Network Analysis on Live Traffic Targeting Known-DGA Malware Families

Lance C. Young

School of Computer Science

Supervisor: Dr Hassan Habibi Gharakheili

The Problem Domain

Contacting Command & Control Centres

Whenever a machine becomes infected with malware and it intends to retrieve instructions from its creator, or send sensitive data, it must contact a command and control centre (C&C) - a server controlled by the attackers. Specifically, it needs the IP address of the C&C server.

Traditionally, IP addresses were hardcoded into the malware. However, as IT professionals become more cyber-aware, blocking all internet packets going to or from said suspicious IP addresses would result in the attacker losing connection to their infected host.

A popular solution was to instead use DNS queries to find the C&C's IP address and register a domain name accordingly. However, the same issue noted above still applies - a domain name can similarly be blocked.

So the domain generation algorithm (DGA) was born where malware would use this algorithm to generate one of many possible domain names depending on some sort of seed (eg. the current time and/or date). Therefore, the domain name used to contact the C&C always changed.

This meant that it is a lot more difficult for IT professionals to detect malware packets being sent in and to the internet and prevent it from occurring even if they knew one of their machines was infected.

Further Explanation of DGA

The actual implementation of DGA is fluid and subject to many tweaks and changes. The important components of DGA is that like a random number generator, the DGA will generate a domain name from a given seed. This seed is typically the date or time, but there is no restrictions. This seed is then fed into the DGA generating a domain name from a fixed, finite set of possible domain names. This generated domain name attempts to be resolved by a DNS query, either resulting in no response or an IP address of a C&C server. Below is a small subset of some domain names that the Bobax worm generated:

quowesuqbbb.mooo.com
hmhxnupkc.mooo.com
adrcgmzrm.dyndns.org
rffcteo.dyndns.org
ycofnn.dyndns.org
humbjatp.mooo.com
zafkgweeyic.yi.org

On the surface, the DGA is crucial to overcoming IP or domain name firewalling as if an IT department either identifies a C&C IP address or a C&C domain name, the attackers can respectively change the DNS records to point to a new IP address or wait for the DGA to generate another DNS query to another domain name.

To explain further, as there is typically no urgent rush with having an infected host connect to a C&C right away, the attackers typically register an ever-changing subset of all possible domain names their DGA can generate. This constant change makes it very difficult for IT security to prevent the infected host contacting C&C servers and mitigate the risk of damage. Sometimes domain names resolved, and sometimes they failed.

A method for automatically analysing network traffic and preventing machines from either contacting C&C servers or mitigating the damage that said contact can cause is needed.

UNSW Project - Nozzle

Overview of Nozzle

Nozzle is a broad project handled by the School of Electrical Engineering and Telecommunications that acts as an umbrella for thesis or PhD students to research techniques and methods for analysing network traffic to identify or prevent malicious traffic.

One method that Nozzle is particularly interested in studying is the use of machine learning to implement behavioural algorithms for identifying malicious traffic. The goal is to create a model that will act as an algorithm that will look at the behaviours of network flows (eg. average bytes per packet, method uses, duration, etc.) to identify whether the flow is malicious. This algorithm is achieved by letting the model examine all traffic from a host for a period of time that it is assumed to be known as safe. After this learning period, a host-based agent will know which traffic is expected (eg. web browsing, email) and which traffic is an anomaly (eg. SSH).

As with the nature of creating models with machine learning, models have to be focused and guided rather than being relied on solely for solving the problem. A PhD student, Jawad Ahmed, is studying the potential uses cases of behavioural algorithms within the Nozzle project.

Training a Model on Malicious Traffic

Rather than training a model on safe traffic, Jawad is currently researching the effectiveness of training a model on malicious traffic. In order to achieve this, Jawad needs a large data set of malicious traffic.

To acquire said traffic, two things were required: access to real life network traffic; and a means to identify malicious traffic and extract it, without relying on the methods that are trying to be developed in the first place.

As mentioned above, large sets of malware use DGA to contact C&C servers to perform their respective missions. A research institute in Germany, Fraunhofer FKIE, reverse engineered many known public malware families that used the DGA method and extracted all sets of domain names that could possibly be generated. Therefore, if one had access to real life network traffic, one could look at all the DNS responses coming into the network and check if any resolved domain names match to the known list of possible DGA domain names.

Analysing UNSW Live Network Traffic

Under the supervision of UNSW academics and with the university's approval, projects can have access to all network traffic via a tap being sent from and to the UNSW network. The tap duplicates the traffic, providing the perfect means for research projects to have access to real live traffic. It peaks at roughly 15 GB per second of data flowing in and out of UNSW.

It should be noted that the Nozzle project has an ethics approval and is sensitive to the privacy of users.

