Network Anomaly Detection using Exponential Random Graph Models and Autoregressive Moving Average

Michail Tsikerdekis, Scott Waldron, Alex Emanuelson

Computer Science Dept., Western Washington University

Abstract

Network anomaly detection solutions have been used as a defense against several attacks especially related to data exfiltration. Approaches that observe a network's data volume and attempt to determine if an event is an anomaly are susceptible to adversarial attacks. For example, machine learning models are susceptible to machine learning attacks such as the boiling frog poisoning attack in which increased traffic slowly raises the baseline of a defense model. We propose a new method that builds on top of existing network security monitoring practices that uses exponential random graph modeling in order to integrate network topology structure statistics in anomaly detection. We demonstrate the effectiveness of our method using real-world as a baseline for experiments on Domain Name System (DNS) data exfiltration scenarios. We highlight how our method provides a better description of what is happening in the network structure and how this can assist cybersecurity analysts in making better decisions in conjunction with existing intrusion detection systems. Finally, we describe some of the strengths of our method as well as its computational requirements.

Keywords: graph, arma, detection, anomaly.

[*]Corresponding author

Email address: Michael.Tsikerdekis@wwu.edu (Michail Tsikerdekis)

1. Introduction

Data exfiltration is a common type of cyberattack that an attacker uses in order to extract data from a network once unauthorized access to private and possibly sensitive data is gained. The exfiltrated data can include personally identifiable information (PII), private financial information, usernames with associated passwords and information involving strategic decisions. There exist many communication mediums in which to data can be exfiltrated from. Techniques include, but are not limited to: HTTP and HTTPS data transfer, email, peer-to-peer file-sharing, SSH, stenography, and protocol tunneling. Every system's communication channel is potentially vulnerable and can data may be extracted through it without authorization. Although the techniques and methodology for data exfiltration are known to cybersecurity professionals, high profile incidents of data exfiltrations (e.g., DNS exfiltration) can still be observed.

The cost of a data breach has consistently risen in recent years indicating a need for additional cybersecurity measures. According to the 2019 independently conducted report by Ponemon Institute, the global average of a data breach rose 6.4 percent to \$3.86 million, while the cost per leaked or stolen record increased by 4.8 percent to \$148 [1]. Companies are currently designating an average budget of \$3.6 million to cybersecurity with an increased preference for automated cyber-resilience. From 2017 to 2019, 57% of organizations reported a cybersecurity incident that resulted in significant disruption to the internal processes while 55% reported a data breach of more than 1,000 sensitive and confidential records. There is no single detection method that is capable of detecting and thwarting all techniques of data exfiltration. Instead, by utilizing a conglomerate of methodologies and techniques, companies and professionals can aim to lower their security risk, and ultimately increase their ability to ensure data confidentiality. Data exfiltration is a growing issue and must be addressed with a variety of anomaly detection methods.

There exists simple techniques for detecting data exfiltration. One such technique measures the total data transfer of a network and detects unusual spikes in volume. This method includes many drawbacks. If an attacker plans on transferring a large amount of accessible data, they can exfiltrate it by progressively increasing their data transmission rate over the span of multiple weeks.

We introduce a graph-based statistical inference anomaly detection approach that focuses on detecting data exfiltration through a network. Our method leverages network topology alongside data flow information in order to identify anomalous events based on specific user-defined time intervals. This awareness of network topology provides significant advantages over other simpler techniques that use whole-network anomaly detection. These techniques focus on local or global characteristics rather than relations between network devices. The main contributions of our approach are as follows:

- The method is topology aware by leveraging a graph representation to summarize a network's structural properties in the statistical detection of an anomaly, as opposed to measuring activities for individual or global network components (e.g., single or whole network gigabytes of traffic).
- The method is efficient and can be implemented with existing data collection practices for security monitoring, requiring no new data as long as common sensor data (e.g. netflows) are collected.
- The method can be incorporated into the workflow of cybersecurity analysts enabling both quantitative as well as qualitative assessments of network conditions and provide an intuitive way of explaining traffic in the network.
- The use of graph-based statistical approaches alongside seasonal trend detection to provide an added layer of protection against adversarial machine learning attacks.

The rest of this paper is structured as follows. Section 2 presents related work on anomaly-based detection for data exfiltration. Section 3 describes our proposed methodology. We discuss on how networks can be analyzed using exponential random graph models followed by time series modeling using autoregressive integrated moving average methods. Section 4 presents our experimental design that focused on evaluating our anomaly-based detection approach. Section 5 categorizes our results on empirical and experimental data and evaluates the validity of our findings. Section 6 highlights some limitations for our approach that can assist cybersecurity professionals that aim to implement this approach in their daily workflow. Finally, in section 7, we provide some future challenges and opportunities that can further improve our method.

2. Related works

Common attack scenarios for organizational networks involve exposed database servers or compromised web servers. For example, databases have been found to be accessible through the internet, often with default credentials. However, a more typical example involves an exploitable web server that would provide an attacker with a staging area from which he or she can eventually access confidential data on a database server. Insider threat [2, 3] is another attack scenario where an agent inside the network (i.e. an employee of an organization) can seek to extract confidential data out of the network. Further, attackers that pose an advanced persistent threat may seek to hide their communications in the network in order to not be detected by cybersecurity professionals. The threat of confidentiality breach through data exfiltration is further exacerbated by the exponential growth of the number of mobile and IoT devices that are used by organizations. Data exfiltration attacks can involve mobile phones (e.g., exploiting iOS pairing service [4]), 3D printers (e.g., exploiting and stealing intellectual property through a printer's network functions [5]), and IoT devices (e.g., data exfiltration of casino's participant records via a compromised IoT thermometer [6]).

Regardless of the attack goal, attackers will make strategic decisions to obfuscate their data communications through the network. Network transmissions through common avenues (e.g., SSH connection) is likely to raise alerts for network operators and cybersecurity analysts. As such, attackers often seek to use a covert channel through which they can obfuscate their data exfiltration process. All possible network protocols become possible vectors for data exfiltration between the compromised machine and the attacker [2]. Examples of such behavior include malware (e.g., botnets) [7, 8, 2, 9], where compromised machines must communicate with a command and control center in order to receive further instructions or update their configuration. Communication is often attempted via some common network protocol channel (e.g., DNS request or TCP reserved bit space utilization). Similarly, in a targeted attack that is focused primarily on data extraction, the data must travel through the network of the compromised machine before arriving at the desired endpoint [3].

Most defenses for data exfiltration focus on analyzing network traffic flow through the corresponding flow-based metrics. This is an important avenue of research because it is difficult to obfuscate the signatures relating to data exfiltration through network communication protocols. In other words, the attackers must abide by the rules of the communication protocols in order to transmit their data through them.

