Detecting Network Attacks with NoSQL Technology

Cohort 2 | Group 4

12/12/2015

Gaurav Diwanji, Nitika,
Lindsay Hefton, Sumit Garg

Table of Contents

DETECTING NETWORK ATTACKS WITH NOSQL

Abstract

With so many businesses now relying on web services in some form, and with so many avenues for access to a company's information through web-related services, one of the most crucial forms of security for a business is network security. Securing a network encompasses a very broad range of measures, one of the most fundamental of which is ensuring that there are mechanisms in place to recognize threats. While many solutions exist for intrusion and attack detection, the best, like Fireeye IDS, are sometimes far too expensive for smaller businesses. Furthermore, they are often only built to cover on one node on a network, and for larger networks this is insufficient as it becomes expensive to install a new instance every node and compile analyses. Finally, with the ever-increasing amount of traffic through any network open to customers, analyzing traffic efficiently becomes difficult. We have proposed the use of free and open source NoSQL technology solve these problems. This project leverages Apache Hadoop technology to process voluminous packet captures in parallel, ensuring the optimum use of resources and tackling the problem of scalability. The program detects of the most commonly occurring attacks in a network, Distributed Denial of Service (DDoS) attacks and Address Resolution Protocol (ARP) Spoofing, in addition to host scans, and presents a simple tool for detection based on a Java Swing user interface which analyses the packet captures.

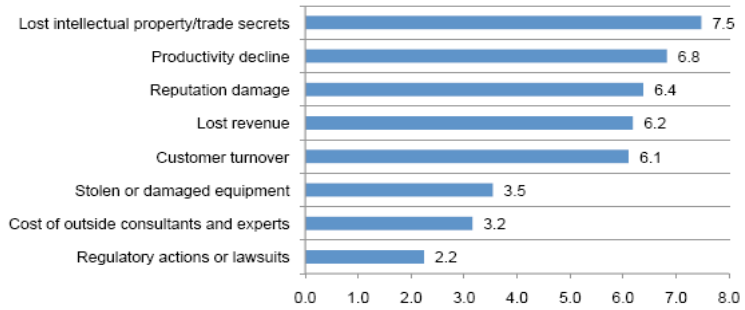
Introduction

In the past ten years or so, no danger is more evident than that of cyber intrustion into networks containing sensitive information. The list of compromised business grows ever on, and the largest profiles have been splashed across front pages continually for around about 5 years. In fact, alist of 4681 breaches amounting to 895,515,488 records stolen since 2005 at the time of this writing is contstantly being added to[i]. This is partly attributed to the massive increase in network traffic in recent years, which is projected to keep growing[ii]. As such, it is high time that emerging technologies such as Hadoop be applied towards network analysis for security purposes.

Implications of Intrusion

As per Norton's Annual Cybercrime Report 2013, "The global cost of cybercrime is USD 113 Billion annually, cost per cybercrime victim up by 50%"[iii]. The graph below from a study by the Ponemon Institute[iv] shows how many businesses ranked the negative consquences of cyber intrusion. Clearly, the loss of intellectual property is of the greatest concern. Trailing not far behind is the loss of reputation a company experiences when it is revealed that they were successfully attacked. This can have far-reaching implications for short-term and possibly long-term revenues. Target reported its revenues to have falled

46% just months after its breach in 2013[v]. Furthermore, cost from lawsuits for stolen sensitive customer information can be just as costly.



Why NoSQL?

There is already some research on the subject, and this project will not come close to the scale of some of those projects. However, the concept remains relevant. We are taking large amounts of traffic and are able to analyze it quickly for attacks using a map-reduce framework, which is invaluable. Furthermore, it is a much cheaper approach than some IDS systems. Target's top-of-the-line IDS is $420,000/year for the pro version or $9,600 for the entry-level version[vi]. Hadoop is free. Other big data platforms and applications that run with Hadoop like Cloudera Impala and Couchbase are not always free. However, they are still more cost-effective than some commercial IDS's and with enough research may become just as effective.

