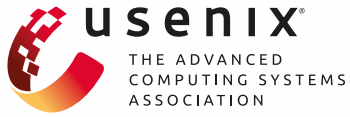

CAUDIT: Continuous Auditing of SSH Servers To Mitigate Brute-Force Attacks

Phuong M. Cao, Yuming Wu, and Subho S. Banerjee, ***UIUC;***
Justin Azoff and Alex Withers, ***NCSA;*** **Zbigniew T. Kalbarczyk and Ravishankar K. Iyer,** ***UIUC***

https://www.usenix.org/conference/nsdi19/presentation/cao

This paper is included in the Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI '19).

February 26–28, 2019 • Boston, MA, USA

ISBN 978-1-931971-49-2

Open access to the Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI '19) is sponsored by

NetApp®

CAUDIT: Continuous Auditing of SSH Servers to Mitigate Brute-Force Attacks

Phuong M. Cao[1], Yuming Wu[1], Subho S. Banerjee[1], Justin Azoff[2,3], Alexander Withers[3], Zbigniew T. Kalbarczyk[1], Ravishankar K. Iyer[1]

[1]*University of Illinois at Urbana-Champaign,* [2]*Corelight,* [3]*National Center for Supercomputing Applications*

Abstract

This paper describes CAUDIT[1], an operational system deployed at the National Center for Supercomputing Applications (NCSA) at the University of Illinois. CAUDIT is a fully automated system that enables the identification and exclusion of hosts that are vulnerable to SSH brute-force attacks. Its key features include: 1) a honeypot for attracting SSH-based attacks over a `/16` IP address range and extracting key metadata (e.g., source IP, password, SSH-client version, or key fingerprint) from these attacks; 2) executing audits on the live production network by replaying of attack attempts recorded by the honeypot; 3) using the IP addresses recorded by the honeypot to block SSH attack attempts at the network border by using a Black Hole Router (BHR) while significantly reducing the load on NCSA's security monitoring system; and 4) the ability to inform peer sites of attack attempts in real-time to ensure containment of coordinated attacks. The system is composed of existing techniques with custom-built components, and its novelty is its ability to execute at a scale that has not been validated earlier (with thousands of nodes and tens of millions of attack attempts per day). Experience over 463 days shows that CAUDIT successfully blocks an average of 57 million attack attempts on a daily basis using the proposed BHR. This represents a 66× reduction in the number of SSH attempts compared to the daily average and has reduced the traffic to the NCSA's internal network-security-monitoring infrastructure by 78%.

1 Introduction

Security auditing of large-scale networks is challenging due to the constantly evolving configurations of networked devices [1, 2]. Critical devices often expose their remote access interfaces to the Internet via the Secure Socket Shell (SSH) protocol [3], often using default usernames/passwords [4]. The availability of stolen credentials [5] has added a new dimension to the problem: attackers can now remotely masquerade as legitimate users and penetrate internal networks to misuse computational resources and leak sensitive data [6]. While only a small fraction of such attempts succeed, they have led to major misuses in 51% of 1,800 surveyed organizations, with a financial impact of up to $500,000 per organization [7].

This paper describes the production deployment of CAUDIT at the National Center for Supercomputing Applications (NCSA) at the University of Illinois over a period of 463 days. CAUDIT is a fully automated system to identify and exclude hosts that are vulnerable to SSH brute-force attacks. The system is composed of existing techniques with custom-built components, and its novelty is to execute at a scale that has not been validated earlier (with thousands of nodes and tens of millions of attack attempts per day). The key components of the proposed system are as follows.

- An SSH-based honeypot deployed on an entire `/16` classless inter-domain routing (CIDR) network[2] that mimics a realistic server farm of 65,536 machines. In contrast with other honeypots [8–12], ours is *non-interactive*, i.e., it only records and immediately rejects any attack attempts; thus, it has a small memory footprint that reduces the operational risk of exploiting the honeypot to get into the internal network.

- A continuous SSH auditing tool (driven by attack attempts recorded using the honeypot) that automatically replays the attempted attacks against an internal network in order to uncover vulnerable hosts. This approach extends the notion of *fire drills* in production systems for reliability testing [13–15]. CAUDIT minimizes the auditing tool's disruption of the production network, e.g., by subscribing to SSH protocol activities by using the deep packet inspection capabilities of existing network security monitors such as the Bro IDS [16].

- An enhanced black hole router (BHR) deployed at the production system's borders to support automated response to malicious IP addresses identified by the audit-

[1] https://pmcao.github.io/caudit

[2] The address space belonged to a Fortune 50 company but has been transferred to NCSA.

ing tool. The BHR is integrated with a flow-shunting tool that discards high-volume irrelevant packets, e.g., virtual private network (VPN) traffic, before they get parsed by the kernel's networking stack. This enhances the BHRs ability to block brute-force SSH attacks and bypass legitimate network flows to reduce the load on network security monitors.

- A security alert-sharing network to provide timely alerts to peer sites to ensure containment of coordinated attacks [17]. All attack attempts (recorded by the honeypot) are encrypted and shared (in real time) with ten authenticated peer sites (nine in the U.S. and one in Singapore).

Placing CAUDIT in perspective. Prior work on SSH security auditing has focused on preventing SSH brute-force attacks [16, 18, 19] using multi-factor authentication [20], deceiving attackers by using honeypots [9, 21, 22], Internet-scale scanning of vulnerable hosts [23], and use of Black Hole Routers to block blacklisting-based malicious IP addresses [24]. Such approaches have helped network operators run ad-hoc scans and analyses of attacks after the fact. Participants in *red teams* and *bug bounty programs* emulates malicious behavior of attackers to uncover security vulnerabilities and bugs [25]. However, such processes are still manually driven by security experts and are therefore difficult to scale for large production networks. The above techniques are often ineffective in practice, e.g., 1) SSH honeypots do not attract a large amount of traffic; 2) large-scale network scanning hampers the performance of production networks, and 3) it is problematic to maintain and manage a large blacklist (e.g., because of false positive filtering). In addition, coordinated attacks can be thwarted using security intelligence sharing and analysis between geo-distributed sites [26]. Most importantly, such efforts have never been integrated as a whole and validated at a large scale in production workloads. Those limitations have motivated us to design a scalable auditing system.

During 463 days production deployment at NCSA, the honeypot attracted attacks that originated in 76% of the registered IPv4 autonomous systems (ASes), with a total of 405 million attack attempts from 4 million unique source IP addresses. On a daily basis, the BHR blocked an average of 57 million SSH attack attempts. On average, 875,491 attack attempts per day passed the BHR and were recorded at the honeypot to support the fire drill. Our operational experiences were as follows.

- The BHR augmented with the fire-drill resulted in a 66× reduction in the number of SSH attempts compared to the daily average. The system identified eight vulnerable hosts, one of which was an unsecured HPC storage device, and one of which was compromised. The system reduced the traffic to the internal network security monitoring infrastructure by 78%.
- We investigated a new observation on SSH attack attempts that use keys. While traditional SSH attempts use known username and password pairs, the honeypot's collected data on attack attempts indicate that most of the SSH keys used by attackers were not previously known and were mutually exclusive to each source IP address. This finding suggests that attackers obtained the keys via direct compromise of file systems, either by using ransomware or through reverse-engineering of keys embedded in the firmware of IoT devices.
- Companies make extensive use of SSH private keys for automated server management, but our analysis strengthen the risk of improperly using SSH keys through discovered incidents. Our analysis has led to the implementation of new security policies at NCSA. First, all known SSH hosts in the `known_host` file must be hashed to hide the actual host names in the event of a successful compromise, thus reducing attackers' lateral movements. Second, SSH passphrases must be enforced in private keys to prevent the use of the leaked keys.

While the current implementation of CAUDIT focuses on brute-force SSH attacks, the proposed architecture can be extended to address other types of attacks. We has open-sourced our implementation.[3]

2 Background

This section provides an overview of the daily operations at NCSA and the typical threats targeting its infrastructure.

2.1 Daily operations at NCSA

NCSA provides integrated cyberinfrastructure that includes computing, data, networking, and visualization resources to enable research of scientists and engineers at the University of Illinois at Urbana-Champaign (UIUC) and across the country. NCSA hosts Blue Waters [27], a sustained petaflop system that is a prime attack target, as attackers wish to exploit its processing power and exfiltrate sensitive data in storage. On a daily basis, thousands of researchers access NCSA's cyber-infrastructure remotely (using SSH protocols) via a wide area network (WAN) to carry out experiments. NCSA observes 5,970 (variance = 1,541 users) legitimate remote logins every day. At the same time, we observed an average of 875,491 credential-guessing activities ($^{405{,}352{,}245\text{ attacks}}/_{463\text{ days}}$), which are 147× more frequent than legitimate login activities ($^{875{,}491\text{ attack SSH}}/_{5{,}970\text{ legitimate SSH}}$). Several computing and data services at NCSA, e.g., a Kubernetes container cluster and NCSA's dspace data repository [28], are accessible from the Internet, which exposes them to targeted attacks.

