

# NEXT-GENERATION POLICY AS CODE

**ANTHONY MALYSZ**

`anthonymalysz@college.harvard.edu`

**A.J. STEIN**

`alexander.stein@nist.gov`

## ABSTRACT

*“NGPaC is an open-sourced tool with which to translate English network and information security policy into PCAP expressions now, and eBPF in the future.”*

— `usnistgov/ngpac/README.md`

IT governance and infosec policies are often complex to formulate, and even more complex to implement. Technical management and organization executives draft policy language with the intent to inform all employees—not necessarily engineers, let alone the computers themselves—who must actualize said policies. After policy language is drafted, expert technical staff are needed to implement the policy to their respective technical domains, such as network and security administration of Linux operating systems.

Oftentimes, rigorous information security programs require additional impartial, third-party experts to come and assess the implementation and review its readiness for production use separate of those that wrote the policy and implemented the systems to support it.

It would be ideal to have software read and interpret these plain-English policy documents instead, providing code in the appropriate domains to implement and assess them. In this project, we aim to explore this research space with the following approach:

1. Review the current state of software to convert policy documents into system network and security administration functions in a semi- or totally-automated fashion.
2. Build a prototype web application to handle policy documents in “formatted” English and convert them into PCAP expressions (and eBPF later): a novel policy-as-code approach to manage network and security functions for modern Linux operating systems.
3. Publish this repository as open-source project for the long-term goal of integrating existing open-source projects into NGPaC for added functionality, searching for specialized data with which to train and test these models with, and more.

# TABLE OF CONTENTS

Abstract .....	1
Table of Contents .....	2
Work Log .....	4
<b>Week I: “<i>Virtual Machines...</i>”</b>	
Orientation Day .....	5
Virtual Machines .....	5
Firewalls .....	7
Starting Documentation .....	9
Blog Catch-Up .....	9
<b>Week II: “<i>Network Protocols...</i>”</b>	
Ubuntu & iptables .....	10
Investigating Protocols .....	12
GitHub Bug Issue & Wireshark .....	13
Wireshark: File Excavation .....	15
<b>Week III: “<i>Intercepting Network Traffic...</i>”</b>	
Firewalls & Network Traffic .....	16
UTM & Suricata .....	17
UTM & Suricata, Continued .....	18
Q&A with VPNs .....	20
VPN Lab with AWS .....	23
<b>Week IV: “<i>Researching PCAP Expressions...</i>”</b>	
PCAP Expressions .....	24
PCAP Literature .....	25
PCAP Translators .....	25
Building PCAP Expressions .....	26
Dev’s Log #01 .....	26
<b>Week V: “<i>Starting Development...</i>”</b>	
Travel & Pen-testing Enrichment .....	27
Generating Test Expressions .....	27
Dev’s Log #02 .....	27
Dev’s Log #03 .....	27
<b>Week VI: “<i>Building a Prototype...</i>”</b>	
Dev’s Log #04 .....	28
Dev’s Log #05 .....	28
Dev’s Log #06 .....	28
Dev’s Log #07 .....	28
Dev’s Log #08 .....	28

<b>Week VII: “OCaml Web Frameworks...”</b>	
Web Frameworks: Incr_dom, by Jane Street .....	29
Web Frameworks: Dream .....	29
Dev’s Log #09 .....	31
Dev’s Log #10 .....	31
<b>Week VIII: “Web Exploits &amp; Testing...”</b>	
XSS & CSRF Attacks .....	32
Insufficient Testing .....	33
Dev’s Log #11 .....	33
Dev’s Log #12 .....	33
Dev’s Log #13 .....	33
<b>Week IX: “Finalizing Scope...”</b>	
Dev’s Log #14 .....	34
Dev’s Log #15 .....	34
Campus Visit .....	34
Dev’s Log #16 .....	34
Dev’s Log #17 .....	34
<b>Week X: “Preparing Presentation...”</b>	
Outlining Presentation .....	35
Drafting Presentation #01 .....	35
Drafting Presentation #02 .....	35
Finalizing Presentation .....	35
Practicing Presentation .....	35
<b>Week XI: “Preparing Presentation...”</b>	
Dev’s Log #18 .....	36
Dev’s Log #19 .....	36
Dev’s Log #20 .....	36
Final Presentation & Parting Thoughts .....	36

# WORK LOG

## WEEK [I.]

Mon 05/22 — Tue 05/23 — Wed 05/24 — Thrs 05/25 — Fri 05/26

## WEEK [II.]

Holiday — Tue 05/30 — Wed 05/31 — Thrs 06/01 — Fri 06/02

## WEEK [III.]

Mon 06/05 — Tue 06/06 — Wed 06/07 — Thrs 06/08 — Fri 06/09

## WEEK [VI.]

Mon 06/12 — Tue 06/13 — Wed 06/14 — Thrs 06/15 — Fri 06/16

## WEEK [V.]

Holiday — Tue 06/20 — Wed 06/21 — Thrs 06/22 — Fri 06/23

## WEEK [VI.]

Mon 06/26 — Tue 06/27 — Wed 06/28 — Thrs 06/29 — Fri 06/30

## WEEK [VII.]

Mon 07/03 — Holiday — Wed 07/05 — Thrs 07/06 — Fri 07/07

## WEEK [VIII.]

Mon 07/10 — Tue 07/11 — Wed 07/12 — Thrs 07/13 — Fri 07/14

## WEEK [IX.]

Mon 07/17 — Tue 07/18 — Wed 07/19 — Thrs 07/20 — Fri 07/21

## WEEK [X.]

Mon 07/24 — Tue 07/25 — Wed 07/26 — Thrs 07/27 — Fri 07/28

## WEEK [XI.]

Mon 07/31 — Tue 08/01 — Wed 08/02 — Thrs 08/03

**WEEK I.** (05/22 - 05/26)*Virtual Machines...***Monday, May 22nd****Orientation Day** 🖥️👋

- Attended welcome meeting via BlueJeans.
- Sat in on a meeting of AJ's with people reporting their week's work in the NIST ITL.

**Tuesday, May 23rd****Virtual Machines** 🍺👤

- Had trouble with iTAC getting into my work laptop; installed VirtualBox with the goal of installing Ubuntu as the VM inside it. Decided using homebrew would be easier.
- Installed multipass using `brew install multipass`.
- Confirmed it worked by running `multipass version` and `multipass find`:

```
anthonymalysz@Air ~ % multipass version
multipass  1.10.1+mac
multipassd 1.10.1+mac

#####
Multipass 1.11.1 bug fix release
Fix Windows path, Virtualbox mount failure, Windows native mounts

Go here for more information:
https://github.com/canonical/multipass/releases/tag/v1.11.1
#####
```

•  
•  
•  
*Continued...*  
•  
•  
•

Tuesday, May 23rd (cont. #1/2)



```
anthonymalysz@Air ~ % multipass find
```

Image	Aliases	Version	Description
18.04	bionic	20230525	Ubuntu 18.04 LTS
20.04	focal	20230523	Ubuntu 20.04 LTS
22.04	jammy,lts	20230518	Ubuntu 22.04 LTS
anbox-cloud-appliance		latest	Anbox Cloud Appliance
charm-dev		latest	A development and testing environment for charmers
docker		0.4	A Docker environment with Portainer and related tools
jellyfin		latest	Jellyfin is a Free Software Media System that puts you in control of managing and streaming your media.
minikube		latest	minikube is local Kubernetes
ros-noetic		0.1	A development and testing environment for ROS Noetic.
ros2-humble		0.1	A development and testing environment for ROS 2 Humble.

- Proceeded to follow [this blog](#) to figure out how to run Ubuntu 22.04 VMs on my Apple M1 ARM-based system for free, whose steps included:

1. `multipass launch 22.04 -n primary -c 2 -m 4G -d 16G`
2. `multipass shell`
3. `sudo passwd ubuntu`

To access the newly-installed desktop, I discovered the IP address of the VM by running `ip a` within the Ubuntu shell. I then used the IP address, and to connect I downloaded Microsoft Remote Desktop from the AppStore. Not sure if it worked.

(Continued ...)

