The Report Committee for Jeremy Gin
Certifies that this is the approved version of the following report:

Evaluation of Open-Source Intrusion Detection Systems for IPv6 Vulnerabilities in Realistic Test Network

APPROVED BY
SUPERVISING COMMITTEE:

Brian L. Evans, Supervisor

William C. Bard

Evaluation of Open-Source Intrusion Detection Systems for IPv6 Vulnerabilities in Realistic Test Network

by

Jeremy Gin, B.S.

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

May 2017

Dedication

This work is dedicated to Abby Gin, whose ongoing, unwavering support in everything I do breathes life into me, and Ezekiel Gin, whose arrival has only enriched my time in Texas.

Acknowledgements

This research was generously supported by Sandia National Laboratories through the Master's Fellowship Program. I humbly thank my many colleagues from Sandia National Laboratories who provided insight and technical expertise that greatly assisted my research.

I thank Dr. Brian L. Evans, whose holistic mentorship and excellent teaching have profoundly impacted my education, this research, and my time at the University of Texas. His dedication to his students and care for their well-being exceed the expectation of an academic advisor.

I thank Prof. William C. Bard for lending his technical expertise to this project through his formal networking class and his network security experience.

I thank my God, family, and friends for loving and supporting me throughout my life and especially during my studies at the University of Texas.

Abstract

Evaluation of Open-Source Intrusion Detection Systems for IPv6 Vulnerabilities in Realistic Test Network

Jeremy Gin, M.S.E.

The University of Texas at Austin, 2017

Supervisor: Brian L. Evans

Abstract: The Internet Protocol (IP) defines the format by which packets are relayed throughout and across networks. A majority of the Internet today uses Internet Protocol version 4 (IPv4), but due to several key industries, a growing share of the Internet is adopting IPv4's successor, Internet Protocol version 6 (IPv6) for its promise of unique addressability, automatic configuration features, built-in security, and more. Since the invention of the Internet, network security has proven a leading and worthwhile concern. The evolution of the information security field has produced an important solution for network security monitoring: the intrusion detection system (IDS). In this report, I explore the difference in detection effectiveness and resource usage of two network monitoring philosophies, signature-based and behavior-based detection. I test these philosophies, represented by leading edge passive monitors Snort and Bro, against several categories of state-of-the-art IPv6 attacks. I model an IPv6 host-to-host intrusion across the Internet in a virtual test network by including benign background traffic and mimicking adverse network conditions. My results suggest that neither IDS philosophy is superior in all categories and a hybrid of the two, leveraging each's strengths, would best secure a network against leading IPv6 vulnerabilities.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1: Transition From IPv4 To IPv6

The Internet Engineering Task Force first defined IPv4 in Request for Comment (RFC) 791 in 1981. Since then, it has been the de facto protocol of internetworking and the world wide web, not only outlining the format of networked messages sent in packets, but also dictating many of the methodologies by which packets are routed in the decentralized, packet-switched Internet [Postel1981]. The primary role of IPv4 is to provide a standardized addressing scheme which can be used to route a packet from one node on an internetwork to another. In the 7-layer Open Systems Interconnection (OSI) model of the International Standards Organization (ISO), IPv4 exists at Layer-3: the network layer (see Figure 1.1). The OSI model employs the concept of data encapsulation to abstract away a range of roles of internetworking from physical hardware tasks to application and presentation software tasks.

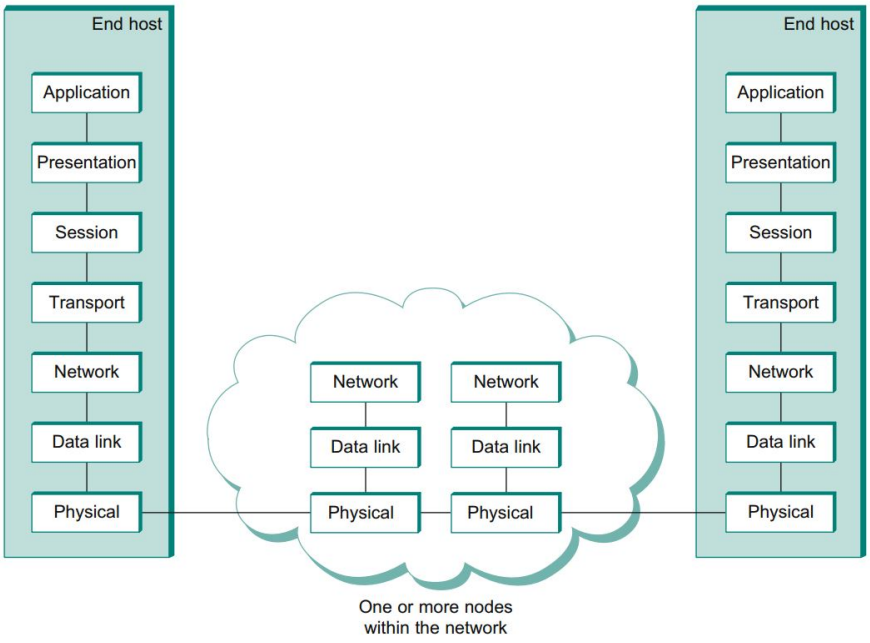

Figure 1.1: OSI 7-layer model from [Peterson2011]

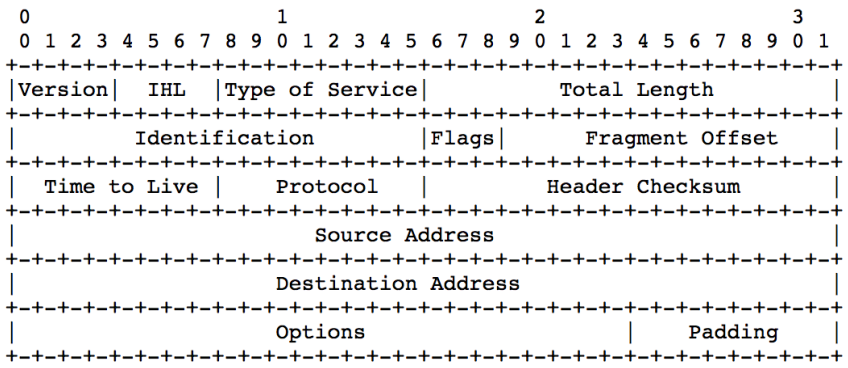

Figure 1.2: Example IPv4 datagram header format from [Postel1981]

According to [Peterson2011], the primary responsibilities of a Layer-3 protocol like IP are: datagram delivery, packet format, and fragmentation and reassembly. This means that each IPv4 packet contains a 20 B IPv4 header and up to 65 KB of data, or encapsulated segments or datagrams from higher layers. The 14 fields in an IPv4 header span basic classifications of packet contents, elementary security checks, and most importantly, source and destination IP addresses (see Figure 1.2). By and large, IPv4 effectively achieves its original purpose of accommodating any future heterogeneous network architecture and technology. However, IPv4 uses a 32-bit addressing scheme (e.g. 192.168.100.101) which limits the address space to 2^{32}, or 4.3E9, unique addresses.

For decades, the inevitable end of IPv4 has been spelled out by a marked lack of unique IP addresses. The past decade has seen an increasing demand for unique IP addresses due to a rise in the Internet-of-Things (IoT), mobile internetworking, social media, and greater global access to Internet. Additionally, organizations like Google, Facebook, Twitter, the Department of Defense, and other government agencies have proactively ramped up research efforts into what must follow IPv4. Even though network engineers have utilized special protocols and network architectures to reuse IP addresses, such as Network Address Translation (NAT), these efforts are temporary fixes to a foundational problem. Mobile internetworking, for example, is fundamentally unsolvable by NAT when users cross NAT boundaries.

The IETF formally introduced IPv6 in 1998 as IPv4's successor in RFC 2460 [Deering1998]. Today, full IPv6 adoption would immediately relieve IPv4's restricted address space, but it also heralds updates in efficiency, complexity, and security to a longstanding standard of the Internet. IPv6 was designed to accommodate automatic configuration, more self-sufficient network administration, and more secure internetworking. IPv6 uses a 128-bit addressing scheme (e.g. 2001:cdba:0000:0000:0000:0000:dead:beef or 2001:cdba::dead:beef) which expands the address space to 2^{128}, or 3.4E38, unique addresses. Having such an astronomically large number of available addresses not only allows for unique addressability of every Internet-connected device, but it also leaves the address space sparsely populated and therefore more resistant to scanning. Under the assumption that worldwide users of Google are an accurate representation of Internet users, Google estimates that current IPv6 adoption rates are approximately 14-16% of the Internet (see Figure 1.3) and estimates that North American and Western European users experience minimal problems connecting to IPv6-enabled networks due to widespread deployment (see Figure 1.4).

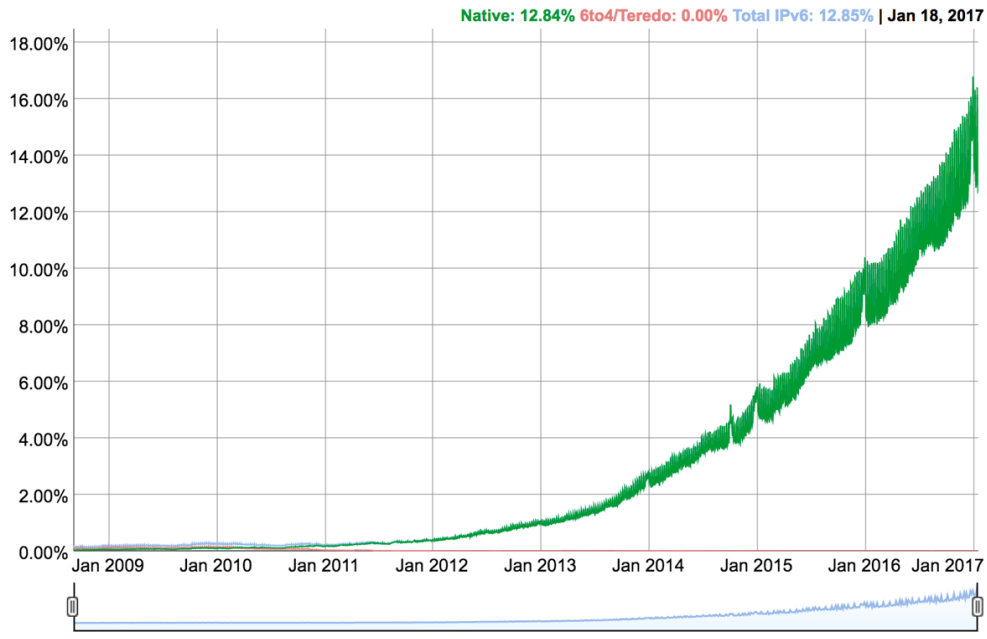

Figure 1.3: Percentage of users who access Google over IPv6 from [GoogleIPv6]

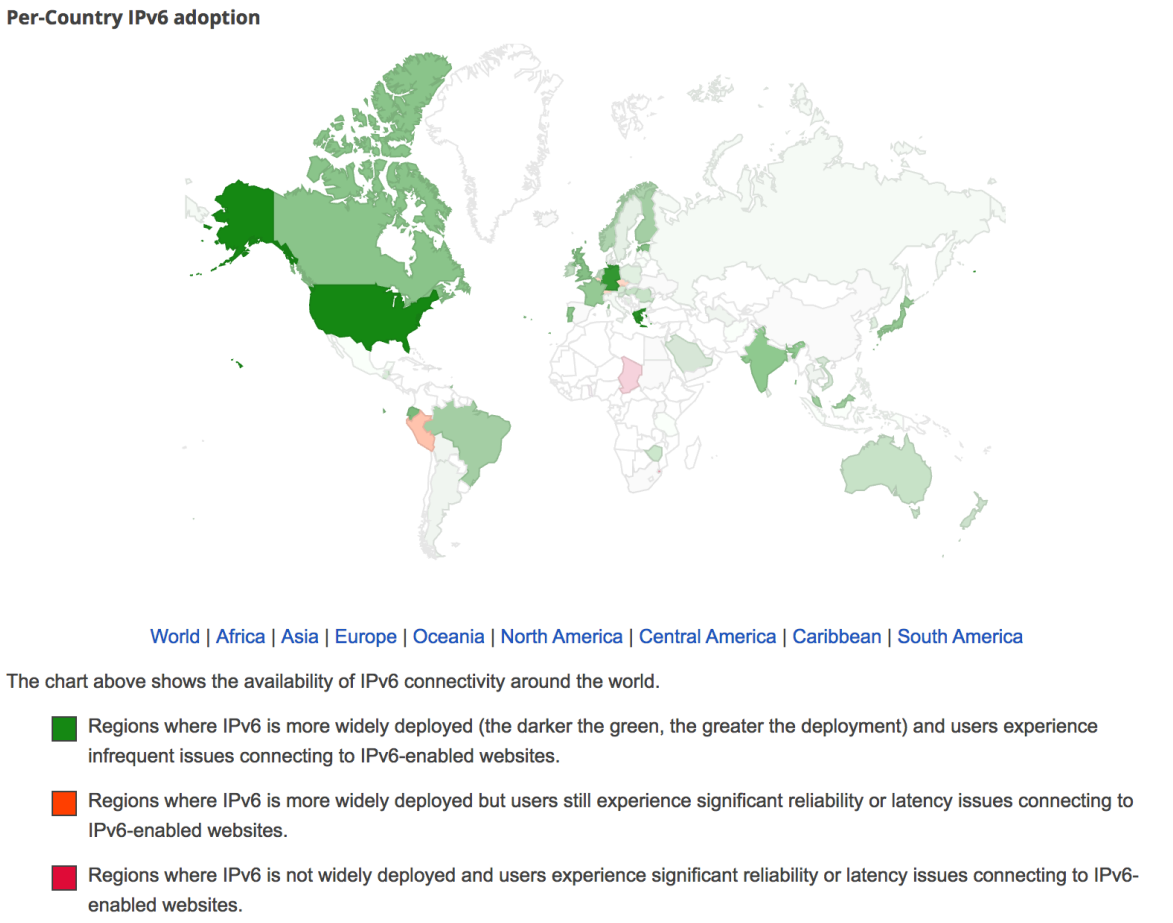

Figure 1.4: Worldwide IPv6 adoption per-country from [GoogleIPv6]

1.2: IPv6 Features

While the primary motivation for IPv6 adoption is an increased address space, early use of the Internet has greatly influenced its design. Major IPv6 design paradigms include [Werny2014]:

- End-to-end addressability: Devices should be able to specifically address devices regardless of network architecture.
- Large scale use: IoT will accommodate tens to hundreds of networked devices per household (e.g. refrigerators, thermostats, mobile phones, glasses, etc.).

- Automatic network and device configurations: Offload configuration from user to equipment. IoT devices should configure themselves on a home Wi-Fi network, and mobile phones should configure themselves as they travel across networks.
- Assumption of low device complexity: Small, single-purpose sensors and devices cannot be expected to handle heavy traffic or complex calculations
- Authentication and encryption: More effective guarantees that traffic is verifiably secure and private

The protocol seeks to achieve these goals by eliminating IPv4 artifacts like Address Resolution Protocol (ARP), NAT, Internet Control Message Protocol (ICMP), and Dynamic Host Control Protocol (DHCP) and instead implementing Stateless Address Auto-configuration (SLAAC), ICMP version 6 (ICMPv6), DHCP version 6 (DHCPv6), Internet Protocol Security (IPSec), extension headers, and more.

1.2.1: ICMPv6

ICMP is used for important utilities like 'ping' and 'traceroute' to test end-to-end connectivity statistics and gain better understanding of internetworking from a source to destination. Internal to a network, ICMP is used as an administrative tool to advertise and discover hosts and routers throughout the network. ICMPv6 supports new functionality for IPv6, but is fundamentally based on the original ICMP. Similar to the original ICMP packet format, ICMPv6 packets include a 1 B type field, a 1 B code field, and variable length payload field. ICMPv6 packets are either error messages or informational messages and different combinations of types and codes provide the following major features [Hogg2008]:

- "Neighbor Discovery Protocol (NDP), Neighbor Advertisements (NA), and Neighbor Solicitations (NS) provide the IPv6 equivalent of the IPv4 Address Resolution Protocol (ARP) functionality.
- Router Advertisements (RA) and Router Solicitations (RS) help nodes determine information about their LAN, such as the network prefix, the default gateway, and other information that can help them communicate.
- Echo Request and Echo Reply support the Ping6 utility.
- PMTUD determines the proper MTU size for communications.
- Multicast Listener Discovery (MLD) provides IGMP-like functionality for communicating IP multicast joins and leaves.
- Multicast Router Discovery (MRD) discovers multicast routers.
- Node Information Query (NIQ) shares information about nodes between nodes.
- Secure Neighbor Discovery (SEND) helps secure communications between neighbors.
- Mobile IPv6 is used for mobile communications."

1.2.2: IPsec

Internet Protocol Security (IPsec) is a framework of RFC standards which, when supported by a network's infrastructure, offers such guarantees as authentication, confidentiality, and encryption for narrow and wide communication channels (range: from one TCP connection to all traffic) [Peterson2007]. Some of the major protocols whose use IPSec standardizes are Authentication Headers (AH), Encapsulating Security Payload (ESP), and Internet Security Association and Key Management Protocol (ISAKMP). IPSec is modular in nature and allows users many options in the kind of services they implement for tasks such as end-to-end encryption or tunneling.

1.3: CYBERSECURITY AND IPV6

Cybersecurity has emerged as an engineering grand challenge of the 21st century. Cyberattacks level the playing field between adversaries who are unequally matched in traditional assets, like manpower, weaponry, and experience. It is a relatively new battlefield whose implications, technologies, and laws have not been concretely formed and whose limitations have not been fully tested. Its non-physical confrontations and exploits are popular among a range of criminals and governments of varying capabilities and intentions. All of these characteristics make the physical and financial consequences of cybercrime and cyberwarfare quite dangerous. According to data compiled by [IC32016], the monetary damage caused by cyber crime reported to the Internet Crime Complaint Center (IC3) has steadily increased from $17.8M in 2001 to $1.07B in 2015 (See Figure 5). On a practical level, cybersecurity research and countermeasures guard against threats to supervisory control and data acquisition (SCADA) systems, critical infrastructure, and commercial systems and machinery. These systems support necessary life-sustaining technologies, industries, and utilities.

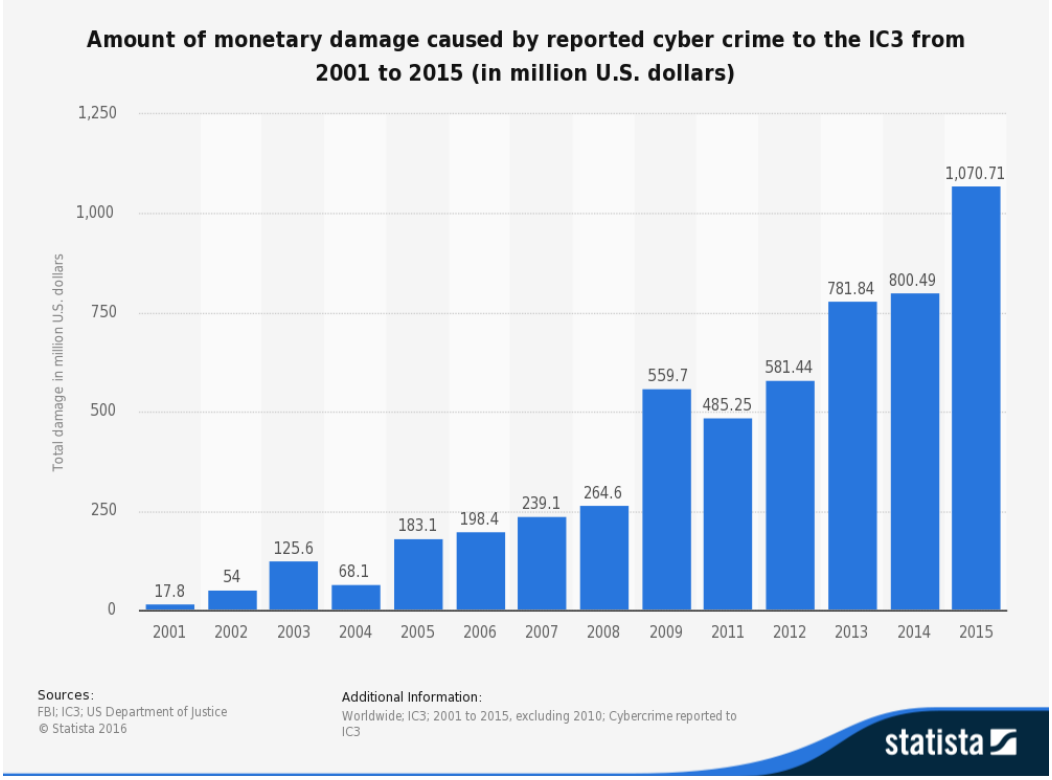

Figure 1.5: Monetary damage of cyber crime from 2001-2015 from [IC32016]

Recent high-profile cyberattacks span the various classes of exploits and caused a wide array of long-lasting consequences. In July 2015, attackers stole the personal records and intimate details contained in the United States security clearance applications of 22 million people from the databases of the Office of Personnel Management [Levine2015]. In October 2016, attackers launched a distributed denial-of-service (DDoS) attack on major Domain Name Service (DNS) host, Dyn, which resulted in regional outages of popular websites like Netflix, Twitter, and Spotify [O'Brien2016]. Most recently, in the election season of 2016, adversaries leaked politically embarrassing content from the servers of the Democratic National Committee in an alleged effort to influence the presidential election.

IPv6 plays a fundamental role in cybersecurity as its protocols and standards represent a variety of attack vectors for a potential attacker in the Layer-3 operations of

future networks. As an emerging technology, its vulnerabilities are not well-researched and seasoned security personnel are not experienced in its defense. In the past, lacking the skills and experience to secure IPv6 on their networks, many network administrators either disabled all IPv6 communication or ignored it [Warfield2003]. Warfield goes on to say, "administrative personnel tasked with defending IPv4 networks have not kept pace with the growth of IPv6". Perhaps, blocking or ignoring IPv6 would suffice 10 years ago, but not today, and definitely not in the future. [Gont2014] warns that unilaterally blocking IPv6 using Layer-2 devices "might create problems that are difficult to diagnose, inclusive of intentional or incidental use of link-local addressing", citing other RFCs for DNS-reliant services. As IPv6 adoption rates rise worldwide, network defenders will see a greater benign-to-malicious ratio IPv6 traffic, and disabling a widely used protocol renders a service or website unreachable for a significant portion of users. [WorldIPv6] keeps a running list of the thousands of websites (including Google, Facebook, YouTube, Yahoo!), tens of network operators, and handful of network equipment vendors who have permanently enabled IPv6 by default on their services, networks, and products.

IPv6 deployment faces substantial security problems. Like any new technology, the security research community requires time and experience to understand new attack vectors and early adopters are discouraged by a lack of existing security best practices. Attack methodologies unique to IPv6 exist in: reconnaissance, network scanning, remote DoS attacks, fragmentation, and abuse of IPv6 extension headers [Werny2014]. As internetworks slowly transition from IPv4 to IPv6, networks must transition through several stages of IP deployment, for example: native IPv4, IPv6-over-IPv4 tunneling, IPv4-over-IPv6 tunneling, dual stack (IPv4 and IPv6 enabled), and IPv6 native. This presents unique security challenges at every stage. This means that even early adopters of IPv6 must continue to defend against IPv4 intrusions in addition to implementation

complications that limit connectivity to the backbone, threaten critical services currently running IPv4, and more. The malicious acts demonstrated in this report represent an IPv6 native environment but can also be used in a dual stack network, as the virtual test network in this report is a dual stack network.

While IPsec promises security improvements over traditional IPv4 networks and its support is mandatory, its implementation relies on other infrastructure such as public-key management systems, and therefore its usage is not required [Caicedo2009]. Like IPv4 attacks, IPv6 attacks will begin with scripted network scanning or mapping for basic and advanced reconnaissance of available hosts, ports, and services even though the IPv6 address space is sparsely allocated. Such scanning programs can utilize IPv6's unique multi-cast addressing which allows for easy communication with all hosts on a LAN [Pilihanto2011]. After scanning a network, an adversary may take advantage of SLAAC, DHCPv6, and NDP to announce phony routers or hosts on the network or launch man-in-the-middle (MITM) attacks ultimately convincing other nodes of its authenticity. Once valuable targets (hosts or routers) are identified, studied and maybe even inadvertently communicating with an adversary, he might launch a DoS attack from locations internal and/or external to the network using ICMPv6 or any combination of protocols, services and ports. Alternatively, quieter attackers may opt to transfer malware, abuse IPv6 extension headers, fragment large packets, or overflow buffers to take down a victim.

Even still, IPv6 will one day subvert IPv4 as the Layer-3 protocol of the world. On World IPv6 Day, June 8th 2011, major Internet content providers enabled IPv6 in their servers and since then, the Internet has sustained and increased the level of IPv6 traffic specifically content-based traffic like Hypertext Transport Protocol (HTTP), rather than

control traffic like ICMPv6 [Sarrar2012]. This suggests that not only did the providers not experience fatal problems, but IPv6's use is approaching that of IPv4.

1.4: Intrusion Detection

In the world of intrusion detection systems (IDSs), there are two main approaches: an network intrusion detection system (NIDS) and a host-based intrusion detection systems (HIDS). For a network administrator, a NIDS is more practically relevant and implementable and is the subject area of this report. The two represent different philosophical approaches to the problem of network exploitation. The basic premise of a HIDS is to run software on all or a selection of hosts in a network searching for signs of attacks on the host. A NIDS runs on one or multiple hosts or servers at different locations in a network passing a selection or all observed network traffic through a variety of algorithms to identify potentially malicious traffic. NIDSs play a crucial role in the cyber arms race between white hat operators and adversaries whether in enterprise networks or government networks. The two most prevalent, open-source IDSs are Snort and Bro.

1.5: Research Questions

This work is an effort to provide data by which to evaluate various aspects of network security approaches in light of the development and adoption of the IPv6 protocol. The following research questions guided my work:

(1) For current state-of-the-art IPv6 attacks, which IDS philosophy is more effective in detecting attempts to compromise a network: signature-based detection or behavior-based detection?

(2) As network conditions deteriorate and routing becomes less reliable or as traffic load overwhelms each IDS, which IDS approach maintains better performance?

(3) Does the more resource-intensive reassembly operations of a behavior-based detection system ultimately render it inferior when it is overwhelmed?

The answers to all of these questions must inform real-world network administrators anticipating a future of IPv6 adoption in sensitive networks. The cost of lapses in cybersecurity are too great to ignore the existence of IPv6 and the real attack surface the protocol presents to adversaries and penetration testers.

1.6: THIS WORK

The research described in this report provides a detailed comparison of the different paradigms for writing and executing IDS algorithms for IPv6 vulnerabilities. It details the testing of the two most popular, current, state-of-the-art, and open-source IDS engines: Snort [Snort2016] and Bro [Bro2016]. Both systems are widely used in enterprise networks, and each has its advantages. My experimentation leverages a state-of-the-art, open-source IPv6 attack suite called 'The Hacker's Choice – IPv6' [Hauser2017]. This work attempts to evaluate the effectiveness of different IDS paradigms in a realistic test network modeled in a virtual laboratory setting. Multiple baseline measurements are taken to ensure that all network components function as described.

To simulate realistic network conditions, I use a combination of assumptions and open-source tools to establish a reasonable confidence in my experimentation. I create many streams of benign traffic in various protocols and processes that are irrelevant to the attack event using Ostinato [Ostinato2016]. At the network interfaces of all hosts emitting packets on the test network, I artificially alter packet transmit rates, drop rates, latencies, and accuracies using a network emulator tool, NetEm [Hemminger2005]. Using multiple scripts running on the network administrator and on each host in the network, I create thousands of trials varying slightly different conditions. In doing so, I hope to build a more holistic picture of real-life attacks in a laboratory setting. Using

additional scripts, I test which alerts, if any, Bro and Snort return on the packet captures and aggregate the detection results into an analyzable form.

Chapter 2 discusses the current state of intrusion detection and specifically IPv6 security research. Chapter 3 gives a more detailed view of the experimental setup I used for this project, specifically various design choices in my virtual test network. Chapter 4 details the generation and processing of the traffic capture data to model realistic data for analysis. In Chapter 5, I provide the results of the experiment through a number visualized metrics and analysis of the data. As part of the analysis, I note trends in the data, present theories to explain trends, and identify sources of error. I conclude the findings of my research and suggest future work in Chapter 6.

