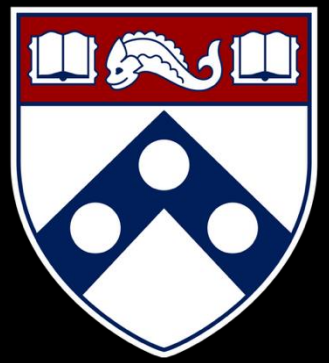
Secure Systems Engineering and Management

A Data-driven Approach

Michael Hicks



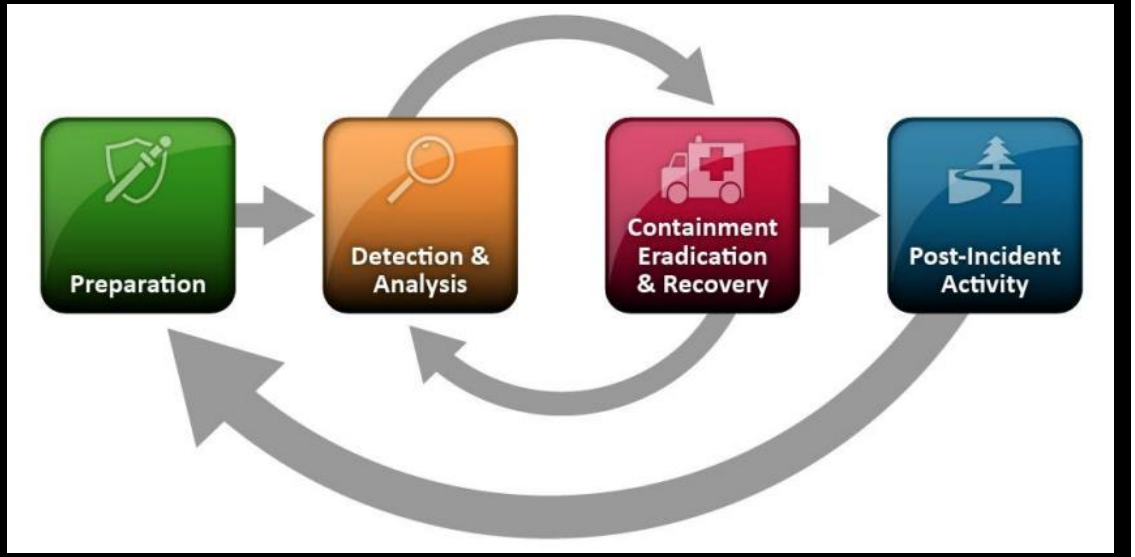
Intrusion Detection and Security Operations

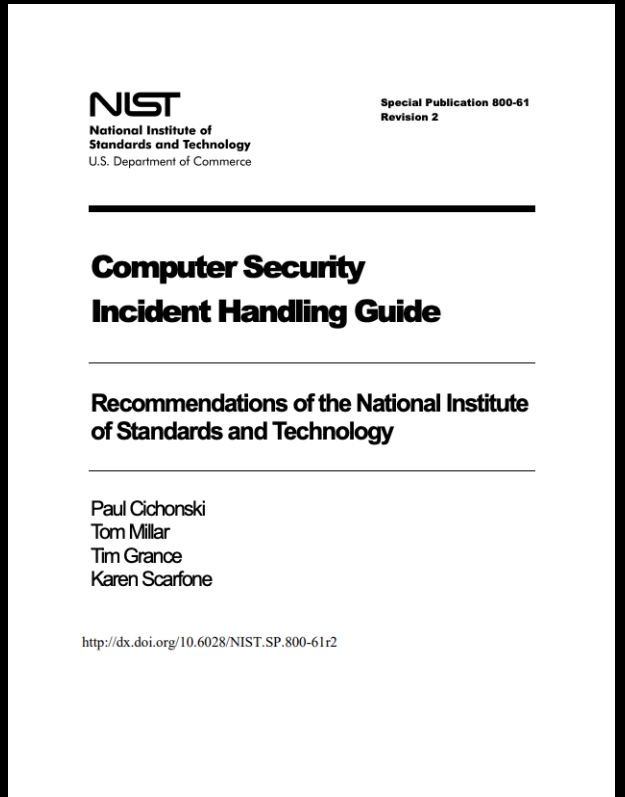
UPenn CIS 7000-003
Spring 2026

Roadmap

- Security incident lifecycle (overview)
- **Detection**: history, tools, challenges
- **Suricata and Zeek** in depth
- Security Operations Centers (SOCs)

The Incident Lifecycle (NIST SP 800-61)





NIST
National Institute of Standards and Technology
U.S. Department of Commerce

Special Publication 800-61
Revision 2

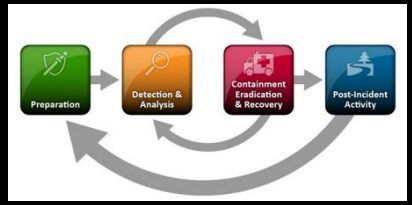
Computer Security Incident Handling Guide

Recommendations of the National Institute of Standards and Technology

Paul Cichonski
Tom Millar
Tim Grance
Karen Scarfone

http://dx.doi.org/10.6028/NIST.SP.800-61r2

The Six NIST Phases



Phase	Goal
Preparation	Tools, plans, trained staff, baselines
Detection & Analysis	Identify and validate malicious activity
Containment	Stop the spread; short-term and long-term
Eradication	Remove the threat from the environment
Recovery	Restore systems to normal operations
Post-incident activity	Lessons learned; feed improvements back

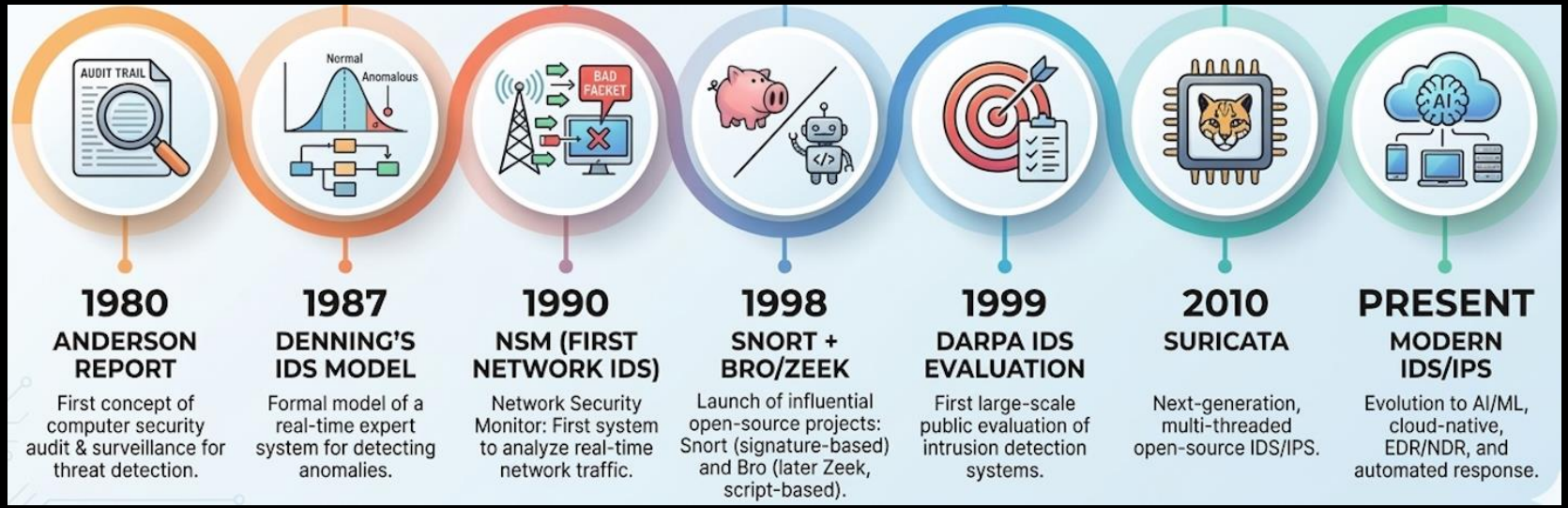
Today's focus: **Detection & Analysis**

Why Detection is Hard

"Attackers need to find **one** vulnerability. Defenders need to protect **everything**."

- Asymmetric advantage for attackers
- Massive volume of normal activity to sift through
- Sophisticated attackers blend in with legitimate traffic
- Detection must be fast, accurate, and actionable

Automating Attack Detection: A Brief History

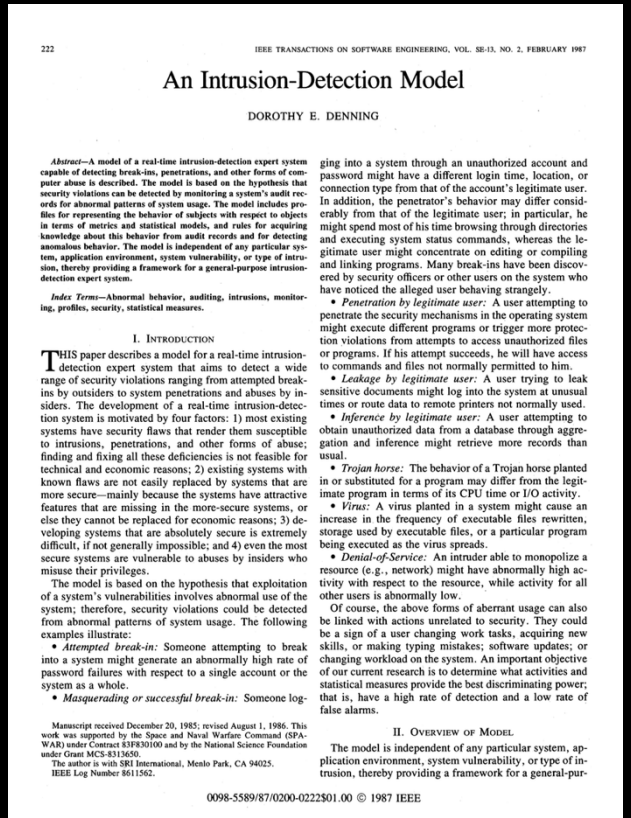


Denning's Intrusion Detection Model (1987)

The **foundational paper** for the entire IDS field

Core thesis: intrusions can be detected in real time by monitoring audit records for **abnormal patterns of system usage**

Key contribution: a **general-purpose, system-independent** model



An Intrusion-Detection Model

DOROTHY E. DENNING

Abstract—**A model of a real-time intrusion-detection expert system capable of detecting break-ins, penetrations, and other forms of computer abuse is described. The model is based on the hypothesis that security violations can be detected by monitoring a system's audit records for abnormal patterns of system usage. The model includes profiles for representing the behavior of subjects with respect to objects in terms of metrics and statistical models, and rules for acquiring knowledge about this behavior from audit records and for detecting anomalous behavior. The model is independent of any particular system, application environment, system vulnerability, or type of intrusion, thereby providing a framework for a general-purpose intrusion-detection expert system.**

Index Terms—**Abnormal behavior, auditing, intrusions, monitoring, profiles, security, statistical measures.**

I. Introduction

THIS paper describes a model for a real-time intrusion-detection expert system that aims to detect a wide range of security violations ranging from attempted break-ins by outsiders to system penetrations and abuses by insiders. The development of a real-time intrusion-detection system is motivated by four factors: 1) most existing systems have security flaws that render them susceptible to intrusions, penetrations, and other forms of abuse; finding and fixing all these deficiencies is not feasible for technical and economic reasons; 2) existing systems with known flaws are not easily replaced by systems that are more secure—mainly because the systems have attractive features that are missing in the more-secure systems, or else they cannot be replaced for economic reasons; 3) developing systems that are absolutely secure is extremely difficult, if not generally impossible; and 4) even the most secure systems are vulnerable to abuses by insiders who misuse their privileges.

The model is based on the hypothesis that exploitation of a system's vulnerabilities involves abnormal use of the system; therefore, security violations could be detected from abnormal patterns of system usage. The following examples illustrate:

- *Attempted break-in:* Someone attempting to break into a system might generate an abnormally high rate of password failures with respect to a single account or the system as a whole.
- *Masquerading or successful break-in:* Someone logging into a system through an unauthorized account and password might have a different login time, location, or connection type from that of the account's legitimate user. In addition, the penetrator's behavior may differ considerably from that of the legitimate user; in particular, he might spend most of his time browsing through directories and executing system status commands, whereas the legitimate user might concentrate on editing or compiling and linking programs. Many break-ins have been discovered by security officers or other users on the system who have noticed the alleged user behaving strangely.
- *Penetration by legitimate user:* A user attempting to penetrate the security mechanisms in the operating system might execute different programs or trigger more protection violations from attempts to access unauthorized files or programs. If his attempt succeeds, he will have access to commands and files not normally permitted to him.
- *Leakage by legitimate user:* A user trying to leak sensitive documents might log into the system at unusual times or route data to remote printers not normally used.
- *Inference by legitimate user:* A user attempting to obtain unauthorized data from a database through aggregation and inference might retrieve more records than usual.
- *Trojan horse:* The behavior of a Trojan horse planted in or substituted for a program may differ from the legitimate program in terms of its CPU time or I/O activity.
- *Virus:* A virus planted in a system might cause an increase in the frequency of executable files rewritten, storage used by executable files, or a particular program being executed as the virus spreads.
- *Denial-of-Service:* An intruder able to monopolize a resource (e.g., network) might have abnormally high activity with respect to the resource, while activity for all other users is abnormally low.