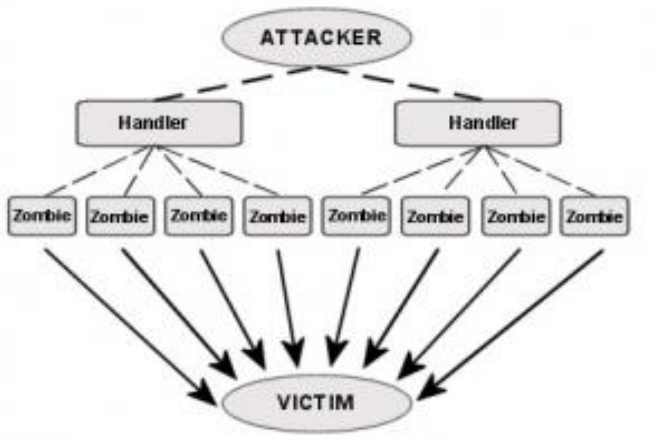
Types of Attacks

Host Scan

Host scanning is usually done in the initial phase of a penetration test. A host scan itself is not an attack; however, it is used by attackers to find hosts on a network, and for some attacks to discover which ports are open to help determine vulnerabilities.

DDoS - Distributed Denial-of-Service

vii



A Distributed Denial of Service (DDoS) attack is an attempt to make an online service unavailable by overwhelming it with traffic from multiple sources. The attacks are targeted at a wide variety of important resources, from banks to news websites, and present a major challenge for people to publish and access important information.

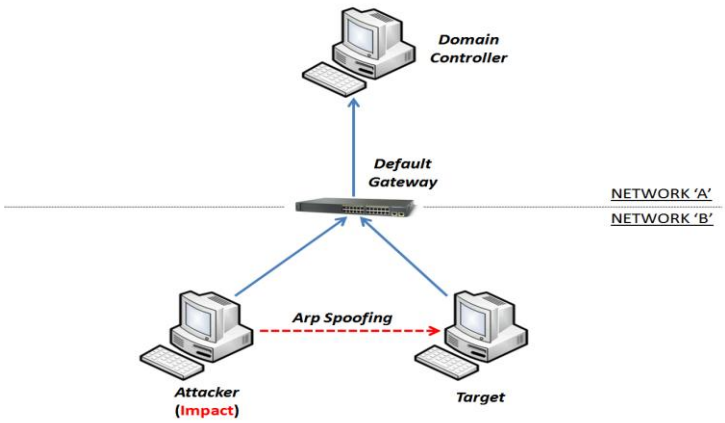
How the attack takes place

Attackers build networks of infected computers, known as 'botnets', by spreading malicious software through emails, websites and social media. Once infected, these machines can be controlled remotely, without their owners' knowledge, and used like an army to launch an attack against any target. Botnets can generate huge floods of traffic at once to overwhelm a target.

Symptoms

A network under attack will experience any of the following: unusually slow network performance (opening files or accessing web sites), unavailability of a particular web site, disconnection of a wireless or wired internet connection, or complete denial of access to the web or any internet services.

ARP spoofing



Address resolution protocol (ARP) spoofing is a malicious technique that causes the redirection of network traffic to a hacker. It is a type of attack in which a malicious actor sends falsified

ARP (Address Resolution Protocol) messages over a local area network. As a result, in a 2-way spoof, the router sends packets destined for the victim to the attacker first, and the victim send packets to the router through the attacker. This enables the attacker to intercept, modify or even stop data in-transit, enabling theft of sometimes highly sensitive information.