Tuesday, May 23rd (cont. #2/2)



- Spent a bit more time figuring out how to run a web server on a multipass VM (on macOS with an M1 processor), mostly using StackOverflow and prompting ChatGPT (I think I was successful, as I was able to visit the IP address at the end in my browser):
  1. **Installation:** Install Multipass by visiting the Multipass website (<https://multipass.run/>) and downloading the appropriate version for macOS M1. Follow the provided instructions to complete the installation.
  2. **Launching a Multipass VM:** Launch a Multipass VM by opening a terminal window and running `multipass launch --name my-vm`. This will create a new virtual machine named `my-vm`.
  3. **Accessing the VM:** To access the Multipass VM, run `multipass shell my-vm`. This will open a shell session inside the VM.
  4. **Installing and Configuring the Web Server:** Install Apache inside the VM by running `sudo apt update` and `sudo apt install apache2`. Once the installation is complete, Apache should be running automatically. You can test it by opening a web browser on your Mac and navigating to `http://<VM-IP-address>`. Replace `<VM-IP-address>` with the IP address of your Multipass VM.
  5. **Sharing Files with the VM:** By default, the Multipass VM doesn't have direct access to your Mac's file system. You can use the Multipass file transfer feature to share files between your Mac and the VM. To transfer files from your Mac to the VM, use the following command:  
`multipass transfer /path/to/local/file my-vm:/path/to/destination/file`  
(for me, the `/path/to/destination/file` was `/Users/anthonymalysz`). To transfer files from the VM to your Mac, use the following command:  
`multipass transfer my-vm:/path/to/vm/file /path/to/destination/file`.  
Replace the paths with the appropriate locations on your Mac and the VM.



Wednesday, May 24th

Firewalls

- Figured out how to configure two HTTP servers, one on port 8080 and one on port 9090 by opening two shells and running `python3 -m http.server 8080` in one shell and `python3 -m http.server 9090` in the other; then, I installed **iptables** and configured the iptables such that access to the 9090 HTTP server was allowed, but access to the 8080 server was prohibited. These were the steps I took (ChatGPT, Stack Overflow):

Wednesday, May 24th (cont. #1/2)



- Open up two terminal windows and run `python3 -m http.server 8080` and `python3 -m http.server 9090`, so that they're both up.
- Create a new Apache configuration file for the first server on port 8080: `sudo nano /etc/apache2/other/first_server.conf`. This command will create a new configuration file specifically for the first server.
- Inside this file, I added the following configuration (then I saved the changes by pressing Ctrl + X, then Y, then Enter) and exited the editor:

```
anthonymalysz@Air ~ % sudo nano /etc/apache2/other/first_server.conf

Listen 8080
<VirtualHost *:8080>
    DocumentRoot "/Users/anthonymalysz"
</VirtualHost>
```

- Similarly, I created a new Apache configuration file for the second server on port 9090 by running: `sudo nano /etc/apache2/other/second_server.conf`. This command created a new configuration file specifically for the second server.
- Inside the file, I added the following configuration (then I saved the changes by pressing Ctrl + X, then Y, then Enter) and exited the editor:

```
anthonymalysz@Air ~ % sudo nano /etc/apache2/other/second_server.conf

Listen 9090
<VirtualHost *:9090>
    DocumentRoot "/Users/anthonymalysz"
</VirtualHost>
```

- At this point, I've created separate configuration files for each server on ports 8080 and 9090, which are stored in `/etc/apache2/other/first_server.conf` and `/etc/apache2/other/second_server.conf`, respectively.
- Next, I installed `pf` (Packet Filter) as a firewall. I edited the `pf` configuration file and added the following two lines to the bottom:



Wednesday, May 24th (cont. #2/2)



```
anthonymalysz@Air ~ % sudo nano /etc/pf.conf

rdr pass inet proto tcp from any to any port 9090 -> 127.0.0.1 port 9090
block in inet proto tcp from any to any port 8080
```

- After saving changes and exiting, I enabled `pf` by running `sudo pfctl -e`.
- After seeing that I could still access both by visiting `http://localhost:8080` and `http://localhost:9090`, it seems that I was supposed to do all of this inside the Ubuntu VM to begin with...oops! In any case, this was more of a learning experience than anything to begin with, to see how these things are still done more “manually” than they “should” be.
- During our sync meeting for the day, we decided to avoid things like this in the future—and to have better structure and records overall going forward—that we should create a  $\text{\LaTeX}$  document to log all of our work.



Thursday, May 25th

Starting Documentation

- Spent the whole day creating this document from scratch, getting it to a point where all weeks' text boxes are colored, styled, and ready to be written in; also, see new sections **Work Log** and **Table of Contents** (works in progress) with clickable hyperlinks to other pages (you can only see these by hovering over them, but I'll try to make it obvious when links are link-y). Will share with AJ soon, so that this can become a collaborative document to mostly log progress, but also to communicate ideas/concerns/goals.



Friday, May 26th

Blog Catch-Up

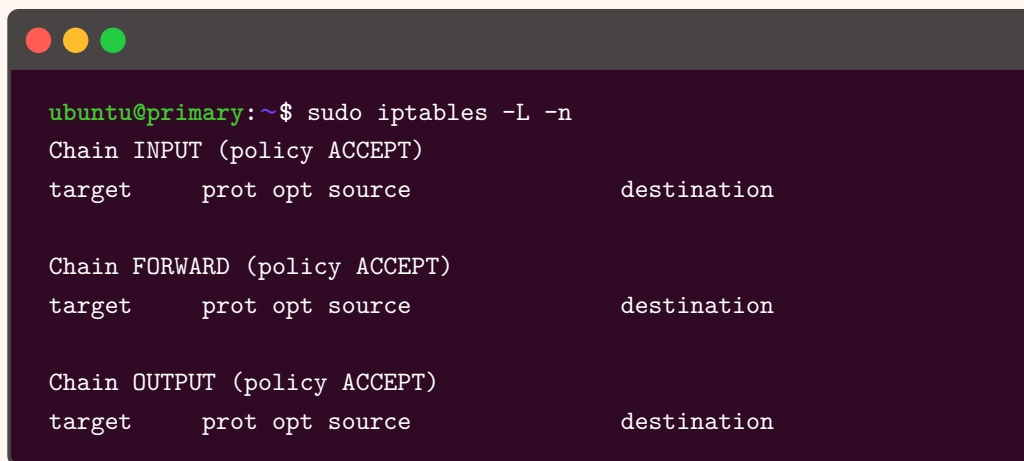
- Gained access to my NIST Laptop and downloaded necessary apps.
- Filled in Monday's, Tuesday's, and Wednesday's activities in this document (using Element DMs, ChatGPT prompt history, and Internset search history), and made new commands that will help with typesetting code and realistic-looking terminal window(s).
- Explored our **GitHub Project Board**, linked Overleaf with public GitHub repo.

**WEEK II.** (05/30 - 06/02)*Network Protocols...*

Tuesday, May 30th

Ubuntu & iptables  

- We refreshed what we did last week, and tried to see if I could run a Linux VM on my work laptop, now that it's accessible. Trying to run Powershell on Windows 10 as administrator (administrative command prompt) didn't work, as I needed a user/login from NIST. Trying to run the command `usl --install` in a normal Powershell prompt didn't work either, with the same result. We'll try to resolve this issue by filing a waiver (asking for permission, in the government bureaucracy) to NIST.
- In the meantime, on my Mac I tried repeating all of last week's shenanigans with `iptables`, only this time actually in the Linux VM:
  - Similar to before, we open two terminal windows and run `multipass shell` to access the Ubuntu shell. We then host HTTP servers on ports 8080 and 9000, using `python3 -m http.server 8080` and `python3 -m http.server 9000`.
  - Like in last week's demo, upon opening a third terminal window and accessing the Ubuntu shell, running `sudo iptables -L -n` brings up the current list of network policies on this Linux VM, which are all set to ACCEPT by default:



```
ubuntu@primary:~$ sudo iptables -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source                destination

Chain FORWARD (policy ACCEPT)
target     prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination
```

- After running `sudo iptables -A INPUT -p tcp --dport 8080 -j ACCEPT` and `sudo iptables -A INPUT -p tcp --dport 9000 -j DROP`, we can check this list of policies again and see that two new “rows” have been added:

*(Continued...)*

Tuesday, May 30th (cont. #1/1)



```
Chain INPUT (policy ACCEPT)
target     prot opt source                destination
ACCEPT     tcp  --  0.0.0.0/0              0.0.0.0/0             tcp dpt:8080
DROP       tcp  --  0.0.0.0/0              0.0.0.0/0             tcp dpt:9000

Chain FORWARD (policy ACCEPT)
target     prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination
```

- By typing `ip a` and checking what the IP address is (in this case, it's not `localhost` like I tried last time—we will go over what `localhost` is later), we visit `http://192.168.64.3:8080` and `http://192.168.64.3:9000`. Done correctly, we should now be able to visit the former page, but not the latter.
- **Note:** to delete (flush) any changes we've made to the network policies on our VM, we can run `sudo iptables -F` to flush the firewall rules.

- Investigated what different **transport layer protocols** are, and how I think a computer is able to distinguish them (so that I can make a firewall treat each differently):

Every **packet** follows a 5-layer structure:

- **Application** → HTTP
- **Transport** → TCP, UDP, ...
- **Network** → IP
- **Link** → WiFi
- **Physical** → Ethernet Cable

The **Transport Layer** allows simultaneous use of a network connection by multiple applications. When the **Application Layer** creates a message, it is sent to the Transport Layer, which then wraps it in a **segment**: it includes additional information like source port and destination port. The segment is then sent to the **Network Layer**.

