Background

Company users require access to the internet to do their job. Although, the internet has a vast amount of looming threats. With the size of corporations and how much data is produced, it is impossible to search through every event or log to judge its threat level. Companies emphasize security so possessing the ability to identify malware threats is important. Therefore, I am going to analyze how the researchers at HP Labs used a large-scale graph inference by constructing a host-domain access graph to identify malicious domains.

HP labs

Researchers at HP Labs modeled detection as an inference problem by creating very large graphs with event logs companies already record. They seeded the graph with minimal truth; a small fraction of domains is labeled as malicious or benign and the rest are unknown. Then a belief propagation is adapted and if a domain is identified to be over the threshold, it is marked as malicious. Belief propagation was used because it is fast approximation that scales to larger graphs. They achieved results which present with over 90 percent accuracy and a low false-positive rate.

Overview

This algorithm detects malicious nodes based on a probability because they can be associated with malicious neighbors. To do this a graph is constructed from event logs recorded by a company. The graph shows connections between hosts and domains. A host will be a node on the known network and a domain will be any external node while an edge will indicate a connection between host and domain. Then a belief propagation is applied on the graph to calculate the probability of a node being malicious. The purpose of the belief propagation is to uncover malware communications that would normally go unnoticed. A belief propagation is used compared to calculating the probability distribution for every node.

**Constructing Host-domain Access Graph**

Host-domain access graph

The host-domain graph will represent who queried what. Therefore, to construct the graph the domain and source IP address is extracted from the resource records. The source IP represents a host node and the domain requested represents a domain node. The host nodes are added to a set V1, the domain nodes are added to a set V2, and any connection between a host to domain node is added to an edge set E. Therefore, the resulting graph is G = ({V1}, {V2}, {E}), where V1 is the set of hosts, V2 is the set of domains, and E is the edges between them. A graph can be constructed from DNS records or HTTP record as both have a source IP attached and destination which can be mined.

Now that the graph has been constructed based on the message requests, the nodes need to be initialized. There are three main kinds of nodes; malicious, benign, and unknown. Malicious nodes are initialized using publicly available blacklists or privately acquired blacklists. Likewise, benign nodes are initialized with a whitelist of nodes that are known to be not malicious. There are online rankings like popular domain lists or the Alexa ranking which can be used to initiate benign nodes. The remaining nodes are initialized with values of malicious as .5, and benign as .5 which would indicate an unknown state.

Although the construction of the graph only considers the request messages, the edges are bi-directional since the intent is to identify malicious domains alike which could be command and control servers and send instructions to the infected host.

Bi-Partite Graph

A graph is called r-partite if it partitions every end of a connection into a separate set called r classes. Bipartite graphs cannot contain odd cycles which is a key characteristic. A bipartite graph is complete because when it is constructed a node is added to the set only if it appears as a host or domain with an edge for the connection. Although almost nearly impossible, one thing to note about bi-partite graphs is that there is not always overlapping connections. Every connection in the graph can be unique, but in the case of a host-domain graph this is nearly impossible due to the volume of events.

HTTP Message Format

HTTP messages show a connection between a host and domain. An HTTP request message has a request line and is followed by header lines. A request line has method field, URL field, and HTTP version field. The method field indicates the action of the request, most of the HTTP requests will have a post method to indicate the protocol is providing data. The URL field indicates the address which the host is trying to reach in the domain. The URL is different from the domain name in that it is a specific location the host is trying to reach. The header liner “host” indicates the domain. These are the main lines of data that are important for parsing and constructing the tree as they lead to the creation or connection of nodes. A header line that could be useful for future implementation would be the accepted-languages line, while it may not be indicative, because if a HTTP request is trying to reach a foreign server not frequented on the network of nodes, then it could potentially indicate a higher probability of initial maliciousness.

DNS Message structure

The DNS message is crucial for the construction of the graph, therefore understanding the structure is important. A DNS message is part of a series of protocols therefore its information and data are in a specified format. This makes the message easy to parse and extract the required fields.

The header of the message is always 12 bytes and contains a 16-bit number identifying the query. The flag contains a 1-bit indicator to indicate whether it is a query or response. For graph construction purposes, we want to focus on messages with a bit of 0. The rest of the flags determine other protocols the message might want the DNS protocol to perform, but for our purposes can be ignored.