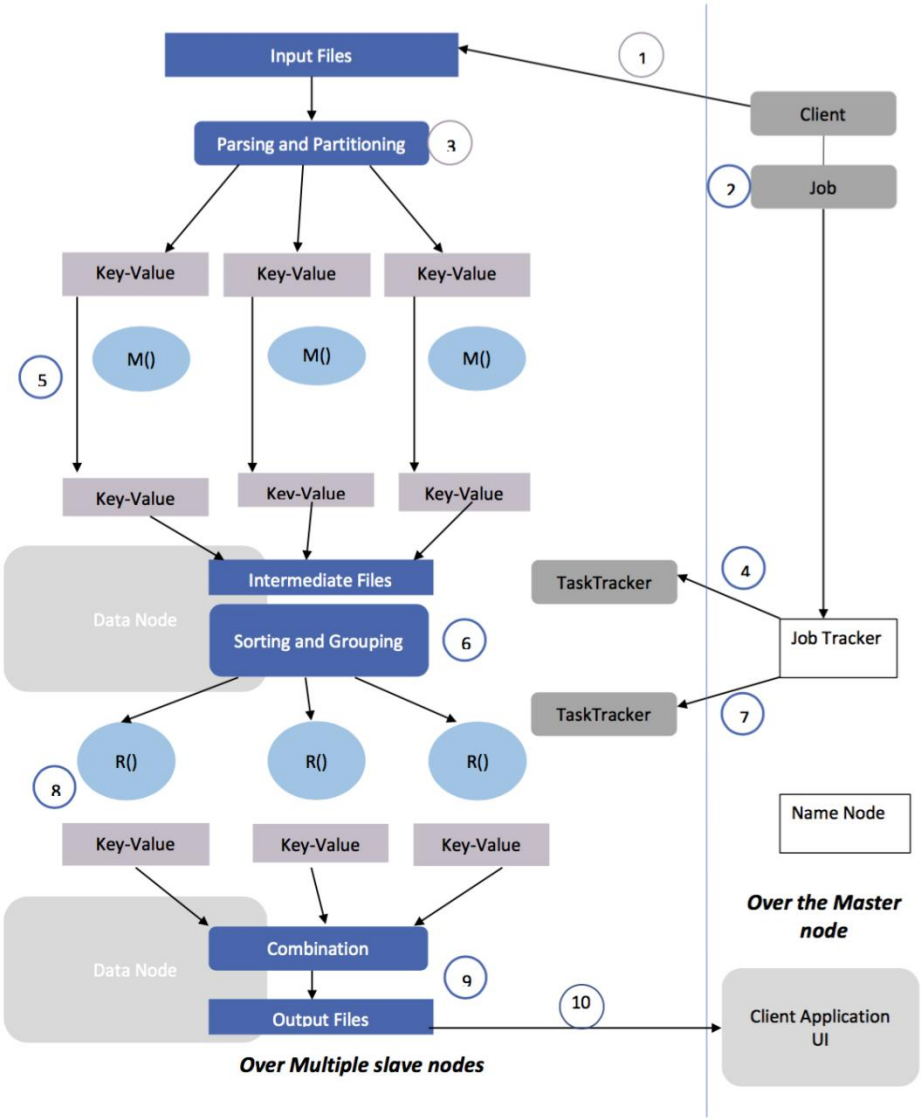
Symptoms

Sometimes, unfortunately, the attack will only be detected after people report information is stolen. However, there are ways to notice it as it happens. The user will notice sites that normally run over HTTPS running on HTTP, especially if using an insecure browser (browsers now support HSTS headers, in which a website tell the browser to connect over HTTPS[viii]. This does not apply in "incognito" modes).

Architecture Diagram

Each stage labeled in the figure is as below:

1. Client stores files to analyze.
2. Client post a job to process files.
3. Data is split into a collection of key-value pairs
4. Pairs are distributed to tasks across the cluster.
5. Attack specific mapper functions will synthesize each key-value pair into a new object.
6. New objects key-value are regrouped, sorted, and saved into intermediate files.
7. Intermediate files will be split to pairs and redistributed to tasks across the cluster.
8. Attack specific reducer functions will synthesize each key-value pair into a final conclusion.
9. Results will be regrouped and stored in a final output file.
10. The output file is moved to the client system again and interpreted on the UI screen.



The basic architecture for the execution of a typical map-reduce program is shown in the above diagram. Each label indicates the activity taking place at that particular point in the execution plan.

Firstly, the input csv file will be broken into parts and distributed over the cluster nodes for processing. Here in the cluster, each record is broken down into a key value pair as defined in the mapper function for the attack being analyzed. They will result in an intermediate file. Once all map tasks are completed (map phase is over), they are sent for combining, in our case the final reducing. The reduction logic is defined separately for each attack type. Once all the key-value pairs are reduced, the framework

will again combine it into a final output file. This output file is sent over to the client side for display in the user interface.

Note: In the project implementation, the UI being a proof of concept, we have configured it to only work on a single node structure. It can easily be extended into a clustered environment if needed.

Preparation of Input Data files.

Discussion of options for capturing traffic

In order for us to analyze network data for attacks, we needed some network data. There are a couple ways to get these files (disregarding how we carried out the attacks). The simplest path and the one we took was to capture traffic for a period of time using Wireshark, then export the capture file as a .csv (files are normally .pcap's). For the scope of this project, exploring different options could have quickly gotten out of hand both in terms of how we captured traffic and how we analyzed it, not only because of the myriad of options but the learning curve associated with each. Wireshark's GUI is easy to use and understand. Another method for the capturing would be tcpdump, a Linux command line sniffer and packet analyzer. However, there would be a learning curve with a new tool on an operating system not used by all group members on a regular basis. Second, being unfamiliar with tcpdump's output format, we were not certain the pcap files would exportable to an easily parsable .csv file, so we may have had to use tools to parse pcaps instead of writing Java code for parsing lines in a .csv. There are (command line) tools in tcpdump for filtering hex dumps and Python packages like dpkt as well. These would of course entail using bash scripts and Python scripts respectively instead of writing Java code, for which there would also be a learning curve. Clearly, Wireshark was the wisest option.