**TCP:** transmits packets in a streamlined, numbered fashion; even if they arrive out-of-order, they will be re-arranged before reception by another application. TCP uses a congestion protocol to handle high-traffic networks, so while slower and with more overhead space, it makes sense where there's greater bandwidth; an effect of this, though, is less control over when packets are sent. With TCP, re-transmission is possible and also secured by a 3-way handshake: Network A asks Network B to set up a connection, Network B responds, and finally Network A communicates the established connection. TCP might be used to stream videos or send pictures and texts.

**UDP:** transmits packets in a discrete fashion, without order and prioritizing efficiency over reliability or security: useful in instances of low bandwidth, or if packets are mutually uncorrelated. When using UDP, packet loss is much more common as well; packets are only sent once, so if they are lost in transmission, they simply go discarded, unannounced. UDP might be used to send emails or play online video games (packet loss > lag).



Wednesday, May 31st

Investigating Protocols  

	TCP/IP Model (1982)	OSI Model (1984)
HTTP, SSH, FTP, ...	APPLICATION	APPLICATION PRESENTATION SESSION
TCP, UDP, ...	TRANSPORT	TRANSPORT
Ethernet, Switches, ...	INTERNET	NETWORK
Cables, NIC, ...	LINK	DATA LINK PHYSICAL

HTTP	Application	<b>Hypertext Transfer Protocol</b> is the foundation of the World Wide Web, and is used to load webpages using hypertext links. HTTP is an application layer protocol designed to transfer information between networked devices and runs on top of other layers of the network protocol stack. In a typical flow over HTTP, a client machine makes a request—these have a version type, URL, method, request headers, and (optional) body—to a server, which then responds.
SSH	Application	<b>Secure Shell</b> is a cryptographic network protocol for operating network services securely over unsecured networks. Its most notable applications are remote login and command-line execution. SSH applications are based on a client-server architecture, connecting an SSH client instance with an SSH server. SSH operates as a layered protocol suite with three principal hierarchical components: the transport layer provides server authentication, confidentiality, and integrity; the user authentication protocol validates the user to the server; and the connection protocol multiplexes the encrypted tunnel into multiple logical communication channels.
FTP	Application	<b>File Transfer Protocol</b> is a standard communication protocol used for the transfer of computer files from a server to a client on a computer network. FTP is built on a client-server model architecture using separate control and data connections between the client and the server. FTP users may authenticate themselves with a clear-text sign-in protocol, usually with a username and password, but can connect anonymously if the server is configured to allow it. FTP is often secured with SSL/TLS or replaced with SSH File Transfer Protocol.
TCP	Transport	<b>Transmission Control Protocol</b> is one of the main protocols of the IP suite. [See Above]
UDP	Transport	<b>User Datagram Protocol</b> is one of the core communication protocols of the Internet protocol suite used to send messages to other hosts on an IP network. [See Above]
IP	Internet	<b>Internet Protocol</b> is the network layer communications protocol for relaying datagrams across network boundaries. Its routing function enables internetworking, and essentially establishes the Internet. IP has the task of delivering packets from the source host to the destination host solely based on the IP addresses in the packet headers. IP defines packet structures that encapsulate the data to be delivered. It also defines addressing methods that are used to label the datagram with source and destination information. The original Internet Protocol Version 4 (IPv4) is the dominant protocol of the Internet, with successor IPv6.
ICMP	Internet	<b>Internet Control Message Protocol</b> is used by network devices (e.g. router), to send error messages and operational info. indicating (un)successful communications with another IP address. ICMP differs from transport protocols (TCP, UDP, etc.) in that it is not typically used to exchange data between systems nor regularly employed by end-user network applications.
ARP	Internet	<b>Address Resolution Protocol</b> is a communication protocol that discovers the link layer address (e.g. MAC address) associated with a given internet layer address (usually IPv4).
Ethernet	Link	<b>Ethernet</b> is a wired communications standard to network computers in a local environment (LAN, or "Local Area Network"). Ethernet cables are usually Twisted Pair cables that can handle speeds up to 10 Gbps (CAT 7); these come in half-duplex and full-duplex variants, allowing information to be transmitted one-way or both-ways, respectively. Fiber-optic cables are reserved for fast, long distance travel of data. Ethernet devices are computers, routers, printers, or any machine with an internal or external NIC (network interface card); switches and routers enable communication between these devices, while communication across Ethernet networks are facilitated by gateways (for dissimilar networks) and bridges (similar ones).



Thursday, June 1st

GitHub Bug Issue &amp; Wireshark 🐍🐉

- During our discussion last time of the way TCP communicates across a network, the command `netstat -ant` rendered a list of active TCP connections, ports on which the computer is listening, Ethernet statistics, the IP routing table, IPv4 statistics (for the IP, ICMP, TCP, and UDP protocols), and IPv6 statistics (for the IPv6, ICMPv6, TCP over IPv6, and UDP over IPv6 protocols). We were interested in investigating why the following ports were actively LISTEN-ing at all:

```
anthonymalysz@Anthonys-Air ~ % netstat -ant
Active Internet connections (including servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         (state)
tcp4    0      0 127.0.0.1.18171         *.*                     LISTEN
tcp6    0      0 *.5000                  *.*                     LISTEN
tcp4    0      0 *.5000                  *.*                     LISTEN
tcp6    0      0 *.7000                  *.*                     LISTEN
tcp4    0      0 *.7000                  *.*                     LISTEN
tcp6    0      0 *.53                    *.*                     LISTEN
tcp4    0      0 *.53                    *.*                     LISTEN
tcp4    0      0 *.50051                 *.*                     LISTEN
tcp6    0      0 *.50051                 *.*                     LISTEN
tcp4    0      0 127.0.0.1.8021         *.*                     LISTEN
tcp6    0      0 ::1.8021                *.*                     LISTEN
```

- Port `*.5000` is listening for an Airplay service, according to [this](#).
- Port `*.7000` is *also* something to do with Airplay, according to [this](#).  
**Note:** early iTunes let you pirate songs by streaming from another person's device, using things of a similar fashion. Why does Apple have Airplay listening ports on by default?
- Port `*.53` is DNS, according to [this](#) list of what ports usually listen to on MacOS.
- Updated to the newest version of MacOS Ventura last night (that might not be relevant) and trying to run `multipass shell` no longer worked, giving the error `shell failed: Cannot retrieve credentials in unknown state`. This prompted us to look into the **source code** on GitHub, and to file my first **bug report**!  
**Note:** adding `.patch` to the end of a URL on an issue request in GitHub might reveal emails!
- Got on the phone with iTAC and gained administrator access - also will no longer need smartcard to log in to work laptop, so no more login error messages.

Thursday, June 1st (cont. #1/1)



## • mailserver.pcap

This file contains information about a mail server in an AWS account. You can see this server communicating with multiple computers, one of them is a user (maybe from the company running) this mail server and other computers in internal/external networks.

1. **The PCAP file encodes information about different protocols used by the mail server, which protocols are used? How do you know this?**

Wireshark can sort packets in columns for: Time received, Source/Destination Port, Protocol, etc. For more specifics, visit [Statistics > Protocol Hierarchy](#)

2. **What is the IP address of the mail server? Besides the mail server, what is the IP address of other sources and destinations?**

We can look at HTTP communications between IP addresses that appear quite frequently at a glance: trying the query `ip.addr == 169.254.169.254 && http` renders a history of communications between the user 10.0.0.41 and the server 169.254.169.254. By also selecting any packet of interest by [Rclick > Follow > TCP Stream](#), we can view its contents if unencrypted. Another common addresses is 52.1.52.89 (ubuntu@cloudxlarge.com, line #53?).

3. **What protocol does the user's computer use to send mail from themselves to the mail server and someone else? In the filter window, how can you filter on only this traffic from the PCAP in Wireshark?**

Over HTTP, packets' "info" section all start with either GET, HTTP, or POST: the user clicking hyperlinks, the mail server loading those pages (`\webmail`), and the user posting mail (`\compose.php`) in php. When user sends mail to others, SMTP (Simple Mail Transfer Protocol) is used. This was discovered using `ip.addr == 10.0.0.41 && smtp`, and following the TCP Stream of any packet.

