire word, or even a sentence, to calculate the likelihood of that occurrence. This concept is tangentially applied section 5.4 N-grams for contiguous sequences of characters in a word.

5.3. Finite State Machines

Another method of observing an arbitrary behavior is through a finite state machine. This machine, or a system, can be in exactly one of a finite number of states at any given time. The state machine can change between given states in response to some arbitrary input. Processes such as user and machine behavior can be framed in terms of these state machines. The most basic models consist of deterministic automatons; each state has exactly one transition for each possible input to the system. However, many of the human-centric systems security analysts may experience come in the form of non-deterministic automatons. These automatons may lead to one, many, or even no state transition, given some input.

The concept of transitioning states is already familiar to security practitioners. Finite state machines are depicted in places such as the TCP state diagram. Given a SYN packet, a proper connection transitions alongside a SYN, ACK, followed by an ACK. However, TCP has alternative state transitions. Defined in RFC 7413, the TCP Fast open facilitates connection alongside the initial SYN with the presence of a cryptographic cookie (Cheng, 2014). Both of these lead to an ESTABLISHED TCP state. What if this scenario is viewed from the perspective of an attempted TCP SYN scan? In this process, the analyst would expect to see the SYN_SENT state transition to TIME_WAIT on timeout.

Analysts can extend this state observation. Given a sample of traffic on a network, the analyst has visibility into a series of independent TCP state machines. What percentage of these sessions lead to a TIME_WAIT sequence after the initial SYN? Markov models can ultimately represent stochastic or probabilistic models. Given an example population, an analyst can determine the likelihood of some new connection going into a TIME_WAIT state.

But these state machines are based on transitions from some arbitrary input. What if these classifiers, such as source IP address, are not only correlated but causally related to the transition? Advanced modeling, such as what is done in machine learning, is precisely where these questions can lead (Li, Xiong, Chin, & Hu, 2019).

5.4. N-grams

Viewed holistically, the data preprocessed in the previous section can be represented statistically within the context of its entropy. Given that the analyst can now evaluate the likelihood of an event's occurrence, and the amount of semantic information it may carry, it is possible to start using those conditional probabilities to evaluate these events.

In computational linguistics and probability, n-grams are contiguous sequences of n items from a given sample of data. For language, these are signifiers: linguistic units or patterns that convey meaning. These are often symbols such as letters, syllables, or words. Latin numeral prefixes often ascribe n-grams: an n-gram of size one is a unigram, two, a bigram, et cetera.

N-grams function symbiotically with the tokenization performed during data collection and preprocessing. Given sequences of network traffic, child processes, and other behavioral relationships, an analyst can establish the likelihood of this n-gram occurring within a given population, such as a corporate network. Analysts may extract these n-grams in methods similar to the ones displayed below in Figure 12:

```
#!/usr/bin/python
# Generalized lambda notation
ngrams = lambda s, n: zip(*[s[i:] for i in range(n)])
# string s as input, int n as n-gram size
list(ngrams("alphabet",2))
#[('a','l'),('l','p'),('p','h'),('h','a'),('a', 'b'),('b','e'),('e','t')]
# NLTK method
from nltk import ngrams
list(ngrams("alphabet",2))
#[('a','l'),('l','p'),('p','h'),('h','a'),('a','b'),('b','e'),('e','t')]
```

Figure 12
N-gram Extraction Methods in Python

Domain Generation Algorithm analysis tools display a classic use of n-grams. An example can be found in Mark Baggett's *freq* (2019). In this tool, strings such as URLs are atomically split into characters. The *freq* algorithm sequences through bigrams. Borrowing from conditional entropy, the tool gathers a frequency table of these bigrams by counting the number of times B follows A. Having ingested a corpus on the English language, the resulting frequency state tables should offer an approximation on the likelihood for character sequences, compared to the abstracted total entropy (Baggett, 2019). The generation of n-gram likelihood can be expanded into the below sequence:

```
#!/usr/bin/python
word = "repetition"
ngrams = lambda s, n: zip(*[s[i:] for i in range(n)])
# iterate through all permutations
tokens = ngrams(word,2)
# get bigrams for word
counter = dict(zip(tokens,[tokens.count(i) for i in tokens]))
# get counts for each bigram

#  [(('i', 'o'), 1), (('i', 't'), 1), (('e', 'p'), 1), (('r', 'e'), 1),
(('o', 'n'), 1), (('t', 'i'), 2), (('e', 't'), 1), (('p', 'e'), 1)]

# get all bigrams starting with character - only bigrams
list_bigrams_starting_i = [value for key,value in counter.items() if 'i' in
key[0]]
sum_bigrams_starting_i = sum(list_bigrams_starting_i)
```