While discussing the actual use of NoSQL in a business environment, tcpdump might actually be a better approach because capturing and saving enormous files in Wireshark would be prohibitively resource-expensive, and exporting them just as troublesome. You also can't save incrementally with Wireshark as you can with tcpdump. This method might make an interesting long-term research project.

Making the files

We generated two pcap files—one for the DDoS attack and another for ARP spoofing and host scanning.

DDoS

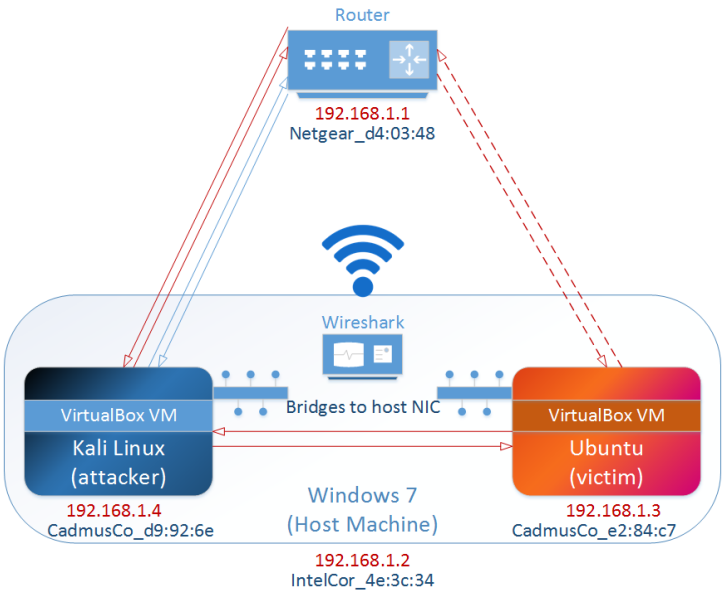
This attack was the simplest to implement. We did not need to monitor any traffic except that coming straight to the host, so we did not have to worry about switched network restrictions. We used 6 VMs on a hypervisor running on a server in a lab; 5 were attackers and the 6th was the victim, running Wireshark to monitor traffic in and out of that host. On the 5 attackers we ran the following command:

```
# hping3 -S --flood [IP of victim]
```

Hping is a versatile tool for packet generation. The -S sets the SYN flag to initiate a connection; the --flood means don't wait for responses; just keep dumping requests onto the network.

ARP Spoofing

This was much harder to get working, not only because the configuration that but because we ran into issues during the attacks, described below. For the setup, we could not use the hypervisor in the lab used for the DDoS attack because the server is on a switched network, and we needed it on Wi-Fi to monitor traffic to and from and between the VMs. And we didn't have GUI access to it anyway. We used 2 VirtualBox VMs on a host running Wireshark. The setup is shown below.



There were a few things that needed to be configured before running the spoof. For the actual spoof we used either Ettercap's GUI tool or # arpspoof. Ettercap configuration files had to be edited to allow for port redirection using iptables before actually running the iptables command. We also had

to edit iptables and another file, which is as simple as running the two commands below. Iptables tool is a tool for making firewall rules. We used it to redirect all traffic on port 80 (normal web traffic) to 8080, where we set up sslstrip to listen so that we could view unencrypted traffic.