2.2 System model

An SSH server is the most critical component of a host because it is typically the single point of entry for authenti-

[3] https://pmcao.github.io/caudit

cating remote users. Notwithstanding, many SSH servers are not properly guarded, e.g., exposed to the Internet while still using default credentials.

2.3 Threat model

This paper considers SSH credential-guessing attacks that originate in various sources, e.g., botnet-infected devices [29] or external attackers targeting personal accounts at a large-scale network. Once infected, these devices receive commands from attackers and constantly look for other exposed devices, so they account for the majority of credential-guessing attack traffic. We assume that attackers are not insiders, i.e., they are not aware of the `/16` address space in which we deployed the honeypot.

3 Motivation

This section describes a real credential-stealing incident that could have led to a data leak at NCSA, and shows the benefit of continuous auditing in securing large-scale networks.

3.1 A Motivating Example

In April 2018, NCSA's security team was notified of suspicious activity on a multiuser host supporting a major science project. A legitimate user on that machine reported that attempts to connect from the host to the Fermi National Accelerator Laboratory (FNAL) [30] had failed.

Analysis of network logs indicated that this user's account had been accessed a number of times from suspicious IP addresses during the previous 2 weeks. Cross-examination of the host's file system revealed that the SSH daemon binary file `/usr/bin/ssh` was different from the official version that should have been installed on the host. The modified file was related to suspicious downloads `181.215.xxx.yyy:24221/op3.tgz` and `182.215.xxx.yyy/sp.tgz` from a remote server. The file, `op3.tgz`, contained the source code for `OpenSSH v5.3.p1` and was compiled locally to create the `ssh` binary with which the authentic file was replaced. Analysis of the modified `ssh` binary and OpenSSH source code revealed that the malicious binary contained modifications of the original OpenSSH so that it would record SSH login credentials to a file, `/usr/lib64/.lib/lib64.so`. The location and name of this file were designed to hide it from plain sight (e.g., simply by running the `ls` command). An examination of the `lib64.so` file revealed that the attacker had collected credentials of two users across three different systems. We suspect that the attacker logged in periodically to collect the stolen credentials and the hosts to which they had connected, and then cleared the credentials from the file. A snippet of the malicious code is shown in Figure 1. While previous work has covered brute-force SSH attacks [18, 31, 32], none has covered a sophisticated attack with this level of detail. Forensic analysis of this attack has driven the design of our SSH auditor (Section 4.2) to identify potentially compromised SSH servers.

```
diff output for openssh/sshconnect2.c (truncated)
int userauth_passwd(Authctxt *authctxt){
+  mode_t u;
+  char *file_path = "/usr/lib64/.lib/lib64.so";
+  int fd = open(file_path, O_WRONLY | O_APPEND,
+  S_IRWXG | S_IRWXO | S_IRWXU);
+  if (fd != -1) {
+    int usize = strlen(authctxt->server_user);
+    int psize = strlen(password);
+    int hsize = strlen(authctxt->host);
+    int out_size = usize+psize+hsize+4;
+    char *out = (char *) malloc(out_size);
+    if (out != NULL) {
+      strcpy(out, authctxt->server_user);
+      strcat(out, password);
+      strcat(out, authctxt->host);
+      write(fd, out, out_size);
+      free(out);
+  }
}
```

Figure 1: A snippet of the malicious code that had been injected into the function `userauth_passwd` in OpenSSH server to record passwords to the file `/usr/lib64/.lib/lib64.so`

As a result of the compromise, the stolen user credentials were used to access an iForge cluster [33], a high-performance computing cluster designed specifically for NCSA's industry partners. Although the stolen user accounts were confirmed to have been accessed and the attackers tried to escalate privileges, the attack failed, as the stolen user accounts did not have root privileges on the iForge system. Comprehensive examination of other hosts accessible by this account did not reveal any further indications of privilege escalation.

Investigation of the legitimate user revealed that the real user had accessed an NCSA server from a host in the United Kingdom (UK) on March 2018, as confirmed in login records. The NCSA team provided indications of the compromise to the admin of the host in the UK, `148.197.xxx.yyy`, and the admin confirmed that they had indeed been compromised. Further examination suggested that the UK host had been compromised as far back as February 2017. Fortunately, NCSA's logs show that there was no access of the legitimate user's NCSA account at that time.

Remark. The compromise of this user's password likely occurred on the UK host. Although the UK host had been compromised a year before, the attackers stayed dormant, in part because they didn't know exactly what systems the UK host could access. Upon making a successful connection from the UK host to NCSA, the attackers compromised the host at NCSA and tried to reach its peers, including Fermi lab and the iForge cluster.

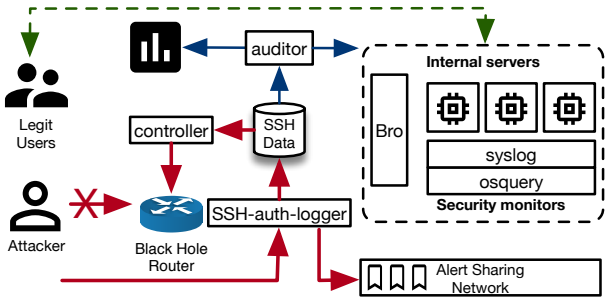

Figure 2: Overview of CAUDIT. Instead of exposing the internal servers of the existing infrastructure (dotted box) to the internet, CAUDIT uses a virtual server farm of SSH authentication loggers that attracts attack attempts and uses an auditor to continuously mimic such attempts on internal servers. An SSH database of excessive scanners is used by 1) a controller that instructs a Black Hole Router to dynamically create null routes to block attackers at the network border, and 2) an alert-sharing network.

4 System Architecture

This section first describes our architecture and implementation details. Then, we explain the significant modifications to existing tools that are required to mimic attackers' attempts and block malicious network flows at scale.

4.1 SSH authentication logger (SAL)

The core component of our approach is a lightweight, non-interactive SSH server (i.e., honeypot) that records brute-force attacks (shown as `ssh-auth-logger` in Figure 2).

Attracting attackers. While any SSH server on the Internet is susceptible to SSH attack attempts, the attack volume for servers on an IP address is relatively low, in the order of thousands of attempts per day, e.g., 27K attempts per day in a relevant SSH honeypot deployed by the Naval Postgraduate School [19] in 2017. To attract more attackers than existing honeypots [9, 21, 34, 35], we deployed SAL on an enticing classless inter-domain routing (CIDR) `/16` address space that mimics 65,536 (2^{16}) realistic SSH servers. Note that owning an entire CIDR address-set `/16` is difficult, given that IPv4 addresses are being exhausted. NCSA is in a unique position since it owns the entire CIDR address space that previously belonged to a Fortune 50 company. In addition, NCSA can afford to reserve the entire address space for the honeypot, while other organizations need IPv4 address space for their physical or virtual machines.

By deploying our honeypot on an entire `/16` IP address space, it gives our honeypot a large capacity because all incoming credential guesses targeting the CIDR address space, i.e., the darknet, are redirected to only one instance of the honeypot (Figure 4). Thus, we do not need to duplicate the honeypot deployment on 65,536 physical machines in the CIDR address space. Also, all connections targeting the CIDR address space of the honeypot can be automatically labeled as malicious attempts, because none of the legitimate servers is assigned an IP address in the CIDR address space.

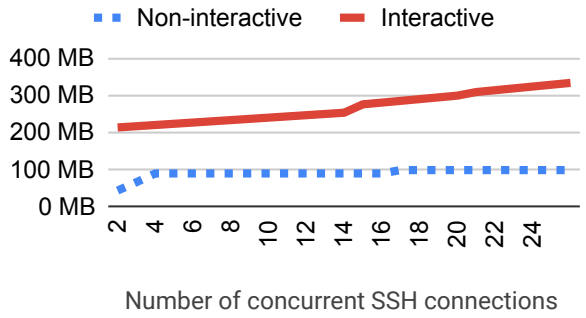

Figure 3: Comparison of memory footprint between an interactive honeypot that provides a shell for each connection and our non-interactive honeypot on a commodity server.

Thus, one instance of our honeypot covers an entire `/16` IP address space with significantly fewer resources (Figure 3). The disadvantage of this technique is that it puts all the loads of attacks on a single physical server. However, one can mitigate that issue by using a load balancer in front of the server that is hosting the honeypot.

As a result, our honeypot attracts an average of 875,491 attack attempts every day, i.e., 33× more than the one in [19].

Deceiving attackers. Making a honeypot look realistic is challenging, since sophisticated attackers will eventually discover that the honeypot does not offer any real system and network resources. Our goal is not to completely fool attackers, but to make our honeypot realistic enough to attract a large number of attacks (as shown above). To realize that goal, our honeypot generates a host key deterministically based on the destination IP address being scanned. Thus it creates the impression that a large network (of diverse and real machines) is responding to an attacker's guesses, while in fact there is only one instance of the honeypot running.