Chapter 2: Related Work

2.1: IDS Approaches

In 1987, [Denning1987] proposed the foundational model for detection of anomalous audit records. Reasons that remain valid in modern systems motivate this model of handling malicious activity in a system: fully securing systems is technically and/or economically unrealistic, upgrading to more secure systems is economically prohibitive and/or sacrifices functionality, and no system is safe from the insider threat. Denning's model uses event counters and statistical models to characterize a normal usage model. It provides a basis for detecting break-ins, viruses, Trojans, and leakage while remaining naïve of a target system's mechanisms or deficiencies. Since [Denning1987], industry has adopted much of the sentiment for network security. At one-time, flow-based intrusion detection (FID) was state-of-the-art in the network security field [Sperotto2010]. This method uses flow information of a network trace (i.e. source IP, destination IP, source port, destination port) to identify malicious flows and alert a network administrator. This methodology is less computationally intensive than deep packet inspection (DPI) which involves using protocol parsers and authorized decryption to read all header information and even packet contents. Given the modern-day skill level of adversaries and security experts alike, FID seems more like a superficial catch-all for any malicious activity that might be identified by its unique flow pattern. To be clear, FID is sufficient for a wide range of attacks, however, DPI is now more computationally plausible with efficiently written software, advances in computing power, and more resources devoted to solid cybersecurity. DPI can be considered an extension of FID capability because DPI can entire packet headers and payloads.

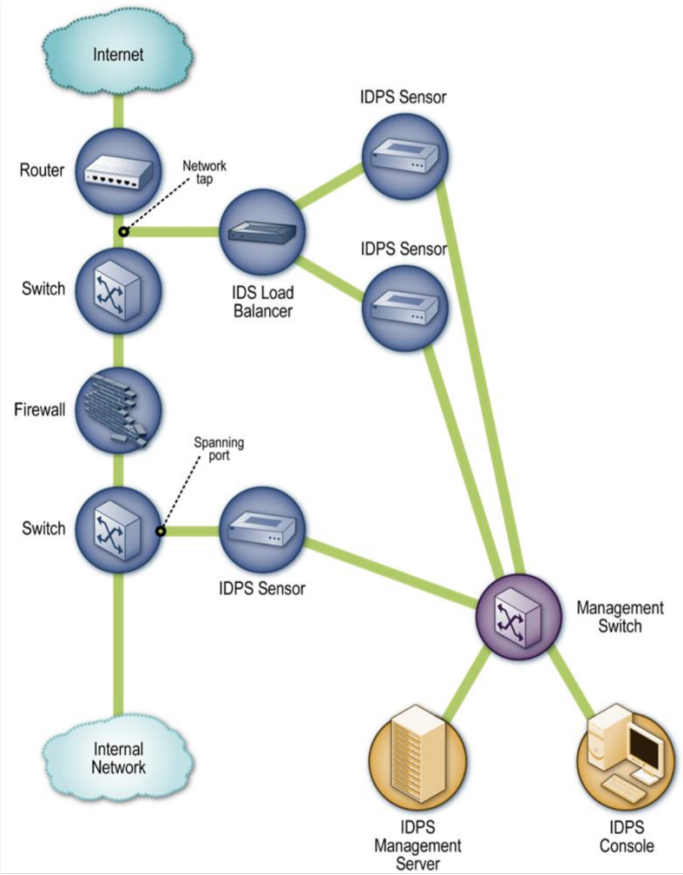

Figure 2.1: Example of NIDS architecture from [Scarfone2012]

The fact that an IDS does not actively accept or reject packets is a fundamental flaw of strictly IDS approaches. State-of-the-art IDSs operate for passive monitoring systems, which branch all or a portion of traffic from a network tap or a switch's switched port analysis (SPAN) port for offline, real-time analysis (see Figure 2.1). A firewall, on the other hand, exists inline of inbound and outbound network traffic and accepts or rejects packets based on superficial information such as program author, certificates, type of service, or specific port. This disadvantage exists by design. IDSs are capable of piecing together longer connections and can build behavioral profiles by which to assess traffic rather than make face value decisions which carry potentially costly consequences for mistakes. The types of intrusions NIDSs most are often best

prepared to detect include: reconnaissance and attacks in the application layer (DHCP, DNS, HTTP, FTP, etc.), transport layer (TCP and UDP) and network layer (IPv4, IPv6, ICMP, and IGMP), unexpected application services (tunneling, back doors, etc.), and policy violations (blacklisted Websites, protocols, ports, etc.) [Scarfone2012]. Some efforts have been made to enable IDSs to prevent intrusions, using a hybrid firewall-IDS architecture, perhaps inline of network traffic, sometimes called an Intrusion Detection and Prevention System (IDPS) (see Figure 2.2) [Scarfone2012].

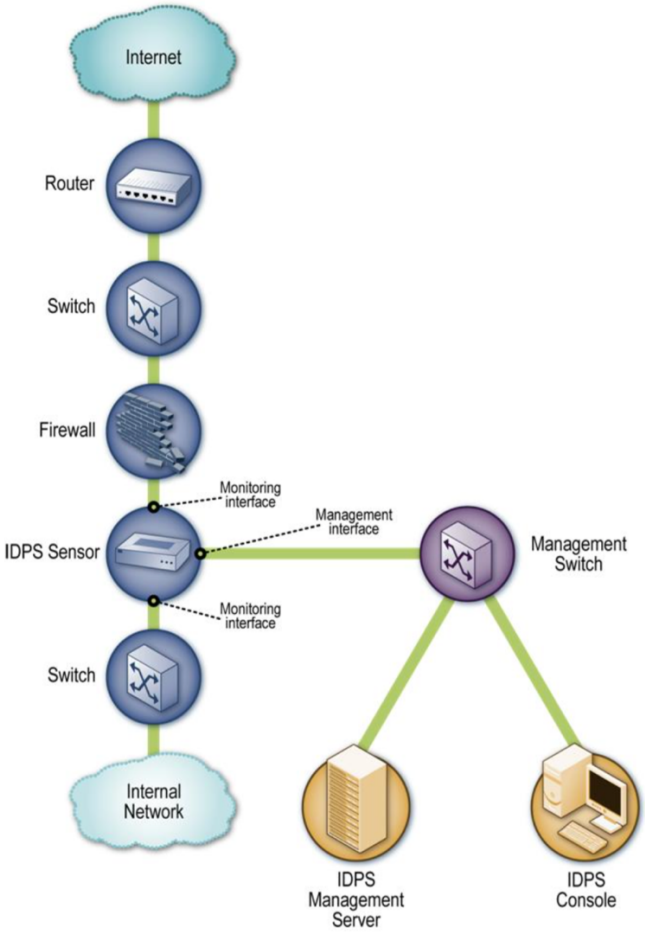

Figure 2.2: Example of inline NIDS/IDPS architecture from [Scarfone2012]

Realistic network traffic volumes often overload IDS sensors with billions of packets from thousands of hosts on the monitored network. IDSs must employ certain methods to build the most complete picture of the network and its connections with the fewest packets necessary. The rest of the packets are either purposefully discarded as unnecessary or inadvertently discarded as resources are maximized. IDSs can employ best effort approaches to discard a majority of packets in high volume connections only making decisions with the first packets, thereby increasing the detection accuracy under high load [Papadogiannakis2010]. To be clear, IDSs are not perfect solutions and attempt to only serve a subset of roles which ultimately comprise comprehensive iron clad network security. There is plenty of argument for the insufficiencies of IDSs, fundamental flaws, and difficulty of testing network security algorithms in a laboratory setting [Ptacek1998][Gates2008]. These basically come down to the heterogeneity and evolution of network contents and architectures complicating the task of modeling a network or understanding how malicious a traffic specimen is. Additionally, IDSs can detect attacks like DoS but prevention requires more than passive monitoring. Some of the genius of IDSs is definitively hard to model in a laboratory setting: the ability to continually build behavioral profiles of diverse, changing networks.

2.2 STATE-OF-THE-ART IDS

Today's state-of-the-art, open-source IDS solutions on the market include: Bro, Snort, and Suricata. These network security monitors passively monitor raw network traffic from a tap on network media hardware. With similar goals, these monitors represent vastly different approaches in IDSs: namely, the tradeoffs between behavior-based detection and signature-based detection. All of them alert network operators when a suspicious item is flagged or a reportable event is noticed. In general, some tradeoffs characterize the behavior-based vs. signature-based comparison (see Table 2.1). In this report, "behavior-based detection" is used synonymously with "anomaly detection".

Characteristic	Behavior-based	Signature-based
Triggers on	Statistical anomalies	Byte/instruction sequences
Relies on data	Representing unique behavioral profiles of a system	From large community databases of signatures
Similar paradigm to	Machine learning	Anti-virus software
Detects	Zero day and day after exploits	Day after exploits
Detection of Unknown Threats	Possible	Impossible
Detection of Known Threats	Good	Good
False Positive Rate	High	Low
False Negative Rate	High	Low
Computational Efficiency	Can be complex	Simple

Table 2.1: High-level comparison chart of detection philosophies

2.2.1: Snort

In 1998, Martin Roesch created today's most widely deployed NIDS, Snort, as a weekend project [Snort2016]. It is now owned by Cisco Systems. Snort is a `libpcap`-based packet sniffer and logger that can detect intrusions by analyzing network traffic and its protocols [tcpdump2017]. Raw network traffic passes through a series of program layers: packet decoder, preprocessors, detection engine, and logging/alerting subsystem (see Figure 2.3).

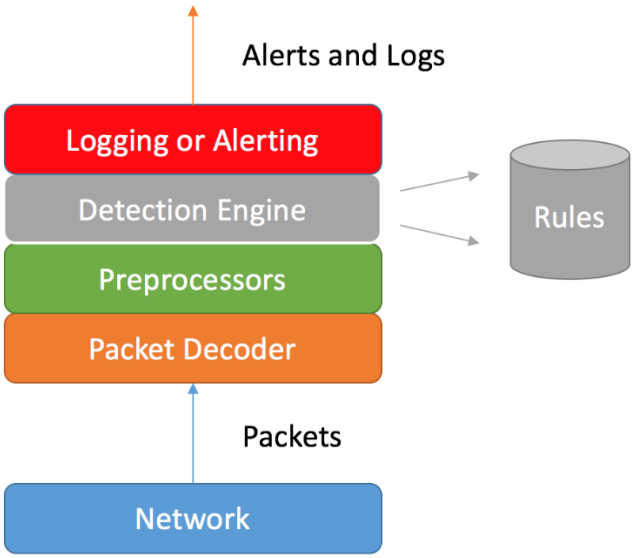

Figure 2.3: High-level Snort architecture

The packet decoder quickly sets pointers throughout each packet at various markers ascending up the OSI model from data link layer to the application layer [Roesch1999]. In the preprocessor layer, the decoded packets are funneled into all preprocessors which are categorized by protocol such as Simple Mail Transfer Protocol (SMTP), Post Office Protocol (POP), or Internet Message Access Protocol (IMAP). By design, users can extend Snort by writing their own preprocessors to augment the modular hierarchy [Snort2016]. Some preprocessors identify suspicious activity in packets and are responsible for the small amount of behavior-based alerts Snort produces. The rest of the preprocessors modify packets in preparation for input into the detection engine, normalizing for distracting traffic patterns. All preprocessors process all packets to detect even those packets which might require more than one preprocessor [Koziol2003]. After packets are preprocessed, the detection engine recursively checks through a user-extendible list of default rules triggering on specific signatures or packet attributes. Because a rule must fully describe an exploit's signature before Snort can detect an exploit, Snort's detection is considered day after detection, as opposed to zero

day detection. In what seems to be a best case scenario, [Roesch1999] states that effective Snort rules can be written hours after an exploit is discovered, and in fact, [Snort2016] offers a subscription service which provides frequent rule updates 30 days before granting access to the public. Finally, once the detection engine triggers on a suspicious signature, a logging or alerting subsystem chosen at run-time issues and/or logs the alert to the specified media for human review.

2.2.2: Bro

Researcher Vern Paxson first created Bro in 1999 at Lawrence Berkeley National Laboratory [Paxson1999]. It has the theoretical capabilities of a low-level signature-based IDS with extensive additional capabilities in behavioral modeling and anomaly detection of network traffic and can run on any UNIX-style system. Bro algorithms, or Bro scripts, are written in an event-driven, scripting language and are run on a Unix-based network monitor written in C++. The language is designed for network-specific applications containing variables that are designed to express network fundamentals/primitives like subnets or IP addresses. Bro's flexibility allows it to run on a specific packet capture (PCAP) or passively monitor a network. In 2012, Bro was updated for full IPv6 compatibility by default, including full IPv6 protocol parsing and various variable and function accommodations in the scripting language.

Bro employs a high-level structure consisting of an event engine and a policy script interpreter (see Figure 2.4). The event engine translates a packet stream into policy-neutral "higher-level network events" to which the interpreter applies Bro's policy scripts [Bro2016]. Not all network packets are processed by the entire hierarchy depicted, in fact, the goal of the structure is to exponentially reduce the information handled as each increasing layer is employed. The popular Linux program `libpcap` is used to capture

raw streams off the network to be filtered in a common file format and input into the event engine [tcpdump2017]. The event engine checks IP checksums and basic header information before invoking individual handlers for TCP and UDP payloads. Depending on its analysis, Bro will retain the header, entire packet, or nothing while tagging certain events based on its observations. The interpreter then invokes event handlers for triggered events. The Bro package provides many event handlers by default, but the highly extensible domain-specific variables and computations allowed in the Bro scripting language allows for arbitrary site-specific policies. Event handlers can even execute third-party programs that can help bridge the gap between intrusion detection and intrusion prevention. One of Bro's design goals is describing network events in a neutral manner. To this end, Bro logs a multitude of information for potential review, such as number, length and nature of all connections, run-time statistics, unusual events, etc. Bro writes all connections and alerts to their own compressed ASCII logs where they can be ingested by a sorting and visualization program stack and ultimately reviewed.

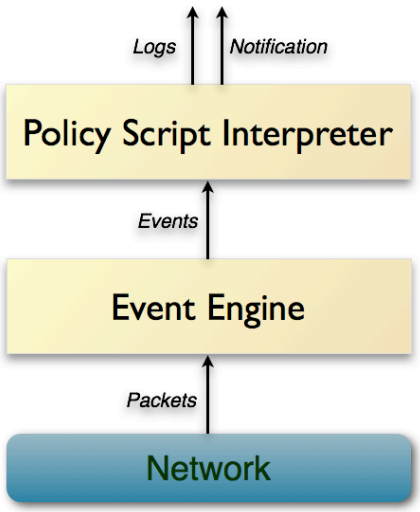

Figure 2.4: Bro architecture from [Bro2016]

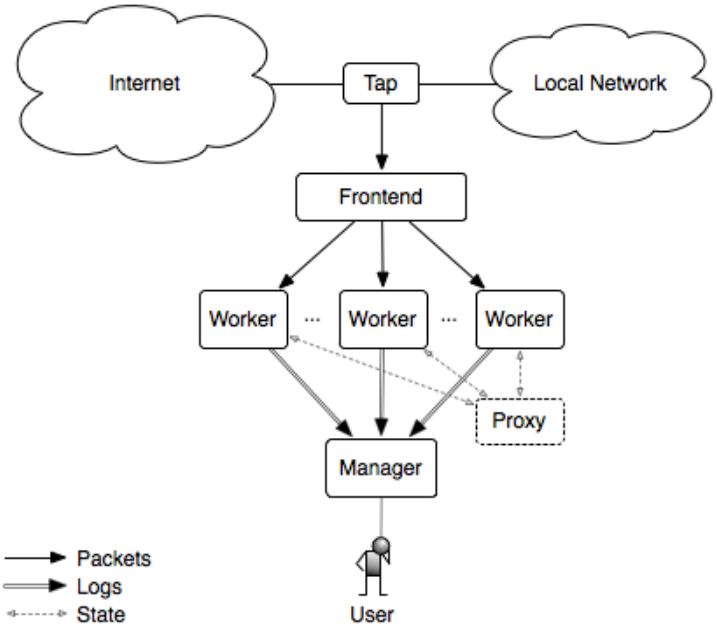

Figure 2.5: Architectural components of a Bro cluster from [Bro2016]

Parameter	**Bro**	**Snort**
Contextual signatures	Yes	No
Flexible site customization	High	Medium
High speed network capability	High	Medium
Large user community	No	Yes
Configuration GUI	No	Yes
Analysis GUI	A few	A lot
Installation/deployment	Difficult	Easy
Operating system compatibility	Unix	Any

Table 2.2: Comparison of Snort and Bro from [Mehra2012]

Because Bro is not multi-threaded, many implementations may employ a Bro cluster which shares the overall workload with as many cores or nodes as are available. Bro's authors have designed in the agility to deploy in such a manner using a cluster architecture (see Figure 2.5). In this figure, the manager processes the logs and notices Bro produces while the Proxy administers the workload sharing between the worker nodes, which represent any number of cores or physical machines that comprise the cluster. The Bro authors generally estimate that each core can handle 250 Mbps of traffic.

Lastly, it is noteworthy that among its various frameworks, Bro uses a signature framework to provide functionality where signatures better identify a malicious connection than its behavior does. At one time there was a script named `snort2bro` which sought to achieve Snort-like capability by importing Snort rules into Bro syntax to be implemented in Bro, but it is no longer maintained because it does not capitalize well on Bro's additional capabilities [Bro2016]. All in all, Bro's signature framework allows it to achieve a somewhat hybrid approach between behavioral and signature detection.

2.3: THE HYBRID APPROACH

The major differences between Snort and Bro originate in differences in their IDS philosophy. One significant low level difference between Snort and Bro and ultimately their IDS philosophy is this: Snort is packet-oriented and Bro is connection-oriented. Bro can process traffic at higher rates and has more configurable and flexible options, but it uses more complex policies, must be more specifically tuned when deployed, and lacks a GUI. Snort boasts a stronger user community and therefore more comprehensive rule database (see Table 2.2) [Mehra2012].

In Table 2.1, red fields indicate faults to be mitigated and green fields indicate strengths of the approach. In anomaly detection, the risk of computation complexity can

be effectively mitigated to avoid adding load to the network. As mentioned above, anomaly detection deployments increasingly rely on specialized, faster hardware. Administrators can also utilize network architecture mechanisms, such as using a hybrid of online and offline processing of packet captures. Particularly, configuring port mirroring or SPAN ports on network switches or routers branches all live traffic to be processed online or offline. [Papadogiannakis2010] proposed using selective packet discarding to handle processing greater traffic loads by discarding high volume low risk traffic such as video streaming, Voice over Internet Protocol (VoIP), or other multimedia content. While it is impossible to detect previously unknown threats, or zero day threats, with signature-based detection alone, administrators might use a collection of approaches, with and without IDSs, to identify these threats. Most of the time, a hybrid of the two approaches is the most effective in which: a behavior-based IDS is used to profile new threats and a signature-based IDS is used to detect these exact profiles.

2.4: IDS AND IPv6

Based on my literature review, research and experimentation in the public domain bridging the gap between IDS solutions and IPv6 vulnerabilities is sparse. The fields of networking and information security are relatively young and develop at a relatively fast pace. Additionally, it is suspected that a majority of innovative information security advancements are retained privately for security reasons. Even still, some works such as [Schütte2014] exist, which presents an IPv6 plugin for Snort. The plugin adds "IPv6-specific rule options" for Snort signatures as well as builds its own network view to track NDP broadcasts. [Elejla2016] reviews several methods utilizing an IDS to flag ICMPv6-based DDoS attacks, claiming that ICMPv6 is the most prevalent protocol used for DoS and DDoS attacks of IPv6 networks.

In the most relevant work I am aware of, [Gehrke2012] samples 13 categories of malicious attacks and tests whether or not Bro and Snort alert on single trials of host-to-

host or host-to-router events in a native IPv6 testbed and a transitional IPv6-over-IPv4 tunneled testbed. This work was published when Google's IPv6 access rate was approximately 0.45% and before popular intrusion detect systems began supporting default comprehensive IPv6 detection [Google2017][Bro2016].

This work focuses on IPv6 attacks in 4 major categories: reconnaissance, flooding, malware delivery, and exploits—in the form of Common Vulnerabilities and Exposures. Attacks in these categories can be dangerously effective at taking IPv6 networks or nodes offline or infecting them with adversarial programs. I compare and evaluate behavior-based and signature-based IDSs for these attack vectors across thousands of trials in laboratory network while modeling varying degrees of degradation of general network conditions. To my knowledge, this work is novel in its statistically significant testing of open-source intrusion detection.

Chapter 3: Experimental Setup

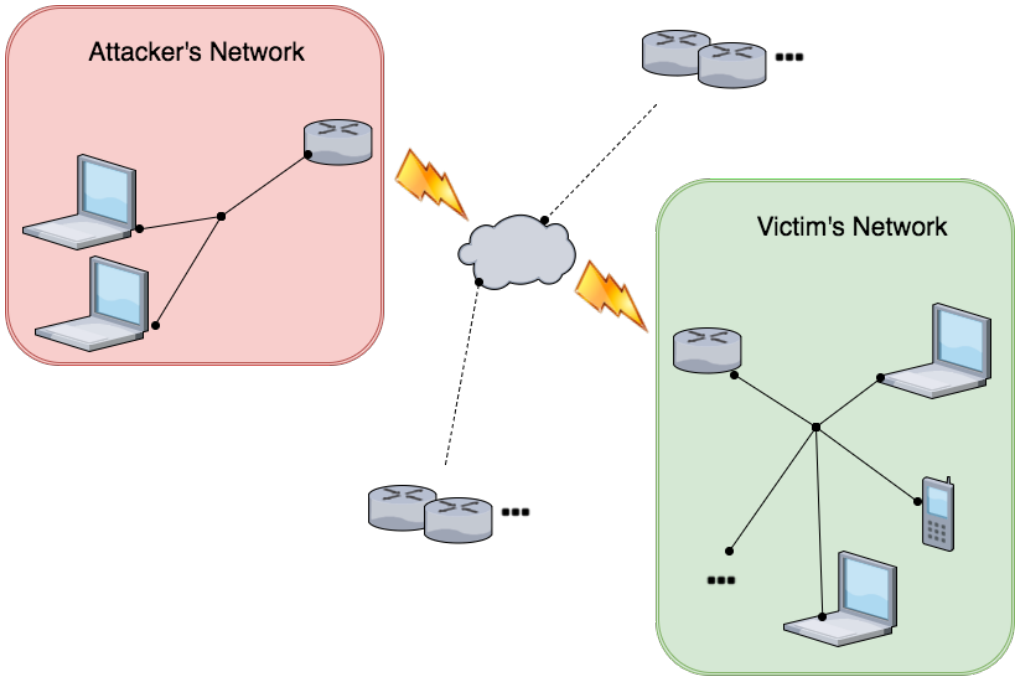

Figure 3.1: Network diagram of a host-to-host cyberattack over the Internet

3.1: Virtual Test Network Architecture

The classes of attacks of interest and reasonable assumptions of network conditions influenced the major design choices of the architecture of the test network. The test network is fully virtualized using VMWare Fusion running on a MacBook Pro equipped with a 2.8GHz Intel Core i7 processor and 16 GB 1600 MHz DDR3 RAM [VMWare2017]. The experimental setup simulates events which originate from one or more hosts on one or more Local Area Networks (LAN), travel across the Internet, and terminate in a victim LAN containing a targeted host (see Figure 3.1). The virtual test network architecture is as follows. One virtual LAN configured as a Host-Only Network

contains two 64-bit virtual machines as guests hereafter referred to as: ATTACKER and TARGET.

ATTACKER is an Ubuntu 16.04 virtual machine (VM) loaded with The Hacker's Choice Internet Protocol version 6 (THC-IPv6) attack suite, Ostinato packet generator, NetEm network emulator tool, and Linux utility `tcpdump` [Hauser2017][P2016][Hemminger2005][tcpdump2017]. German security developer van Hauser created THC-IPv6 in 2005 to fill a gap in open-source security research tools for IPv6 [Hauser2017]. He has continued to maintain it through the present. Indian network engineer Srivats P. first publicly released a flexible packet generator, Ostinato, in 2010 which filled a gap in open-source networking load testing tools and the like. Ostinato's architecture uses a controller as an administrator to control an agent, often running on the same machine, to transmit streams of random network traffic to a Device Under Test (DUT) (see Figure 3.2). In my setup, the DUT is TARGET. NetEm is a popular network emulation tool installed on selected network interfaces to develop and test products that rely on internetworking, like online games [Hemminger2005]. The NetEm tool enhances Linux's traditional traffic control (`tc`) facilities by allowing a developer to arbitrarily designate packet drop rate, latency, duplication rate, reordering, probability of bit error, and more.

TARGET is a fully updated Ubuntu 16.04 VM with no major additions to its software. Its function is simply to exist as an addressable and discoverable node on the network which responds as expected to appropriate requests. I did not use a vulnerable VM image because it places an unnecessary burden on the automation of thousands of trials to restart or recreate an addressable node on the network.

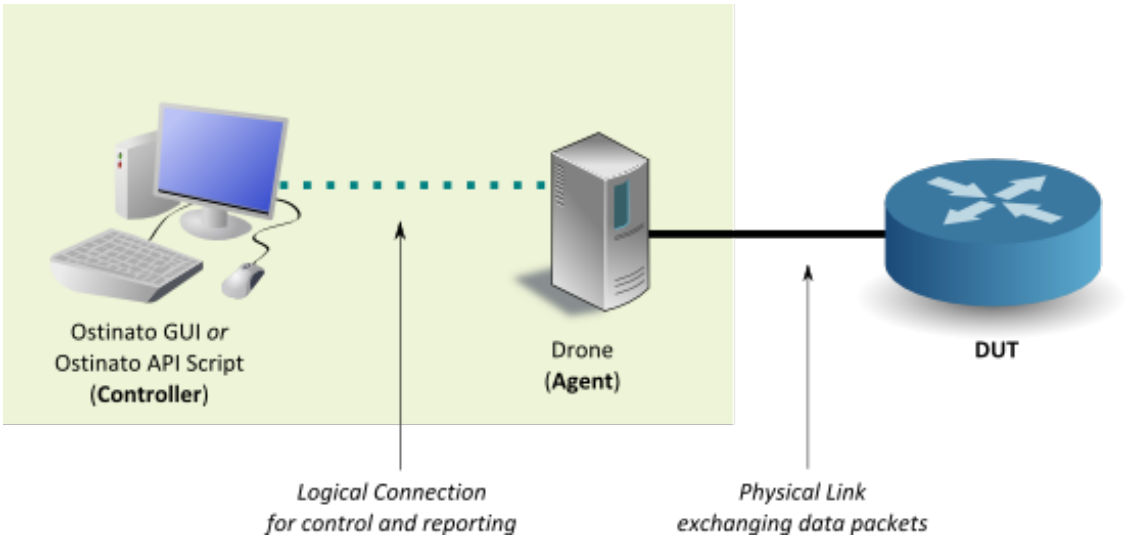

Figure 3.2: Ostinato's controller-agent architecture to transmit to DUT from [Ostinato2016]

I use an additional VM, IDS-VM, is a 64-bit Ubuntu 16.04 VM, but this VM is not connected to the test network. Bro and Snort are installed from source, configured, and run on this VM.

In a Host-Only Network configuration, the Network Interface Card (NIC) in each VM is fully isolated from any networking that the host engages in, such as the Internet or any communication on the host's real LAN. This design ensures that data recorded in the experiment is precise and intentional.

3.2: Design of Network Architecture

Due to the difficulty of testing IDS algorithms in a laboratory setting, I make a number of assumptions which directly influence all design choices. Major design choices for the network are:

- One LAN
- No routers

- Benign traffic generator on attacker node
- No firewall

The test network consists of only a single LAN because this experiment is only concerned with host addressable attacks. It assumes that an attacker already knows the IP address of the target or is scanning for it. While multiple LANs and the processing and forwarding between and through them would demonstrate scalability of this experiment, I perform load testing in other more measurable ways.

The decision to exclude any routers in the test network is based on a number of assumptions. First, while plenty of interesting and effective IPv6 attacks involved targeting vulnerable routers (dumping router information, flooding routers, man in the middle attacks), such attacks are beyond the scope of this project and none of the attacks used explicitly target or require routers in the network. DHCP, NAT, and IP forwarding are major protocols and services for which routers are needed in today's networks. DHCP is not necessary in this experiment because each guest VM is assigned a unique IPv6 address by the virtualization software. NAT is a redundant service in an ideal IPv6 internetwork in which all network nodes are uniquely addressable from any other node in the internetwork. Finally, this setup assumes that for such an experiment in which a raw view of egress packets is necessary for analysis, any included router would simply forward packets to their destinations.