4. **RE the previous question, follow all Application-Layer packets in this TCP-based protocol to see what is being sent via email. What is it?**

```
To: "Joe Chandler" <joe@northamericanlumbercoalition.com>
Subject: Proposed salary structure
X-Mailer: mail (GNU Mailutils 2.99.99)
Message-Id: <20171026160829.5F311465F4@northamericanlumbercoalition.com>
Date: Thu, 26 Oct 2017 16:08:29 +0000 (UTC)
From: "Annie Smith" <annie@northamericanlumbercoalition.com>
--952363535-1509298501=:17199
Content-ID: <20171029173501.17199@ip-10-0-80-181.ec2.internal>
Content-Type: text/plain
Here is the latest proposal for the new salary matrix.
thanks,
Annie
```



Friday, June 2nd

Wireshark: File Excavation 

## • attack.pcap

1. When you open this file, there is a lot of data from different protocols. Can you use Wireshark to analyze this PCAP and find the number of packets/segments, the total number of bytes, packets/segments transferred and received, and look at them by protocol (Ethernet, IPv4, TCP, UDP). Does Wireshark have the ability to do that?

Yes, go to `Statistics > Protocol Hierarchy`.

2. What Layer 7 TCP-based protocols are used in this alleged attack that are not HTTP and not HTTPS? What query did you use to find them?

According to [this source](#), port 443 is used explicitly for HTTPS services; so, all packets not using HTTP nor HTTPS can be filtered using `tcp && not http && tcp.port != 443`, which consists of 64.3% of all packets. Go to `Statistics > Protocol Hierarchy` to see all stats within this query.

3. What query can you use, without specifying the tcp source or destination ports, to find TCP packets with data inside of them that have the characters `SMB`?

Well, to search for TCP packets that have the protocol SMB, the query `tcp && smb` works; however, if you want to search for TCP packets with some string "foo" in the info box, the query `tcp` along with `CMD+F > foo` with the case (in)sensitive option works (for `attack.pcap`, the latter returns nothing).

4. RE the previous two questions, one of these protocols shows up more than the others and it seems a lot of packets are exchanged with them, can you use Wireshark to extract what files look like from this traffic? Do some research and tell me: how would you do that with Wireshark?

The former query returns about 64% of all packets, while the latter returns only 2%; in particular, the former is all traffic excluding (most?) of the Layer-7 TCP Protocols, and when inspecting any of these that remain using `Rclick > Follow > TCP Stream`, we can see that they are (all?) encrypted, even when viewed as ASCII characters. By visiting `File > Export Objects > SMB` and saving them to our computer, we can open the excel file `%5cHR%5cNALC-salaries.xls` (it's still cool, despite actually being malware!).

**WEEK III.** (06/05 - 06/09)*Intercepting Network Traffic...*

Monday, June 5th

Firewalls & Network Traffic  

- Research firewalls: their hierarchy, interactions with IDS/IPS, place in a network, etc.

Next-Generation Firewall (NGFW):

- Deep Packet Inspection (DPI), Stuff similar to IDS/IPS, Malware Filtering, AntiVirus, phishing, etc.

Stateful Inspection Firewall:

- Keeps track of the state of network connections (TCP streams, UDP datagrams, etc.).
- Applies labels (LISTEN, ESTABLISHED, CLOSING).
- State table entries are created for protocols that are allowed to communicate through the firewall in accordance with the configured security policy.

Packet Filter:

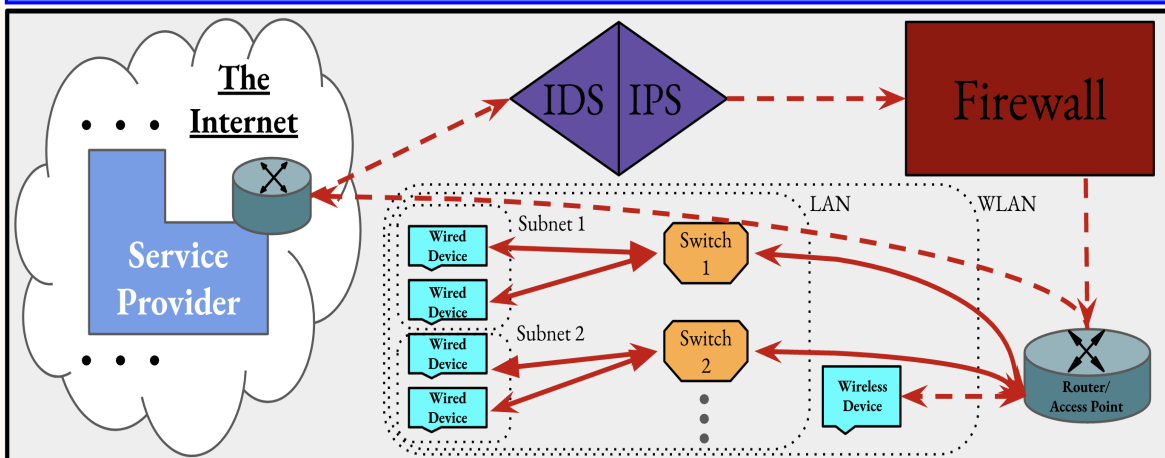
- Sits at the junction with routers & switches.
- Compares packets sent between devices against header info: packet type, port number, IP adr, etc.
- Either silently discards (default), discards with TCP response to sender, or allows packet to pass.

Circuit-Level Gateway:

- Between Application and Transport Layers.
- Avoids the filtering of individual packets.
- Manages endpoint conversations by remembering which port number the two IP addresses use.
- ✓'s TCP handshakes.

Application-Level Gateway (Proxy Firewall):

- Sits between Internet and (W)LAN.
- Proxy firewall  $\Rightarrow$  proxy server, but proxy servers  $\neq$  proxy firewalls.
- An intermediary between clients and servers: caches webpages to reduce bandwidth demand, filters traffic, compresses data, detects viruses, etc.
- Can hide user information or to connect to services that would be blocked.
- Inspects packets at the Application Layer, with knowledge of standard ports and their respective protocols to filter abnormal traffic.
- Uses virtual private networks (VPNs) and encrypted DNS to provide unified security management.







Tuesday, June 6th

UTM &amp; Suricata 🐧 🐱

- **multipass** is becoming more trouble than it's worth; when trying to launch a shell from the primary (or any) instance, it takes forever and times out. Other people seem to have similar problems: see Issues tab of the **multipass GitHub repo**. I suspect this is because of my M1 Mac, so I've looked into what people recommend and found **UTM** as a possible alternative for a Linux VM. Installation steps:
  - Install **Ubuntu 22.04**, choosing the **64-bit ARM (ARMv8/AArch64)** desktop image option instead of the AMD option for this Mac's M1 processor. This will produce a **jammy-desktop-arm64.iso**, which you can save anywhere (desktop).
  - Follow **this video's** instructions to install **UTM**.
  - In the VM, follow **this guide** to install **Suricata**, an open-source based IDS/IPS.
- **Today's Assignment:** You should research how to test a rule against a PCAP file after you install Suricata in Linux, not configuring the whole utility as a daemonized service and leaving it running long-term. You will only need to do the former.
  1. You've been alerted that `27.43.100.29` is on a threat list targeting web services. Write a signature that alerts on accesses from this host to any local web service.
  2. Your organization recently noticed that malicious actors were scanning for `phpMyAdmin` at `/phpMyAdmin/scripts/setup.php` on our web servers. The scan engine did not set a User-Agent. Write an alert for whenever an external host tries to fetch that URI without a User-Agent set. Test with `http.pcapDownload http.pcap`.
  3. A few years ago a phishing attack was launched where the attacker got multiple users to add a malicious service as a trusted service on their Google account. The attack required the victim to click on a link to `https://accounts.google.com/o/oauth2/auth`. Write a signature that finds all email with a link to that address. Test with `smtp.pcapDownload smtp.pcap`.
  4. You are looking for data inside the friends lists on Yahoo! Messenger packets. The protocol is documented here: `http://libyahoo2.sourceforge.net/ymsg-9.txt`. Write a rule that will identify Yahoo messenger packets with a service of `00 f1` (this replaced `YAHOO_SERVICE_LIST` at some undocumented point in ancient history). You can test this against `ymsg2.pcap Download ymsg2.pcap`.



Wednesday, June 7th

UTM &amp; Suricata, Continued 🐼🔄

- Beginning where we left off yesterday, the first necessary step is to set up file sharing between the host and guest machines. For UTM, this will make use of the SPICE WebDAV directory share mode, which which you can access a selected shared directory:
  - Following **these instructions**, ensure that the SPICE agent is installed. The SPICE agent is required for clipboard sharing (both QEMU and Apple backend) as well as dynamic display resolution in QEMU backend. Run these commands:

```
$ sudo apt install spice-vdagent
```

```
$ sudo apt install qemu-guest-agent
```
  - Install the SPICE WebDAV with 

```
$ sudo apt install spice-webdavd
```

.
  - Ensure no virtual machines are running, and open UTM. Select the desired VM, then click the top-rightmost icon in the window. Now, we are in **Settings > Sharing > Directory Share Mode**. Select SPICE WebDAV. Also ensure that the box above it is checked to enable clipboard sharing. Below it is a path where the shared directory folder with the .pcap files can be selected.
  - Then, you should be able to access `http://127.0.0.1:9843` from a browser inside the guest machine once it is running. This will render an HTML view of the shared directory folder that was manually selected in the previous step. You may click through the hyperlinks and click on the files to download them.