Isolation and memory footprint. Properly isolating a honeypot is difficult. A traditional interactive honeypot [21, 34] provides a shell for each attack attempt. Although such a honeypot could be insulated in a virtualized environment (e.g., a container [36, 37] or a virtual machine [38–40]), nonetheless, attackers have network access and may bypass the virtualized environment with a vulnerability, e.g., CVE-2017-5123, allowing the attack to escape from the container. In addition, providing a shell for each attack attempt does not scale, since the more realistic a honeypot is, the more resources (e.g., memory) are needed for deceiving the attackers. Figure 3 compares the memory footprint of an interactive

```
for x in \$(seq 1 254); do
  ip route add local 143.x.\${x}.0/24 dev p5p4
  ip rule add from 143.x.\${x}.0/24 table darknet
done
ip rule add from 141.x.y.z dev p5p4 table darknet
```

Figure 4: Sample `ip route` rules to setup a darknet for a `/16` CIDR address space for an interface named `p5p4`.

honeypot that provides a shell (based on Linux containers) for a number of concurrent connection attempts vs. our non-interactive honeypot. While the non-interactive honeypot maintains a constant memory usage of $\sim$ 98MB, the interative-honeypot uses linearly more memory for new containers as each new connection is made. Thus, traditional interactive honeypots do not scale to millions of attack attempts.

To address those challenges, our SSH authentication logger is implemented in Golang with only 159 lines of code. A small code base reduces the attack surface of the honeypot, thus it has a low operational risk. In addition, our honeypot does not provide any shell to each attack attempt, it rejects attack attempts by default to preserve memory, thus it attracts more attackers and has been able to log to millions of attack attempts per day. Furthermore, a small code base eases its deployment, i.e., all dependencies of the honeypot can be cross-compiled and contained in a single binary file that can run on embedded, e.g., ARM devices such as Raspberry Pi.

Attack attempts records. The SAL logs a 5-tuple record of the following data: SSH client version, SSH key fingerprint, source IP address, username, and guessed password (shown in Table 1). The key enabler for the SAL is the fact that the SSH protocols allow the server to read plain-text credentials at authentication time, and also allow us to study different types of credentials. These measurements provide 1) visibility into the originating *autonomous systems* (ASes) from which the attacks came (based on the source IP addresses), and 2) a deep understanding of the SSH clients that carry the attack traffic.

Identifying malicious scanners. Our goal is to have high precision in identifying malicious scanners (i.e., a low rate of false positives). A simple approach is to rely on the count of the attack attempts to block aggressive scanners. Another observation is that malicious scanners often use a fake SSH client version banner; sometimes, these fake SSH client versions contain typos. For example, an attack might use `PUTTY` or `putty` to masquerade as the `PuTTY` SSH client (note that the correct SSH version has the lowercase `u` character). This issue is analyzed in detail in Section 5.3.

Table 1: An example of an SSH record

Field	Example Value
source_ip	`123.201.xxx.yyy`
client_version	`SSH-2.0-libssh2_1.7.0`
key_fingerprint	`N/A`
username	`dspace`
password	`dspace@123`

key_fingerprint is intended for key-based authentication

4.2 SSH credential auditor (SCA)

Despite that existing tools such as the UNIX `passwd` utility [41, 42] can give warnings for dictionary-based passwords and misconfigurations, none of the available tools can be used as a fire drill, i.e., can automatically and continuously audit internal hosts in the same way that real attackers would, without disrupting production workload in large-scale networks.

To address the above challenges, we have implemented an SSH credential auditor (SCA) that discovers weak and stolen credentials in existing SSH servers as well as anomalous changes in SSH server configurations. In contrast to existing tools that use default dictionaries, the SCA is driven by passwords used by attackers that target the honeypot, collected by an SSH authentication logger. Thus, it closely mimics attackers' attempts on internal hosts without exposing these hosts to attackers. SCA works as follows.

Discovery of internal SSH hosts. In a large-scale network, scanning an entire IP space, e.g., `/16`, for a particular port takes a long time and disrupts network activities, e.g., triggers false alerts. Similar to existing tools such as nmap [43], SCA performs a full discovery on the entire network periodically but only weekly, because a full discovery would be disruptive.

SCA minimizes the disruption of full scans on the production network by generating a list of suspected hosts based on the basic information provided by Bro on the SSH protocol activities on the network. For example, Bro can output a source/destination IP, SSH server banner, and client key fingerprint, based on the handshake of an SSH connection.

Audit of SSH hosts. The SSH credential auditor performs following the audit schedules.

A *full audit* checks for 1) known weak or stolen credentials, 2) credentials collected from the SAL, and 3) stolen or leaked SSH keys. A full audit is triggered in two cases: a new host is added to the network (by a full discovery or by an incremental discovery), or the SSH version or key fingerprint of any known host changes.

A *partial audit* only checks for new credentials, e.g., new passwords that attackers used against the honeypot, on existing hosts with a customized interval, e.g., every day.

Localization and isolation of weak/compromised SSH hosts. Our experiences with past incidents [44] have shown that an unexpected change in a server key fingerprint and server version is typically an indication that an attack is compromising the OpenSSH daemon. Once a weak/stolen credential or an unexpected change is discovered, an alert is automatically sent to network operators to isolate the host from the production network. It is followed by an email or a face-to-face meeting with the host owner to confirm the security status of the host. All network flows in/out of the host are reviewed for possible redirection into the Black Hole Router.

4.3 Black hole router (BHR)

Although the Black Hole Router seems logically similar to blacklisting of IP addresses, our BHR's goal is not to block 100% of the attack attempts, but to reduce the loads on existing network security monitors. There is only one global

blacklist in the BHR, at the network border. That makes it easier to manage and to update the list (i.e., by removing inactive IP addresses from the list). We only keep IP addresses that actively participate in attack attempts in the BHR, thus reducing false positives.

Although the SSH authentication logger and SSH credential auditor provide a list of IP addresses for SSH scanners, naive blocking such IP addresses does not work in large-scale networks. Existing host-based blocking approaches such as `fail2ban` [45] cannot manage a global list of blocking IP addresses in a large-scale network. Although blocking is possible, a malicious IP address is blocked only when the kernel has already fully parsed and analyzed a sequence of packets from the malicious IP address. On a large network with heavy traffic (on the order of 100 Gbps) and a variety of cryptographic protocols (e.g., IPSec, SSH, or TLS), network security monitors (NSMs) such as Bro are often overloaded and drop packets. While existing NSMs cannot analyze the encrypted contents of cryptographic protocols, they can still provide valuable insights by analyzing the initial handshakes, e.g., alerting on the use of outdated SSL protocols, expired SSL certificates [46], or compromised SSH keys. Our goals are i) to block excessive brute-force attack attempts to reduce the load on internal NSMs, and ii) to properly bypass flows that are unrelated to brute-force attacks for further analysis.

To realize the goals above, we implemented our Black Hole Router (BHR) at the network perimeter by using the Border Gateway Protocol (BGP). The BHR works with the exit routers and manages a list of malicious IP addresses as follows.

Null route. Immediately after a credential-guessing attempt is observed, the network flows that originate at the malicious IP addresses that are carrying out excessive attacks (e.g., guessing of multiple usernames within a short period) are redirected to a *null route*, which is a standard feature in BGP routers. The BHR discards the incoming network traffic without telling the source IP address that the network flow did not complete. Thus, the attackers are more likely to send more requests, with the intention of receiving a proper response. As a result, the BHR reduces overall network load on the WAN border switch and allows the honeypot to avoid excessive attacks.

Catch-and-release. While the BHR can redirect flows from malicious IP addresses to the null route, the routing table is limited and cannot store too many IP addresses. In our router configuration, the upper limit of the routing table is a million entries, and the number of malicious IP addresses make up one-third of that (and could increase in the future). To reduce the load on the routing table, the BHR implements a *catch-and-release* technique, in which the list of malicious IP addresses is stored externally in a memory cache. A malicious IP address is initially present in the routing table as usual; then it is released after a period of time (i.e., an hour) of inactivity. When there is a new flow from a new IP address, the flow is compared with the memory cache, and the malicious IP address is re-inserted into the routing table if it is present in the cache.

Flow shunting. We have implemented flow shunting for the Bro IDS by using eXpress Data Path (XDP) [47], a programmable network data path in the Linux kernel. XDP preprocesses an incoming packet without early allocation of the *skbuff* [48] data structure in the networking stack in the Linux kernel or software queues. XDP works by looking at a specific offset in the packet, e.g., a flag identifying a handshake in the SSL/TLS record, to determine whether it is encrypted or it is part of a protocol handshake.

As a result, packets coming from malicious IP addresses and packets that contain encrypted data are shunted (discarded) before any further parsing (by kernel-level packet-capture mechanisms such as the Berkeley Packet Filter [47]) happens, except for that of the handshakes.

