# Hopper: Modeling and Detecting Lateral Movement

*Grant Ho*[⋆†∘] *Mayank Dhiman*[∘] *Devdatta Akhawe*[Ψ]
*Vern Paxson*[†∮] *Stefan Savage*[⋆] *Geoffrey M. Voelker*[⋆] *David Wagner*[†]

[∘]*Dropbox* [†]*UC Berkeley* [⋆]*UC San Diego* [Ψ]*Figma, Inc.* [∮]*International Computer Science Institute*

## Abstract

In successful enterprise attacks, adversaries often need to gain access to additional machines beyond their initial point of compromise, a set of internal movements known as *lateral movement*. We present Hopper, a system for detecting lateral movement based on commonly available enterprise logs. Hopper constructs a graph of login activity among internal machines and then identifies suspicious sequences of logins that correspond to lateral movement. To understand the larger context of each login, Hopper employs an inference algorithm to identify the broader path(s) of movement that each login belongs to and the *causal* user responsible for performing a path's logins. Hopper then leverages this path inference algorithm, in conjunction with a set of detection rules and a new anomaly scoring algorithm, to surface the login paths most likely to reflect lateral movement. On a 15-month enterprise dataset consisting of over 780 million internal logins, Hopper achieves a 94.5% detection rate across over 300 realistic attack scenarios, including one red team attack, while generating an average of $< 9$ alerts per day. In contrast, to detect the same number of attacks, prior state-of-the-art systems would need to generate nearly $8\times$ as many false positives.

## 1 Introduction

Organizations routinely fall victim to sophisticated attacks, resulting in billions of dollars in financial harm, the theft of sensitive data, and the disruption of critical infrastructure [11, 15, 33, 37, 41]. In many of these attacks, adversaries need to move beyond their initial point of compromise to achieve their goal [28, 33, 48]. For example, an employee compromised by a spearphishing attack often does not have all of an organization's sensitive secrets readily accessible from their machine; thus, attackers will need to move to other machines to access their desired data. This set of malicious *internal* movements is known as *lateral movement* [8, 47].

In this work, we focus on detecting lateral movement in enterprise networks. We present Hopper, a system that uses commonly-collected log data to detect lateral movement attacks with a manageable rate of false alarms. Hopper builds a graph of user movement (logins) between internal machines and then identifies suspicious movement paths within this graph. While prior work has proposed similar graphical models, these approaches have either relied on narrowly crafted signatures [30], leaving them unable to detect many lateral movement attacks, or applied standard anomaly detection methods that alert on rare login paths [27, 29, 44]. Unfortunately, the scale of modern enterprises inherently produces large numbers of anomalous-but-benign logins, causing traditional anomaly detection to generate too many false alarms.

Hopper overcomes these challenges by employing a different approach, which we call specification-based anomaly detection. Our approach leverages an attack specification that captures fundamental characteristics of lateral movement as a set of key path properties (§ 4). This specification states that successful lateral movement attacks will (1) switch to a new set of credentials and (2) eventually access a server that the original actor could not access. We then combine this specification with anomaly detection, to reduce false positives and imprecision due to the limitations of real-world data.

Our attack specification capitalizes on a key observation: adversaries generally perform lateral movement to access a machine that their initial victim lacked access to. Thus, as part of their lateral movement activity, attackers will need to acquire and switch to a new set of credentials that enables their sought-for access. As a result, lateral movement paths will exhibit the two key attack properties identified in our specification. In the context of an attack's full lifecycle, our specification observes that standard authentication logs not only provide a window into lateral movement activity, but also contain implicit artifacts of other key stages in an enterprise attack. For example, attackers use a variety of techniques to acquire privileged credentials (as detailed in the *Credential Access* and *Privilege Escalation* stages of the MITRE ATT&CK Framework [46]). While prior work detects these other attack stages through intricate host-activity analysis [18, 24, 32], the fruits of these malicious actions manifest themselves during lateral movement, since attackers use these new credentials to access new data and machines. Through the detection methods that we develop, Hopper infers and leverages such signals (reflected in our two key attack properties) to help uncover lateral movement activity.

To identify paths with the two key properties, we develop methods for reconstructing a user's global movement activity from the point-wise login events reported in common authentication logs. These methods allow Hopper to infer the *causal* user responsible for performing each login and the broader path of movement a login belongs to (§ 5). Unfortunately, real-world authentication logs do not always contain sufficient information for Hopper to clearly identify the causal user who made each login, resulting in uncertainty about whether some paths truly exhibit the two key attack properties. To resolve these cases of uncertainty, Hopper employs a new anomaly detection algorithm to identify the most suspicious paths to alert on (§ 6). This selective approach to anomaly detection is a key distinction that allows Hopper to significantly outperform prior work that relies on traditional anomaly detection [44] or signature-based detection [30].

We evaluate Hopper on a 15-month enterprise data set that contains over 780 million internal login events (§ 7). This data includes one lateral movement attack performed by a professional red team and 326 simulated attacks that span a diverse array of real-world scenarios (ranging from ransomware to stealthy, targeted machine compromise). On this data set, Hopper can detect 309 / 327 attacks while generating $< 9$ false positives per day on average, which is an $8\times$ improvement over prior state-of-the-art systems [44].

In summary, we make the following contributions:

- We present Hopper, a novel system that uses commonly-collected authentication logs to detect lateral movement. Hopper employs a new detection approach based on a principled set of properties that successful lateral movement paths will exhibit (§ 4).
- Our approach identifies paths with these key properties by inferring the broader paths of movement that users make (§ 5), and strategically applies a new anomaly scoring algorithm to handle uncertainty that arises due to the limited information in real-world logs (§ 6).
- We evaluate Hopper on 15 months of enterprise data, including a red team attack and over 300 realistic attack simulations. Hopper detects 94.5% of these attacks, and produces $8\times$ fewer false alarms than prior work (§ 7).

## 2 Background

The internal movements that attackers make between machines within an enterprise is known as *lateral movement* (Figure 1). In this section, we review prior work on defending against lateral movement and describe the goals and assumptions that underlie our detection approach.

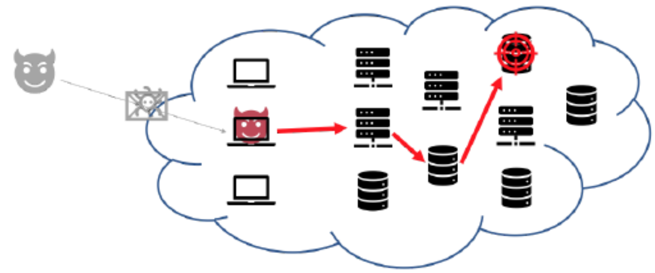

**Figure 1:** Lateral movement, depicted as red arrows, is the set of attacker movements between *internal* machines in an enterprise.

### 2.1 Related Work

Prior work pursues three general strategies for mitigating lateral movement: improving security policies to limit attacker movement; detecting lateral movement activity; and developing forensic techniques to help remediate a known attack. We consider the first and last lines of work as complementary directions to our work; we focus on developing practical detection for lateral movement attacks. The first direction, proactively improving security policies, enables an organization to implement better least privilege policies and identify high-risk machines that warrant additional monitoring [10, 12, 16, 42]. While beneficial, these policies do not fully eliminate all possible lateral movement paths; indeed, our work aims to detect attacks that can succeed even at organizations with good least privilege hygiene. The third line of related work, investigating a known attack, assumes that an organization has already identified the existence of a breach. Enterprises can use these prior methods to effectively analyze and remediate a lateral movement attack identified by Hopper.

Prior work on detecting lateral movement frequently models internal logins as a graph of machine-to-machine movement [2, 4, 27, 29, 30, 40, 44, 50], an idea that we draw upon. However, unlike our work, prior systems detect lateral movement by applying narrow signatures or traditional machine learning techniques to flag anomalous activity. Kent et al. [27] detect the use of compromised credentials by training a logistic regression model to detect when an account accesses an unusual set of machines; their classifier achieves a true positive rate of 28% and incorrectly flags 1 / 800 users as compromised. Bowman et al. [4] and log2vec [29] use deep-learning methods to build anomaly detection systems, with hand-tuned thresholds, that identify clusters of suspicious logins. These approaches incur false positive rates ranging from 0.9% [4] to 10% [29] to detect 80–90% of simulated attacks and/or red team exercises in their data.