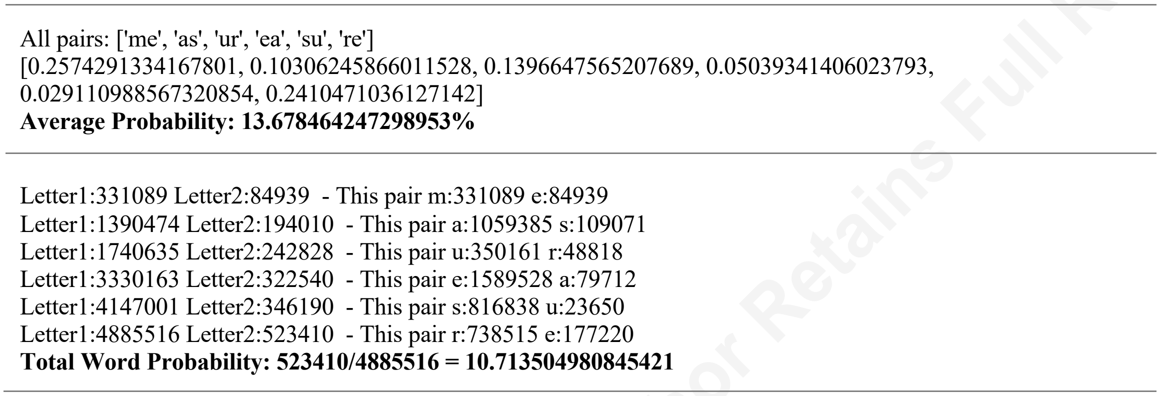
Figure 13
N-gram Vectorization in Python

From these numbers, the likelihood can be established. While the letter "i" appears twice in the word repetition, there is only a 50% chance it will be followed by the letter "t". However, the letter "t" has a 100% chance of being followed by the letter "i". Generation of likelihood is replicated across the entire string within the probability function of *freq*. Another example has been provided within the companion piece presented in the Appendix.

However, n-gram sequences can be used with alternative tokens. Provided the earlier domain name example, bigrams may be created through dot segmentation or split on the character sequences themselves. Once frequencies are established, the analyst can actively test the unknown domains against this set to calculate its likelihood given the sample. Alternatively, the analyst may use tools such as TF-IDF, presented in section 4.4 Vocabulary Indexing, to extract context from the given sample itself.

Analysts may vary the length of the n-grams just as the encompassed tokens are varied. Monograms effectively present a frequency table of the alphabet, as no relationships are established. However, this proximity presents the greatest weakness of n-grams as well. The series of contiguous sequences that build an n-gram present no long-range dependency. Simply stated, the further away from the n-gram, the less the association within the model. As analysts, the lack of ranged dependency must be retained, as behavioral sequences have no long durational context. Impact may be reduced with an increased sequence length at the cost of its own drawbacks (Naptali, 2011). Note that the preprocessing phase can be executed against both the source corpus and the target value. Standardization methods such as casing may simplify models at the cost of run-time complexity. For example, the n-gram pair "ar" may be represented in 2^2 case states. This requirement scales accordingly with the length of the n-gram.

The n-gram theory in this section directly applies to domain generation algorithms culminating in multiple detection methods. These methods rely on some prior knowledge built upon a selected corpus. The counters outlined in this section establish a baseline frequency upon some data set such as text from the target language, or a subset of known legitimate domain names like those provided in the top visited domain listings. The established model is then compared against the n-grams collected for the target domain, building the likelihood of characters in the target domain name appearing adjacent. The likelihood of each n-gram can then be combined through methods such as the mean to establish an overall likelihood of that domain name appearing. This naturally differentiates infrequent or truly randomized pairs like "qt" from more common pairs such as "re". Figure 14 below is an example for the word ‘measure’ utilizing the default frequency table within the *freq* domain analysis tool (Baggett, 2019).