The DNS protocol is a channel that has been used for data exfiltrations and has been the subject of study over the past decade. There exist multiple attack vectors that leverage weak points in the DNS protocol, aiming primarily to send data through an unsolicited channel. This method is known as tunneling. The approach is very attractive due the fact that most firewalls do not block DNS queries, which nullifies the first line of defense on a victim's machine. More specifically, DNS tunneling has been researched with a variety of methods to analyze possible exfiltrations [10, 9]. There are also DNS exfiltration malware that use a variety of targeted techniques (e.g., DNSChanger, OilRig) [11, 12, 13].

Existing research in DNS data exfiltration detection falls under two large categories: byte-level analysis of DNS packet flows and analysis of domain name strings in packets. The most rudimentary of examples in this category focus on raising an alert past a number of DNS requests over some time window or by measuring domain name entropy as a crude metric for detecting random strings [14] [15]. However, more comprehensive solutions also exist.

The work presented in [16] details a byte-level analysis of DNS packet flow in a network. Analyzing just DNS traffic and its corresponding protocol, three separate attack vectors were explored and analyzed. These attack vectors represented data exfiltration via a file transfer, an interactive session to mimic command and control communications, and a scenario of web browsing over the DNS protocol. A temporal analysis was applied to the data, which helped analyze the behavior of DNS communications for these attack events over time. Various detection methods were then discussed as a means to effectively combat these attack vectors. One such detection method uses a *time-of-the-day-dependent* threshold (i.e, less traffic is expected during night hours so the detection threshold should be lowered) while an alternative approach looks at various time windows to determine increases or decreases in the average network DNS flow as a means of raising an alert. If the data collected in this work had been analyzed as a time series, both detection methods could be seen to represent aspects present in a typical time series of data where there might exist seasonality and a general trend. However, limitations for these methods also exist. Strictly analyzing the byte flow for DNS traffic leaves the detection method open to a boiling frog poisoning attack [17], where the data is exfiltrated slowly, aiming to progressively raise DNS network traffic and in turn alert thresholds.

Alternate approaches for detecting DNS exfiltration that do not focus on the sum of bytes also exist. Namely, character frequency analysis of DNS domain and subdomain names have provided an alternate approach to detecting a possibly compromised communication channel [10]. Anomalies were quickly discovered when tunneled traffic was compared to legitimate domain traffic by analyzing unigram, bigram, and trigram frequencies of characters. This approach is effective because it has empirically been shown that domain names follow Zipf's law [18], that expects that the frequency of a word is inversely proportional to its position in the frequency table. Using natural language processing these domain name "words" can be categorized and identified. The study demonstrated that by using $n - gram$ frequency analysis, they can detect an English language exfiltration. This approach is novel in that it doesn't consider bytes of DNS flow as a predictive factor for data exfiltration, but instead compares likely domain signatures with patterns in the English language. A shortcoming of this approach is the lack of robustness when considering multiple languages. Additionally, an attacker could obfuscate data extraction leading to a decreased anomaly detection performance for this method.

3. Proposed method

Our proposed method consists of two main components: the *statistical analysis of the network topology* graph using Exponential Random Graph Models (ERGMs) and the *time series analysis of the coefficients* using an AutoRegressiveMoving-Average (ARMA) model. Figure 1 shows the process through which these elements come together.

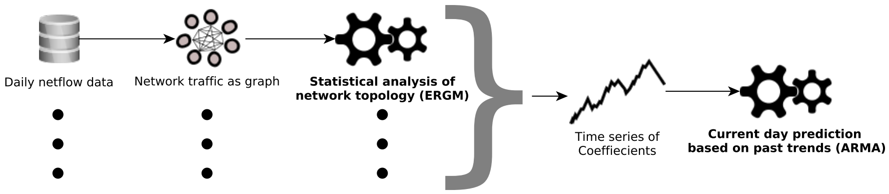
Daily netflow data
Network traffic as graph
Statistical analysis of network topology (ERGM)
Time series of Coeffiecients
Current day prediction based on past trends (ARMA)

Figure 1: A bird's eye view of the components that comprise our topology aware anomaly detection method.

Implementation of the ERGM component involves identifying the type of traffic that needs to be monitored for anomalies and then representing the traffic (obtained from packet captures or netflows) as a graph consisting of nodes and edges. The graph represents a snapshot in time for the network (i.e., cross-sectional data) that would be typically collected in regular intervals (e.g., hourly or daily). ERGM produces coefficients that describe the local properties of the graph. An example of such a local property is the probability of three devices communicating with one another forming a triangle in the graph. These coefficients form a temporal dataset that can then be validated with a seasonally trained ARMA model to classify an anomaly outside of a desired threshold value. For example, if over the course of a week, triangle formation has low probability of occurring in our observed network and that probability suddenly changes beyond a certain threshold (e.g., standard deviation) then, a network anomaly has occurred requiring a cybersecurity analyst to investigate further.

In the following subsections, we describe the two main components of our method in detail.

3.1. Statistical Analysis of Network Topology

Analysis of network topology is effectively an analysis of graphs. The exponential family of functions that are used for statistical analysis of graphs covers a broad range of applications. Exponential Random Graph Models (ERGMs), a subset of this exponential family, characterize graph topology to model and analyze structural properties of networks. A statistical model of a network is used to capture regularities as well as recognize uncertainties. The outcome is represented as log-odds (a coefficient). The general probability formula is as follows [19]: $\mathrm{P(Y = y—}\theta) = \frac{exp\{\theta' s(y)\}}{c(\theta)}$ where Y is a random network of n vertices while y is the observed network, and θ is a vector of parameters associated with $s(y)$, which are similar to regression coefficients. We assume an observable graph's structure y can be explained with a statistic vector $s(y)$ depending on vertex attributes and the observable network. Finally, c is a normalizing constant to ensure probabilities sum to 1. More specifically, $c(\theta)$ can be rewritten as: $\mathrm{c}(\theta) = \sum_{\forall y \in Y} exp\{\theta' s(y)\}$ Each network tie is a random variable and as such we can make a probabilistic prediction for a tie between vertex i and j. The network statistics to be added and counted (e.g. edges, 2-star, triangle) must be known a priori and specified by the user. For each additional term, the model must generate an additional statistical parameter influencing the graph. Figure 2 provides simple examples of types of ERGM terms, which describe structural properties in the network.