Of course, the above forms of aberrant usage can also be linked with actions unrelated to security. They could be a sign of a user changing work tasks, acquiring new skills, or making typing mistakes; software updates; or changing workload on the system. An important objective of our current research is to determine what activities and statistical measures provide the best discriminating power; that is, have a high rate of detection and a low rate of false alarms.

II. Overview of Model

The model is independent of any particular system, application environment, system vulnerability, or type of intrusion, thereby providing a framework for a general-pur-

Manuscript received December 20, 1985; revised August 1, 1986. This work was supported by the Space and Naval Warfare Command (SPAWAR) under Contract 83F830100 and by the National Science Foundation under Grant MCS-8313650.

The author is with SRI International, Menlo Park, CA 94025.

IEEE Log Number 8611562.

Denning's Model: Six Components

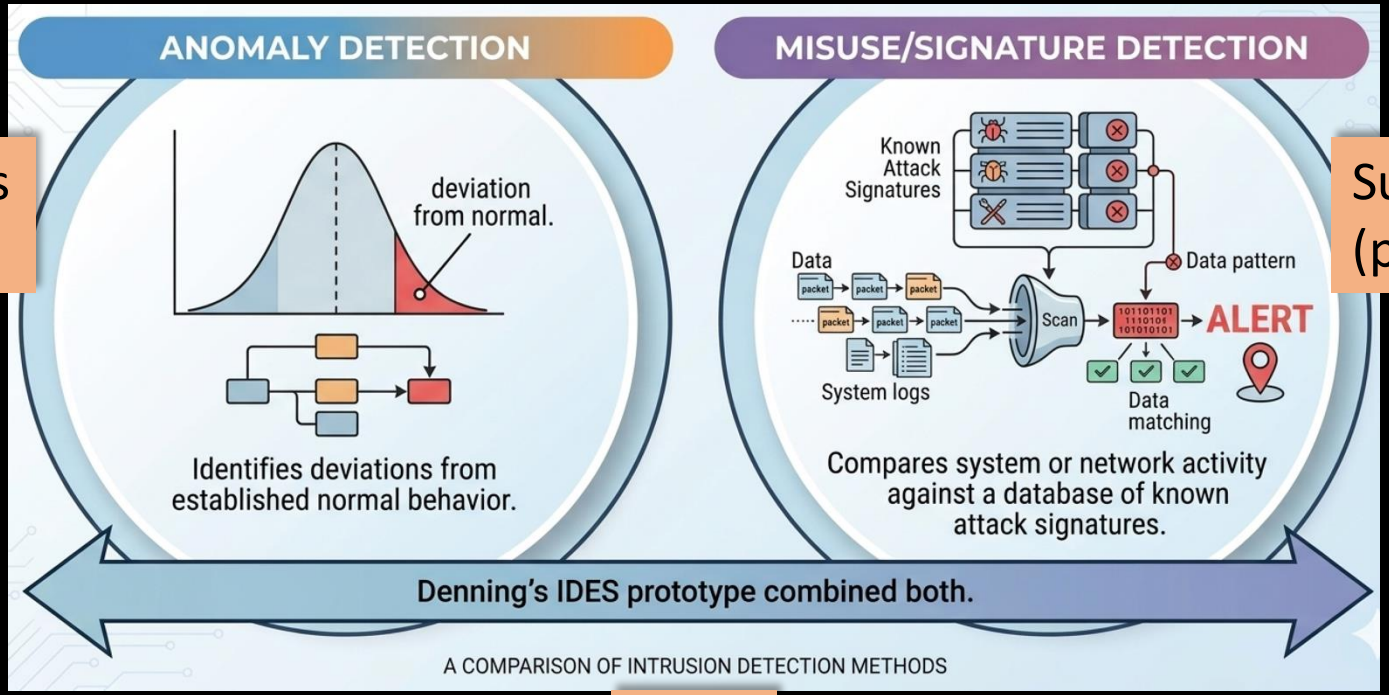
1. **Subjects** — who initiates activity (users, processes)
2. **Objects** — targets of activity (files, programs, resources)
3. **Audit Records** — `<Subject, Action, Object, Exception, Resource-Usage, Timestamp>`
4. **Activity Profiles** — statistical models of "normal" behavior
5. **Anomaly Records** — generated when behavior deviates from profile
6. **Activity Rules** — what to do when anomalies are detected

Denning's Statistical Approaches

Method	Idea	Modern echo
Event Counter	Count occurrences per time unit	Suricata `threshold` keyword
Mean + Std Dev	Flag activity >*d* std devs from mean	UEBA behavioral baselines
Multivariate	Correlate multiple metrics	SIEM correlation rules
Markov Process	Model command sequences as state transitions	Largely fallen out of favor for IDS
Time Series	Predict next observation, flag deviations	ML anomaly detection research

Denning: Two Detection Paradigms

Denning's focus



Suricata (primarily)

Zeek

Denning identified challenges still unsolved

Base-rate fallacy
even low FP rates produce many false alarms at scale

Training poisoning
if attacker present during training, malicious behavior becomes “normal”

Slow evasion
attackers gradually shift behavior to train the profile

Network Security Monitor (1990)

Todd Heberlein at UC Davis built the **first network-based IDS**

- Shifted IDS from **host audit trails** to **network traffic**
- Monitored network packets rather than system logs

Demonstrated that network traffic analysis could detect attacks

A NETWORK SECURITY MONITOR

L. Todd Heberlein, Gihan V. Dias, Karl N. Levitt, Biswanath Mukherjee, Jeff Wood and David Wolber

Division of Computer Science
Department of Electrical Engineering and Computer Science
University of California, Davis
Davis, CA 95616

Abstract

The study of security in computer networks is a rapidly growing area of interest because of the proliferation of networks and the paucity of security measures in most current networks. Since most networks consist of a collection of inter-connected local area networks (LANs), this paper concentrates on the security-related issues in a single broadcast LAN such as Ethernet. We formalize various possible network attacks. Our basic strategy is to develop profiles of usage of network resources and then compare current usage patterns with the historical profile to determine possible security violations. Thus, our work is similar to the host-based intrusion-detection systems such as SRI's IDES [9]. Different from such systems, however, is our use of a hierarchical model to refine the focus of the intrusion-detection mechanism. We also report on the development of our experimental LAN monitor currently under implementation. Several network attacks have been simulated and results on how the monitor has been able to detect these attacks are also analyzed. Initial results demonstrate that many network attacks are detectable with our monitor, although it can surely be defeated. Current work is focusing on the integration of network monitoring with host-based techniques.

1 INTRODUCTION

The study of security in computer networks is a rapidly growing area of interest [6, 7, 21]. This activity has been fueled by several recent network attacks (or network intrusions). The task of providing and maintaining security in a network is a particularly challenging one because of the following facts. First, there is a proliferation of local area networks (LANs) in academic, business, and research institutions, and these LANs are in turn interconnected with the "outside world" via gateways and wide area networks (WANs). Second, these networks and their associated equipment (including LANs, WANs, and gateways), when they were developed, were done so with trusted users in mind; the issue was to solve the networking problem and very few, if any, security measures were instituted. Consequently, network attacks or intrusions such as eavesdropping on information meant for someone else, illegally accessing information remotely, breaking into computers remotely, inserting erroneous information into files and flooding the network thereby reducing its effective channel capacity are not uncommon (see, for example, [17]).

To overcome these problems, several proposals suggest the deployment of *new, secure*, and possibly *closed* systems by using methods that can prevent network attacks, e.g., by using encryption techniques [13, 14, 16, 18, 20]. But we recognize that these solutions will not work because of the tremendous investment already made in the existing infrastructure of *open* data networks, however insecure the latter might be. Furthermore, encryption techniques cannot protect against stolen keys or legitimate users misusing their privileges. Hence, we approach the problem from a different angle. Specifically, our goal is to develop monitoring techniques that will enable us to maintain information of *normal* network activity (including those of the network's individual nodes, their users, their offered services, etc.) The monitor will be capable of *observing* current network activity, which, when compared with historical behavior, will enable it to detect in real-time possible security violations on the network – regardless of the network type, organization, and topology. Since our goal is to detect network intrusions, note that we are borrowing some of the basic concepts that have been developed or proposed for non-networked, stand-alone, intrusion-detection systems, e.g., IDES [2, 9], MIDAS [22], and others [10]. See [10] for a survey of intrusion-detection development efforts.

The focus of our present activity is narrowed to the local environment. In particular, we are developing our concepts for an Ethernet – Carrier Sense Multiple Access with Collision Detection (CSMA/CD) [11] – LAN which, because of its broadcast property, enables us to design and test a single secure monitor that has access to all of the network traffic. (Distributed monitoring of wide area networks will be considerably more complex, and will be taken up after our LAN monitoring problems have been properly tackled.) A prototype LAN security monitor – hereafter referred to as our Network Security Monitor (NSM) – has been in operation for over a year, and it is continuously being upgraded as we incorporate into it newer concepts as they emerge. The NSM in its most elementary (lowest) level of operation can measure network utilization and host-to-host activity. But when it suspects a possible intrusion or under the control of a Security Officer, it can also refine its focus on an individual user, a group of users, individual or group(s) of services they are using, etc., in a hierarchical fashion. Probabilistic, rule-based, and mixed approaches are being employed by the monitor, and it raises alarms for the Security Officer upon detecting anomalous behavior. The Security Officer interfaces with the monitor via a user-friendly window system, using which he/she can manually alter (usually refine) the monitor's focus as well. At present, the monitor is being employed to study network behavior and possible intrusions, and we report on them later in the paper.

The system model is described in the next section. We model network attacks in Section 3. The conceptual view of the NSM

CH2884-5/90/0000/0296$01.00 © 1990 IEEE

Snort and Bro (1998): Two Philosophies

Snort – Lightweight Intrusion Detection for Networks

Martin Roesch – Stanford Telecommunications, Inc.

ABSTRACT

Network intrusion detection systems (NIDS) are an important part of any network security architecture. They provide a layer of defense which monitors network traffic for predefined suspicious activity or patterns, and alert system administrators when potential hostile traffic is detected. Commercial NIDS have many differences, but Information Systems departments must face the commonalities that they share such as significant system footprint, complex deployment and high monetary cost. Snort was designed to address these issues.

Introduction

Snort fills an important "ecological niche" in the the realm of network security: a cross-platform, lightweight network intrusion detection tool that can be deployed to monitor small TCP/IP networks and detect a wide variety of suspicious network traffic as well as outright attacks. It can provide administrators with enough data to make informed decisions on the proper course of action in the face of suspicious activity. Snort can also be deployed rapidly to fill potential holes in a network's security coverage, such as when a new attack emerges and commercial security vendors are slow to release new attack recognition signatures. This paper discusses the background of Snort and its rules-based traffic collection engine, as well as new and different applications where it can be very useful as a part of an integrated network security infrastructure.

Snort is a tool for small, lightly utilized networks. Snort is useful when it is not cost efficient to deploy commercial NIDS sensors. Modern commercial intrusion detection systems cost thousands of dollars at minimum, tens or even hundreds of thousands in extreme cases. Snort is available under the GNU General Public License [GNU89], and is free for use in any environment, making the employment of Snort as a network security system more of a network management and coordination issue than one of affordability.

What is "lightweight" intrusion detection?

A lightweight intrusion detection system can easily be deployed on most any node of a network, with minimal disruption to operations. Lightweight IDS' should be cross-platform, have a small system footprint, and be easily configured by system administrators who need to implement a specific security solution in a short amount of time. They can be any set of software tools which can be assembled and put into action in response to evolving security situations. Lightweight IDS' are small, powerful, and flexible enough to be used as permanent elements of the network security infrastructure.

Snort is well suited to fill these roles, weighing in at roughly 100 kilobytes in its compressed source distribution. On most modern architectures Snort takes only a few minutes to compile and put into place, and perhaps another ten minutes to configure and activate. Compare this with many commercial NIDS, which require dedicated platforms and user training to deploy in a meaningful way. Snort can be configured and left running for long periods of time without requiring monitoring or administrative maintenance, and can therefore also be utilized as an integral part of most network security infrastructures.

What is Snort?

Snort is a libpcap-based [PCAP94] packet sniffer and logger that can be used as a lightweight network intrusion detection system (NIDS). It features rules based logging to perform content pattern matching and detect a variety of attacks and probes, such as buffer overflows [ALE96], stealth port scans, CGI attacks, SMB probes, and much more. Snort has real-time alerting capability, with alerts being sent to syslog, Server Message Block (SMB) "WinPopup" messages, or a separate "alert" file. Snort is configured using command line switches and optional Berkeley Packet Filter [BPF93] commands. The detection engine is programmed using a simple language that describes per packet tests and actions. Ease of use simplifies and expedites the development of new exploit detection rules. For example, when the IIS Showcode [IISBT99] web exploits were revealed on the Bugtraq mailing list [BTQ99], Snort rules to detect the probes were available within a few hours.

Snort vs. The World!

Snort shares commonalities with both sniffers and NIDS. Two programs that lend themselves to direct comparison with Snort, tcpdump and Network Flight Recorder [NFR97], will be examined and contrasted in this section. In many cases, Snort is financially, technically, and/or administratively easier to implement than other Open Source [OSS98] or commercially available tools.

