CUPID: A Labeled Dataset with Pentesting for Evaluation of Network Intrusion Detection

Heather Lawrence[a], Uchenna Ezeobi[a], Orly Tauil[b], Jacob Nosal[b], Owen Redwood[b], Yanyan Zhuang[a], Gedare Bloom[a]
a University of Colorado Colorado Springs, Colorado Springs, Colorado 80918
b Nebraska Applied Research Institute, University of Nebraska Omaha, Omaha, Nebraska 68106
Corresponding Author: hlawrenc@uccs.edu

Abstract

Reproducibility of network intrusion detection research necessitates widely available datasets that represent real-world scenarios. One of the key omissions of existing datasets used in empirical evaluations of network intrusions is the lack of human-generated traffic with accurate labels to distinguish benign and malicious behavior. Using an emulated network environment with a vulnerable web application, we collected baseline traffic, human-generated normal user traffic, automated attacks, and the attacks of ten human penetration testers of varying abilities. We preprocessed this collected data to produce a new dataset named the Colorado University Pentesting Intrusion Dataset (CUPID). The attacks span from reconnaissance activities to delivery of an exploit payload. To our knowledge, this is the first collection that provides labeled, Institutional Review Board-approved, benign and attacker data that is publicly available. The CUPID dataset can be used to train and test the limits of classification-based machine learning algorithms used for network intrusion detection systems.

1. Introduction

Network intrusion detection is a process of ingesting and analyzing large amounts of traffic to determine the presence of an adversarial agent. A significant problem in the evaluation of intrusion detection techniques is the lack of high-fidelity data that is representative of the operational environment. Usable network data has been traditionally difficult to obtain [1, 2, 3]. As a result of poor data, problems in conducting network intrusion detection research include: limited availability of labeled malicious and benign research data, poor translation of experimental results to operational validity, detection mechanisms that suffer from superfluous false positives at scale and incur nontrivial analyst overhead and eventual distrust [4, 5], network data that cannot be released due to leaking network stakeholder information or violation of user confidentiality [6], and poor fidelity due to cleanliness of simulated data versus real data [1].

A robust network intrusion detection dataset includes (1) labels, (2) a flexible data model, and (3) realistic generation. Labeled data is necessary for evaluating ground truth in measuring classifier performance and a prerequisite for exploring supervised learning algorithms. Flexible data models like packet captures that include richer data, such as fully unencrypted network traffic payloads, are necessary for some experiments but require more storage and processing capabilities. Network flows, in contrast to packet captures, represent a distillation of raw traffic data into predefined features, like source IP address or protocol, that can save on space and processing. However, feature visibility can be lost in distillation and the original traffic cannot be reconstructed [6].

To realistically generate network traffic for datasets, three methods are commonly used. These methods include: (1) model-based approaches that use stochastic traffic models; (2) trace-based approaches that use data and measurements from a real environment; and (3) network testing devices that test devices by generating packets following a specific protocol [7, 8]. Model-based approaches use complex models to generate traffic that can be difficult to implement depending on the complexity of the model, but they can yield highly accurate results. Trace-based approaches, in contrast, contain real packets and unmodified headers because the data is recorded live from the network. However, trace-based approaches require automation scripts, system configurations, and system updates that add complexity. Returning to a baseline configuration post-experiment or to add functionality incurs additional overhead. A network testing device accurately times packets and has great performance. However, testing devices are considered unsuitable for generating a wide variety of traffic at once as the devices are generally designed to test a specific protocol at a time and the packets do not contain authentic payload data or headers. Among the three common methods, the trace-based approach offers the most realism but also the greatest complexity.

In this paper, we present the Colorado University Pentesting Intrusion Dataset (CUPID), a trace-based dataset that is intended to aid network intrusion detection research by providing updated, accessible, and labeled network traffic. CUPID was designed to allow researchers to investigate the differences between automated and human-generated attacks at the feature level, as it contains both scripted and human-generated attack traffic, packet headers, and payloads while sensitive network implementation details and user data were not included.

We designed the network to provide protocol variety, maintain a majority of TLS 1.2 traffic, and provide a simulated enterprise environment for human penetration testers. We document our approach to enable thorough understanding of the dataset and to facilitate possible reproduction by others to create their own variations of CUPID. Since packet captures cannot be reconstructed from network flows [6], the original packet captures from the network tap, the processed network flows, and the feature preprocessing scripts are provided as part of the dataset. CUPID is freely available[1] for public use.

[1] `https://www.cupid.directory`

The remainder of this paper is organized as follows. We discuss the related work in Section 2, which is followed by our methodology in Section 3. Section 4 describes the involvement of human participants in the creation of CUPID. We explain the data collection approach in Section 5. Section 6 provides analysis of the dataset features. We discuss some of the challenges we faced and how we overcame them in Section 7. We discuss possible future work in Section 8 and we conclude the main body of this paper in Section 9. Finally, Appendix A summarizes interview responses of the penetration testers involved in this project, and Appendix B details the hardware/software specifications of the CUPID collection infrastructure.

2. Related Work

We reviewed related academic and community cybersecurity datasets to determine the extent of datasets that contained human- and automatically-generated traffic that was also labeled. We reviewed network data generation methods to determine suitability to generate such a dataset. In this section we summarize our findings.

2.1. Early Datasets

Early popular public network intrusion detection datasets include packet traces from the 1998 DARPA Intrusion Detection Assessment and the data provided for the Third International Knowledge Discovery and Data (KDD) Mining Tools Competition, colloquially known as the **KDD99** dataset [9]. KDD99 contains 4.9 million attacks gathered from monitoring a simulated U.S. Air Force network for a total of 9 weeks of which 7 weeks contained training data and 2 weeks contained test data [10]. This data was gathered in 1998 and refined in 1999. The KDD99 dataset distills the network traffic into a custom data structure and adds labels for machine learning [9]. This archetypal dataset establishes the following attack taxonomy: DoS (Denial of Service), R2L (Root to Local), U2R (User to Root), Probe (reconnaissance), and Normal. It contains 41 crafted features over 37 attacks including some features that specifically flag an attack such as `root_shell` or `num_shells`. Unlike an IP address, these engineered features are not inherently evident solely from a base network traffic file and require post analysis and correlation to identify and label.

NSL-KDD [11], marketed as an improvement upon the KDD99 dataset, provided 41 features across 22 different attack groups within the 4 attack types established within KDD99. NSL-KDD eliminated redundant features contained within the KDD99 dataset [11] that could cause bias when training a classifier, the number of selected records from each level is inversely proportional to the percentage of records in the original KDD99 set, and the number of records in the train and test sets are more balanced [12, 13]. This improved dataset spawned another wave of advancement. From 2000 to 2008, more than 50% of 276 peer-reviewed IDS studies used the KDD99 or NSL-KDD dataset, and 85% provided their own datasets gathered data directly from their target environment [14]. Despite documented limitations with artificiality and dated attacks

and protocols, KDD99 and NSL-KDD have provided the foundation for most network intrusion detection research [15, 16, 17].

By 2010 it was apparent that datasets required newer attack techniques to continue to push the state of the art with regards to network intrusion detection models. While the NSL-KDD dataset eliminated redundant data that skewed the original KDD99 dataset, neither dataset contains current attack techniques or methodologies [1] and, due to their simulated nature, they also do not contain real packet headers or payload data. There were several releases of new datasets [3] to meet this demand though the sentiment for additional usable network data perpetuated as late as 2017 [18].

