Group Report: Network Security Visualisation

Clockwork Dragon Group - Group 4

May 19, 2013

Introduction

Our application consists of a series of six visualisations of network traffic, each of which offers a different representation of the data. These are designed to enable a user to detect a variety of potential cyber-attacks. We have also constructed filters which allow the user to limit the visualisations to show only a selection of the data according to their individual needs.

All of our visualisations are drawn in real-time, i.e. they are updated as new data is available. While the application now focuses on analysing pre-collected data, it is easily extendable to support visualising network traffic as it happens due to the modular nature of the Object Oriented programming style we adopted.

Implementation

Here are brief descriptions of our visualisations:

1. **Heat Map** A hexagonal grid displaying clients active in the network. Colour indicates how much data is being transferred. Nodes "heat up" when amount of data goes over a specified threshold and "heat down" in time.
2. **Grouped Map** Similar to the Heat Map, but each server is represented by a blue hexagon and surrounded by all its clients.
3. **Attribute Distribution** This is a graded log graph showing how much traffic was recorded on a range of values of any desired packet attribute.
4. **Cube** Visualisation based on the Spinning Cube of Potential Doom. Every dimension can be customised depending on the available 'normalisers' of the data. Packets also have a little randomness just to make it more evident when a line of packets is dense or not.

5. **Data Flow** This visualisation draws lines between source nodes and destination nodes to show which connections are used most. Saturation indicates time (gray lines are past lines); coloured lines are more recent. It is possible to follow a packet by its colour. Red bars show traffic volume. They shrink at each iteration by some percentage, but grow linearly with new packets. They also have an upper limit.

6. **Traffic Volume** A stacked bar chart which displays the volume of data arriving in each interval, with segments proportional to the totals for each protocol. This allows users to see at a glance if a particular protocol is being exploited.

If the user wants to analyse only a selection of the data, it is possible to filter the input data by over 400 protocols (such as TCP, HTTP, etc), by port, and by IP address and MAC address using whitelists and blacklists. The main advantage of the filter system is that it works application-wide and applies to all visualisations.

We used normalisers for different attributes of a packet (e.g. source ip, destination port). These took data from a packet and returned a number between 0 and 1. This functionality was used in many of the visualisations, such as Data Flow and the Spinning Cube.

In part due to this, the application is very extensible. A new visualisation could be written to use any of the common utilities, such as the normalisers or colour palette, and then only one line of code need be changed to include this in the application. Similarly, if a normaliser for a new attribute were written it would take one line of code to include it. This normaliser would then be automatically included in the visualisations it would be useful in. It would even be possible to add a new Data Feeder that took packets from another source, again with minimal difficulty.

The application in its current state uses packet captures in CSV format[1]. The data includes IP address, MAC address, and port for both source and destination, information on the protocol used, and the size of the packet data. The packets listed in the CSV file are then 'simulated' to arrive in real-time. The user can choose to slow down or speed up the simulation, to pause it, or to skip to the end of the data capture.

An Analysis Panel is included to provide real-time aggregate data on the processed data packets. It makes use of a tabbed pane and a context panel for details on demand, to make the best use of the available space. The Analysis Panel makes use of a job queue and runs calculations in a separate thread, detached from the calculations for the visualisations, to keep the GUI responsive when a lot of data is being processed.

[1] see appendix for a specification of the file input format

Problems We Encountered

During Hilary term, we met several times as a group to discuss the assignment, and possible approaches to implementation. We produced an initial specification outlining what the final product was supposed to achieve. We then converted this design brief into a list of common interfaces we needed to design, and of required features to implement. We then each selected tasks to work on, proceeding to split them into smaller tasks as it became necessary. Later, we migrated to GitHub, and used the service's issue management.

During the vacation, we kept in touch and discussed the progress of the project using a combination of GitHub (to track bugs, comment on each other's code, and as version control) and Google Hangout (to plan further development). It was during the vacation that most of our work was done. Particularly helpful was the prompt development of the back-end of the application which is responsible for processing the input data and making it available to the visualisations, but also to display background data to the user.

After this, we focused on producing a number of visualisations to illustrate the data. During the entire development phase, the use of GitHub facilitated collaborative and concurrent development to a great extent.

Before the development phase, we were unsure about how to allocate work to people in the group. We didn't see a very natural splitting. While we did assign responsibilities for data collection, data processing, and visualisations to different people, we kept the working area of each person flexible so as to allow their input on all parts of the application. This proved to be a valuable decision. In particular, since there was a kind of linear structure to the tasks (data has to be collected before it can be processed, and it has to be processed before it can be visuablised), this informal approach to splitting tasks meant that development proceeded faster than it might have done otherwise.