Bro: A System for Detecting Network Intruders in Real-Time

Vern Paxson
Network Research Group
Lawrence Berkeley National Laboratory[†]
Berkeley, CA 94720
vern@ee.lbl.gov

Abstract

We describe Bro, a stand-alone system for detecting network intruders in real-time by passively monitoring a network link over which the intruder's traffic transits. We give an overview of the system's design, which emphasizes high-speed (FDDI-rate) monitoring, real-time notification, clear separation between mechanism and policy, and extensibility. To achieve these ends, Bro is divided into an "event engine" that reduces a kernel-filtered network traffic stream into a series of higher-level events, and a "policy script interpreter" that interprets event handlers written in a specialized language used to express a site's security policy. Event handlers can update state information, synthesize new events, record information to disk, and generate real-time notifications via *syslog*. We also discuss a number of attacks that attempt to subvert passive monitoring systems and defenses against these, and give particulars of how Bro analyzes the four applications integrated into it so far: Finger, FTP, Portmapper and Telnet. The system is publicly available in source code form.

1 Introduction

With growing Internet connectivity comes growing opportunities for attackers to illicitly access computers over the network. The problem of detecting such attacks is termed *network intrusion detection*, a relatively new area of security research [MHL94]. We can divide these systems into two types, those that rely on audit information gathered by the hosts in the network they are trying to protect, and those that operate "stand-alone" by observing network traffic directly, and a packet filter. In this paper we focus building stand-alone systems, which w itors." Though monitors necessarily fa of more limited information than syst to audit trails, monitors also gain the they can be added to a network with changes to the hosts. For our purpos collection of several thousand heteroge administered hosts—this advantage is immense.

Our monitoring system is called Bro (an Orwellian reminder that monitoring comes hand in hand with the potential for privacy violations). A number of commercial products exist that do what Bro does, generally with much more sophisticated interfaces and management software [In97, To97, Wh97],[1] and larger "attack signature" libraries. To our knowledge, however, there are no detailed accounts in the network security literature of how monitors can be built. Furthermore, monitors can be susceptible to a number of attacks aimed at subverting the monitoring; we believe the attacks we discuss here have not been previously described in the literature. Thus, the contribution of this paper is not at heart a novel idea (though we believed it novel when we undertook the project, in 1995), but rather a detailed overview of some experiences with building such a system.

Prior to developing Bro, we had significant operational experience with a simpler system based on off-line analysis of `tcpdump` trace files. Out of this experience we formulated a number of design goals and requirements:

High-speed, large volume monitoring For our environment, we view the greatest source of threats as external hosts connecting to our hosts over the Internet. Since the network we want to protect has

[†] This work was supported by the Director, Office of Energy Research, Office of Computational and Technology Research, Mathematical, Information, and Computational Sciences Division of the United States Department of Energy under Contract No. DE-AC03-76SF00098. *This on-line version of the paper corrects an error in the version in the printed Proceedings, which in § 7 overstated the traffic level on the FDDI ring by a factor of two.*

[1] Or at least appear, according to their product literature, to do the same things—we do not have direct experience with any of these products.

A somewhat different sort of product, the "Network Flight Recorder," is described in [RLSSLW97, Ne97].

As in "Big Bro" from 1984 (later: Zeek)

Snort and Bro (1998): Two Philosophies

	Snort	Bro (Zeek)
Creator	Martin Roesch	Vern Paxson
Philosophy	Fast signature matching	Deep protocol analysis + scripting
Output	Alerts on known attacks	Rich logs of all network activity
Approach	“Does this match a known bad pattern?”	“What is happening on this network?”

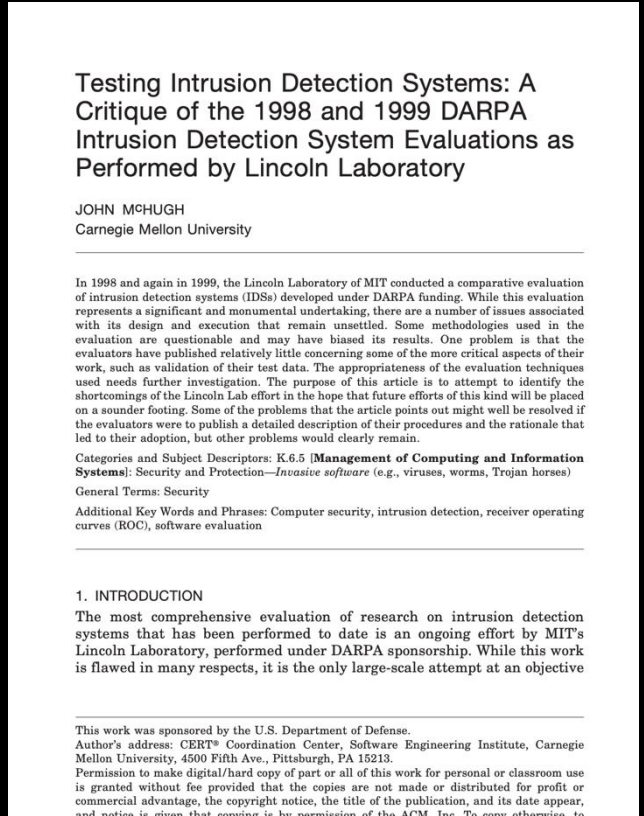
Both open-source, both transformative, both still in use today

The DARPA IDS Evaluation (1998–99)

- Lincoln Labs created a **simulated network** with injected attacks
- First attempt at **rigorous, standardized IDS testing**
- Most research papers used this dataset for over a decade

Shown to have **serious methodological flaws** (McHugh'00)

- Synthetic traffic not representative of real networks
- Attack/normal ratio unrealistic
- Became a cautionary tale about **evaluation methodology**



Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory

JOHN McHUGH
Carnegie Mellon University

In 1998 and again in 1999, the Lincoln Laboratory of MIT conducted a comparative evaluation of intrusion detection systems (IDSs) developed under DARPA funding. While this evaluation represents a significant and monumental undertaking, there are a number of issues associated with its design and execution that remain unsettled. Some methodologies used in the evaluation are questionable and may have biased its results. One problem is that the evaluators have published relatively little concerning some of the more critical aspects of their work, such as validation of their test data. The appropriateness of the evaluation techniques used needs further investigation. The purpose of this article is to attempt to identify the shortcomings of the Lincoln Lab effort in the hope that future efforts of this kind will be placed on a sounder footing. Some of the problems that the article points out might well be resolved if the evaluators were to publish a detailed description of their procedures and the rationale that led to their adoption, but other problems would clearly remain.

Categories and Subject Descriptors: K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Invasive software* (e.g., viruses, worms, Trojan horses)

General Terms: Security

Additional Key Words and Phrases: Computer security, intrusion detection, receiver operating curves (ROC), software evaluation

1. INTRODUCTION

The most comprehensive evaluation of research on intrusion detection systems that has been performed to date is an ongoing effort by MIT's Lincoln Laboratory, performed under DARPA sponsorship. While this work is flawed in many respects, it is the only large-scale attempt at an objective

This work was sponsored by the U.S. Department of Defense.
Author's address: CERT® Coordination Center, Software Engineering Institute, Carnegie Mellon University, 4500 Fifth Ave., Pittsburgh, PA 15213.

Suricata (2010)

- Written by Open Information Security Foundation (OISF) in 2009
  - OISF founded by DHS and others
- Inspired by but improved on Snort
  - Added multithreading support
  - OISF development model prevents commercial capture
  - Snort-compatible rule syntax — easy migration path

NB. In 2023, Snort added multithreading …

Quantitative Analysis of Intrusion Detection Systems: Snort and Suricata

Joshua S. White[a], Thomas T. Fitzsimmons[b], Jeanna N. Matthews[c]
[a]Wallace H. Coulter School of Engineering
[b,c]Department of Computer Science
{[a]whitejs, [b]fitzsitd, [c]jnm}@clarkson.edu
Clarkson University, Potsdam, NY USA

ABSTRACT

Given competing claims, an objective head-to-head comparison of the performance of both Snort® and Suricata Intrusion Detection Systems is important. In this paper, we present a thorough, repeatable, quantitative, apples-to-apples comparison of the performance of *Snort* and Suricata. We compare both the default, "out of the box" performance of both systems and investigate alternative configurations such as multi-instance *Snort* and improvements to Suricata that were inspired by our initial results; changes to Suricata inspired by our testing resulted in performance improvements of up to 20X. We examine the performance of both systems as we scale system resources such as the number of CPU cores. We vary both the rulesets used and the workloads processed. Our results show that a single instance of Suricata is able to deliver substantially higher performance than a corresponding single instance of *Snort* and higher performance than multi-instance *Snort* as well.

Keywords: Intrusion Detection Systems, *Snort*, Suricata, Benchmark

1. INTRODUCTION

Any modern organization that is serious about security, deploys a network intrusion detection system (NIDS) to monitor network traffic for signs of malicious activity. The most widely deployed NIDS system is Snort®, an open source system originally released in 1998. *Snort* is a single threaded system that uses a set of clear text rules to instruct a base engine on proper reactions to particular traffic patterns as they are detected. In 2009, the US Department of Homeland Security and a consortium of private companies provided substantial grant funding to a newly created organization known as the Open Information Security Foundation (OISF). Its purpose was to build a multi-threaded alternative to *Snort*, called Suricata. Despite many similarities between *Snort* and Suricata, the OISF stated it was essential to replace the older single-threaded *Snort* engine with a multi-threaded system that could deliver higher performance and better scalability. Key *Snort* developers argued that Suricata's multi-threaded architecture would actually slow the detection process.

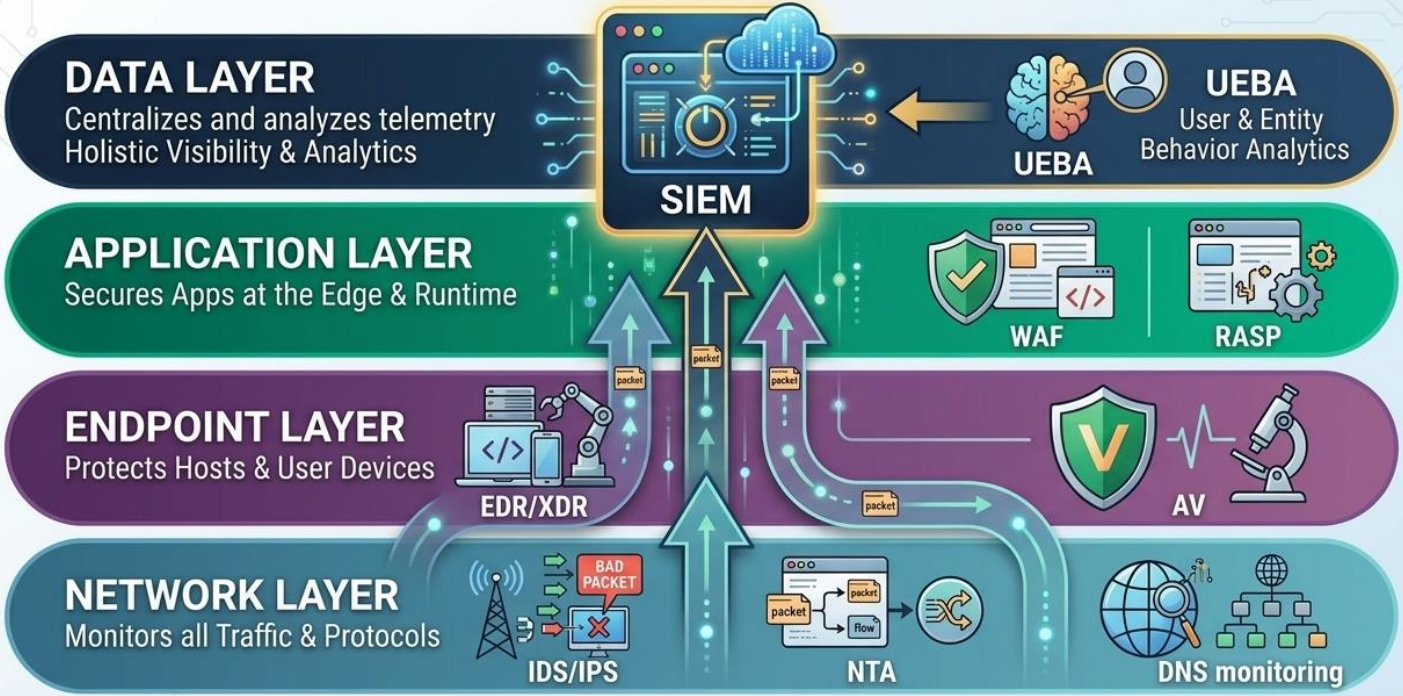
Given these competing claims, an objective head-to-head comparison of the performance of *Snort* and Suricata is important. In this paper, we present a comprehensive quantitative comparison of the two systems. We have developed a rigorous testing framework that examines the performance of both systems as we scale system resources through 24 cores. We compare Suricata to both single instance and multi-instance *Snort*. We document how changes in rule-sets and workloads affect the results. Our results led directly to substantial improvements (up to 20X) in the ability of Suricata to scale to large numbers of cores.

Network Intrusion Detection Systems (NIDS) capture and inspect network traffic for signs of malicious or unauthorized activity. NIDS can be compared on many dimensions including performance, scalability, accuracy, ability to detect different kinds of attacks, ease of use, the responsiveness of the development community to requests for new features, documentation quality, and many others. In this paper, we focus specifically on comparing the performance and scalability of *Snort* and Suricata. Keeping up with all the traffic on a busy network is a performance intensive activity. If the NIDS is unable to keep up with the traffic in real-time, then uninspected packets are either dropped, causing problems for legitimate traffic or allowed to flow, causing problems for security.