2.2. Post KDD99 and NSL-KDD

Several datasets have been released since KDD99 and NSL-KDD with varying levels of protocol variety, formats, feature engineering, and accessibility [19]. Ideal datasets contain realistic network traffic, labels, include all network interactions, contain a complete capture, preserve privacy, and capture diverse intrusion scenarios [20]. Raw network data produced by private enterprises, like Cisco Systems, Inc., often carries data confidentiality risks where sensitive network implementation details could be inadvertently leaked [21]. Artificial post-capture trace insertions are discouraged. Several datasets [22, 23, 24] provide traffic specific to a class of attack pattern (like SSH attacks or botnet traffic) or over a different medium (e.g., solely 802.11 traffic versus Ethernet traffic), while others attempt to provide a large dataset with a variety of attack patterns [25, 26]. We provide a high level comparison of these datasets in Table 3, and briefly summarize them in the following.

The **CTU-13** dataset [22], published in 2013, comprises thirteen network traffic captures containing botnet traffic. The network consisted of virtualized computers running Windows XP SP2—what the botnet malware could run on at the time—on a Linux Debian host bridged into a university network. The released set of network traffic contains both the traffic from the virtualized computers and the university router, though some of the traffic was removed due to privacy concerns. The authors distinguished botnet traffic as traffic that travels to or from any known infected IP addresses [25].

In 2014 Hofstede et al. [23] refined a detection algorithm for SSH attacks and tested the algorithm on network flows containing the network traffic from almost 100 servers, workstations, and honeypots before releasing the anonymized network traffic in the form of flow data and host log files. This traffic was collected from the University of Twente campus network. Analysis of the dataset was limited to SSH attacks and detections.

Moustafa and Slay [25] also reflected on the lack of quality datasets since KDD99 and NSL-KDD and created the **UNSW-NB15** dataset in 2015. UNSW-NB15 used the IXIA PerfectStorm tool [27] to generate nine families of attacks. The tool simulates a large-scale network and popular web applications. Network traffic was captured using tcpdump, distilled into network flows using Argus [28], and analyzed using Bro (now called Zeek [29]). The IXIA tool generates attacks at a rate of one attack per second for the first 50 GB of the

dataset and increases to 10 attacks per second for the second 50 GB of the dataset. The dataset contains 49 features aggregated from the capture data and Zeek logs with all connection information seen, HTTP requests, and FTP services. Attack labels are contained in a truth table generated by the IXIA tool.

The Aegean Wi-Fi Intrusion Dataset (**AWID**) is a publicly available curated dataset of 802.11 normal and attack traffic, with 15GB of total data containing about 1.6 million normal records and 160 thousand attack records. Kolias et al. [24] used Kali Linux to conduct penetration testing and Wireshark to log traffic. They solely relied on automated attacks and included attack-free traffic in which users conducted normal network operations such as browsing and file sharing.

The **CICIDS2017** dataset [26] includes a variety of attacks such as password brute forcing, a heartbleed exploit, botnet traffic, traffic floods resulting in DoS and distributed DoS, a web server SQL injection, cross site scripting, and an infiltration. These attacks were recorded on a diverse network consisting of Windows and Ubuntu hosts, a firewall, several switches, and using both Windows 8.1 and Kali as attacking nodes. Of note the CICIDS2017 dataset contains redundant features that require removal (i.e., "Fwd Header Length") and rows containing "Infinity" and "NaN" [30].

Despite the call for more available training data of a higher caliber [1, 18], some datasets remain difficult to publicly access or focus on capturing specific attributes like accurate labeling [31] or uncommonly captured protocols like 802.11 [24]. We agree with Ring et al. [19] that the importance of different dataset properties vary on a researcher's use case and some data that is useful for an area of research may not be applicable to another. Thus we follow as closely to the guidelines outlined by Ring et al. [19] to create a strong and reproducible dataset, but recognize that this dataset is not applicable in every scenario. Those guidelines include: up-to-date network-based data and protocols, publicly available, real network traffic, a variety of malicious and normal user behavior, and a meaningful payload [19].

Community Datasets. Several community capture the flag datasets are available, such as DEF CON's Wall of Sheep or CTF datasets [32], that contain both malicious and benign user traffic. User traffic is captured as it is seen on the network, gathered, and presented to the public as free datasets for analysis. These datasets, while large and representative of actual networks and protocols, would require labeling prior to use in a supervised learning environment.

2.3. Data Generation and Testbed Network Infrastructure

Several datasets have been established in recent years including the National Collegiate Cyber Defense Competition [33, 34] which tests a collegiate-level network defenders' ability to repel and defend their infrastructure against a set of professional penetration testers. The simulated enterprise networks they use to monitor the performance of defenders are modelled after networks in use by small businesses using similar services like a web server, mail server, and e-commerce site. With recent advancements in virtualization, using a model

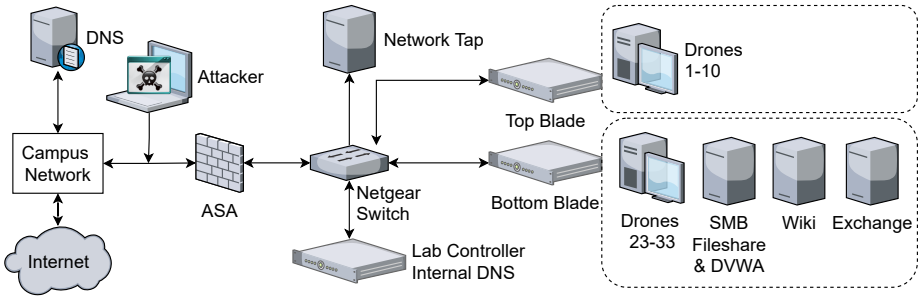


Figure 1: Network Architecture used in CUPID Collection.

network to generate data provides usable data that does not carry the risk of leaking sensitive details nor relies on simulated data that does not contain real packet information. Ring et al. [3] uses a similar approach by virtualizing a small business network through OpenStack[2].

Several testbeds exist that provide network resources, like CPU and disk space, and can generate network traffic but they were not ideal for this approach. PlanetLab [35], for example, is a testbed project from 2003 that is distributed globally over 100 nodes and 42 sites using slices of distributed resources to provide for user needs. The FABRIC testbed [36] and Global Environment for Network Innovations (GENI) [37] also provide experiment infrastructure connecting different national resources. While these resources are available for large cybersecurity experiments requiring physics simulation, we needed a smaller scale infrastructure that we could control with seeded vulnerabilities to leverage for attack traffic and provide access to different human attackers in multiple sessions. Further, the noise included from OS updates is less artificial than merely connecting services and hosts.

3. Methodology

The objective of CUPID is to reflect both human interaction on a network and automated traffic for benign and malicious activities. In this section, we describe the network design and the automated means to generate traffic for the CUPID dataset.

3.1. Network Design

Figure 1 depicts a network diagram of the mock enterprise network we created for collecting CUPID. The data range of workstation drones created network traffic data in addition to the normal communication required to maintain service to the network. The lab controller provided DNS to workstations seeking internal network services. External DNS was required when users were web

[2] `https://www.openstack.org/`

surfing and was provided by the campus network. The campus network was isolated from the CUPID network by a Cisco ASA firewall. The network tap monitored and recorded traffic traversing through the switch.

The network hardware consisted of three servers and a desktop computer that provided endpoints, services, and network monitoring capabilities. The top and bottom blades virtualized all drone workstations. Drones were installed with Windows 7 Service Pack 2. Intentionally vulnerable operating systems were chosen partially as a way to seed exploitable vulnerabilities throughout the network and partially as a reflection of the market share that obsolete Windows versions still maintain [38]. The bottom blade also provided an Exchange server, an SMB fileshare, a Wiki page and a vulnerable web application (DVWA[3]). The last blade served as a lab controller and provided DNS for the internal network. Lastly, the data science tower served as data storage and a network monitoring device.