Among the best performing prior work, Siadati and Memon propose a detector for identifying "structurally anomalous logins", which we refer to as SAL [44]. On one month of data, SAL can detect 82% of randomly generated attack logins at a 0.3% false positive rate ($>$ 500 false alarms/day on their dataset). Whereas SAL focuses on identifying point-wise anomalous logins ("one-hop" paths), Latte [30] detects two-hop lateral movement attacks by identifying paths where each login has rarely occurred in prior history. Latte then uses a specific signature to reduce false positives by only alerting on rare paths that also include a remote file execution operation on the path's final machine (identified by a set of hard-coded Windows events). Based on one day of data and a specific anomaly threshold, Latte can detect a pentester exercise while generating 13 false alarms. Although Latte can identify longer attack paths, its narrow signature, which requires the attacker to perform a specific action on the final host, can lead to false negatives. Moreover, implementing this signature faces practical challenges, since common authentication logs from Linux and Mac OS systems do not provide an easy way to re-implement Latte's Windows-specific signature.

Although they provide good starting points for detection, prior systems generate an impractical volume of false positives or incur too many false negatives (Section 7.4 reports the performance of SAL on our data set). Our work addresses these challenges with a new approach to identifying suspicious login paths. Rather than alerting on paths that are simply anomalous or relying on signatures that target specific host operations, we identify a set of key properties about attack paths based on the overarching goals of lateral movement. By focusing on paths with these properties, and only applying anomaly detection in scenarios with high uncertainty, our approach detects a wider range of attacks than those that employ a narrow signature, while also generating fewer false positives than traditional anomaly detection methods.

## 2.2 Security Model

**Detection Goals:** Hopper aims to (1) detect a diverse range of lateral movement attacks, while (2) generating a very low volume of false positives. We focus on developing detection for settings where an organization has a team of security analysts with a limited time budget for reviewing alerts. In particular, we design Hopper to score a set of movement paths in terms of how problematic the activity appears to be, allowing an organization to specify their own bound on the number of alerts that Hopper generates. Based on prior work [3,23] and the practical experiences of our industry collaborators, this alert-budget design accurately reflects a real-world operating model for many organizations. We consider Hopper successful if it produces an alert for any login made by an attacker. Upon confirming the presence of an attack, organizations can use forensic techniques from complementary work [19,25,50] to perform further analysis and remediation.

**Threat Model:** Similar to prior work, we focus on detecting interactive and credential-based lateral movement attacks [44]. Under this threat model, we assume that an attacker has managed to compromise an initial "foothold" machine within the enterprise, but they (1) need to acquire additional credentials to access the data or systems they ultimately seek, and (2) move between machines via login or remote command execution events that use a set of credentials for authentication. In particular, attackers may exploit vulnerabilities on machines or weak authentication protocols (e.g., privilege escalation or pass-the-hash attacks), but we assume that their movement between machines produces a login event visible to our detector. Additionally, this threat model focuses on attackers who manually perform the movement (login) operations during their attack, as opposed to an attack that installs malware that moves to new systems autonomously. Our threat model reflects the behavior of many real-world lateral movement attacks, ranging from targeted attacks by state-sponsored actors [5,20,31,34,36,39,45] to newer and stealthier forms of ransomware [13,48].

| Nodes (Source + Destination Machines) | Edge (Login) |
|---|---|
| Hostname | Timestamp |
| Client vs. server | Target username |
| Owner's username (clients only) | |

**Table 1:** The information for each login event in our data. Each login creates a unique edge between two nodes (internal machines) in the graph that Hopper constructs (§ 4.2).

# 3 Data

Our work uses a collection of successful login events between internal machines by employees at Dropbox,[1] a large enterprise that provides storage and cloud collaboration services to hundreds of millions of users. Whenever a machine receives a remote access attempt from another machine (e.g., an inbound ssh session or a remote command execution issued via utilities like psexec), the receiving machine generates a record of a remote "login". Because most operating systems record these login events by default, organizations collect these authentication logs as part of standard security best practices.

This data provides visibility into the internal logins between machines within Dropbox's corporate network, such as client laptops, authentication servers (e.g., Windows Domain Controller), and a variety of infrastructure and application servers (e.g., DNS servers, machines that test and build applications, and analytics servers). Representative of the heterogeneous nature of modern enterprises, the logins in our data span a variety of authentication protocols (e.g., Kerberos and ssh) across many types of devices (laptops, physical servers, and virtual machines), operating systems (Windows, Mac OS, and Linux), and account types (e.g., regular users, administrators, and service accounts).

[1] Because our work focuses on mitigating successful lateral movement, our analysis omits failed logins; however, future work could investigate ways to incorporate such failures as additional detection signals.

### 3.1 Data Size and Schema

Our data contains 784,459,506 successful logins from Jan 1, 2019 to Apr 1, 2020 (15 months). As shown in Table 1, each login event contains a timestamp, the target username of the login, the source and destination machines that initiate and receive the login, respectively, and metadata about these machines. These logins span 634 accounts and occur between 2,327 machines. Section 8.2 provides more details about the graph topology of our login data, and how different network configurations might affect our detection algorithms.

### 3.2 Data Cleaning

The vast majority of our data's login events do not reflect meaningful remote access events (i.e., did not enable a user to remotely execute commands or access sensitive data on the destination machine). Hopper applies four filtering rules described below to remove these logins from our data set. Excluding these spurious logins, our data set contains 3,527,844 successful logins, with a median of 4,098 logins per day.

**Filtering Windows logins:** As noted in prior work [27], many "logins" between internal machines in Windows enterprise environments do not represent a meaningful remote access event. Rather, these logins often correspond to uninteresting artifacts and special API calls that result from Windows enterprise logging, and do not provide a user with the ability to access data or alter the destination machine. Removing these logins from our data results in a 40× reduction, which comes primarily from removing three types of logins: printing jobs, authentications into update and logging servers, and non-administrator logins to Windows Domain Controllers. Most non-administrator logins to Domain Controllers correspond to artifacts of Kerberos authentication, where Domain Controllers serve the role of a Kerberos Key Distribution Center (KDC) and requests for a Kerberos ticket generate a record of a "login" into the Domain Controller. After removing this collection of spurious logins, our data set contains roughly 19.5 million login events.

**Filtering automation logins:** We further winnow our data set by removing internal logins that result from low-risk automation. Hopper analyzes a historical set of logins and identifies a set of login edges that correspond to automation. Specifically, each automation edge consists of a triplet (source, destination, and username), that (1) occurs frequently across our data,[2] (2) occurs on at least 50% of the historical days, and (3) has a target username that does *not* match any employee's account (i.e., a non-human username). Hopper then outputs a list of these edges as candidates for automation related logins. After a review by the organization's security team, Hopper removes any login whose (source, destination, and target user) matches an edge listed in the approved automation set.

In our data, Hopper identifies a set of approximately 30 automation edges that account for over 16 million login events. Manually inspecting these automation logins reveals that they correspond to mundane operations with minimally privileged service accounts via a restricted set of remote-API calls (e.g., specific remctl calls [1] exposed by the destination machines). For example, many of these logins resulted from file synchronization operations between a central "leader" node and geographic replicas (e.g., a central software repository machine syncing its content with replicated, regional servers). Another common category of these automation logins corresponds to version control and bug tracking software performing git operations to synchronize state among each other; these internal logins occurred under a restricted "git" user account that has access to a limited API of git operations.

### 3.3 Ethics

This work involved a collaboration between academia and industry. Our research used an existing, historical data set of employee logins between internal machines at Dropbox, which enterprises commonly collect to secure their environment. Only authorized security employees at Dropbox accessed this data; no sensitive data or personally identifying information was shared outside of Dropbox. Additionally, the machines that store and operate directly on data from Dropbox's customers reside on separate infrastructure; our study did not involve that infrastructure or access any customer-related data. This project underwent internal review and received approval by the legal, privacy, and security teams at Dropbox.