The question section contains information about the query. It has a name field which has the domain being requested. This section also contains a field to indicate the type of query being made about the host (i.e. host address, mail server, etc.). The answers field is not used for request or queries, which are the messages being used to construct the graph, but it could be beneficial when adding depth to a domain-access graph as the response could correlate factors to a responses validity. DNS request is an application level protocol, so it does not have the sender’s IP address. Therefore, to know where the request originated, the company saving the DNS events will have to log the source IP with the request.

**Analyzing the Node states through a Belief Propogation**

Inference Problem(Why a belief propagation works)

Inference is determining probabilities of any statement and representing the degree of belief in determined probabilities. Bayes’ theorem shows the relation of two probabilities. Using Bayes’ theorem to calculate probability distribution is called Bayesian inference.

Bayesian inference is based on Bayes’ theorem where: Y is the observations, A is parameter set theta, and pr is densities. P(theta) is the prior distributions, P(y, theta) is likelihood of y under the model in a full probability model, p(theta, y) is “joint posterior probability” which is the distribution of probabilities. It is important to note that p(theta, y) and p(y, theta) do not equate. P(y) is the marginal likelihood of y and is set to a constant. This indicates what y should like look and normalizes the posterior distribution to make sure it has a proper distribution. Thus, the posterior distribution of probabilities becomes proportional to the likelihood of y times the prior observations. The parameter set can be partitioned into the sub-vectors to focus on the components of interest which is called the marginal posterior distributions.

Algorithm for belief propagation

The belief propagation algorithm is used to approximate the probability distribution of a node. In the belief propagation algorithm, m­ij (xJ) is the message to node j about what state the node i should be in. A message is a vector of the same dimensions. The belief at node i is proportional to all messages being received at node i and the current belief at that node. The belief at a node has a constant to normalize the sum of incoming beliefs to 1. Messages sent out from a node are created at that node where the message is the product of all messages coming in. Belief propagation gives the exact marginal probabilities for all nodes in a singly-connected graph. To practically compute marginal probabilities, we start with the edges of the graph for which we know and iterate through from there. The time to compute all beliefs for the nodes is proportional to number of links in the graph which is less than exponential time when calculating marginal probabilities naively. Starting with a principle set of messages, one iterates through until the values converge. Once the values converge, we take the individual component of interest, the malicious belief, and compare it to the threshold value. Nodes that are over the threshold are identified as malicious.

Algorithm Comparison

Constructing the graph is always dependent on the number of inputs. The number of inputs is the volume of event logs that must be parsed. Every log must be parsed to ensure a complete domain-access graph. Because the structures holding the nodes and connections are sets, then we do not need to worry about redundant nodes or connections. Therefore, to most efficiently construct the graph, iterating through all event logs is fastest since there is no need to check for already existing nodes or edges.

Determining marginal probabilities for every node versus applying a belief propagation is where the difference in algorithmic efficiency is important. To determine the efficiency for an iterative algorithm, we must: (1) determine input size, (2) identify algorithm’s basic operations, (3) check if number of times basic operation executes is dependent on only size of input, (4) sum the number of times algorithm executes for a basic operation, and (5) find closed-form formula or establish order of growth. Determining the efficiency class of a recursive algorithm is the similar because one must: (1) Decide on a parameter of input size, (2) identify algorithms basic operations, (3) check if times basic operation executes is dependent on input size, but recursive algorithm is different one must (4) set up recurrence relation and (5) solve recurrence or establish order of growth~~.~~

The Belief propagation algorithm is iterative and so, first it must be identified that the input size is the number of edges. This is because a message is created for every connection between nodes. Then the basic operation is the summation of the marginal belief at a node and the messages received. The basic operation is dependent only on the input of messages from surrounding nodes. Therefore, the number of times the basic operation executes is the number of connections at every node. This shows that the summation is n(n+1)/2 which indicates an order of growth that is at worst equal to O( n2).

Determining the marginal probabilities of every node is recursive. The parameter for number of inputs is the number of nodes because the basic operation will be happening for every node. The basic operation is calculating the marginal probability of all surrounding nodes and summing those up. The number of times a basic operation executes is dependent on the number of nodes neighboring the node of interest. The base case is when the node is not receiving any messages and already has a probability for its state. Otherwise the node will sum on the marginal probability of all its neighbors which it would calculate recursively. The recurrence relation for this algorithm is M(n) = sum of M(n-1) from its first neighbor to its last neighbor and the base case is M(1) = 0. Due to the recurrence relation, the algorithm is proven to be exponential.