In both *Snort* and Suricata, a base engine is controlled by a set of rules. Each rule describes network activity that is considered malicious or unwanted by specifying the content of network packets. Each rule also specifies an action to be taken in the event that a packet is suspect, such as raising an alert or dropping the packet. The base engine must read each

Detecting Attacks Today

LAYERED CYBERSECURITY DETECTION TECHNOLOGIES



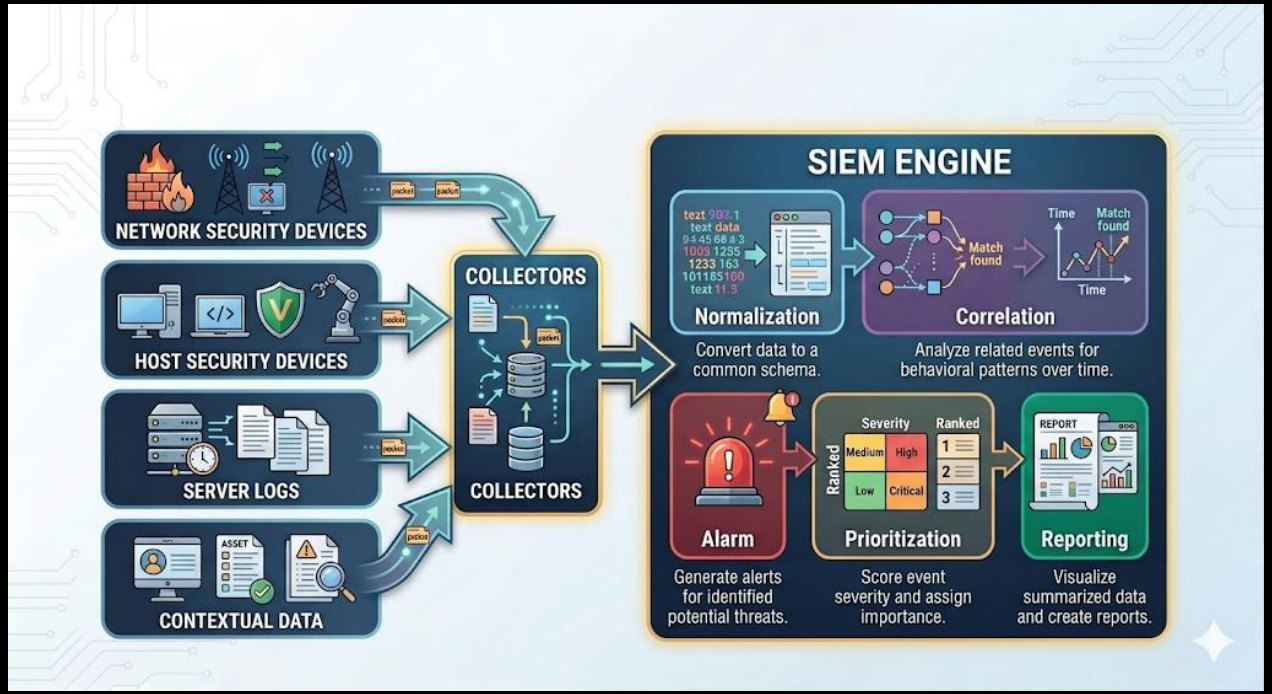
Unified Threat Visibility and Correlation through Integrated Telemetry

Logging and SIEM

SIEM = Security Information and Event Management

- Aggregate logs from endpoints, servers, network, cloud
- Normalize, correlate, detect
- Platforms: Splunk, Microsoft Sentinel, Elastic Security, IBM QRadar

SIEM Architecture



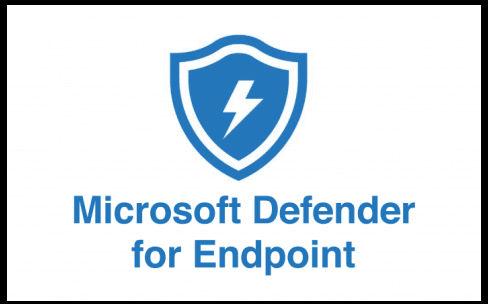
What to Log

- Authentication events (logins, failures, MFA)
- Privilege escalation
- Process execution (especially unusual binaries)
- Network connections (especially outbound)
- File changes (especially on sensitive files)
- DNS queries
- Cloud API calls

Endpoint Detection and Response (EDR)

Continuous monitoring of endpoint activity

- Process trees, file operations, registry changes, network connections
- **Behavioral detection**: suspicious patterns, not just known signatures
- Telemetry recording for forensic investigation
- Response: isolate host, kill process, quarantine file



XDR: Extending Beyond Endpoints

Extended Detection and Response

- Correlates across endpoints, network, email, identity, cloud
- Unified view of an incident across multiple attack surfaces
- Reduces alert fatigue through automated correlation

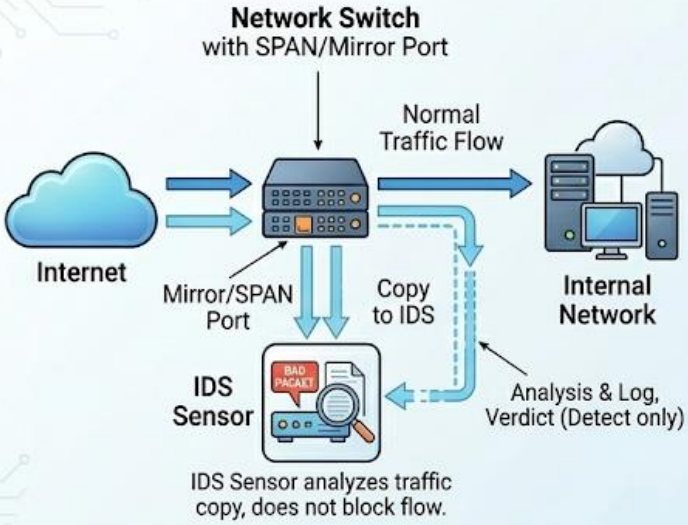
Comparing XDR with SIEM:

- SIEM is a **correlation bus for heterogeneous, already-processed data**.
- XDR is a **unified detection platform over raw, cross-domain telemetry from an integrated collection stack**.

Network-Based Detection: Overview

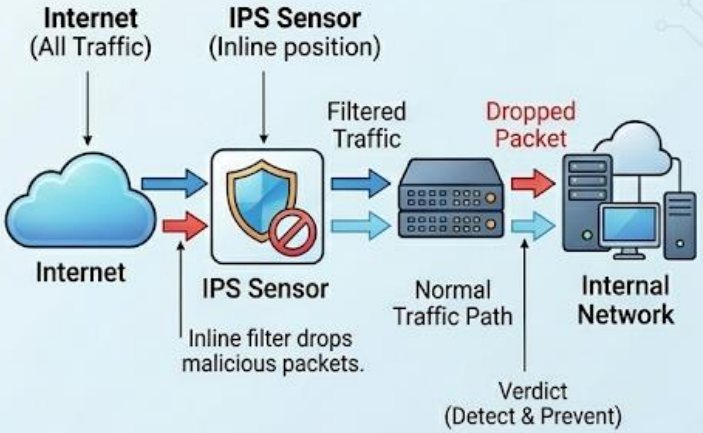
Approach	What It Does	Examples
Signature IDS/IPS	Match known attack patterns	Suricata, Snort
Network Traffic Analysis	Detect anomalous traffic patterns	Zeek, Darktrace
DNS Monitoring	Detect tunneling, DGA, malicious domains	Passive DNS sensors

Intrusion Detection System
(Passive Monitoring)



Normal traffic path is not modified.
Verdict does not affect flow.

Intrusion Prevention System
(Active Prevention)



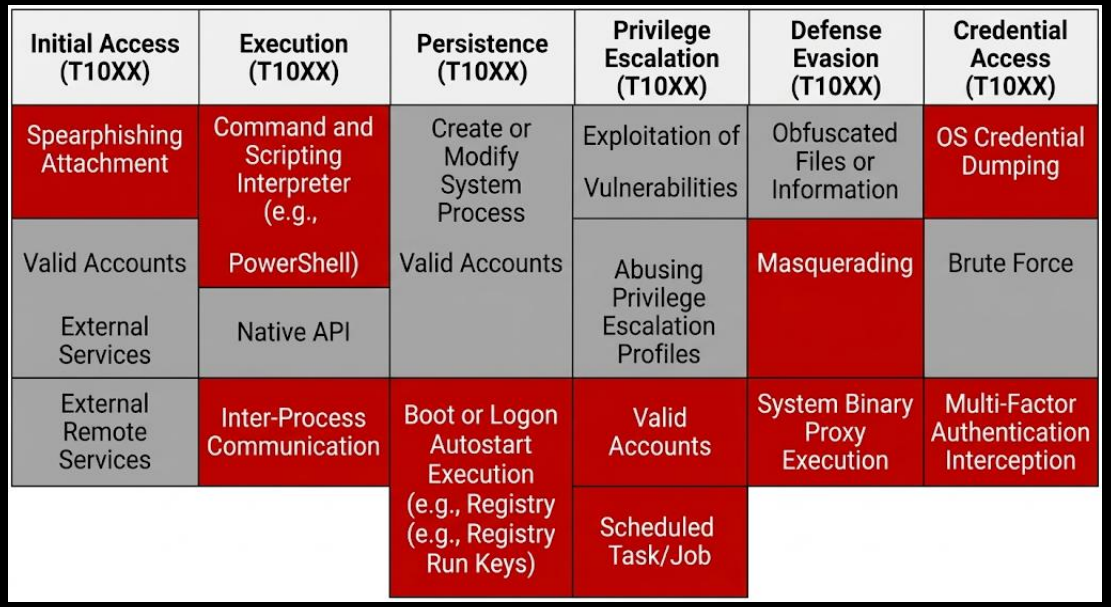
All traffic must pass through the IPS.
Malicious traffic can be dropped in real-time.

Vulnerability Scanning and AV

- **Vulnerability scanners** (Nessus, Qualys, OpenVAS)
  - Authenticated and unauthenticated scanning
  - Integration with patch management
- **Anti-virus / Anti-malware**
  - Traditional signature-based detection
  - Increasingly subsumed by EDR
- **Cloud Security Posture Management (CSPM)**
  - Detect misconfigurations (public S3 buckets, overly permissive IAM)
  - Tools: Prisma Cloud, AWS Security Hub, Wiz

Threat Intelligence

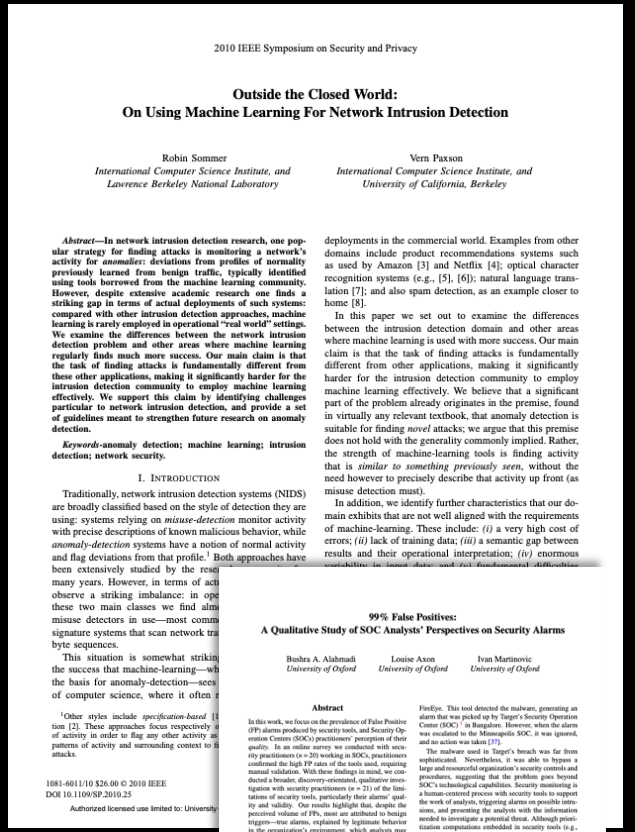
- **Threat feeds**: STIX/TAXII, MISP
- **MITRE ATT&CK**: framework mapping attacker techniques
- Use intelligence to **prioritize** detection engineering



Initial Access (T10XX)	Execution (T10XX)	Persistence (T10XX)	Privilege Escalation (T10XX)	Defense Evasion (T10XX)	Credential Access (T10XX)
Spearphishing Attachment	Command and Scripting Interpreter (e.g., PowerShell)	Create or Modify System Process	Exploitation of Vulnerabilities	Obfuscated Files or Information	OS Credential Dumping
Valid Accounts		Valid Accounts	Abusing Privilege Escalation Profiles	Masquerading	Brute Force
External Services	Native API				
External Remote Services	Inter-Process Communication	Boot or Logon Autostart Execution (e.g., Registry (e.g., Registry Run Keys)	Valid Accounts	System Binary Proxy Execution	Multi-Factor Authentication Interception
			Scheduled Task/Job		

MITRE ATT&CK

Challenges of IDS: Theory Meets Reality



2010 IEEE Symposium on Security and Privacy

Outside the Closed World:
On Using Machine Learning For Network Intrusion Detection

Robin Sommer
International Computer Science Institute, and
Lawrence Berkeley National Laboratory

Vern Paxson
International Computer Science Institute, and
University of California, Berkeley

***Abstract*—In network intrusion detection research, one popular strategy for finding attacks is monitoring a network's activity for *anomalies*: deviations from profiles of normality previously learned from benign traffic, typically identified using tools borrowed from the machine learning community. However, despite extensive academic research one finds a striking gap in terms of actual deployments of such systems: compared with other intrusion detection approaches, machine learning is rarely employed in operational "real world" settings. We examine the differences between the network intrusion detection problem and other areas where machine learning regularly finds much more success. Our main claim is that the task of finding attacks is fundamentally different from these other applications, making it significantly harder for the intrusion detection community to employ machine learning effectively. We support this claim by identifying challenges particular to network intrusion detection, and provide a set of guidelines meant to strengthen future research on anomaly detection.**

***Keywords*-anomaly detection; machine learning; intrusion detection; network security.**

I. Introduction

Traditionally, network intrusion detection systems (NIDS) are broadly classified based on the style of detection they are using: systems relying on *misuse-detection* monitor activity with precise descriptions of known malicious behavior, while *anomaly-detection* systems have a notion of normal activity and flag deviations from that profile.[1] Both approaches have been extensively studied by the rese
many years. However, in terms of act
observe a striking imbalance: in ope
these two main classes we find alm
misuse detectors in use—most comm
signature systems that scan network tra
byte sequences.