ATTACKER creates multiple traffic streams speaking different protocols and conversations to and from target and imaginary hosts on the internetwork. It is assumed that a targeted machine would witness some of this background noise in a real network in addition to any traffic sent to and from the target. The benign traffic generator greatly

amplifies the integrity of simulation of the network events and more than one node is excessive.

No firewall or packet blocking mechanism is implemented in the network so as to allow the IDS a full and unfettered view of the network traffic. Any realistic enterprise network employs a firewall and many other security measures to protect its servers and internal networks. However, implementing such technologies would bias the results of this experiment. Additionally, we assume that in a real network, Bro or Snort would passively monitor network traffic at the gateway of the targeted enterprise network and would therefore have the most comprehensive view of traffic leaving and entering.

From the original figure of a host versus host attack vector across the Internet, one could create an experimental network in which one router separates the attacker and traffic generator from the target (see Figure 3.3). In this case, the one router represents all routing across the Internet and between the two LANs. Given the assumptions and explanations above, I abstract one step further and eliminate the router to achieve a simpler network architecture (see Figure 3.4).

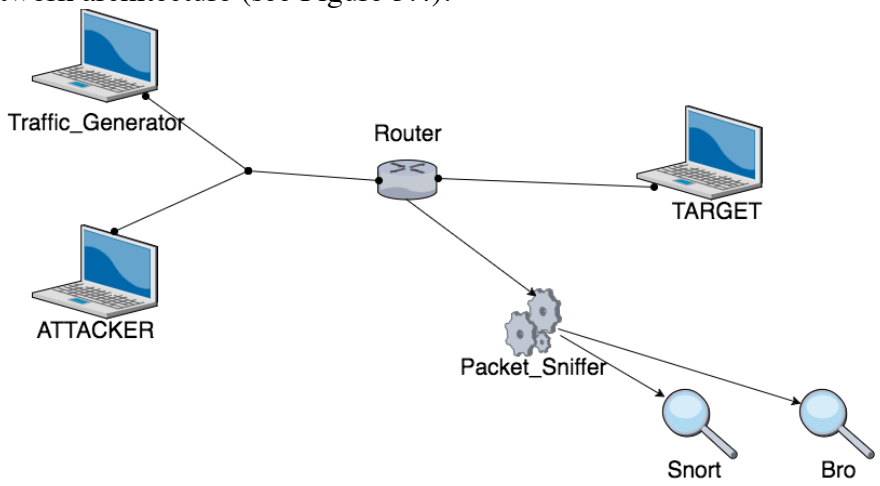

Figure 3.3: Network diagram of host-to-host attack over Internet abstracting away all routing over the Internet

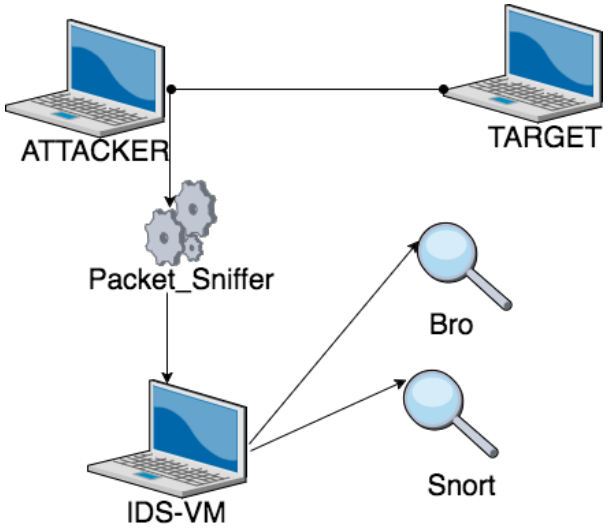

Figure 3.4: Network diagram of the virtual test network used

3.3: Design of Experiment Variables

Throughout my experiments, I seek to model network traffic as influenced by the realities of many idiosyncratic causes of non-ideal network infrastructure. Because the information security community should anticipate IPv6 cyberattacks to originate anywhere from a local LAN to intercontinental networks, I experiment with a variety of network conditions. I use 4 independent variables as parameters to model a real network in the virtual test network: benign traffic streams, drop rate, transmit latency, and probability of byte error.

I configure Ostinato to generate benign traffic streams consisting of variable-length TCP, UDP, IGMP, and ICMPv6 packets and streams carrying random payloads to and from random IPv6 addresses. I provide more detailed statistical characteristics of the Ostinato streams in §4.3. These benign streams simultaneously increase the load on the monitors and model the innocent traffic on a network without increasing the amount of malicious attacks. I treat the benign traffic stream variable as a binary parameter. I turn it

off when acquiring baseline measurements and on when testing with the other independent variables.

The packet loss rate variable represents any packet that is not successfully delivered to its intended receiver. This variable models network degradations such as improper Layer-3 routing, a router's buffer overflowing, mistakes in Layer-2 transmissions, etc.

The transmit latency variable represents a wide variety of latencies for Internet communications caused by physical distance between endpoints, routing delays, Layer-1 hardware abnormalities, etc.

Finally, the packet corruption variable catches a remainder of errors mostly due to Layer-2 noise. Many error detection mechanisms are typically used like cyclic redundancy checks (CRC), parity checks, and checksums which can often result in a retransmission of the information. However, there is still a probability of error at a receiver because some undetected errors can still pass CRC or parity checks and any errors that are not checked until arriving at the receiving host or LAN will still pass through a NIDS placed at the top of the network.

3.4: ATTACK TOOLS

Table 3.1 shows the attack categories sampled and which attacks represent them in the virtual network. The source code for the attacks are written by [Hauser2017] and are configured and initiated on the UNIX command line interface (CLI) terminal. A large majority of the attack tools used leverage vulnerabilities in ICMPv6. It makes sense that a study on IPv6-specific attacks would focus on Layer-3 protocols, the most prominent of which is ICMPv6. However, for being a relatively rarely used protocol in the wild, ICMPv6 poses a disproportionately large security risk. [Labovitz2008] analyzed

anonymously gathered Internet traffic statistics from 67 Internet Service Providers (ISPs) over 2 years in 17 countries, and found that 27% of attacks observed used the ICMP protocol even though ICMP is responsible for less than 0.001% of Internet traffic. While the overwhelming majority of these statistics undoubtedly referred to ICMPv4, ICMPv6 should expect similar usage as IPv6 networks increase in number.

The THC-IPv6 tools used in this experiment exploit vulnerabilities that are unique to protocols and services in IPv6. It targets ICMPv6 weaknesses such as neighbor advertisements, neighbor solicitations, IPv6 multicast broadcasts, and generic ICMPv6 echo request/reply pairs. There are plenty of IPv4 attack tools that fall into reconnaissance, flooding, malware delivery, and various CVE weaknesses. In fact, a large portion of tools in the THC-IPv6 suite are designed to implement IPv4-era mechanisms on IPv6 networks by using IPv6-specific technologies and protocols. To be clear, the IPv6 attacks used in this experiment or in the rest of the tool suite would not be effective on strictly IPv4-configured networks or hosts.

IPv6 Attacks Used	
Attack Category	**Tool**
Reconnaissance	alive6
Flooding	denial6
	flood_advertise6
	flood_solicitate6
Malware Delivery	covert_send6
Common Vulnerabilities and Exposures (CVE)	exploit6

Table 3.1: Table of IPv6 attack categories and tools used in experiments

It is important to note where these attacks fall in Lockheed Martin's industry standard Cyber Kill Chain which guides security researchers in mitigating risk and defending information systems from advanced persistent threats (APT) (see Figure 3.5). APTs are patient, surgically targeted cyber threats that may last for months to years and are generally orchestrated by highly resourced, motivated, and capable nation states. The Cyber Kill Chain describes the 7 stages of an APT: reconnaissance, weaponization, delivery, exploitation, installation, command and control, and actions on objectives. This kill chain generally outlines each step of a successful malicious operation that must be completed before proceeding to the next.

As a Layer-3 protocol, IPv6 vulnerabilities fall in steps 1-4 of the kill chain model: reconnaissance, weaponization and delivery. The subsequent steps tend to require more access to specific host conversations and victim manipulation that is greatly benefited by information transmitted via Layer-4 to Layer-7 protocols. Additionally, many of these exploits are more host-based as opposed to network-based. Usually, the only aspects of steps 4-7 that would be visible to the network is the command and control stage, which establishes a reliable communication link between the attacker and his target(s) over the network.

As a result, the categories of IPv6 attacks sampled for this experiment and described in the sections above cover potential IPv6 tools to be used in steps 1-4 of the Cyber Kill Chain model.

Figure 3.5: Lockheed Martin's industry-standard Cyber Kill Chain model from [Lockheed2017]

3.4.1: Reconnaissance

alive6 is a reconnaissance tool capable of using a variety of methods to identify alive IPv6 nodes on a network visible to the attacker's network interface. An attacker can specify an arbitrary range of addresses or scan the entire link-local network. The basic premise of the method sends requests in various forms (ICMP, DNS resolve alive, TCP SYN, TCP ACK, UDP) to many addresses and analyzes whether or not there was a reply. The nature of each reply may also indicate information such as host OS, open ports, and services running. For this experiment, I use a default configuration of alive6 which sends an ICMPv6 ping request packet to the link-local multicast address ff02::1.

3.4.2: Flooding

The following attacks are IPv6-specific DoS or flooding attacks.

denial6 offers 7 different ICMPv6 ping request DoS attacks which emit high volumes of packets with attributes that incur heavy workload for any victim to process them. The options that are available include: "large hop-by-hop header with router-alert and filled with unknown options", "large destination header filled with unknown options", "hop-by-hop header with router alert option plus 180 headers", etc. [Hauser2017]. In my experiment, I configure denial6 under the first option using large hop-by-hop headers and a router alert. On my hardware, denial6 transmitted approximately 66000 packets/sec to the victim from random IPv6 addresses for the 10 seconds it was allowed to run.

flood_solicitate6 floods a victim's network with ICMPv6 NSs from randomized MAC and IPv6 addresses. On my hardware, flood_solicitate6 was capable of increasing traffic on the network by approximately 66000 packets/sec during the 10 seconds it ran, which includes any attempts by the victim of replying to the solicitations. I specifically

configured flood_solicitate6 to target the IPv6 address of TARGET, instead of simply flooding the network with random destinations.

flood_advertise6 floods a victim's network with ICMPv6 NAs addressed from random IPv6 hosts. Since NAs are designed to be replies to solicitations, the victim did not transmit any replies during experimentation. Even still, flood_advertise6 increased traffic on the network by approximately 66000 packets/sec for the approximately 10 seconds it ran.

3.4.3: Malware Delivery

covert_send6 is a proof-of-concept malware delivery method for IPv6. It covertly sends files from a random IPv6 address to an already compromised, expecting victim by injecting file partitions into the destination headers of a series of ICMPv6 ping requests. For this experiment, I configure covert_send6 to send the binary for the popular 2009 virus Conficker. covert_send6 generates 64 packets to send the designated binary and the victim replies with 64 symmetric ICMPv6 ping replies, in turn.

3.4.4: CVE

exploit6 implements various CVE and non-CVE techniques to compromise or kill a victim host. It begins and ends by checking whether a host is alive or not using a normal ICMPv6 ping request. It sends a series of large ping requests with checksum mistakes followed by implementations of CVE-2003-0429 and CVE-2004-0257. Submitted in June 2003, CVE-2003-0429 describes the use of buffer overflow attacks to cause the packet interpretation software Ethereal 0.9.12 to crash while dissecting packets with invalid IPv6 prefix lengths [Mitre2017]. Submitted in September 2004, CVE-2004-0257 describes the transmission of an IPv6 packet with a low Maximum Transmission

Unit (MTU) followed by a TCP connection to an open victim port to cause a DoS attack against vulnerable OpenBSD and NetBSD distributions [Mitre2017].

Chapter 4: Procedure/Data Generation

The scripts which automate generation, evaluation, and visualization can be found as part of the software release which accompanies this work [GinCode2017]. The data generated throughout the course of experimentation consists of all PCAPs, ASCII output files, aggregated data, and visualizations. These can be found as part of the dataset release which accompanies this work [GinData2017].

4.1: Baseline Measurements

To ensure the various systems are functioning correctly, I establish baseline measurements of the various attacks in their most basic forms without network degradation or background noise added to the controlled virtual environment. These measurements verify whether or not Bro and Snort can successfully identify the chosen IPv6 attacks and malware in their most obvious form (see Table 4.1). Collecting and verifying baseline measurements ensures the experimenter that each step of the analysis process is functioning before proceeding to the next. This is an important step in verifying whether or not each IDS is configured and functioning correctly. The feedback that a good IDS provides network administrators is not a true fact about whether or not a compromise occurred. Instead, it relays a detailed description of what events its sensor observed, which rules generated the alert(s), how many alerts were generated, among a whole host of additional information that helps an administrator form a contingency plan going forward.

4.1.1: Baseline Binary Detection Results

The most basic baseline is whether or not Bro or Snort detect *any* unusual traffic for each of the attacks with no superfluous network traffic or network degradation. This is a binary assessment that does not relay the IDSs' qualitative alert messages.

Binary detection matrix for baseline measurement			
Attack Category	**Tool**	**Bro Detected?**	**Snort Detected?**
Reconnaissance	alive6	no	yes
Flooding	denial6	no	yes
	flood_advertise6	yes	yes
	flood_solicitate6	no	yes
Malware Delivery	covert_send6	no	yes
CVE	exploit6	yes	yes

Table 4.1: Binary detection matrix for baseline measurement of all attacks by Bro and Snort, listed by attack category

For the binary detection baseline measurement, I use 1 PCAP of each attack and determine if at least a single alert is generated on any communication involving either IPv6 address of ATTACKER or TARGET. Theoretically, binary detection baseline measurement only requires a single PCAP of each attack since these captures are relatively deterministic with no uncertainty from network degradation or network traffic. Snort detects all attacks, while Bro detects only flood_advertise6 and exploit6.

4.1.2: Baseline Qualitative Detection Results

Binary detection is a good baseline with which to start, but it does not tell the whole picture. The alert messages that Bro and Snort generate are crucial in determining the severity of an event and in describing the overall attack picture. To further describe the attacks to the network administrator, the IDSs provide alert messages. I cover alert messages generated by both IDSs corresponding to each attack in the next subsections. Bro provides two flavors of feedback in weird.log and notices.log. weird.log records network events deemed unexpected or anomalous but not necessarily malicious. notices.log records network events deemed more malicious in nature. In this work, when Bro messages are provided, I differentiate between the two by parenthetically labeling each message.

4.1.2.1: alive6

Described in §3.4.1, alive6 does lead to Snort alerts but not to Bro weird records or notices. Snort alerts on generic ICMPv6 Echo Request/Replies because inter-domain ICMP usage is commonly blocked. Snort also more specifically generates alert code 1:24303:6 which correctly reports a multicast neighbor add attempt.

alive6 alert messages for baseline measurement by Snort	
Snort Message	Number/Total
[1:24303:6] PROTOCOL-ICMP IPv6 multicast neighbor add attempt	2/8
[1:28292:1] PROTOCOL-ICMP IPv6 0xfacebabe ICMP ping attempt	2/8
[1:18474:3] PROTOCOL-ICMP ICMPv6 Echo Request	2/8
[1:18473:3] PROTOCOL-ICMP ICMPv6 Echo Reply	1/8
[1:448:10] PROTOCOL-ICMP Source Quench undefined code	1/8

Table 4.2: Qualitative detection matrix showing alert messages for baseline measurement of alive6 by Snort

4.1.2.2: denial6

Described in §3.4.2, denial6 does lead to Snort alerts and to 1 Bro weird record. Because the Bro record does not identify any anomalous behavior of denial6 and only produces 1 record, I include it in Table 4.3 below but disregard it as an outlier for the rest of this work. Snort again alerts on generic ICMPv6 Echo Requests, but does not pick up on the large hop-by-hop headers in these packets. However, a security administrator could deduce that Snort has witnessed a DoS attack from this data alone due to the large number of the same alerts in a short period of time.

denial6 alert messages for baseline measurement by Bro	
Bro Message	Number/Total
truncated_inner_IP (weird)	1/1

Table 4.3: Qualitative detection matrix showing alert messages for baseline measurement of denial6 by Bro

denial6 alert messages for baseline measurement by Snort	
Snort Message	Number/Total
[1:28292:1] PROTOCOL-ICMP IPv6 0xfacebabe ICMP ping attempt	76973/153946
[1:18474:3] PROTOCOL-ICMP ICMPv6 Echo Request	76973/153946

Table 4.4: Qualitative detection matrix showing alert messages for baseline measurement of denial6 by Snort

4.1.2.3: flood_advertise6

Also described in §3.4.2, flood_advertise6 leads to Snort alerts and Bro weird records. Snort correctly identifies a large number of IPv6 NA flood attempts.

flood_advertise6 alert messages for baseline measurement by Bro	
Bro Message	Number/Total
zero_length_ICMPv6_ND_option (weird)	247/247

Table 4.5: Qualitative detection matrix showing alert messages for baseline measurement of flood_advertise6 by Bro

flood_advertise6 alert messages for baseline measurement by Snort	
Snort Message	Number/Total
[1:24294:2] PROTOCOL-ICMP IPv6 neighbor advertisement flood attempt	678162/678162

Table 4.6: Qualitative detection matrix showing alert messages for baseline measurement of flood_advertise6 by Snort

4.1.2.4: flood_solicitate6

Also described in §3.4.2, flood_solicitate6 leads to Snort alerts, but no Bro weird records or notices. Snort correctly identifies a large number of IPv6 multicast neighbor add attempts.

flood_solicitate6 alert messages for baseline measurement by Snort	
Snort Message	Number/Total
[1:24303:6] PROTOCOL-ICMP IPv6 multicast neighbor add attempt	6143/6143

Table 4.7: Qualitative detection matrix showing alert messages for baseline measurement of flood_solicitate6 by Snort

4.1.2.5: covert_send6

Described in §3.4.3, covert_send6 leads to Snort alerts, but no Bro weird records or notices. Snort correctly identifies 128 pairs of ICMPv6 Echo Request/Replies. Snort is likely alerting on the 128 transmitted packets covertly containing the malicious binary, and the 128 symmetric packets sent by TARGET in reply.

covert_send6 alert messages for baseline measurement by Snort	
Snort Message	Number/Total
[1:18473:3] PROTOCOL-ICMP ICMPv6 Echo Reply	128/258
[1:18474:3] PROTOCOL-ICMP ICMPv6 Echo Request	128/258
[1:24303:6] PROTOCOL-ICMP IPv6 multicast neighbor add attempt	1/258
[1:27611:1] PROTOCOL-ICMP Truncated ICMPv6 denial of service attempt	1/258

Table 4.8: Qualitative detection matrix showing alert messages for baseline measurement of covert_send6 by Snort

4.1.2.6: exploit6

Described in §3.4.4, exploit6 leads to Snort alerts and Bro weird records. Snort correctly identifies each test within the exploit6 tool. Bro correctly identifies the 2 CVE techniques in exploit6. It makes sense that Bro does not identify the ICMPv6 Echo Requests because these are fairly normal behaviors and are ignored in the testing of the other tools in this experiment.

exploit6 alert messages for baseline measurement by Bro	
Bro Message	Number/Total
excessively_large_fragment (weird)	1/2
zero_length_ICMPv6_ND_option (weird)	1/2

Table 4.9: Qualitative detection matrix showing alert messages for baseline measurement of exploit6 by Bro

exploit6 alert messages for baseline measurement by Snort	
Snort Message	Number/Total
[1:18474:3] PROTOCOL-ICMP ICMPv6 Echo Request	4/15
[1:28292:1] PROTOCOL-ICMP IPv6 0xfacebabe ICMP ping attempt	3/15
[1:18473:3] PROTOCOL-ICMP ICMPv6 Echo Reply	3/15
[1:24296:6] PROTOCOL-ICMP IPv6 router advertisement invalid prefix option attempt	1/15
[1:461:12] PROTOCOL-ICMP unassigned type 2 undefined code	1/15
[1:460:12] PROTOCOL-ICMP unassigned type 2	1/15
[1:24297:2] PROTOCOL-ICMP IPv6 oversized ICMP ping attempt	1/15
[1:24295:2] PROTOCOL-ICMP suspicious IPv6 router advertisement attempt	1/15

Table 4.10: Qualitative detection matrix showing alert messages for baseline measurement of exploit6 by Snort

4.1.3: Baseline Resource Usage Results

Since a good comparison of the two NIDS approaches is also concerned with the NIDS' abilities to function under complexity and memory constraints, I take baseline measurements for the resources used in the baseline measurements as well (see Table 4.4). Throughout the analysis, I varied the technical specifications of IDS-VM to accommodate for the resource needs of each operation. The RAM for the VM varied from 4 GB to 10 GB and the number processor cores were between 1 and 4. These can be varied to accommodate each operation without affecting the results because I collect the resource usage of each trial. I varied the number of processor cores for Bro operations only because Bro is single-threaded, and this would not affect Bro's run-time.

Baseline measurement of run-time and memory allocation				
Tool	**Bro Time (s)**	**Bro Memory (MB)**	**Snort Time (s)**	**Snort Memory (MB)**
alive6	0.0036	2.0	1.138	156.60
denial6	0.4486	2.0	4.538	156.61
flood_advertise6	18.5839	4897.0	18.6875	156.60
flood_solicitate6	17.7756	4730.0	17.16	156.87
covert_send6	2.2985	2.0	3.14338	156.60
exploit6	0.0193	2.0	1.445	156.60

Table 4.11: Baseline measurements for run-time and memory usage

4.1.4: Baseline Malware Signatures

Finally, I verify that Bro and Snort should be reasonably expected to identify the delivery of a specific piece of malware. For this experiment, I chose an older, well-known worm called Conficker. Conficker quickly spread across an estimated 3 million

Windows XP and Windows 2003 hosts worldwide in 2008-2009 by using new evasion techniques and blocking the user from accessing security countermeasures [Shearer2013]. Notably, it used the `autorun.exe` feature of external drives to run its executable to install as legitimate sounding Windows registry entries and delete security-related ones. Over the course of about a year, five known versions surfaced, of which I experiment using the first, CONFICKER.A [Parkour2017]. The origin of the malware is believed to be Ukraine because some versions beaconed to a Ukrainian server for updates and the malware avoided infecting Ukrainian IP addresses. The malware has been given different names by different anti-malware companies, but its hash is consistently verifiable. I submitted the hash of the Conficker executable to Internet malware lookup service VirusTotal which checked it against 61 malware databases [VirusTotal]. 55 of the 61 databases verified the file's hash as malicious. Since I am testing the effectiveness of an IDS to detect specific malware delivery methods and not how correctly an IDS assesses malware, an old, verifiably malicious file is ideal for testing.

4.2: Scripting And Data Flow

The methodology for generating and evaluating the data relies heavily on Python scripting for automation. Assembling statistically significant results and a realistic view of a network is crucial to the analysis. During the experiment, I generate over 7000 35-second PCAPs for evaluation representing different attacks and network conditions.

Because Bro and Snort are built for real-time human analysis, the alerts provided by Bro and Snort are qualitative in nature and qualitative judgments determine how effective such an alert would be to a human administrator. I design my scripting process to preserve this information instead of reducing the results down to true positive and false positive rates.

4.2.1: PCAP Generation

The first automated task is that of generating the PCAP data for analysis [GinData2017]. Initially, I intended to use the proprietary VMWare utility vmrun.exe to manage multiple VMs simultaneously, but its functionality proved inconsistent when executing various commands. Ultimately, a single Python script, run.py, running on ATTACKER iterates through the list of 6 IPv6 attacks and 8 network settings [GinCode2017]. For each combination, it modifies the NIC setting using NetEm and then performs 100 trials. For each trial, the script initiates a `tcpdump` capture as a Python subprocess, sleeps for 10-15 seconds before executing an attack (also as a Python subprocess), waits for the attack to terminate independently, and then waits for the `tcpdump` capture to reach 35 seconds in length whereupon it terminates independently. For each trial featuring a DoS attack, the script deviates in only two ways: (1) it sleeps for 5-10 seconds before executing a DoS attack and (2) it terminates a DoS attack after 10 seconds of running. During the 35 seconds while `tcpdump` is running, any malicious or generated benign traffic is captured, as well as any normal operation of the 2 VMs on the network, such as Domain Name Service queries.

4.2.2: NIDS Analysis

All PCAPs are processed and visualized offline in IDS-VM through additional Python scripts: snort_eval.py, bro_eval.py, snort_plot.py, and bro_plot.py [GinCode2017]. Each of the first two 'evaluation' scripts iterates through all expected PCAPs, creates a bash file with execution commands for each NIDS in PCAP reading mode, executes the bash file in the terminal, reads the output in the respective form of each NIDS, and then aggregates the quantitative and qualitative results in a human-readable results file for further analysis. The script for Bro evaluation reads the weird.log

and/or notices.log of each trial and appends the relevant details of each trial to the Bro results file. The script for Snort evaluation outputs Snort's results to the console and to a file (using the `tee` command), reads the file, and appends the relevant details of each trial to the Snort results file. The 'plot' scripts parse the ASCII results files for visualization and reporting. It aggregates all data points into a series of matrices, exports the statistics in a .csv file format designed to create various charts in Microsoft Excel [GinData2017]. Box and whisker charts and scatter plots are generated and used to compare results across network degradation settings and across Bro and Snort [GinData2017].

4.3: OSTINATO

For all benign and malicious PCAPs generated (except some of the baseline measurements), the Ostinato agent-controller architecture generated 6600 packets/sec of traffic in the IPv6 protocols expected to be most popular: ICMPv6, IGMPv6, TCP, and UDP. The packets were assigned a variety of attributes to roughly simulate real network traffic (see Table 4.4). These characteristics were partially influenced by [Labovitz2008] which analyzed the protocol breakdown of traffic anonymously gathered at the aforementioned 67 ISPs. [Labovitz2008] found that of approximately 1 Tbps of average inter-domain traffic, TCP comprised an approximate average of 900 Gbps, UDP comprised an approximate average of 60 Gbps, and all other protocols comprised an average of under 10 Gbps. There is no fine-grained restriction on the similarity of traffic emulated in the lab network to Internet traffic since Internet traffic varies so much across networks, not to mention that [Labovitz2008] only aggregates statistics on inter-domain traffic (see Table 4.5).

Protocol	Frame Length	Payload Data	Packets/ Bursts?	Transmission Rate	Transmission in Bits/Sec
TCP	78-500 bits	Random	Bursts	20 bursts/sec and 250 packets/burst	12.34 Mbps
UDP	78-750 bits	Random	Bursts	4 burst/sec and 250 packets/burst	3.47 Mbps
ICMPv6	78-200 bits	Random	Packets	100 packets/sec	127.2 kbps
IGMPv6	78-500 bits	Random	Packets	500 packets/sec	1.24 Mbps
Total					18.07 Mbps

Table 4.12: Attributes of benign traffic streams generated by Ostinato by protocol

Protocol	This experiment	The wild
TCP	68.29%	~90%
UDP	19.20%	~6%
ICMP	0.70%	<0.5%
IGMP	6.86%	<0.1%
Other	0.00%	~4%

Table 4.13: Side-by-side comparison of benign traffic generated by Ostinato and Internet traffic statistics gathered by [Labovitz2008] by protocol by number of packets

4.4: NETWORK INTERFACE MANIPULATION

NetEm manipulates ATTACKER's network interface from which all malicious and benign traffic is transmitted. For each attack, NetEm implements 8 different network degradation settings by independently adjusting 3 unique parameters: packet loss rate, packet delay, and packet corruption rate. Designating a packet loss rate causes the NIC to randomly drop an amount of packets equal to or near the desired rate. In this experiment, I also add a 25% correlation sub-parameter from packet to packet, which forces each packet's probability of loss to be 25% dependent on whether or not the previous packet was dropped. Designating a packet delay induces a mean delay with normal distribution on all outbound packets. In this experiment, I specified a range of variation sub-parameter equal to 20% of the delay and a 25% correlation sub-parameter from packet to packet. For example, a NIC configured to induce 128 ms delay transmits packets with 128 ms ± 25 ms and with a 25% dependency on the delay of the previous packet. Finally, designating a packet corruption rate instructs the NIC to flip a bit at a random offset in an amount of packets equal to the corruption rate.