## 4 Modeling Lateral Movement

**Our Approach:** Hopper, our system, constructs a graph of user logins between internal machines and then detects lateral movement by identifying suspicious paths in this graph. A suspicious path corresponds to a sequence of logins made by a single actor with two properties: (1) the path has at least one login where the actor uses a set of credentials that does not match their own, (2) the path accesses at least one machine that the actor does not have access to under their own credentials.

**Motivating Intuition:** This approach leverages a simple yet powerful observation: in many real-world enterprise attacks, adversaries conduct lateral movement to acquire additional credentials and access new machines that their initial foothold did not have access to [9,20,31,34,36,39,45]. For example, at many organizations, access to sensitive data and/or powerful internal capabilities requires a special set of privileges, which most enterprise users lack. Thus, attacker lateral movement will produce paths that use a new (elevated) set of credentials

[2] In our work, we define a frequently occurring edge as one that occurs greater than $N = 24 \times D$ times, where $D$ equals the number of days in the historical data set (i.e., in total, the edge occurs at least as often as a process that runs once every hour on each day in the historical data set).

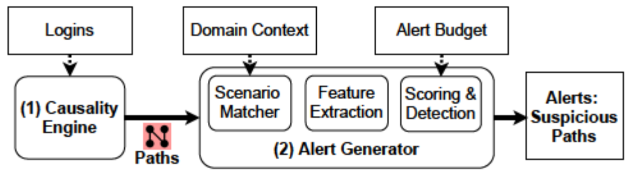


**Figure 2:** Hopper analyzes login events between internal machines within an enterprise and generates alerts for paths of logins that correspond to suspicious lateral movement activity. Hopper has two key components: (1) a causality engine that infers a set of causal paths that a login might belong to (§ 5), and (2) detection and scoring algorithms that decide whether to alert on a path of logins (§ 6).

(Property 1) and access sensitive machines that their initial victim could not access (Property 2). By searching for these two key properties, Hopper also illustrates how login data not only provides visibility into attacker lateral movement, but also contains latent signals that reveal the completion of other core stages of an attack's lifecycle. For example, Property 1 captures the fact that attackers frequently acquire privileged credentials (the "privilege escalation" and "credential access" stages from the MITRE ATT&CK Framework [46]) to access additional machines within an organization.

Moreover, the combination of these two attack path properties corresponds to characteristics that we do not expect in benign paths: users should access machines under their own credentials and they should only login to machines that they have legitimate privileges to access.

### 4.1 Challenge: Anomalies at Scale

Prior work detects lateral movement by identifying logins that traverse rare graph edges, under the assumption that attacker movement will occur between users and machines that rarely interact with each other [2, 30, 44]. While intuitive, these approaches generate too many false positives, due to the volume of rare-but-benign behavior that occurs in large enterprises.

Even after applying Hopper's data cleaning steps (§ 3.1), tens of thousands of logins create "rare" graph edges in our data set. If we alerted on logins whose edges have never occurred in recent history, such a detector would produce over 24,000 alerts across our data (over 1,600 alerts / month). These rare-but-benign logins stem from a diverse set of causes, such as users performing maintenance on machines they rarely access (e.g., a user serving on their team's on-call rotation), new users or employees returning from a long vacation, and users simply accessing rare-for-their-role services. Although prior work introduces techniques to refine this anomaly detection approach, they still produce too many false positives (§ 7.4). By re-framing the definition of an attack path from simply anomalous paths, to paths that contain the key properties we highlight, Hopper can detect a range of lateral movement attacks with significantly fewer false positives.

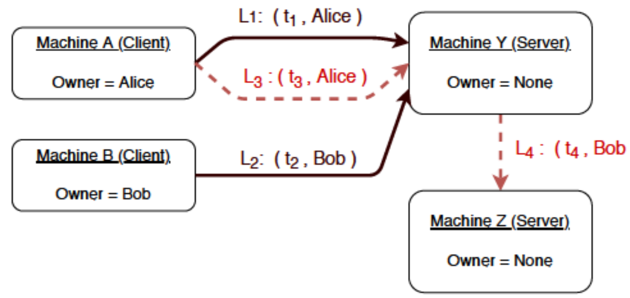


**Figure 3:** An example of a simple login graph. Solid black edges ($L_1$ and $L_2$) correspond to benign login events. Dashed red edges ($L_3$ and $L_4$) correspond to a lateral movement attack path.

### 4.2 Hopper: System Overview

Hopper consists of two stages, shown in Figure 2. The first stage of Hopper (§ 5) runs a "causality engine" that aggregates a set of logins into a graph of user movement and identifies broader paths of movement formed by groups of logically-related logins. The second stage of Hopper (§ 6) takes a set of login paths and decides whether to generate an alert by identifying which login paths contain the two key attack properties described above. During this final stage, Hopper prunes common benign movement paths, extracts a set of features for each path, and uses a combination of detection rules and a new anomaly scoring algorithm to compute the "suspiciousness" of each login path.

**The Login Graph:** Given a set of logins, Hopper constructs a directed multi-graph that captures the interactions among users and internal machines. Figure 3 shows a simple example of a login graph constructed by Hopper. Each login creates a directed edge in the graph, where the edge's source and destination nodes correspond to the machine initiating and receiving the login. Edges represent unique, timestamped logins from the source to the destination machine; multiple logins between the same two machines generate multiple edges. Each edge is annotated with a target username: the account that was logged into on the destination machine (the username and permissions that the new session operates under).

**Login Paths and Causal Users:** A path of logins corresponds to a series of connected edges, where each edge is "caused" by the same actor. We use the term *causal user* to refer to the actor whose machine initiated a path of logins, which might not be the same as the *target user* recorded in each login. The causal user is the original actor responsible for making these logins (taken from the first edge in each path), while each login's target user reflects the credentials that the login's destination machine received.

For example, in Figure 3, an attacker compromises Alice's machine ($A$) and makes a series of internal logins that forms a two-hop lateral movement path from Machine $A$ to $Z$. The attacker first uses Alice's credentials in a login to Machine $Y$, shown as $L_3$. Then the attacker compromises Bob's credentials on $Y$ and uses them to login to Bob's account on $Z$, labeled $L_4$. For each of the logins in this path, Alice is the causal user, since all of the logins were made (caused) by a user starting from Alice's machine. Alice and Bob are the target users of $L_3$ and $L_4$ respectively, since each login presented those usernames and credentials during authentication.

**Path Types:** One of the key attack properties that Hopper looks for is whether a path's causal user ever authenticates into a machine with a new set of credentials. As described later in Section 5, the information provided in standard authentication logs does not always enable Hopper to precisely infer whether a path exhibits this property. Accordingly, Hopper makes a distinction between three types of paths: a BENIGN path, a path with a CLEAR credential switch, or an UNCLEAR path.

Hopper labels a path as BENIGN if every login in the path uses the causal user's credentials (e.g., no switch in credentials occurred). A path has a CLEAR credential switch if at least one login in the path must have switched to a new set of credentials. For example, in Figure 3, assume that login $L_2$ did not occur at all, then the paths $(L_1, L_4)$ and $(L_3, L_4)$ correspond to paths with a CLEAR switch, because all paths leading to $L_4$ previously used a different set of credentials. On the other hand, if all of $L_1, L_2, L_3$ occurred and Hopper cannot clearly determine which of them caused $L_4$, then Hopper will treat both the paths $(L_1, L_4)$ and $(L_3, L_4)$ as UNCLEAR paths. An UNCLEAR path corresponds to a situation where Hopper cannot cleanly infer a causal path for a given login, but rather infers multiple potential paths, where some of the paths involve a switch in credentials (e.g., $L_3$ to $L_4$), but others do not (e.g., $L_2$ to $L_4$). As discussed in Section 6, because of these different levels of certainty, Hopper uses two sets of detection algorithms to classify a path as malicious. For paths with a CLEAR credential switch, Hopper applies a simple rule-set (§ 6.1). However, when limitations in real-world logs create uncertainty about the paths that Hopper's causality engine infers (i.e., UNCLEAR paths), Hopper uses an anomaly scoring algorithm to determine when to alert on a path (§ 6.2).