A belief propagation has an order of growth of O(n2). While calculating every marginal probability has an order of growth of O(nk) where k is the number of times a node will have to recurse for every neighbor. The order of growth for calculating every marginal probability will take long and not provide actionable results because these algorithms are being run on millions of nodes and edges. A belief propagation’s order of growth is manageable and provides actionable results in a reasonable time. The researchers at HP labs were able to run 15 iterations of belief propagation on a day’s worth of data in 115 minutes, 6 hours’ worth of data in 37.5 minutes, and 3 hours’ worth of data in 16.6 minutes.

**Malware characteristics leveraged**

Malware communication structure

Zhou and researchers confirmed out of 1,172 samples, 93% were turned into bots for remote control. These bots use HTTP-based traffic to receive commands. Some malware families encrypt their URLs to the command and control server. Most command and control servers are controlled by the attackers. Although sometimes they are cloud based. The researchers also confirmed that 138 samples relay phone numbers, 563 samples gather and relay phone numbers, and 43 collect user accounts. These malwares communicate the information gathered to a command and control server. In some cases, the server address is stored in a plain-text file. A malware that turns a host into a bot must receive commands from a command server. There is some bot malware that have a secondary command server which utilizes large scale blogs to push out encrypted updates for the direct-action command server. Researchers also confirmed some malware intentionally cause financial charges. An attacker can cause victims to subscribe to a premium-rate service by sending an SMS message. When confirmation is required, these malwares will respond in the background. 55 of the samples were confirmed to send messages to premium-rate numbers. Identifying that a portion of malware require constant communication is important for identifying malware using a domain-access graph. The more hosts communicate to a suspected malicious website or a website with a confirmed malicious ability, the higher probability the node gets marked as malicious. Understanding the communication structure of malware helps prevent data breaches, data loss, and other security risks.

Command and Control Server

A trade off for a command and control server is visibility versus communication. Servers need to be visible to their bots to receive data but must also avoid detection. A botnet would use DNS because it is more difficult to catch since DNS requests are usually not restricted. When transmitting through DNS, a message from command and control would be 220 bytes per message. This message would be carried in the rdata field of resource records in a DNS message. Message chunks can be encrypted to avoid detection. These characteristics of DNS messages can be checked and affect a node’s probability of being malicious since the event is recorded.

Another characteristic of a possible botnet is domain flux. Hosts can identify their server by using IP address or DNS name. With IP fast-flux bots query a certain domain which is in set of continuously changing IPs. Even though the server has changing IP addresses, the domain name stays the same. With domain flux, each bot independently generates a set of domains that resolve to an IP address, and if the request is denied then the bot moves onto another domain until it works.

**Concerns**

A concern that I could not resolve is determining an appropriate threshold value. The researchers at HP labs did not indicate the threshold value they used to compare to the malicious beliefs. If the threshold value is set too high, then some nodes that could potentially be malicious could be marked as benign which would greatly increase the false negatives. Another concern is not having enough initial node states to justify the resulting node states after the propagation.

**Demonstration**

For my demonstration, I am going to show my ability to apply a belief propagation a graph. I will begin by creating a parser that will properly remove the required fields from a DNS message. For now, I will focus on only DNS and not HTTP messages. Upon successfully parsing the information, I will create host and domain nodes initialized to an unknown state. These nodes will be placed in their corresponding sets. After every message I will add a connection in the edge class to show communication between the nodes. Then I will finish by using an online domain ranking list to initialize benign nodes. Then I will select nodes in the graph to be marked as malicious.

After construction of the host-domain access graph, I will need to properly apply a belief propagation algorithm. This will be my initial knowledge demonstration. I will be able to analyze the results of the converged node states to see what state they are believedto be. I will also be able to apply a belief propagation with different threshold values and analyze how the value affects determining the state of a node and possibly correlate it to the number of nodes and initial knowledge of node states.

**Conclusion**

Malware communication architecture can go unnoticed when analyzing individual event logs since malware strives to communicate in covertly. Therefore, the application of a communication graph based on event logs captures and uncovers communication that could be hidden. A belief propagation algorithm for a graph could be concerning, but it has been proven to provide actionable results. One of the reasons it would not provide actionable results is if it does not have the proper parameters, and for this one would need to analyze their approach and fit parameters to their understanding of ground truth initialization data. A belief propagation provides a realistic and reliable approximation for a nodes probability distribution compared to calculating every single probability distribution which would take too long to utilize the results.