This situation is somewhat strikin
the success that machine-learning—wh
the basis for anomaly-detection—sees
of computer science, where it often

[1]Other styles include *specification-based* [
tion [2]. These approaches focus respectively
of activity in order to flag any other activity a
patterns of activity and surrounding context to
attacks.

1081-6011/10 $26.00 © 2010 IEEE
DOI 10.1109/SP.2010.25

deployments in the commercial world. Examples from other domains include product recommendations systems such as used by Amazon [3] and Netflix [4]; optical character recognition systems (e.g., [5], [6]); natural language translation [7]; and also spam detection, as an example closer to home [8].

In this paper we set out to examine the differences between the intrusion detection domain and other areas where machine learning is used with more success. Our main claim is that the task of finding attacks is fundamentally different from other applications, making it significantly harder for the intrusion detection community to employ machine learning effectively. We believe that a significant part of the problem already originates in the premise, found in virtually any relevant textbook, that anomaly detection is suitable for finding *novel* attacks; we argue that this premise does not hold with the generality commonly implied. Rather, the strength of machine-learning tools is finding activity that is *similar to something previously seen*, without the need however to precisely describe that activity up front (as misuse detection must).

In addition, we identify further characteristics that our domain exhibits that are not well aligned with the requirements of machine-learning. These include: *(i)* a very high cost of errors; *(ii)* lack of training data; *(iii)* a semantic gap between results and their operational interpretation; *(iv)* enormous

99% False Positives:
A Qualitative Study of SOC Analysts' Perspectives on Security Alarms

Bushra A. Alahmadi
University of Oxford

Louise Axon
University of Oxford

Ivan Martinovic
University of Oxford

Abstract

In this work, we focus on the prevalence of False Positive (FP) alarms produced by security tools, and Security Operation Centers (SOCs) practitioners' perception of their *quality*. In an online survey we conducted with security practitioners ($n = 20$) working in SOCs, practitioners confirmed the high FP rates of the tools used, requiring manual validation. With these findings in mind, we conducted a broader, discovery-orientated, qualitative investigation with security practitioners ($n = 21$) of the limitations of security tools, particularly their alarms' quality and validity. Our results highlight that, despite the perceived volume of FPs, most are attributed to benign triggers—true alarms, explained by legitimate behavior in the organization's environment, which analysts may choose to ignore. To properly evaluate security tools' adequacy and performance, it is critical that vendors and researchers are able make such distinctions between types of FP. Alarm validation is a tedious task that can cause alarm burnout and eventually desensitization. Therefore, we investigated the process of alarm validation in SOCs, identifying factors that may influence the outcome of this process. To improve security alarm quality, we elicit five properties (**R**eliable, **E**xplainable, **A**nalytical, **C**ontextual, **T**ransferable) required to foster effective and quick validation of alarms. Incorporating these requirements in future tools will not only reduce alarm burnout but improve SOC analysts' decision-making process by generating interpretable and meaningful alarms that enable prompt reaction.

1 Introduction

In 2013, Target was hit by the most prominent retail hack in U.S. history, in which attackers infiltrated Target's network, installing malware designed to steal customers' credit card data. Months before the breach, Target had installed a new $1.6m malware-detection technology by FireEye. This tool detected the malware, generating an alarm that was picked up by Target's Security Operation Center (SOC) [1] in Bangalore. However, when the alarm was escalated to the Minneapolis SOC, it was ignored, and no action was taken [37].

The malware used in Target's breach was far from sophisticated. Nevertheless, it was able to bypass a large and resourceful organization's security controls and procedures, suggesting that the problem goes beyond SOC's technological capabilities. Security monitoring is a human-centered process with security tools to support the work of analysts, triggering alarms on possible intrusions, and presenting the analysts with the information needed to investigate a potential threat. Although prioritization computations embedded in security tools (e.g., Security Information and Event Management—SIEM) can do much of the heavy lifting, SOC practitioners face the difficult task of figuring out which alarms are False Positives (FPs) and which indicate something dangerous.

Security tool vendors have been competing on the ability of their tools to detect threats, hence, focusing on metrics such as False Negatives (FNs)—*failure to detect security events when they occur*. In contrast, more focus needs to be placed on reducing the FPs—*flagging a security event when it is not a threat*, which is equally critical [11]. Analysts often spend time manually going through alarms to determine their validity, as well as performing monotonous tasks to reduce FPs. Such tasks include reconfiguring security tools, baselining normal behavior, and filtering out noise, when time could be spent on detecting more sophisticated attacks (i.e., threat hunting). Analysts are also under constant pressure to close tickets, as some SOCs evaluate their practitioners' performance accordingly [40]. This process not only leads to human error [13] but causes analyst fatigue and burnout [40].

An excessive number of alarms, in any system, contributes to alarm desensitization, mistrust, and lack of hu-

[1]We provide a background on SOCs in Appendix A.1

with some insights from:

Challenge 1: The Base-Rate Fallacy

Even with **99.9% true positive** and **0.1% false positive** rates:

If attacks are **0.001%** of traffic →

For every 1 true alert, ~100 false alerts

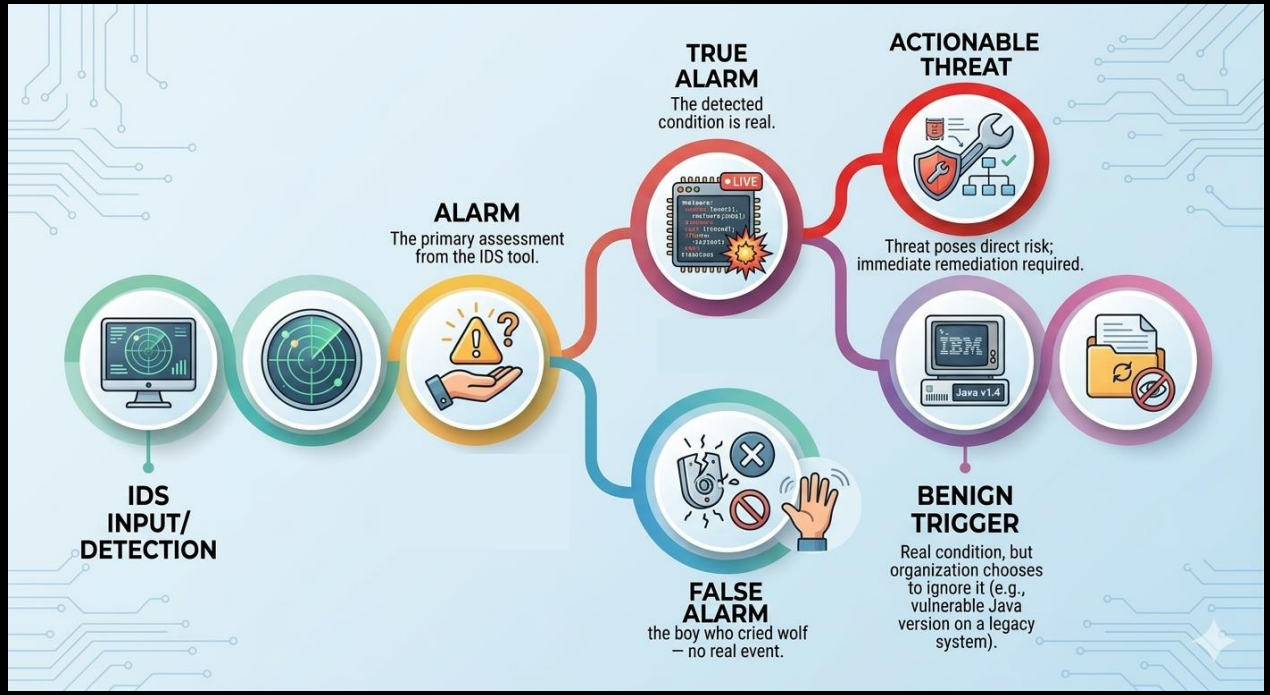
2022 Confirmation: “99% False Positives”

> “We know **99% of the alarms** we generate are false positives, but we still have to look at them.” — SOC Analyst B3

- 45% of analysts receive **<5K alerts daily**; some receive **>100K**
- One analyst: **1 in 100** alerts investigated is an actual threat
- Another: **50 in every 200** are legitimate (but benign)

The base-rate problem isn’t theoretical — it’s daily reality

False Alarms vs. Benign Triggers



Challenge 2: The Semantic Gap

ML system: “This flow is anomalous”

Signature IDS: “This is CVE-2024-1234 exploiting a buffer overflow in OpenSSH 8.9”

Operators need to know **what** the alert means and **what to do**

2022 Confirmation: “Black Box” Tools

Analysts describe commercial security tools as:

- *“Untrustworthy”, “unuseful”, “poor”*
- *“Sometimes it’s a generic filter… you don’t know what logic they put inside”*
- *“If you don’t tell me the reason you fire, you’re not ready to open up”*

Only **10%** of surveyed SOCs use ML-based tools

Challenge 3: Adversarial Adaptation and Drift

- Attackers craft traffic to **evade** ML models
- **Concept drift**: “normal” changes constantly (new apps, users, configs)
- **No clean training data**: real networks always have some compromise

Alahmadi et al. (2022) — Change management failures generate false alarms
“We were seeing a huge amount of ICMP traffic… the networking team deployed a new tool and nobody told us”

Challenge 4: Evaluation Methodology

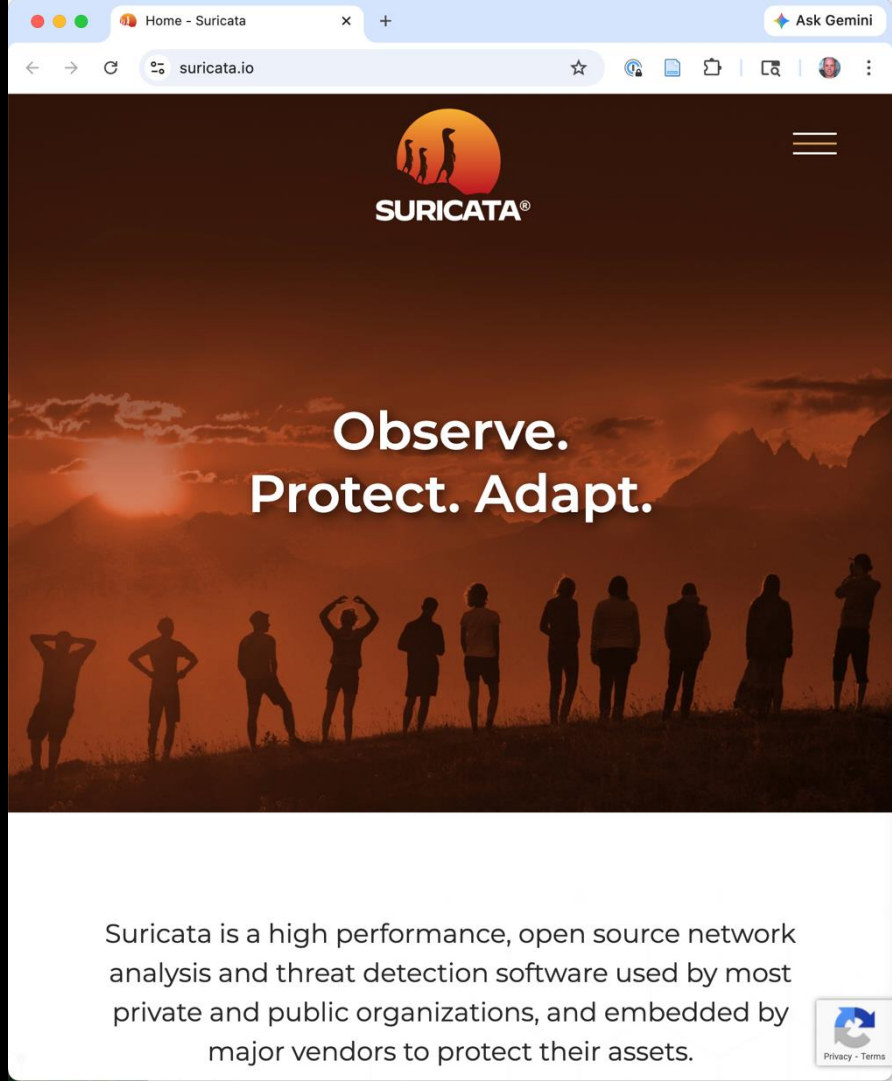
- Most papers use **synthetic/outdated datasets** (KDD Cup 1999, DARPA eval)
- **No operational testing** — lab results don't transfer to deployment
- **Cherry-picked metrics** that obscure the base-rate problem
- **No comparison with simple baselines** — is ML actually better than rules?