The COMP9301 Project

Network Analysing Tool - My Contribution to Nozzle

The project will be to create a software tool capable of analysing the real-time network traffic of UNSW to identify traffic from known DGA malware families and record said malicious traffic to then be researched further.

This project could be refined into a software tool that IT departments are able to use to identify the scale of infection by known malware families using DGA on their network.

For the future, if Jawad's model is successful, it could be used in conjunction to help identify malicious traffic of unknown malware families in addition to known families by the nature of behavioural algorithms.

The Project's Results

Currently, the program is running on a UNSW machine and parsing all DNS responses sent into the network and performing a check to identify if it contains a domain name from a known malware family. In addition, it can perform basic flow analysis on network traffic, extracting some behavioural information.

The project can be found at the following GitHub repo: KeyboardMonkey221/COMP9301

The Project's Development

Golang - The Chosen Language

As it was to be a software tool, the language it was written in was a crucial decision. There were two key requirements: speed and library support. Speed for efficiently analysing network traffic so it could handle large flows, and the library support to handle network packets and protocols.

There were two languages I was deciding between: C++ and Golang. Snort 3, an industry standard network Intrusion Prevention System is built in C++ and is capable of handling the largest of network flows. Golang on the other hand has inbuilt concurrency designed to scale. Both are considered to be fast languages with existing networking libraries.

Without an extensive understanding of both languages, both seemed like viable options. Golang was chosen as it is seen to be easier to learn than C++ and it was used in other networking projects at UNSW. Therefore, the learning curve would be smaller, allowing for more time on development.

