Passive Network Topology Scanner

Daniel Lockyer
Electronics and Computer Science
University of Southampton
United Kingdom
Email: dl6g14@soton.ac.uk

***Abstract*—The report details the motivation and progress of my networking project to build a passive network topology scanner. Active scanning is often used to probe hosts on a network but it is a noisy method due to the number of generated packets. Passive scanning is not a new idea but most tools are quite old so I wanted to take a modern approach to develop a new one. The resulting application allows the user to visualise the network of packets as captured by client software.**

1. Introduction

During the summer of 2014 I had an internship at a cybersecurity consulting company where I worked on their network intrusion detection system, Countercept[1]. A fellow intern was investigating Supervisory Control And Data Acquisition (SCADA) systems as the company was routinely hired to assess the security of industrial control systems. They looked into reverse engineering the encrypted network configuration files to retrieve the network topology.

Consultants often found that the client's system administrators would not have an up-to-date topology diagram, mapping out the different hosts, as these were often legacy systems installed over 20 years ago. It is important that the consultants have an idea of the network topology so they can ensure the network has been fully assessed.

As a potential solution to this I came up with the idea of passive packet capturing to build up a topology model. I didn't get time to create it and so this project looks at implementing such a tool.

2. Motivation for Passive Detection

Some of these industrial control systems are so old and temperamental that they could crash upon being subjected to an active scan. I must emphasise that this is incredibly rare but even `nmap` discuss this issue on their website (Nmap.org 2017). Nevertheless, these SCADA systems are used to run critical infrastructure such as water distribution centres and nuclear power stations so disturbing these systems can have massive consequences.

With that, the system administrators were reluctant to let the consultants introduce packets into the network. A passive scan differs from an active one in that it only listens to the network traffic, decoding the protocols and extracting relevant information.

2.1. Advantages of Passive

Active scanning tools such as nmap[2] often try and fingerprint the services by sending valid queries or malformed input and checking the response. These legacy SCADA systems may encounter an error when receiving the packet and become unstable. As the passive scan doesn't actively probe hosts on the network, it doesn't add any packets into the network.

Given a good vantage point of the network (ie. a router or firewall), a passive scan can build up a virtually complete overview of the network topology and make a good guess as to the specifics of the hosts and application types.

2.2. Downsides of Passive

As the network isn't probed with packets, only machines that are sending or receiving packets are visible. This means that if a host sends out packets when the sniffer isn't running, the machine will not be known to the application and could get missed.

Secondly, the extent of the traffic seen depends on where the sniffer is in the network. If the scanner is on a single host, it can only see traffic involving that machine or general broadcast traffic. It would be best to run the tool on a machine where traffic passes though, such as a firewall or router.

Finally, depending on the complexity of decoding used, protocols running on non-standard ports or containing malformed data may trick the generated model. In my application this is something I have tried to avoid but I feel I could further improve it.

3. Implemented Solution

For this project I have created a passive network topology scanner and visualisation tool. It is cross platform and decodes common protocols.

1. https://countercept.com/

2. https://nmap.org

3.1. Overview

The application is split into two components; a client and server. Hosts on a network typically only receive traffic designated for their IP or broadcast traffic. The client is intended to run on network routers or firewalls in order to capture the majority of network traffic. A range of clients installed across the network would be the best way of capturing the broadest range of traffic.

The data is decoded using a protocol parser and then sent to the server. The client can also load .pcap files, which is how the majority of testing was completed.

The concept behind how the client works is that it can see all packets going through an interface on the machine. These packets will have a source and destination. When the packet has been decoded, I know there is some sort of communication between the two machines. My application will have limitations; if s packet is TLS encapsulated, I will not be able to decode the contents to fingerprint what sort of data is being sent.

The server would run on a central machine and its role is to accept the data from the clients and provide a web interface to visualise the network topology. It runs a simple web server which provides an API to web clients. The visualisation shows hosts, color coded by type, and linked by the protocol type used to communicate.

The client and server have been implemented using the Rust programming language. It could have implemented in Python or C but I am familiar with Rust as it is being used for my Part III individual project. The web application uses standard web languages and the vis.js[3] network library.