4.4 Alert-sharing Network (ASN)

Large-scale networks employ a variety of monitoring tools and corresponding analysis techniques to provide comprehensive coverage. Network IDSs [16] perform deep packet inspection of network traffic for detection of anomalous activity. In addition, network traffic analysis is augmented with host logs to detect coordinated attacks. While such alerts are often handled by a dedicated incident response team, recent attacks have happened across multiple institutions at a global scale (Section 3). Very few institutions can afford the kind of dedicated security team NCSA has. Thus, a new coordination effort is needed to prevent such attacks. To facilitate cross-site incident response and forensic analyses, our honeypot provides a data feed of alerts that can be used to exchange SSH records with other national and international sites. Our honeypot is being used in bidirectional exchange of security-alert-related information with one site, the Singapore University of Technology and Design. The major participating sites in the U.S. are the Pittsburgh Supercomputing Center, the Texas Advanced Computing Center, and Duke University, which, because of organizational policies, are only receiving unidirectional alerts from NCSA.

Site authentication and alert encryption. Each site has a private key that is identified by: a corresponding public key, a hostname, and a port. Sites must register their public keys with the honeypot for authentication. Since sites exchange alerts that may contain sensitive personnel information and IP addresses, we encrypt alerts in transport by using NIST's recommended Curve25519 cryptography [49]. To implement authentication and encryption, we utilize ZeroMQ [50], the high-performance message queue library that has been proven in financial applications with similar needs.

Site discovery. Sites use a simple gossip protocol for discovery and alert exchange [50]. First, a new site needs to be introduced to the network by a trusted peer. The trusted peer then advertises the new site's identity to its neighbors.

Table 2: Summary of attack attempts

Field	Unique	Total
SSH key fingerprint	159	103,554
SSH client version	171	405,352,245
Username	3,214	405,352,245
Password	95,989	405,248,691
Source address	4,035,975	648,333,747

The SSH key fingerprint is an SHA-256 hash of the SSH public key

Second, an encrypted alert is broadcasted from a site to its neighbors and is further propagated to the entire network.

Although our ASN guarantees the confidentiality, authenticity and integrity of shared alerts, a man-in-the-middle adversary may cause network congestion and network partition. Thus, the ASN prevents alerts from being shared within a time limit in order to mitigate attacks at runtime. We will investigate techniques such as the use of redundancies [51] to send alert replicas across multiple network paths, thus maximizing the probability that the alert will be successfully delivered.

5 Measurement Results

This section presents the main results from our operational experiences with CAUDIT during a 463-day period of its deployment at NCSA.

5.1 Dataset

A total of 405 million SSH attack attempts were observed in our longitudinal data collection period of 463 days (Feb. 7, 2017 to May 17, 2018). All the attacks were observed at NCSA, as summarized in Table 2.

5.2 Attack sources

The majority of the attack sources were Cloud and VPN providers and Internet service providers (ISPs) (shown in Table 3). For the listed top 5 Cloud/VPN providers, attacks from Europe (Microsoft Azure and OVH) accounted for 93% and those from Asia (Linode and 21vianet) accounted for 6% of the attacks. For ISPs, over 31% of the attacks originated in Asia. Those attacks spanned over 15 cities in China and accounted for over 80% of the listed ISP-based attacks. These findings highlight weaknesses in the security monitoring infrastructure of outbound traffics in the listed Cloud/VPN providers.

5.3 SSH clients in attacks

When a device initiates an attack, an SSH client version can be observed, per the SSH 2.0 protocol, that can be used to identify the type of device. We observed a total of 171 unique SSH clients (Table 2). Among the top six client versions observed in attacks in Table 4, 63.8% of attacks used C/C++ libraries that included `libssh2` and `sshlib`. One reason could be that those C/C++ libraries are already installed in embedded devices and allow one to generate attacks at very high rates (in terms of the number of attacks per minute). We observe that 26.4% of attacks used Python and Golang (whose SSH client version strings are `paramiko` and `Go`, respectively).

Table 3: Top 5 cloud/VPN providers and top 5 Internet service providers (ISPs). 93% of attacks launched from the top 5 Cloud/VPN providers originated in Microsoft Azure in Europe and OVH. Over 31% of attacks from the top 5 ISPs originated in Asia.

Cloud/VPN	%	ISP	%
Microsoft Azure	4.60	China Telecom	22.36
OVH	0.28	Indonesia Comnets	5.85
Linode	0.20	China Unicom	3.19
21vianet	0.12	MCI Comm	0.13
FrootVPN	0.03	Infonet Comm	0.12

Table 4: SSH-2.0 client versions with a daily count percentage greater than 50%.

Client	Version	Count	Release Year
`sshlib`	0.1	76.7M	2010
	0.5.2	1.8M	2011
`libssh2`	1.7.0	26.8M	2016
`paramiko`	2.4.0	25.1K	2017
`Go`	N/A	19.4M	–
`PUTTY`	N/A	20.4M	–

Attack device cloaking. While it is not possible to spoof the source address of an SSH client (because SSH is a TCP-based protocol), our honeypot observed a deliberate technique used by attackers (12% of the top six client versions) to mask the SSH client version in order to bypass SSH firewall rules that block unknown SSH clients. Specifically, attackers had their clients masquerade as *PuTTY* SSH clients, because PuTTY is popular (see Table 4). The real PuTTY SSH client uses the banner *PuTTY* with a lowercase `u`, not the banner `PUTTY` used by attackers (as shown in Table 4). Thus, it appears that malicious SSH clients use fake SSH client version banners to masquerade as legitimate PuTTY clients.

Age of SSH clients. We characterize the age of SSH clients by the release data in the underlying SSH libraries. Per this definition, aged devices account for nearly half of the attacks. Our honeypot observed 77.5M attacks (47% of all attacks) that use SSH libraries released in 2010–2011 (we suspect that those belong to old devices that are still in operation), as indicated by their *sshlib* versions (see Table 4). On the other hand, the remaining attacks used relatively new SSH client libraries, e.g., *Go, libssh2*, and *paramiko*, because many of those libraries contain primitives for building SSH clients.

5.4 Attacks using personalized passwords

The availability of leaked password databases [5, 52] enables attackers to run targeted attacks that use less popular, e.g., personalized passwords [53]. Our goal is to characterize the intention of attackers, i.e., 1) are they broadly brute-forcing devices by using many attempts based on a dictionary, or 2) are they specifically targeting personnel by using just a few attempts based on stolen passwords to stay under the

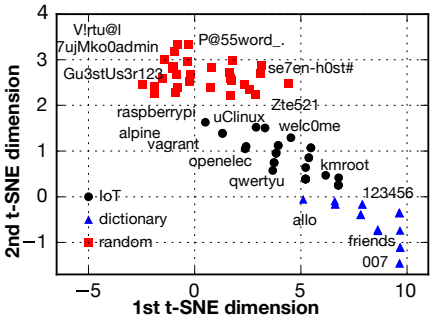

Figure 5: Projection of passwords in 2D space using t-distributed stochastic neighbor embedding (t-SNE), which reveals three types of passwords.

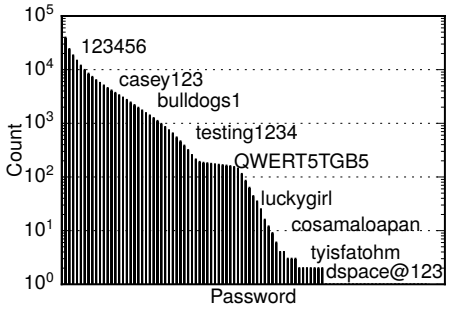

Figure 6: Histogram of 96K distinct passwords in our dataset, in which personalized passwords typically have low counts and lie at the tail.

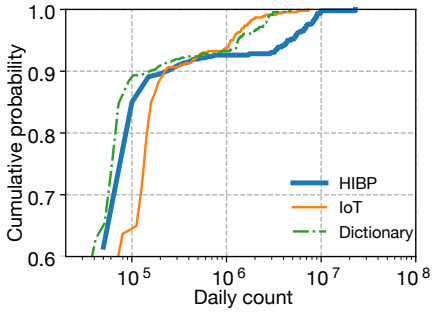

Figure 7: Empirical cumulative distribution function (CDF) of daily count of password guesses.

radar?

Visualizing features of passwords. To support the security team in recognizing the intentions of the attackers, we divide the attempted passwords into clusters. Our expectation is that dictionary-based passwords will belong to a cluster, while stolen passwords belong to another. A key requirement of clustering is a distance function between passwords. Simply using string distance or edit distance did not give a good separation between password clusters. Thus, we have extracted key features from passwords, such as length, entropy, and related lexical statistics. Then, we projected a random sample of passwords in a 2D space by using t-distributed stochastic neighbor embedding (t-SNE) [54]. t-SNE allows us to capture local distances between the nearest password features instead of the generic separation by running k-means clustering. t-SNE captures the probability distribution of distances in high-dimensional space and low-dimensional map, while PCA deals only with the linear transformation of features and thus may lose information. Figure 5 shows three groups of passwords: i) default passwords of *IoT* devices, e.g., `raspberry`; ii) common passwords in *dictionaries*, e.g., `007`; and iii) *other* passwords, e.g., `se7en-h0st#`.