**Note:** a **common bug** where the VM's `Spice client tools` folder can't open.
- Continuing with yesterday's assignment, question 1 asks to write a signature that alerts on accesses from to any local web services, with `27.43.100.29` being on a threat list:

```
amalyz@ubuntum1:~/Desktop$ pwd
/home/amalyz/Desktop
amalyz@ubuntum1:~/Desktop$ ls
eve.json fast.log http.pcap smtp.pcap stats.log suricata.log ymsg2.pcap
amalyz@ubuntum1:~/Desktop$ ls -lh
total 148K
-rw-r--r-- 1 root    root      70K Jun  7 13:40 eve.json
-rw-r--r-- 1 root    root       1.4K Jun  7 13:40 fast.log
-rw-rw-r-- 1 amalyz  amalyz   1.9K Jun  7 12:23 http.pcap
-rw-rw-r-- 1 amalyz  amalyz   5.2K Jun  7 12:25 smtp.pcap
-rw-r--r-- 1 root    root       21K Jun  7 13:40 stats.log
-rw-r--r-- 1 root    root       13K Jun  7 13:40 suricata.log
-rw-rw-r-- 1 amalyz  amalyz   9.9K Jun  7 12:25 ymsg2.pcap
amalyz@ubuntum1:~/Desktop$ cd /var/lib/suricata
```

Wednesday, June 7th (cont. #1/1)



```
amalyz@ubuntu1:~/var/lib/suricata$ ls
rules update
amalyz@ubuntu1:~/var/lib/suricata$ sudo nano custom-rules.rules
```

- **Note:** The reason we `cd` into `/var/lib/suricata` is because that is the default rule path for configuring Suricata to load Suricata-Update managed rules, as we can find by `cd`-ing into `cd /etc/suricata`, opening `open suricata.yaml`, and finally searching for the `rule-files:` config directive, as below:

```
1923 ##
1924 ## Configure Suricata to load Suricata-Update managed rules.
1925 ##
1926
1927 default-rule-path: /var/lib/suricata/rules|
1928
1929 rule-files:
1930   - suricata.rules
1931
1932 ##
1933 ## Auxiliary configuration files.
1934 ##
1935
1936 classification-file: /etc/suricata/classification.config
1937 reference-config-file: /etc/suricata/reference.config
1938 # threshold-file: /etc/suricata/threshold.config
1939
```

- Write and **test** four custom Suricata rules in `custom-rules.rules` using its **syntax**.  
**Note:** pre-set labels like `$HOME_NET` and `$EXTERNAL_NET` are in `<some file>`, where custom labels can also be configured for individual/ranges of IPs, including ports:

```
amalyz@ubuntu1:~/var/lib/suricata$ sudo nano custom-rules.rules
alert http 27.43.100.29 any -> $HOME_NET any (msg:"threat list";
  flow:to_server; sid:1; rev:1;)
alert http any any -> $HOME_NET$ any (msg:"phpMyAdmin scan";
  flow:established,to_server; content:"GET";
  http_uri:"/phpMyAdmin/scripts/setup.php"; http_header:"User-Agent";
  sid:2; rev:2;)
alert smtp any any -> $HOME_NET any (msg:"phishing"; flow:to_server;
  content:"https://accounts.google.com/o/oauth2/auth"; sid:3; rev:3;)
alert tcp any any -> $HOME_NET any (msg:"Yahoo"; content:"|59 4d 53 47 00 13
  00 00|"; content:"|00 f1|"; distance:2; within:4; sid:4; rev:4;)
```



Thursday, June 8th

Q&amp;A with VPNs 🧑!?

- **What is the basic premise of a VPN?**

VPN means “virtual private network”. VPN services will pass your data through their VPN client downloaded onto your device instead of going directly to your ISP from your modem—this is to encrypt the traffic from third parties. It then goes from them to your VPN server, and from there to the internet. This last step allows you to configure your appeared location, as companies have VPN servers all over the world.

- **How does a VPN's cryptography work? Do you retain total anonymity?**

When the VPN client and VPN server initiate a connection, they must authenticate each other and establish an encrypted tunnel. Authentication methods are usually...

Pre-shared keys are exchanged by both client and server as a means of mutual authentication to initiate the tunneling protocol: this begs the question of how sharing of said keys is secure in the first place, and may take place such that each downloaded client comes with a unique key and a key to expect from the server (or, these passwords may be exchanged via encrypted traffic over the network).

Certificates are the better-scaling option, but they need to be purchased and issued from the CA (Certificate Authority). They contain the public key for a digital signature and specifies the identity associated with the key, such as the name of an organization. The certificate is used to confirm that the public key belongs to said organization.

Privacy and anonymity are often confused services that a VPN provides, which only includes the former. When intercepting traffic from a VPN or attempting to keep logs of it as an ISP, packet contents are encrypted, but meta data such as source/destination ports, the VPN's IP address(es), number and frequency of connections, amount of data transferred, etc. will always remain visible. Profiles can be built around this meta data to associate with individual devices. The VPN services still remain useful, though.

- **Like with overlay of security protocols, how can multiple services be using the same port(s) simultaneously? How many ports are there?**

To some degree of abstraction, ports are just reserved lines of communication that can be recognized by a sender and receiver, and they shouldn't be thought of as physical space residing anywhere. For reasons that “seemed to be correct at the time,” ports are assigned 16-bits each in the header blocks of each TCP and UDP packet—in total, 32-bits. This means  $2^{16} - 1 = 65535$  possible ports to be used at any given time. The reason a port can receive multiple communications simultaneously is because of the different sender port labels: they're just a way for the computer to keep ongoing conversations separate, and for said lines to know when to be expecting answers.

Thursday, June 8th (cont. #1/2)



- **Is it more secure to run SSH through a non-default port? How can someone find out an SSH service is running on any port without prior knowledge?**

By running SSH through a non-default port, you prevent communication from all sources that know to look for it, and lower the scope of sources that can contact the server. This filters false-malicious connections to a minimum from scripted attacks, meaning repeated failed connection requests suggest the server is being targeted. This also means that any automated or trusted sources that know to look for the default port must also be informed, however.

Taken from this [source](#), to know what port a particular ssh connection will use, you can look in `/etc/ssh/ssh_config` for any system/global hosts configured, and in `/.ssh/config` for any user/local hosts. To know what port will be used by default, you can use the command `$ getent services ssh` (the standard port is 22). To know what ports are actively in use, the command `$ netstat --tcp --programs --numeric | grep ssh` should show you.

- **What are the VPN protocols IPsec, OpenVPN, and Wireguard?**

Where IP is the standard that says how and where data should go over the internet, IPsec adds encryption and authentication on top of that process to avoid unwanted monitoring of network traffic; it scrambles information at the source and unscrambles information at the destination. IPsec connections are commonly used with VPNs when accessing files remotely, for example (like with NIST).

OpenVPN is an open-sourced VPN protocol integrated into a lot of commercial VPNs' models. It uses UDP primarily for the sake of having fast connections—some of which are across the world streaming video, so speed is more of the essence (and if data is scrambled on top as a security measure, packet order won't make a difference and neither will dropping a handful of them in comparison)—but also supports TCP connections as a backup option.

Wireguard is a third free, open-sourced tunneling protocol that uses only UDP with the goal of improved performance over the aforementioned two. It's also said to be fast because its low-level components are built into the Linux kernel, making it faster than userspace VPNs. It launched in 2015 with newer and faster cryptographic methods, also focusing on minimizing attack surface against hackers. It does not counter deep-packet inspection alone, however, but has the architecture to support further obfuscation.

•  
•  
•

*(Continued...)*

Thursday, June 8th (cont. #2/2)



- **How does a VPN client (say, Ubuntu Linux) know to not leak data by sending traffic outside the VPN (through the normal network connection)?**

1. **Routing:** Upon establishing a connection, the VPN client modifies the routing table on the OS, adding a route that directs all traffic through the VPN tunnel.
2. **Network Firewall:** The VPN client may also employ a network firewall to block any outgoing network connections that are not intended to go through the VPN tunnel. The firewall rules are set up to allow only the VPN traffic to pass through.
3. **DNS Leak Protection:** DNS (Domain Name System) leak is a common issue where DNS requests made by the operating system bypass the VPN tunnel and are resolved by the default DNS servers of the network. To prevent this, the VPN client typically configures the system to use its own DNS servers.
4. **Kill Switch:** A kill switch is a fail-safe that monitors the VPN connection status and immediately blocks all network traffic if the VPN connection drops unexpectedly, preventing any data from being transmitted outside the VPN tunnel.