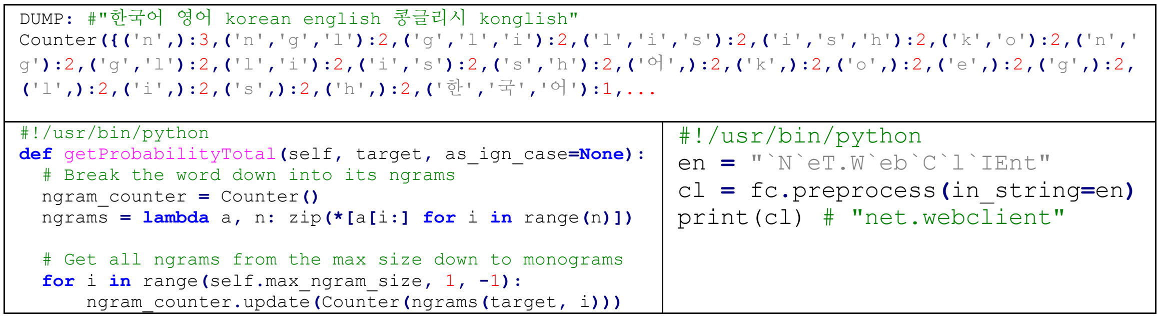
```
All pairs: ['me', 'as', 'ur', 'ea', 'su', 're']
[0.2574291334167801, 0.10306245866011528, 0.1396647565207689, 0.05039341406023793,
0.029110988567320854, 0.2410471036127142]
Average Probability: 13.678464247298953%
```

```
Letter1:331089 Letter2:84939  - This pair m:331089 e:84939
Letter1:1390474 Letter2:194010  - This pair a:1059385 s:109071
Letter1:1740635 Letter2:242828  - This pair u:350161 r:48818
Letter1:3330163 Letter2:322540  - This pair e:1589528 a:79712
Letter1:4147001 Letter2:346190  - This pair s:816838 u:23650
Letter1:4885516 Letter2:523410  - This pair r:738515 e:177220
Total Word Probability: 523410/4885516 = 10.713504980845421
```

Figure 14
Aggregation of Probability for N-gram Models

These n-gram models extend to sets of contiguous words within a domain as well. The randomized selection of words used within URL shorteners and hosting providers may be compared against English texts to identify the potential likelihood of their occurrence. The goal is to identify both randomness and determinism where they should not exist. Elementary spam mail borrows from this determinism as well. Given an established corpus of sentences, spam can utilize these stochastic Markov models to build semi-reasonable target sentences of a given meaning from these contiguous sequences.

Understanding of this theory provides the flexibility to apply these quantitative techniques to any problem. This value can be presented when an analyst baselines an already useful tool like Mark Baggett's *freq* (2019). With the combination of these concepts it is possible to repurpose techniques to work with multiple concurrent languages, utilize multiple n-gram lengths for dependency, preprocess text on ingestion, and weight additional concepts like edit distance. None of these features are presented in depth for a purpose-driven DGA tool. These additional features are sampled in Figure 15 below with additional code provided through the link in the Appendix.



```
DUMP: #"한국어 영어 korean english 콩글리시 konglish"
Counter({('n',):3,('n','g','l'):2,('g','l','i'):2,('l','i','s'):2,('i','s','h'):2,('k','o'):2,('n','
g'):2,('g','l'):2,('l','i'):2,('i','s'):2,('s','h'):2,('어',):2,('k',):2,('o',):2,('e',):2,('g',):2,
('l',):2,('i',):2,('s',):2,('h',):2,('한','국','어'):1,...
```

```
#!/usr/bin/python
def getProbabilityTotal(self, target, as_ign_case=None):
  # Break the word down into its ngrams
  ngram_counter = Counter()
  ngrams = lambda a, n: zip(*[a[i:] for i in range(n)])

  # Get all ngrams from the max size down to monograms
  for i in range(self.max_ngram_size, 1, -1):
      ngram_counter.update(Counter(ngrams(target, i)))
```

```
#!/usr/bin/python
en = "`N`eT.W`eb`C`l`IEnt"
cl = fc.preprocess(in_string=en)
print(cl)  # "net.webclient"
```