3.2. Packet Parsing

For packet parsing, I used the `peel-ip`[4] Rust library. It worked well but it was limited in the number of protocols that were implemented. I ended up rewriting the majority of the library and added more protocols. These protocols were typically ones that I saw in testing so there are still some key ones missing.

I really enjoyed reading the RFCs and learning about how they are defined. Once finished, I plan on submitting my changes upstream. I implemented the following protocols, using the following resources:

- DNS (Mockapetris 2004)
- NTP (Mills 1985)
- SSDP (Goland 1999)
- NAT-PMP (Cheshire and Krochmal 2013)
- RADIUS (Willens et al. 2000)
- IGMP (Fenner 1997)
- DHCP (R Droms, n.d.)
- DHCPv6 (Ralph Droms et al. 2003)
- ICMP (Postel, n.d.)
- ICMPv6 (Conta and Gupta 2006)

3. http://visjs.org/
4. https://github.com/saschagrunert/peel-ip

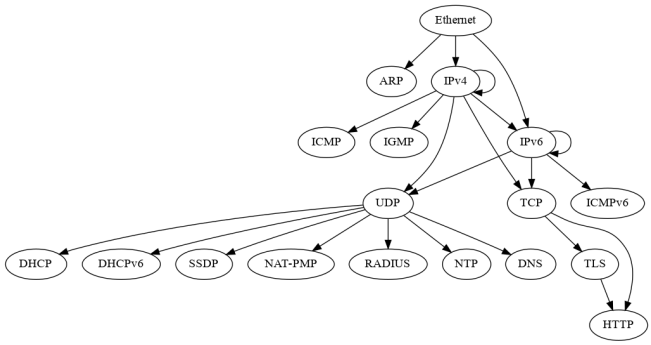

Figure 1. Packet Parsing Structure

The `peel-ip` library implements the protocol parsing using a tree. The arrows indicate encapsulation and the parser will try and traverse the network from the Ethernet node as far as possible. After my additions, Figure 1 shows the current structure.

3.3. Testing

Throughout the project I needed to test the protocol parsing library to ensure it was working as expected. I downloaded a selection of PCAP files from the Wireshark wiki[5] and loaded them using the client program. Some packets failed to parse so I analysed them in Wireshark[6] and adapted my parsing logic to accommodate any changes.

I would have liked to have tested it in an environment similar to the original use case, a network, but I do not have access to this so I am using my local machine and PCAP files to demonstrate.

Below are some examples of the output of my program with a brief description on what we can determine from it.

5. https://wiki.wireshark.org/SampleCaptures
6. https://wireshark.org

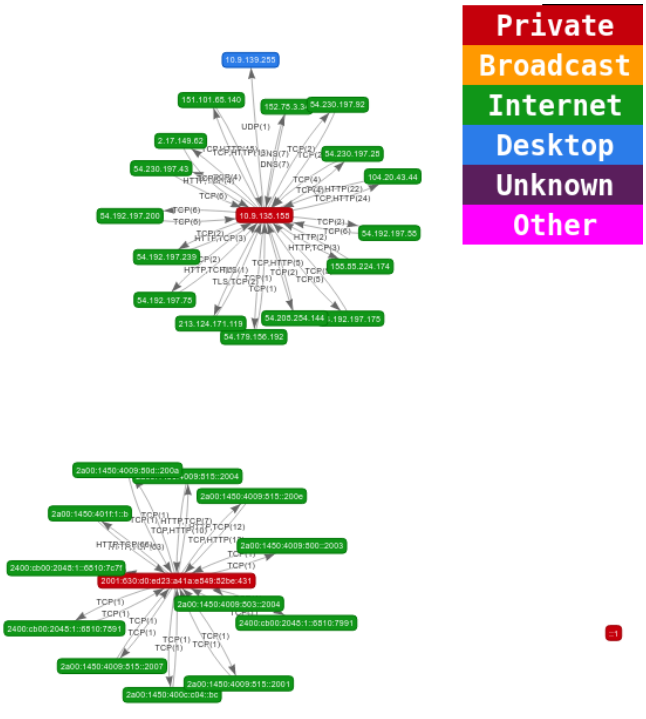

Figure 2. Browsing a few websites

This is a snapshot of the traffic going through my local machine after browsing a few websites. There are two main clusters, showing the IPv4 and IPv6 networks. The legend is also visible, which should be referred to for the following examples.