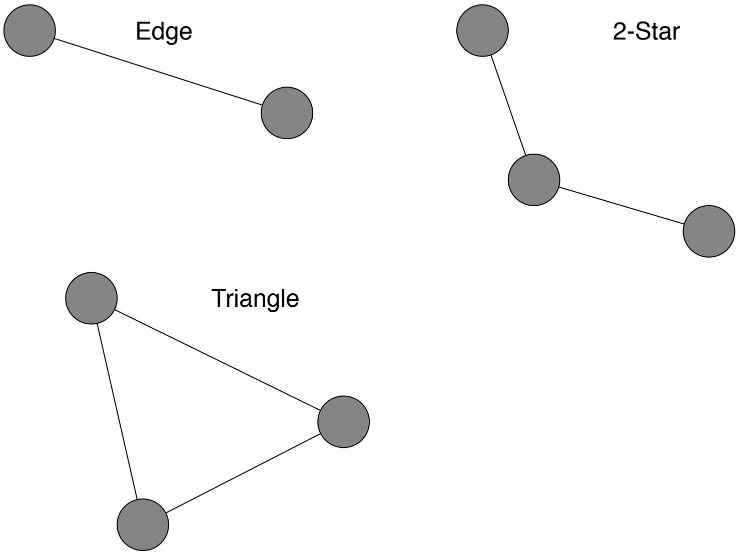

Figure 2: An example of network terms to include when calculating probabilities.

In a directed graph of size n, there are $2^{n(n-1)}$ possible permutations. Due to the size of the domain, calculating statistics of of all possible graphs grows exponentially with the amount of vertices in the network. Instead, ERGM leverages Markov Chain Monte Carlo (MCMC), a family of probability distribution sampling algorithms, to estimate the statistical probabilities. MCMC is heavily relied upon when computing models requiring integration over a multitude of unknown statistical parameters.

In layman's terms, ERGM provides the statistical probability for user specified structural properties of an observed network based on a large sample of networks that are equal in size to the observed network. This way, we can estimate the probability of triangle formation in an observed network, which in turn, informs us of how rare or common that graph property is.

Although originally intended to be used with social graphs (sociographs), ERGM can also be used for network topology analysis since computer networks can be mapped in graphs. For example, the flow of a network can be captured across multiple protocols and can be extended into a graph representation of nodes and edges. This relationship can then be analyzed using a variety of statistical user specified terms.

The *statnet*[19] package available through R [20] provides the necessary libraries to perform ERGM calculations on an observed network. In our proposed method for network anomaly detection, the library *ergm.count* in the *statnet* package is used to handle the introduction of valued edges. This allowed us to represent not just the network topology on a graph but also the traffic that traversed through different network paths. The process for estimating coefficients for valued networks is similar with the only difference being the need to sample estimates for the sampled random networks' edges. The sum of all edge values is polled from one of the user specified distributions available from the *ergm.count* package (e.g. Geometric, Poisson, Bernoulli).

3.2. Time Series Analysis of Coefficients

The aim of time series analysis is to find patterns in noisy data[21]. Typically, such data are empirical (e.g., real world network traffic) and are in part the result of non-deterministic processes. More specifically, these data may involve events that take place over time and are measured at non-random intervals. Examples of time series data include stock market trading data, sport metrics, weather forecasting, etc. The value in modeling and predicting such systems is obvious, but the difficulty lies in discovering a ground truth that can be isolated from the noise. The largest contributor to this difficulty is stationarity. In a time series, the random variable of the distribution has stationarity if its mean, variance, or covariance change over time. This characteristic is the critical factor in the performance of the model's ability to forecast. It would be extremely difficult for a model to accurately predict values for a system that is truly random. Luckily, there are several methods of data manipulation that can take raw non-stationary data and transform them to become stationary.

An ARMA model is a method for time series analysis that is aptly suited for weakly stationary, stochastic time series data. That is, data with a constant mean, variance and autocorrelation. ARMA models are a combination of two separate time series analysis techniques, namely autoregession and moving average. The first component of ARMA (AR) is represented by this autoregressive process, where a prediction at time t is influenced by past observations and a random white noise component. An AR model is defined as follows [22]: $\mathrm{x}_t = \zeta + \varphi_1 x_{t-1} + \varphi_2 x_{t-2} + \varphi_3 x_{t-3} + ... \varepsilon$ where ζ represents a constant (intercept), ϕ_1, ϕ_2, and ϕ_3 are the autoregressive model parameters, and ε a random noise component.

The second component of ARMA (MA) consists of the moving average process, also known as smoothing. This process is generally used when analyzing data that might have trends or exploring how different time windows might affect trends in the future. The moving average process aims to remove noise from observations in order to get closer to the underlying trend of the data. This process can be described as follows[22]: $\mathrm{x}_t = \mu + \varepsilon_t - \theta_1\varepsilon_{t-1} - \theta_2\varepsilon_{t-2} - \theta_3\varepsilon_{t-3} - ...$ where μ is a constant, θ_1, θ_2, θ_3 are the moving average model parameters, and ε is a random error component.

Combining the autoregressive and moving average equations together yields our ARMA model, also written as ARMA(p, q), defined as follows [23]: $\mathrm{X}_t = c + \varepsilon_t + \sum_{i=1}^{p} \varphi_i X_{t-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i}$ where φ are the parameters of the autoreggresive component, θ are the parameters of the moving average component, c is a constant and ε is the noise component.

We selected ARMA models for our proposed method as a means to raise alerts due to the models having a component specializing in forecasting from past observations and a component attempting to smooth observations by “learning” the underlying trend. Although these models are typically used for forecasting (e.g., a value 7 days into the future), our use of ARMA models focuses instead on verifying how our observed traffic fits our baseline prediction (i.e., is what we are seeing now what we should be seeing). Typical network traffic follows a strict repetitive pattern. This is especially true for many corporate or critical infrastructure networks where the number of computing devices in a network do not change often. Even networks that include public devices (e.g., public wireless) such as those found in municipal networks can have a fairly stable traffic pattern that varies little aside from on and off days[14]. An example of such an observable pattern is the lesser network activity over the weekends, as there are fewer people using a network. It is also expected that such network traffic is unlikely to deviate from a norm (once established). That is, traffic on a particular Monday is unlikely to significantly differ from traffic on a following Monday. The configuration of this model and methodology of raising alerts is discussed in greater detail in section 4.4.

4. Experimental Design

We tested our proposed method in order to evaluate its accuracy and computational performance on empirical (real-world) data. Our scenario of choice focuses on DNS exfiltration attempts. Although this type of a data exfiltration could be detected with reasonably good signatures (e.g., measuring entropy of domain names or number of requests to the same FQDN) [10], it was chosen due to its simplicity as a vignette that can demonstrate the efficacy of our method. In practice, our method can be applied in detecting any traffic that can be projected into a graph of nodes and weighted edges. Further, we posit that even though cybersecurity experts are aware of such obvious attacks, large data breaches are still occurring using DNS exfiltration with the most recent example being Equifax's data breach [24].