At run-time, via visualization and clustering of passwords, t-SNE visualization helps the security team understand attackers' intentions better; in particular, is the attack is targeting devices, or targeting personnel accounts by using leaked passwords? Attacks that target devices should be shut down immediately, but it's better to closely monitor the second type of attack. Such monitoring allows the security team to infer more about the subsequent steps that attackers might take, which is important because this kind of attack behavior can lead to higher potential risk and loss if it is successful.

The long tail of password counts. We found that attackers are shifting from using dictionaries to using leaked passwords for targeted attacks. Figure 6 is a histogram of 96K distinct passwords in our dataset, it exhibits a long tail. Although the most common passwords, e.g., `123456`, have been being guessed for $\sim 10^5$ times, attackers are changing their strategy to guess unique and personalized passwords, explicitly targeting site-specific personnel or infrastructure based on the leaked password database. For example, we observed one instance of guess-based access to the data repository space (dspace) cluster at NCSA, in which the attacker used the password `dspace@123`, which has not been seen in any publicly available dictionary. This observation indicates a targeted attack attempt at NCSA. Furthermore, Figure 7 shows that, at 90 percentile, (from www.haveibeenpwned.com) HIBP passwords are attempted around 10× more frequent than default passwords when attacks target IoT devices. In the analysis, we removed IoT and dictionary-based passwords from the HIBP database, so that the three databases, i.e., HIBP, IoT, and the dictionary, would be mutually exclusive. Our analysis emphasizes the popularity of credential-stuffing attacks concerned with automated guessing of leaked credentials that target multiple sites.

Usernames in SSH attack attempts. Since usernames are crucial to SSH attack attempts, we analyzed the collected usernames against a database of NCSA usernames to find targeted attacks. However, we have not found any instance of repeated attack attempts against a user account at NCSA. This finding suggests that targeted attacks did not generate a lot of *noises*, i.e., attack attempts. Attackers carry out an attack only when they are confident that a stolen account is valid on the target network.

5.5 Attack attempts using SSH keys

In addition to guessing passwords, attackers have attempt to use SSH keys. Over the past few months, we observed up to 56.8K attacks that used SSH keys (see Table 5). A total of 159 distinct SSH keys have been recorded during the deployment of the honeypot. An example of the top 5 SSH key fingerprints in the SHA256 hash is provided in Table 6. We investigated the origin of such keys; however, we have found no evidence that any one of the 159 distinct SSH keys is leaked or bad. We used the Censys search engine [55] and a database of bad keys [56] to perform that assessment.

Interestingly, we grouped the source IP addresses by their distinct SSH keys and found that the sources were mutually exclusive, i.e., no two of them used the same key in their

Table 5: SSH key and password attempts from 2017/12 to 2018/04).

	17/12	18/01	18/02	18/03	18/04
Key	0.2K	7.9K	13.6K	3.8K	56.8K
Pass	984.1K	951.0K	393.8K	174.0K	99.1M

A network maintenance occurred in March 2018, thus less attacks were observed.

Table 6: Top 5 SSH key fingerprints

Key Fingerprint (SHA256)	Count
`oHhjwxYH9v+ChV4Vr...Pk6KHla6P7g443w`	20,307
`qOd/Gr8bWftEu8HDU...aNCXA3Q/0zWMCdo`	17,026
`YEYlq2GOCueBnJRoS...f7KzN5meQVVQFmA`	9,542
`+UJNIlXcTgv4BLeaZ...QH//L2cG5GRQJUE`	8,199
`oU4y6kZLH2kAdhwWU...1eBJCButjeEhIwo`	7,870

attack attempts. Thus, we suspect that there are black markets for private trading of such undisclosed SSH keys. Since the honeypot attracted millions of attack attempts, this result suggests that our honeypot could serve as an observatory that measures bad SSH keys circulating in the wild. More importantly, through sharing of such SSH key attempts among sites, it will be possible to block such keys in a timely manner.

5.6 Impact of attacks

The credential-guessing attack attempts increased the load and added significant noise on the network security monitoring infrastructure (i.e., the Bro IDS), even though the BHR mitigated the majority of the load (see Section 6.4). On a typical day, e.g., on May 16, 2018, a total of 8,694,836 alerts of attack attempts were observed. Of them, 8,361,159 (96.2%) were brute-force attempts that targeted our honeypot. The remaining alerts (3.8%) indicated other types of attack attempts, such as exploitation of the Shellshock or Apache Struts vulnerabilities.

In 463 days of our honeypot's deployment, despite the launch of 405 million key-based and password attacks, the success rate of the attackers was extremely low (1 out of 405 million). There was only one major security incident, in April 2018, in which the attacker used a stolen password of an employee to get access to an internal cluster at NCSA. Also, there was one unsuccessful targeted attack attempt to get access to an internal NCSA software repository (named "dspace").

Our honeypot deployment helped to uncover eight vulnerable hosts before there were eslations to major security incidents. Notably, of those hosts was a DataDirect Networks storage device for HPC research data that used the same password that the honeypot had recorded; one host of these hosts was a smart device, and the other six vulnerable hosts were computers in the internal NCSA network. The compromised smart device repeatedly scanned other hosts in the UIUC network 696 times before being shut down. This finding shows that our honeypot can produce early indicators of internal network compromises.

6 Evaluation

This section provides a brief history of NCSA's production network, in which CAUDIT has been gradually deployed. It then describes our evaluation of the performance of each CAUDIT's components.

6.1 Gradual deployment of CAUDIT in NCSA's production network

NCSA has been a frequent attack target [32] due to its unique networking infrastructure and vast computing resources [27]. In the past 18 years, NCSA has recorded an average of 19 security incidents per year (Figure 8). Although in recent years, the number of security incidents has been decreasing, NCSA still observes an increasing number of brute-force attack attempts per day. Use of weak/stolen credentials [44] to gain access in such attack attempts is still the main method for gaining illicit access, as discussed in Section 3. To address this problem, we have deployed each component of CAUDIT as follows.

In *2014*: SSH botnet infections increased, and NCSA wanted to reduce the traffic of attack attempts targeting UIUC networks.

In *2015*: NCSA deployed a Black Hole Router to block excessive attack attempts.

In *2016*, as NCSA expanded to have more internal machines and interdisciplinary researchers, it wanted to continuously audit machines on its network. Thus, it developed and deployed the SSH auditor tool.

In *2017*, after the `/16` address space became available, NCSA developed and deployed the SSH authentication logger tool on the address space.

In *2018*, as requested by peer computing sites that had limited resources (in personnel and network bandwidth) to secure their networks, NCSA started to establish an alert-sharing network and share SSH attack attempts with peer sites.

6.2 Overall impact of our system

Our system has contributed to annual decreases in the number of critical security incidents at NCSA, i.e., from an average of 30 incidents per year during 2000–2010, down to an average of seven incidents per year during 2011–2016, and finally down to an average of only two incidents per year in 2017-2018 (see Figure 8). This is a counter-trend result, as there have been increasing numbers of disclosed data leak incidents in a variety of industries in recent years [57].

6.3 Honeypot

The honeypot is deployed on a physical server with a 14-core Intel Xeon CPU 2.00 GHz with 128 GB memory running Red Hat Enterprise Linux (RHEL). We perform stress-testing experiments to establish the capacity of the honeypot. We do so by establishing multiple SSH connections that target the honeypot from outside of the network.

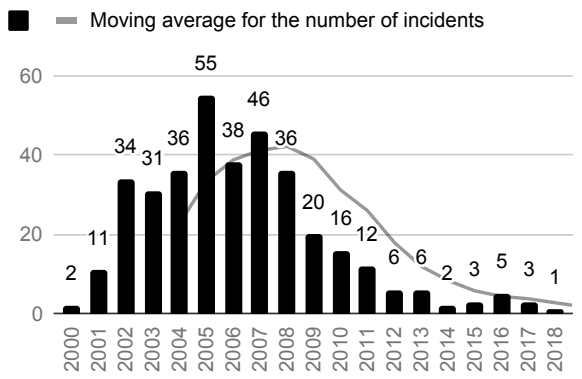

Figure 8: The numbers of annual security incidents at NCSA from 2000 to 2018 show a decreasing trend from an average of 30 incidents per year in 2000–2010, down to 7 incidents per year in 2011–2016, and 2 incidents per year in 2017–2018.

Capacity. The average load capacity of our honeypot is 50,400 ± 4,115 SSH connections per second, which is equivalent to 4.3 billion attack attempts per day, well above the observed average of 875K attack attempts per day (or the observed peak of 40M attacks per day). Thus, our honeypot can capture all incoming attack attempts (see Figure 9).

6.4 Black Hole Router

The BHR is deployed on a mixed set of Arista and Juniper network routers that can handle at most 100 Gbps and has an upper limit of one million routing entries for the BHR. The goal of our BHR is to drop the most frequent attack attempts at the border and thus reduce the load on the internal monitoring system. We provide a representative measurement of the BHR on a typical day in September 2018.