3.2. Scripted Normal Traffic

Normal data was generated by scripting virtual user actions through PowerShell. Each workstation belongs to a specific mock user where each mock user is assigned a fixed profile: administration, engineering, or business. Each virtual user randomly browsed from a pool of 30 Azure-hosted websites using cURL, accessed email using the Exchange server, read or wrote to an SMB fileshare, or browsed a pool of internal Wiki addresses. Windows 7 comes with PowerShell version 2.0 installed and was updated to version 5.0 to run the automation. Baseline data consists of this profile-generated traffic, inter-endpoint traffic, and traffic required to maintain network services.

3.3. Scripted Malicious Traffic

Malicious data was generated by triggering tools available through the Kali Linux distribution. A preconfigured script was used on each attacking node using command line arguments for existing exploit tools. Additionally, human-generated malicious and benign traffic was gathered, as described in Section 4.

The network tap recorded packet captures which were reviewed and manually validated afterwards. The captures were opened to verify that they were uncorrupted and traffic was analyzed to determine the attack could be seen in the capture, such that the capture could be labeled malicious or benign. To isolate attacks we constructed Wireshark filter rules that separated malicious from benign using their source IPs, converted the rules to Pandas dataframe filters, and used the rules to apply the applicable Label. Malicious traffic was labeled '1' and benign traffic was labeled '0'. Additional details are available in Section 5.

[3] `https://github.com/ethicalhack3r/DVWA`

3.4. Decrypting Encrypted Payloads

As a design decision we valued fully unencrypted payloads as opposed to encrypted data (e.g., HTTPS). A key reason to avoid encryption is that the use of TLS 1.3 changes the structure of the TLS handshake, particularly the `ClientHello` message. When used with HTTP/2, which summarizes HTTP headers to decrease the time to download web pages, the HTTP headers are not human readable and impacts the features available for training [21].

User traffic for web surfing creates substantial TLS traffic, especially by the automated surfing scripts. The set of websites originally used for automated user web surfing consisted of sites that correlated with the user's business function, like an Excel tutorial for a business user, or were pulled from the Cisco Popularity List [39]. Several of these websites had already enabled HTTP/2 for an enhanced user experience, thus obscuring their plaintext web traffic headers and undermining their usefulness for classifier training.

The goal to provide decrypted SSL/TLS traffic to the classifier required several architecture configuration changes. The first step involved using Powershell scripting to negotiate HTTP/1.1 or lower for web browsing. Using the ephemeral Diffie-Hellman encryption keys from each browser and HTTP/1.1 or lower traffic, Wireshark could decrypt the web traffic and web traffic headers could be viewed in plaintext. Wireshark, however, does not support *saving* decrypted packet capture files. Thus, we chose to mirror websites in the cloud and configured each to negotiate TLS 1.2 or lower. A small amount of TLS 1.3 application traffic could not be removed from servers providing system updates. cURL was introduced to replace the browser as a troubleshooting variable thus reducing complexity. Note that although mirroring reduced the impact of TLS 1.3 via artificial means, it also reduced the feature space due to the reduction of third-party advertisers.[4] Now, with most of the web traffic under TLS 1.2, the network was able to support an unencrypted data pipeline where traffic is monitored from a network tap, saved, and distilled by CICFlowMeter [26] into features.

Although the site mirroring technique added artificiality to the dataset, the majority of web traffic in the wild still prefers TLS 1.2 traffic with a growing segment deploying TLS 1.3. Qualys SSL Labs scans 150,000 of the most popular websites on a monthly basis [40]. In January 2022, 99.6% of these websites supported TLS 1.2 while only 51.4% supported TLS 1.3. Hardware and software specifications used in the data generation range are available in Appendix B, and the pipeline of data collection and processing is described further in Section 5.

[4] We noticed that the number of hosts in the dataset was significantly bigger than the direct surfing would imply, due to the number of third-party advertisers attached to each visit to a website.

4. Human-in-the-Loop Traffic

A key feature of CUPID is the involvement of human-guided traffic in the dataset. We recruited ten ethical penetration testers (pentesters) to participate in the creation of CUPID. We captured the normal browsing traffic of these ten pentesters as they conducted the same activities as the scripted users over the span of an hour before capturing malicious traffic over the span of another hour, or when the pentester decided to cease operations (generally if the server was successfully exploited before the time expired). As mentioned before, benign traffic generated this way was labeled with a '0', and human-generated malicious traffic was labeled with a '1' based on the IP of the Kali instance.

We followed best practices to protect the identities of the participants while releasing the data publicly. We conducted semi-structured interviews with the pentesters in October 2019. Appendix A summarizes those interviews. In this section we describe our recruitment process, candidate screening, and limitations of our approach. This study was evaluated and approved by the University of Nebraska Medical Center's Institutional Review Board (IRB).

Recruitment Process. We advertised this research study to the following organizations to recruit participants: a local DEF CON group, the university's cybersecurity student organization, on virtual chatrooms (Slack) known to contain local cybersecurity professionals, to university employees, and via personal contacts. Employees, as they are potentially a vulnerable population, were specifically briefed that their participation was voluntary and refusal to participate would not affect their employment or any benefit to which they were already entitled.

Participant Screening. We asked all pentesters to self-assess their abilities to complete tasks on the network. Specifically, we asked them to rate themselves from 1 to 5 on their ability to interact with services like a normal user and how proficient they were with finding and leveraging web application vulnerabilities. Pentesters were asked to keep notes of their strategy. For example, a strategy could have been using SQL injection (SQLi) to discover usernames and passwords. Pentester responses to the self-evaluation questions and their self-documented strategies are available in Appendix A.

Limitations. We suppose that insider threats need not be seasoned penetration testers to leverage vulnerabilities using public resources, and we sought a variety of skill levels for CUPID. However, pentesters may inflate or deflate the perception of their skill sets [41]. So we cannot be sure of the actual range of abilities included solely based on the information volunteered by the pentesters themselves.

To save time for attack success we opted to use a vulnerable web application (DVWA) that is commonly used to teach web application vulnerabilities. This approach has one major limitation, which is that the vulnerabilities are common and easily exploited. Thus, the time (and network traffic) associated with finding vulnerabilities from scratch is not representative of the time it may take an attacker to conduct reconnaissance to discover flaws.

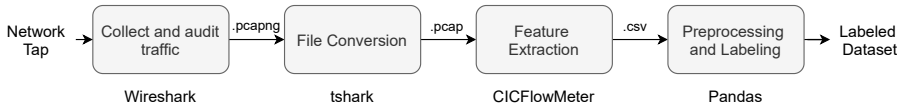


Figure 2: Data processing pipeline.

5. Data Collection

The network tap collected traffic from a mirrored port on the switch. Hardware specifications are available in Appendix B. Wireshark collected network traffic at the switch during all traffic monitoring. Traffic captures were saved in a `.pcapng` format. Figure 2 depicts the network data processing pipeline used during the collection and preprocessing stages of CUPID's creation, which we describe in the remainder of this section.

5.1. Feature Extraction

The output of `tshark` is a capture saved in the `.pcap` file format. CICFlowMeter [26] analyzes packet captures for meta features including interpacket times and bytes between endpoints. The output of CICFlowMeter is in comma separated value (`.csv`) format, which we process in Python3 to feature engineer and label conversations as malicious or benign, as shown in Figure 2.

5.2. Data Preprocessing

The csv output for CICFlowMeter for each packet capture is no longer describing traffic at a packet level but instead at a conversation level. That is, what was a list of packets between nodes is now a data object containing information about a conversation between nodes (i.e., a network flow). The data now requires labeling, where label '1' will be used to indicate malicious traffic and '0' for benign.