- **How could you tell that someone is using a VPN, and how could you hide that fact from someone else?**

An ISP logs the data of its users, and when it expects to read that data and finds that it's encrypted upon arrival, then it knows a VPN is being used—this is true for anyone intercepting traffic from the (W)LAN to the ISP. The solution is to hide the packets in plain sight, disguising their contents as something else that *is* readable. A real-life motivation for disguising a VPN is in countries where VPNs are illegal. The scope of IP addresses should also be taken into consideration, as some entire countries may even share (a small range of) them.

- **Looking for meta data—even surrounding encrypted packet contents—is a standard practice; how could someone use cryptography to apply a further layer of security and hide that meta data?**

The concept of analyzing meta data hinges on the fact that said data is truthful, meaning said user's actions are uninfluenced by surveillance. If someone were to apply noise to this meta data by sending garbage packets randomly in the mix, the data becomes harder to extrapolate. For instance, a second layer of meta-security could be enforced where packets are sent along a TCP stream along with garbage packets: the real packets are encrypted by the usual encrypted tunneling, but the garbage packets' placement is well-ordered and adheres to an additional pre-exchanged sequence (e.g. the real packets' indices in the stream correspond to Fibonacci numbers).



Friday, June 9th

VPN Lab with AWS 

- Today's agenda will be to perform a VPN lab. We will buy digital infrastructure from AWS to run a server that we will communicate with using OpenVPN, in order to observe how it encapsulates packets. The goal will be to SSH into the server using our remote machine (Ubuntu Linux VM), capture that traffic on our host machine in the form of a `.pcap` file using `tcpdump`, and finally examine said file using Wireshark:

1. Accept an invitation to create an account with **Okta** (an American identity and access management company), which will allow AJ to add you to an AWS account `surf-vpn-research` that can be accessed from the Okta dashboard.
2. Visit the Console Home of `surf-vpn-research`, and then go to the EC2 tab (Elastic Control Cloud allows users to rent virtual computers on which to run their own computer applications).
3. Select the running instance `openvpn-test`, and click "Connect"; in the connection options, select the Session Manager tab to enable SSH connections to the server.
4. As long as the server is running, ideally you should be able to ssh into the server on the Ubuntu Linux VM's console; however, we experienced some trouble with this, and will try again on Monday. Instead, we can SSH into the server on our Mac's terminal with the server's IP address: `ssh ssm-user@54.211.93.59`.  
**Note:** at this point, the server is configured to only allow connections over two ports from public Internet: TCP/22 for SSH and TCP/443 for OpenVPN TCP.

5. Following this **guide**, first ensure that `tcpdump` is installed by running `$ brew install tcpdump` in a separate terminal.

6. After SSH-ing into the server, run `$ sudo tcpdump -s 0 -i eth0 -w tcpdump.pcap` to begin capturing traffic with `tcpdump`. You may let it run for about a minute or two (or until you've done something you'd like to examine while collecting traffic) before ending the session with `Ctrl+C`.

7. Before you can copy the traffic from (what should be) your remote computer to the local one for analysis with Wireshark, you'll have to change the permissions. By default, `tcpdump` sessions captured by the root user can't be copied. Change this by running `$ sudo chmod 644 tcpdump.pcap`.

8. To copy the new file `tcpdump.pcap` to the host machine, run `$ scp ssm-user@54.211.93.59:home/ssm-user/tcpdump.pcap .\`.

The `.pcap` file can now be opened and analyzed with Wireshark on the host computer.



**WEEK IV.** (06/12 - 06/16)**Researching PCAP Expressions...**

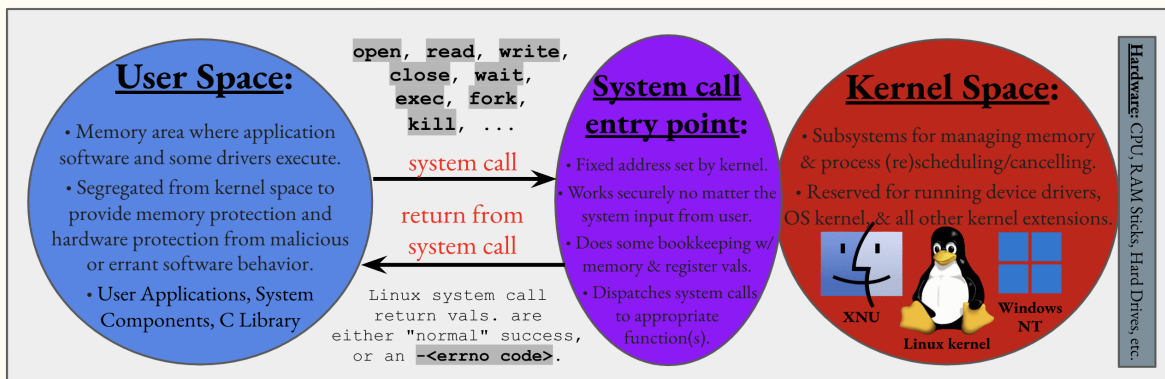
Monday, June 12th

PCAP Expressions **What we talk about when we talk about pcap expressions**

Nik Sultana, University of Pennsylvania

Nik Sultana. 2019. What we talk about when we talk about pcap expressions. In Real World Domain Specific Languages (RWDSL), February 17, 2019, Washington D. C., DC, USA. ACM, New York, NY, USA, 9 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

- **BSD Packet Filter (BPF - Berkeley Packet Filter) : PCAP :: Assembly : C**
  - The most successful host-based system for capturing network packets.
  - Included in the `libpcap` library, the basis for `tcpdump`.
  - BPF is in the form of instructions for a VM, which are compiled into machine code by a just-in-time (JIT) mechanism and executed in the kernel to reduce overhead from userspace-kernel transitions (see below).
- These filters form a domain-specific language (DSL) called “**PCAP expressions**”.
- These are either fed to IDS/IPS or can be exported as `.pcap` files for offline analysis.



- **Caper** is a free open-source language provided with the formal semantics of the PCAP expression language to act as an equivalence-checker tool.
  - It can emit warnings for queries with unexpected behaviors, such as non-isomorphic permutations of clauses that lead to subsumations (see below).
  - It implements a parser for the full PCAP language and its semantics to expand them into equivalent PCAP expressions.
  - These can then be fed to **SMT solvers**, which are like SAT Solvers except generalized to mathematical formulae instead of just boolean statements.
  - Equivalences are determined using bit-vector semantics (see Figs. 1-3).





Tuesday, June 13th

PCAP Literature 🍷📖

*The BSD Packet Filter:  
A New Architecture for User-level  
Packet Capture*

Steven McCanne & Van Jacobson,  
Lawrence Berkeley Laboratory

McCanne, S., Jacobson, V. 1993. The BSD Packet Filter: A New Architecture for User-level Packet Capture. In Proceedings of the 1993 Winter USENIX Conference (January 25-29, 1993, San Diego, CA). Preprint version. Lawrence Berkeley Laboratory, Berkeley, CA. Technical Report No. LBL-33878. December 19, 1992. Supported by the Director, Office of Energy Research, Scientific Computing Staff, U.S. Department of Energy, Contract No. DE-AC03-76SF00098.

- **BSD Packet Filter (BPF - Berkeley Packet Filter) : PCAP :: Assembly : C**
- This paper is likely more involved in eBPF than PCAP expressions for our purposes.



Wednesday, June 14th

PCAP Translators 🍷🎉

- There is a production team behind **tcpdump** and **libpcap**—a powerful command-line packet analyzer and a portable C/C++ library for network traffic capture, respectively—that have produced a tool called BPFExam; essentially, borrowing some features from Professor Sultana's project **Caper** (which we reviewed a few days ago) to provide the caper expansion and caper translation from PCAP to English, and also from PCAP to bytecode accessible by the OS kernel.
  - Ideally, the tool we are building should be able to do the reverse: take an English expression of the format translated by BPFExam, and translate it into a PCAP expression. This would obviously never include all free-form English, but rather a formatted version of it—perhaps including artificial symbols to parse clauses, etc.
  - In addition to its functionality as a translator from English to PCAP, our project should likewise rely on some frontend web development like Caper; this will at some point require some research into web frameworks that integrate OCaml, if they exist at all. I suspect the backend would be too ambitious in scope.
- Without getting too much ahead of ourselves, we would need to keep in mind a “zero-trust” policy of user-input into our program; not only to produce the intended output correctly, but for possible security concerns regarding injection attacks. We can calm it down to simple phrases, and maybe we can define keywords like “workstation network” and “Internet” depending on a provided definition? I suspect this would work like an OCaml interpreter would, using a substitution environment and defining these words in various dictionaries. Everything should cater towards **Behavior-Driven-Development (BDD)**, or “customer representatives” who would use the product.