## 5 Inferring Causal Login Paths

Standard authentication logs describe point-wise activity that lacks broader context about each login, such as from whom and where the login originated. For example, in Figure 3, given login $L_4$ in isolation, a detector does not know whether *Bob* accurately reflects the user responsible for making the login, or whether another user such as Alice has stolen Bob's credentials and used them in a malicious login. Thus, for each login ($L_i$) that occurs, the first stage of Hopper runs a "causality engine" that coarsely infers the broader path of movement that a login belongs to and the *causal user* responsible for initiating the movement path. To do so, Hopper uses a time-based heuristic to infer a set of "causal paths" for $L_i$, where each path corresponds to a unique sequence of connected logins that could have led to $L_i$ and occurred within the maximum time limit for a remote login session.

| Path Component | Description |
|---|---|
| Login List | List of logins in the path |
| Causal User | Username of the employee whose machine initiated the path |
| Changepoint Logins | A list of logins where the username differs from the path's preceding login |
| Path Type | BENIGN, CLEAR, or UNCLEAR: whether the path switches to new credentials |

**Table 2:** Information in each path generated by Hopper's causality engine (§ 5). Given a new login, Hopper infers a set of these causal paths, each of which reflects a sequence of logins that an actor could have made up to and including the new login.

**Identifying Causally-Related Logins:** Hopper produces a set of causal paths by running a backwards-tracing search from $L_i$ to identify a sequence of causally-related logins that include $L_i$. Two logins are causally related if they (1) form a connected set of edges in the login graph and (2) occur within $T$ hours of each other. Concretely, we say that $L_k$ is a causal, inbound login for $L_i$ if the destination of $L_k$ equals the source machine of $L_i$, and $L_k$ occurred within 24 hours prior to the time of $L_i$. We choose a threshold of 24 hours based on the maximum duration of a login session at Dropbox; for sessions that exceed this duration, the company requires the source machine to re-authenticate, which produces a fresh login event in our data. For example, in Figure 3, $L_1$, $L_2$, and $L_3$ are all causal logins for $L_4$ if they occurred within 24 hours prior to $t_4$. Using this causal rule, Hopper infers a set of login paths by identifying all of the causal logins for $L_i$, and then recursively repeats this search on each of those causal logins.

This process is similar to provenance and taint-tracking methods that trace the flow of information from a sink ($L_i$'s destination machine) back to its source (the root node of $L_i$'s login path) [18, 24, 25]. As with these flow-tracking methods, naive backwards-tracing risks a "dependency explosion", where each backwards step can exponentially increase the number of paths that Hopper infers, but only one of these paths represents $L_i$'s true causal path. We find that four optimizations and environmental factors mitigate this risk.

First, Hopper can use an optimized implementation that requires only a single-step of backwards-tracing per login. At a high-level, based on our key attack properties, Hopper only needs to analyze paths that involve a switch in credentials (Property 1). As a result, Hopper can incrementally build a set of "watchlist" paths that contain a potential switch in credentials. For each new login, Hopper only needs to perform one step of backwards-tracing to determine if the new login involves a switch in credentials, or if it extends one of these watchlist paths; Appendix A in our extended technical report [22] describes this implementation in more detail.

Second, we observe that enterprise networks tend to have a relatively flat topology, since most users prefer to directly access their target server; this behavior limits dependency explosion, which we discuss more in Section 8.2. Third, due to the natural workflows of users and a standard implementation of least privileges, most machines only get accessed by a handful of users for specific job duties. This clustering limits the number of inbound logins per machine, which reduces the potential for path explosion (§ 8.2). Finally, to mitigate path explosion that can occur from users or scripts making many repeated logins to/from a machine, Hopper deduplicates paths to one unique path per day (i.e., one unique set of daily login edges, where a daily edge is a four-tuple of a login's source, destination, target username, and timestamp rounded to the date it occurred).

**Path Components and Types:** Every causal path inferred by Hopper contains the information in Table 2. Each path includes a list of "changepoint" logins: logins that used a different username than the preceding login in the path. For logins that occurred from a client source machine, if the target username does not match the source machine's owner, Hopper also adds this login to its changepoint list.

Hopper computes a path's *causal user* by examining the first (earliest) login in the path. If the login's source machine is a server, then Hopper treats the target username as the path's causal user. However, if the first login's source machine is a client, Hopper takes the owner of that source machine and treats that username as the causal user: clients typically correspond to the start of a user's movement path and logins from these machines should use their owner's credentials. Additionally, Hopper takes a user-provided list of special "bastion" machines: hardened gateway servers that provide access to restricted network segments or machines, and which require users to perform heightened authentication to access these protected parts of the network (e.g., password and hardware-based 2FA authentication during each login). Whenever Hopper encounters a login that originates from a bastion source machine, it treats this login as the root login for the path: i.e., Hopper treats the username of the bastion login as the path's causal user, and stops performing backwards-tracing for the path. Because bastions require robust forms of authentication, logins forwarded from bastion source machines (i.e., logins that successfully authenticated to the bastion server) indicate that the login's purported username does reflect the true actor responsible for making the login.

Paths belong to one of three types: a BENIGN path, a path with a CLEAR credential switch, or a path with UNCLEAR causality. For each changepoint login in a path, Hopper checks whether the changepoint login's username matches any of the usernames across its potential inbound (causal) logins. If all of the inbound hops used a different username, or if the changepoint login originated from a client source machine, then the path has a CLEAR credential switch; otherwise, Hopper labels the path as UNCLEAR. If a path does not have any changepoint logins, then Hopper marks the path as BENIGN.

For example, in Figure 3, if $L_1$, $L_2$, and $L_3$ occurred within 24 hours prior to $L_4$, Hopper will produce 3 causal paths for $L_4$. The paths starting with $L_1$ and $L_3$ will form UNCLEAR paths, and the path starting with $L_2$ will get marked as BENIGN. The path from $L_2$ to $L_4$ will list *Bob* as its causal user and have no changepoints logins. Both the attack path ($L_3$ to $L_4$) and the path from $L_1$ to $L_4$ will list Alice as their causal user, and contain $L_4$ in their list of changepoint logins.

## 6 Detection and Alerting

Hopper classifies each path given two additional inputs: a set of historical logins for feature extraction and a user-provided "budget" that controls the daily number of alerts that Hopper produces for UNCLEAR paths (§ 6.2). Hopper first checks whether the path matches one of five benign scenarios; if so, it does not generate an alert. For paths that do not match a benign scenario, Hopper identifies which of two attack scenarios the path might belong to and applies the scenario's corresponding detector. These detectors apply either a rule set (§ 6.1) or an anomaly scoring algorithm (§ 6.2), and produce an alert if the path is marked as suspicious.

**Benign Movement Scenarios:** In the first benign scenario, Hopper marks a path as benign if every one of its logins uses its causal user's credential (i.e., a path labeled as BENIGN by the causality engine); because these paths do exhibit the first key attack property, Hopper discards them. Hopper also labels approximately 170,000 paths as benign if they match one of four other benign and low-risk scenarios.

First Hopper identifies one-hop paths (i.e., logins) from new machines and new users: Hopper labels the path as benign if either the user and/or source machine have existed for less than one week (based on their earliest occurrence in historical logins and the organization's inventory databases). Second, Hopper ignores all paths that originate from a machine undergoing provisioning for a new owner. As part of this process, an administrator runs a script that authenticates into several specialized servers to configure the machine (e.g., installing the operating system and configuring the new owner's account). These logins will seem suspicious to Hopper because they will use an administrator's credentials (target username) that differs from the machine's owner (the causal user). To identify login events that relate to machine re-provisioning, Hopper checks for three properties: (1) the login's destination belongs to a set of dedicated provisioning servers, (2) the login's target user is a system administrator, and (3) the login originates from a dedicated subnet used for machine provisioning. If Hopper encounters a login with these three properties, it does not run its causality engine or generate an alert. In total, Hopper removes approximately 125,000 logins related to new machines or those undergoing provisioning.