The DARPA evaluation flaws we discussed earlier: **still being repeated**

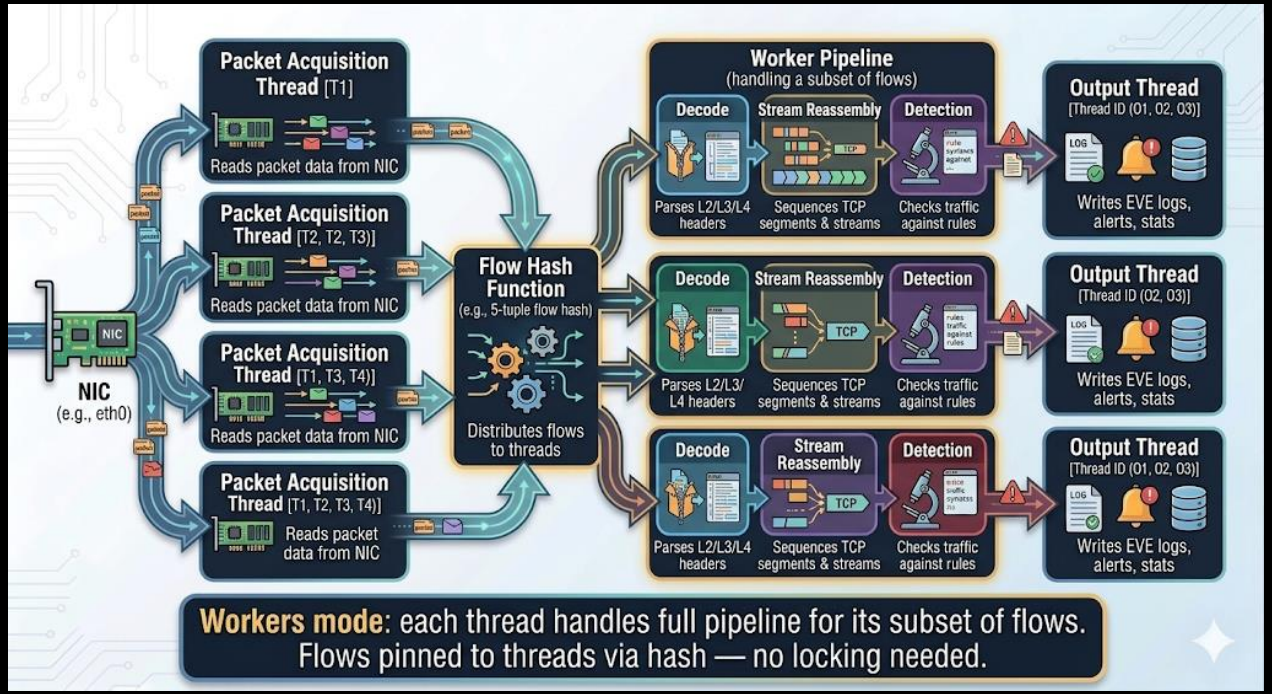
Sommer & Paxson: Recommendations

1. Evaluate on **realistic, modern traffic**
2. Report **operationally meaningful metrics** (not just ROC curves)
3. Provide **semantic context** with alerts
4. Consider the **adversarial setting** explicitly
5. Focus ML on **well-scoped sub-problems**
6. **Compare against simple baselines**
7. **Deploy and test in real environments**

Suricata: Next-Gen Network IDS/IPS



Suricata Multi-Threading Architecture

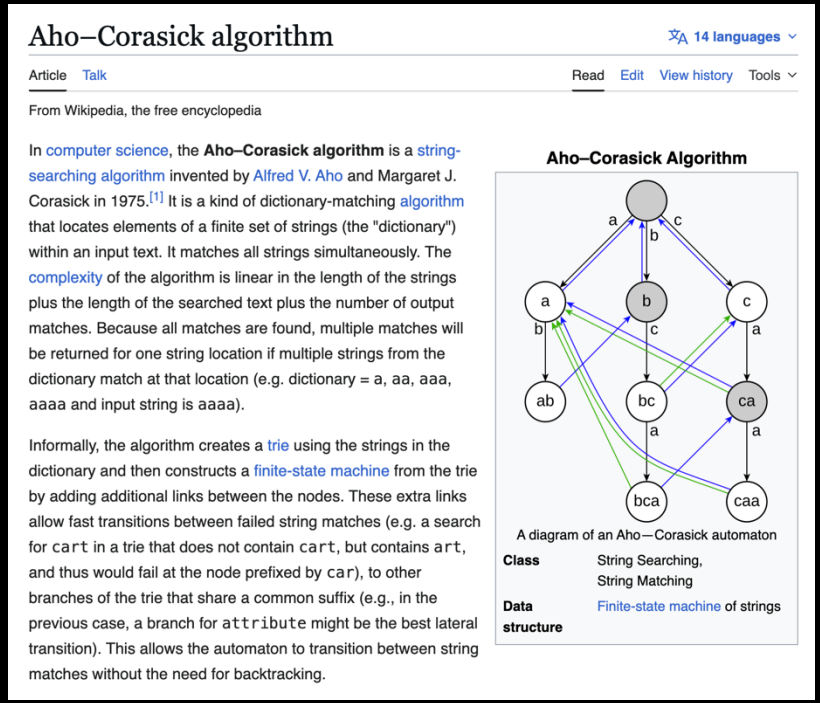


Suricata Detection Engine

Multi-Pattern Matcher (MPM) using Aho-Corasick algorithm:

1. All `content` strings from all rules compiled into **one** automaton
2. Scan payload **once** against all patterns simultaneously
3. Only rules whose content matched proceed to **full evaluation**

Also: **20+ protocol parsers** (HTTP, TLS, DNS, SMB, SSH, …) run independently of rules



Aho–Corasick algorithm

14 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

In computer science, the **Aho–Corasick algorithm** is a string-searching algorithm invented by Alfred V. Aho and Margaret J. Corasick in 1975.[1] It is a kind of dictionary-matching algorithm that locates elements of a finite set of strings (the "dictionary") within an input text. It matches all strings simultaneously. The complexity of the algorithm is linear in the length of the strings plus the length of the searched text plus the number of output matches. Because all matches are found, multiple matches will be returned for one string location if multiple strings from the dictionary match at that location (e.g. dictionary = a, aa, aaa, aaaa and input string is aaaa).

Informally, the algorithm creates a trie using the strings in the dictionary and then constructs a finite-state machine from the trie by adding additional links between the nodes. These extra links allow fast transitions between failed string matches (e.g. a search for `cart` in a trie that does not contain `cart`, but contains `art`, and thus would fail at the node prefixed by `car`), to other branches of the trie that share a common suffix (e.g., in the previous case, a branch for `attribute` might be the best lateral transition). This allows the automaton to transition between string matches without the need for backtracking.

Aho–Corasick Algorithm

A diagram of an Aho–Corasick automaton

Class	String Searching, String Matching
Data structure	Finite-state machine of strings

Suricata Rule Anatomy

```
alert http $HOME_NET any -> $EXTERNAL_NET any
(
    msg:"Possible malicious download";
    flow:established,to_server;
    content:"GET"; http_method;
    content:"/malware.exe"; http_uri;
    sid:2001001; rev:1;
)
```

Rule **Header**: Actions

Action	Behavior
`alert`	Generate an alert
`pass`	Stop evaluation, allow packet
`drop`	Drop packet + alert (IPS mode only)
`reject`	Send TCP RST or ICMP unreachable + drop

Rule Header: Protocols and Addresses

Protocols: `tcp`, `udp`, `icmp`, `ip` + application-layer: `http`, `tls`, `dns`, `ftp`, `ssh`, `smtp`, `smb`, …

Addresses: IPs, CIDRs, variables (`$HOME_NET`), groups, negation

Direction: `->` (unidirectional), `<>` (bidirectional)

```
alert dns $HOME_NET any -> any any (
    msg:"DNS query for suspicious TLD";
    dns_query; content:".xyz"; endswith;
    sid:2001002; rev:1;
)
```

Rule Options: Content Matching

- `content:"pattern"` — match bytes (text or `|hex|`)
- `nocase` — case-insensitive
- `depth`, `offset`, `distance`, `within` — positional constraints
- `pcre:"/regex/"` — Perl-compatible regex
- **Sticky buffers** (protocol-aware):
  - `http_uri`, `http_host`, `http_user_agent`, `http_header`
  - `dns_query`, `tls_sni`, `ja3.hash`
  - `file_data` (file content)

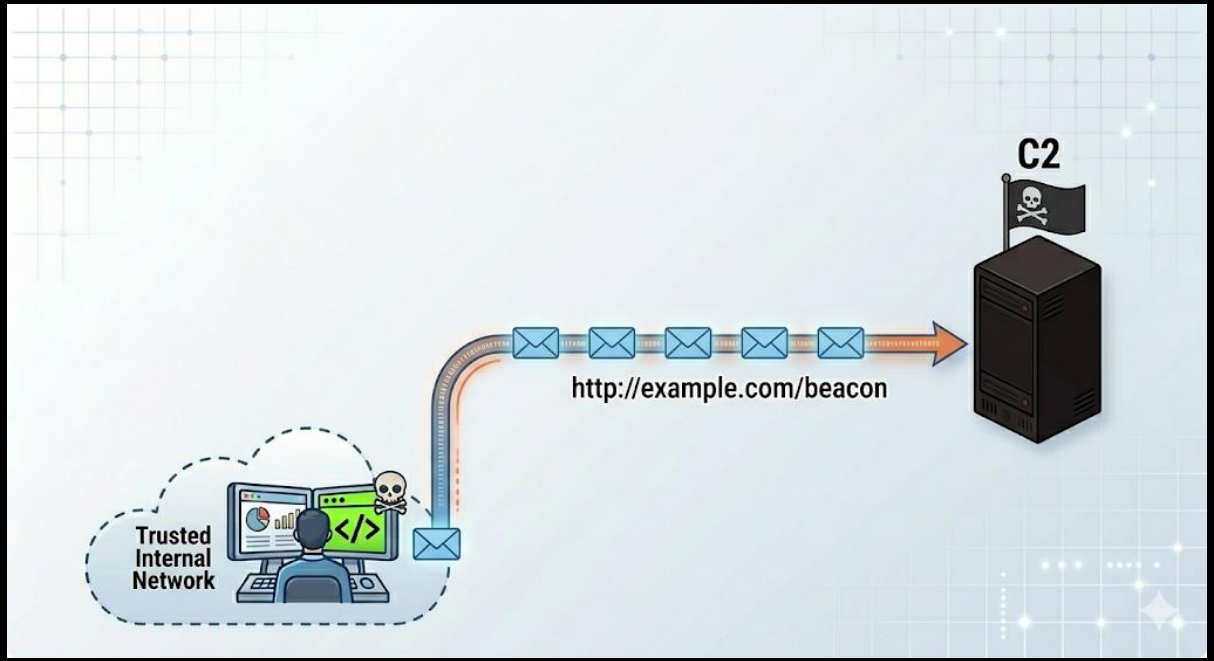
Rule Options: Flow and State

```
# Match only on established connections, client to server
flow:established,to_server;

# Stateful detection across packets with flowbits
```

```
alert http any any -> any any (
    msg:"S1: suspicious beacon";
    content:"/beacon"; http_uri;
    flowbits:set,beacon_detected;
    sid:2001003;
)
```

```
alert http any any -> any any (
    msg:"S2: data exfil after beacon";
    flowbits:isset,beacon_detected;
    content:"POST"; http_method;
    content:"/upload"; http_uri;
    sid:2001004;
)
```

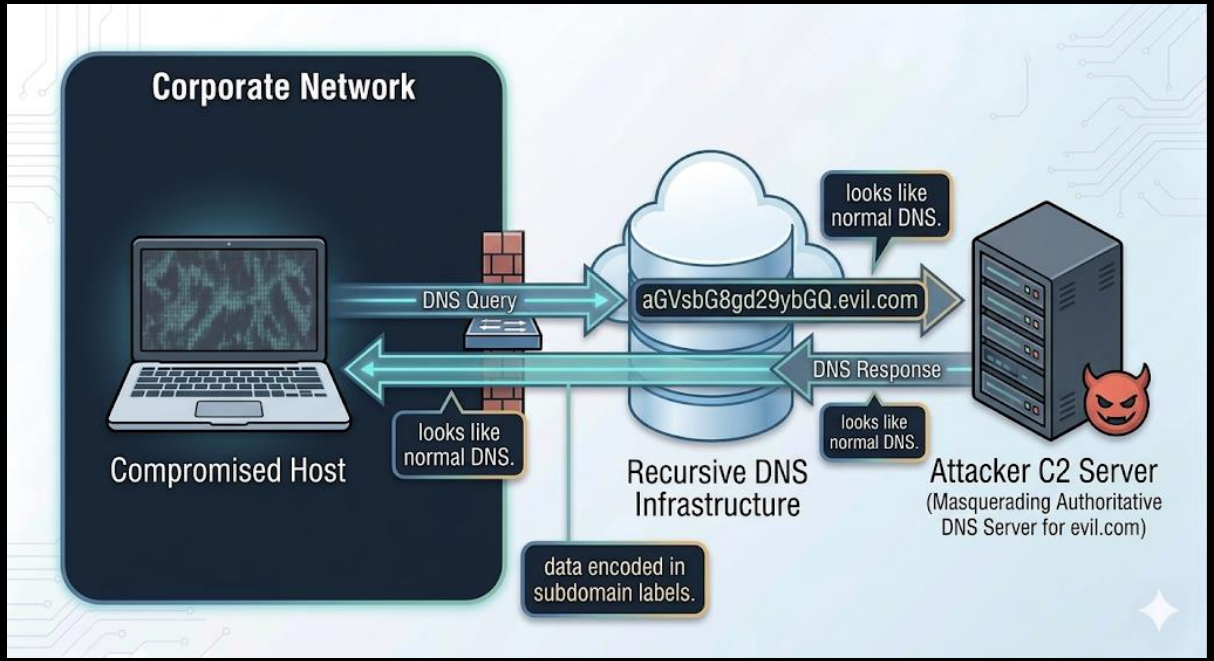
Example: Malware C2 Beaconing



Malware C2 Beaconing: Sample Rule

```
# Detect C2 beaconing by interval regularity
alert http $HOME_NET any -> $EXTERNAL_NET any (
    msg:"Possible C2 beacon - regular interval";
    flow:established,to_server;
    content:"GET"; http_method;
    threshold:type threshold, track by_src,
              count 10, seconds 60;
    sid:9000001; rev:1;
)
```