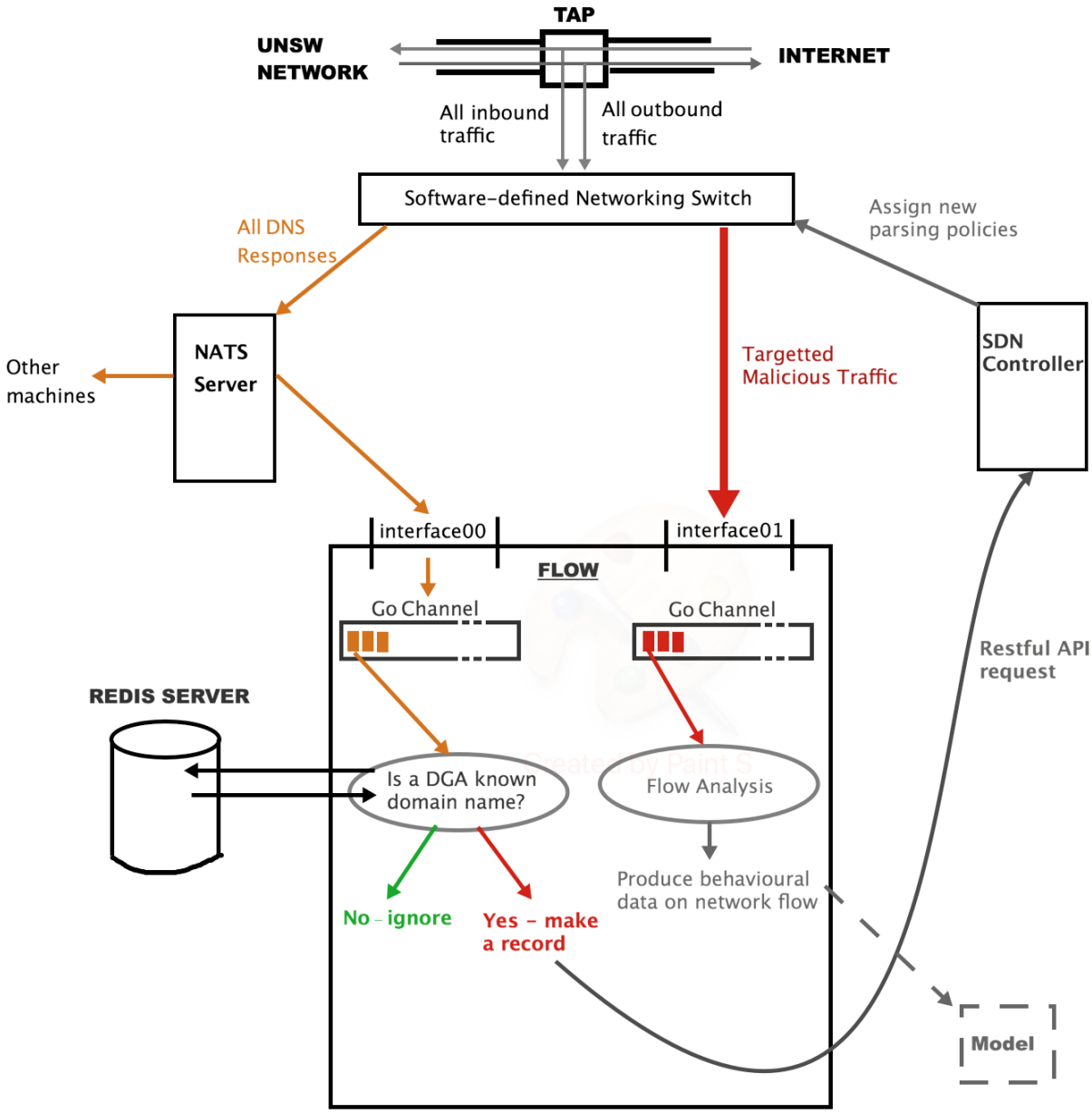
Domain Name Look-Ups

A crucial component of this tool was its ability to check whether a given domain name was known to be generated from a particular malware family. The data set was roughly 5 GB.

A simple SQL-database was not seen as an option. One table of data is only needed (domain name, malware family), and a large amount of queries need to be handled. Typically, databases aren't every performant in handling a large amount of small queries very quickly - to compensate, one can place a load balancing layer between multiple instances of the same database to handle the traffic. Learning how to configure load balancing was not an option.

The solution was an in-memory database. A database that stores its data entirely in RAM. I started to implement said database with a Go library - hashicorp/go-memdb. The library basically implements a radix-tree. This worked well for small datasets of 500MB, however, became increasingly slower the larger the dataset. After searching for alternatives, Redis, an open-sourced in-memory database implementation. Redis completely satisfied all the project requirements. On average for a negative look-up (domain name is not a malware family) it takes approximately 35 microseconds, and a successful look-up is 80 microseconds. With only a generous portion of 5% of all DNS responses containing a DGA domain name, on average, one redis server could handle approximately 26 850 queries a second. With roughly 2000 DNS response packets incoming at peak time, Redis was a clear improvement. Unfortunately due my lack of experience, rigorous data examples comparing the two in-memory databases was not collected, but the improvement in performance was clear.

Technical Analysis of the Software - 'Flow'



Extracting DNS Responses from UNSW traffic

As the rate of data flow through the tap is 15GB/sec, one has to use a Software-defined Networking Switch (SDN) to forward target packets to a location depending on the SDN policy. As the name suggests, the SDN policy's can be programmed and changed dynamically on the fly. Policies for packet forward in non-SDN switches are defined by the manufacturer. The flexibility provided by SDN and the ability to parse large amounts of data (commonly +20GB/sec) means that it is becoming the go-to switch for tech giants like Google who know how to write software, but not build hardware. The SDN at UNSW contains the policy to forward all DNS responses to a NATS server.

NATS is a simple, secure and high performance open source messaging system for cloud native applications. It provides the means for servers to quickly communicate data to each other using a unique protocol that has a lean overhead. As there are multiple networking projects operating at UNSW, servers can easily subscribe and unsubscribe to a NATS server on a particular channel and start receiving messages. One may argue that one would lose performance, though, having the layer between the SDN and one's project allows for mistakes to occur without affecting other people projects.

Therefore, after configuring a subscription to the UNSW NATS server, Flow has a dedicated worker, or go subroutine, listening for messages and placing them into Go channel. It is to be said that multiple workers could exist, placing data from different NATS channels into

the same, or different channels. In addition, multiple workers could exist pulling packets from a channel. A clear advantage of Go which takes care of all the concurrency management.

Performing a Redis Lookup

Effectively, a worker is created to listen and pull incoming packets from the go channel containing DNS response packets. This worker will perform a simple query to the redis database. Either the domain name exists in the database, in which it is a known DGA generated domain name, or it isn't and the packet it ignored. If domain name is DGA generated, then the resolved IP address is extracted and is sent in a Restful API post request to the SDN controller. The SDN controller is then responsible for adding a policy to the SDN to forward packets going to and from that IP address to the machine hosting Flow.

In addition, a timestamp, the IP address of the UNSW internet-facing router that sent the DNS query, the domain name, and whether it was a known DGA domain name. This is simple was done to allow Jawad to keep working without having the complete program cycle complete.

Live/Offline Flow/Wave Form Analysis

For the flow analysis, for each incoming packet, its flow signature is extracted - source IP and port number; destination IP and port number; and finally the protocol being used (TCP, UDP). A hashmap is maintained with the key being the flow signature and the value being a flow entry - a data structure containing the bytes in the flow, the number of packets in the flow, the timestamp of creation, a last updated timestamp, and finally a link to the previous and next flows. Effectively, a double-linked list is being stored in a hashmap.