DNS requests were captured from a local municipal network in order to produce a realistic network traffic baseline based on which we could test DNS exfiltration incidents. The experimental design consists of four parts. First, we describe our empirical data collection method as well as the injection of DNS exfiltration traffic in our dataset. Next, we discuss the internal validity of the ERGM model and detail the model configuration in order to provide insight into network analysis applied to DNS exfiltration. We subsequently describe the experimental setup for our ARMA model, which used the coefficients produced by our ERGM model for anomaly detection. Finally, we detail the process followed for our classification experiments that produced results of our method's accuracy in detecting anomalies.

4.1. Data Collection

The Public Infrastructure Security Collaboration and Exchange System (PISCES) is an incorporated nonprofit providing free cybersecurity event monitoring for several local municipalities in Washington State [25]. PISCES partners include Global Business Resources, Cyber Range Poulsbo, the Washington State Fusion Center and the Department of Homeland Security Science and Technology Directorate. The participating local governments transmit network traffic (in the form of netflow, IDS alerts and DNS records) to be monitored in exchange for no-cost analysis, protection, or best course of action recommendations. PISCES stores the resulting data in an ELK stack.

Since PISCES's beginning of operations in March 2018, petabytes of network traffic data has been collected and organized. However, due to the nature of a deployed system, crashes, bugs, and other oddities occasionally occur during the system's uptime causing numerous gaps in the collected data. For an ARMA model to provide effective feedback using a small dataset, the sample must be contiguous and must reflect a standard traffic flux void of anomalies. In order to guarantee this requirement, a week of data for which quality was manually evaluated to be satisfactory was selected, and then proceeding weeks were generated using statistical noise. This way, we were able to preserve continuity without having to "cherry pick" traffic from different calendar days far apart from one another.

It is important to note, the gathered network traffic is dependent on the days of the week. Weekends experience much less volume of DNS requests while Monday typically has the highest volume. There exists a seasonal trend of network traffic in which weekdays and weekends are distinguishable by their volume of DNS requests and resulting coefficients. Using the gathered ERGM log-odds from the empirical week as a baseline, new data was generated by introducing a randomized multiplicative factor between 0.8 and 1.2. This range was used due to observing the range of ergm values due to the lowest and highest traffic accordingly. The resulting values generated by this process were further evaluated based on whether they reasonably corresponded to a similar empirical day of the week. The result positively satisfied this requirement. Figure 3 shows a time series of an ERGM coefficient that includes the first week based on empirical data and the subsequent generated weeks using the aforementioned process.

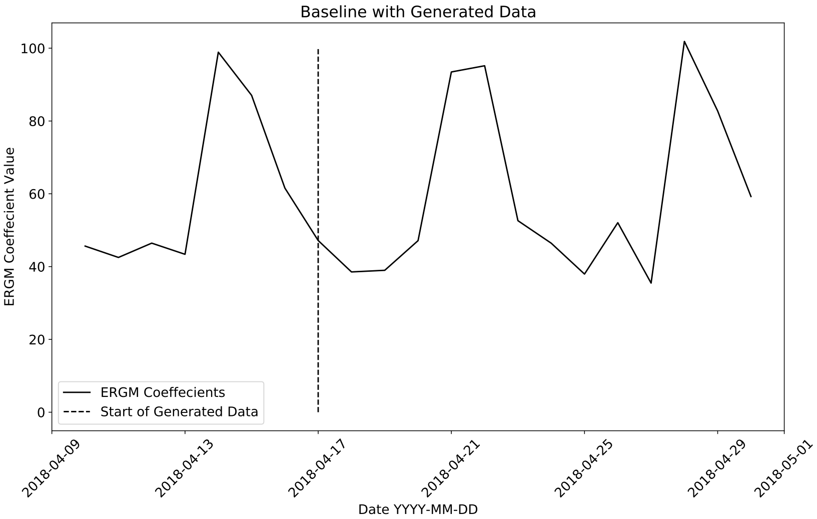

Figure 3: One empirical week followed by two generated weeks of data.

4.2. Generating DNS exfiltration

Once a time series of ERGM coefficients based on our baseline network traffic was generated, we proceeded by designing different DNS exfiltration scenarios for various days of the week. In table 1, we can see that a 10MB of DNS exfiltration corresponds to approximately 43,690 additional DNS requests added to edges in the graph. This is due to the limitation that DNS exfiltration has to use the available subdomain space of a domain request, which in practice is less than 253 bytes. For the purposes of our experiment we assumed that the domain name is 13 characters long and that the attacker has 240 bytes of available space to exfiltrate data from.

For simplicity, we ignore the fact that often Base64 encoding is used for such purposes (due to non-ASCII compatible characters in the data) that further

DNS Volume to Request		
Volume	Approx. No. Requests	2 Hops
10MB	43,690	87,380
50MB	218,453	436,906
100MB	436,906	873,812
500MB	2,184,533	4,369,066
1GB	4,473,924	8,947,848

Table 1: The approximate number of requests needed for an attacker to exfiltrate volumes of data.

inflates the number of requests needed. Additionally, because of the presence of DNS relays in network topology, a DNS request may traverse through multiple nodes in a network. In our graph representation of network DNS traffic we also included the public DNS server which further increases the amount of DNS requests that would appear in the graph should a DNS exfiltration occur. In fact, as data extraction affects every edge that it travels through, the further away from the public DNS a request is placed, the more "noise" it would generate in the graph. This in turn will affect the probabilities of a graph's structural properties that are generated by ERGM. Figure 4 shows a simple graph that demonstrates the aforementioned scenario. It is clear to see the more isolated a node is in the network, the more "noisy" an exfiltration attempt will be as more edges are affected by the outflux of requests. The effect of this behavior can be pronounced in larger graphs as the established paths rarely change.

Figure 5 shows a graph that includes DNS relay nodes as well as public DNS nodes. Black nodes are associated with IP addresses with outgoing requests, gray are major internal routers and white represent public DNS servers. This cross-sectional graph represents one day capture of DNS requests on the traffic of the municipal network that PISCES monitored.

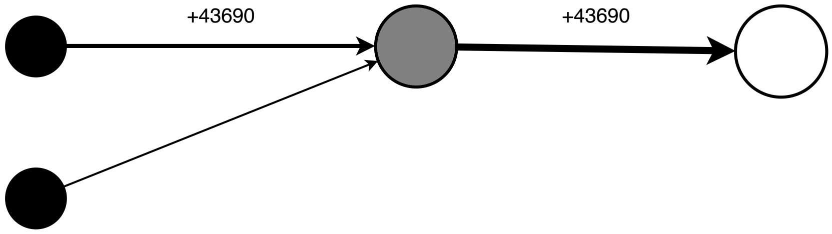

Figure 4: An example of a 2-hop flow increase in a network. This is a typical DNS traffic where the middle node is a DNS relay node that forwards or DNS requests on behalf of clients to a public DNS server.