Figure 15
Feature Samples for Theory

Note: Hangul Unicode characters still tokenized upon syllabic block
Differences in language structure will impact models accordingly

d.S3VERANCE@gmail.com

5.5. Comparison Metrics

Now with data, behavioral sequences, and a few ways to manipulate them, analysts can start looking at comparison metrics to relate this data at scale. String metrics contribute to the first category of comparison metrics presented herein. These string metrics provide methods for numerically identifying the difference between two strings or series of symbols. Edit distance, described previously in section 4.2 Text Standardization, falls within this category. Originally intended to represent the minimum number of character edits required to change one string into another, methods such as the Levenshtein Distance can also utilize the latter interpretation of strings as a sequence of symbolic units.

Commonly used within phishing implementation and detection, Levenshtein Distance provides methods to identify lookalike domains (Yadav, Reddy, Reddy, & Ranjan, 2010). Common implementations of this algorithm are found in most popular programming languages (Algorithm Implementation, 2020). Levenshtein Distance is logically presented as:

$$lev_{a,b}(i,j) = \begin{cases} \max(i,j) & if\ \min(i,j) = 0 \\ \min\begin{cases} lev_{a,b}(i-1,j)+1 \\ lev_{a,b}(i,j-1)+1 \\ lev_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} & otherwise. \end{cases}$$

In a written format, the Levenshtein Distance accounts for the insertion, deletion, and substitution of characters as one string is compared against another. When compared as a sequence of symbolic units, Levenshtein may also be used in the comparison of two lists of neighbor nodes, and other data sets. Much like the application of a base-26 numeral system ascribed within section 5.2 Entropy, alternative foundations may be used. IP addresses are merely 2^{32} or 256^4 or $4{,}294{,}967{,}296^1$ character sets.

This concept can be extended to the Kendell Tau Distance, or the bubble-sort distance, which counts the number of pairwise disagreements between two ranked lists. Much like Levenshtein distance, the Kendell Tau Distance can be appropriated to represent concepts in security space, such as the comparison of most-visited sites between peer groups or any two days d_m and d_n. This method has been used successfully amongst other similarity measures in host-based intrusion detection systems (Rawat, Gulati, & Pujari, 2004).

Solutions such as Hunt–Szymanski algorithm, or Hunt–McIlroy algorithm identify the longest common subsequence in a series. Commonly used in version control systems, the Hunt–Szymanski algorithm is already largely tangential to the security space. Longest common subsequences (LCS) can be applied to any discrete contiguous series such as process trees.

LCS techniques inherently identify automated and procedural tasks due to the consistency in repetition compared to human behavior. Within this manner, LCS techniques can be weaponized directly against session activity and execution logs to establish a background.

6. Future Research

Future research opportunities are abundantly available in modern natural language processing techniques. This paper only highlights a few of the foundational topics relevant to the security domain. In closer proximity to natural language, analysts can utilize further techniques in processing natural language concepts. Syntactical tasks such as parts-of-speech tagging can be applied to assist in determining the role of values in the absence of native normalization. Originally intended to separate words into morphemes, morphological segmentation can facilitate the grouping of verbs for access control and other classifications (e.g., add, remove, and change access). Other foundational techniques in standardization, such as stemming and lemmatization, focus on the reduction of inflections such as verb tense, which can be applied to other groupings such as domains, subnets, or zones.

Natural language processing also offers additional opportunity in semantic analysis in the form of relationship extraction and named entity recognition (NER). For example, these techniques attempt to identify sequences that are frequently found in proximity, as well as extract formal nouns or concepts in a text. Both techniques are found in text classification methods (Gomez-Hidalgo et al, 2010).

7. Conclusion

Natural language has had a probabilistic reawakening over the last thirty years. While extensive educational backgrounds may be required for some analysis and modeling, the core principles remain largely accessible to security analysts. The information security space shares many of the same underlying tasks under a different name. The same probabilistic models in the natural language processing space can be applied to the security realm with minimal retooling.