Example: DNS Tunneling



DNS Tunneling: Sample Rule

```
# Detect DNS tunneling by query length
alert dns $HOME_NET any -> any any (
    msg:"Possible DNS tunneling - long query";
    dns_query; content:"|00|";
    byte_test:1,>,50,0,relative;
    sid:9000002; rev:1;
)
```

EVE JSON: Suricata's Output

Single JSON log file where each line is a self-contained event:

- `alert` — signature match with rule metadata and payload
- `http` — full HTTP transaction logs
- `dns` — query and answer records
- `tls` — SNI, certificate info, JA3/JA4 hashes
- `flow` — connection metadata, byte/packet counts
- `fileinfo` — extracted file metadata and hashes

Running Suricata

```
# IDS mode on interface
suricata -c /etc/suricata/suricata.yaml -i eth0

# IPS mode via NFQUEUE
suricata -c /etc/suricata/suricata.yaml -q 0

# Offline PCAP analysis
suricata -c /etc/suricata/suricata.yaml -r
capture.pcap

# Rule management
suricata-update    # Download/update rulesets
```

Suricata Configuration Highlights

Key sections in `suricata.yaml`:

- **`vars`**: define `$HOME_NET`, `$EXTERNAL_NET`
- **`default-rule-path`** + **`rule-files`**: where rules live
- **`af-packet`** / **`pcap`**: capture method
- **`threading`**: number of worker threads
- **`detect-engine`**: MPM algorithm (Aho-Corasick, Hyperscan)
- **`app-layer`**: enable/disable protocol parsers
- **`outputs`**: configure EVE JSON, fast.log, file extraction

Suricata: Summary

Suricata gives you **fast, signature-based detection**:

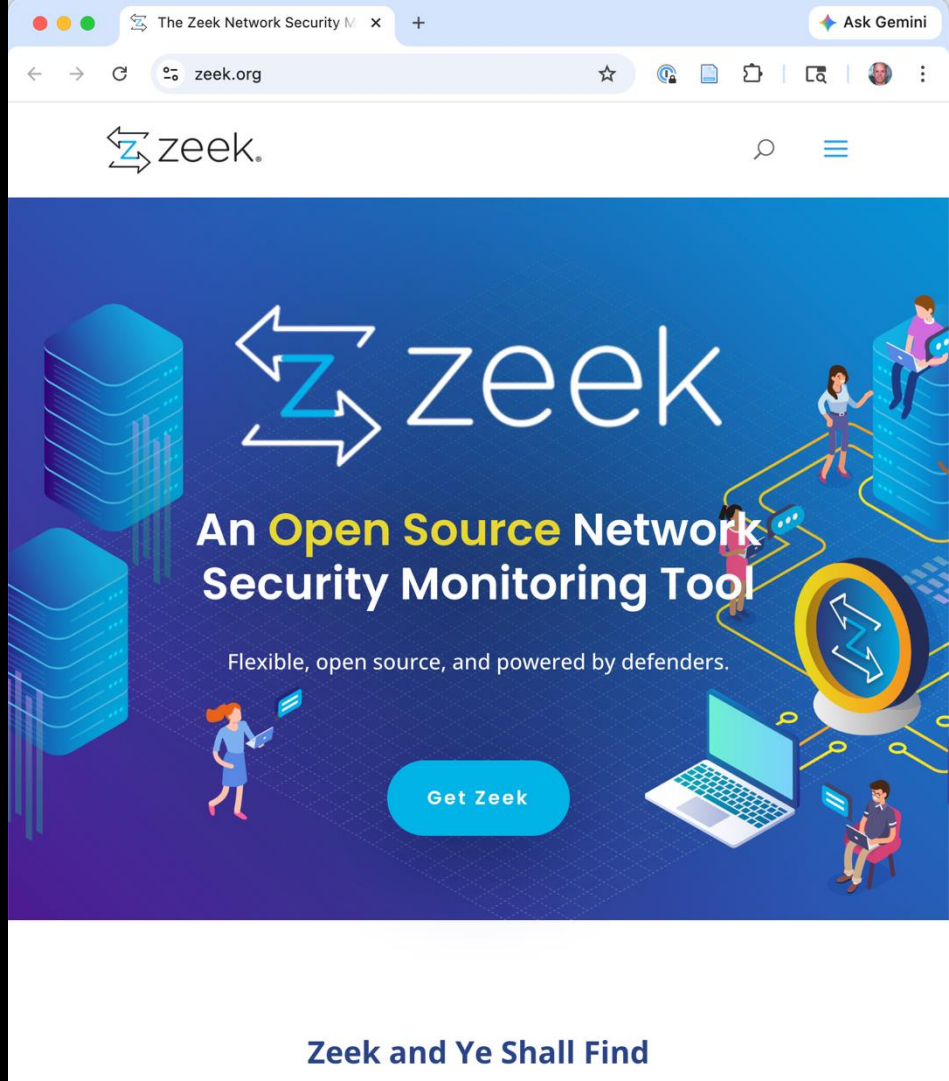
- Matches packets against thousands of rules efficiently
- Produces **alerts** when known attack patterns are found
- Also logs protocol metadata (EVE JSON)

What Suricata doesn't do well: answer "what else happened?"

- If it alerts on a C2 beacon, what other connections did that host make?
- What DNS queries preceded the alert? What files were transferred?

For that, we need **Zeek**

Zeek: Network Security Monitor

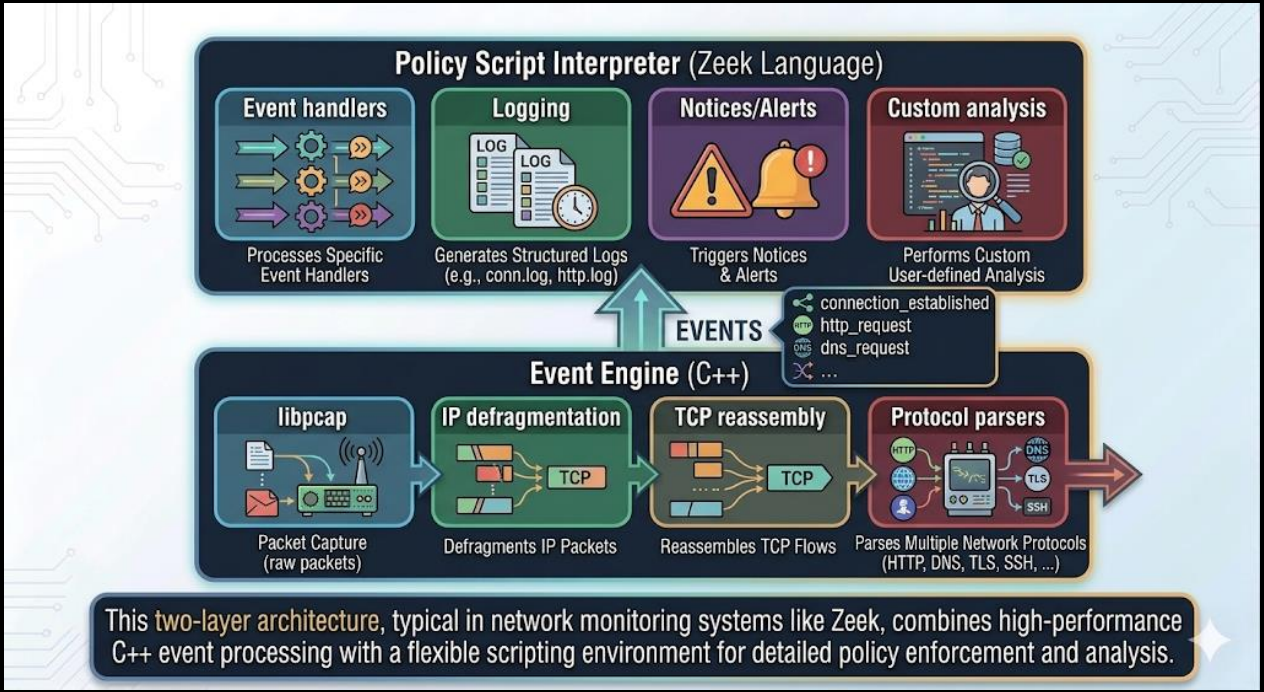


What is Zeek?

A fundamentally **different philosophy** from Suricata/Snort:

	Suricata	Zeek
Question	“Does this match a known bad pattern?”	“What is happening on this network?”
Output	Alerts when rules match	Structured logs of **all** activity
Detection	Signature-based (fast, specific)	Scriptable (flexible, open-ended)
Forensics	Limited to alert context	Complete network audit trail

Zeek Architecture



Zeek Log Types

Log File	Contents
conn.log	Every connection: IPs, ports, protocol, duration, bytes
dns.log	DNS queries and responses
http.log	HTTP method, URI, host, user-agent, status code
ssl.log	TLS SNI, certificate info, cipher suite, JA3 hashes
files.log	Files transferred: MIME type, size, MD5/SHA256
weird.log	Unusual protocol events
notice.log	Alerts raised by Zeek scripts

Anomaly signal

Judgment output

The Zeek uid: Linking Everything Together

```
# conn.log
ts          uid              id.orig_h     id.resp_h       id.resp_p
1609459200  CYwp7n1a2gkHQ    10.0.0.5      93.184.216.34   80

# http.log
ts          uid              method  host            uri
1609459200  CYwp7n1a2gkHQ    GET     example.com     /malware.exe

# files.log
ts          uid              mime_type         md5
1609459200  CYwp7n1a2gkHQ    application/exe   d41d8cd98f00b204...
```

Zeek Scripting Language

```
event http_request(c: connection, method: string,
                   original_URI: string,
                   unescaped_URI: string, version: string)
    {
    if ( /\.exe$/ in original_URI )
        NOTICE([$note=HTTP::Suspicious_Download,
                $msg=fmt("EXE download: %s%s",
                         c$http$host, original_URI),
                $conn=c]);
    }
```

Zeek Scripting Language

- Domain-specific, **event-driven** language
  - All relevant handlers for an event will run, and there can be many events that match a packet (can be prioritized)
- Strongly typed: `addr`, `port`, `subnet`, `time`, `interval`, `pattern`, … all strictly checked
- Rich data structures: Tables, sets, records
- `redef` keyword for customization without modifying base scripts (for constants, variables, some record types)

More sophisticated state than Suricata

```
# A table mapping IP addresses to a count of connections
global connection_counts: table[addr] of count;

event new_connection(c: connection)
    {
    local src = c$id$orig_h;

    if ( src in connection_counts )
        connection_counts[src] += 1;
    else
        connection_counts[src] = 1;
    }
```

Many built-in event types

Connection lifecycle

- `new_connection(c: connection)` — fires when any new connection is first seen
- `connection_established(c: connection)` — TCP handshake completed
- `connection_finished(c: connection)` — connection closed cleanly
- `connection_rejected(c: connection)` — RST received
- `connection_timeout(c: connection)` — connection expired without closing

DNS

- `dns_request(c, msg, query, qtype, qclass)` — any DNS query
- `dns_A_reply(c, msg, ans)` — A record response specifically
- `dns_AAAA_reply(...)`, `dns_MX_reply(...)`, `dns_TXT_reply(...)` — per record type

TLS/SSL

- `ssl_client_hello(…)`, `ssl_server_hello(…)`
- `x509_certificate(f, cert, chain)` — certificate observed

Define your own starting from these

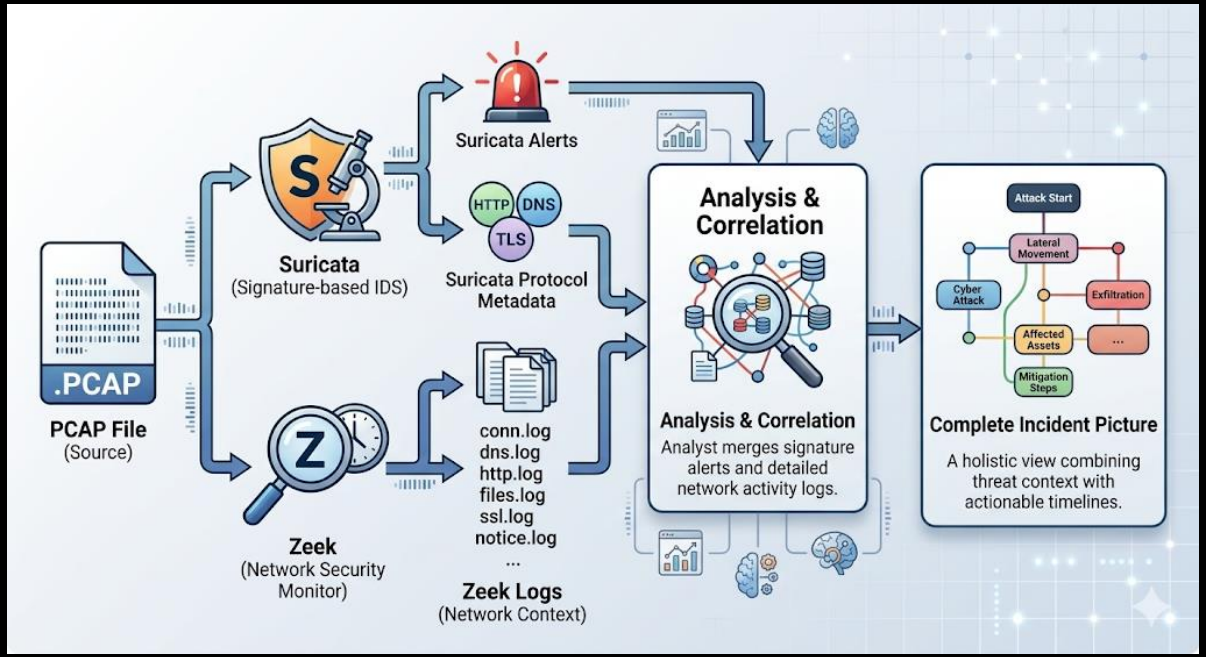
Running Zeek

```
# Analyze a PCAP file (logs into current directory)
zeek -r capture.pcap

# Live capture on an interface
zeek -i eth0

# Enable JSON output
zeek -r capture.pcap LogAscii::use_json=T

# Extract specific fields with zeek-cut
cat conn.log | zeek-cut id.orig_h id.resp_h
id.resp_p duration
```