Thursday, June 15th

Building PCAP Expressions 🎨👤

- Tried my hand at manually translating English network policy into PCAP expressions:
  - “The organization should only allow DNS traffic for all staff workstation networks to the approved internal DNS service.”  $\implies$ 

```
dst port 53 and (src net <work_net> or dst net <work_net>)
```
  - “The organization should only allow web traffic from all staff workstation networks to the public Internet from the organization’s web proxy service.”  $\implies$ 

```
(dst port 80 or dst port 443) and src net <staff_net> and ...
... dst net <public_net> and src host <web_proxy>
```
  - “The organization should not allow any VPN traffic from the inside all staff workstation networks to the public Internet.”  $\implies$ 

```
not (src net <staff_net> and dst net <public_net> and ...
... (src port <vpn_port> or dst port <vpn_port>))
```
  - “The organization’s web traffic from all staff workstations network through the organization’s web proxy service (VPN). These workstations should not be able to visit any of the following websites in the list `prohib_list = [www.foo.com, www.bar.com, www.zoo.com]`.”  $\implies$ 

```
src net <staff_net> and dst net <public_net> and src host <web_proxy> and
not (dst host www.foo.com or dst host www.bar.com or dst host www.zoo.com)
```
- Drew up a **rough outline** of our project’s soon-to-be substitution algorithm in GDocs.



Friday, June 16th

Dev's Log #01 🧑📖

- Reviewed some publication/intellectual property licensing stuff with A.J. and Nikita.
- Inaugural commit on **GitHub**.
- Finished first working prototype over the weekend:
  - Added `eng_to_pcap.ml` file, wherein all code is hosted currently: will soon abstract this into separate files when project scales up.
  - Added usage comments: will move these to a `README.md` file eventually once project scales, and hopefully put together a makefile soon.
  - Added `tests.ml` file, which prints results of tests of the form:
 

```
let test_cases = ["<eng_input>", <pcap_output>"; (* other tests... *)]
```

**WEEK V.** (06/19 - 06/23)*Starting Development...*

Tuesday, June 20th

Travel & Pen-testing Enrichment 

- Travel to Washington, D.C.
- Some neat discussion about **pentesting airplane Wi-Fi** to steal others' paid in-flight services via **Mac address tampering**, and how it relates to our discussion of VPN and their tunneling protocols in our policy examples—those tools, like **this one**, use only DNS and a custom server (so tunneling all traffic over DNS, not just DNS queries) since those captive portal WiFis never track that.



Wednesday, June 21st

Generating Test Expressions 

- Asked ChatGPT to generate 50 network policy **examples**; translate them into PCAP expressions if possible, and if not, explain why it wasn't possible; and answer select questions for certain policies to steer it in a particular solution's direction, prompting it again to try and generate PCAP expressions for those.
- There are certain scopes of technology outside of the OSI Model that simply can't be touched by PCAP expressions alone: how **Role-Based Access Control (RBAC)** could be implemented on a network level, checking permissions for specific machines/hardware on the network, whether foreign devices are plugged into work machines (using a **netstat** command to see which ports are in use), checking network traffic to see if something is being downloaded, etc.



Thursday, June 22nd

Dev's Log #02 

- Added a **makefile**, some **more tests**, and **more dictionary substitutions**.



Friday, June 23rd

Dev's Log #03 

- Added a **debugging mode**, **improved style**, and **fixed a clause duplication bug**.

**WEEK VI.** (06/26 - 06/30)*Building a Prototype...*

Monday, June 26th

Dev's Log #04 

- Brainstormed potential additional functionality that wouldn't exceed the scope of the project, and eventually settled on augmenting how lists of clauses vs. lists of primitive filters could be passed in by the user and recognized by the algorithm.
- Updated the **README.md** file to include our newly-drafted Abstract for submission.



Tuesday, June 27th

Dev's Log #05 

- Made some improvements that resulted in a **new bug**. Finalized our **Abstract**.



Wednesday, June 28th

Dev's Log #06 

- Hiatus.



Thursday, June 29th

Dev's Log #07 

- Fixed a bug that incorrectly searched for a `list_finisher`.



Friday, June 30th


Dev's Log #08 

- **Massive overhaul**: split the `eng_to_pcap.ml` file into several files that group relevant pieces of code together. Looking into using **dune** as a building tool rather than a makefile in hopes of fixing that `"Error: Unbound"` bug in the codebase. After that's done, I'll make `.mli` files and start documentation.

## WEEK VII. (07/03 - 07/07)

## OCaml Web Frameworks...

Monday, July 3rd

Web Frameworks: Incr\_dom, by Jane Street 

- Added a `.bashrc` file to allow execution of `eng_to_pcap.exe` and `tests.exe` from any path in the repository (ease-of-use).
- Found and resolved an issue with the `/bin/dune` stanzas wherein `eng_to_pcap.ml` was meant to be treated both as a library and an executable; resolved this by simply calling `eng_to_pcap.ml` with `main.ml`, having the former remain a library defined in `/lib/dune` and the latter being defined as an executable in `/bin/dune`.
- Investigated potential frontend web frameworks that allow integration with OCaml code, and found Jane Street's **incr\_dom** library. Watched through its tutorial demo [here](#), and added it as a dependency library in our **project**.



Wednesday, July 5th

Web Frameworks: Dream 

- At this point, we've shifted our focus from writing the program to building its digital interface in the form of a web application. While **Incr\_dom by Jane Street** looked promising, a web framework called **Dream** looks to be simpler and therefore more promising. Its repository includes several **examples**, which we'll go through today:
  - `1-hello` We first notice the command `npm install esy && npx esy`; recalling the syntax of `&&`, we know that what follows it is only executed if what precedes it is successful (or `true`, 0, etc.). The package manager **npm** is written in JS and is the default package manager for the JavaScript runtime environment **Node.js**. Recall in **opam** the concept of **switches**, or independent installation prefixes with their own compiler and sets of installed and pinned packages; typically, you would need to create and choose which would be the active one, but **esy** is a rapid workflow for developing Reason/OCaml projects without switches. The command `npx esy` initializes the Esy environment in the project: it looks for an `esy.json` or `package.json` file in the current directory or the nearest parent directory and sets up the necessary dependencies and build configurations defined in that file. More on `.json` files later. Taking a look at the code, `let () = Dream.run (fun _ -> Dream.html "Good morning, world!")` is where Dream “translates” OCaml to HTML quite succinctly: curious to see the CSS.

Wednesday, July 5th (cont. #1/2)



- **2-middleware** The concept of *Middleware* is simple to understand, and perhaps the best-known example is the *logger*: it can be made super useful with some syntactical sugar (on that note, I learned `@@` is neater function composition).
- **3-router** The concept of a *router* invites our site to be dynamic, but should also usher some caution from trusting user-input; we will touch upon this in our discussion of injection attacks. So far, the simplest action our router can take is receive GET requests to `/echo/*` and regurgitate them somehow.
- **4-counter** This is a good example of proper ordering in code: `let count_requests inner_handler request = count := !count + 1;` comes first before the page is GET request is even processed, and afterwards the arguments `inner_handler request` are themselves just executed together!
- **5-promise** The problem with the previous example was its inability to distinguish between successfully-handled requests and those that aren't; here, we explore the actions both preceding and *following* the handling with a `try%lwt ... with` `catch` (I suspect the use of the `Lwt` library because we're outside of the application layer?). We can then match the output to the respective HTML with `Dream.get`; a question: does the order of these matter?
- **6-echo** We pivot to echoing post requests from a separate terminal.
- **7-template** This is our first look into the interweaving of OCaml and HTML, and a fantastic example of *cross-site-scripting attacks (XSS)*. That's a misnomer and could more **injection attack**. I want to look at `Dream.html_escape`, which "escapes a string so that it is suitable for use as text inside HTML elements and quoted attribute values", but "is not suitable for use with unquoted attributes, inline scripts, or inline CSS." The former functionality is borrowed from **OWASP**, a leading international non-profit cybersecurity org, which publishes an annual *Top 10 Web Application Security Risks* poll. How `Dream.html_escape` replaces the harmful formats (`%s`, `%S`, `%c`, `%C`, `%a`, and `%t`) capable of emitting dangerous characters (`<`, `>`, `&`, `"`, or `'`) is by replacing them with a "look-alike" via **URL encoding**, such as `<` → `%3C`, `>` → `%3E`, `"` → `%22`, etc. A very useful general tool surrounding encryption, encoding, compression and data analysis is **CyberChef**, a tool developed by the **GCHQ** (British NSA).