Our labeling is based on the host's source IP address as traffic is generated from the host. IP spoofing was not used during the creation of the dataset so we have ground truth of source IP addresses. We used Python Pandas to parse csv files into DataFrames and label rows based on Wireshark rules. An example of such a rule can be seen in 3 where 'x' is column in a DataFrame. Figure 3 shows that, assuming malicious traffic originated from 10.10.10.4, rows corresponding to traffic originating from the malicious IP address are labeled with a '1'. Hence, the rules label the malicious traffic to distinguish it from normal so that ground truth is available within the dataset. The two dataframes—malicious and benign—were concatenated prior to further preprocessing. This extraction, splitting, labeling, and concatenating is repeated over each malicious file until all fields are contained within the same DataFrame.

```
def malicious(x):
    # If traffic was from 10.10.10.4
    # AND the protocol was ICMP, it's malicious
    if DataFrame.loc[DataFrame['pr'] == 1.0]:
        DataFrame['label'] == '1'
```

Figure 3: Rule Conversion

6. Dataset Analysis

In this section we describe and analyze CUPID. We provide a detailed breakdown of one of the four 24-hour normal (baseline) data samples. Summaries of the full set of samples in the dataset are also given with comparison to similar public datasets for network intrusion detection.

Protocol Breakdown. One of the 24-hour baseline data samples was taken on the first day of sampling in April 2019. The raw `042219_1000.pcapng` data file contains 4,346,077 packets requiring approximately 3.3 GB of storage. The entirety of the CUPID dataset comprises approximately 50 GB. Outside of the local address space, the sample contains 179 unique hosts. Most of the packets in this sample were TCP-based (75%) and include protocols like Internet Control Message Protocol (ICMP), Distributed Computing Environment (DCE) / Remote Procedure Call (RPC), Hypertext Transfer Protocol (HTTP), Simple Mail Transfer Protocol (SMTP), Kerberos, Lightweight Directory Access Protocol (LDAP), and Network Basic Input/Output System (NetBIOS) / Server Message Block (SMB). The packet capture contains UDP-based data (7.5%) with Network Time Protocol (NTP) and Domain Name System (DNS). Addressing protocols like ARP and 802.1Q Virtual LAN comprise the remaining traffic. Table 1 provides a breakdown of the most commonly seen protocols in this dataset. The rich variety of protocols in CUPID is due to including services that are representative of enterprise environments like email, DNS lookups, and active directory accesses. Of note, SSL/TLS traffic is present in the capture with 25,341 (0.6%) total packets. The majority of this traffic is in TLS 1.2 or earlier and certificate issuer details can be read in plaintext.

Attack Variety. The sample of attacks available within CUPID was not intended to be exhaustive but representative of stages seen within the cyber kill chain attack strategy. Reconnaissance, for example, is required to understand the network ecosystem. In CUPID reconnaissance is performed using common system utilities like `nmap` and `nslookup` to generate the traffic from an attacker Kali Linux host. Similarly, BoNeSi[5] generates ICMP, UDP and TCP (HTTP) flooding attacks that were used to indicate command and control traffic. Commonly available tools were used to increase reproducibility and reduce implementation complexity. A list of the attacks in CUPID mapped to

[5] https://github.com/Markus-Go/bonesi

Protocol		Pkts (%)	Pkts
TCP	Internet Control Message Protocol (ICMP)	10.48	455,600
	Distributed Computing Environment / Remote Procedure Call (DCE/RPC)	4.00	173,931
	Hypertext Transfer Protocol (HTTP)	1.71	74,576
	Simple Mail Transfer Protocol (SMTP)	0.11	4,937
	Kerberos	0.80	35,151
	Lightweight Directory Access Protocol (LDAP)	8.61	374,467
	NetBIOS Session Service	8.48	368,961
UDP	Network Time Protocol (NTP)	1.79	78,110
	Domain Name System (DNS)	2.32	101,258
Addressing	Address Resolution Protocol (ARP)	6.50	282,632
	802.1Q Virtual LAN	13.92	605,126

Table 1: Protocol breakdown (non-exhaustive) for a single 24-hour baseline sample.

Attacks in CUPID	CKC	KDD99[†]
Webcrawling	Reconnaissance	Normal
Recorded live user interaction	Various	Probe, U2R, R2L
ARP, nmap, Dig, DNSMap, DNSTracer, nslookup	Reconnaissance	Probe
SQLi, Directory Traversal, Password brute forcing	Weaponization	R2L
Delivery of reverse Meterpreter shell	Delivery	R2L
STP, DHCP attacks	Exploitation	DOS
BoNeSi	C2 traffic	DOS

[†]The KDD99 taxonomy includes DoS (Denial of Service), R2L (Root to Local), U2R (User to Root), Probe (reconnaissance), and Normal.

Table 2: Attacks in CUPID Mapped to Cyber Kill Chain (CKC) and KDD99 Taxonomy.

the KDD99 taxonomy is available in Table 2.

Training Example Comparison. We provide the CUPID dataset in its native packet capture form and as extracted features using CICFlowMeter [26]. As shown in Table 3, CUPID provides a comparable number of testing and training examples as similar NIDS datasets.

6.1. Variety

Networks and attacks continually increase in complexity and acceptable datasets for intrusion detection should capture diverse intrusion scenarios [31] on endpoints with different hardware. Here we attempt to quantify this diversity by comparing CUPID against other publicly available datasets, including KDD99, using the destination port occurrence. Well-known ports, or system ports, are ports 0 to ports 1023 and are registered with the Internet Assigned Numbers Authority (IANA). These ports are registered to specific system applications that, by standard, accept data on the port registered. Secure shell

Name (Year)	Entropy	Format	Data (rows)	Features	Human
KDD99 & NSL-KDD (1998)	0.705	custom	148,515	43	N
CTU-13 (2013)	0.718	packet, uni- & bi-directional flow	312,913	84	N
UNSW-NB15 (2015)	1.581	packet, other	2,540,043	157	N
CICIDS2017 (2017)	1.005	packet, bi-directional flow	2,884,588	79	N
CUPID (2019)	1.258	packet	1,464,190	84	Y

Table 3: CUPID comparison to other datasets. Variety metric calculated using entropy of destination port occurrence as discussed in Section 6.1.1. "Custom" indicates a special format specific to KDD99.

(SSH), for example, is registered to port 22 and accepts traffic on that port by default if enabled.

6.1.1. Data Variety through Entropy

We analyzed each dataset for the destination port occurrence and compared them via relative entropy, calculated as

$$H = \Sigma_{i=0}^{65535} n \log\left(\frac{d_i}{n}\right) \tag{1}$$

where d_i is the number of occurrences of destination port i, and n is the sum of all destination port occurrences.

As shown in the Entropy column in Table 3, the KDD99 and CTU-13 datasets are the least entropic, or least random, datasets. The KDD99 dataset was simulated and CTU-13 focused solely on botnet-related attacks. As KDD99 and CTU-13 are also the least entropic, we suggest that entropy may be a measure that indicates the artificiality of a dataset. CICIDS2017 and CUPID are close in value and were both created on network ranges and captured various, more recent, attack techniques. UNSW-NB15 was generated using an enterprise cyber range tool and includes multiple attack techniques which may indicate why it leads the other datasets in this metric.

6.1.2. Destination Port Distribution

Figure 4 shows the frequency of destination ports across the KDD99, UNSW-NB15, CUPID, CTU-13, and CICIDS17 datasets. These plots show the top services based on the destination port where both the TCP and UDP port number are the same (for example, SSH is standard on port 22 regardless if over TCP or UDP) or where the destination port is protocol-specific (e.g., NTP uses UDP over port 123) to reduce the complexity of requiring the protocol name to verify the service. Measured protocol traffic consisting of less than one percent of the total traffic was removed. This simplified analysis is used as

neither the KDD99 nor the CICIDS17 datasets include the protocol name field. Ports 49152-65535 were grouped together to reflect the 'Private' port group designated by IANA. Services are color-coded across datasets.