Effect of the BHR. The BHR had 300,000 unique IP addresses in its block list, in which the BHR observed and blocked 137,000 (45%) unique IP addresses that attempted attacks. Note that the BHR may block legitimate IP addresses (i.e., have false positives). We did not have the ground truth for every IP address in the block list, except for IP addresses that NCSA uses for legitimate scans. Thus, we cannot quantify the false positives. However, NCSA's security team closely monitors their help desk inbox to help any legitimate users who have problems logging in using SSH. To date, we have not observed any issue from legitimate users.

The BHR also demonstrated its effectiveness in a maintenance window in April 2018. Figure 9 shows a ∼100× increase in scanning traffic (in the 50th percentile) when the BHR did not operate.

Effect of flow shunting. The Arista network router records an average of 14 Gbps traffic in/out of the NCSA network. Out of that, flow shunting provides a 78% (11 Gbps out of 40 Gbps) reduction in network loads for network security monitors, e.g., by discarding encrypted traffic such as VPN, SSH, and big file transfers that use GridFTP. The remaining 22% of the traffic (3 Gbps) is forwarded to network security monitors for analysis.

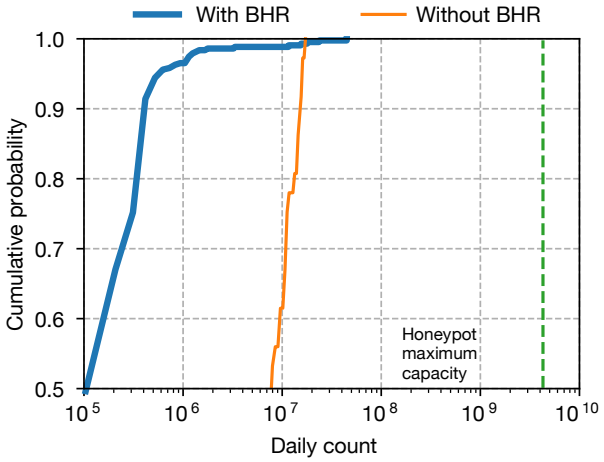

Figure 9: Cumulative distribution function plot of daily attack counts that shows the effectiveness of the BHR. At the 50th percentile, the BHR blocks ∼ 100× of the attack attempts than the case when BHR is not deployed. The attack attempt traffic is well within the capacity of our honeypot.

6.5 SSH credential auditor

The SCA performs regular audits of the NCSA network. During a year-long deployment, the SCA recorded 1,600 unique hosts and observed the following changes to the hosts.

Version changes. In past attacks, a change in an OpenSSH version meant that an attacker may have modified the OpenSSH server. However, that is not always the case.

An OpenSSH server version typically changes when an upgrade happens, e.g., `SSH-2.0-OpenSSH_7.3` is upgraded to `SSH-2.0-OpenSSH_7.4`. Interestingly, the SCA has also observed version downgrades. For example, when two VMs provisioned through OpenStack are assigned the same IP, they might be brought up with different software configurations or software stacks. Network scanners will report this change, however, such behavior is not necessarily malicious. Overall, SCA has observed 5,500 changes to OpenSSH versions.

SSH server fingerprint changes. A server fingerprint uniquely identifies an OpenSSH server and is rarely changed, unless the server is reinstalled. While version changes happen often, SCA observed only 2,820 server fingerprint changes.

Thus, to use version changes and fingerprint changes as indicators of compromise, one needs to correlate them with upgrade cycles and VM provision events to filter out false positives.

6.6 Alert-sharing network

The alert-sharing network is capable of exchanging up to 5,000 alerts per second with all subscribed peer sites. However, it is still underused. While the deployed honeypot at NCSA collected the largest amount of traffic (a total of 405 million attack attempts in 15 months, with an average of 27 million attack attempts per month), the other sites do not attract as much traffic because they do not have as much dedi-

cated network resources (CIDR /16 IP space) as NCSA. For example, at our international site at SUTD, only 2 million attack attempts have been collected in a one-month period, which is 13× lower than NCSA's number.

7 Discussions

Extending CAUDIT to other networks. The measurements and analyses in this paper were performed at an academic site (NCSA) that implements a more open networking infrastructure than corporate networks have, e.g., allowing inter-institutional access to its internal resources. Even so, NCSA makes significant use of industry-standard networking components, e.g., by extending OpenSSH to support single sign-on[4]. Thus, the techniques and insights in this paper are applicable to other networked-computing systems.

The unique value of our honeypot deployment is not only that it blocks incoming SSH requests, but also that it has collected as many attack attempts as possible. Our measurements aim at characterizing SSH attackers behaviors (e.g., the use of SSH keys and personalized passwords) in the wild. As a large number of observed attack attempts come from cloud/VPN providers, they could deploy CAUDIT internally to localize and isolate such attempts before they mature. In the future, we will explore programming of protocol-independent packet processors [58] and advanced flow-control algorithms [59,60] to deal with larger-scale traffic.

Integration with Machine Learning- based IDS. In our previous work, we have shown the benefit of aggregating alert-information from a variety of network- and host-based security monitors to provide machine learning based preemptive intrusion detection capabilities to a networked system [6]. The password clustering analysis (described in Section 5.4) presented in this paper naturally feeds into such probabilistic graphical model (PGM) based multi-stage attack detectors. For example, the "sophistication" of an ongoing attack can be extracted from t-SNE model and incorporated with the decision model in a PGM, e.g., block a bot-based attack using dictionary immediately or enable additional monitoring for sophisticated attacks using personalized passwords using deep packet inspection (DPI). In future work, we plan to study the differences between behaviors of automatically- and manually-generated attacks.

Consensus in Distributed Alert Sharing Network. Without coordinated alerts sharing among the sites, it is challenging to preemptively detect coordinated attacks across sites as illustrated in our motivating example. We plan to work with peer sites to simulate coordinated attacks, i.e., attacks that occur at the same time at multiple sites to achieve the overarching attack goals. We anticipate two main challenges in alert sharing. The first is using redundancy to ensure the timely arrival (availability [51,61,62]) of the shared alerts under a stronger threat model. For example, man-in-the-middle attackers might deliberately prevent or delay critical alerts from being shared or malicious insiders might intentionally share irrelevant alerts in mimicry attacks. The second is that of ensuring the immutability of stored and shared alerts for forensic analysis.

Adversarial Evolution and Adaptation. To address the case in which an attacker may discover our /16 IP space and avoid targeting it, we will leave the address of the /16 IP space out of our public dataset. In the future, we will not use the /16 IP space exclusively for the honeypot. Instead, we may start deploying a small number of legitimate servers, using random IP addresses, in the IP space. These legitimate computers act as canaries and allow us to assess how they perform under heavy attack related traffic.

8 Related Work

This section discusses prior work in honeypot design, security auditing, black hole router, and alert sharing networks.

Honeypots. HoneyStat has been deployed for local worm detection. However, 1) it is deployed on local networks whereas ours is deployed on NCSA's global peer-to-peer sharing infrastructure; and 2) it only carries out logit analysis for worm detection, and thus lacks a mechanism for protecting inner network infrastructure from honeypot intrusions, while NCSA applies auditing tools to preempt compromises based on honeypot intrusion data [63]. John et al. [64] deployed Web honeypots in a university network, which attracted $\sim 44,000$ attacker visits from $\sim 6,000$ distinct IPs, which inspired NCSA's honeypot deployment in a similar campus deployment environment. However, NCSA's honeypot traffic is at least 1,000 times greater. With that relatively limited attack surface, John et al.'s honeypots rely on other Web pages with high page ranks and dynamic linking of search engines to attract up-to-date or zero-day attacks. Therefore, our non-interactive honeypot is scalable: it can handle an order of magnitude more attack attempts compared to interactive honeypot such as Kippo [34], which is also more expensive to maintain and pose an unnecessary risk to our system. Such interactive honeypots allow attackers to interact with a shell: thus, they require more resources and need careful network configuration (blocking of new outgoing connections) to isolate attackers. There have also been studies in VoIP honeypots [65,66]. The main limitation of [65] is a lack of decisions in reaction to attackers, compared to the NCSA honeypot's deployment of real-time decision infrastructure based on the collected data from the honeypots. [66] has only been implemented in a preliminary stage; it has not been deployed on a large scale, and its honeypots do not maintain interactions with the rest of the security components, leading to delayed enforcement of security policies in response to real-time dynamic attacks.

Provos [21] presented a framework that simulates virtual honeypots and Vrable et al. [22] built a prototype of virtual honeyfarm system, both in opposition to a physical one, with Varable's honeyfarm system motivated NCSA's honey-

[4] www.grid.ncsa.illinois.edu/ssh

pot deployment. Provo's work was driven by the IP space limitations placed on traditional physical honeypots, and that is not an issue for the NCSA honeypot. [67] looked into the effectiveness of building deceptive honeypots within a virtual environment that uses Linux containers. However, [11] illustrated the limitations of these virtual honeypots from both attackers, and system architecture's points of view. 1) From the attackers' viewpoint, there is ease of detection without any privileges; and 2) in the underlying system architecture, there were fundamental flaws in virtualization. Other honeypots in the literature have inspired our design decisions. However, their source codes are typically not available for immediate use.