•  
•  
•

(Continued...)

Wednesday, July 5th (cont. #2/2)



- `8-debug` Like some previous examples, uses `Dream.get` to capture the path component and match it to different error pages like `/bad` or `/fail`. Also uses the logger to display information like the error message; a stack trace; if the error is an exception; `From:` which part of the HTTP stack reported the error (TLS, HTTP, HTTP/2, WebSockets, or the app); `Blame:` who is likely responsible for the error, the server or the client; `Severity:` a suggested log level for the error; `Client:` the client address; request headers; any other request variables.
  - `9-error` This is the final major example: cleverly customizing error messages.
- All examples that follow this will be looked into after the prototype has been integrated with Dream and examples 1 through 9 seem to be working smoothly. Eager to get this visual interface up-and-running to finish working on the functionality of the program, and then to make it look prettier and perhaps check out examples `a-` through `z-`



Thursday, July 6th

Dev's Log #09



- After getting the separate moving parts of `/NGPaC/test` and `/NGPaC/bin` to work, unresolved errors are left with the web app in `/NGPaC/app`: severed communication between `/bin` and `/app` that didn't exist before: it's caused by the presence of `/app/dune-project`, which painstakingly also needs to be there. **TODO:** fix this.



Friday, July 7th

Dev's Log #10



- Fixed the **TODO:** `/app/dune-project` issue temporarily by simply removing the file and ensuring the project's `main.exe` and `tests.exe` executables compiled and ran smoothly separate from launching the site in `/app`. The next step will likely be to either try to expand functionality of `eng_to_pcap.ml` by writing the `remove_excess_parens` function and perhaps looking at some more ChatGPT-generated test cases, or integrate the program into the site via a more complicated prototype; this involve as little complication as possible, as I already see a fullstack-/backend effort going horribly wrong for several weeks. On that note, there's a cool blog by a Penn State student using OCaml Web frameworks to write a **fullstack project**.

## WEEK VIII. (07/10 - 07/14)

## Web Exploits &amp; Testing...

Monday, July 10th

XSS &amp; CSRF Attacks 🍪🐱

- Updated this blog with the last 2 weeks' worth of entries, because I was being lazy.
- Pushed **changes** that removed redundant code, fixed the whitespacing issue, and elaborated upon the debug statements. **Note:** I should look over the project to look for similar code, not sure why I thought to trust the Internet...
- Discovered that the **reason** why my `.gitignore` wasn't ignoring `_esy/` or related folders was because their compilation includes their own `.gitignore` file with a `!*` override command; this isn't an issue, but the project appears as mostly JavaScript.
- Will begin fleshing out the site, starting with a **switch** from the Dream `example/7-template` to `example/d-form`, to allow for secure user English input.
  - Looking through `example/8-debug` and `example/9-error`, I see potential to include a beautified error page once the site's main functionality is finished. The problem with this, though, is that `~error_handler:Dream.debug_error_handler` looks in the URL for a `Dream.get` matching, and coincidentally, the `example/d-form` template is implemented such that an XSS attack is made possible by inputting `/bad` the textbox. **TODO:** fix this (via URL decoding?).
  - Another security-related concern presented by `example/d-form` is the threat of **Cross-Site Request Forgery (CSRF “c-surf”)** attacks. This is prevented in the form's first field via `Dream.csrf_tag`, which returns a token ``Ok _`.
  - Real-life validation tokens will be used in a required challenge-response mechanism that verifies the identity (origin) and authority of the requester, instead of immediately trusting them because of the session cookies kept by the browser.
  - Bad tokens like ``Expired _` or ``Wrong_session _` are indicative of a compromised session, and require special handling for each; additionally, a safer implementation over replying to the form POST directly with HTML involves a redirect request opened in a separate tab with `Dream.redirect` to avoid compromise of sensitive data when dealing with login/authentication pages. It is also important not to use GET requests for state-changing operations.
- Pushed **changes** that integrated custom error-page display with the form template.



Tuesday, July 11th

Insufficient Testing  

- I spoke with AJ about some immediately-relevant career advice and drawing up a short-term roadmap to prepare for the upcoming job search in cyber for Summer 2024.
- I decided experimenting a little bit with how to beautify the site, and ended up including some tables in the body of the HTML in `index.eml.html` (it can be renamed to end in either `.html` or `.ml`, depending on whether it's mostly HTML or not, so that the syntax highlighting picks up most of the relevant code) that display the Egyptian deity Thoth. (I think that's a more interesting name an theme that can be added to the name, perhaps, instead of a long-winded acronym like NGPaC. Also, Thoth is the Ancient Egyptian god of writing, magic, and wisdom. He is believed to have invented hieroglyphic writing and was considered the scribe of the gods. He was also associated with translation and interpretation, hence the possible project name.) **TODO:** figure out how to import HTML in Dream instead of copy-pasting huge chunks of it sloppily.
- I realized that this project is getting way too big to not have tests for every single helper function to ensure that they're working properly at this point (because some of them definitely are not for edge cases). **TODO:** make a scalable testing file for each file in `/lib` to test its respective functions: much like in the current style of `tests.ml`, which should afterwards be renamed to something like `final_tests.ml`



Wednesday, July 12th

Dev's Log #11  

- Spent the day writing **test files** for functions in `lib/helpers.ml`.  
**TODO:** generate tests for each of them, across all files in `/lib`.



Thursday, July 13th

Dev's Log #12  

- Finished the testing file template for `test/test_helpers.ml`. **TODO:** fix the style.



Friday, July 14th

Dev's Log #13  

- Fixed the **style** in `test/test_helpers.ml`, and also introduced some beautified print-methods. **TODO:** replicate for remaining `test` dir, and start generating tests.

**WEEK IX.** (07/17 - 07/21)*Finalizing Scope...*

Monday, July 17th

Dev's Log #14 

- **Implemented** remaining testing files in `test/`, abstracted away common printing functions into `test/printer.ml`, and added new path script(s) for the new functions.
- **Updated** and beautified printing format of `test/final_tests.ml`.
- **Enabled** site communication with `Eng_to_pcap`, so you can now input English prompts and have them translated into PCAP expressions that display as HTML. **TODO:** Research government and industry policy to add relevant tests to `test/`; finalize scope; beautify site; write documentation; create presentation/final-writeup.



Tuesday, July 18th

Dev's Log #15 

- Began writing tests for and debugging all functions in `test/test_helpers.ml`.



Wednesday, July 19th

Campus Visit 

- Visited the NIST Gaithersburg Campus, where I continued writing tests for `test/`.



Thursday, July 20th

Dev's Log #16 

- Finished writing bulk of tests, but will postpone the remaining few until after everything else is finished—things seem to be working for now.
- **Debugged** the project such that all tests in `final_tests.ml` now pass. **TODO:** project needs support for breaking apart separate sentences/sentences that can be immediately split apart according to “, and”, etc. Project needs finalized scope.



Friday, July 21st

Dev's Log #17 

- **Postponed** completing tests, instead debugged project so all `final_tests` pass.

**WEEK X.** (07/24 - 07/28)*Preparing Presentation...***Monday, July 24th****Outlining Presentation** 

- Created **presentation** in Google Slides, outline all slides, and finished slides 1-4.

**Tuesday, July 25th****Drafting Presentation #01** 

- Finished slides 5-8 of presentation.

**Wednesday, July 26th****Drafting Presentation #02** 

- Finished slides 9-16 of presentation.

**Thursday, July 27th****Finalizing Presentation** 

- Started building site to take screenshots of for presentation's Demo Appendix.
- Finished presentation (27 slides), including References & Acknowledgements.

**Friday, July 28th****Practicing Presentation** 

- Practiced presentation.

**WEEK XI.** (07/31 - 08/02)*Student Colloquium...*

Monday, July 31st

Dev's Log #18



- Continued site development.



Tuesday, August 1st

Dev's Log #19



- Continued site development.



Wednesday, August 2nd

Dev's Log #20



- Borrowed `LICENSE.md` and `CONTRIBUTING.md` from the NIST **OSCAL** team.



Thursday, August 3rd

Final Presentation &amp; Parting Thoughts

- Gave presentation to Student Colloquium with other IT Lab SURFers.
- **Published** site homepage prototype.
- Finished developer's blog.
- **Parting Thoughts:**
  - Publish public repo on my profile.
  - Set up frontend & backend.
  - Add file upload feature (`.txt`, `.json`, `.pdf`, etc.)
  - Create database populated by data of users' volunteered unrecognized inputs (to manually expand functionality, i.e. Behavior-Driven-Development).
  - Integrate other PCAP query tools (e.g. Caper, BPF Exam, etc. subject to licenses).
  - Offer a plugin to Wireshark to filter network traffic and alert users automatically (like an IDS would).
  - Look into MIDAS record with A.J.