To achieve the 8 network degradation settings, I adjust a single parameter per setting to 2-3 different values. I choose the values to approximately sweep across common values for a mediocre trans-Atlantic Internet connection. There is no fine-grained restriction for these parameters since these factors vary widely depending on Internet connection, specific routing decisions, and numerous other variables. The parameter values I used are listed below:

- Packet loss rate: 0.1%, 1%, 10%, ('loss_1', 'loss_2', 'loss_3')
- Packet delay: 128 ms, 256 ms, 512 ms ('delay_1', 'delay_2', 'delay_3'), and
- Packet corruption rate: 10%, 25% ('corrupt_1', 'corrupt_2').

Chapter 5: Results and Analysis

5.1 Analysis of Baseline Measurements

A brief analysis of the baseline results provided in §4.1 justifies the presentation of the more in-depth results in this chapter. The most general detection results are the binary detection performance of both IDS philosophies, given in Table 4.1. Because Bro correctly alerts on flood_advertise6 and exploit6, I do not present true positive results of Bro detection for any other attack. However, I do include false positive results and resource usage of Bro for these attacks because they still demonstrate the effectiveness of Bro to a network security administrator who would not have access to labeled datasets in the real world.

The alerts produced by either Snort or Bro are specifically mentioned in Tables 4.2-4.10. The alerts produced by Snort span a range of general messages alerting based on ICMPv6 codes to specifically identifying neighbor add attempts and network flood attempts. The alerts produced by Bro tend to be more specific about the nature of the behavior observed, which makes sense, since the goal of Bro's alerting and logging system is to provide a neutral, low-level view to the user.

The resources observed in the baseline measurement (see Table 4.11) show great disparity between Snort and Bro and even between attacks. Bro shows an exceptionally fast

5.2: Results

In §5.2.1-§5.2.6, I present visualized results from all experiments in Figures 5.1-5.31 [GinData2017]. These figures contain true positive, false positive, run time, and memory usage distributions of IDS detection of all 6 attacks across all 9 network

degradation scenarios with side-by-side Snort and Bro comparisons where available. The 9 scenarios are labeled as mentioned in §4.4 with the addition of 'none', which denotes no network degradation or an ideal network.

The results are best presented as distributions of numbers of observed alerts and resources used because it is misleading in both a laboratory setting and a real-world setting for a single trial or PCAP to return a binary result of detection or lack of detection. These results are presented using box and whisker charts in which the length of each box shows the range between the first and third quartile of the data, an 'X' marks the median, whiskers show the range of outliers, and points mark the far outliers.

Also among Figures 5.1-5.31 are true positive vs. false positive scatter plots similar to a traditional receiver operating characteristic (ROC) plot for a binary classifier. A true ROC curve is not ideal to present the data in this experiment because of the nature of data collected and the subject field. A ROC curve in this experiment would potentially treat an IDS like a binary classifier for each packet or classification decision. However, the data collected are PCAPs, each containing a diverse number of packets on the order of 10^4 to 10^5. True positive rates could be on the order of 10^{-4} or 10^{-5} per packet. Treating an IDS like a binary classifier for an entire PCAP, especially without a standard length of PCAP, risks dichotomizing the experiment variables and drops useful quantitative and qualitative information about each trial's alerts.

Since Bro alerts on high level network behaviors and Snort alerts on each packet that triggers a rule, the number of alerts they produce based on an arbitrary amount of data varies widely. Snort can alert 10,000 times on 10,000 packets in a PCAP, whereas Bro might identify all 10,000 packets as components of one anomalous behavior and alert once. With this disparity in mind, I present the number of alerts on an absolute scale in

my plots so as to display the difference in philosophies. Each point on the plots is placed on the statistical mean of the number of true and false positives in 100 trials. In these plots, points closer to an infinite y-value and 0 x-value represent a better relative performance as it is ideal for an IDS to produce a high true positive rate and a low false positive rate. A way to mathematically determine the IDS performance for each point is to compare the geometric distance of each point to an arbitrarily high point on the y-axis, where the points reflect greater performance for smaller distance.

For each attack in §5.2.1-§5.2.6, I present the visualized results for that attack followed by any relevant analysis. More general analysis is presented in §5.3-§5.5.

5.2.1: alive6

The detection rates of alive6 across the network scenarios are fairly consistent with very little variation. One notable deviation is the spike in Snort's true positive and false positive detection for the loss_1 scenario (see Figures 5.1-5.2). There also seems to be a statistical downward trend of Snort's true positive detection as the packet loss rate is increased from loss_1, to loss_2, and then loss_3 (see Figure 5.1). This fact remains unexplained as packet loss seems like it would decrease Snort's detection which evaluates largely on a per packet basis.

Bro's false positive detection increases markedly in the corrupt_1 and corrupt_2 scenarios (see Figure 5.2). This can be explained by Bro's effective header checking rules that generate weird records for improper headers. As random bits are corrupted in packets, a small percentage of headers are corrupted and become invalid.

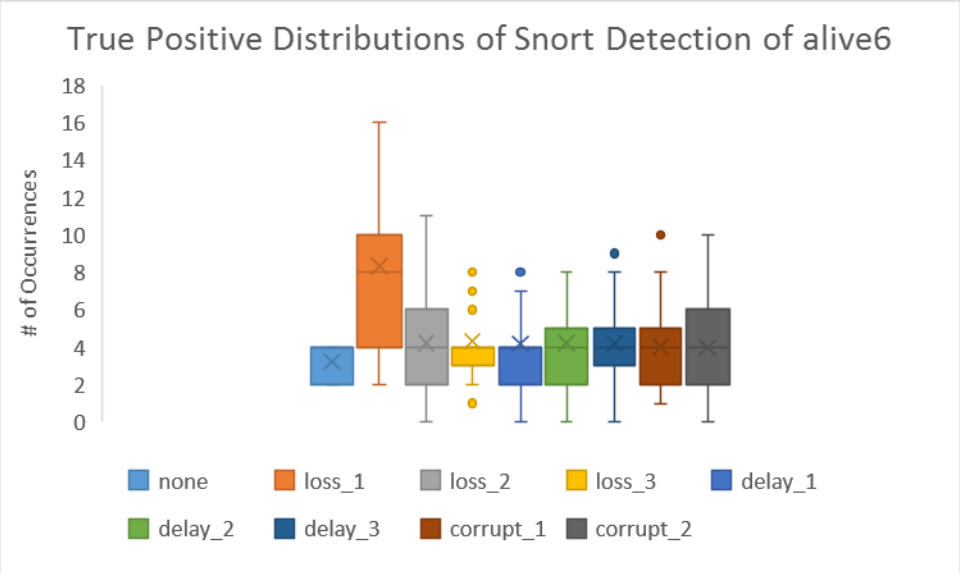

Figure 5.1: True positive distributions of Snort detection of alive6 across 9 different network degradation scenarios

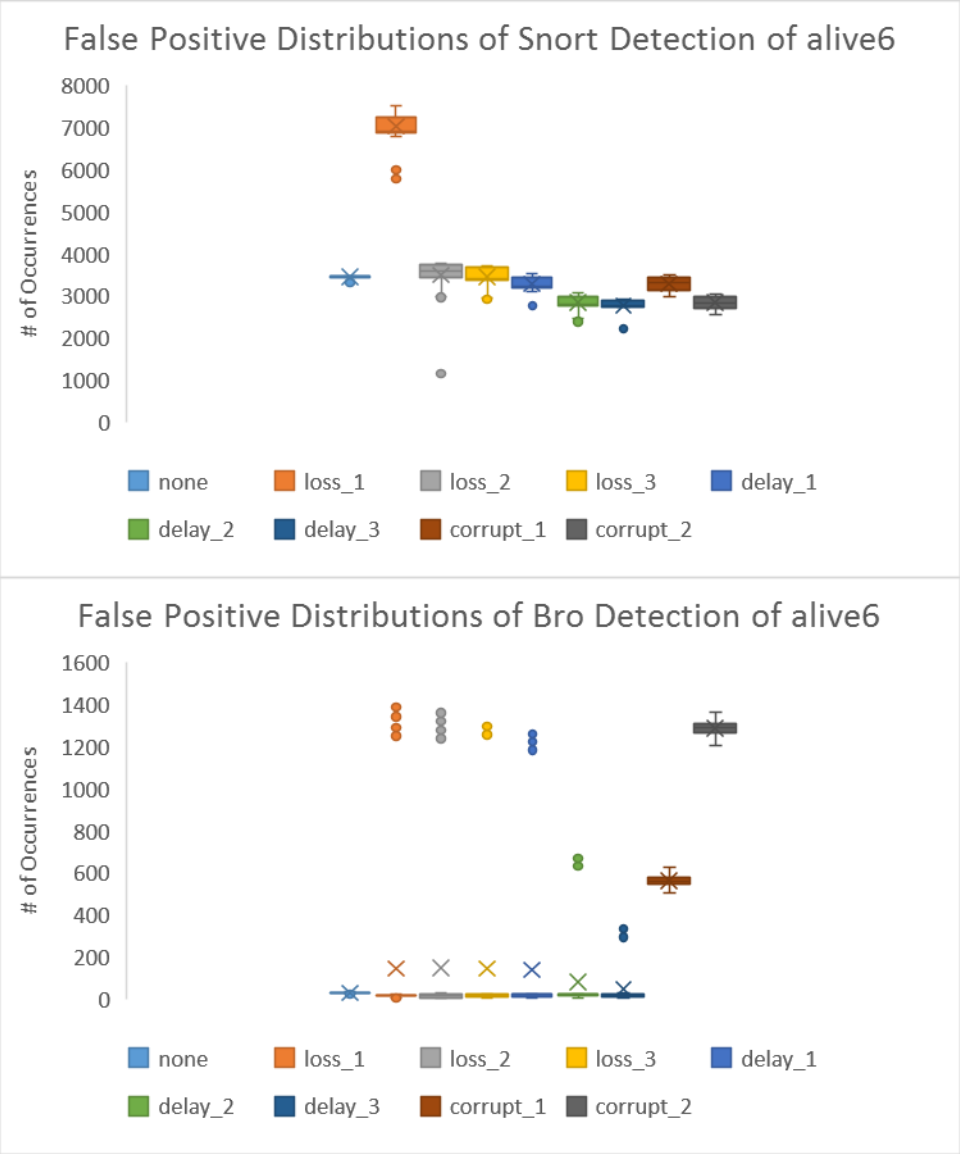

Figure 5.2: False positive distributions of both Snort and Bro detection of alive6 across 9 different network degradation scenarios

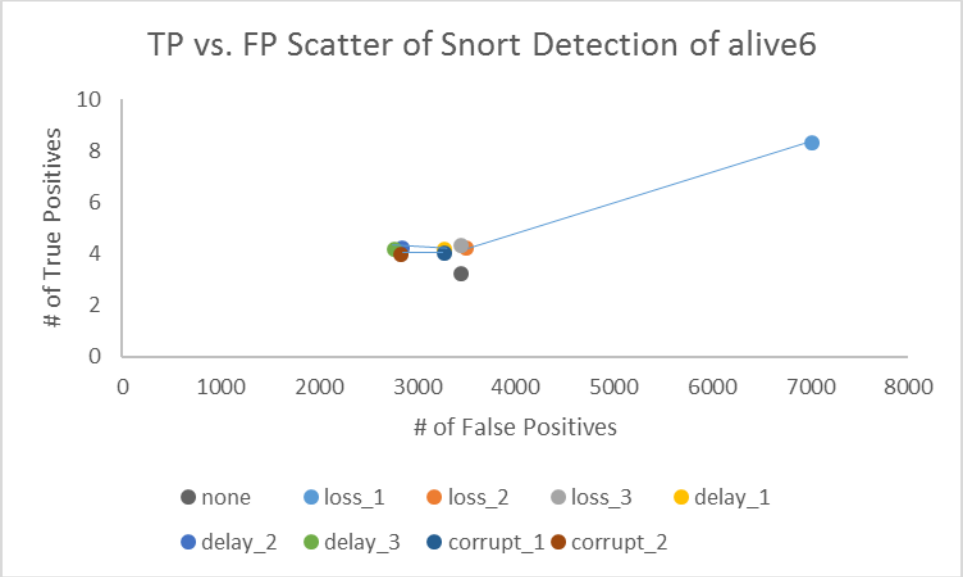

Figure 5.3: ROC-like scatter plot of Snort detection of alive6 with line segments connecting points of similar scenarios

From Figure 5.3, the more effective scenarios are found in the cluster consisting of delay_2, delay_3, and corrupt_2. The high false positive rates of the packet loss scenarios make their statistical results less favorable for security administrators.

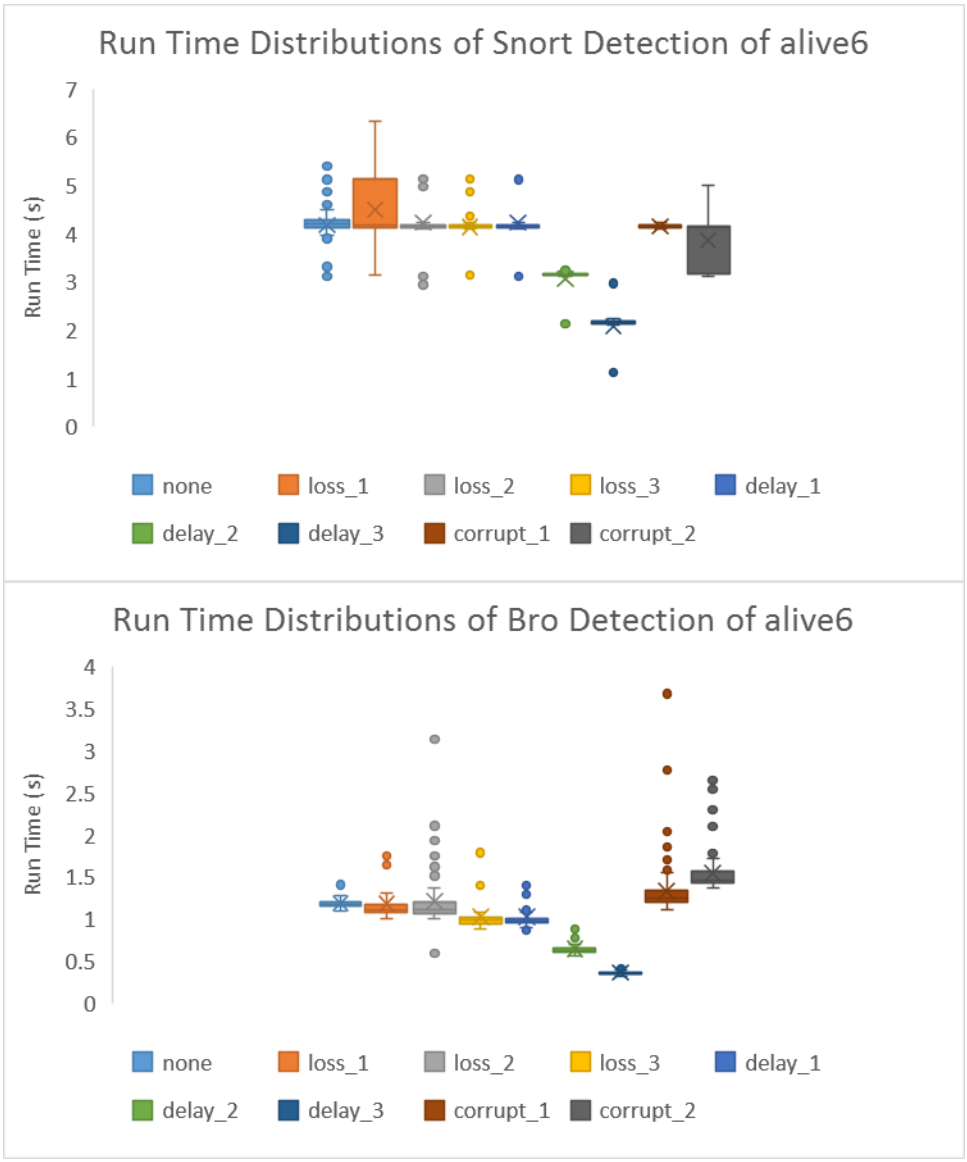

Figure 5.4: Run time distributions of both Snort and Bro detection of alive6 across 9 different network degradation scenarios

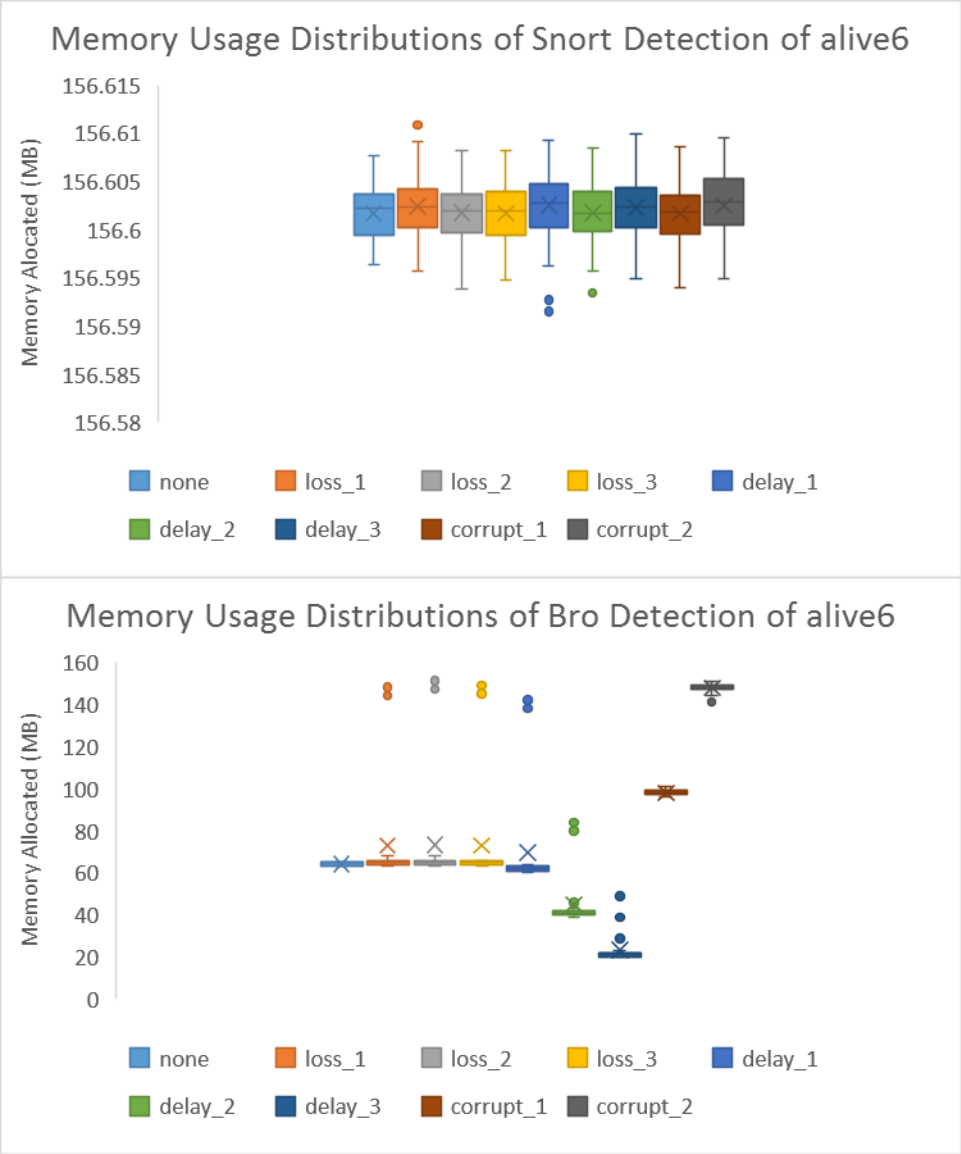

Figure 5.5: Memory usage distributions of both Snort and Bro detection of alive6 across 9 different network degradation scenarios

Snort and Bro roughly correlate in run times (see Figure 5.4) but not memory usage (see Figure 5.5). They both require more time to process packet corruption scenarios especially compared to processing packet delay scenarios. Run time decreases

as packet delay increases and increases as packet corruption increases. Better run times for both IDSs and memory usage for Bro for the packet delay scenarios can be explained by a built-in threshold for tracking IPv6 flows. Once the observation window has closed for a conversation, the IDSs ignore future packets from the same conversation.

Snort's memory usage remains consistently around 156 MB throughout the rest of this experiment. This is significantly more than Bro's memory usage, at least for alive6. This indicates that Snort's processes are fairly constant no matter the specific task it is performing, whereas Bro allocates only the memory necessary to create the various structures it uses to track traffic.

It should be noted that for alive6, Bro's run time results are roughly 1/3 to 1/5 of Snort's run time, but they both can be considered to achieve real-time processing for a 35 second long PCAP.

5.2.2: denial6

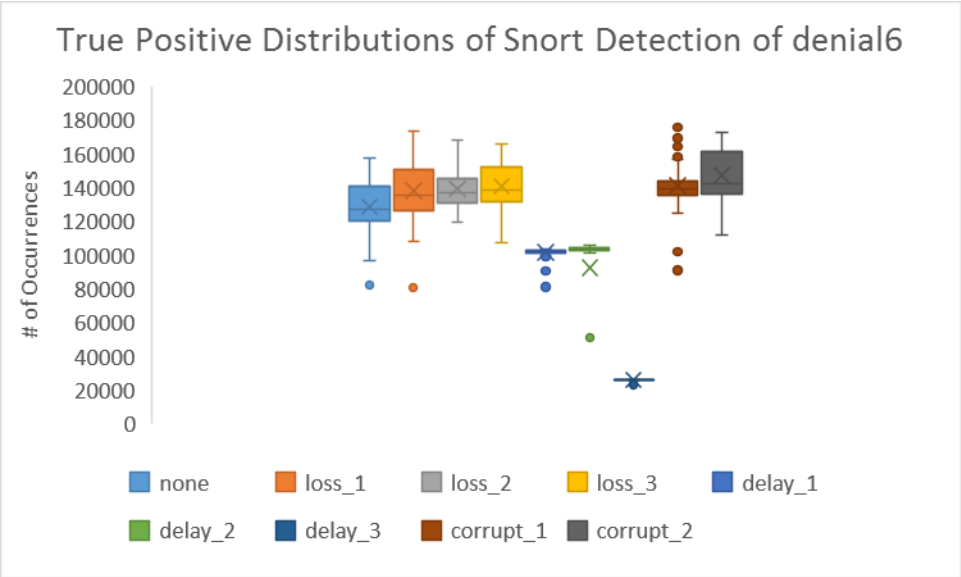

Figure 5.6: True positive distribution of Snort detection of denial6 across 9 different network degradation scenarios

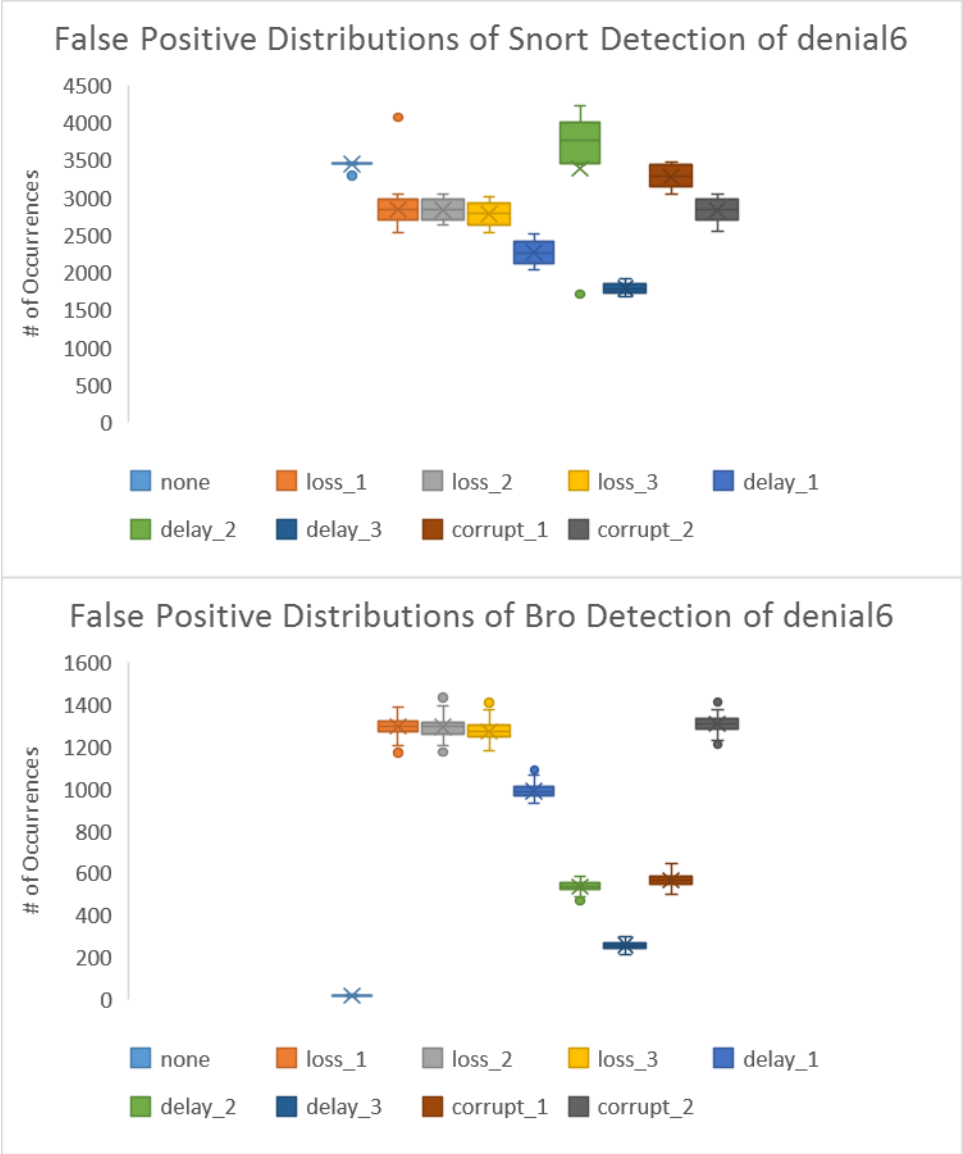

Figure 5.7: False positive distributions of both Snort and Bro detection of denial6 across 9 different network scenarios

Snort's true positive detection is fairly consistent except for the packet delay scenarios where there is a steep drop in detection (see Figure 5.6). The same downward trend manifests itself in the false positive detection of both Snort and Bro. There is also a

slight downward trend in false positive detection for both IDSs in the packet loss scenarios, which can be explained by slightly less packets being transmitted on the network in general.

Notably, Bro's false positive rate is near perfect for the scenario with no network degradation (see Figure 5.7). The fact that Bro then identifies 100s of false positives in the other scenarios seems to indicate errors directly caused by network degradation, whether corrupted headers, delayed conversation replies, or missing expected packets.

The true positive vs. false positive scatter plot for Snort seems to indicate that ideal detection conditions exist near the delay_1 scenario (see Figure 5.8).