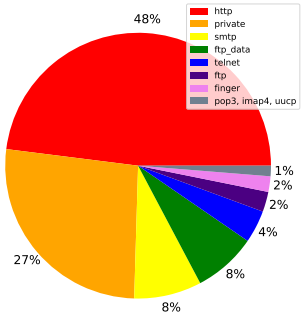
These plots yield insights into the network protocols popular across the datasets. Several protocols are only available in KDD99—the first and oldest dataset—like `telnet` and `imap4`. Common protocols like Hypertext Transfer Protocol (HTTP) and private services were in all the datasets to varying degrees. CUPID stands out by providing more Lightweight Directory Access Protocol (LDAP) and Network Basic In-Out System (NetBIOS) traffic, which is likely due to running active directory.

6.1.3. Human-Generated Pentester Traffic

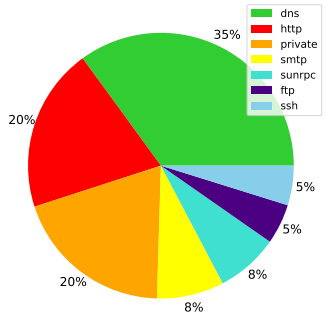
The CUPID dataset provides the network traces of ten individual penetration testers. Each penetration tester was given an hour to engage with network resources and an additional hour to intentionally attack the DVWA instance. Visualizing additional features in the CUPID set from human input shows differences between normal and abnormal behavior. In Figure 5 the number of forward packets per second is shown for each participant's trace. The x-axis shows all the unique flow IDs, and the y-axis shows the traffic volume in each flow. The normal user behavior (light green) reflects user actions as they used resources on the network while abnormal behavior (black) reflects intentional misuse of the DVWA instance. Aside from participant 10, distinctly abnormal clusters appear in the malicious behavior that do not follow the normal user behavior. Participant 10's traffic did not have significant differences between benign and malicious traffic. Participant 10 did not seem to search or utilize network resources differently regardless of acting as a normal user or an attacker. Their self-assessment, available in Appendix A, indicated they knew little about leveraging vulnerabilities and did not complete the task before timeout. The malicious traffic by participant 9 stands out among the samples in both traffic volume and number of flows. In their self-assessment they indicated they used both a port scan and `gobuster`—a brute force tool used to find web application vulnerabilities—against DVWA. Both of these examples seem to indicate that abnormal behavior is visually separable from normal user behavior across participants.

6.2. NIDS Classifier Training

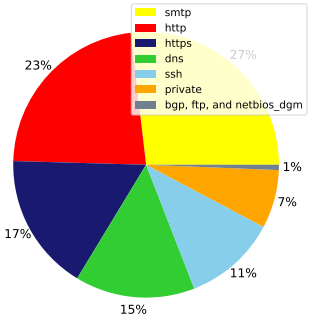
CUPID was compared against CICIDS17 and CTU-13 to determine how classic NIDS classifiers perform when trained against each dataset. Categorical and time-based features were dropped including the Flow ID, Src IP, Dst IP, and Timestamp. Once cleaned, the data was split into training and testing folds. Benign examples out-numbered attack examples at almost 10 to 1. Repeated stratified k-fold was used due to the unbalanced data with 5 splits and 2 repeats resulting in 10 folds. Features were determined based on a variance threshold of 0.25 before training using a standard suite of classifiers commonly found in NIDS research show in Tables 4, 5, and 6. Classifiers were tuned for performance



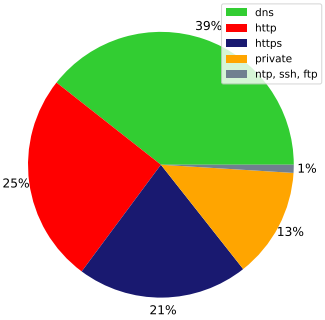
(a) KDD99



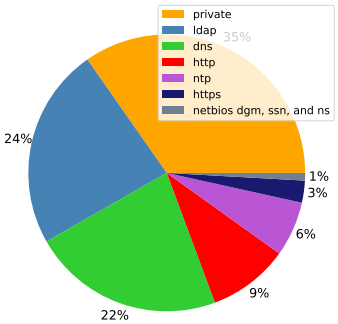
(b) UNSW-NB15



(c) CTU-13

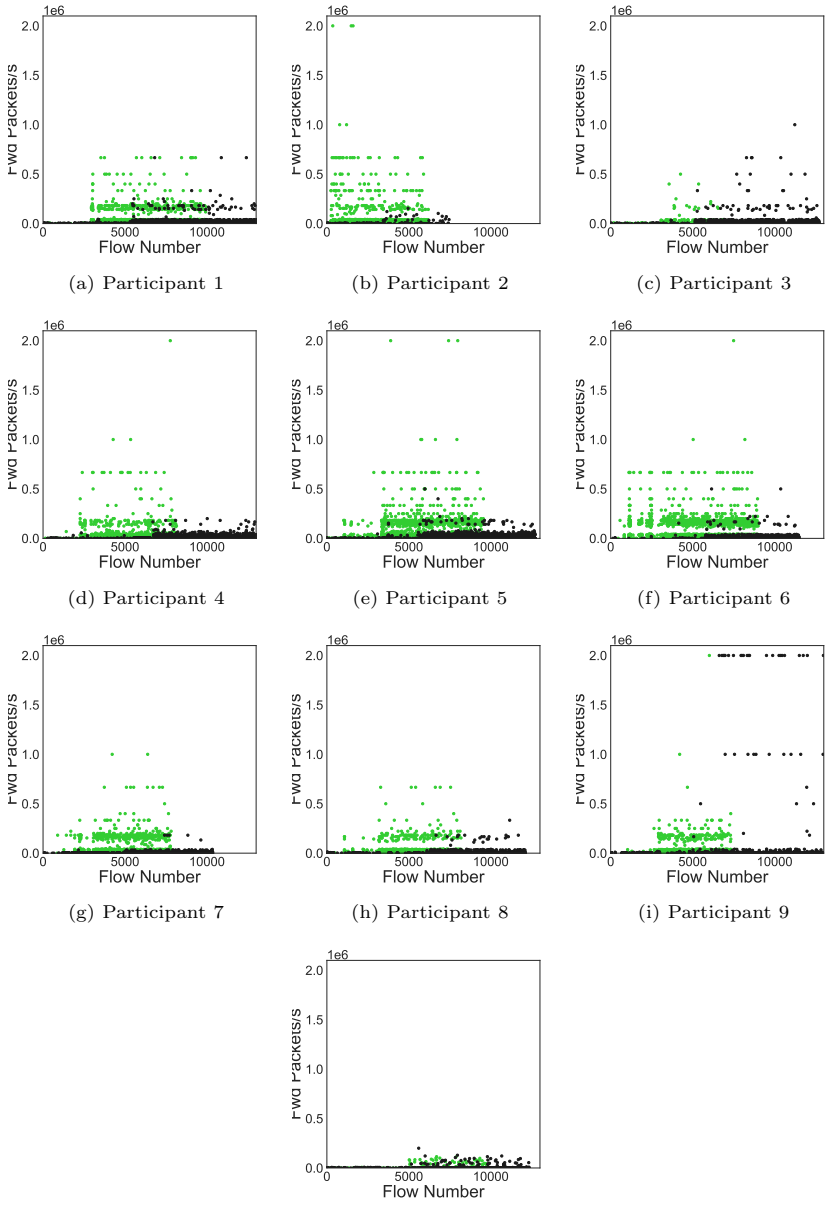


(d) CICIDS17



(e) CUPID

Figure 4: Destination Port Count Across Datasets. Ports 49152-65535 were grouped together to reflect the 'Private' port group designated by IANA. Each legend provides the top protocol based on destination port. The ordering of the protocols in each legend reflects the top percentage sorted in descending order.