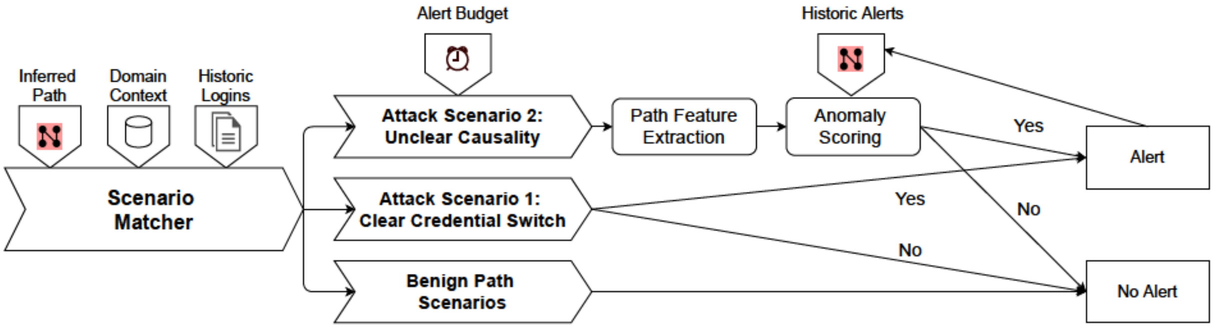


**Figure 4:** Architecture of Hopper's alert generator (§ 6). Given a login path (§ 5), Hopper checks whether the path matches a benign scenario or an attack scenario. Based on the path's scenario, Hopper either discards the path or generates an alert if the scenario's detector triggers.

Third, the use of (non-human) service accounts produces roughly 42,000 one-hop paths that Hopper would otherwise label as cases of clear-credential switching. In these logins, a legitimate user performed a login using a "mismatched" set of credentials that correspond to a service account; however, the credential "switch" in these logins reflects the benign, expected way to access these enterprise services. For example, these logins include users running a script to launch testing jobs when building a new version of Dropbox's desktop application; part of this script includes remote commands issued to the build and test machines under a service account (e.g., user = *test-services*). Hopper infers a set of these service usernames by identifying any username that (1) does not match an employee username, and (2) was used in successful logins from more than ten different source machines across a set of historical data. To ensure that usernames inferred by Hopper do not provide widespread access or highly privileged capabilities, Hopper outputs the set of inferred service accounts for an organization's security team to confirm, and uses only the set of approved service usernames when filtering these benign logins. Because these accounts are designed for a limited and specific service operation, organizations can mitigate the risk of lateral movement via these credentials by configuring them with a limited set of permissions to a specific set of machines; at Dropbox, many of these service accounts also access their destinations via a limited remote command API [1], as opposed to creating a full interactive session.

The final benign scenario involves logins to and from a bastion host. Organizations often segment parts of their network for improved efficiency, maintenance, and security by placing a set of machines behind a hardened bastion host [6, 49]. To access a server within this network segment, a user must first tunnel and authenticate through the network segment's bastion. Dropbox's corporate network contains a few such network segments. Because bastion machines correspond to hardened hosts, perform a limited set of operations (authentication and connection forwarding), and often do not allow users to establish logins onto the host itself, a login that originates from a bastion likely reflects legitimate user activity. Given a list of bastion hosts at an organization, Hopper does not alert on any one-hop path that originates from a bastion or any two-hop paths that traverse a bastion.

**Attack Scenarios:** If a path does not match any of these benign scenarios, Hopper checks whether it matches one of two attack scenarios and, if so, applies the corresponding detection algorithm to see whether it should produce an alert. First, if the path contains a login that switches credentials and the causality engine has high confidence that the switch occurred (a CLEAR path), Hopper applies a simple rule set to classify the path as suspicious or not (§ 6.1). However, because of imperfect information contained in real-world authentication logs, Hopper's causality engine sometimes infers multiple potential paths that a login could belong to, where not all of the paths contain a credential switch (i.e., paths with UNCLEAR causality). Because of this uncertainty, Hopper's second detector evaluates how suspicious each such path is with a probabilistic scoring algorithm (§ 6.2) and alerts if the path has one of the most suspicious scores in recent history.

## 6.1 Attack Scenario 1: Paths with a Clear Credential Switch

Paths with a clear credential switch contain at least one login where Hopper knows that the causal user it inferred for the path must have switched to a different set of credentials (the first key attack property). For these paths, Hopper generates an alert if the path accesses any destination that its causal user has never accessed in prior history; a conservative estimate of when a path's causal user accesses an unauthorized machine.

More formally, let $P$ represent a path with a causal user of Alice and $\text{Dest}_P$ refer to the destination machines across all of $P$'s logins. Hopper generates an alert if $P$ exhibits the two key attack properties:

1. Property 1: $P$ has a CLEAR credential switch (path type).
2. Property 2: $P$ contains at least one destination in $\text{Dest}_P$ that Alice has never accessed in the historical training data (e.g., past 30 days).

## 6.2 Attack Scenario 2: Paths with Unclear Causality

The second attack scenario handles paths with UNCLEAR causality: when Hopper infers multiple causal paths for a login, where some paths contain a credential switch and others do not (§ 5). To handle unclear paths, Hopper uses a probabilistic detection algorithm to identify and alert on paths that are highly anomalous. This selective use of anomaly detection, only in cases where the limitations of authentication logs introduce uncertainty about whether a path contains the key attack properties, distinguishes Hopper from prior work, which simply applies anomaly detection to every path.

**Alert Overview: Unclear Causality:** Given an UNCLEAR path ($P$), Hopper first checks whether the path ever visits a machine that its causal user (Alice) has not previously accessed in the training data (the second attack property). If Alice has access to all of the path's destinations, then Hopper marks the path as benign.[3] Otherwise, Hopper runs the following anomaly detection algorithm on $P$.

First, Hopper extracts three features that characterize $P$'s rareness. Next, Hopper uses $P$'s features to compute a "suspiciousness" score for the path, which it then uses to rank $P$ relative to a historical batch of paths (e.g., the past 30 days). If $P$ ranks among the top $30 \times B$ most suspicious historical paths, then Hopper generates an alert. $B$ corresponds to a user-provided budget that specifies the average number of daily alerts that an analyst has time to investigate for these types of attack paths.

**Path Features:** Hopper uses a set of historical "training" logins to extract three features for a path. Let $A$ refer to the path's starting machine and $Z$ refer to the path's final destination. Given a path's changepoint login ($L_c$), Hopper computes two numerical features. First, Hopper computes the historical edge frequency for each login preceding $L_c$, where an edge's historical frequency equals the number of *days* that a successful login with the exact same edge (source, destination, and target username) has occurred in the training data; the first feature value equals the minimum (lowest) frequency among these preceding logins. Second, Hopper computes the historical edge frequency for each login in the remainder of the path, and takes the lowest frequency value among these hops; i.e., the historical frequency of the rarest login starting at $L_c$ until the path's final hop. For the third feature, Hopper computes the number of historical days where any successful login path connects Machine $A$ and Machine $Z$. If a path has multiple changepoint logins, Hopper computes these three features for each changepoint login, runs its anomaly scoring algorithm (below) for each feature set, and then uses the most suspicious score for the path.

[3] Future logins in the path will cause Hopper to produce extended paths that its detection algorithm will subsequently examine.