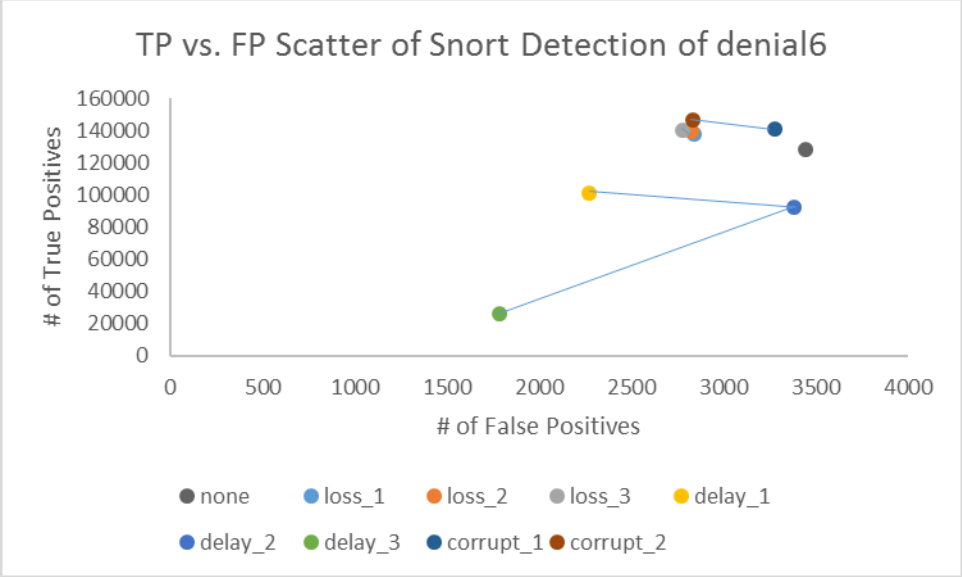

Figure 5.8: ROC-like scatter plot of Snort detection of denial6 with line segments connecting points of similar scenarios

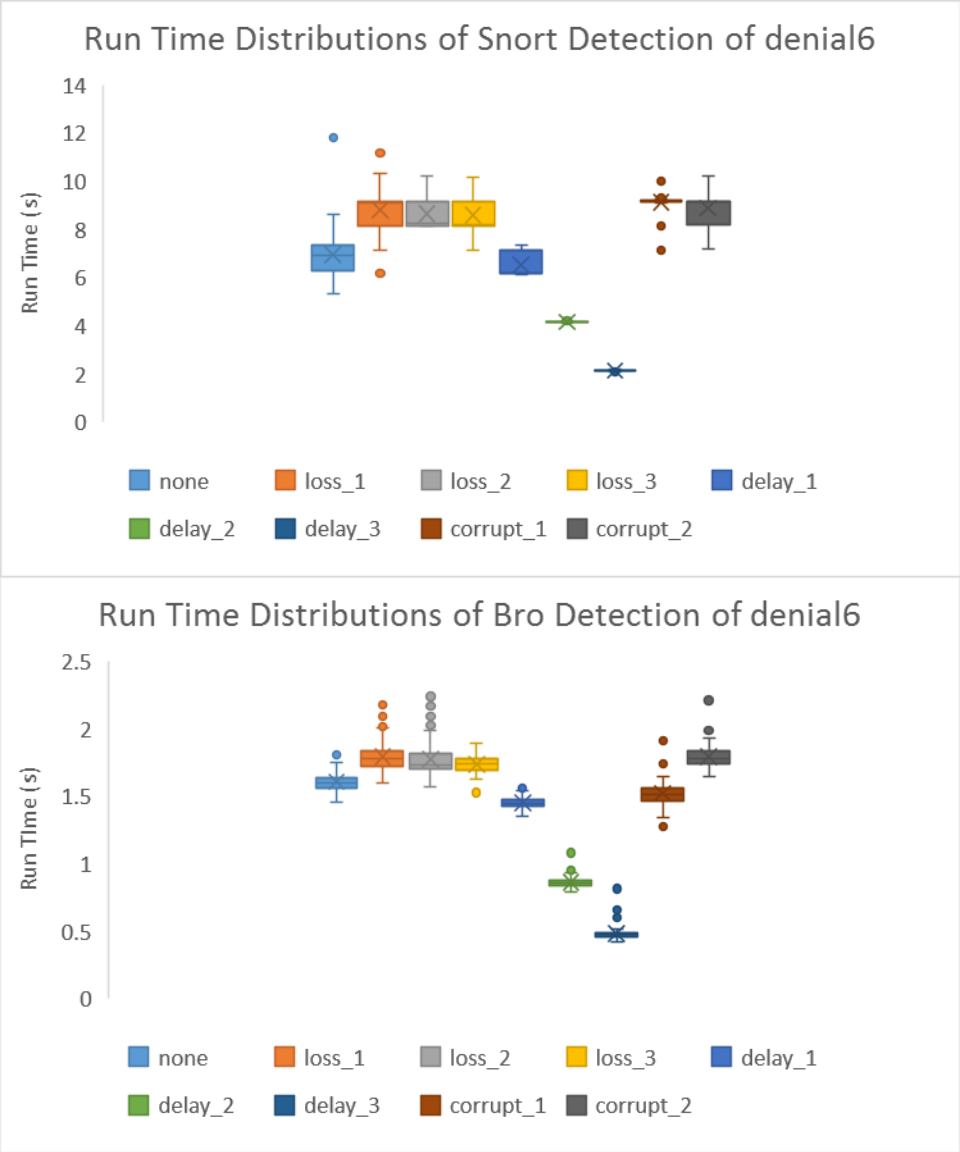

Figure 5.9: Run time distributions of both Snort and Bro detection of denial6 across 9 different network degradation scenarios

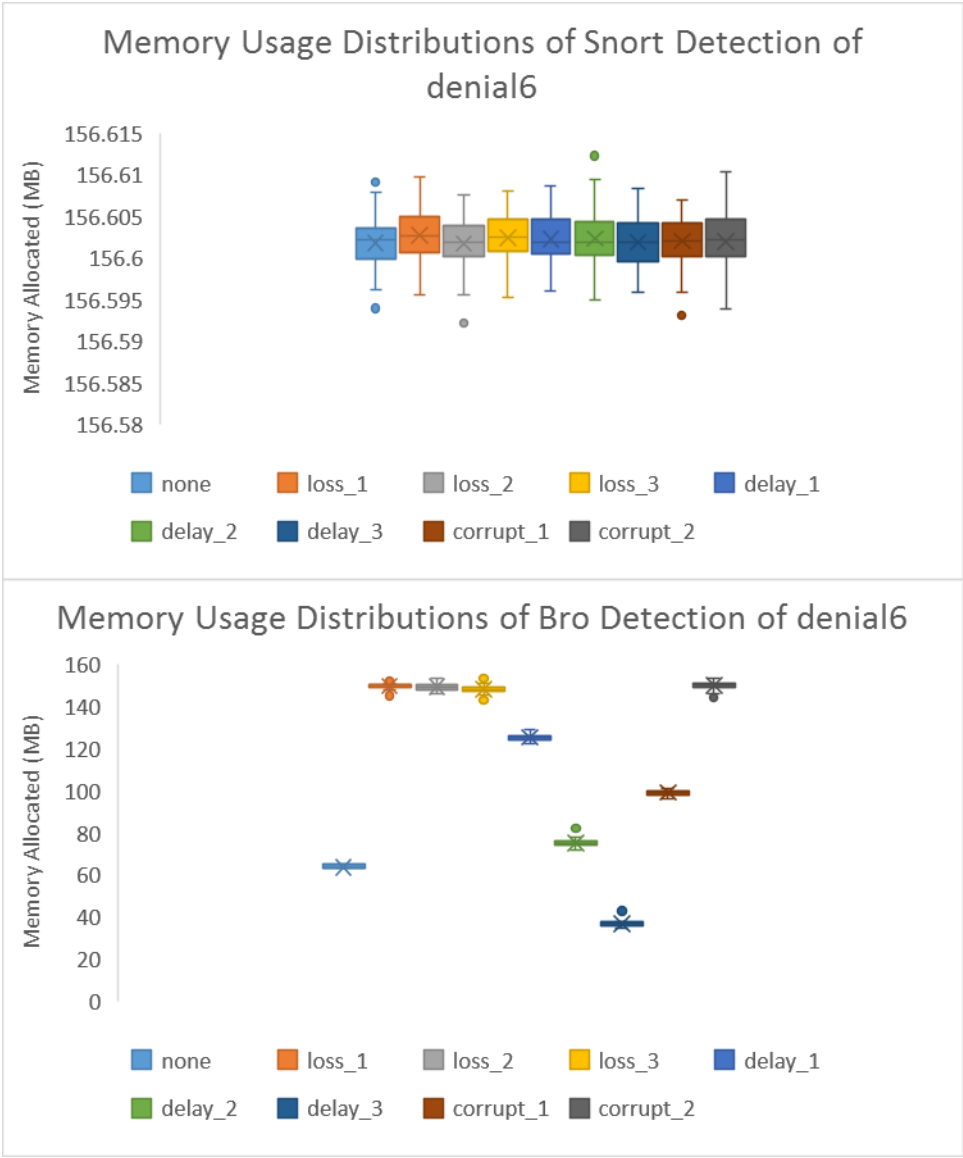

Figure 5.10: Memory usage distributions of both Snort and Bro detection of denial6 across 9 different network degradation scenarios

Similar to alive6 results, denial6 results show resource usage by both IDSs trend downward for packet delay scenarios and trend high for packet corruption scenarios (see Figure 5.9-5.10). Additionally, Snort's memory usage remains around 156 MB, and Bro

averages roughly 50 MB lower (see Figure 5.10). Interestingly, Bro's memory usage is much lower for the ideal network than for the degraded networks. Bro most likely requires more memory to track delayed, unexpected IPv6 conversations.

5.2.3: flood_advertise6

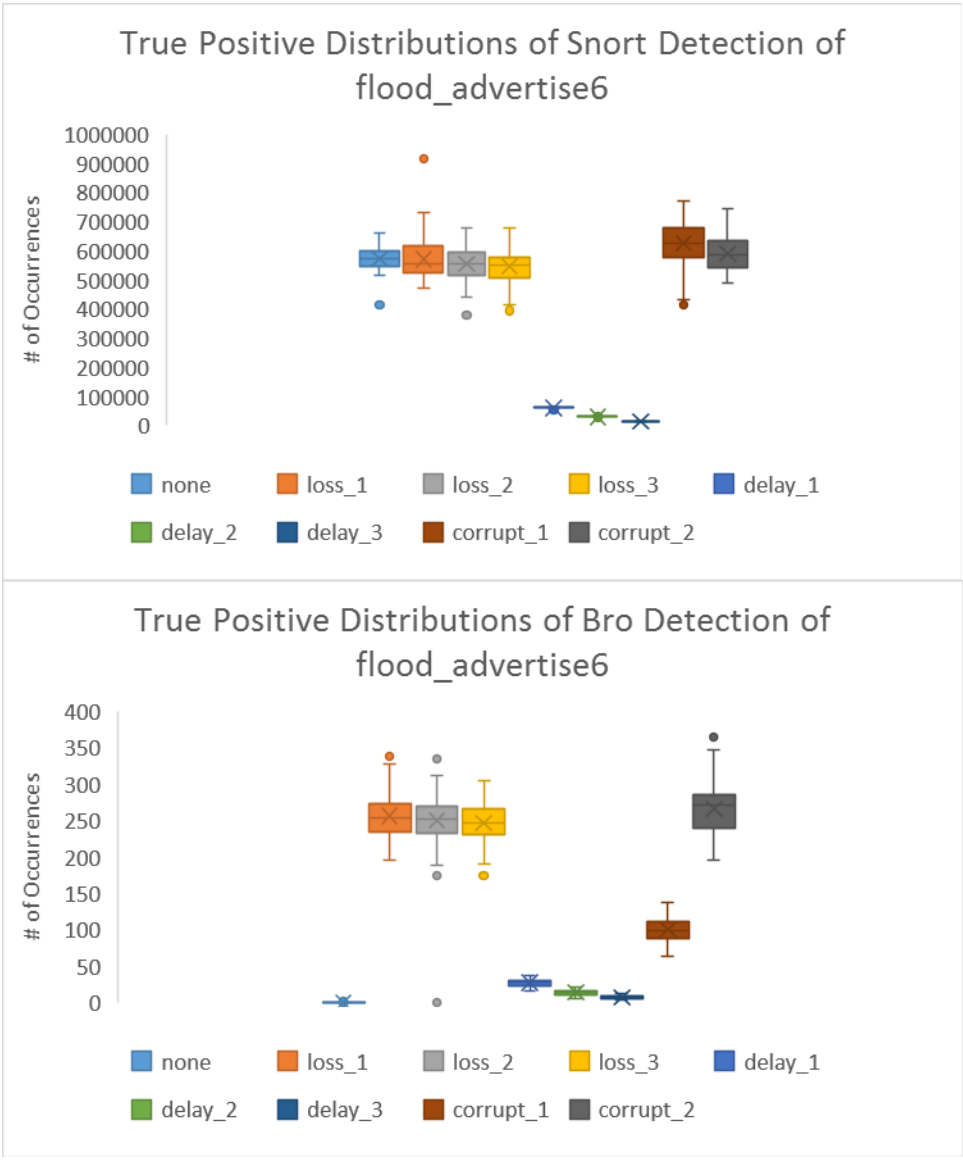

Figure 5.11: True positive distributions of both Snort and Bro detection of flood_advertise6 across 9 different network degradation scenarios

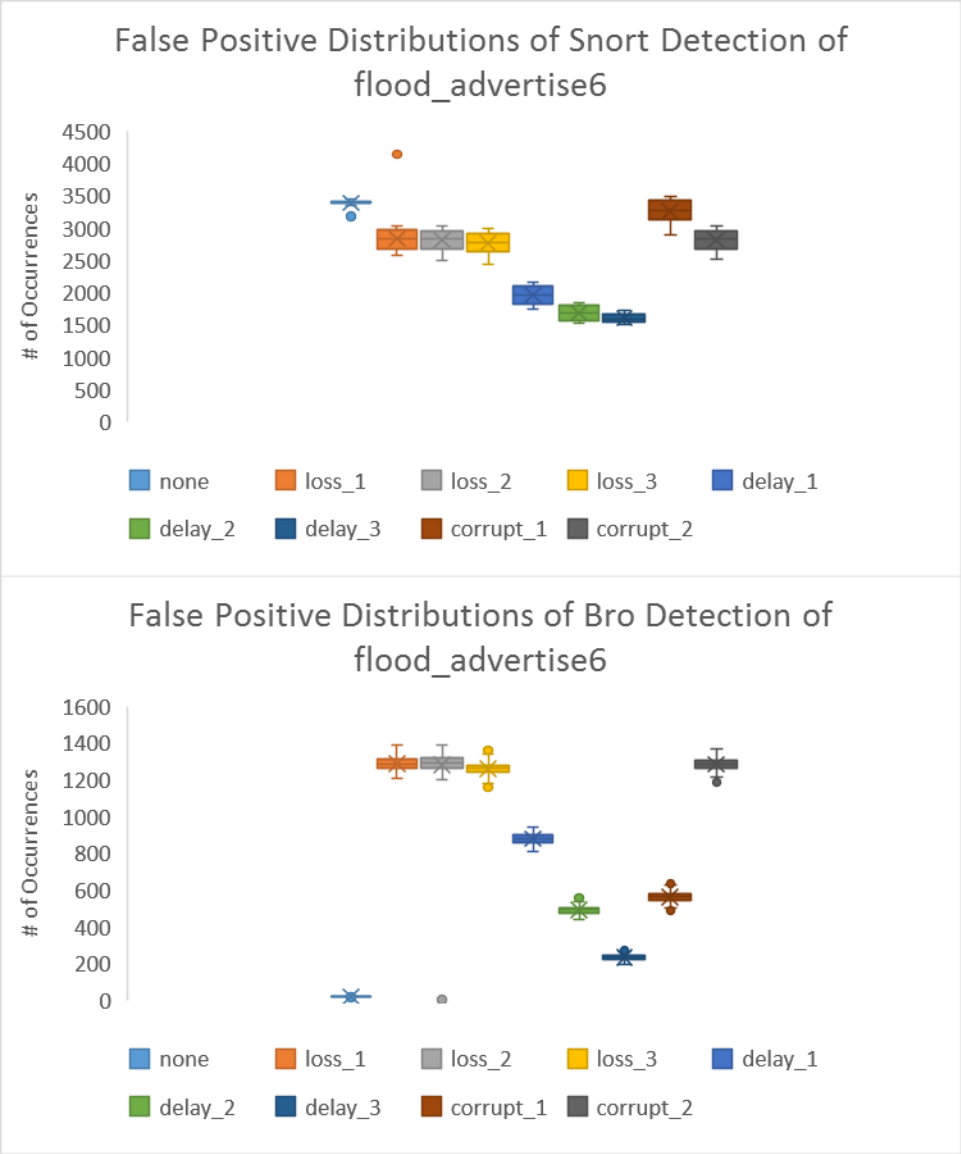

Figure 5.12: False positive distributions of Snort detection of flood_advertise6 across 9 different network degradation scenarios

The true positive detection results of both Snort and Bro trend similar to each other without considering the magnitude difference (see Figure 5.11) and similar among the network scenarios as attacks discussed above. One exception is Bro's low true and

false positive detection for the ideal network, which suggests that a majority of Bro's alerts indicate faulty network conditions rather than malicious activity (see Figure 5.11-12). Bro's false positive performance is consistent with Snort's, given that a disparity in number of observations does not necessarily indicate a disparity in performance.

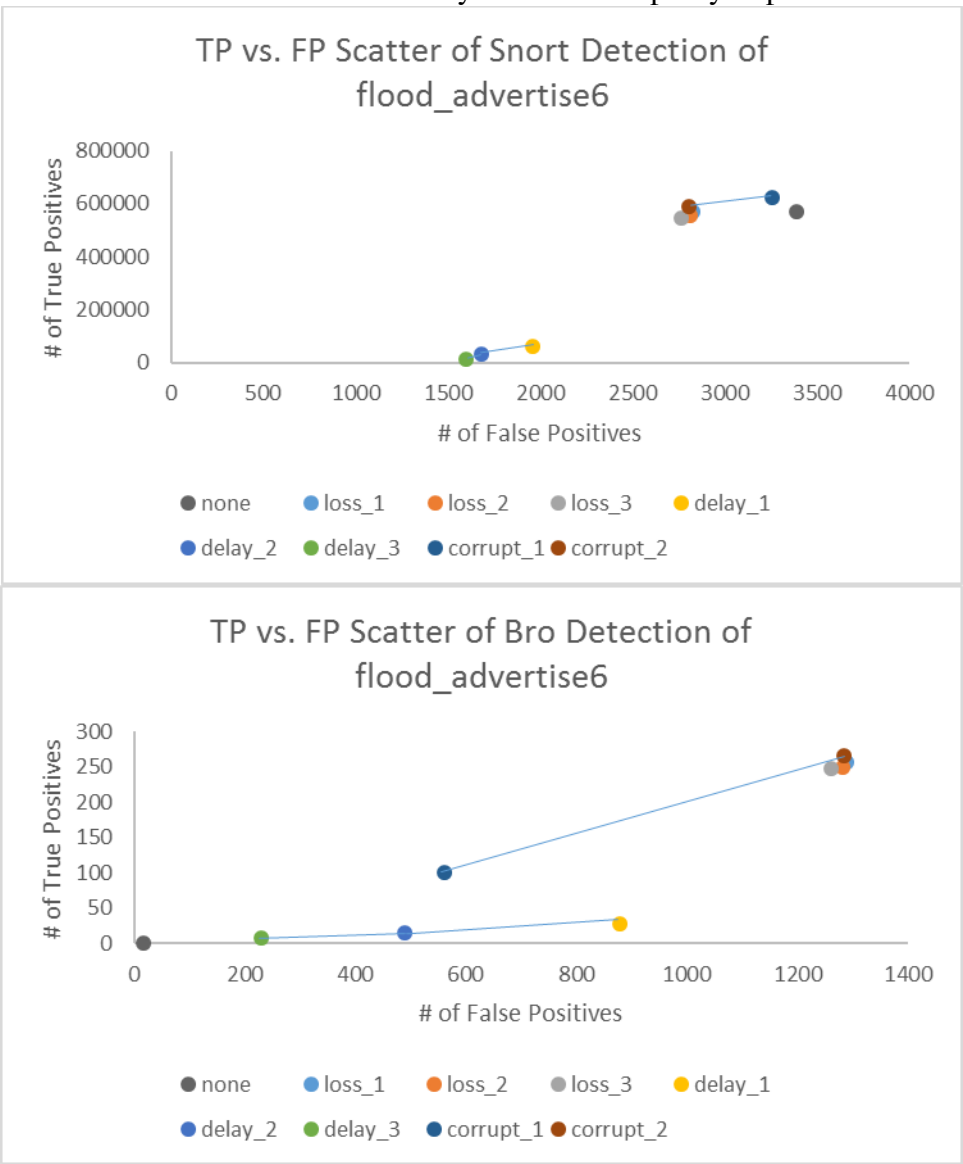

Figure 5.13: ROC-like scatter plots of both Snort and Bro detection of flood_advertise6 with line segments connecting points of similar scenarios

Because Snort and Bro performed closely to each other for flood_advertise6, their ROC-like scatter plots show scenario clusters in relatively the same locations. This similarity reflects consistent interpretations of the ground truth.

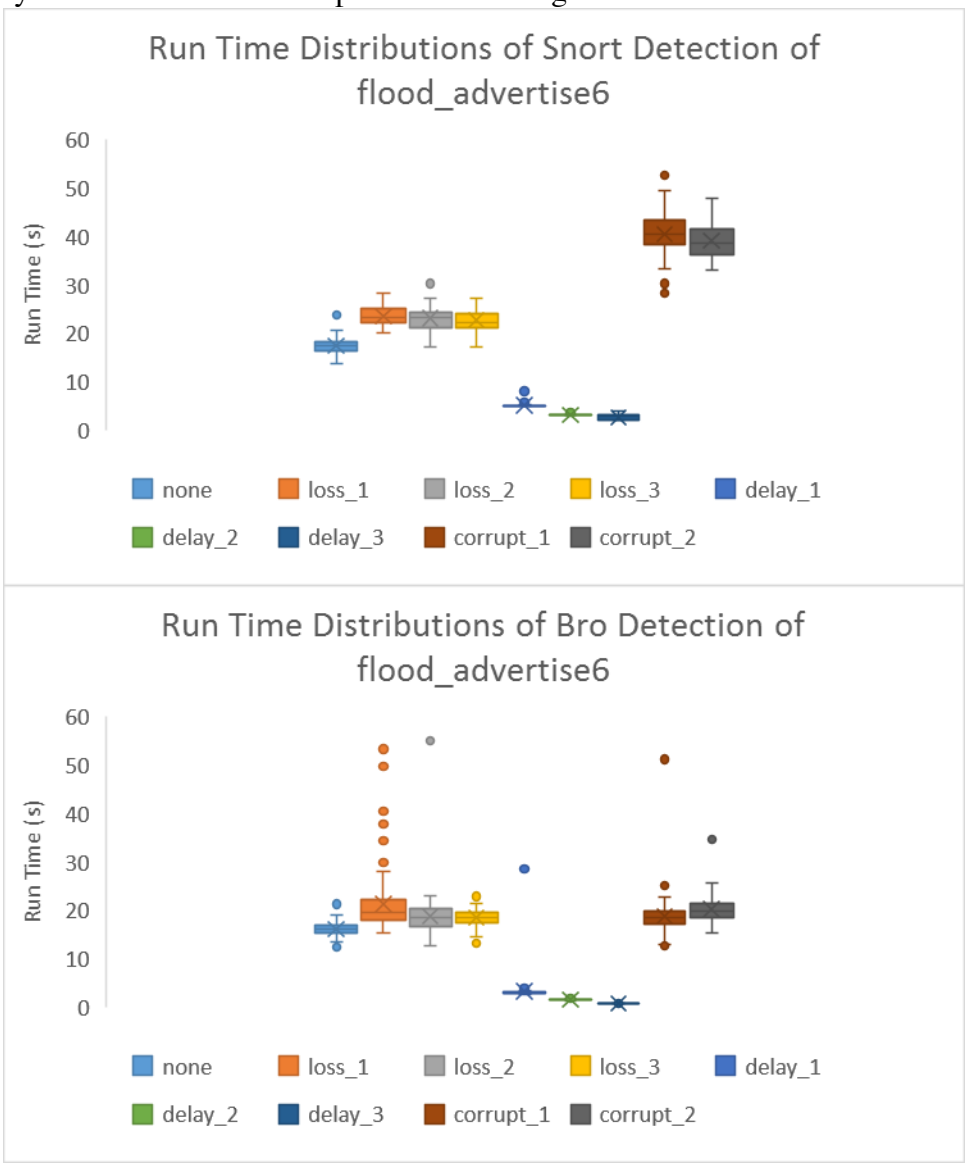

Figure 5.14: Run time distributions of both Snort and Bro detection of flood_advertise6 across 8 different network degradation scenarios

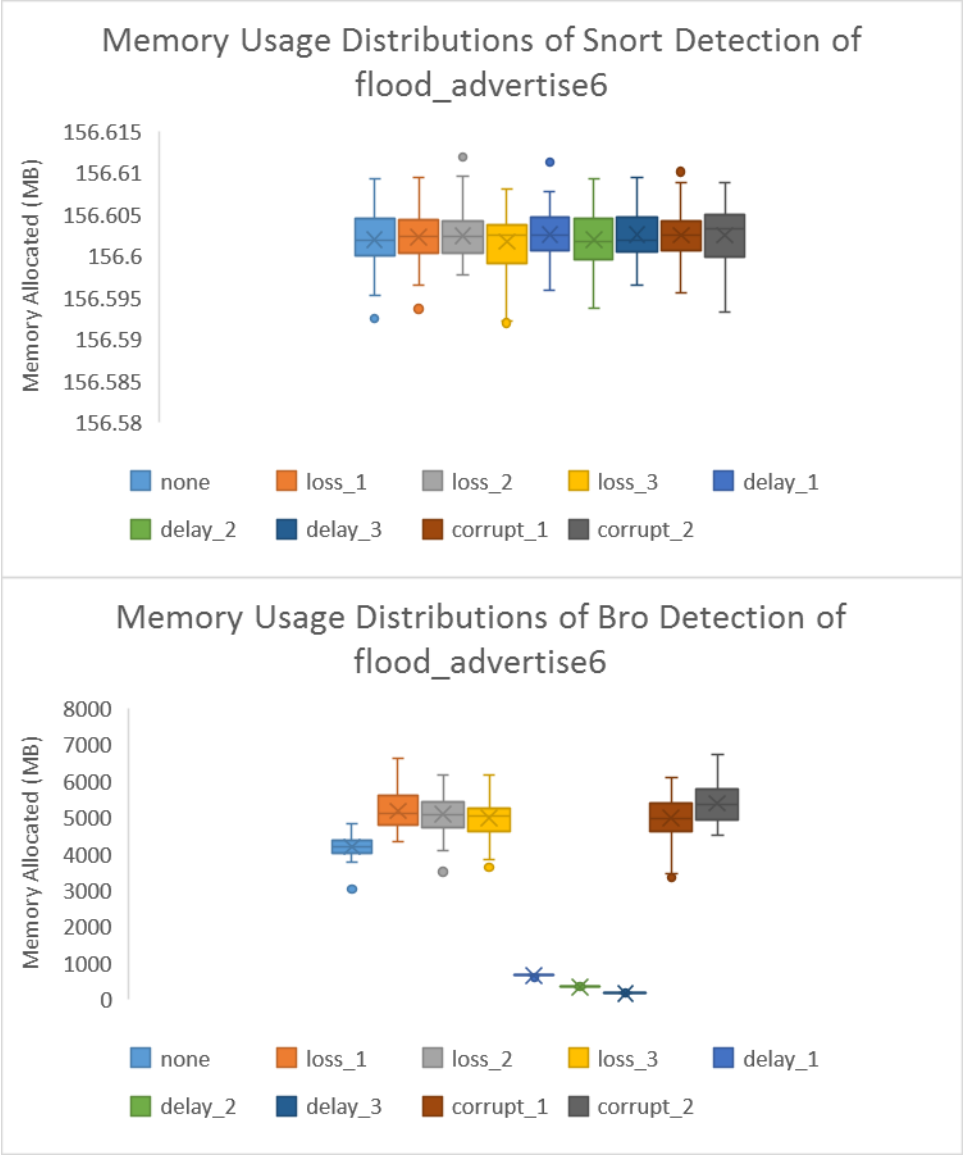

Figure 5.15: Memory usage distributions of both Snort and Bro detection of flood_advertise6 across 9 different network degradation scenarios

Resource usage across the scenarios and for both IDSs is similar to previous attacks. The delay scenarios require less scenarios as observation windows expire more frequently, corruption scenarios sometimes require more resources (see Figure 5.14-

5.15). For flood_advertise6, for run time results and Bro's memory usage results, the ideal network requires less resources to monitor than the packet loss and packet corruption scenarios. It makes sense that, in general, as network conditions deteriorate, IDSs must work harder to reassemble traffic and make decisions. Finally, as expected, Snort allocates approximately 156 MB for all trials with even very few outliers, however, Bro allocated 5-6 GB for most flood_advertise6 scenarios! This is because Bro builds data structures that scale directly with the number of unique flows, such as conn.log.

5.2.4: flood_solicitate6

The true positive detection results by Snort of flood_solicitate6 are among the more unique distributions (Figure 5.16). Across all scenarios, Snort detects 6144 malicious activities or patterns. This unique event is hard to explain as even packet loss should cause a slight drop in detection from the ideal network. However, Figure 5.17 shows similar false positive results to the rest of the attacks tested.