(a) Participant 1 (b) Participant 2 (c) Participant 3

(d) Participant 4 (e) Participant 5 (f) Participant 6

(g) Participant 7 (h) Participant 8 (i) Participant 9

(j) Participant 10

Figure 5: Forward packets per second graphed by participant across flows. Black indicates attack traffic while green indicates benign traffic. Flow Number is a specific conversation in the network traffic captured chronologically.

Name	Precision	Recall	Accuracy	F1 Score
LR	0.714246	0.723102	0.700936	0.718643
RF	0.963150	0.783244	0.869682	0.863928
NB	0.561325	0.945076	0.580873	0.704319
SVM	0.713791	0.720782	0.699863	0.717267
KNN	0.902897	0.843573	0.869456	0.872226
MLP	0.878844	0.792951	0.832851	0.833625

Table 4: CTU13 Classifier Training.

Name	Precision	Recall	Accuracy	F1 Score
LR	0.850949	0.823433	0.936863	0.836964
RF	0.995481	0.795245	0.958992	0.884167
NB	0.220144	0.997860	0.303830	0.360707
SVM	0.847798	0.796430	0.931795	0.821312
KNN	0.985311	0.993704	0.995845	0.989490
MLP	0.987823	0.990273	0.995681	0.989039

Table 5: CICIDS Classifier Training

Name	Precision	Recall	Accuracy	F1 Score
LR	0.940299	0.839514	0.975092	0.887051
RF	0.994513	0.804239	0.976676	0.889311
NB	0.189232	0.976009	0.510007	0.317003
SVM	0.944529	0.844213	0.976073	0.891556
KNN	0.977893	0.968190	0.993744	0.973017
MLP	0.983376	0.965093	0.994032	0.974146

Table 6: CUPID Classifier Training

(training speed) and not optimized per dataset. More specifically, the same classifier configuration was used for each dataset.

Classifiers trained with CUPID performed similarly to the other two datasets. Precision, recall, accuracy, and the F1-score varied between 0.6998 and 0.9978 across CUPID, CTU-13, and CICIDS 2017. Classification using Naïve Bayes presents as an outlier in Tables 4, 5, and 6 with significantly poorer results. This outlier shows across all metrics and datasets, and is consistent with prior work that shows Naïve Bayes performs poorly on CTU-13 [42] and CICIDS [43].

7. Discussion

In addition to the conscious design choices we made (such as plaintext packet capture), we discovered several pitfalls for future researchers to consider carefully when constructing a dataset collection study.

Visibility. As data is recorded and transformed, the loss of data fidelity is inevitably sacrificed for reduced storage space and reduced preprocessing time. The conversion of packets into conversational flows can cause loss of useful fields that could aid in detecting attacks which are available only at a packet-level. More specifically, layer 2 attacks are most easily filterable by a hardware address, but hardware addresses are not visible at higher levels of introspection, specifically at the conversational level. We resolved this problem by finding other means to filter malicious traffic that did not rely on the features that had been removed by CICFlowMeter.

Switch Sabotage. Several rounds of rework were required between recording the attack and validating the attack occurred. Layer 2 attacks, particularly the Spanning Tree Protocol attack available from the tool Yersinia, were difficult to validate. After verifying tool functionality and network configurations, we determined that attacks were filtered by the switch. Once the configuration was verified and the attacks re-run, the malicious traffic was visible on the network. Quality assurance inspections on data samples are vital to ensure the intended activities are represented in collected data. Interestingly, if we had used automated means of labeling malicious traffic, we may not have determined that there was a configuration problem at the networking level.

8. Future Work

Workarounds required to obtain unencrypted data increasingly make the data more artificial. Datasets, and the machine learning algorithms that depend on them, should consider the reality that payloads will be encrypted. Future work could investigate means of detecting anomalous traffic without reliance on cleartext.

Additional work could also capture data from networks containing both enterprise and operational technology (OT) network traffic as the vast majority of datasets in this area focus on ISP- or enterprise-level IT traffic. Few datasets are available with OT-specific protocols or hybrid networks containing a business

enterprise network attached to a control system network. However, the rise of the industrial Internet-of-Things (IIoT) means such complex hybrid networks will become increasingly connected and vulnerable [44].

Participation from live pentesters was limited. A virtual testbed or range, or utilizing a capture the flag event, could enable further participation beyond local geographical regions. Future iterations of this research would benefit from further automation when deploying changes to the range that are now available from modern Infrastructure-as-a-Service providers. A larger scope of attacks linked to a framework such as MITRE's ATT&CK framework would aid in applicability to professional security operations center analyst task pipelines.

9. Conclusion

In this paper we have presented the design of a framework used to collect CUPID, which to our knowledge is the first publicly available, labeled network traffic dataset with human pentester activity. The network activities within CUPID are representative of modern enterprise networking. The goal of CUPID is to facilitate further investigation of anomaly-based intrusion detection systems. We have also provided discussion about design choices, their inherent challenges, and how we overcame them. We hope that our lessons learned will help others to create useful intrusion datasets with pentesters in the future.

Acknowledgments

This work is supported in part by NSF grant OAC-2115134, OAC-2001789, NSF grant OAC-1920462, Colorado State Bill 18-086, and by a grant from the Silicon Valley Foundation as an award through the Cisco Research Center. The authors would like to thank Chris Shenefiel, Blake Anderson, and the security professionals that donated their expertise to help make this research a success.

References

[1] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In Proceedings of the 2010 IEEE Symposium on Security and Privacy, SP '10, pages 305–316, Washington, DC, USA, 2010. IEEE Computer Society.

[2] Idan Amit, John Matherly, William Hewlett, Zhi Xu, Yinnon Meshi, and Yigal Weinberger. Machine learning in cyber-security-problems, challenges and data sets. arXiv preprint arXiv:1812.07858, 2018.

[3] Markus Ring, Sarah Wunderlich, Dominik Grüdl, Dieter Landes, and Andreas Hotho. A toolset for intrusion and insider threat detection. In Data Analytics and Decision Support for Cybersecurity, pages 3–31. Springer, 2017.

[4] Blake Anderson and David McGrew. Identifying encrypted malware traffic with contextual flow data. In Proceedings of the 2016 ACM workshop on artificial intelligence and security, pages 35–46. ACM, 2016.

[5] Stefan Axelsson. The base-rate fallacy and the difficulty of intrusion detection. ACM Transactions on Information and System Security (TISSEC), 3(3):186–205, 2000.

[6] Markus Ring, Daniel Schlör, Dieter Landes, and Andreas Hotho. Flow-based network traffic generation using generative adversarial networks. Computers & Security, 82:156–172, 2019.

[7] M. Kacic, D. Ovsonka, P. Hanacek, and M. Barabas. Traffic generator based on behavioral pattern. In The 9th International Conference for Internet Technology and Secured Transactions (ICITST-2014), pages 229–223, Dec 2014.

[8] Colin Gilmore and Jason Haydaman. Anomaly detection and machine learning methods for network intrusion detection: An industrially focused literature review. In Proceedings of the International Conference on Security and Management (SAM), page 292. The Steering Committee of The World Congress in Computer Science, Computer . . . , 2016.

[9] University of California Irvine. Kdd cup data, 1999.

[10] Richard Lippmann, Joshua W. Haines, David J. Fried, Jonathan Korba, and Kumar Das. Analysis and results of the 1999 darpa off-line intrusion detection evaluation. pages 162–182, 10 2000.