As incoming packets get parsed, their signatures are used as keys to find the corresponding flow entry. The number of packets in the flow is incremented; the number of bytes in the incoming packet is added to the total in the flow; and the last updated timestamp is updated. If no flow entry exists, one is simply created.

The latest and oldest flow entry is noted. A subroutine exists to constantly check if the oldest entry hasn't been updated for more than the timeout period (currently set to 5 seconds). If a flow entry times out, the flow entry's data is collected and deleted from the hashmap.

The attributes that are currently being analysed is the number of packets in a flow, the total number of bytes in a flow, the average number of bytes per packet in the flow and finally the total duration of the flow. These are considered to be behavioural characteristics of the flow, and in the future, would be parsed to Jawad's model for a decision to be made on whether the flow is malicious or not.

Flows are typically only a few seconds long - anything longer is suspicious. Therefore, this analysis can be done live or offline.

Future Improvements

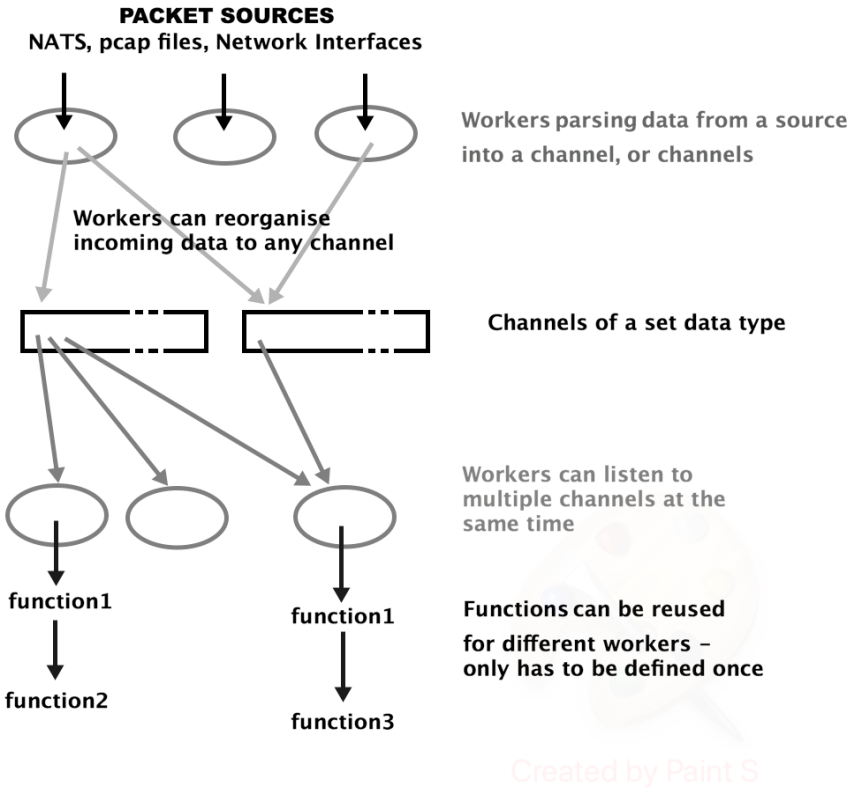
Completing the Cycle

In its current state, there's no link between the model and Flow, and hence, no decisions or actions can be taken. In the future, with more time, this gap can be closed, providing a valuable tool for IT departments.

The flow analysis can be performed on packets incoming on an interface (ie. live), but currently, the SDN isn't set up to parse packets onto the machine running Flow. A stress test would be important to dictate whether Flow can handle the traffic.

Flow's Flexible Design

Below the hood, Flow was not intended to just serve the use-case of Nozzle, but of any network analysis. Currently, there is no general Golang open-source project one can use to help perform any form of live network analysis, even though it can be broken down into various components. Go's inbuilt concurrency system can support said framework.



Under the hood, the software defines ‘flow functions’ that can be added dynamically added or removed from a worker. This software tool could act as a basic network analyser for future projects.

Pulling Packets Directly from the Interface

In its current state, packets are being handled by the kernel before they are given to Flow, and hence, adds on addition overhead. Extracting the data straight from the interface would remove said overhead and improve performance.

Further Testing on Efficiency of the Program

Currently, the program doesn't have much analysis being done on the time taken to perform tasks. With this in place, one can begin to understand what programming decisions would make the program more or less efficient.

Project Reflection

I definitely learnt a lot of fundamentals needed to analyse large amounts of network traffic. With a combination of more time management and more time, I feel as though I would've been able to complete the cycle and have the program give back its performance data.

I'm glad I had the opportunity to learn Golang and work with real life traffic. I definitely see that my lack of experience with managing a system with multiple different servers and coordinating the communication between said servers slowed my progress down.

I hope to continue fleshing out the project into a more complete state and complete the cycle for Jawad.