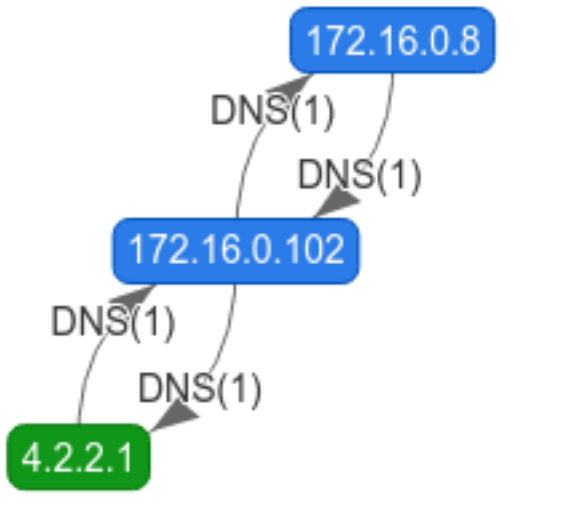

Figure 3. A recursive DNS query

Here we can see that `172.16.0.102` and `172.16.0.8` have been detected as internal IP addresses and `4.2.2.1` as a machine outside the network. All of the traffic between the machines are detected as DNS packets. We can determine that `172.16.0.8` does a DNS lookup by querying the network DNS server, `172.16.0.102`. `172.16.0.102` does not have the response in a local cache so the request is forwarded to an upstream DNS server, `4.2.2.1`, which is operated by Level 3 Communications. `4.2.2.1` responds with the IP to `172.16.0.102` who then forwards it back to `172.16.0.8`. This is a great way to visualise how DNS works.

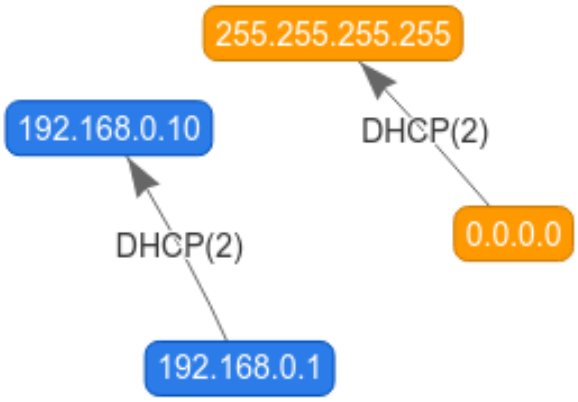

Figure 4. DHCP request and response

In this traffic capture there are 4 IP addresses involved. There are two broadcast DHCP packets from `0.0.0.0` to `255.255.255.255` and two responses from `192.168.0.1` to `192.168.0.10`. By knowing a little about how DHCP works we can determine the following; the `192.168.0.10` machine broadcasts for discovery on `0.0.0.0` to `255.255.255.255`. The DHCP server, `192.168.0.1`, picks this up and sends an offer to the client. `192.168.0.10` broadcasts a request and then the DHCP server responds with an acknowledgement. As mentioned in the *Future Work* section below, I think it would be useful to add more details into the labels such as the type of message that is sent.