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
# iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 -j REDIRECT --to-port 8080
# sslstrip -l 8080 -w /root/Desktop/sslstriplog
```

Troubles

There weren't many issues with the DDoS simulation. The only speedbump is that Citrix XenServer, the hypervisor we were using, doesn't play well with Kali Linux and we couldn't get a single Kali 2.0 VM to actually work. There were older versions of Kali available, but we just decided to install hping on several Ubuntu machines, which worked just as well.

One issue we ran into with ARP spoofing is that all the tutorials we viewed used an older version of iptables that used --destination-port instead of --dport; so for a long time we couldn't figure out why commands weren't working. Looking at the (very long) iptables manual fixed the issue. Another problem was the network simply shutting down on the victim VM when we did the arpspoof; the issue was either the way VirtualBox handles bridged adapters or an issue with the router or both, because it kept assigning one of the VMs the same MAC as our host machine once communication between them was established. Tests seemed wildly unpredictable; successful tests were run at least three times before making the video for the presentation, then things suddenly stopped working. We had to revert VirtualBox to the previous version. Luckily we had already made a successful capture, so only making the video was affected.

Application Deployment and Output Explanation

This application is designed as a stand-alone JAR (Java Archive) file, making it extremely easy to deploy and run. It is also environment-independent and can be run on Windows, Mac and Linux environments. The steps to run the application in a Macintosh environment are described below. Similar steps can be performed for setup in a Windows or Linux as well.

1. Firstly, make sure that the Hadoop is configured to run as a single node configuration. (The program is well capable of running in a pseudo cluster but due the UI is currently programmed only for single node operation for the sake of simplicity.)
    a. **Note:** For single node operation, the configuration files /etc/hadoop/hdfs-site.xml and /etc/hadoop/core-site.xml should have nothing between the <configuration> </configuration> tags
2. Place the Attacks.jar file in the /hadoop/ (Hadoop home directory).
3. In the terminal, run the Attacks.jar using the in-built Hadoop jar command from the local hadoop home directory, shown below. (We do not need any command line arguments for the program)
4. The program is now running and the user interface window will now pop up.