A basic understanding of these principles facilitates more investigation avenues for both qualitative and quantitative research of text data for security analysts. These methods provide for the ability to identify randomness and the lack thereof within discrete data sets. The identification of randomness directly applies to detection techniques for covert communications, automated activity, and malicious or anomalous behaviors.

The information security realm often does not have a data problem, but an information extraction problem inherent to the DIKW transformation hierarchy. The techniques and procedures derived from neighboring educational fields provide alternative thought processes that may assist in providing more transformation pathways to the security analysis pipeline.

Scaling security analysis isn’t a solved problem; however, methods exist to analyze data within scalable capacities. Analysis processes can be augmented beyond current bottlenecks in the same way that automated data extraction facilitates incident response. With companies proselytizing data-driven security methods, the industry is starting to do more good with data. Borrowing from fields such as natural language processing, security analysts can too.

References

Adamic, L. A., Huberman, B. A. (2002). Zipf's Law and the Internet. Glottometrics, 3:143-150

Algorithm Implementation/Strings/Levenshtein distance. (2020, February 18). Retrieved from https://en.wikibooks.org/wiki/Algorithm_Implementation/Strings/Levenshtein_distance

Auger, R. (2009, December 30). Null Byte Injection. Retrieved from http://projects.webappsec.org/w/page/13246949/Null Byte Injection

Baggett, M. (2019, July 20). freq. Retrieved from https://github.com/MarkBaggett/freq

Batayneh, F. (2011). "International domain names from a multilingualism and security perspective", Internet Governance Capacity Development Programme. Vol. 2010-2011.

Bianco, D. J. (2013, March). The Pyramid of Pain. Retrieved April 1, 2020, from https://detect-respond.blogspot.com/2013/03/the-pyramid-of-pain.html

Biondi, P. (2019, August 6). BeautifulSoup. 4.8.2. Retrieved from https://scapy.net/

Bird, S., Loper, E., Klein, E. (2019, August 20). Natural Language Toolkit. 3.4.5. Retrieved from https://www.nltk.org/

Biskup, J., & Flegel, U. (2000). Transaction-Based Pseudonyms in Audit Data for Privacy Respecting Intrusion Detection. *Lecture Notes in Computer Science Recent Advances in Intrusion Detection*, 28–48. doi: 10.1007/3-540-39945-3_3

Broemeling, L. D. (2011). An Account of Early Statistical Inference in Arab Cryptology. *The American Statistician*, *65*(4), 255–257. doi: 10.1198/tas.2011.10191

Brown, J. D., & Charlebois, D. (2010). Security classification using automated learning (scale): optimizing statistical natural language processing techniques to assign security labels to unstructured text (No. DRDC-OTTAWA-TM-2010-215). DEFENCE RESEARCH AND DEVELOPMENT CANADA OTTAWA (ONTARIO).

Cheng, Y., Chu, J., Radhakrishnan, S., & Jain, A. (2014). TCP Fast Open. doi: 10.17487/rfc7413

Chifflier, P., & Couprie, G. (2017). Writing Parsers Like it is 2017. 2017 IEEE Security and Privacy Workshops (SPW). doi: 10.1109/spw.2017.39

Chollet, F. (2019, October 7). Keras. 2.3.1. Retrieved from https://keras.io/

Chomsky, N. (1956). Three models for the description of language. *IEEE Transactions on Information Theory*, *2*(3), 113–124. doi: 10.1109/tit.1956.1056813

Cisco Systems NetFlow Services Export Version 9. (2004). doi: 10.17487/rfc3954

Davey, A. M., & Flores, B. E. (1993). Identification of seasonality in time series: A note. Mathematical and computer modelling, 18(6), 73-81.

Davis, M., Suignard, M. (2014, September 19). Unicode Technical Report #36 UNICODE SECURITY CONSIDERATIONS. Retrieved from http://www.unicode.org/reports/tr36/

Fifield, D., Lan, C., Hynes, R., Wegmann, P., & Paxson, V. (2015). Blocking-resistant communication through domain fronting. Proceedings on Privacy Enhancing Technologies, 2015(2), 46–64. doi: 10.1515/popets-2015-0009

Frisch, A., & Cardelli, L. (2004). Greedy Regular Expression Matching. *Automata, Languages and Programming Lecture Notes in Computer Science*, 618–629. doi: 10.1007/978-3-540-27836-8_53

Gomez-Hidalgo, J. M., Martin-Abreu, J. M., Nieves, J., Santos, I., Brezo, F., & Bringas, P. G. (2010, August). Data leak prevention through named entity recognition. In *2010 IEEE Second International Conference on Social Computing* (pp. 1129-1134). IEEE.