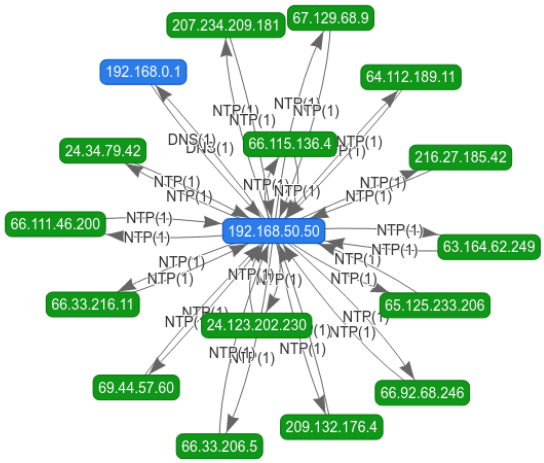

Figure 5. DNS request and NTP sync

In this image there is one machine communicating to a range of external machines over NTP and to an internal machine over DNS. `192.168.50.50` makes a DNS request to `192.168.0.1`, an internal DNS server. The DNS server sends a response back to `192.168.50.50`. `192.168.50.50` sends an NTP request to a range of external machines and receives a reply. From this, we can determine that the DNS lookup was querying the IP addresses of NTP servers and that the local machine was doing an NTP sync.

4. Individual Contribution

This project was an entirely individual effort with me writing all of the code for the client, server and front end.

As mentioned earlier, I also rewrote a lot of the protocol parsing library.

5. Future Work

I would most definitely like to continue working on this project. Here are my ideas for what needs to be done.

Firstly, I would like to further improve the protocol parsing library, potentially replacing the packet format with a standard format for ease of extendability.

Secondly, in order to get a prototype working as quickly as possible, the client –> server communication is done over HTTP. This is inefficient as a new connection must be made each time. I would like to switch to using the ZeroMQ[7] framework to reduce this overhead.

Furthermore, I would like to add more details into the visualisation. For example, labelling whether a DHCP packet is a discovery, offer, request or acknowledge message would help determine the flow of traffic between hosts. It might be useful to incorporate a time slider to alter the time frame in which packets are shown.

Finally, I would like to add documentation to the project and make it open source.

7. http://zeromq.org/

6. Conclusion

I really enjoyed working on this project and it was something I have wanted to do for a long time. It is interesting to see the flow of packets visualised as machines connected via DHCP or queried DNS servers. I hope to continue working on this and improve the functionality.

7. Notes

The source code can be found at *https://github.com/neosilky/passive-packet*.

My work on the underlying packet parsing library can be found at *https://github.com/neosilky/peel-ip*.

8. References

Cheshire, Stuart, and Marc Krochmal. 2013. "NAT Port Mapping Protocol (Nat-Pmp)." *URL Https://Tools.ietf.org/Html/Rfc6886*.

Conta, Alex, and Mukesh Gupta. 2006. "Internet Control Message Protocol (Icmpv6) for the Internet Protocol Version 6 (Ipv6) Specification." *URL Https://Tools.ietf.org/Html/Rfc4443*.

Droms, R. n.d. "Dynamic Host Configuration Protocol." *URL Https://Www.ietf.org/Rfc/Rfc2131.txt*.

Droms, Ralph, Jim Bound, Bernie Volz, Ted Lemon, Charles Perkins, and Mike Carney. 2003. "Dynamic Host Configuration Protocol for Ipv6 (Dhcpv6)." *URL Https://Www.ietf.org/Rfc/Rfc3315.txt*.

Fenner, William C. 1997. "Internet Group Management Protocol, Version 3." *URL Https://Tools.ietf.org/Html/Rfc3376*.

Goland, Y. 1999. "Simple Service Discovery Protocol." *URL Https://Tools.ietf.org/Html/Draft-Cai-Ssdp-V1-03*.

Mills, Dave L. 1985. "Network Time Protocol Version 4: Protocol and Algorithms Specification." *URL Https://Www.ietf.org/Rfc/Rfc5905.txt*.

Mockapetris, Paul. 2004. "RFC 1035—Domain Names—implementation and Specification, November 1987." *URL Http://Www.ietf.org/Rfc/Rfc1035.txt*.

Nmap.org. 2017. "Can Port Scanning Crash the Target Computer/Networks?" *Https://Nmap.org/Book/Legal-Issues.html*.

Postel, J. n.d. "Internet Control Message Protocol." *URL Https://Tools.ietf.org/Html/Rfc792*.

Willens, Steve, Allan C Rubens, Carl Rigney, and William Allen Simpson. 2000. "Remote Authentication Dial in User Service (Radius)." *URL Https://Tools.ietf.org/Html/Rfc2138*.