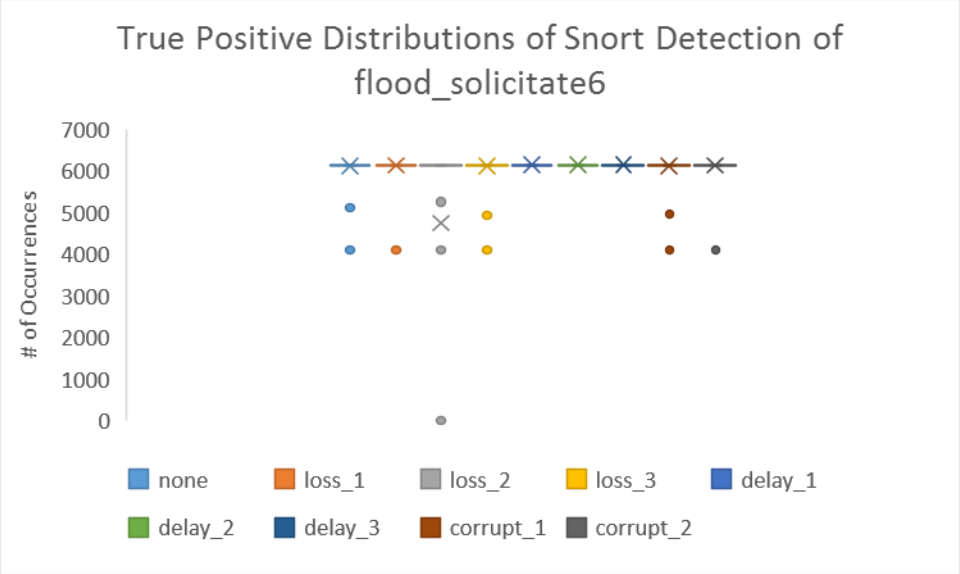

Figure 5.16: True positive distributions of Snort detection of flood_solicitate6 across 9 different network degradation scenarios

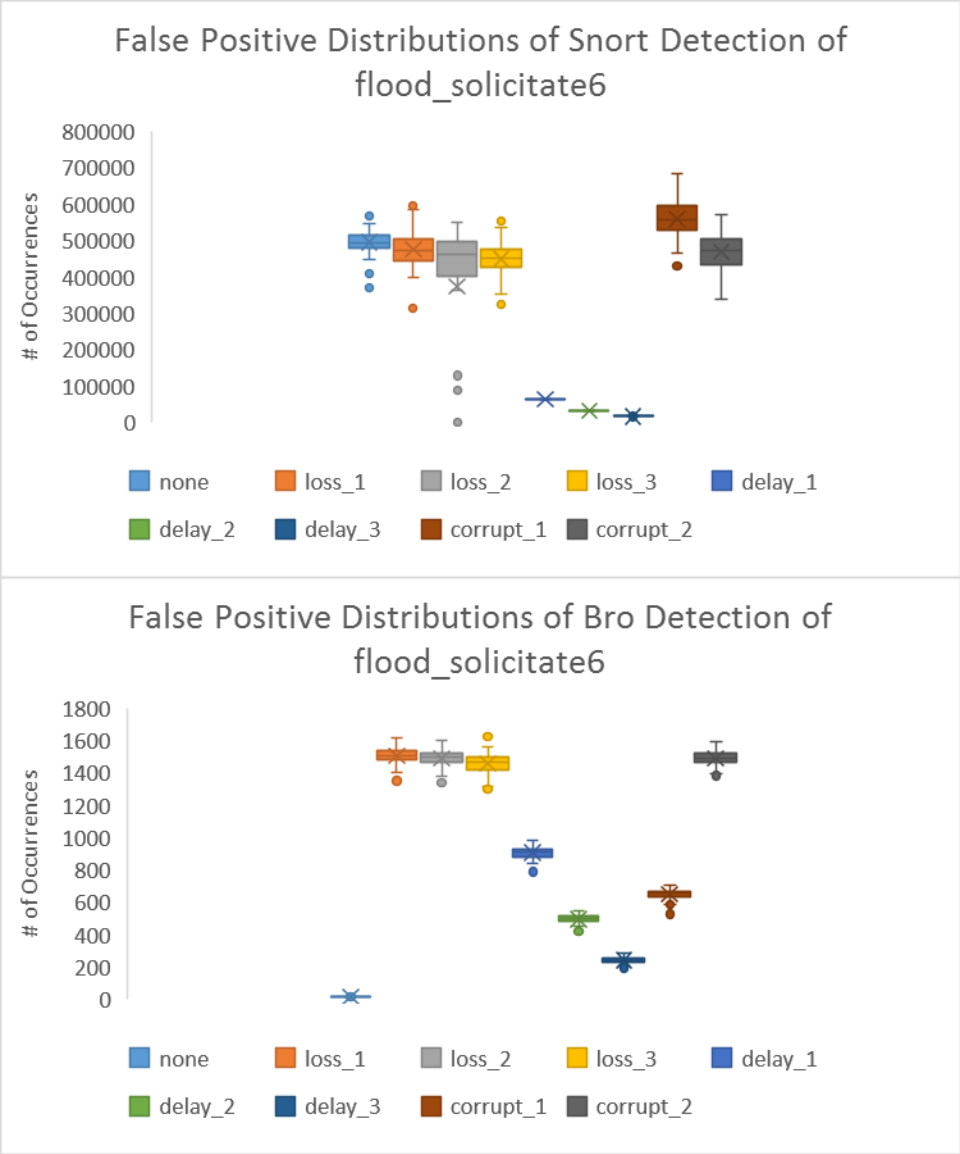

Figure 5.17: False positive distributions of both Snort and Bro detection of flood_solicitate6 across 9 different network degradation scenarios

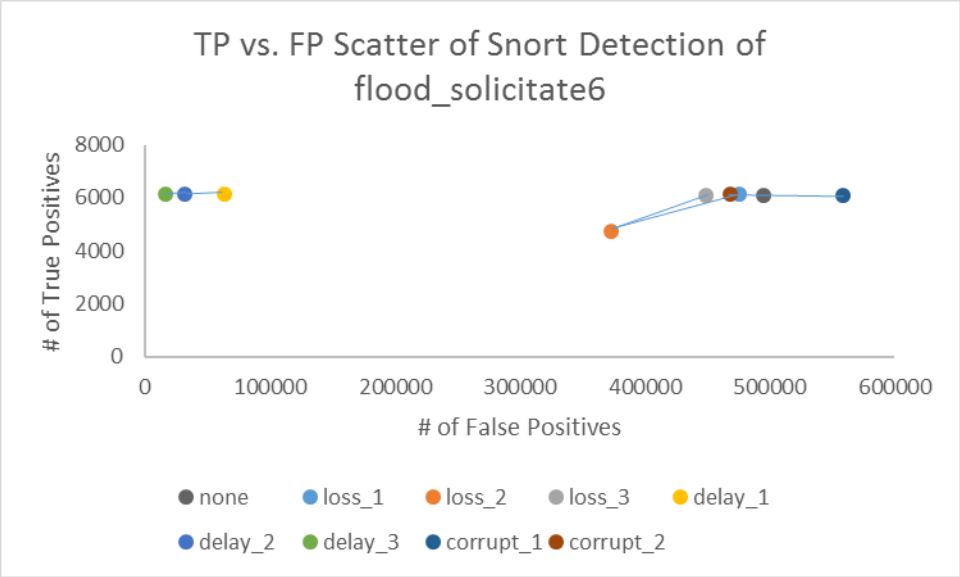

Figure 5.18: ROC-like scatter plot of Snort detection of flood_solicitate6 with line segments connecting points of similar scenarios

The ROC-like scatter plot of Snort's detection of flood_solicitate6 clearly shows the best detection conditions exist near the cluster of packet delay scenarios (see Figure 5.18). This cluster is followed by the packet loss cluster, and finally the packet corruption scenario.

One caveat when interpreting the false positive distributions of detection of flood_solicitate6 is how I define a false positive in my experiment code. The code identifies any alerts which do not contain the IPv6 address of either ATTACKER or TARGET as a false positive. However, as described in §3.4.2, flood_solicitate6 floods a victim's network with packets to and from random IPv6 addresses. The observation of and alerting on these packets could be considered a true positive. As stated in §3.4.2, flood_solicitate6 was able to generate approximately 660,000 packets for the 10 second duration of each DoS attack before network degradation is accounted for.

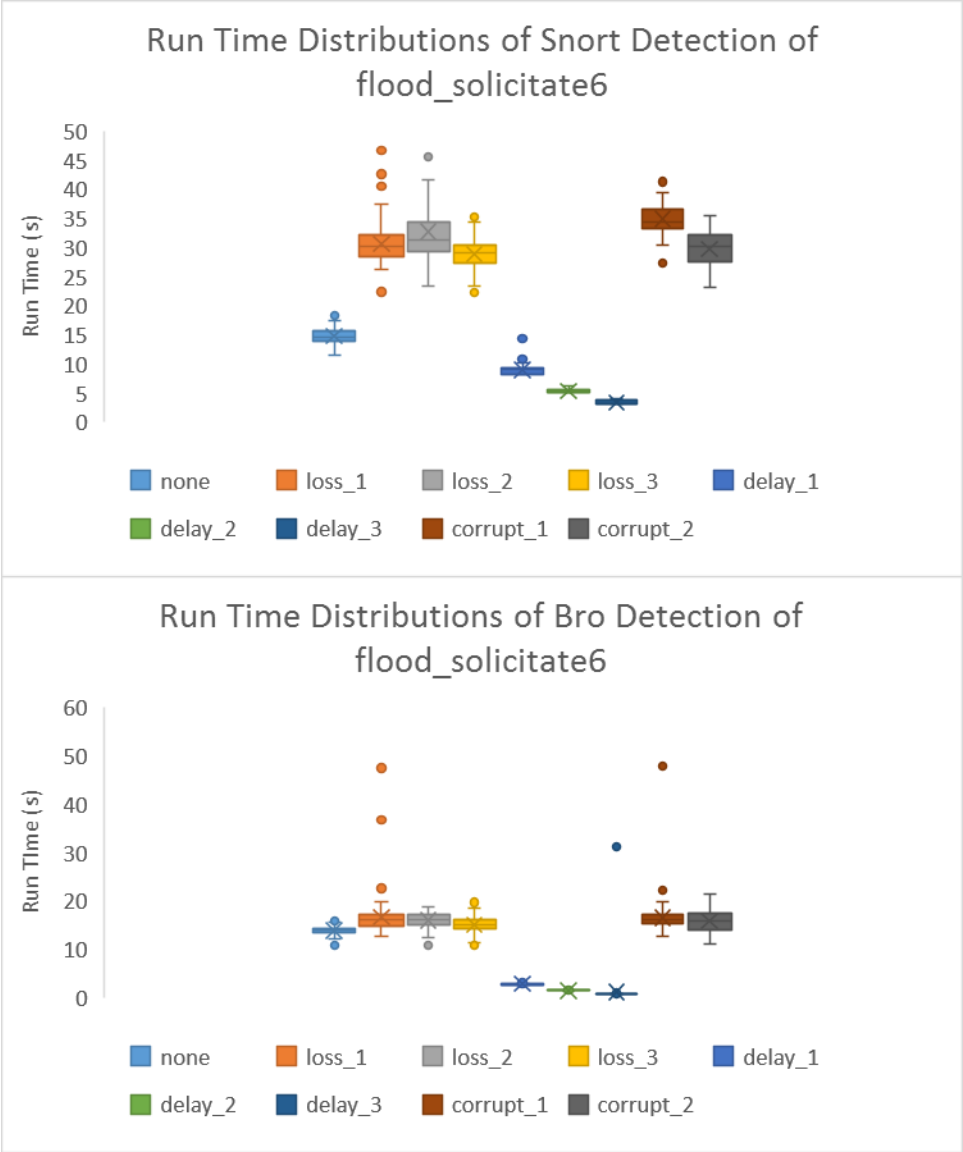

Figure 5.19: Run time distributions of both Snort and Bro detection of flood_solicitate6 across 9 different network degradation scenarios

Snort and Bro both processed the packet delay scenarios much more quickly than the rest of them, and Snort required approximately 30 seconds to process the packet loss and packet corruption scenarios compared to only 15 for the ideal network (see Figure

5.19). Overall, Bro processed flood_solicitate6 PCAPs in roughly half the time. At ~30 seconds of run time for most scenarios but some outliers exceeding 35 seconds, Snort is close to being able to process packets in real-time. Bro, on the other hand, is well below the 35 second threshold.

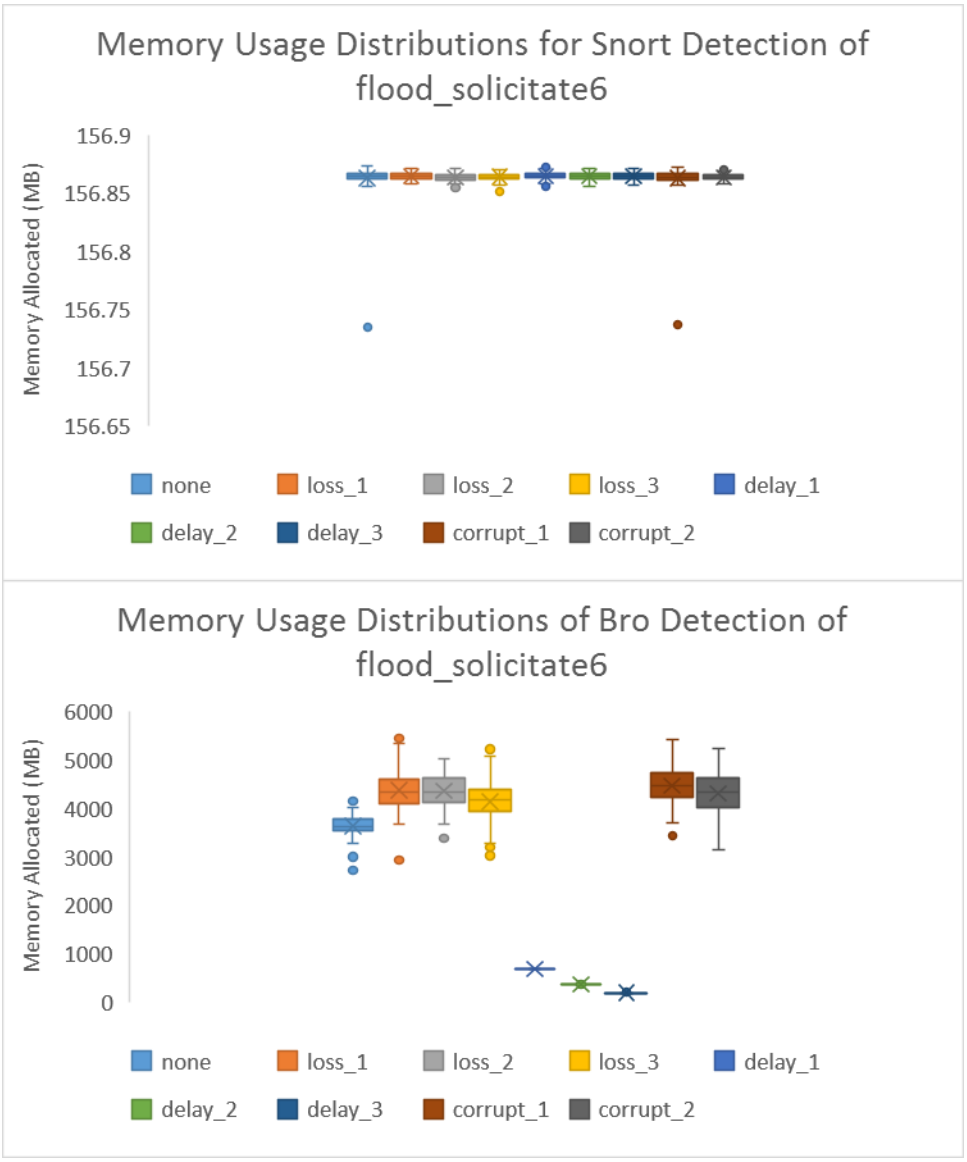

Figure 5.20: Memory usage distributions of both Snort and Bro detection of flood_solicitate6 across 9 different network degradation scenarios

The cost of quickly processing so many connections due to the DoS nature of flood_solicitate6 was magnitudes of difference in memory allocation. Bro allocated ~4.5 GB compared to Snort's consistent 156 MB (see Figure 5.20). Again, this can be explained by Bro's data structures that it uses to build a large, informed perspective of the traffic which scales directly with the number of unique flows it observes in a short time. Because flood_solicitate6 creates ICMPv6 packets to and from random addresses, Bro requires large amounts of memory to successfully track them.

Once again, Bro's lower memory allocation for the packet delay scenarios can be explained by a limited observation window that expires when Bro does not observe enough activity in a particular flow. The effectiveness of DoS attacks rely on the rapid fire approach of flooding work-intensive packets and significant packet delay can lessen the damaging effects.

5.2.5: covert_send6

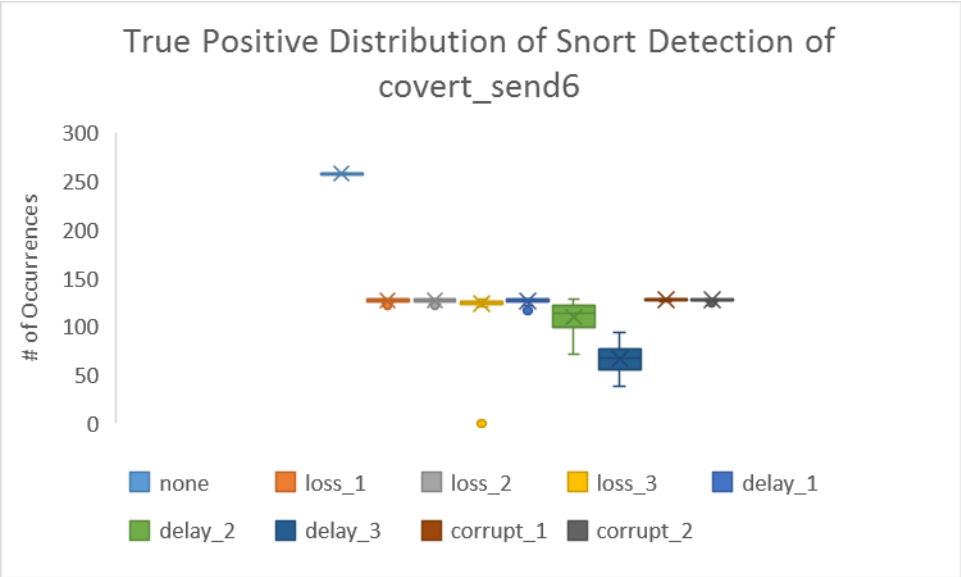

Figure 5.21: True positive distributions of Snort detection of covert_send6 across 9 different network degradation scenarios

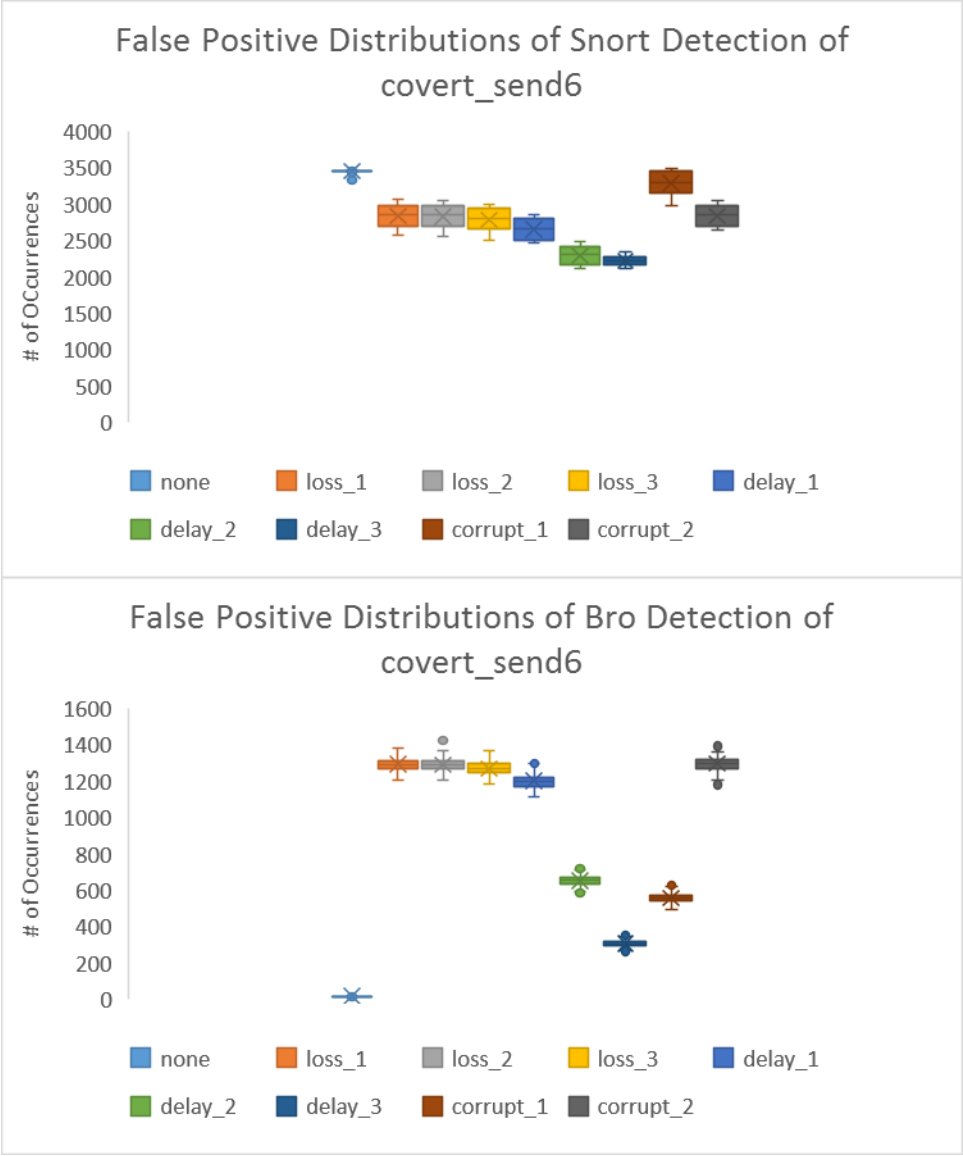

Figure 5.22: False positive distributions of both Snort and Bro detection of covert_send6 across 9 different network degradation scenarios

Snort's true positive detection of covert_send6 demonstrates its excellent accuracy as a detection engine. To covertly send the Conficker binary in ICMPv6 headers, covert_send6 transmitted 64 packets. As an ICMPv6 Echo Request, each of the

64 packets should receive a symmetrical reply by TARGET. The interquartile range (IQR) of Snort's true positive detection is narrow and the median centers on 128 packets (see Figure 5.21). This indicates that Snort correctly identifies almost all covert_send6 packets, except for those in packet delay scenarios, as symmetric replies are delayed.

Figure 5.22 shows Snort and Bro detecting similar relative numbers of alerts across the scenarios with Bro's detection between 1/3 and 1/2 of Snort's number of alerts. There is no explanation for Bro's false positive detection of the ideal network being near non-existent.

The ROC-like scatter plot for Snort's detection of covert_send6 shows the ideal detection scenario being either delay_2 or none (ideal network) (see Figure 5.23).

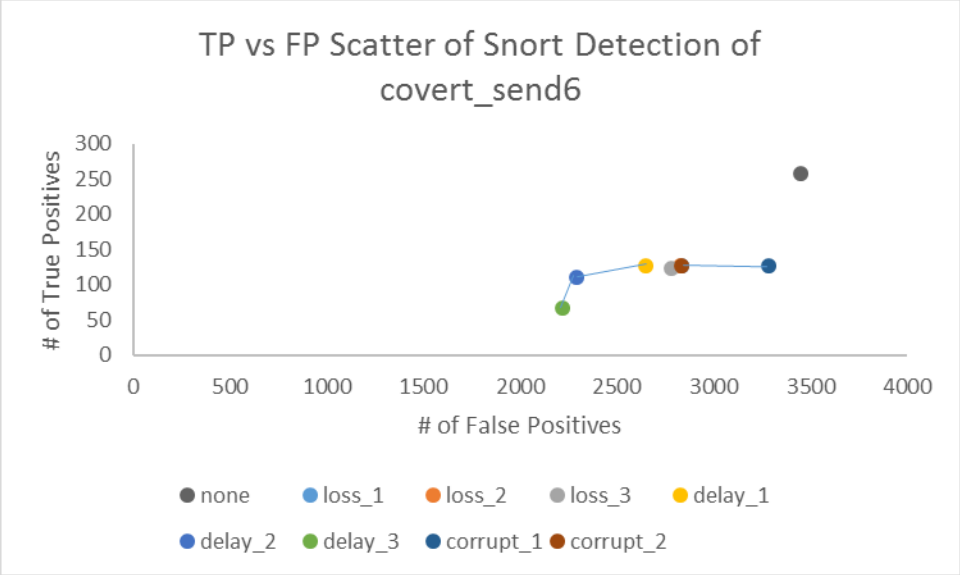

Figure 5.23: ROC-like scatter plot of Snort detection of covert_send6 with line segments connecting points of similar scenarios

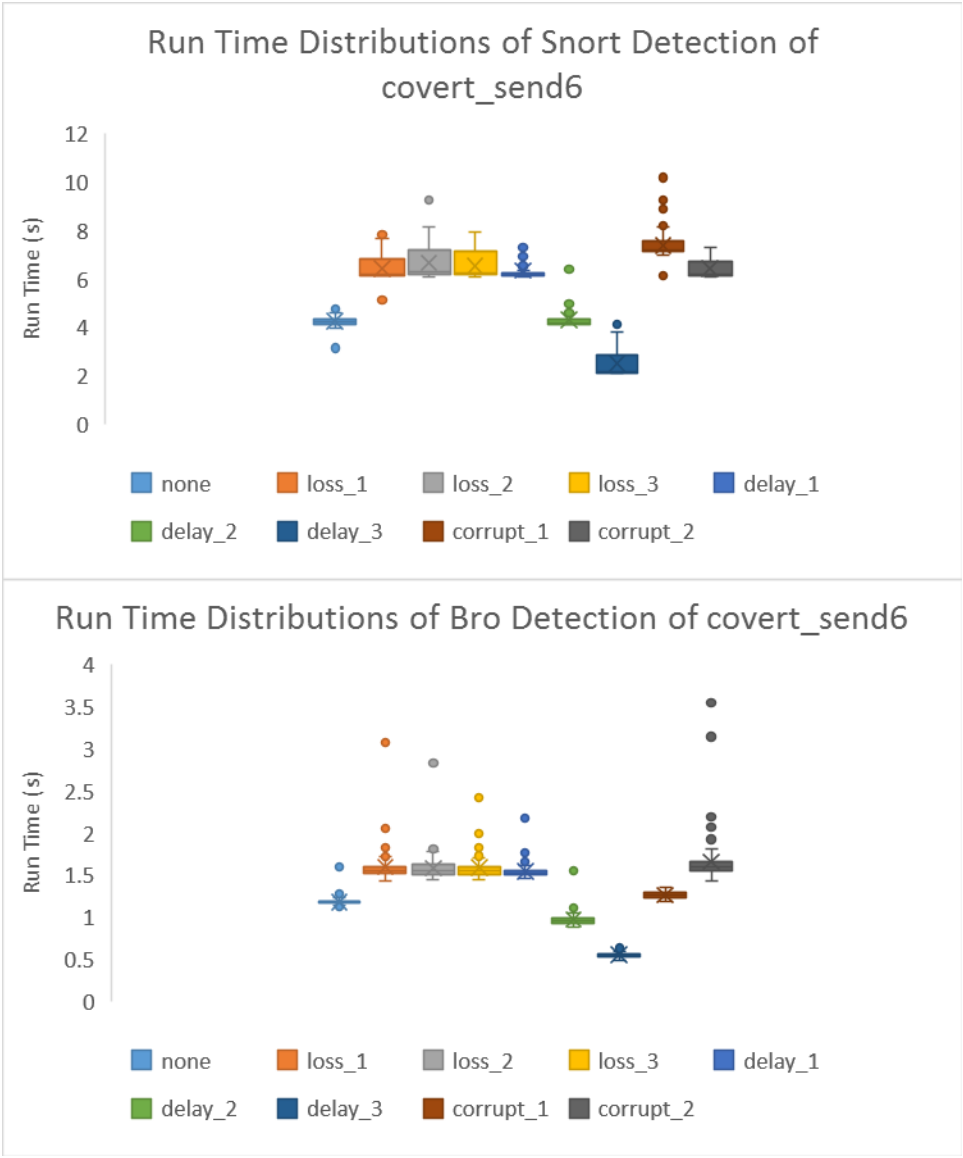

Figure 5.24: Run time distributions of both Snort and Bro detection of covert_send6 across 9 different network degradation scenarios

The IQR of the run time distributions of each scenario for both Snort and Bro are narrow (see Figure 5.24). Both Snort and Bro are well below the 35 second threshold for a real-time detection system. Snort and Bro detection of covert_send6 result in rather

favorable resource usage because covert_send6 is not a voluminous attack like the 3 DoS attacks mentioned prior. Bro requires roughly 1/4 of the time that Snort requires. As always, Snort allocates its 156 MB of memory, while Bro allocates less (see Figure 5.25).