Security auditing. In [68], the testbed is embedded in the production network, while our system runs with the real production traffic that includes a mixture of attack and benign traffic in a large-scale deployment: [69] emulated a wider range of virtual topologies, not just physical hardware but only to the extent of more than a thousand virtual nodes; the NCSA honeypot scales well, such that one physical server takes all the loads of attacks from 65,536 virtualized servers. It will likely be beneficial to apply the automated network monitoring tools [70, 71] to prioritize and customize the network flows [72] of the honeypot data; and to extend techniques in [73] to formally verify authentic SSH login flows.

Routing malicious traffic. Yu et al. proposed a precise network instrumentation framework to mitigate malicious traffic, e.g., by forcing the user to change the default password of an IoT device [70] instead of redirecting the user to the null route as it is done in our approach. Wu et al. presented a packet filter [74] with low filter update latency and high-speed packet processing. 007 is an application deployed to diagnose, detect, and trace source causes for TCP connection packet drops [75]. APUNet integrated GPU in APU platforms to accelerate packet processing in network applications [76], while, on the other hand, Netmap enables rapid network packet delivery without requiring for customized hardware or modified applications [77]. Sarma et al. broadened the availability of hardware switches for network resource allocation algorithms, thus making the implementations of network protocols more flexible [78]. To achieve easier and more efficient network flow processing development in stateful middleboxes, Jamshed et al. designed and implemented a reusable network stack [79].

Network auditors or scanners. Compared to ZMap, for which analyses of new protocols were performed on random samples [23], NCSA's honeypot has adopted a more intelligent scanning methodology that subscribes to new protocols logged by IDS to incrementally discover newly added SSH servers, therefore lowering the burden of probe traffic on the production network.

Alert-sharing network. R-cisc is a cybersecurity sharing center for retail ecosystem [26]. However, unlike NCSA's sharing network, that sharing center shares security incident data among retail sites in a manner that is neither real-time nor encrypted. The publish of new threats in Facebook ThreatExchange [80] is not automated. However, it is promising to integrate NCSA's alert-sharing network with Facebook ThreatExchange and IBM X-Force to make use of the APIs for threat intelligence sharing [80, 81].

9 Conclusion

This paper presents the operational experiences with the proposed framework at the National Center for Supercomputing Applications. Our experience over 463 days shows that CAUDIT successfully blocks an average of 57 million attack attempts on a daily basis by using the proposed BHR. This represents a 66× reduction in the number of SSH attempts compared to the daily average, and has reduced 78% of the traffic to the internal network-security-monitoring infrastructure. We posit that the measurements and insights presented in this paper can be used to propose new research directions in IDS systems deployed in adversarial environments.

10 Code and Data Availability

We have open-sourced CAUDIT's implementation and its dataset at `https://pmcao.github.io/caudit`.

Acknowledgement

We thank the NCSA security team, students participated in the SDAIA project, and the partnering sites for supporting CAUDIT operational deployment; DEPEND group members, anonymous reviewers, and our shepherd, Prof. Vyas Sekar, for providing valuable feedback; and Ms. Jenny Applequist for proofreading. This material is based upon work supported by the National Science Foundation under Grant No `1535070,1547249`. The opinions, findings, and conclusions stated herein are those of the authors and do not necessarily reflect those of the sponsors.

References

[1] Theophilus Benson, Aditya Akella, and David A Maltz. Unraveling the complexity of network management. In *NSDI*, pages 335–348, 2009.

[2] Taous Madi, Suryadipta Majumdar, Yushun Wang, Yosr Jarraya, Makan Pourzandi, and Lingyu Wang. Auditing security compliance of the virtualized infrastructure in the cloud: Application to openstack. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, pages 195–206. ACM, 2016.

[3] Nicholas DeMarinis, Stefanie Tellex, Vasileios Kemerlis, George Konidaris, and Rodrigo Fonseca. Scanning the internet for ros: A view of security in robotics research. *arXiv preprint arXiv:1808.03322*, 2018.

[4] Tianlong Yu, Vyas Sekar, Srinivasan Seshan, Yuvraj Agarwal, and Chenren Xu. Handling a trillion (unfixable) flaws on a billion devices: Rethinking network security for the internet-of-things. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, page 5. ACM, 2015.

[5] Have i been pwned, 2018. https://haveibeenpwned.com/.

[6] Phuong Cao, Eric Badger, Zbigniew Kalbarczyk, Ravishankar Iyer, and Adam Slagell. Preemptive intrusion detection: Theoretical framework and real-world measurements. In *Proceedings of the 2015 Symposium and Bootcamp on the Science of Security*, page 5. ACM, 2015.

[7] You're already compromised: Exposing ssh as an attack vector, 2016. https://www.cisco.com/c/en/us/about/security-center/ssh-honeypot.html.

[8] Drawing the foul: Operation of a ddos honeypot, 2017. https://www.usenix.org/conference/enigma2017/summit-program/presentation/drawing-foul-operation-ddos-honeypot.

[9] Ping Wang, Lei Wu, Ryan Cunningham, and Cliff C Zou. Honeypot detection in advanced botnet attacks. *International Journal of Information and Computer Security*, 4(1):30–51, 2010.

[10] Payas Gupta, Bharat Srinivasan, Vijay Balasubramaniyan, and Mustaque Ahamad. Phoneypot: Data-driven understanding of telephony threats. In *NDSS*, 2015.

[11] Maximillian Dornseif, Thorsten Holz, and Und Sven Müller. Honeypots and limitations of deception. 2005.

[12] Vladimir B Oliveira, Zair Abdelouahab, Denivaldo Lopes, Mario H Santos, and Valéria P Fernandes. Honeypotlabsac: a virtual honeypot framework for android. *International Journal of Computer Networks & Communications*, 5(4):159, 2013.

[13] Todd Hoff. Netflix: Continually test by failing servers with chaos monkey, 2010.

[14] Yury Izrailevsky and Ariel Tseitlin. The netflix simian army. *The Netflix Tech Blog, July*, 2011.

[15] Catello Di Martino, Ugo Giordano, Nishok Mohanasamy, Stefano Russo, and Marina Thottan. In production performance testing of sdn control plane for telecom operators. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 642–653. IEEE, 2018.

[16] The bro network security monitor. 2018. https://bro.org.

[17] Yu-Ming Ke, Chih-Wei Chen, Hsu-Chun Hsiao, Adrian Perrig, and Vyas Sekar. Cicadas: congesting the internet with coordinated and decentralized pulsating attacks. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, pages 699–710. ACM, 2016.

[18] Mobin Javed and Vern Paxson. Detecting stealthy, distributed ssh brute-forcing. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 85–96. ACM, 2013.

[19] Ryan J. McCaughey. Deception using an ssh honeypot.

[20] D M'raihi, M Bellare, F Hoornaert, D Naccache, and O Ranen. Hotp: An hmac-based one-time password algorithm. Technical report, 2005.

[21] Niels Provos et al. A virtual honeypot framework. In *USENIX Security Symposium*, volume 173, pages 1–14, 2004.

[22] Michael Vrable, Justin Ma, Jay Chen, David Moore, Erik Vandekieft, Alex C Snoeren, Geoffrey M Voelker, and Stefan Savage. Scalability, fidelity, and containment in the potemkin virtual honeyfarm. In *ACM SIGOPS Operating Systems Review*, volume 39, pages 148–162. ACM, 2005.

[23] Zakir Durumeric, Eric Wustrow, and J Alex Halderman. Zmap: Fast internet-wide scanning and its security applications. In *USENIX Security Symposium*, volume 8, pages 47–53, 2013.

[24] Warren Kumari and Danny McPherson. Remote triggered black hole filtering with unicast reverse path forwarding (urpf). Technical report, 2009.

[25] Andreas Kuehn and Milton Mueller. Analyzing bug bounty programs: An institutional perspective on the economics of software vulnerabilities. 2014.

[26] R-cisc, 2018. https://r-cisc.org/#homeResources.

[27] Catello Di Martino, Zbigniew Kalbarczyk, Ravishankar K Iyer, Fabio Baccanico, Joseph Fullop, and William Kramer. Lessons learned from the analysis of system failures at petascale: The case of blue waters. In *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*, pages 610–621. IEEE, 2014.

[28] Nds services. 2018. https://wiki.ncsa.illinois.edu/display/NDS/NDS+Services.

[29] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the mirai botnet. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1093–1110, Vancouver, BC, 2017. USENIX Association.

[30] Fermi national accelerator laboratory, 2018. http://www.fnal.gov/.

[31] Antonio Pecchia, Aashish Sharma, Zbigniew Kalbarczyk, Domenico Cotroneo, and Ravishankar K Iyer. Identifying compromised users in shared computing infrastructures: A data-driven bayesian network approach. In *2011 30th IEEE International Symposium on Reliable Distributed Systems*, pages 127–136. IEEE, 2011.