4.3. ERGM Configuration

Since the ERGM methodology is probabilistic, we ran repeated tests in order to verify that the coefficients produced for the same graph by ERGM are stable (i.e., do not vary substantially for the same graph). With an adequately large predefined sampling size and a sufficient burnin value (where the initial data generated from MCMC are discarded), coefficients converge to a satisfactory variance. It is these coefficients that were then randomized with the predefined statistical noise described in section 4.1 to produce a generated day of traffic for a similar day of the week.

We built the representation of network DNS traffic as a directed graph with valued edges. We used two statistical terms for our ERGM model in order to maintain a computationally lightweight MCMC estimation. The first term, *sum*, was used with a geometric distribution to represent probabilities associated with valued edges in a network. This term measures the probability of a single unit increase for a given potential edge in the network. The second term that we used, $atLeast(x)$, measures the probability of finding an edge in the graph with at least x weight. x is the threshold value that is required to determine the log-odds and it was provided by us. In the case of our experiment, the threshold value for *atLeast* was set to the arithmetic mean of all edges in our gathered empirical time series. This is an important hyperparameter to consider when deploying this method if this particular ERGM term is used. Additionally, it is important to note that by establishing a static threshold value using a normal

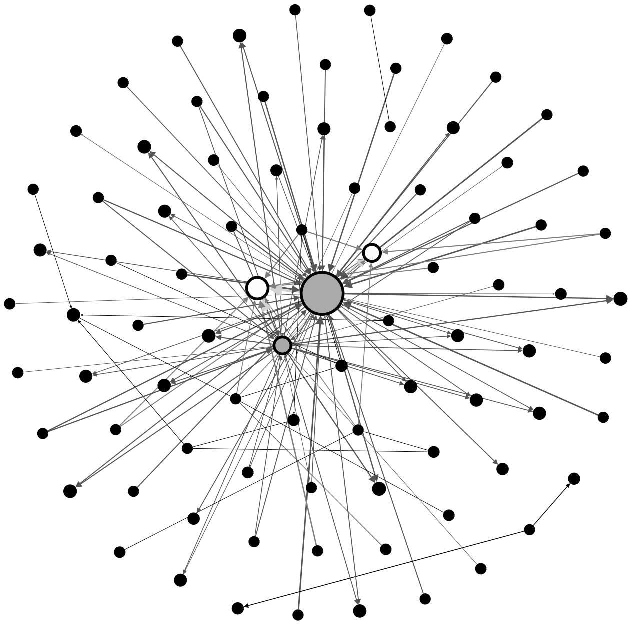

Figure 5: A graph representation of DNS requests over a 24-hour window. Black vertices indicate requests being sent, gray vertices are major internal routers, and white vertices indicate public DNS services.

week of data, it safeguards against possible adversarial machine learning attacks. For example, in the case of the boiling frog scenario[17], a gradual increase of traffic will not affect the previously defined hyperparameter value unless it is automatically or manually adjusted at time intervals. As a result, this type of attack will eventually be detected as an anomaly. The log-odd coefficients produced by these two terms were then passed onto the ARMA model.

4.4. ARMA Configuration

Once the ERGM estimation phase of data analysis was completed, we utilized the log-odd coefficients produced by the model to form a time series dataset than can be used by ARMA for prediction. Due to the small sample size of observed days in our sample (14 weeks), any amount of significant noise greatly affects the stationarity of the data. To combat this, a method of differencing was used where a rolling $z-score$ of a seven day window was subtracted from the observed value. Augmented Dickey-Fuller unit root tests confirmed the stationarity of the transformed time series data within a 99% confidence interval.

The *pmdarima* [26] software package was used to perform grid search hyperparameter optimization for ARMA focusing on the minimization of the model's Akaike Information Criterion (AIC). Note that this software package can be used to fit ARIMA models, also seen as ARIMA(p, d, q). However, since data deflation was to be tightly controlled, in order to achieve stationarity, the differencing term d was omitted.

Although large training data (e.g., a year worth of ERGM coefficients derived from daily network graphs) may intuitively mean better predictions, we posit that a) it is practically unfeasible and even excessive to collect such a large dataset, and b) it may hinder the prediction ability of an ARMA model due to variance caused by natural changes in computer networks over the course of a year. We recommend that for a more realistic model evaluation for daily traffic, a training set should consist of a few weeks of traffic if a weekly seasonality is assumed. The number of observations in the test set should also be kept small due to the decreasing prediction confidence of the ARMA model as time progresses from the start of a prediction. In practice, our method is expected to be implemented without any forecasting. Instead, a given current observed traffic would be triangulated with the predicted traffic for any current moment.

The alert threshold used in our proposed method is a threshold of varying standard deviation over the training period. Due to the nature of the time series, this method of detection incorporates temporal variations in the threshold. As such, it allows for a higher threshold on days typically associated with higher volumes of traffic. An alert is raised if the observed value is outside the configured threshold from the predicted value. In our experimental design, this translates to the log-odd ERGM coefficient being outside the baseline established by the ARMA model. Figure 6 shows an example of predictions made by a trained ARMA model along with the alert threshold of 1 standard deviation (shown as an error bar). The dashed lines for the final two weeks of the time series represents the observed traffic coefficients. The traffic is considered normal for these days since the observed and predicted traffic does not deviate beyond our alert threshold.

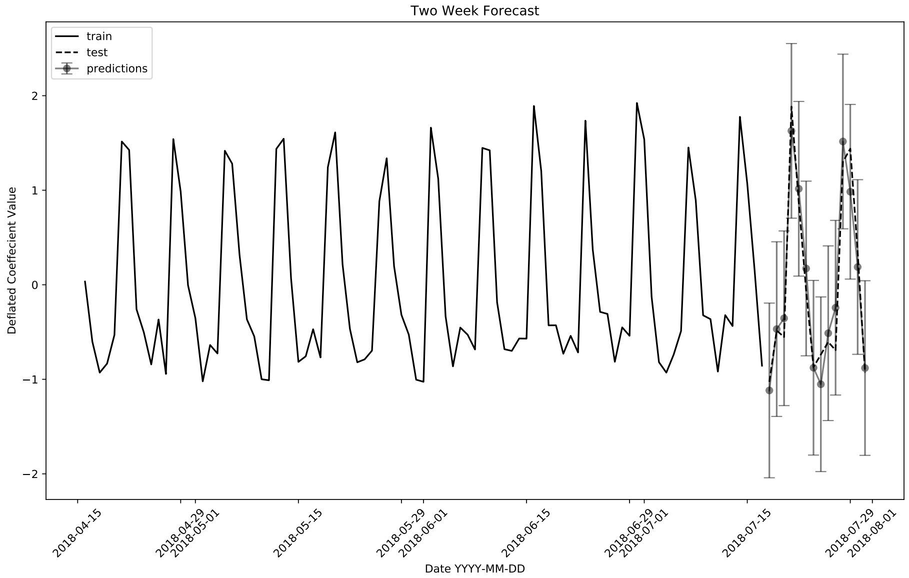

Figure 6: ARMA Model fit using approx. 3 months of training data and predicting over a two week period with 1 std. error bars. Observed traffic is shown in dashed lines for the predicted two weeks. Predicted and observed coefficients do not deviated beyond 1 std. error bar.