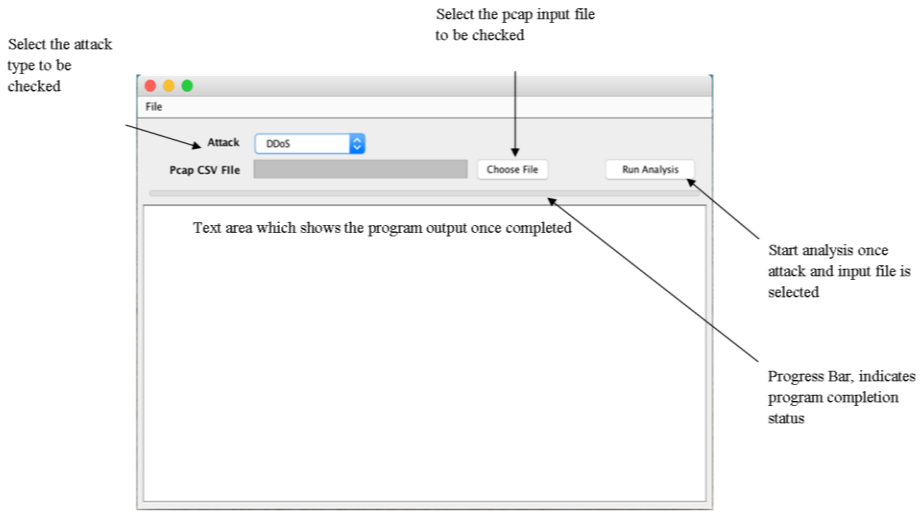
```
# bin/hadoop jar Attacks.jar
```

Using the application

The application is designed to be very intuitive and clutter free. The detailed screen shots and a description of the output are given in the screens below. The steps to run it are as follows:

1. Select Attack type (DDoS, ARP Spoof or Host Scan) from 'Attack' drop down list.
2. Select input pcap csv file from the local machine, using the file browser that pops up on clicking the 'Choose File' button.
    a. **Note**: It will only accept .csv type file extensions. Pcaps can be easily converted into .csv in Wireshark. In Linux, files may have to be renamed with the .csv extension.
3. Press the 'Run Analysis' button to start the map reduce attack detection program. The progress will be reflected in the progress bar.
4. Once completed, the program output will be displayed in the text area.
5. To start a new analysis, you can clear the application state using File → Clear Application

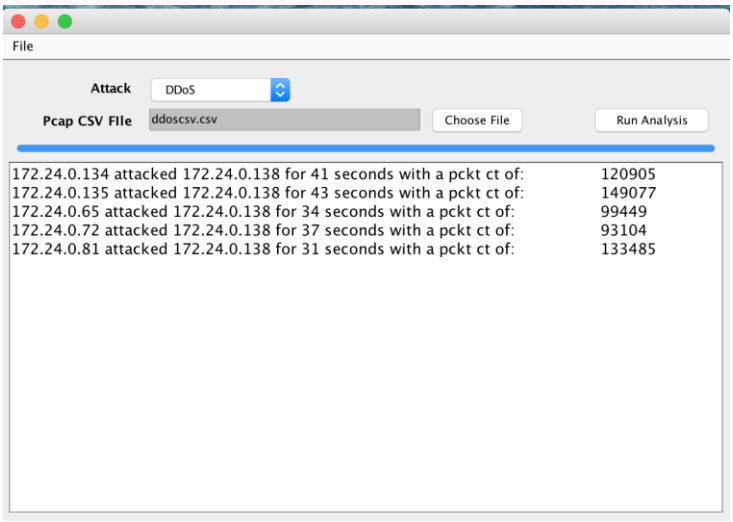
The User Interface Window



Sample output

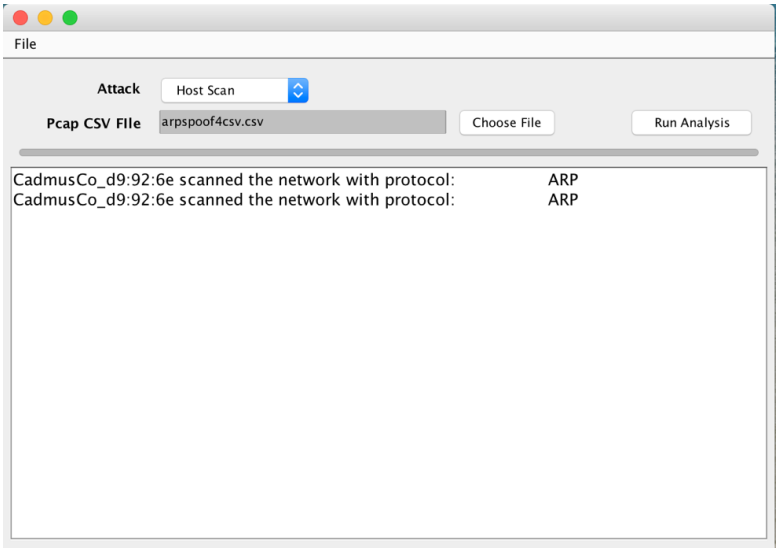
The program output is displayed in the Text Area on the user interface on successful completion of the map-reduce task. Explanations of output for each attack type are given here briefly and explained in more detail in the "Solutions and Pseudo-code" section of this document, below.

DDoS output



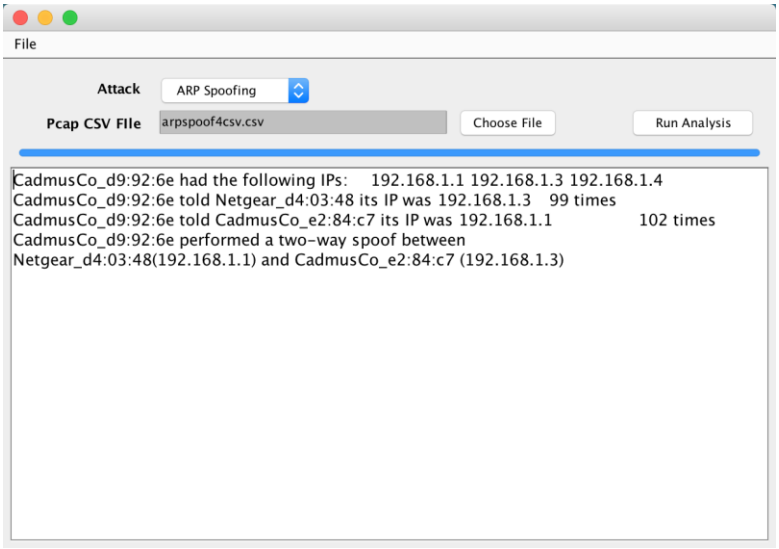
The output here indicates 5 DDoS attackers *(172.24.0.134, 172.0.135, 172.0.65, 172.24.0.72, 172.24.0.81)* who target the same IP *171.24.0.138*. The output also includes information on the duration of the attack from each attacker in seconds and the total number of packets it sent.

Host Scan output



The output here indicates the MAC address which scanned the network and the protocol it used to do so. Here, the MAC address *CadmusCo_d9:92:6e* scanned the network subnet twice using the ARP protocol.

ARP spoof output



We write four different key-value pairs to the reducer for this attack, as a result of three tests. The first test checks whether a MAC had more than 1 IP address in the capture window and lists them. The second test checks if a MAC sent too many ARP 'reply' messages. It writes up to two lines showing how many were sent and to whom, with the destinations being the victims. The third check checks for the gateway among the victims and outputs the fourth line if true; if the gateway is involved, it's likely a 2-way spoof between it and the other victim.

Solution and Pseudo Code

We had a class called "Pcap" on which all other classes depended. It parses each line in the csv to extract data about each packet.