Suricata + Zeek: Better Together

Suricata tells you **“something bad happened”**

Zeek tells you **“here’s everything that happened before, during, and after”**



`zeek-cut` useful for extracting specific fields from Zeek's tab-separated logs

Suricata + Zeek: Better Together

- Why not Zeek for everything?
  - Too slow: If you tried to express Suricata's 30,000-rule Emerging Threats ruleset as Zeek scripts checking payload bytes, you would almost certainly fall behind line rate on a busy network.
  - Related: Would fight against script language design of higher level abstractions to match individual packets (quickly)
  - Zeek is purely passive: Because it operates by building up multi-packet abstractions it is only designed for detection, not prevention (by dropping packets inline)

Security Operations Centers

What a SOC Does

The **organizational unit** that staffs and operates the tools we've discussed

- **Continuously monitors** the IT environment
- **Detects and triages** alerts
- **Investigates** and **contains** incidents
- **Coordinates** eradication and recovery

Alahmadi et al. interviewed practitioners across **7 SOCs** — their findings ground this section

99% False Positives:
A Qualitative Study of SOC Analysts' Perspectives on Security Alarms

Bushra A. Alahmadi, *University of Oxford*
Louise Axon, *University of Oxford*
Ivan Martinovic, *University of Oxford*

Abstract

In this work, we focus on the prevalence of False Positive (FP) alarms produced by security tools, and Security Operation Centers (SOCs) practitioners' perception of their *quality*. In an online survey we conducted with security practitioners ($n = 20$) working in SOCs, practitioners confirmed the high FP rates of the tools used, requiring manual validation. With these findings in mind, we conducted a broader, discovery-orientated, qualitative investigation with security practitioners ($n = 21$) of the limitations of security tools, particularly their alarms' quality and validity. Our results highlight that, despite the perceived volume of FPs, most are attributed to benign triggers—true alarms, explained by legitimate behavior in the organization's environment, which analysts may choose to ignore. To properly evaluate security tools' adequacy and performance, it is critical that vendors and researchers are able make such distinctions between types of FP. Alarm validation is a tedious task that can cause alarm burnout and eventually desensitization. Therefore, we investigated the process of alarm validation in SOCs, identifying factors that may influence the outcome of this process. To improve security alarm quality, we elicit five properties (**R**eliable, **E**xplainable, **A**nalytical, **C**ontextual, **T**ransferable) required to foster effective and quick validation of alarms. Incorporating these requirements in future tools will not only reduce alarm burnout but improve SOC analysts' decision-making process by generating interpretable and meaningful alarms that enable prompt reaction.

1 Introduction

In 2013, Target was hit by the most prominent retail hack in U.S. history, in which attackers infiltrated Target's network, installing malware designed to steal customers' credit card data. Months before the breach, Target had installed a new $1.6m malware-detection technology by FireEye. This tool detected the malware, generating an alarm that was picked up by Target's Security Operation Center (SOC) [1] in Bangalore. However, when the alarm was escalated to the Minneapolis SOC, it was ignored, and no action was taken [37].

The malware used in Target's breach was far from sophisticated. Nevertheless, it was able to bypass a large and resourceful organization's security controls and procedures, suggesting that the problem goes beyond SOC's technological capabilities. Security monitoring is a human-centered process with security tools to support the work of analysts, triggering alarms on possible intrusions, and presenting the analysts with the information needed to investigate a potential threat. Although prioritization computations embedded in security tools (e.g., Security Information and Event Management—SIEM) can do much of the heavy lifting, SOC practitioners face the difficult task of figuring out which alarms are False Positives (FPs) and which indicate something dangerous.

Security tool vendors have been competing on the ability of their tools to detect threats, hence, focusing on metrics such as False Negatives (FNs)—*failure to detect security events when they occur*. In contrast, more focus needs to be placed on reducing the FPs—*flagging a security event when it is not a threat*, which is equally critical [11]. Analysts often spend time manually going through alarms to determine their validity, as well as performing monotonous tasks to reduce FPs. Such tasks include reconfiguring security tools, baselining normal behavior, and filtering out noise, when time could be spent on detecting more sophisticated attacks (i.e., threat hunting). Analysts are also under constant pressure to close tickets, as some SOCs evaluate their practitioners' performance accordingly [40]. This process not only leads to human error [13] but causes analyst fatigue and burnout [40].

An excessive number of alarms, in any system, contributes to alarm desensitization, mistrust, and lack of bu-

[1] We provide a background on SOCs in Appendix A.1

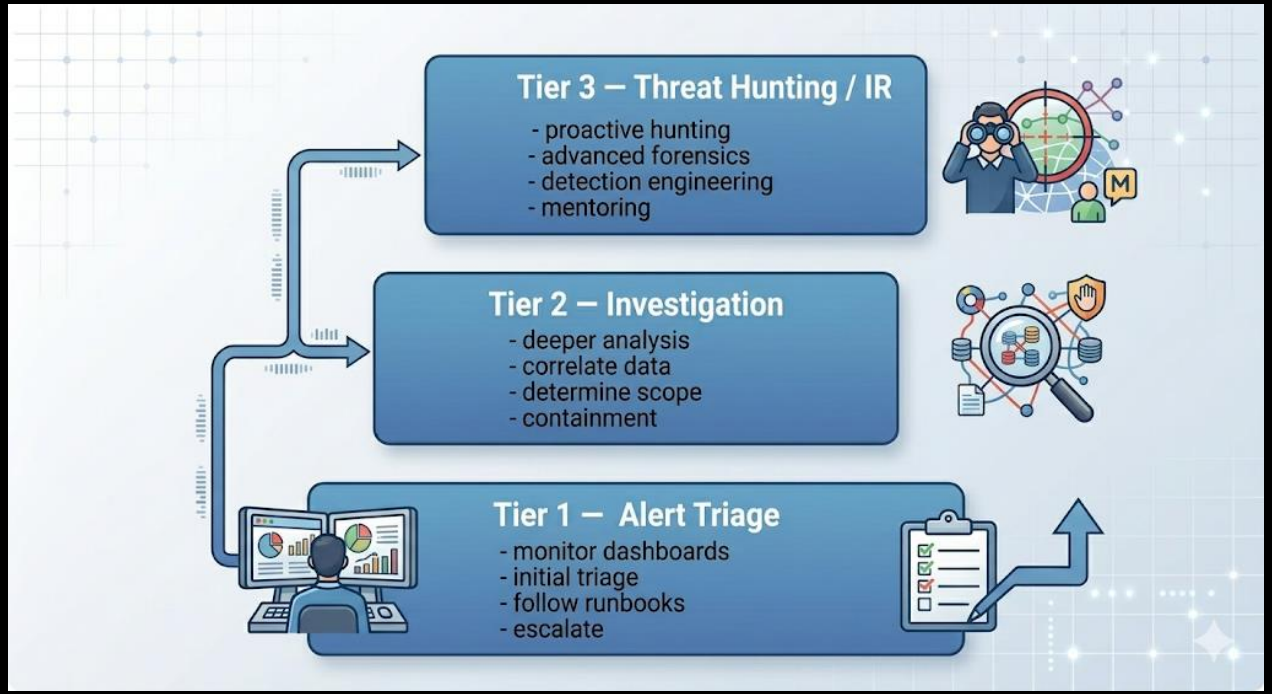
SOC Staffing Models

Model	Trade-offs
In-house	Full control and org knowledge; expensive 24/7 staffing
MSSP / MDR	24/7 coverage without headcount; less organizational context
Hybrid	Vendor handles Tier 1 and off-hours; in-house handles escalations

Alahmadi et al. found that MSSP analysts face pressure to over-report:

"Being analysts, most of them are afraid not to raise them to the customer… they do tend to raise quite a lot of alarms"

Tiered Analyst Structure



The Target Breach: Detection Without Response

- 2013: Target deployed **$1.6M FireEye** malware detection system
- FireEye **detected** the malware and generated an alert
- Alert picked up by SOC in **Bangalore**
- Escalated to **Minneapolis** SOC — **ignored, no action taken**
- Result: **40 million** credit card numbers stolen

The **central operational challenge** for SOCs: Analysts may see **thousands to 100K+ alerts per day** (Alahmadi et al.)
Consequences: alerts ignored, real attacks missed, analyst burnout

Four Limitations of Security Alarms

Limitation	Problem
Unreliable	Signatures use volatile features (IPs, domain names); loosely written
Unexplainable	Black-box tools; short/generic descriptions
Lack context	No awareness of network topology, business function, asset criticality
Not transferable	Each environment is unique; one-size-fits-all fails

The REACT Model for Better Alarms

Alarms should be:

- **R**eliable — use stable features; incorporate analyst feedback
- **E**xplainable — communicate reasoning, not just “anomalous”
- **A**nalytical — support human reasoning and hypothesis testing
- **C**ontextual — incorporate network, business, and environmental context
- **T**ransferable — adapt to different environments via transfer learning

Other Alert Fatigue Mitigations

Strategies:

- **Tuning**: refine rules to reduce noise (Alahmadi: “the tedious part is FPs that we deal with”)
- **Enrichment**: add context automatically (asset criticality, threat intel)
- **SOAR automation**: auto-close known FP patterns, auto-escalate high-confidence alerts
- **Detection quality metrics**: track FP rate per rule; retire noisy rules

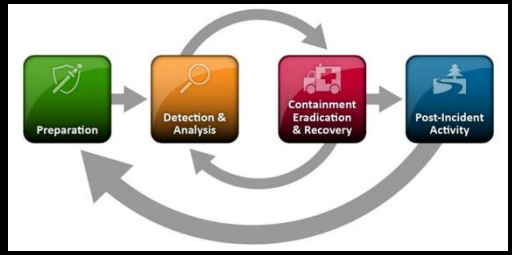
The Human Element in SOC Operations

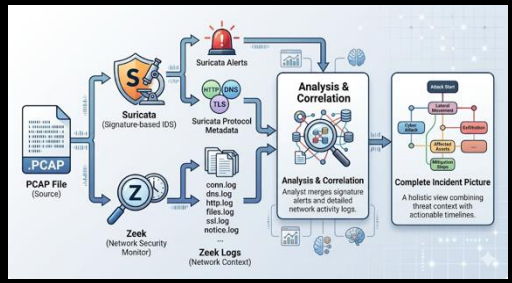
Alahmadi et al.'s key findings:

- **55%** of analysts rely on **tacit knowledge and intuition**
- **58%** process alerts based on knowledge of **regular network activity**
- Alarm validation requires understanding the **business**, not just the network
  - *"It's not just about what you see in the tools, it's about what you know of the customer and the customer's nature — business nature"*
- **Threat hunting** (Tier 3): proactive, hypothesis-driven search for threats that evade automated detection

Key Takeaways

1. The **NIST incident lifecycle** (six phases) frames how organizations handle security events
2. IDS concepts trace from **Anderson (1980)** through **Denning (1987)** to modern tools
3. The **base-rate fallacy** and **semantic gap** remain fundamental challenges (Sommer & Paxson, confirmed by Alahmadi et al.)
4. **Zeek** provides network visibility; **Suricata** provides fast signature matching — use both
5. SOCs are where technology meets **human judgment**
6. **Alert fatigue** is the central operational challenge; alarms should be **REACT**





References

- Anderson, J. “Computer Security Threat Monitoring and Surveillance.” Technical report, 1980.
- Denning, D. “An Intrusion-Detection Model.” *IEEE TSE*, 1987.
- Heberlein, T. et al. “A Network Security Monitor.” *IEEE S&P*, 1990.
- McHugh, J. “Testing Intrusion Detection Systems.” *ACM TISSEC*, 2000.
- Sommer, R. and Paxson, V. “Outside the Closed World: On Using Machine Learning for Network Intrusion Detection.” *IEEE S&P*, 2010.
- Alahmadi, B., Axon, L., and Martinovic, I. “99% False Positives: A Qualitative Study of SOC Analysts’ Perspectives on Security Alarms.” *USENIX Security*, 2022.
- Wan, G., Gong, F., Barbette, T., and Durumeric, Z. “Retina: Analyzing 100 GbE Traffic on Commodity Hardware.” *SIGCOMM*, 2022.
- Zeek Documentation: docs.zeek.org
- Suricata Documentation: docs.suricata.io