**Algorithm 1** Hopper's anomaly scoring algorithm

AlertGen($P$, $A$ (historical alerts), $L$ (historical paths)):
**for** each path $X$ in $A$ **do:**
  **if** Score($P$, $L$) $\geq$ Score($X$, $L$):
    Alert on $P$

Score($P$, $L$): $\prod_F$ Sub-Score($P$, $L$, $F$)

Sub-Score($P$, $L$, $F$ (feature)):
$\text{Sum}_F \leftarrow 0$
$N \leftarrow 0$ (the total # of true causal paths)
**for** each path $X$ in $L$ **do:**
  **if** $P$ has a smaller value for $F$ than $X$:
    $\text{Sum}_F \leftarrow \text{Sum}_F + C_x$
    where $C_x$ = the path certainty for $X$ (§6.2)
  $N \leftarrow N + C_x$,
Sub-Score$_F \leftarrow \text{Sum}_F$ / $N$

**Anomaly Scoring:** Given a path $P$ and its features, Algorithm 1 shows the anomaly scoring procedure that Hopper uses to make its alerting decision. Intuitively, Hopper's scoring algorithm generates an alert for $P$ if it has one of the most suspicious feature sets in recent history.

Hopper's alerting algorithm, ALERTGEN, takes three inputs: a path to score ($P$), a set of historical paths ($L$) to compute $P$'s anomaly score, and a set of historical alerts ($A$) for paths with unclear causality. Hopper generates the set of historical paths ($L$) by iterating over each login in the historical training data and running Hopper's causality engine to produce an aggregate set of all paths for each login. For efficiency, Hopper can compute this set of historical paths as a batch job at the beginning of each week, and reuse it for the entire week's scoring. The historical set of alerts ($A$) consists of the $B \times H$ most suspicious paths during the historical training window, where $H$ is the number of days in the historical window and $B$ is the user-provided alert budget.

With these three inputs, Hopper computes an anomaly score for $P$ that represents the fraction of historical paths where $P$ had more (or equally) suspicious feature values. Hopper then compares $P$'s anomaly score against the scores of the historical alerts, and generates an alert for $P$ if its score exceeds any historical alert's score; i.e., Hopper produces an alert if $P$ is at least as suspicious as a previous alert's path.

**Computing Scores:** Conceptually, a path $P$'s anomaly score corresponds to a cumulative tail probability: how much more suspicious (unlikely) is $P$ relative to the kinds of paths that benign users historically make? As described in the SCORE subroutine in Algorithm 1, Hopper calculates this score by computing a sub-score for each of the path's features, and then multiplies these sub-scores to get an overall score.

Each feature's sub-score estimates the fraction of historical paths where $P$ had a more suspicious feature value. In practice, imprecision from Hopper's path inference algorithm could lead a naive computation of this fraction to over-count certain historical paths. For example, a historical login from a server with many ($N$) inbound logins will generate $N$ historical paths, even though only one of those paths reflects a true causal path. These types of paths, that involve servers with many inbound logins, will have an inflated volume that could skew the anomaly sub-scores that Hopper computes; i.e., their features will be over-represented in the historical distribution. To mitigate this problem, when computing the set of paths for each historical login $L_i$, Hopper annotates each path with a "Path Certainty" fraction, denoted as $C$, that equals 1 / the total number of causal paths that Hopper inferred for $L_i$. When Hopper computes each sub-score for the current path $P$, it uses $C$ to down-weight the impact of each historical path (Line 5 of the SUB-SCORE routine in Algorithm 1).

**Alert Clustering:** To avoid generating redundant alerts for the same path, Hopper clusters its alerts each day. Hopper maintains a list of every alert (path) it generates on the current day. If a new alert path traverses the same exact edges as any path on the day's alert list, Hopper updates the existing alert with information about this duplicate path and does not generate a new alert.

### 6.3 Real-time Detection

Organizations can run Hopper as a real-time detector using a design similar to the architecture described above. For real-time detection, Hopper would maintain a "recent login" queue of all logins over the past $T$ hours, where $T$ corresponds to the causality threshold described in § 5. For each new login, Hopper can run the path inference procedure described in Section 5, and then apply its scoring algorithms to determine whether any path produces an alert. Each night, Hopper can prune the queue of recent logins to only retain those in the past $T$ hours, recompute the set of historical paths used for feature extraction, and update the set of the historical alert paths that Hopper uses when assessing a new path's anomaly score (Section 6.2). This real-time architecture retains the same detection accuracy as running Hopper as a batch detector, since it makes no difference whether Hopper classifies each day's logins individually or in one aggregate batch.

## 7 Evaluation

We evaluated Hopper on our 15-month data set, measuring its detection rate (fraction of attacks detected) and the volume of false positives it generates. Our data does not contain any known lateral movement attacks, but it does contain one in-situ lateral movement attack conducted by Dropbox's professional red team. Additionally, we generated and injected a realistic and diverse set of 326 simulated attacks into our data for a more thorough evaluation (§ 7.2). Hopper successfully detected 94.5% of the attacks in our data, including the red team attack, while generating an average of 9 false positives per day (§ 7.3): an 8× reduction in the number of false positives produced by prior state-of-the-art (§ 7.4).

| Path Length | # of Paths with Potential Credential Switch |
|---|---|
| 2 | 3,357,353 |
| 3 | 829,044 |
| 4 | 128 |
| 5 | 6 |
| 6 | 4 |

**Table 3:** The volume of multi-hop paths, with a potential switch in credentials, inferred by Hopper's causality engine. The left column reports the path length and the right column reports the total number of paths with that length that Hopper generated, across our dataset.

### 7.1 Implementation

For our experiments, we implemented Hopper in Python 2.7 on a Linux server with 64GB of RAM and a 16-core processor. Table 3 shows the total number of multi-hop paths that Hopper generated, based on the optimized implementation described in our extended technical report [22]. In aggregate, the full set of paths (containing the attributes described in Table 2 and their feature values) consume a total of 2.5GB of memory. Running Hopper's path generation algorithm across our entire data set took a total CPU time of 35 minutes and 13 seconds, and running Hopper's feature extraction and detection algorithms on every day in our data set took a cumulative CPU time of 83 minutes and 9 seconds.

The dramatic drop in long-length paths reflects the fairly flat topology of Dropbox's network, the filtering steps that Hopper takes to remove noisy and spurious login activity (§ 3.2), and the optimization Hopper uses of only tracking paths with potential (or clear) credential switching. System administrator activity predominates these multi-hop paths, since most other users perform logins directly into their target service (e.g., short one-hop paths).

### 7.2 Attack Data

**Red Team Attack:** Our data contains one lateral movement attack generated by Dropbox's professional red team. The red team began their attack from a "compromised" employee's laptop (selected from a preexisting pool of volunteers).[4] Their attack simulated a common APT scenario [17, 51], where an

[4] The red team followed their standard safety protocols when conducting this simulation, which included obtaining prior consent from all "compromised users", coordinating extensively with the security incident response team, and conducting any necessary remediation that resulted from the simulated attack (e.g., resetting any credentials that they accessed).

attacker conducts lateral movement to access an organization's Domain Controllers (credential management servers). From their initial foothold, the red team conducted a series of reconnaissance and internal login (lateral movement) operations. They identified and acquired a new, elevated set of credentials, which they then used to access one of the organization's Domain Controllers. Apart from requiring that their movement occurred via logins (as opposed to exploiting a remote access vulnerability), the red team performed this attack under no constraints or input from us. We did not examine the red team data until we had frozen the design and parameters of our detector. The red team's attack created an UNCLEAR path, because the attack "stole" and used a sysadmin's credentials from a server that had a recent inbound login by the sysadmin. Hopper's unclear causality detector successfully identified this attack. Based on its anomaly score, Hopper ranked this attack path as the most suspicious path on that day and the 45th most suspicious path across all paths during the month of the attack.

**Realistic Attack Simulations:** Dropbox employs multiple sets of security controls and detection approaches, including commercial security products, external security audits, and custom tools developed by in-house security teams. Across all of these sources, no incidents of real-world lateral movement have been detected. Given the lack of real-world attack instances, we developed an attack synthesis framework and generated an additional 326 realistic lateral movement attacks. Our attack framework covers a wide range of real-world attacks described in public breach reports and academic surveys [43], ranging from ransomware to targeted APT attacks.[5]

We randomly selected 50 employees in our data as starting victims, whose machines served as "compromised" footholds for attackers to launch their lateral movement. For each starting victim, our framework synthesized twelve different attack scenarios, corresponding to a pairing of one of three ATTACK GOALS with one of four types of STEALTHINESS.