```
def writeFromCSV(packet):
    headers = ('No.' 'Time' 'Src' 'Dst' 'Protocol' 'Length' 'Info' '\n')
    parser = new CSVParser(headers, packet) #concatenate headers and s
    try:
        Protocol = #info under header 'Protocol'
        Time = #info under header 'Time'
        etc...
    except error as e:
        print e
def getTime():
    return Time
def getProtocol():
    return Number
etc...
```

DDoS

Detecting DDoS is a simple count per second formula. The mapper logs every (src||dest||protocol) as a key and puts the 8-place second values and packet info as value. The reducer counts packets for each second using the integer value of each key. If there are more than 1000, we increment a duration count

and add the packet count to a total packet count. Since we sent SYN packets, the victim kept reponding with (ACK, RST) packets, so to exclude it from the attacker list we ignored packets with ACK flags.

DDoS mapper pseudo-code:

```
pcap = new Pcap(packet)
key = (pcap.getSource()+'||'+pcap.getDestination()+'||'+pcap.getProtocol())
time_info =(pcap.getTime()+'<>'+pcap.getInfo())
```

DDoS reducer pseudo-code:

```
keySplit[] = key.split('||')
src = keySplit[0]
dst = keySplit[1]

map = new HashMap()
for value in values:
    vSplit[] = value.split('<>')
    time = int(vSplit[0])
    info = vSplit[1]
    if 'ACK' in info:
        NOP
    elif time in map.keySet():
        total = map.get(time) + 1
        map.put (time, total)
    else:
        map.put(time, 1)

duration = 0
totalPkts = 0

for tkey in map:
    if map.get(tkey) > 1000:
        duration +=1
        totalPkts += map.get(tkey)

KVP = (src+' attacked '+dst+' for '+duration+' with', totalPkts)
```

Host scan

This one was the easiest to implement. The mapper checks for ARP or ICMP in the protocol. If ARP, we only looked at "who has" requests and wrote ([source destination 'ARP'], IP requested) to context. If ICMP, we wrote ([source 'request' 'ICMP'], IP requested) to context. The reducer counts the hosts requested for every key (this assumes the entire scan is done using one protocol, which is likely).

Host scan mapper pseudo-code:

```
pcap = new Pcap(packet)
src = pcap.getSource()
dst = pcap.getDestination()
protocol = pcap.getProtocol()
if 'ARP' in protocol:
```

```
    ipReq = pcap.getInfo().substring(ip_requested)
    KVP = (src+'<>'+dst+'<>'+'ARP', ipReq)
elif 'ICMP' in protocol:
    ipReq = dst
    icmpType = pcap.getInfo().substring(request_or_reply)

if 'request' in icmpType:
        KVP = (src+'<>'+'request'+'<>'+'ICMP', ipReq)
```

Host scan reducer pseudo-code:

```
keySplit[] = key.split('<>')
src = keySplit[0]
protocol = keySplit[2]

map = new HashMap()
for IP in values:
    if IP in map.keySet():
        total = map.get(IP) + 1
        map.put (IP, total)
    else:
        map.put(IP, 1)

hostsScanned = map.size()

if hostsScanned > 250:
    KVP = (src+' scanned the network with protocol ', protocol)
```

ARP

This was a rather complicated program to implement ultimately. One of the limitations in our progress report was that our method for detection was insufficient, which we fixed here. Instead of just detecting whether a MAC had multiple IPs, we added 2 checks for 3 total:

1. Did a MAC have more than 1 IP in the capture window?
2. Were an abnormal number of "[IP] is at [my MAC]" messages were sent?
3. Was one of the victims the gateway?

The second and third checks are used because an attacker will have to continually mess with the router's ARP table in a sort of constant arm wrestle with the router, which will periodically reset the table and resend correct information. We write to the reducer 4 times per attack. Output is explained above.

ARP mapper pseudo-code:

```
pcap = new Pcap(packet)
MAC = pcap.getSource()
dst = pcap.getDestination()
protocol = pcap.getProtocol()
info = pcap.getInfo()

if not 'ARP' in protocol:
```

```
    return;

if 'Who has' in info:
    IP = info.substring(IP after "who has")
    arpType = 'whohas'
elif 'is at' in info:
    arpType = 'isat'
    IP = info.substring(IP before "is at")

KVP = (MAC, dst+IP+arpType)
```

ARP reducer pseudo-code:

```
protocol = keySplit[2]
map = new HashMap()
ip_list = []
MAC = key

for value in values:
    vSplit[] = value.split('<>')
    dst = vSplit[0]
    IP = vSplit[1]
    type = vSplit[2]

    if not IP in ip_list:      #check 1: count a MAC's IPs
        ip_list.add(IP)

    if type == 'isat':          #check2: count 'is at' messages; take top 2 values
        if not value in map:
        map.put (value, 1)
    else:
        map.put(value, map.get(value)+1)

    max1 = 0
    max2 = 0
    greatestKey1 = ''
    greatestKey2 = ''

    for mapKey in map:
        if map.get(mapKey) > max1 or max2
            #the lesser of max(1 or 2) = map.get(mapKey)
            #the corresponding greatestKey(1 or 2) = mapKey

    maxSplitX[] = greatestKeyX.split()
    victimX = maxSplitX[0]       #destination
    tmpIPX = maxSplitX[1]

KVP1 = ([MAC+' had these IPs: '], ip_list)
KVP2 = ([MAC+' told '+victim1+' its IP was '+tmpIP1], max1+' times')
KVP3 = ([MAC+' told '+victim2+' its IP was '+tmpIP2], max2+' times')

if victimX == gatewayIP:                #check 3: is gateway a victim?
    KVP4 = ([MAC+' performed a two-way spoof between '], gateway+' and '+victimX)
```

Works Cited

Cisco. *The Zettabyte Era- Trends and Analysis.* White Paper, Cisco, May 2015.

Harris, Elizabeth A. "Faltering Target Parts Ways with Chief." *New York Times.* May 5, 2014. http://www.nytimes.com/2014/05/06/business/target-chief-executive-resigns.html?ref=technology&_r=1 (accessed December 6, 2015).

Ko, Daniel. "Samsung Prepares for Next Possible Cyberattack." *Korea Times.* September 14, 2009. http://www.koreaittimes.com/story/5025/samsung-prepares-next-possible-cyberattack (accessed December 6, 2015).

Krebs, Brian. "The Target Breach, By the Numbers." *Krebs on Security.* May 6, 2014. http://krebsonsecurity.com/2014/05/the-target-breach-by-the-numbers/ (accessed December 6, 2015).

Newman, David. "Review: FireEye fights off multi-stage malware." *Network World.* May 5, 2014. http://www.networkworld.com/article/2176480/network-security/review--fireeye-fights-off-multi-stage-malware.htm (accessed December 6, 2015).

OWASP. "HTTP Strict Transport Security." *OWASP.* September 7, 2015. https://www.owasp.org/index.php/HTTP_Strict_Transport_Security (accessed December 6, 2015).

Parks, Miles. "Target Offers 10 Million Settlement in Data Breach Lawsuit." *www.npr.org.* March 31, 2015. http://www.npr.org/sections/thetwo-way/2015/03/19/394039055/target-offers-10-million-settlement-in-data-breach-lawsuit (accessed December 6, 2015).

Ponemon Institute LLC; Radware. "Cyber Security on the Offense: A Study of IT Security Experts." Research Report, November 2012.

Privacy Rights Clearinghouse. *Chronology of Data Breaches.* 2015. http://www.privacyrights.org/data-breach (accessed December 6, 2015).

Symantec. *2013 Norton Report.* Research Report, Norton, 2013.

---

[i] Privacy Rights Clearinghouse. *Chronology of Data Breaches.* 2015. http://www.privacyrights.org/data-breach (accessed December 6, 2015).

[ii] Cisco. *The Zettabyte Era- Trends and Analysis.* White Paper, Cisco, May 2015.

[iii] Symantec. *2013 Norton Report.* Research Report, Norton, 2013.

[iv] Ponemon Institute LLC; Radware. "Cyber Security on the Offense: A Study of IT Security Experts." Research Report, November 2012: 3.

[v] Harris, Elizabeth A. "Faltering Target Parts Ways with Chief." *New York Times.* May 5, 2014. http://www.nytimes.com/2014/05/06/business/target-chief-executive-resigns.html?ref=technology&_r=1 (accessed December 6, 2015).

[vi] Ibid.

[vii] Ko, Daniel. "Samsung Prepares for Next Possible Cyberattack." *Korea Times.* September 14, 2009. http://www.koreaittimes.com/story/5025/samsung-prepares-next-possible-cyberattack (accessed December 6, 2015).

[viii] OWASP. "HTTP Strict Transport Security." *OWASP.* September 7, 2015. https://www.owasp.org/index.php/HTTP_Strict_Transport_Security (accessed December 6, 2015).