[11] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali Ghorbani. A detailed analysis of the kdd cup 99 data set. IEEE Symposium. Computational Intelligence for Security and Defense Applications, CISDA, 2, 07 2009.

[12] S Revathi and A Malathi. A detailed analysis on nsl-kdd dataset using various machine learning techniques for intrusion detection. International Journal of Engineering Research & Technology (IJERT), 2(12):1848–1853, 2013.

[13] Hee-su Chae, Byung-oh Jo, Sang-Hyun Choi, and Twae-kyung Park. Feature selection for intrusion detection using nsl-kdd. Recent advances in computer science, 20132:184–187, 2013.

[14] M. Tavallaee, N. Stakhanova, and A. A. Ghorbani. Toward credible evaluation of anomaly-based intrusion-detection methods. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 40(5):516–524, 2010.

[15] Y. Xin, L. Kong, Z. Liu, Y. Chen, Y. Li, H. Zhu, M. Gao, H. Hou, and C. Wang. Machine learning and deep learning methods for cybersecurity. IEEE Access, 6:35365–35381, 2018.

[16] P. Mishra, V. Varadharajan, U. Tupakula, and E. S. Pilli. A detailed investigation and analysis of using machine learning techniques for intrusion detection. IEEE Communications Surveys Tutorials, 21(1):686–728, 2019.

[17] A. L. Buczak and E. Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. IEEE Communications Surveys Tutorials, 18(2):1153–1176, 2016.

[18] Waqas Haider, Jiankun Hu, Jill Slay, Benjamin P Turnbull, and Yi Xie. Generating realistic intrusion detection system dataset based on fuzzy qualitative modeling. Journal of Network and Computer Applications, 87:185–192, 2017.

[19] Markus Ring, Sarah Wunderlich, Deniz Scheuring, Dieter Landes, and Andreas Hotho. A survey of network-based intrusion detection data sets. Computers & Security, 2019.

[20] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A. Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. Computers & Security, 31(3):357 – 374, 2012.

[21] Blake Anderson and David McGrew. Machine learning for encrypted malware traffic classification: Accounting for noisy labels and non-stationarity. In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '17, pages 1723–1732, New York, NY, USA, 2017. ACM.

[22] Sebastian Garcia, Martin Grill, Jan Stiborek, and Alejandro Zunino. An empirical comparison of botnet detection methods. computers & security, 45:100–123, 2014.

[23] Rick Hofstede, Luuk Hendriks, Anna Sperotto, and Aiko Pras. Ssh compromise detection using netflow/ipfix. ACM SIGCOMM computer communication review, 44(5):20–26, 2014.

[24] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Stefanos Gritzalis. Intrusion detection in 802.11 networks: empirical evaluation of threats and a public dataset. IEEE Communications Surveys & Tutorials, 18(1):184–208, 2015.

[25] Nour Moustafa and Jill Slay. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In 2015 military communications and information systems conference (MilCIS), pages 1–6. IEEE, 2015.

[26] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. 2018.

[27] Keysight. Perfectstorm. https://www.keysight.com/us/en/products/network-test/network-test-hardware/perfectstorm.html, 2020.

[28] LLC. QoSient. Openargus. https://openargus.org/, 2020.

[29] Zeek. Zeek network security monitor. https://github.com/zeek/zeek, 2020.

[30] Gholamreza Farahani. Feature selection based on cross-correlation for the intrusion detection system. Security and Communication Networks, 2020, 2020.

[31] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A Ghorbani. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. computers & security, 31(3):357–374, 2012.

[32] DEF CON. Defcon 8, 10 and 11 ctf datasets. http://cctf.shmoo.com, 2000.

[33] G. B. White, D. Williams, and K. Harrison. The cyberpatriot national high school cyber defense competition. IEEE Security Privacy, 8(5):59–61, Sep. 2010.

[34] Dwayne Williams. About the collegiate cyber defense competition. 2019.

[35] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. Planetlab: An overlay testbed for broad-coverage services. SIGCOMM Comput. Commun. Rev., 33(3):3–12, July 2003.

[36] I. Baldin, A. Nikolich, J. Griffioen, I. I. S. Monga, K. Wang, T. Lehman, and P. Ruth. Fabric: A national-scale programmable experimental network infrastructure. IEEE Internet Computing, 23(6):38–47, 2019.

[37] C. Elliott. Geni - global environment for network innovations. In 2008 33rd IEEE Conference on Local Computer Networks (LCN), pages 8–8, 2008.

[38] Gregg Keizer. Windows by the numbers: Upgraders press pause. 2019.

[39] Cisco Umbrella. Cisco umbrella popularity list. https://s3-us-west-1.amazonaws.com/umbrella-static/index.html, 2016.

[40] Qualys SSL Labs. Ssl pulse. https://www.ssllabs.com/ssl-pulse/, 2021.

[41] Allyson L Holbrook, Melanie C Green, and Jon A Krosnick. Telephone versus face-to-face interviewing of national probability samples with long questionnaires: Comparisons of respondent satisficing and social desirability response bias. Public opinion quarterly, 67(1):79–125, 2003.

[42] Songhui Ryu, Baijian Yang, et al. A comparative study of machine learning algorithms and their ensembles for botnet detection. Journal of Computer and Communications, 6(05):119, 2018.

[43] Deris Stiawan, Mohd Yazid Bin Idris, Alwi M Bamhdi, Rahmat Budiarto, et al. Cicids-2017 dataset feature analysis with information gain for anomaly detection. IEEE Access, 8:132911–132921, 2020.

[44] G. Bloom, B. Alsulami, E. Nwafor, and I. C. Bertolotti. Design patterns for the industrial internet of things. In 2018 14th IEEE International Workshop on Factory Communication Systems (WFCS), pages 1–10, 2018.

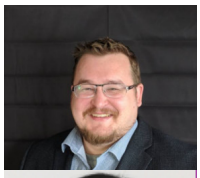
10. Authors

Heather Lawrence is a security data scientist for ThreatKey pursuing a PhD in Security at the University of Colorado, Colorado Springs with a focus on the application of machine learning to intrusion detection. She volunteers with B-Sides Orlando and DEF CON.

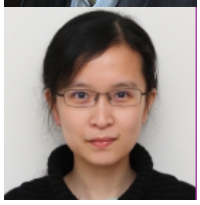
Uchenna Ezeobi received his undergraduate degree in mechanical engineering from University of Minnesota and his masters in computer engineering from university of New Mexico. He is currently working on his Ph.D. in computer science at University of Colorado Colorado Springs. His research interest is in computer systems security, robotics, and application of machine learning to high performance computers

Orly Tauil is an agile IT security engineer and passionate penetration tester with broad expertise in the cybersecurity and technology space.

Jacob Nosal is a security oriented platform and cloud engineer focused on enabling success in cloud native developments teams.



Owen Redwood is a Principal Security Researcher at Research Innovations Inc.



Yanyan Zhuang is currently an Assistant Professor at the University of Colorado, Colorado Springs. Her research interests include networked systems, security and privacy, and software engineering. More information is available on her homepage: `http://www.cs.uccs.edu/~yzhuang/`.

Gedare Bloom received his Ph.D. in computer science from The George Washington University in 2013. He joined the University of Colorado Colorado Springs as an Assistant Professor of Computer Science in 2019. He was an Assistant Professor of Computer Science at Howard University from 2015-2019. His research expertise is computer system security with particular focus on real-time embedded systems. He has published over twenty peer reviewed articles, and served as a program committee member and technical referee for flagship conferences and journals in these areas.