4.5. Experimental Procedure

We developed three data exfiltration scenarios based on data volume for the purposes of this experiment. Low volume data exfiltration was defined as any amount in the range of 10MB to 100MB. Medium corresponded to 100MB up until 500MB. Lastly, high data exfiltration volume was any amount of data between 500MB and 1GB. In terms of data exfiltration volume, all these categories are relatively conservative. In past incidents of data exfiltration, multiple gigabytes of data are typically exfiltrated out of a network [6].

Our experiments aimed to identify the classification accuracy of our approach on a) detecting incidents of DNS exfiltrations, and b) classifying days with normal traffic correctly. To generate a malicious attempt, the observed network for a given day of the week was injected with requests corresponding to each value of exfiltration in table 1. A two-hop process was used as the basis for DNS exfiltration. This is a typical network path that DNS requests follow in many networks (see figure 4 for an understanding of a two-hop increase). Seven normal week days from our generated set were selected as candidates for testing our method's ability to detect anomalies. These days were then injected with DNS traffic at random. The graphs containing the injected DNS traffic (representing data exfiltration) were then used as ERGM input in order to calculate the statistical terms (mentioned in section 4.3) and extract the coefficient values.

We trained our ARMA model on 14 weeks of generated traffic that was based on our initial PISCES data. Then a testing week was randomly generated as normal traffic (see figure 6 for visual depiction of this approach). Then a random day in the testing week was replaced with a randomized coefficient value relating to the volume of DNS exfiltration based on table 1. Then the trained ARMA model would predict the testing week. Observed as well as predicted coefficients would then be compared in respect to our alert threshold values. Results were obtained for three separate alert threshold values in order to demonstrate how fine-tuning may affect the detection accuracy of our approach. These threshold values were: a) one standard deviation, b) one half standard deviation, and c) one quarter of a standard deviation.

This process was repeated for a total of 500 times generating a testing week each time that was comprised of six normal days of traffic and one day that DNS exfiltration was attempted. Put simply, our experiments contained 3000 normal and 500 abnormal day classification attempts for every alert threshold (e.g., one standard deviation) and every DNS exfiltration category (e.g., low data exfiltration, 10MB to 100MB).

5. Results

In this section we show the results of our experiment for our proposed method. We describe below the accuracy performance results and highlight the computational requirements for our method.

5.1. Performance

For our evaluation we utilized the following common classification metrics: accuracy, precision, recall and F_2 score.

We calculated precision in order to identify the false positive rate for our method. This was defined as follows: Precision $= \mathrm{T}_P \frac{}{T_P+F_P}$ where T_P is the true positives and F_P the false positives.

We subsequently estimated recall in order to identify how well our method will help us avoid false negative incidents. Simply put, how many data exfiltrations the method failed to detect. Recall was defined as follows: Recall $=$ $\mathrm{T}_P \frac{}{T_P+F_N}$ where T_P is the true positives and F_N is the false positives.

We summarized the interplay of precision and recall using accuracy, which we defined as: Accuracy $= \mathrm{T}_P + T_N \frac{}{n}$

where T_P is the true positives, T_N is true negatives, and n is the size of the experimental trials (i.e., the number of days that we tried to identified as normal or anomalous).

The standard F_1 score is the harmonic average of both precision and recall, weighing both of these values equally. In the case of our results, an F_2 score is the preferred method of displaying performance. An F_2 score weighs more heavily on recall providing a better performance metric when evaluating models to detect anomalous behavior. The F_2 score was defined as follows: $\mathrm{F}_{\beta=2} =$ $(1+\beta^2)\cdot\frac{Precision\cdot Recall}{(\beta^2\cdot Precision)+Recall}$ where β is 2 in this example, indicating an F_2 score.

Results of all our experiments are summarized on table 2. There exists a positive correlation between threshold value and accuracy. That is, as the threshold value decreases, accuracy follows, which makes intuitive sense. As the threshold value decreases, the model is more strict about the temporal pattern of the data therefore classifying more false positives and less true negatives. Precision can be observed to have a positive correlation with the threshold value as we would expect. If the system classifies an anomaly, precision details how often that classification is correct. As the threshold value decreases, we increase the classification of false positives attributing to the drop of precision.

DNS Exfiltration Detection Performance (n=3500)					
Volume	Threshold	Precision	Recall	F_2 Score	Accuracy
Low	1 std.	0.644	0.294	0.330	**0.876**
	1/2 std.	0.433	0.524	0.503	0.834
	1/4 std.	0.209	0.78	**0.504**	0.548
Medium	1 std.	0.847	0.622	0.657	**0.93**
	1/2 std.	0.549	0.918	**0.809**	0.88
	1/4 std.	0.263	1.0	0.641	0.6
High	1 std.	0.931	0.712	0.747	**0.951**
	1/2 std.	0.546	1.0	**0.857**	0.881
	1/4 std.	0.262	1.0	0.640	0.597

Table 2: Performance metrics across different volumes of exfiltration with varying threshold values.

Possibly, a more important metric to consider is recall. Recall is representative of a system's "completeness." If a data exfiltration event were to occur, recall details how often it would be caught by the system. In the case of medium data exfiltration, using a threshold of one half standard deviation provided a recall score of .918 (or 91.8%). By restricting the threshold to one quarter of standard deviation, there is not much improvement in recall and in fact precision decreases. As such, this threshold value is an important hyperparameter to consider when implementing our method.

In practice, this indicates a point of diminishing returns in defining a lower threshold value. By increasing the amount of false positives the usefulness of anomaly detection is decreased. As such, an ideal “golden” threshold can be found where recall remains high without substantially influencing precision. In terms of our evaluation metrics, accuracy can be rather deceptive since it focuses in the overall improvement of either precision or recall (much like F_1 score which also represents the harmonic mean between these two metrics). However, in practice, we want to prioritize recall first in anomaly detection to avoid incidents that are costly to organizations.

Detecting DNS exfiltration that is below 100MB (a rare event for data exfiltration) is difficult for our method unless a more aggressive threshold is selected at the expense of increasing false positive rate. However, our method can identify almost all DNS exfiltration attempts above the 100MB data exfiltration threshold (medium and high data exfiltration) with a reasonable amount of false positive alerts. In fact, as the volume of DNS exfiltration increases, the accuracy (F_2 score also increases) indicating that any exfiltration beyond the upper bound of our experiment (1GB) is almost definitively detectable.