[32] Aashish Sharma, Zbigniew Kalbarczyk, James Barlow, and Ravishankar Iyer. Analysis of security data from a large computing organization. 2011.

[33] Iforge cluster, 2018. http://www.ncsa.illinois.edu/industry/iforge.

[34] Kippo - ssh honeypot, 2016. https://github.com/desaster/kippo.

[35] Cláudia J Barenco Abbas, L Javier García Villalba, and Victoria López López. Implementation and attacks analysis of a honeypot. In *International Conference on Computational Science and Its Applications*, pages 489–502. Springer, 2007.

[36] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz, and Nick McKeown. Reproducible network experiments using container-based emulation. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 253–264. ACM, 2012.

[37] Phuong Cao, Eric C Badger, Zbigniew T Kalbarczyk, and Ravishankar K Iyer. A framework for generation, replay, and analysis of real-world attack variants. In *Proceedings of the Symposium and Bootcamp on the Science of Security*, pages 28–37. ACM, 2016.

[38] Cuong Pham, Zachary J Estrada, Phuong Cao, Zbigniew Kalbarczyk, and Ravishankar K Iyer. Building reliable and secure virtual machines using architectural invariants. *IEEE Security & Privacy*, 12(5):82–85, 2014.

[39] Cuong Pham, Zachary Estrada, Phuong Cao, Zbigniew Kalbarczyk, and Ravishankar K Iyer. Reliability and security monitoring of virtual machines using hardware architectural invariants. In *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*, pages 13–24. IEEE, 2014.

[40] Robert P Goldberg. Architecture of virtual machines. In *Proceedings of the workshop on virtual computer systems*, pages 74–112. ACM, 1973.

[41] man page for passwd, 2018. https://www.unix.com/man-page/linux/1/passwd/.

[42] Ssh server auditing, 2018. https://github.com/arthepsy/ssh-audit.

[43] Gordon Fyodor Lyon. *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure, 2009.

[44] Aashish Sharma, Zbigniew Kalbarczyk, R Iyer, and James Barlow. Analysis of credential stealing attacks in an open networked environment. In *Network and System Security (NSS), 2010 4th International Conference on*, pages 144–151. IEEE, 2010.

[45] fail2ban, 2018. https://www.fail2ban.org/wiki/index.php/Main_Page.

[46] Liang Zhang, David Choffnes, Dave Levin, Tudor Dumitras, Alan Mislove, Aaron Schulman, and Christo Wilson. Analysis of ssl certificate reissues and revocations in the wake of heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 489–502. ACM, 2014.

[47] Bpf and xdp reference guide, 2018. https://cilium.readthedocs.io/en/latest/bpf/.

[48] Gianluca Insolvibile. Kernel korner: Inside the linux packet filter. *Linux journal*, 2002(94):7, 2002.

[49] Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography, 2018. https://csrc.nist.gov/CSRC/media/Publications/sp/800-56a/rev-3/draft/documents/sp800-56ar3-draft.pdf.

[50] Pieter Hintjens. *ZeroMQ: messaging for many applications*. " O'Reilly Media, Inc.", 2013.

[51] Sushant Jain, Michael Demmer, Rabin Patra, and Kevin Fall. Using redundancy to cope with failures in a delay tolerant network. In *ACM SIGCOMM Computer Communication Review*, volume 35, pages 109–120. ACM, 2005.

[52] Joe DeBlasio, Stefan Savage, Geoffrey M Voelker, and Alex C Snoeren. Tripwire: Inferring internet site compromise. In *Proceedings of the 2017 Internet Measurement Conference*, pages 341–354. ACM, 2017.

[53] Phuong Cao, Hongyang Li, Klara Nahrstedt, Zbigniew Kalbarczyk, Ravishankar Iyer, and Adam J Slagell. Personalized password guessing: a new security threat. In *Proceedings of the 2014 Symposium and Bootcamp on the Science of Security*, page 22. ACM, 2014.

[54] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[55] Censys, 2018. https://censys.io/.

[56] Ssh bad keys, 2017. https://github.com/rapid7/ssh-badkeys.

[57] Verizon RISK Team and R Team. 2018 data breach investigations report. 2018.

[58] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[59] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 101–114. ACM, 2016.

[60] Seyed Kaveh Fayaz, Yoshiaki Tobioka, Vyas Sekar, and Michael Bailey. Bohatei: Flexible and elastic ddos defense.

[61] Costin Raiciu, Christoph Paasch, Sebastien Barre, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark Handley. How hard can it be? designing and implementing a deployable multipath tcp. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 29–29. USENIX Association, 2012.

[62] Cuong Pham, Phuong Cao, Zbigniew Kalbarczyk, and Ravishankar K Iyer. Toward a high availability cloud: Techniques and challenges. In *Dependable Systems and Networks Workshops (DSN-W), 2012 IEEE/IFIP 42nd International Conference on*, pages 1–6. IEEE, 2012.

[63] David Dagon, Xinzhou Qin, Guofei Gu, Wenke Lee, Julian Grizzard, John Levine, and Henry Owen. Honeystat: Local worm detection using honeypots. In *International Workshop on Recent Advances in Intrusion Detection*, pages 39–58. Springer, 2004.

[64] John P John, Fang Yu, Yinglian Xie, Arvind Krishnamurthy, and Martín Abadi. Heat-seeking honeypots: design and experience. In *Proceedings of the 20th international conference on World wide web*, pages 207–216. ACM, 2011.

[65] Mohamed Nassar, Radu State, and Olivier Festor. Voip honeypot architecture. In *2007 10th IFIP/IEEE International Symposium on Integrated Network Management*, pages 109–118. IEEE, 2007.

[66] Rodrigo Do Carmo, Mohamed Nassar, and Olivier Festor. Artemisa: An open-source honeypot back-end to support security in voip domains. In *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, pages 361–368. IEEE, 2011.

[67] Alexander Kedrowitsch, Danfeng Daphne Yao, Gang Wang, and Kirk Cameron. A first look: Using linux containers for deceptive honeypots. In *Proceedings of the 2017 Workshop on Automated Decision Making for Active Cyber Defense*, pages 15–22. ACM, 2017.

[68] Rob Sherwood, Glen Gibb, Kok-Kiong Yap, Guido Appenzeller, Martin Casado, Nick McKeown, and Guru M Parulkar. Can the production network be the testbed? In *OSDI*, volume 10, pages 1–6, 2010.

[69] M Hibler R Ricci L Stoller, Jonathon Duerig, Shashi Guruprasad, Tim Stack, Kirk Webb, and Jay Lepreau. Large-scale virtualization in the emulab network testbed. In *USENIX Annual Technical Conference, Boston, MA*, 2008.

[70] Tianlong Yu, Seyed Kaveh Fayaz, Michael P Collins, Vyas Sekar, and Srinivasan Seshan. Psi: Precise security instrumentation for enterprise networks. 2017.

[71] Michael Golightly and Jack Brassil. Automating network monitoring on experimental testbeds. In *CSET*, 2011.

[72] Kimberly C. Claffy, H-W Braun, and George C. Polyzos. A parameterizable methodology for internet traffic flow profiling. *IEEE Journal on selected areas in communications*, 13(8):1481–1494, 1995.

[73] Shuo Chen, Matt McCutchen, Phuong Cao, Shaz Qadeer, and Ravishankar K Iyer. Svauth–a single-sign-on integration solution with runtime verification. In *International Conference on Runtime Verification*, pages 349–358. Springer, 2017.

[74] Zhenyu Wu, Mengjun Xie, and Haining Wang. Swift: A fast dynamic packet filter. In *NSDI*, volume 8, pages 279–292, 2008.

[75] Behnaz Arzani, Selim Ciraci, Luiz Chamon, Yibo Zhu, Hingqiang Liu, Jitu Padhye, Boon Thau Loo, and Geoff Outhred. 007: Democratically finding the cause of packet drops. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*. USENIX Association, 2018.

[76] Younghwan Go, Muhammad Asim Jamshed, YoungGyoun Moon, Changho Hwang, and KyoungSoo Park. Apunet: Revitalizing gpu as packet processing accelerator. In *NSDI*, pages 83–96, 2017.

[77] Luigi Rizzo. Netmap: a novel framework for fast packet i/o. In *21st USENIX Security Symposium (USENIX Security 12)*, pages 101–112, 2012.

[78] Naveen Kr Sharma, Antoine Kaufmann, Thomas E Anderson, Arvind Krishnamurthy, Jacob Nelson, and Simon Peter. Evaluating the power of flexible packet processing for network resource allocation. In *NSDI*, pages 67–82, 2017.

[79] Muhammad Asim Jamshed, YoungGyoun Moon, Donghwi Kim, Dongsu Han, and KyoungSoo Park. mos: A reusable networking stack for flow monitoring middleboxes. In *NSDI*, pages 113–129, 2017.

[80] Getting started with threatexchange, 2018. https://developers.facebook.com/docs/threat-exchange/getting-started/v3.1.

[81] Ibm x-force, 2018. https://www.ibm.com/security/xforce.