Appendix A. Pentester Interviews

This appendix includes the digitized version of the penetration tester responses. Each section includes the following 5 self-evaluation questions, followed by a high level overview of what each participant was attempting to accomplish.

1. Rate yourself from 1 to 5 on how well you would evaluate yourself at surfing webpages.
2. Rate yourself from 1 to 5 on how well you would evaluate yourself at surfing a wiki.
3. Rate yourself from 1 to 5 on how well you would evaluate yourself at uploading, downloading, modifying, or deleting files from a fileshare.
4. Rate yourself from 1 to 5 on how well you would evaluate yourself at finding web application vulnerabilities.
5. Rate yourself from 1 to 5 on how well you would evaluate yourself at leveraging web application vulnerabilities.

Appendix A.1. P1

Webpage	Wiki	Fileshare	Finding vulns	Using vulns
5	4	5	2	3

- Attempted brute forcing of the password for DVWA
- Attempted SQLi to get usernames and passwords or IDs
- Attempted XSS - got scripts to pop the box on low and medium
- Attempted command injection

Appendix A.2. P2

Webpage	Wiki	Fileshare	Finding vulns	Using vulns
5	5	5	2	3

- Login and set security to Low
- Navigate to command injection page
- Run network layout recon
- Check file permissions for files I can add or edit
- Test adding files to the exec dir
- Use command injection to write a php file for stealing cookies
- Attempt to include page on the php inclusion page #woot!
- Check to see if cookie file was added
- Debug cookie thief file write permissions
- Check contents of php file with more

Appendix A.3. P3

Webpage	Wiki	Fileshare	Finding vulns	Using vulns
5	3	5	5	5

- Logged in
- Recon network
- Setup Burp Intruder
- Established reverse shell
- Escalated privileges to root

Appendix A.4. P4

Webpage	Wiki	Fileshare	Finding vulns	Using vulns
5	5	4	1	1

Attempted to use Hydra to brute force the login

Appendix A.5. P5

Webpage	Wiki	Fileshare	Finding vulns	Using vulns
4	4	3	1	3

- Started the Brute Forcing exercise
- Started the Command Injection exercise. Running '127.0.0.1 && dir' displays the host machine's information.
- Started the File Inclusion exercise.
- Started the SQLi exercise.

Appendix A.6. P6

Webpage	Wiki	Fileshare	Finding vulns	Using vulns
5	5	3	3	3

- Plan to attempt a blind SQLi in order to retrieve the login information for the DVWA
- Decided to use Burp Suite to perform the SQLi
- Configured Burp Suite
- Attempted SQLi

Appendix A.7. P7

Webpage	Wiki	Fileshare	Finding vulns	Using vulns
5	5	5	3	3

- I started my connection to DVWA
- I use the file upload to open a backdoor
- First I want to use weevely. I go to the directory cd /usr/share/weevely
- Run the Python ./weevely.py generate secret my.php. This creates a php script called 'my' with the password of "secret"
- Uploaded this script to DVWA
- Changed the filename in burp from php to jpg to bypass the image filter
- Enabled the back door

- Privileges determined to be nt authority system
- Made myself an account
- Made myself an admin
- Shutdown the box

Appendix A.8. P8

Webpage	Wiki	Fileshare	Finding vulns	Using vulns
5	5	5	5	4

- nmap scan -t4 -f
- file inclusion for etc/passwd
- generate php w/ weevely - password.php
- reverse shell was not working because Kali is NAT'ed behind the host IP
- Weevely works despite being unroutable from the DVWA

Appendix A.9. P9

Webpage	Wiki	Fileshare	Finding vulns	Using vulns
5	5	5	5	2

- Started port scan
- Saw https
- Ran gobuster against dvwa
- Moved to command injection
- Moved to SQLi
- Started Metasploit
- Searched for rdb
- Used the windows/smb/smb delivery Metasploit module
- Attempted command injection
- Crashed dvwa

Appendix A.10. P10

Webpage	Wiki	Fileshare	Finding vulns	Using vulns
5	5	5	1	1

- Login and drop security to low
- Navigate to command injection page
- Perform recon on webapp and server configs
- Attempt to append to webpage js frame to steal cookies
- js files are ro for the www-data user
- Search for more mutable server configs
- Attempt sqli to fetch user tables
- Lack of sql familiarity caused complications until timeout

Appendix B. Hardware & Software Specifications

Network Tap

Name	Internal or Virtual	IP
eno1	Internal	192.168.1.109
enp8s0	Internal	68:05:ca:53:c8:05 (no IP)

Table B.7: Network Tap Interfaces

- Ubuntu 64-bit version 18.04.2 LTS running on an Intel Xeon(R) CPU E5-1607 v4 @3.10Ghz with 32 GB RAM and 1TB hard drive space.

Top Blade

- Windows Server 2016 Datacenter 64-bit running on an Intel Xeon(R) CPU E5-2640 v4 @2.40Ghz with 32 GB RAM and 1TB hard drive space. Drones were provided 1024 MB of RAM, 1 virtual processor, 40GB hard drive space.

Bottom Blade

- Windows Server 2016 Datacenter 64-bit running on an Intel Xeon(R) CPU E5-2640 v4 @2.40Ghz with 32 GB RAM and 1TB hard drive space. Unless otherwise noted by *, VMs were provided 1024 MB of RAM, 1 virtual processor, 40GB hard drive space.

Name	Internal or Virtual	IP
NIC1	Internal	192.168.1.4
vEthernet	Internal	192.168.1.121
DroneVM01	Virtual	192.168.1.31
DroneVM02	Virtual	192.168.1.34
DroneVM03	Virtual	192.168.1.35
DroneVM04	Virtual	192.168.1.36
DroneVM05	Virtual	192.168.1.37
DroneVM06	Virtual	192.168.1.38
DroneVM07	Virtual	192.168.1.43
DroneVM08	Virtual	192.168.1.39
DroneVM09	Virtual	192.168.1.40
DroneVM10	Virtual	192.168.1.42

Table B.8: Top Blade Interfaces

Name	Internal or Virtual	IP
NIC_4	Internal	192.168.1.5
vEthernet	Internal	192.168.1.120
DroneVM23	Virtual	192.168.1.108
DroneVM24	Virtual	192.168.1.45
DroneVM25	Virtual	192.168.1.46
DroneVM26	Virtual	192.168.1.47
DroneVM27	Virtual	192.168.1.48
DroneVM28	Virtual	192.168.1.49
DroneVM29	Virtual	192.168.1.50
DroneVM30	Virtual	192.168.1.51
DroneVM31	Virtual	192.168.1.52
DroneVM32	Virtual	192.168.1.53
DroneVM33	Virtual	192.168.1.54
Exchange*	Virtual	192.168.1.9
Fileshare*	Virtual	192.168.1.11
Wiki*	Virtual	192.168.1.13

Table B.9: Bottom Blade Interfaces

Name	Internal or Virtual	IP
NIC1	Internal	192.168.1.7

Table B.10: Laboratory Controller Interfaces

Laboratory Controller

- Windows Server 2016 Datacenter 64-bit running on an Intel Xeon(R) CPU E5-2630 v4 @2.20Ghz with 32 GB RAM and 9TB hard drive space.

Firewall

- Cisco ASA 5506-X with FirePOWER Services. 750 Mbps stateful inspection throughput. 4GB Memory. 8GB System Flash.

Switch

- Netgear GS724T - 24-Port Gigabit Ethernet Smart Managed Pro Switch with 2 SFP Ports.

Pentesting Laptop

- Host: Windows 10 Home 64-bit running on an Intel(R) Core CPU i7-7700HQ @2.80Ghz with 8 GB RAM.
- VM: Kali Linux Rolling Release 2019.4 with Burp Suite Community Edition x2.1.04 with 2 GB RAM.