5.2. Computational Requirements

Network topology is the main factor affecting the ability of our method to scale. Given that ERGM estimates an exponential number of graph permutations, increasing the number of nodes (i.e., increasing the number of network devices) would exponentially increase the number of possible permutations that need to be computed. However, organizational networks or those of a government building, rarely exceed a reasonably small upper bound (e.g., a few hundred to a few thousand network devices). In our demonstrated experiment, the local municipality had 120 devices. One ERGM estimation of the network using the experimental dataset required roughly 20 minutes of computing time on an *Intel Core* i7-7700 @ 3.60Ghz. As such, this method can be reasonably implemented on an hourly or daily basis. Typically, that is a reasonable time-frame for a cybersecurity analyst to investigate an incident. Faster infrastructure could potentially allow for minute by minute classification of traffic, however, it is unclear on whether that time window may contain enough meaningful information (device communications and seasonality) for our method. Future research will aim to investigate the application of our method to micro scale events.

A further requirement relates to ARMA requiring a minimal time window to predict and train reliably. Additionally, the model only needs to be trained periodically to predict new trends in the network flow. If a data point is known to be void of anomalous behavior, it can be quickly added to the training dataset and retraining is not computationally expensive. Further, ARMA predictions can potentially be made in near real-time.

6. Limitations

There are a few limitations that we need to highlight in regards to our method. These relate to the constraints on the initial data, problems that might arise from ERGM calculations, and assuring best model performance for anomaly detection.

First, the collected input data for our method must be contiguous and devoid of abnormalities. If it is non-contiguous, another time series analysis model (other than ARMA) will need to be used that can account for data gaps. The training data must also be representative of the network traffic's norm. In other words, devoid of any extreme abnormalities such as an outage that makes computers inoperable. This would greatly affect model performance as the model needs to learn the proper network data trend. When training data is rotated (e.g., updated every 7 days with more recent data), the effect of these abnormalities can be minimized. However, it is still worth noting that such network traffic abnormalities will raise an alert as an anomaly. Additionally, other methods may need to be used in order to increase the stationarity of the time series (a requirement by ARMA). Ultimately, this decision would need to be made a priori as there are multiple detrending methods for time-series models [27, 28].

The size of the graph is a large limiting factor for our proposed method. As stated in section 3.1, the amount of permutations of the graph grows exponentially with the amount of nodes added (devices in the network). Therefore, there is a limit to the size of the network where the computation time required by the ERGM process becomes prohibitively excessive.

Finally, configuration of the ARMA model will more than likely require human input. The model will need to be validated before it is deployed. This can ensure that there is enough data for the model to learn the network traffic norm and that model predictions are within reasonable expectations. A final challenge that requires further investigation relates to networks that change structure very frequently. This does not affect the majority of applications of the method but highly dynamic networks will require additional considerations. A radically dynamic structure of a network will reflect a dynamic graph, which in turn will lead to ERGM coefficients potentially becoming unreliable. In turn, ARMA's predictions would not be able to identify a proper network trend. The end result is likely to create an anomaly detection method that is either highly insensitive to rapid network changes or highly sensitive (depending on the thresholds that are defined for anomaly alerting).

7. Future Work

Future work should explore the method's detection accuracy for data exfiltration in a real-time implementation. We anticipate that the mean and variance of any real-time time series is likely to change over time and become non-stationary, which is an important condition for the goodness of fit of ARMA. Studies need to investigate how further manipulation of the data and relationships should be approached in such incidents.

Furthermore, real-world traffic introduces more network noise requiring a more finely tuned approach of parameter values for the models. For example, a network's traffic constant growth (e.g., more workstations added to the network) will result in a steady increase of network traffic leading to the mean and variance of transmitted bytes to increase. This would result in a decrease in the stationarity of the training data used in the ARMA prediction model. One approach that can resolve this challenge is the process of retraining the ARMA model periodically. This approach can help predictions stay accurate in relation to the actual underlying network topology and developing trends.

A further focus of our work aims to adapt the methodology to other network compromises other than data exfiltration over the DNS protocol. For example, attackers can utilize protocols such as HTTP and HTTPS traffic for command and control communication as well as data exfiltration[29]. Network graphs for HTTP and HTTPS netflow relations is different from DNS traffic. In HTTP (or HTTPS) traffic, the graph of observable vertices is substantially larger if external IP addresses were to be included. In turn, ERGM coefficient estimation becomes prohibitively computationally expensive (e.g., 30 hour run time). A potential approach that can reduce the size of these graphs (e.g., derived by HTTP traffic) is by classifying any address outside the observable local area network as a single supernode (i.e., aggregating external network nodes into one) or by clustering this external subnets based on some other networking factor (e.g., autonomous system number).

8. Conclusion

In conclusion, companies and organizations are becoming invested in building cyber-resilience as data exfiltration attempts become increasingly frequent [1]. New methods for anomaly detection can help cybersecurity analysts secure the infrastructure of organizations. The method proposed in this paper serves as an additional tool to analysts that can assist in detecting data exfiltration. We have demonstrated the efficacy of our method in an example scenario of data exfiltration and further exploration will determine the accuracy of the method in other domain of application. Our hope is that our methodology can be useful in reducing the possible workload of cybersecurity analysts by allowing them to only investigate truly anomalous network events.

[1] Ponemon Institute, The 2019 Study on the Cyber Resilient Organization, Tech. rep. (2019).
URL https://www.ibm.com/downloads/cas/GAVGOVNV

[2] A. Giani, V. H. Berk, G. V. Cybenko, Data exfiltration and covert channels, Sensors, and Command, Control, Communications, and Intelligence (C3I) Technologies for Homeland Security and Homeland Defense V 6201 (May 2006) (2006) 620103. doi:10.1117/12.670123.

[3] Y. Liu, C. Corbett, K. Chiang, R. Archibald, SIDD: A Framework for Detecting Sensitive Data Exfiltration by an Insider Attack (2011) 1–10doi:10.1109/hicss.2009.390.

[4] C. J. D'Orazio, K. K. R. Choo, L. T. Yang, Data Exfiltration from Internet of Things Devices: IOS Devices as Case Studies, IEEE Internet of Things Journal 4 (2) (2017) 524–535. doi:10.1109/JIOT.2016.2569094.

[5] Q. Do, B. Martini, K. K. R. Choo, A Data Exfiltration and Remote Exploitation Attack on Consumer 3D Printers, IEEE Transactions on Information Forensics and Security 11 (10) (2016) 2174–2186. doi:10.1109/TIFS.2016.2578285.