Reflections

Reflecting back on the things we learned from this project, there is a list of technical tools each of us needed to become familar with: Git and GitHub, JOGL, and an understanding of the workings of computer networks. We encountered both the difficulties and the opportunities that collaborative software development brings with it, and understood the importance of a clear modularisation of code, provision of easy-to-understand interfaces, and good documentation, so that other people can quickly familiarise themselves with and work on other's code.

In retrospect, we had a somewhat slow start to our project. This was because many of us were unfamiliar with JOGL and networks in general which meant that it took a while of familiarisation with the basic concepts to be able to think up ideas for the program and be creative in our visualisations. During this time, we were helped by the variety of resources that our sponsor made available to us.

Looking at the application in the state it is now in, we can see a number of ways it could be extended. Live processing of data would obviously increase real-world usability. There are also ways in which the filter system could be made more powerful. Finally, by using more statistical (pre-)processing of the input data it could be possible to render more insightful visualisations.

However, we feel we have met all the design goals set out in our initial specification document, and the application is implemented to a high standard.

Appendix: Packet Capture Specification

Collected by James Nicholls

This document provides a specification and index of the CSV files which are in use as test data for the network visualisation application NetVis.

File format specification

All capture files are provided to the application as CSV files with the following headers;

• `No.`	Packet number
• `Time`	Time elapsed since first packet (seconds)
• `Source IP`	Source IPv4/6 address
• `Source HW`	Source hardware (MAC) address
• `Source Port`	Source port
• `Dest IP`	Destination IPv4/6 address
• `Dest HW`	Destination hardware (MAC) address
• `Dest Port`	Destination port
• `Protocol`	Communication protocol
• `Length`	Packet length (bytes)
• `Info`	Detected description of packet purpose

Sources

Notable external sources of packet trace (pcap) files.

- `https://www.evilfingers.com/`
  A community portal for Information Security, who publish internet security papers and keep a public archive of PCAP samples, among other resources.
- `http://www.honeynet.org/`
  The Honeynet Project is a leading international 501c3 non-profit security research organization, dedicated to investigating the latest attacks and developing open source security tools to improve Internet security.

Capture files

This section comprises a list of CSV files currently in use in application development and testing, as well as a short description of each.

eduroam.csv

Source: J. Nicholls
Original filename: `eduroam.pcap`
Size: 85664 packets - 16.4 MB

All traffic seen by an Ubuntu laptop with minimal running services, connected to the Eduroam network on the wlan0 interface. Approximately 85000 packets over 35 minutes.

jre-overflow.csv

Source: `https://www.evilfingers.com/`
Original filename:
`Sun_jre1.6.0_X_isInstalled.dnsResolve_Function_Overflow_PoC.pcap`
Size: 65561 packets - 14.7 MB

Proof-of-concept packet capture of a denial of service attack on JRE 1.6.0 by exploiting the DNS resolution function. A local server is flooded with 65000 packets in 11 minutes.

port-scan.csv

Source: J. Happa
Original filename: `portscan.pcap`
Size: 1818 packets - 371 kB

A port scan of a Windows Vista PC, originating from an Ubuntu PC, concluding that only port 80 (http) is open.

remote-execution.csv

Source: `http://www.honeynet.org/`
Original filename: `attack-trace.pcap_.gz`
Size: 348 packets - 53.6 kB

Packet trace of a malware attack which distributes a payload exploiting the Windows Local Security Authority (LSA) Remote Procedure Call (RPC) service of the victim host, compromising the IPC$ share. Once the share is exploited, a script is invoked, causing a connection to an FTP server named NzmxFtpd and the acquisition of an infected executable, `ssms.exe`.

skype.csv

Source: J. Nicholls
Original filename: `skype.pcap`
Size: 418 packets - 73 kB

Packets transferred during the authentication and initialisation of a Skype session. Recorded on an Ubuntu PC with minimal services running.

ssh-attack.csv

Source: `http://www.honeynet.org/`
Original filename: `hp_challenge.pcap`
Size: 5447 packets - 951.2 kB

Packet trace of an intruder gaining access to a server using a brute-force attack via SSH, before planting malware to download and execute software on the compromised host.

telnet-freebsd-exploit.csv

Source: `http://www.honeynet.org/`
Original filename: `fc.pcap`
Size: 238 packets - 35.2 kB

Demonstration of a buffer overflow exploit (CVE-2011-4862) that allows arbitrary code execution on a vulnerable FreeBSD server via telnet.

ubuntu-update.csv

Source: J. Happa
Original filename: `ubuntu-update.pcap`
Size: 497 packets - 84.6 kB

Packet trace of an Ubuntu PC communicating with a Canonical server to check for software updates. No new updates were found or downloaded.

nitroba.csv

Source: `http://digitalcorpora.org/corpora/scenarios/`
`nitroba-university-harassment-scenario`
Original filename: `nitroba.pcap`
Size: 95175 packets - 17.2 MB

Digital forensics excercise scenario involving a large packet capture from a shared wireless router, in a case of teacher harassment. See the above URL for details of the exercise.