Given a starting victim and attack scenario, our framework synthesizes a set of lateral movement login entries that begin at a random date and time (when the starting victim was still active in our data). Leveraging the global graph of all logins in our data set, our framework simulates an attacker who iteratively (1) accrues a set of "compromised" credentials (the starting victim's credentials, and after each new login, the users who recently accessed the login's destination machine), and then (2) synthesizes login entries to new destinations that the attack's compromised credential set can access.

The three attack goals specify when an attack succeeds (stops generating new logins) and the shape of the attack's movement. Modeling ransomware, an *Aggressive Spread* attack generates new logins by iterating over its compromised credential set and performs logins into every machine accessible by each credential; this attack terminates after accessing 50 machines, or once it makes a login into every machine available to its final credential set. An *Exploratory Attack* stops generating new logins once it accesses a machine that its initial victim did not have access to; this attack iteratively generates new logins by randomly selecting a credential from its compromised set and a new destination accessible to the selected credentials. *Targeted Attacks* perform logins until they access a high-value server (e.g., Domain Controllers). These attacks generate logins by computing a shortest path to elevated credentials that can access a high-value server, and then compute a shortest path that uses these new credentials to access the high-value server.

[5]Our simulation code is available at https://github.com/grantho/lateral-movement-simulator

| | Exploratory | Aggressive | Targeted | TP Rate |
|---|---|---|---|---|
| **No stealth†** | 37 / 41 | 38 / 41 | 38 / 40 | 113 / 122 |
| **Prior Edge** | 13 / 14 | 14 / 14 | 10 / 13 | 37 / 41 |
| **Active Cred.** | 41 / 41 | 41 / 41 | *39 / 41 | 121 / 123 |
| **Combined** | 12 / 14 | 14 / 14 | 12 / 13 | 38 / 41 |
| **Detection Rate** | 103 / 110 | 107 / 110 | 99 / 107 | 309 / 327 |

**Table 4:** Summary of Hopper's detection (true positive) rate across the different scenarios simulated by our attack framework and the red team attack (§ 7.2). Rows correspond to the four different stealthiness levels and columns correspond to the three attack goals that our framework simulated for each user. The last column and last row report Hopper's overall detection (TP) rate. The scenario marked with an asterisk (TARGETED and ACTIVE CRED) includes one red team attack, which Hopper detected. †The false negatives in the "No stealth" row stem from inaccurate attributes in the attack logins.

Additionally, our attack framework only produces logins that follow the scenario's specified stealthiness. An attack with *Prior Edge* stealthiness only generates logins that traverse edges that legitimate users have previously made. An attack with *Active Credential* stealthiness only uses a set of credentials in a login if the credential's legitimate user was recently logged into the source machine (i.e., creating login paths with unclear causality). An attack with *Combined Stealthiness* only generates logins with both of the properties above (e.g., mimicry-style attacks). The fourth type corresponds to an attacker without any stealthiness requirements.

We generated 326 successful attacks, with 205 attacks across the three stealthier levels (Table 4); users did not always have viable attack paths, leading to less than 50 attacks per scenario (e.g., users with limited access or who lacked stealthy paths for a targeted attack). The red team attack corresponded to a *Targeted Attack* with *Active Credential* stealthiness; our framework can produce the same attack path if we run it from the same starting victim with these parameters.

## 7.3 Results

**Evaluation Procedure:** We divided our data into a 2-month training window (Jan 1 – Mar 1, 2019), which we used to bootstrap the feature extraction and scoring components of Hopper that require historical data, and a 13-month evaluation window (Mar 1, 2019 to Apr 1, 2020). Our evaluation data contained 713,617,425 successful logins, and 2,941,173 logins after applying Hopper's data filtering steps (§ 3.1). We ran Hopper over this evaluation data to compute its false positive rate and detection (true positive) rate. For any detection component that required historical training data, we used a rolling window of the preceding 30 days. For our anomaly scoring algorithm (§ 6.2), we used a budget of 5 alerts / day, and explore the sensitivity of this parameter below.

**Attack Detection Rate (True Positives):** For each of the 326 attacks synthesized by our framework, we injected the attack's logins into our evaluation data and ran Hopper on the day(s) when the attack occurred. For the red team exercise, we examined the alerts that Hopper generated on the day of the attack. We deemed Hopper successful if it generated an alert for any attack path made by the simulated attacker or red team.

Table 4 shows that Hopper successfully detected a total of 309 attacks (94.5%), which includes the attack performed by Dropbox's expert red team. Hopper detected 138 attacks through its rule set for paths with clear credential switching (§ 6.1). In all of these attacks, the simulated attacker either used a new set of credentials in a login from their initial foothold machine or from a server that the legitimate user (of the new credentials) had not recently accessed, enabling Hopper to identify a movement path where the attacker clearly switched to using new credentials.

However, most (180) attacks created paths with UNCLEAR causality, either because the attack quickly capitalized on new credentials that were recently used on a server, or because the attack simulated a stealthy adversary who only used new credentials from machines where the legitimate user was recently or currently active. Detecting these paths falls to Hopper's anomaly scoring detector (§ 6.2). With a budget of 5 alerts per day, Hopper successfully identified 171 of these attacks (95%), including the red team attack.

**False Negatives:** Of the 18 false negatives, Hopper missed 9 attacks because of attribute errors in the login data. For each of these 9 false negatives, the attack logins had an incorrect client vs. server label for a machine, and/or contained incorrect information about a machine's owner. If we replaced this inaccurate login information with the correct attributes (acquired from additional, up-to-date data sources at Dropbox), Hopper could successfully detect all 9 of these false negatives with its *clear credential switch* detector. Nonetheless, we count these attacks as false negatives since real data inevitably contains imprecise information. Additionally, Hopper failed to detect 9 stealthy attacks using a daily budget of 5 alerts. For all of these false negatives, every attack login traversed an edge with at least three prior days where the legitimate user had performed a login along the edge.

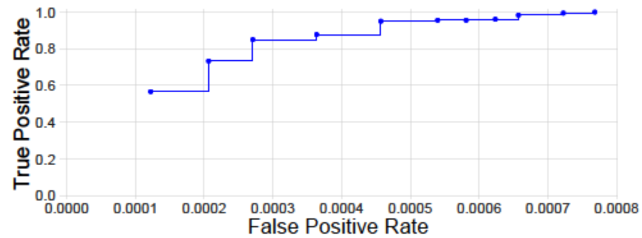


**Figure 5:** ROC Curve for Hopper's unclear causality detector (§ 6.2) at different budgets (1–11 daily alerts). The True Positive Rate reports the fraction of (180) attacks with unclear causality that Hopper detects. The FP Rate reports the number of false alarms divided by the number of logins in our evaluation data (2.94M).

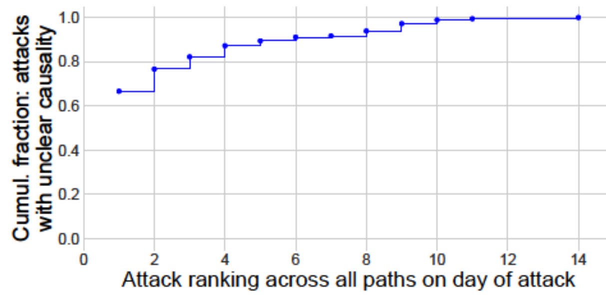


**Figure 6:** The ranking of attack paths with UNCLEAR causality, relative to all of the login paths that occurred on the day of an attack.

**Budget Sensitivity and Attack Rankings:** Including the red team attack, 180 attacks produced paths with unclear causality. Figure 5 shows the detection performance of Hopper for these attacks, using different daily budgets for its anomaly scoring detector. Hopper uses this budget to build a set of the historical alerts over the past month, and then alerts on a new path (with unclear causality) if its score is greater than or equal to any scores of the historical alerts (§ 6.2). If Hopper used a daily budget of 11 alerts, it could eliminate 9 false negatives and detect all 180 attacks with a false positive rate of 0.00076.