[6] O. Williams-Grut, Hackers once stole a casino's high-roller database through a thermometer in the lobby fish tank, https://www.businessinsider.com/hackers-stole-a-casinos-database-through-a-thermometer-in-the-lobby-fish-tank-2018-4. Accessed on: 2019-06-04 (2018).
URL https://www.businessinsider.com/hackers-stole-a-casinos-database-through-a-thermometer-in-the-lobby-fish-tank-2018-4

[7] A. Al-Bataineh, G. White, Analysis and detection of malicious data exfiltration in web traffic, Proceedings of the 2012 7th International Conference on Malicious and Unwanted Software, Malware 2012 (2012) 26–31doi:10.1109/MALWARE.2012.6461004.

[8] T. W. Fawcett, C. J. Cotton, W. D. Sincoskie, Detection of Data Exfiltration Using Entropy and Encryption Characteristics of Network Traffic, Ph.D. thesis, University of Delaware (2006).

[9] N. M. Hands, B. Yang, R. A. Hansen, A Study on Botnets Utilizing DNS, in: Proceedings of the 4th Annual ACM Conference on Research in Information Technology - RIIT '15, ACM Press, New York, New York, USA, 2015, pp. 23–28. doi:10.1145/2808062.2808070.
URL http://dl.acm.org/citation.cfm?doid=2808062.2808070

[10] K. Born, D. Gustafson, Detecting DNS Tunnels Using Character Frequency Analysis, SciencePrimer.com (2010) 1arXiv:1004.4358.
URL http://scienceprimer.com/boyles-lawhttp://arxiv.org/abs/1004.4358

[11] Y. Nadji, R. Perdisci, M. Antonakakis, Still Beheading Hydras: Botnet Takedowns Then and Now, IEEE Transactions on Dependable and Secure Computing 14 (5) (2017) 535–549. doi:10.1109/TDSC.2015.2496176.
URL http://ieeexplore.ieee.org/document/7312442/

[12] P. Paganini, Analyzing OilRig's malware that uses DNS Tunneling, https://securityaffairs.co/wordpress/84125/apt/oilrig-dns-tunneling.html. Accessed on: 2019-06-04 (2019).
URL https://securityaffairs.co/wordpress/84125/apt/oilrig-dns-tunneling.html

[13] A. Nadler, A. Aminov, A. Shabtai, Detection of malicious and low throughput data exfiltration over the DNS protocol, Computers & Security 80 (2019) 36–53. doi:10.1016/j.cose.2018.09.006.
URL https://linkinghub.elsevier.com/retrieve/pii/S0167404818304000

[14] M. P. Collins, Network security through data analysis from data to action, 2nd Edition, O'Reily Media Inc., 2017.

URL https://www.oreilly.com/library/view/network-security-through/9781491962831/

[15] C. Sanders, J. Smith, Applied Network Security Monitoring: Collection, Detection, and Analysis, Elsevier Science, 2013.
URL https://books.google.com/books?id=TTIDAQAAQBAJ

[16] W. Ellens, P. Żuraniewski, A. Sperotto, H. Schotanus, M. Mandjes, E. Meeuwissen, Flow-Based Detection of DNS Tunnels, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 7943 LNCS, 2013, pp. 124–135. doi:10.1007/978-3-642-38998-6_16.
URL http://link.springer.com/10.1007/978-3-642-38998-6{_}16

[17] J. Tygar, Adversarial Machine Learning, IEEE Internet Computing 15 (5) (2011) 4–6. doi:10.1109/MIC.2011.112.
URL http://ieeexplore.ieee.org/document/6015575/

[18] L. A. Adamic, B. A. Huberman, Glottometrics, Glottometrics 3 (1) (2002) 143–150.

[19] M. S. Handcock, D. R. Hunter, C. T. Butts, S. M. Goodreau, M. Morris, statnet: Software Tools for the Representation, Visualization, Analysis and Simulation of Network Data Mark 24 (1) (2008) 1–11.
URL http://www.jstatsoft.org/v24/i01/paper{%}5Cnpapers2://publication/uuid/8CCE9B1B-3345-4D85-87FE-B2A8D6E8F50E

[20] R Core Team, R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria, https://www.R-project.org/. Accessed on: 2019-06-04 (2018).
URL https://www.R-project.org/

[21] J. Contreras, R. Espínola, F. J. Nogales, A. J. Conejo, ARIMA models to predict next-day electricity prices, IEEE Transactions on Power Systems 18 (3) (2003) 1014–1020. doi:10.1109/TPWRS.2002.804943.

[22] J. Torres, A. García, M. De Blas, A. De Francisco, Forecast of hourly average wind speed with ARMA models in Navarre (Spain), Solar Energy 79 (1) (2005) 65–77. doi:10.1016/j.solener.2004.09.013.
URL http://www.sciencedirect.com/science/article/pii/S0038092X04002877https://linkinghub.elsevier.com/retrieve/pii/S0038092X04002877

[23] C. Peng, M. Xu, S. Xu, T. Hu, Modeling multivariate cybersecurity risks, Journal of Applied Statistics 45 (15) (2018) 2718–2740. doi:10.1080/02664763.2018.1436701.
URL https://doi.org/10.1080/02664763.2018.1436701https://www.tandfonline.com/doi/full/10.1080/02664763.2018.1436701

[24] D. Williamson, Protecting networks from DNS exfiltration, https://www.helpnetsecurity.com/2017/10/02/dns-exfiltration/. Accessed on: 2019-06-04 (2017).
URL https://www.helpnetsecurity.com/2017/10/02/dns-exfiltration/

[25] M. Hamilton, CI Security Partners with PISCES to Provide a Public Option for Cybersecurity Monitoring, https://ci.security/news/article/ci-security-partnership-pisces-cybersecurity-monitoring. Accessed on: 2019-06-04 (2018).
URL https://ci.security/news/article/ci-security-partnership-pisces-cybersecurity-monitoring

[26] T. G. Smith, pmdarima, https://pypi.org/project/pmdarima/. Accessed on: 2019-06-04 (2019).
URL https://pypi.org/project/pmdarima/

[27] B. Podobnik, H. E. Stanley, Detrended cross-correlation analysis: A new method for analyzing two nonstationary time series, Physical Review Letters 100 (8) (2008) 1–4. doi:10.1103/PhysRevLett.100.084102.

[28] J. W. Kantelhardt, S. A. Zschiegner, E. Koscielny-Bunde, S. Havlin, A. Bunde, H. E. Stanley, Multifractal detrended fluctuation analysis of nonstationary time series, Physica A: Statistical Mechanics and its Applications 316 (1-4) (2002) 87–114. `arXiv:0202070v1`, `doi:10.1016/S0378-4371(02)01383-3`.

[29] J. Parfet, Conducting and Detecting Data Exfiltration [Blog Post], https://www.mindpointgroup.com/blog/operations/conducting-and-detecting-data-exfiltration/. Accessed on: 2019-06-04 (2018).
URL `https://www.mindpointgroup.com/blog/operations/conducting-and-detecting-data-exfiltration/`