Gupta, S., Goyal, A., & Bhushan, B. (2012). Information Hiding Using Least Significant Bit Steganography and Cryptography. International Journal of Modern Education and Computer Science, 4(6), 27–34. doi: 10.5815/ijmecs.2012.06.04

Haddadi, H., Landa, R., Moore, A. W., Bhatti, S., Rio, M., & Che, X. (2008). Revisiting the issues on netflow sample and export performance. *2008 Third International Conference on Communications and Networking in China*. doi: 10.1109/chinacom.2008.4685060

Harris, Z. S. (1954). Distributional Structure. *WORD*, *10*(2-3), 146–162. doi: 10.1080/00437956.1954.11659520

Hendler, D., Kels, S., & Rubin, A. (2018). Detecting Malicious PowerShell Commands using Deep Neural Networks. *Proceedings of the 2018 on Asia Conference on Computer and Communications Security - ASIACCS 18*. doi: 10.1145/3196494.3196511

Honnibal, M. (2020, March 12). spaCy. 2.2.4. Retrieved from https://spacy.io/

Jensen, F. V., Nielsen, T. D. (2007). Bayesian Networks and Decision Graphs. *Information Science and Statistics*. doi: 10.1007/978-0-387-68282-2

Johnson, C. S., Badger, M. L., Waltermire, D. A., Snyder, J., & Skorupka, C. (2016). Guide to Cyber Threat Information Sharing. https://doi.org/10.6028/nist.sp.800-150

Li, Y., Xiong, K., Chin, T., & Hu, C. (2019). A Machine Learning Framework for Domain Generation Algorithm-Based Malware Detection. IEEE Access, 7, 32765–32782. doi: 10.1109/access.2019.2891588

McCanne, S., Jacobson, V. (1993). The BSD packet filter: a new architecture for user-level packet capture. Proceedings of the USENIX Winter 1993 Conference Proceedings on USENIX Winter 1993 Conference Proceedings (USENIX'93).

Naptali, W. (2011). Study on n-gram language models for topic and out-of-vocabulary words. (dissertation).

OWASP. (2008). Best Practices: Use of Web Application Firewalls. Retrieved From https://www.owasp.org/images/b/b0/Best_Practices_WAF_v105.en.pdf

Paszke, A., Gross, S., Chintala, S., Chanan, G. (2020, January 15). PyTorch. 1.4.0. Retrieved from https://pytorch.org/

Pearl, J. (1988). Probabilistic reasoning in intelligent systems: networks of plausible inference. San Francisco: Morgan Kaufmann.

Postel, J. (1981). Transmission Control Protocol. doi: 10.17487/rfc0793

Ptacek, T. H., & Newsham, T. N. (1998). Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection. Secure Networks Inc Calgary Alberta.

Quan, Q., Jinlin, W., Wei, Z., & Mingjun, X. (2012, October). Improved edit distance method for system call anomaly detection. In 2012 IEEE 12th International Conference on Computer and Information Technology (pp. 1097-1102). IEEE.

Ramakrishnan, K., Floyd, S., & Black, D. (2001). The Addition of Explicit Congestion Notification (ECN) to IP. doi: 10.17487/rfc3168

Rawat, S., Gulati, V. P., & Pujari, A. K. (2004). Frequency-and ordering-based similarity measure for host-based intrusion detection. Information management & computer security.