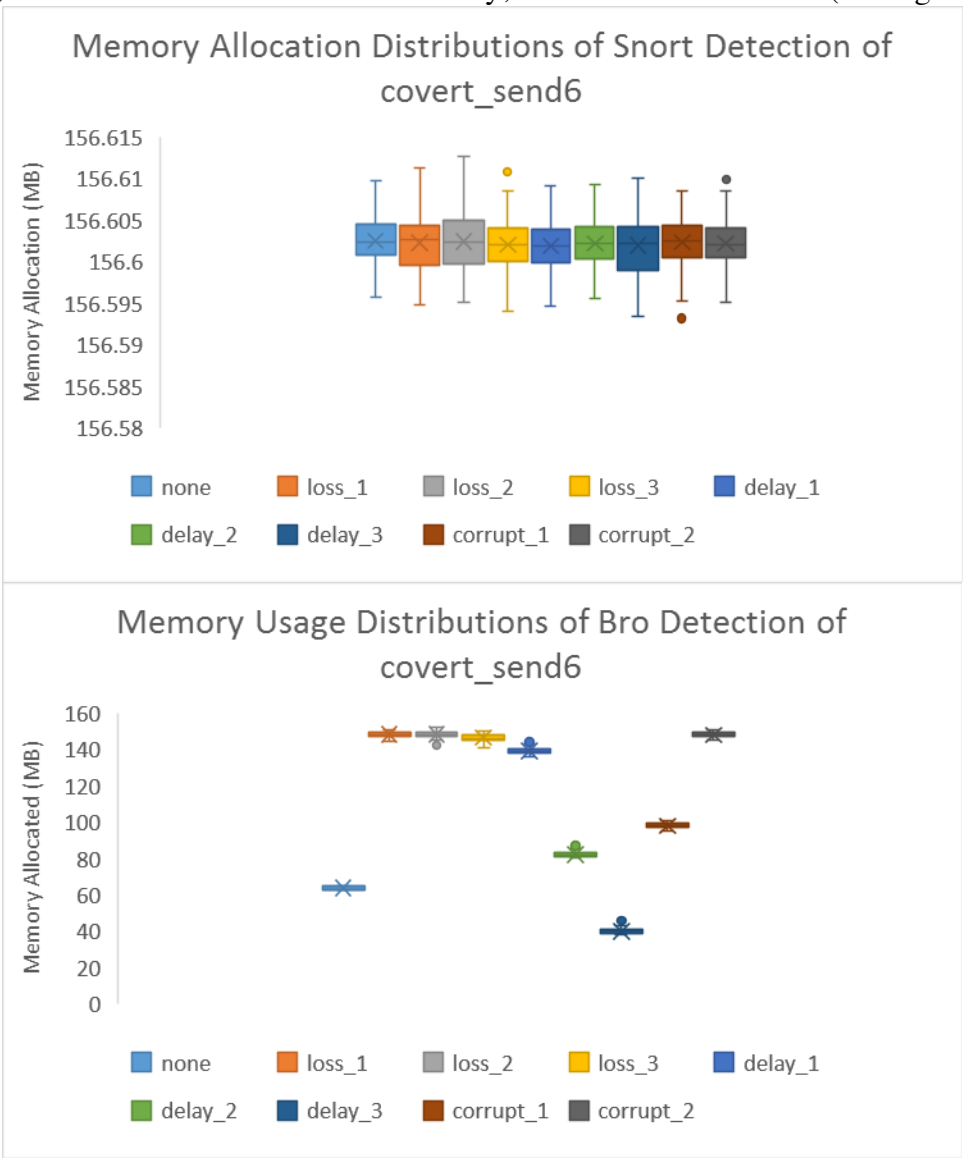

Figure 5.25: Memory usage distributions of both Snort and Bro detection of covert_send6 across 9 different network degradation scenarios

5.2.6: exploit6

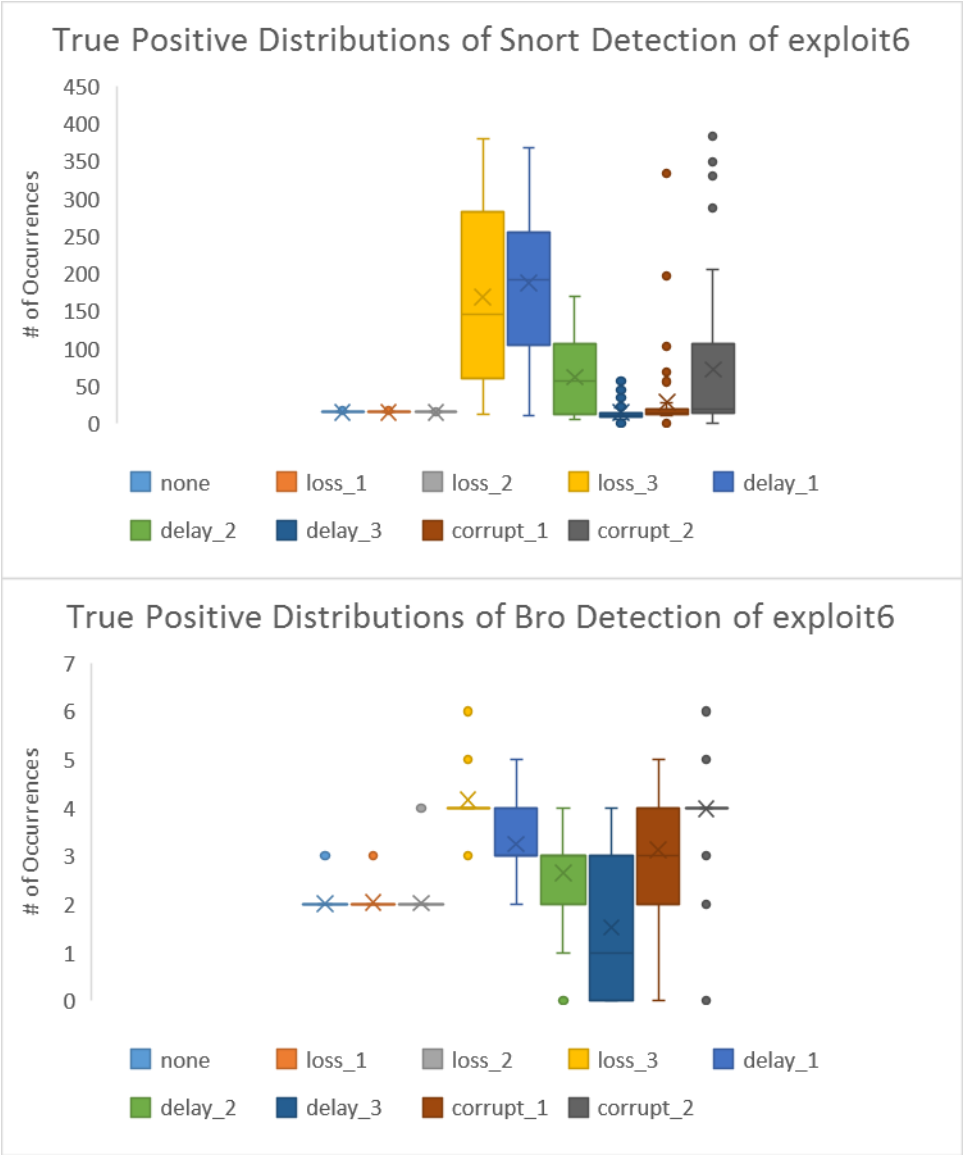

Figure 5.26: True positive distributions of Snort detection of exploit6 across 9 different network degradation scenarios

Bro's extensive header checking capabilities detect true positives for the CVE-based attacks which abuse ICMPv6 headers (see Figure 5.26). Packet delay introduces

the most variance in the true positive distributions for both IDSs. The IQR of the false positive distributions for both IDSs are minimal. Because exploit6 is not a DoS attack and the benign traffic does not differ significantly from other scenarios and trials, the false positive detection does not vary much within each scenario (see Figure 5.27).

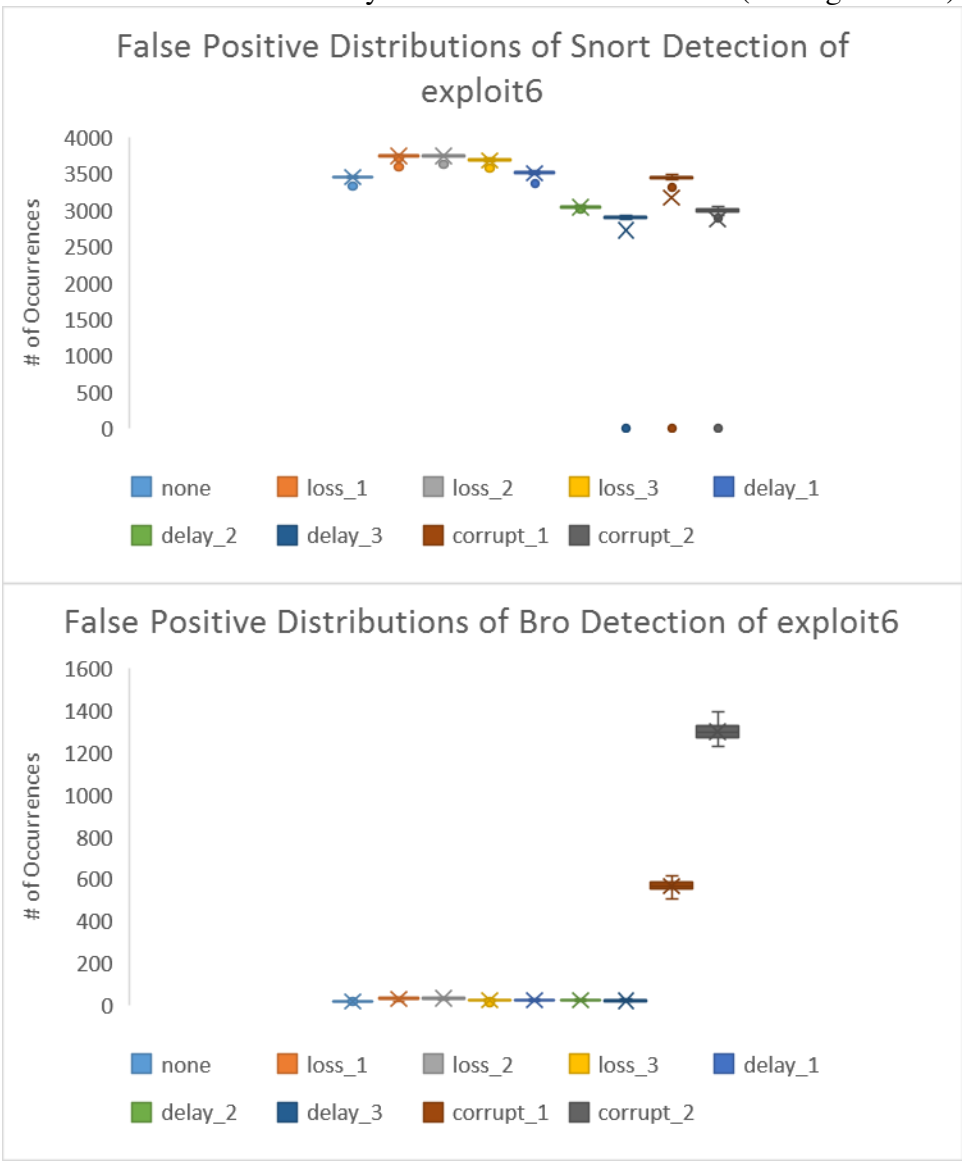

Figure 5.27: False positive distributions of both Snort and Bro detection of exploit6 across 9 different network degradation scenarios

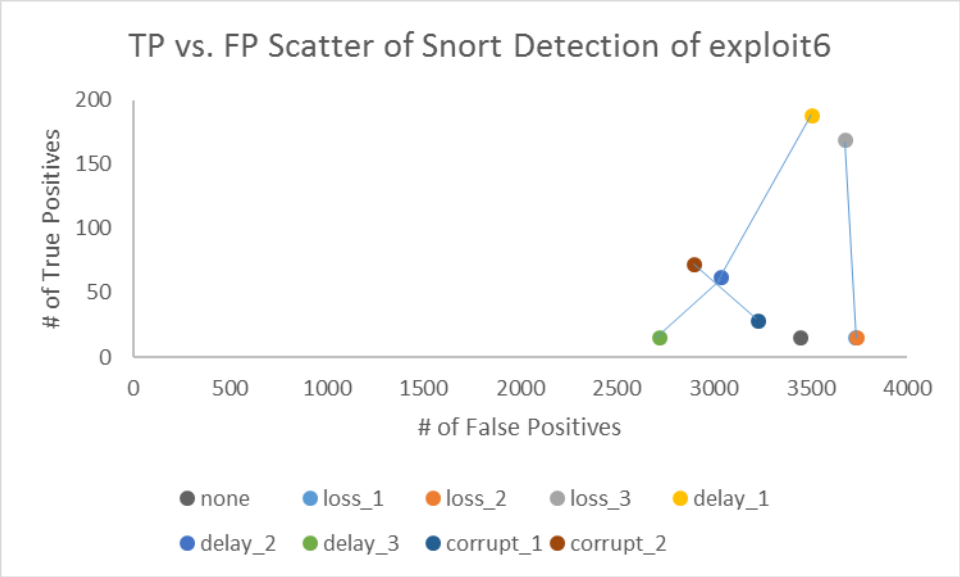

Figure 5.28: ROC-like scatter plot of Snort detection of exploit6 with line segments connecting points of similar scenarios

Because Snort's header checking rules are not as detailed as Bro's are, corrupt_2 is one of the better scenarios for Snort's detection based on it's ROC-like scatter plot (see Figure 5.28). The delay_1 scenario also seems a favorable condition for detection of exploit6.

On the other hand, as shown in Figure 5.27, Bro's high false positive detection of the packet corruption scenarios makes them the least effective scenarios for Bro's detection of exploit6 (see Figure 5.29). Because the corrupt_1 and corrupt_2 cluster is so high in false positives, I also include the ROC-like scatter plot while omitting the packet corruption scenario data. It is clear from the resulting plot that the none and loss_3 scenario are the most favorable to effective Bro detection of exploit6.

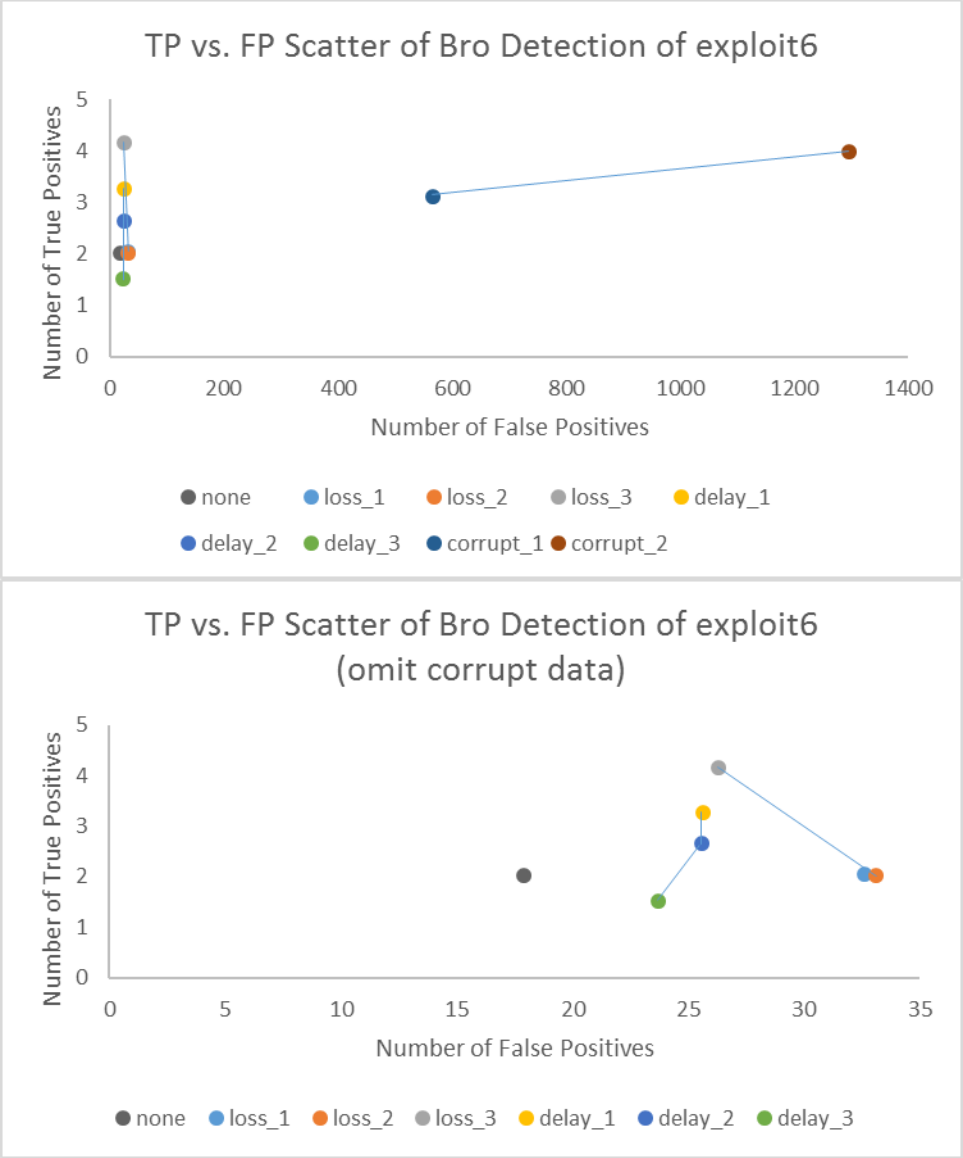

Figure 5.29: ROC-like scatter plots of Bro detection of exploit6 across 9 different network degradation scenarios (top) and across 7 scenarios omitting bit corruption scenarios

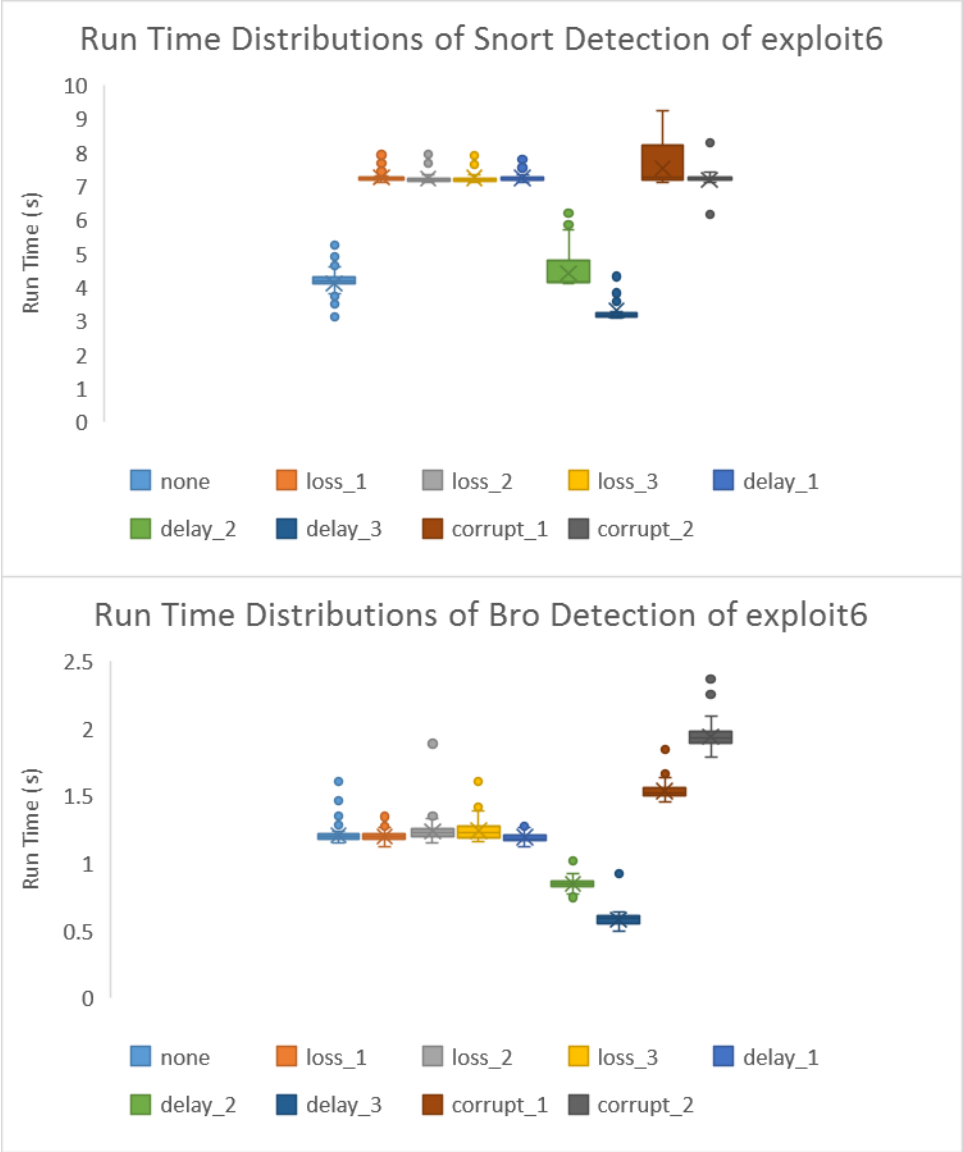

Figure 5.30: Run time distributions of both Snort and Bro detection of exploit6 across 9 different network degradation scenarios

exploit6 is one of the lightest of the attacks transmitting only a handful of packets with precisely incorrect header values and payloads (see §3.4.4). This can be expected of a CVE-based attack. The lightness of the attack results in narrow IQRs of resource usage

in processing the traffic with relatively fewer and closer outliers (see Figures 5.30-5.31). As noted in previous sections, Bro predictably requires more time and allocates more memory to process the packet corruption scenarios due to more extensive header validation rules.

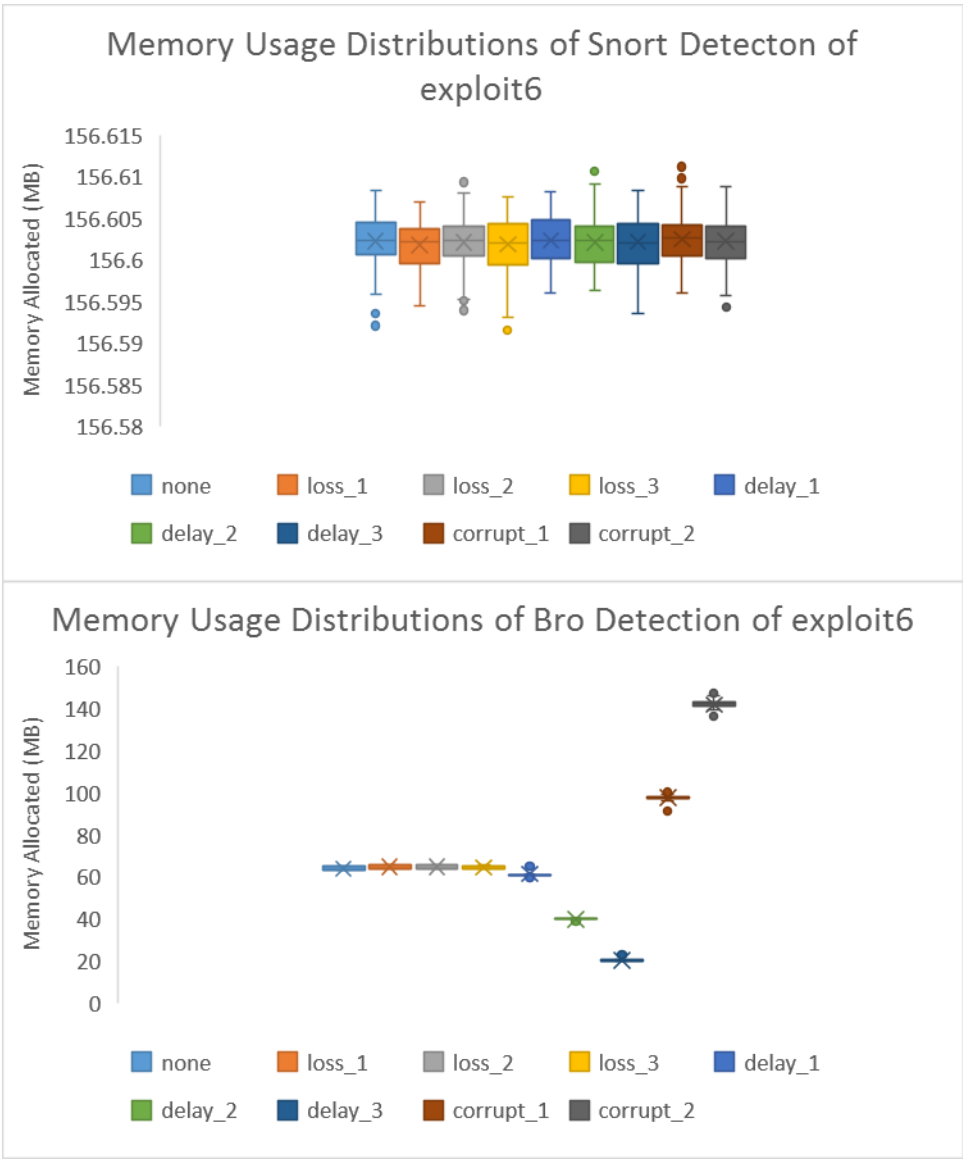

Figure 5.31: Memory usage distributions of both Snort and Bro detection of exploit6 across 9 different network degradation scenarios

5.3: Analysis of False Positive Results

It is important to note the qualitative messages of false positive results detected by both Bro and Snort. These messages are common to all settings and attacks because they are primarily caused by the benign traffic generated by Ostinato and its interactions with malicious traffic. The most common false positive messages are presented in Table 5.1 for Bro detection and Table 5.2 for Snort detection.

Number	Qualitative false positive messages produced by Bro detection
1	TCP_seq_underflow_or_misorder
2	window_recision
3	above_hole_data_without_any_acks
4	UDP_datagram_length_mismatch
5	bad_TCP_header_len
6	unknown_protocol
7	SYN_with_data
8	data_before_established
9	active_connection_reuse
10	data_after_reset
11	inner_IP_payload_length_mismatch
12	bad_HTTP_request
13	dns_unmatched_msg

Table 5.1: False positive messages produced by Bro throughout experiment

Number	Qualitative false positive messages produced by Snort detection
1	[1:24303:6] PROTOCOL-ICMP IPv6 multicast neighbor add attempt
2	[1:18474:3] PROTOCOL-ICMP ICMPv6 Echo Request
3	[119:14:1] (http_inspect) NON-RFC DEFINED CHAR
4	[3:39065:1] SERVER-OTHER Cisco IOS NX invalid ICMPv6 neighbor discovery hop limit denial of service attempt
5	[1:27611:1] PROTOCOL-ICMP Truncated ICMPv6 denial of service attempt
6	[138:5:1] SENSITIVE-DATA Email Addresses
7	[139:1:1] (spp_sdf) SDF Combination Alert

Table 5.2: False positive messages produced by Snort throughout experiment

Most of the alerts in Table 5.1 describe network events which fall into two categories: improper packet headers (#4, 5, 6, and 11) and non-traditional/prohibited TCP/UDP conversations (#1, 7, 8, 9, and 10). I suspect that most of the alerts pertaining to improper packet headers are related to packet corruption events caused by NetEm's bit flipping scenario. This is satisfactory because it demonstrates how Layer-2 packet corruption affects IDS performance. Additionally, I suspect that most of the alerts pertaining to non-traditional/prohibited TCP/UDP conversations are related to how Ostinato creates and transmits packets. The TCP/UDP packets all contain variable length payloads containing random bits without regard to details in the header. Hence, many TCP/UDP packets end up containing mismatching headers and payloads or packets with payloads are received when either packets or payloads are not expected. The nature of a virtualized host-only test network that does not contain its own DHCP or DNS servers

also leads to a small number of DNS errors (#13). The vast majority of these alert messages are different from the true positive alert messages.