Richardson, L. (2019, December 24). Scapy. 2.4.3. Retrieved from https://www.crummy.com/software/BeautifulSoup/

Robertson, S. (2004). "Understanding inverse document frequency: on theoretical arguments for IDF", Journal of Documentation, Vol. 60 No. 5, pp. 503-520. https://doi.org/10.1108/00220410410560582

Rodriguez, R. (2019, February 24) Hunting ELK. v0.1.7-alpha02242019. Retrieved from https://github.com/Cyb3rWard0g/HELK

Rowley, J. (2007). The wisdom hierarchy: representations of the DIKW hierarchy. Journal of Information Science, 33(2), 163–180. doi: 10.1177/0165551506070706

Scientific computing tools for Python. (2020). SciPy. Retrieved from https://www.scipy.org/about.html

Seymour, J., & Tully, P. (2016). Weaponizing data science for social engineering: Automated E2E spear phishing on Twitter. *Black Hat USA*, *37*, 1-39.

Shannon, C. E. (1948). A Mathematical Theory of Communication. *Bell System Technical Journal*, *27*(3), 379–423. doi: 10.1002/j.1538-7305.1948.tb01338.x

Sparck Jones, K. (1972), "A STATISTICAL INTERPRETATION OF TERM SPECIFICITY AND ITS APPLICATION IN RETRIEVAL", Journal of Documentation, Vol. 28 No. 1, pp. 11-21. https://doi.org/10.1108/eb026526

Tensorflow. (2020, January 8). Tensorflow. 2.1.0. Retrieved from https://www.tensorflow.org/

Yadav, S., Reddy, A. K. K., Reddy, A. N., & Ranjan, S. (2010). Detecting algorithmically generated malicious domain names. *Proceedings of the 10th Annual Conference on Internet Measurement - IMC 10*. doi: 10.1145/1879141.1879148

Wheeler, D. L. (2016). zxcvbn: low-budget password strength estimation. Proceedings of the 25th USENIX Conference on Security Symposium (SEC'16).

Whistler, K. (2020, February 24). Unicode Standard Annex #15 Unicode Normalization Forms. Retrieved from http://www.unicode.org/reports/tr15/

Wittenstein, A. (2015, September 30). US10044752B1. United States.

Wu, H. C., Luk, R. W. P., Wong, K. F., & Kwok, K. L. (2008). Interpreting TF-IDF term weights as making relevance decisions. *ACM Transactions on Information Systems*, *26*(3), 1–37. doi: 10.1145/1361684.1361686

Xie, Y., & Aiken, A. (2006, July). Static Detection of Security Vulnerabilities in Scripting Languages. In USENIX Security Symposium (Vol. 15, pp. 179-192).

Appendix

A companion script implementing a few of these concepts has been made available at https://github.com/S3V3R4NCE/freq

Last Updated: September 20th, 2020

Upcoming SANS Training

Click here to view a list of all SANS Courses

SANS Australia Spring 2020	**, AU**	**Sep 21, 2020 - Oct 03, 2020**	**Live Event**
SANS October Singapore 2020	**Singapore, SG**	**Oct 12, 2020 - Oct 24, 2020**	**Live Event**
SANS SEC504 Rennes 2020 (In French)	**Rennes, FR**	**Oct 19, 2020 - Oct 24, 2020**	**Live Event**
SANS SEC560 Lille 2020 (In French)	**Lille, FR**	**Oct 26, 2020 - Oct 31, 2020**	**Live Event**
SANS Tel Aviv November 2020	**Tel Aviv, IL**	**Nov 01, 2020 - Nov 06, 2020**	**Live Event**
SANS Sydney 2020	**Sydney, AU**	**Nov 02, 2020 - Nov 14, 2020**	**Live Event**
SANS Secure Thailand	**Bangkok, TH**	**Nov 09, 2020 - Nov 14, 2020**	**Live Event**
APAC ICS Summit & Training 2020	**Singapore, SG**	**Nov 13, 2020 - Nov 21, 2020**	**Live Event**
SANS FOR508 Rome 2020 (in Italian)	**Rome, IT**	**Nov 16, 2020 - Nov 21, 2020**	**Live Event**
SANS Oslo Local November 2020	**Oslo, NO**	**Nov 23, 2020 - Nov 28, 2020**	**Live Event**
SANS Wellington 2020	**Wellington, NZ**	**Nov 30, 2020 - Dec 12, 2020**	**Live Event**
SANS OnDemand	**OnlineUS**	**Anytime**	**Self Paced**
SANS SelfStudy	**Books & MP3s OnlyUS**	**Anytime**	**Self Paced**