Many of the alert messages found in Table 5.2 are also associated with true positives identified in this experiment (#1, 2, 3). This occurs because many of Snort's alerts trigger on general ICMPv6 types. Since various ICMPv6 types were included in the generation of benign traffic, this phenomenon is unavoidable. No. 5 in Table 5.2 is an artifact of the explanation given in §5.1.4 about how flood_solicitate6 could artificially inflate the false positive statistics returned by either IDS. Other false positive Snort messages seem to occur for miscellaneous reasons or are related to improper headers. The explanation for these is similar to that given for Bro and its detection of improper headers.

5.4: Analysis Of Detection Results

Overall, for the selected IPv6 attacks, it is clear that Snort is more effective in achieving an IDS's main purpose: true positive detection. It detects all IPv6 attacks tested while Bro only detects flood_advertise6 and exploit6. Snort casts a wider net by triggering on more general rules like ICMPv6 message types such as Echo Requests or Echo Replies. It also triggers very specifically on flood attempts and neighbor add attempts (see Tables 4.6, 4.10). The network degradation scenarios that most drastically affect Snort's true positive detection are those including packet delay, which caused drastic drops in true positive rates as delay was increased for denial6, flood_advertise6, and covert_send6. Otherwise, Snort performed relatively evenly across all network scenarios.

Where Snort employs slightly more general rules that catch more potentially malicious packets, Bro seems to use more specific rules that catch less potentially

malicious packets with more detail. This results in lower overall false positive detection by Bro, which is especially clear for non-DoS attacks alive6 and exploit6 (see Figures 5.2 and 5.27). Because 0.7% of all benign Ostinato-generated traffic in all trials were ICMPv6, much of Snort's false positive alerts probably come from benign ICMPv6 traffic.

As stated above, true and false positives in this experiment are differentiated by whether or not the source or destination IPv6 addresses of the observed packet matches that of ATTACKER or TARGET. In a production IDS system with more specific network architectures and more unique, known host attributes, a network security administrator should be able to create stronger rules to differentiate between a true and false alert. For example, some hosts on the network may be vulnerable to specific attacks, protocols, or services, and these types of communications could trigger alerts while hosts immune to such mechanisms can ignore them as false positives. In this way, malicious packets correctly addressed to hosts on a victim network can be dismissed as incredible threats if they are malicious but of no cause for concern. Such qualification would be helpful in improving the general Echo Request/Reply alerts that Snort produced for benign and malicious ICMPv6 packets alike.

It should be noted that both Snort and Bro required local configuration. They should not be considered plug and play solutions to the network security problem. Configuration for either IDS can include: specifying known good/bad IP addresses, inclusion/exclusion of any rule, severity level of any alert, custom behavior-specific/site-specific rules, whether or not to verify checksums, specifying reading PCAPs versus live monitoring, which logs/alerts to evaluate as output, multi-node architectures, among many other helpful configurable aspects of using the systems. A source of error for true

positive and false positive detection is the specific configuration used. In configuring both IDSs, I took care to use the most general configurations available while still enabling rules that are relevant to IPv6, ICMPv4, ICMPv6, and DoS attacks. Where rules alerting on irrelevant behavior existed, I chose the default configuration.

5.5: Analysis of Resource Usage Results

Perhaps just as important to the detection performance of the detection systems are their management of resources. The two most significant metrics to measure resource usage are run times and memory allocation. Passive network monitors must meet a run time threshold to analyze live network traffic without dropping packets and risk significant reductions in detection performance. Even if an IDS is processing traffic offline, it must process it faster than traffic is extracted from the live wire for inspection. Memory allocation is another helpful metric that helps to assess the hardware needs of the IDS to maintain effective operability and sufficient run times. Moreover, observing memory allocation also helps establish expectations for the performance of the IDS when it is processing different kinds of traffic.

Finally, it is important to consider memory allocation and run time together, as they are interdependent. One method is to hold memory allocation constant while observing run time. Perhaps what offers a better view into the IDS's efficiency is observing how both variables trend with each other as different scenarios and traffic are observed.

On both metrics, Bro proved more computationally efficient in most cases. Because the IDS philosophies are so disparate, resource usage comes down to more than just the efficiency of the code. Bro creates different kinds and numbers of data structures to track complex overall behaviors in traffic, while Snort runs all packets through all

preprocessors (see §2.2.1-2.2.2). It appeared that Bro's data structures caused memory allocation to positively correlate with the number of unique IPv6 flows while decreasing sharply for increases in packet delay. This trend is evident in Bro's high memory allocation (~4-5 GB) while processing PCAPs containing flood_advertise6 and flood_solicitate6. Other than those attacks, Bro's memory usage was less than Snort's rather constant 156 MB with most scenarios only requiring 30-140 MB allocations. Across all attacks, Bro's memory usage dips sharply for processing scenarios containing packet delay. As previously mentioned, I expect that this is caused by Bro rebalancing its data structures and optimizing memory allocation once observation windows of unique IPv6 flows expire. Finally, the variance in Bro's memory allocation for a given attack and scenario is higher for the DoS attacks compared to the other attacks. This occurs because the DoS attacks create a large, varying number of unique conversations every trial, while unique conversations are relatively constant in PCAPs for other attacks.

Bro's data structures help its engine extrapolate salient characteristics of observed flows. It is not unexpected that Bro's engine processes significantly faster than Snort's. Between the scenarios, the two IDS engines require similar relative run times: across all attacks, both IDSs showed dramatic drops in run time for scenarios containing packet delay. Perhaps more noticeably, Snort's run time consistently averages 4-6 times that of Bro's for non-DoS attacks. Bro's run times for non-DoS attacks sits around 1-1.5 seconds while Snort's varies from 4-9 sec. For these attacks, both Snort and Bro satisfy real-time requirements for operation on live networks, but Bro shows better invulnerability to time-intensive calculations. For DoS attacks, specifically flood_advertse6 and flood_solicitate6, Bro still beats Snort but by less time. Bro took 15-20 seconds to process the 35-second PCAPs, while Snort needed 25-35 seconds. The fact that Snort

averages near the 35-second threshold with a statistically significant minority of trials exceeding the threshold distinguishes these run time results in particular from the rest. Snort no longer can be considered real-time for those trials.

To my knowledge, PCAP reading modes for both IDSs are not significantly different from monitoring traffic live. However, site-specific architecture of the processing node(s) for either IDS could make each IDS significantly more adaptable to a site's needs. For example, Bro's multi-node architecture could leverage separate servers for high volume traffic. Alternatively, whether an IDS is configured to process traffic inline versus offline would impact network traffic. Both Snort and Bro, by default, fail open, meaning when they malfunction, potentially harmful traffic still reaches its target. This occurs by design because they are inherently detection systems, not prevention systems. However, administrators are able to configure them differently, and a different configuration could affect both run time and memory allocation.

Chapter 6: Conclusion

6.1: Answers To Research Questions

The original research questions are revisited below:

(1) For current state-of-the-art IPv6 attacks, which IDS philosophy is more effective in detecting attempts to compromise a network: signature-based detection or behavior-based detection?

(2) As network conditions deteriorate and routing becomes less reliable or as traffic load overwhelms each IDS, which IDS approach maintains better performance?

(3) Does the more resource-intensive reassembly operations of a behavior-based detection system ultimately render it inferior when it is overwhelmed?

To answer (1), I reference the general trends for true positive detection observed. In this category, Snort and its signature-based detection, are clear winners. As a Layer-3 protocol with its protocol-specific behaviors clearly defined in various RFCs, IPv6 exhibits predictable enough behavior that Snort's per packet inspection is more than sufficient to alert a security analyst of malicious traffic for the attacks tested. It is arguable whether Snort or Bro are superior in mitigation of false positive detection. To its advantage, Bro's false positive rates fall when packet delay is introduced, but it also decreases in some cases when packet corruption is introduced. Snort is more consistent across the network scenarios. Overall, I believe Snort's signature-based detection surpasses Bro's behavior-based detection in detection effectiveness.

Answering (2), Snort is largely immune to varying the network conditions by introducing packet loss, packet delay, and packet corruption. Snort demonstrates greater

resistance to changing network conditions while Bro's true positive and false positive detection increased for situations in which packet corruption was introduced. While the introduction of packet delay did significantly affect both engines for both detection metrics, they seemed to be affected in similar ways—generally lower detection. On the other hand, Bro's detection with degraded network conditions differed more from the ideal network more, being especially affected by packet corruption. This indicates that if networks suffer infrastructural errors, Bro is more likely to alert in error. For this reason, Snort is more resilient to adverse network conditions. This may be because in compensation for fundamentally not being a signature-based detection engine, Bro's rules alert less proficiently on malicious activity evident in signatures. In other words, since adverse network conditions primarily affect headers and signatures and Bro is inferior in signature-based detection, a fundamentally signature-based method performs better.

In answer to (3), the results and distributions for run time and memory allocation are more easily compared between the two IDSs on an absolute scale. Because of this, most of the time, Bro is the winner in its ability to quickly initialize its engine, process the PCAP, and produce an actionable result. Only in outlier trials, does Bro's run time exceed Snort's for the same PCAP. Additionally, besides for the DoS attacks, Bro's memory allocation is lower than Snort's consistent 156 MB allocation. When processing DoS attacks however, Bro uses 4-5 GB while Snort maintains its 156 MB load. Bro shows a remarkably wide variance in its memory allocation and extreme efficiency for normal benign traffic. Even though memory allocation is so traffic-specific, comparing Snort and Bro side-by-side with the scenarios tested in this experiment give reasonable confidence that Bro's memory allocation is not only more efficient than Snort's, but it even allows Bro to report more important and specific details about the traffic it sees.

This is proven by the number of logs that Bro creates such as conn.log and dns.log which allow a white hat researcher a better picture into the state of a network at any given time. In some tests, Snort automatically fails the real-time analysis test by exceeding 35 seconds of run time. In short, the answer to (3) is multi-faceted. Bro fails miserably in memory allocation for DoS attacks, but even when processing DoS attacks, its run time is shorter than Snort's. For non-DoS attacks, Bro is a clear winner.

I conclude that with the proper hardware to handle or ignore DoS attacks, Bro's computational efficiency is superior to Snort's. I believe this is consistent with the differences in their philosophies as well. A signature-based IDS checks traffic on a per packet basis. The single, most important traffic characteristic for such an IDS is the volume of traffic. On the other hand, a behavior-based IDS tracks flows and high level behaviors that manifest themselves progressively throughout time. This requires allocating more memory for larger windows of traffic in order to revisit details on traffic already seen. For this reason, traffic that contains many long or unfinished conversations will cause spikes in memory allocation.

To summarize these conclusions, Table 6.1 shows the winner in each high level category.

Metric	**Snort**	**Bro**
True positive	Winner	
False positive	Winner	
Run Time		Winner
Memory Usage		Winner; excepting DoS

Table 6.1: High level Snort and Bro comparison

As mentioned in §2.3, a hybrid approach to both IDS philosophies offers the best realistic solution between the two. Even though Snort is superior in overall true positive detection, Bro can provide more detail on malicious true positives. Bro's helpful logs can offer even a Snort user, better visibility into the network at an arbitrary point in time. Bro's unique language allows custom rule/analytic writers to create site-specific alerts that can monitor more complex behaviors and be efficiently executed in terms of run time and memory allocation. Bro's computational efficiency seems to lack only better rules, which may not be a problem for sufficiently resourced security groups. Meanwhile, Snort provides a great, community-supported rules database that is easily installed and casts wide nets for potential malicious activity. Its signature-based approach uses predictable allocations of memory and detects every single IPv6 attack tested in this experiment. Its comprehensiveness and frequent rule updates make its detection superior. If one approach had to be chosen, I would recommend Snort and signature-based detection since not detecting large majority of true positives puts Bro and behavior-based detection at a serious disadvantage.

6.2: FUTURE WORK

One important contribution of this work is the methodology and framework for testing IDSs and CLI-initiated attacks. The Python scripting and other automated procedure and data generation mechanisms create a sort of plug and play capability for testing any network attack with Bro or Snort. A future researcher can easily scale to hundreds of trials, creating PCAPs, evaluating the results, and presenting them in a spreadsheet. Additionally, great effort was made to assemble an apparatus of many software projects and tools to test IDS engines in realistic conditions such as load, high false positives, and adverse network conditions. The limitations to these contributions lie

in the architecture and assumptions made. These methods were not perfect models of real world network security problems. With significant lab hardware, the test network does not have to be virtualized and with sufficient time, Snort and Bro are not required to run in PCAP read mode. It is possible that different architecture and use cases of the IDSs could lead to different results.

Besides for modifying experimental setup, more IPv6 attacks can and should be tested. Only a handful of attacks from the THC-IPv6 tool suite were sampled based on the Cyber Kill Chain steps. The source code for these tools can even be modified for more precise testing. All of the tools used in this experiment are open source.

I am providing all of my code and data under copyright for public use for the purpose of leveraging this work to create better experiments. These can be found at: [GinCode2017] and [GinData2017].

Glossary

ALL ABBREVIATIONS

ACK: TCP Acknowledge

AH: Authentication Header

APT: Advanced Persistent Threat

ARP: Address Resolution Protocol

ARPANET: Advanced Research Projects Agency Network

AS: Autonomous System

ASCII: American Standard Code for Information Interchange

CLI: Command-line Interface

CRC: Cyclic Redundancy Check

CVE: Common Vulnerabilities and Exposures

DAD: Duplicate Address Detection

DDoS: Distributed Denial of Service

DHCP: Dynamic Host Control Protocol

DNS: Domain Name Service

DoS or DOS: Denial of Service

DPI: Deep Packet Inspection

DUT: Device Under Test

ESP: Encapsulating Security Payload

FID: Flow-based Intrusion Detection

FTP: File Transfer Protocol

GB: Gigabyte

GUI: Graphical User Interface

HIDS: Host-based Intrusion Detection System

HTTP: Hypertext Transfer Protocol

IANA: International Assigned Numbers Authority

ICMP: Internet Control Message Protocol

ICMPv6: Internet Control Message Protocol version 6

IDPS: Intrusion Detection and Prevention System

IDS: Intrusion Detection System(s)

IETF: Internet Engineering Task Force

IGMP: Internet Group Management Protocol

IMAP: Internet Message Access Protocol

IoT: Internet of Things

IP: Internet Protocol

IPS: Intrusion Prevention System

IPSec: Internet Protocol Security

IPv4: Internet Protocol version 4

IPv6: Internet Protocol version 6

IQR: Interquartile Range

ISAKMP: Internet Security Association and Key Management Protocol

ISO: International Standards Organization

ISP: Internet Service Provider

LAN: Local Area Network

MAC: Media Access Control

MB: Megabyte

MITM: Man-In-The-Middle

MLD: Multicast Listener Discovery

MRD: Multicast Router Discovery

MTU: Maximum Transmission Unit

NA: Neighbor Advertisement

NAT: Network Address Translation

ND: Neighbor Discovery

NDP: Neighbor Discovery Protocol

NetEm: Network Emulator

NIC: Network Interface Card

NIDS: Network Intrusion Detection System(s)

NIQ: Node Information Query

NS: Neighbor Solicitation

OS: Operating System

OSI: Open Systems Interconnection

PCAP: Packet capture (.pcap file format)

PMTUD: Path Maximum Transmission Unit Discovery

POP: Post Office Protocol

RA: Router Advertisement

RAM: Random Access Memory

RFC: Request for Comment

ROC: Receiver Operating Characteristics

RS: Router Solicitations

SCADA: Supervisory Control and Data Acquisition

SEND: Secure Neighbor Discovery

SLAAC: Stateless Autoconfiguration

SMTP: Simple Mail Transfer Protocol

SPAN: Switched Port Analysis

SYN: Synchronize

TCP: Transmission Control Protocol

THC: The Hacker's Choice

UDP: User Datagram Protocol

VM: Virtual Machine

References

[Bro2016] The Bro Project. “Bro Manual.” Accessed: 9 Feb 2017. https://www.bro.org/sphinx/index.html

[Caicedo2009] Caicedo, Carlos E., James BD Joshi, and Summit R. Tuladhar. “IPv6 security challenges.” *Computer* 42.2 (2009): 36-42. Accessed 3 Feb 2017. https://pdfs.semanticscholar.org/8c31/c745c6caf4ee4185453459069f4dfcd1d54f.pdf

[Deering1998] Deering, Stephen E. “Internet protocol, version 6 (IPv6) specification.” (1998). Accessed: 3 Feb 2017. https://www.ietf.org/rfc/rfc2460.txt

[Denning1987] Denning, Dorothy E. “An intrusion-detection model.” *IEEE Transactions on software engineering* 2 (1987): 222-232. Accessed: 3 Feb 2017. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6234848

[Elejla2016] Elejla, Omar E., et al. “Intrusion Detection Systems of ICMPv6-based DDoS attacks.” *Neural Computing and Applications* (2016): 1-12. Accessed: 3 Feb 2017. http://link.springer.com/article/10.1007/s00521-016-2812-8

[Garcia-Teodoro2009] Garcia-Teodoro, Pedro, et al. "Anomaly-based network intrusion detection: Techniques, systems and challenges." computers & security 28.1 (2009): 18-28. http://ac.els-cdn.com/S0167404808000692/1-s2.0-S0167404808000692-main.pdf?_tid=954f5e7e-fd30-11e6-9bdd-00000aab0f27&acdnat=1488229758_431304e30f94ed91048b26b580c2af31

[Gates2008] Gates, Carrie E. “A case study in testing a network security algorithm.” *Proceedings of the 4th International Conference on Testbeds and research infrastructures for the development of networks & communities*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008. Accessed: 3 Feb 2017. https://web.cs.dal.ca/~gates/papers/trident08.pdf

[Gehrke2012] Gehrke, Keith A. *The unexplored impact of IPv6 on intrusion detection systems*. Diss. Monterey, California. Naval Postgraduate School, 2012. Accessed 3 Feb 2017. http://calhoun.nps.edu/handle/10945/6800

[GinCode2017] Gin, Jeremy. Python scripts for MS Report: IDS IPv6 Analysis [Software]. 2017. Available at: https://github.com/jgin1

[GinData2017] Gin, Jeremy. PCAP and Excel Datasets for MS Report: IDS IPv6 Analysis [Dataset]. 2017. Available at: http://users.ece.utexas.edu/~bevans/students/ms/jeremy_gin/

[Gont2014] Gont, Fernando. "Security implications of IPv6 on IPv4 networks." (2014). Accessed: 3 Feb 2017. https://tools.ietf.org/html/rfc7123.html

[Google2017] Google IPv6. "Statistics." Web. Accessed 1 Feb 2017. https://www.google.com/intl/en/ipv6/statistics.html

[Hauser2017] Hauser, Van. "THC-IPv6 Attack Tool Kit." *The Hacker's Choice*. January 2017. Web. Accessed 1 Feb 2017. http://www.thc.org/thc-ipv6

[Hemminger2005] Hemminger, Stephen. "Network emulation with NetEm." Linux conf au. 2005. https://pdfs.semanticscholar.org/9180/aa7b7978c62363e4af3a9053371775fbcbdc.pdf

[Hogg2008] Hogg, Scott, and Eric Vyncke. IPv6 security. Pearson Education, 2008.

[IC32016] IC3. "Amount of Monetary Damage Caused by Reported Cyber Crime to The Ic3 from 2001 to 2015 (in Million U.S. Dollars)." *Statista - The Statistics Portal*. Statista. May 2016. Web. Accessed: 1 Feb 2017. https://www-statista-com.ezproxy.lib.utexas.edu/statistics/267132/total-damage-caused-by-by-cyber-crime-in-the-us/

[Koziol2003] Koziol, Jack. Intrusion detection with Snort. Sams Publishing, 2003.

[Labovitz2008] Labovitz, Craig, et al. "Internet traffic trends-a view from 67 ISPs." *North American Network Operators Meeting*. Vol. 33. 2008. Accessed 23 Mar 2017. https://www.nanog.org/meetings/nanog43/presentations/Labovitz_internetstats_N43.pdf

[Levine2015] Levine, Mike and Date, Jack. "22 Million Affected by OPM Hack, Officials Say." *abc NEWS*. July 2015. Web. Accessed: 1 Feb 2017. http://abcnews.go.com/US/exclusive-25-million-affected-opm-hack-sources/story?id=32332731

[Lockheed2017] Official Lockheed Martin Website. Cyber Kill Chain Model Web Page. Web. Accessed 5 Apr 2016. http://www.lockheedmartin.com/us/what-we-do/aerospace-defense/cyber/cyber-kill-chain.html

[Mehra2012] Mehra, Pritika. "A brief study and comparison of snort and bro open source network intrusion detection systems." *International Journal of Advanced Research in Computer and Communication Engineering* 1.6 (2012): 383-386. Accessed: 3 Feb 2017. http://www.ijarcce.com/upload/august/4-A%20brief%20study%20and%20comparison%20of.pdf

[Mitre2017] Common Vulnerabilities and Exposures. Web. Accessed 23 Mar 2017. https://cve.mitre.org

[Moya2008] Moya, Miguel A. Calvo. "Analysis and evaluation of the snort and bro network intrusion detection systems." *Intrusion Detection System, Universidad Pontificia Comillas* (2008). Accessed: 3 Feb 2017. https://www.iit.comillas.edu/pfc/resumenes/48cd357480a88.pdf

[Netem2016] The Linux Foundation. NetEm manual page. Web. Accessed 23 Feb 2017. https://wiki.linuxfoundation.org/networking/netem

[O'Brien2016] O'Brien, Sara Ashley. "Widespread cyberattack takes down sites worldwide." *CNN Tech*. October 2016. Web. Accessed: 1 Feb 2017. http://money.cnn.com/2016/10/21/technology/ddos-attack-popular-sites/

[P2016] P, Srivats. Official Ostinato Website. Web. Accessed 23 Feb 2017. http://ostinato.org/

[Papadogiannakis2010] Papadogiannakis, Antonis, Michalis Polychronakis, and Evangelos P. Markatos. "Improving the accuracy of network intrusion detection systems under load using selective packet discarding." *Proceedings of the Third European Workshop on System Security*. ACM, 2010. Accessed 3 Feb 2017. https://www.ics.forth.gr/_publications/discarding.eurosec10.pdf

[Parkour2017] Parkour, Mila. Contagio Blog Malware Dump Collection. Accessed 23 Mar 2017. Web. http://contagiodump.blogspot.com/

[Paxson1999] Paxson, Vern. "Bro: a system for detecting network intruders in real-time." *Computer networks* 31.23 (1999): 2435-2463. Accessed: 3 Feb 2017. https://pdfs.semanticscholar.org/f121/cde8d3d5364717caf2b91d27a270ec004cac.pdf

[Peterson2007] Peterson, Larry L., and Bruce S. Davie. Computer networks: a systems approach. Elsevier, 2007.

[Pilihanto2011] Pilihanto, Atik. "A Complete Guide on IPv6 Attack and Defense." *Sans.org* November 2011. Web. Accessed 1 Feb 2017. https://www.sans.org/reading-room/whitepapers/detection/complete-guide-ipv6-attack-defense-33904

[Postel1981] Postel, Jon. "RFC 791: Internet protocol." (1981). Accessed 3 Feb 2017. https://tools.ietf.org/html/rfc791

[Ptacek1998] Ptacek, Thomas H., and Timothy N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. SECURE NETWORKS INC CALGARY ALBERTA, 1998. http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA391565

[Puketza1996] Puketza, Nicholas J., et al. "A methodology for testing intrusion detection systems." *IEEE Transactions on Software Engineering* 22.10 (1996): 719-729. Accessed 3 Feb 2017. http://ieeexplore.ieee.org/abstract/document/544350/?reload=true

[Rehman2003] Rehman, Rafeeq Ur. *Intrusion detection systems with Snort: advanced IDS techniques using Snort, Apache, MySQL, PHP, and ACID*. Prentice Hall Professional, 2003. Accessed 3 Feb 2017.

https://books.google.com/books?hl=en&lr=&id=1WKrLbh23LAC&oi=fnd&pg=PA1&dq=intrusion+detection+systems+with+snort+advanced+ids+techniques+using+snort+apache&ots=5p9809RLaA&sig=1xfhjpCDbbgtu78UknDAtZM6jbY#v=onepage&q=intrusion%20detection%20systems%20with%20snort%20advanced%20ids%20techniques%20using%20snort%20apache&f=false

[Roesch1999] Roesch, Martin. "Snort: Lightweight intrusion detection for networks." Lisa. Vol. 99. No. 1. 1999. http://static.usenix.org/publications/library/proceedings/lisa99/full_papers/roesch/roesch.pdf

[Sarrar2012] Sarrar, Nadi, et al. "Investigating IPv6 Traffic." *International Conference on Passive and Active Network Measurement*. Springer Berlin Heidelberg, 2012. Accessed 3 Feb 2017. http://link.springer.com/chapter/10.1007%2F978-3-642-28537-0_2#page-1

[Scarfone2007] Scarfone, Karen, and Peter Mell. "Guide to intrusion detection and prevention systems (idps)." *NIST special publication* 800.2007 (2007): 94. Accessed 3 Feb 2017. http://ecinetworks.com/wp-content/uploads/bsk-files-manager/86_SP800-94.pdf

[Schütte2016] Schütte, Martin, Thomas Scheffler, and Bettina Schnor. "Development of a snort ipv6 plugin." (2012). Web. Accessed 4 Mar 2017. https://prof.beuth-hochschule.de/fileadmin/user/scheffler/Publikationen/SECRYPT_2012.pdf

[Shearer2013] Shearer, Jarrad. "W32.Downadup". (2013). Symantec. Web. Accessed 7 Mar 2017. https://www.symantec.com/security_response/writeup.jsp?docid=2008-112203-2408-99

[Snort2016] Official Snort Website. Web. Accessed 1 Feb 2017. http://www.snort.org

[Sperotto2012] Sperotto, Anna, et al. "An overview of ip flow-based intrusion detection." *IEEE Communications Surveys and Tutorials* 12.3 (2010): 343-356. Accessed 3 Feb 2017. http://eprints.eemcs.utwente.nl/18341/01/Surveys_and_tutorial.pdf

[tcpdump2017] tcpdump version 4.9.0. Web. Accessed 27 Feb 2017. http://www.tcpdump.org/

[VirusTotal] VirusTotal Website. Web. Accessed 25 Feb 2017. https://www.virustotal.com/

[VMWare2017] Fusion for Mac. Web. Accessed 23 Feb 2017. http://www.vmware.com/products/fusion.html

[Warfield2003] Warfield, Michael H. "Security implications of IPv6." *Internet Security Systems* 4.1 (2003): 2-5. Accessed 3 Feb 2017.

https://www.blackhat.com/presentations/bh-federal-03/bh-federal-03-warfield/bh-fed-03-paper-warfield.doc

[Werny2014] Werny, Christopher and Schaefer, Rafael. "IPv6 Attack & Defense Strategies". *Blackhat Conference*. August 2014. Presentation. Accessed 1 Feb 2017. https://www.blackhat.com/docs/sp-14/materials/arsenal/sp-14-Schaefer-Workshop-Slides.pdf

[WorldIPv6] World IPv6 Launch "Participants." Web. Accessed 1 Feb 2017. http://www.worldipv6launch.org/participants/

Vita

Jeremy Gin earned his Bachelor of Science in Electrical and Computer Engineering from the University of Arizona in 2015. During his studies at the University of Texas at Austin, Jeremy has been interested securing networks, communications and control systems through robust architecture and algorithms. An updated resume can be found at: https://www.linkedin.com/in/jgin1/.

Jeremy is a follower of Jesus Christ.

"I have been crucified with Christ. It is no longer I who live, but Christ who lives in me. And the life I now live in the flesh I live by faith in the Son of God, who loved me and gave himself for me." - Galatians 2:20 (ESV)

Permanent email: jgin@utexas.edu

This report was typed by the author.