We also assessed the ranking of these UNCLEAR PATH attacks relative to the benign paths in our data, based on their anomaly scores. Figure 6 shows that Hopper ranks these attacks as highly suspicious, with over 66% of attacks ranked as the most suspicious path on the day each attack occurred.

**False Positives:** To compute Hopper's false positive rate, we ran Hopper on all non-synthesized logins for each day in our evaluation data. We conservatively labeled all of the alerts Hopper produced as false positives if they did not relate to the red team attack.

With a daily budget of 5 alerts for its anomaly scoring detector, Hopper's two detection algorithms generated a total of 3,560 false positives (FP) across the 396-day evaluation window: an average of 9 alerts / day and a false positive rate of 0.0012 across the 2.94M filtered logins in our evaluation data. Hopper's rule-based detector for CLEAR paths produced

| Detector | Detection Rate | False Positives |
|---|---|---|
| SAL (equal FP) | 156 / 327 (47.7%) | 3,556 (0.12%) |
| SAL (equal TP) | 309 / 327 (94.5%) | 27,927 (0.94%) |
| Hopper | 309 / 327 (94.5%) | 3,560 (0.12%) |

**Table 5:** Prior state-of-the-art, SAL [44], produces 8× as many FP as Hopper to detect the same number of attacks. At a similar number of FP's as Hopper, SAL detects roughly half as many attacks (§ 7.4).

2,216 FP's, and the remaining 1,344 FP's come from Hopper's anomaly scoring detector. On some days, Hopper's anomaly scoring detector generated less than 5 alerts because (1) not every day had 5 suspicious paths with unclear causality (e.g., weekends and holidays), and (2) our alert clustering resulted in some days with fewer alerts (§ 6.2).

We identified several common reasons for many of these false positives. Across the 2,216 false positives generated by our CLEAR path detector, approximately 10% of these false positives correspond to logins where a user's laptop accesses a particular service using a special service account. Another 41.5% correspond to machine imaging and provisioning activity, where a sysadmin runs a script that uses their elevated set of credentials to configure a laptop for a new owner (these logins occurred at a remote office that Hopper's data cleaning steps did not filter out). Finally imprecision in Hopper's causality engine contributed to 19% of Hopper's CLEAR path false positives and over 49% of Hopper's UNCLEAR-causality false positives. Many of these false positives are paths, initiated by one system administrator, that purportedly make a login that switches to another system administrator's credentials. These alerts often involve a handful of "gateway" machines that sysadmins use to access important internal servers (e.g., Domain Controllers). Hopper generates these false alerts when multiple sysadmins have recently logged into a gateway machine, and one sysadmin launches a login from the gateway machine to a rarely-accessed or niche server. Because these paths involve only administrator credentials, Hopper could reduce its false positives by filtering them out; any credential switch between two administrators likely provides limited additional access.

### 7.4 Comparison with Prior State-of-the-Art

We compared Hopper's performance against the best performing prior work, the Structurally Anomalous Login (SAL) detector proposed by Siadati and Memon [44]. SAL detects lateral movement by generating a set of logins that traverse a rare edge in the login graph (based on a user-specified threshold). Next, SAL learns and uses a set of "benign login patterns" to identify which rare edges to alert on. Each login pattern corresponds to a triplet of (source machine attributes, destination machine attributes, and user attributes). For example, given the login (src = Machine A, dest = Machine B, user = Alice), (src = New York, dest = San Francisco, user = Engineering) would be one login pattern, if Machine A resides within New York, Machine B resides within San Francisco, and Alice works on the Engineering team. SAL learns a set of benign patterns by using a historical set of logins to identify patterns where a sufficiently large fraction of source machines, destination machines, and/or users have at least one historical login that matches a pattern. SAL then produces an alert for every rare-edge login that does not match a benign pattern.

Based on the data available to us, we use the following set of login attributes from the SAL paper: each user has two attributes: (the user's team, and the user's type: system administrator, regular user, or service account) and each machine has two attributes: (the machine's type: client or server, and the machine's geographic location). We applied SAL with a rolling two-month training window on all of the filtered logins in our evaluation window (i.e., the same data used for Hopper's evaluation; we also applied both the data filtering and benign scenario pruning outlined in § 3.1 and § 6). SAL takes two user-provided thresholds for training and classification, respectively.[6] Table 5 reports the results for SAL using the parameters that produced the minimum volume of FP's to detect (1) the same number of attacks as Hopper and (2) (approximately) half as many attacks as Hopper. We report the number of FP's SAL produces after de-duplicating the alerts to only include one edge (source, destination, and target user) per day, and we considered SAL successful if it produced an alert for *any* malicious login in an attack.

SAL produces nearly 8× as many false positives as Hopper to detect the same number of attacks. Whereas Hopper selectively chooses when to apply anomaly detection (to resolve uncertainty in paths that might have the two key attack properties), SAL follows a traditional machine learning approach by simply applying anomaly detection to every login, resulting in significantly more false positives.

### 7.5 Attack Case Studies

Below, we describe two attacks created by our synthesis framework, and examine how Hopper and traditional anomaly detection approaches, such as SAL, handle them.

**Example Attack 1: Targeted Compromise:** One attack simulated an adversary who began their lateral movement from an engineer's laptop and then attempted to access one of several high-value machines within an organization (e.g., a Domain Controller). After three logins, the attacker arrived on a machine where a system administrator, Bob, had recently logged into the machine via ssh. Simulating an attacker compromising and using Bob's ssh credentials (e.g., by abusing a forwarded SSH agent), our framework created a fourth attack

[6] Our extended technical report shows SAL's performance under the range of parameters we explored [22].

login that leveraged Bob's credentials to access a server that manages user permissions and SSH keys.

The last two logins involved in this attack path rarely occur, enabling SAL to detect this attack with a low volume of false positives. Similarly, Hopper successfully detects this attack, even though it involves an attack path with unclear causality (since the sysadmin had an active ssh session that could have launched the final login into the ssh management server); the rareness of the attack path's edges led Hopper to rank it among the top 10 most suspicious paths that month.

**Example Attack 2: Stealthy, Short Paths:** For each user, our framework also simulated attacks that modeled a stealthy adversary who only accesses machines via previously traversed graph edges. In one such attack, starting from a compromised user (Alice)'s machine, our framework first synthesized a login to a server ($Y$) that Alice had previously accessed (4 out of the past 60 days). After moving to Server $Y$, the attacker observed that Server $Y$ still had the credentials of a sysadmin, Bob, cached from a login during the past week, enabling the attacker to acquire them. The attacker (our framework) also observed that Bob had previously logged into a powerful remote management machine from Server $Y$ (3 out of the past 60 days). Accordingly, our framework synthesized a final, second attack login using Bob's credentials to access this high-value server. Although seemingly simple, this attack reflects a realistic path for a stealthy attacker, since shorter paths provide fewer opportunities for detection.

Hopper detected this attack with its CLEAR path detector: the second login switched to a new target username, but over 24 hours elapsed since Bob accessed Server $Y$. Even if Bob had logged into Server $Y$ more recently, Hopper would still have caught this attack under its anomaly scoring detector (which ranks the attack path among the top 20 most suspicious in the past month). In contrast, because this attack only traverses edges with prior history, SAL would produce at least 14,000 alerts across our 13-month evaluation data to detect it.

## 8 Discussion

Hopper achieves good results on the real-world data set we used. However, a number of interesting future directions remain, including overcoming potential evasion strategies, understanding how Hopper generalizes across different enterprise network architectures, and extending Hopper's detection approach to achieve better performance.

### 8.1 Evasion and Limitations

An attacker might evade detection if they can access their target machines by piggybacking on a series of logins made by legitimate users [35], or if the attacker finds a frequently traveled login path that provides access to their target. Our evaluation explicitly generated attacks that pursued this stealthy strategy, and Hopper could detect many of these attacks. The attacks that Hopper failed to detect had UNCLEAR causality, followed paths with frequently traveled edges, and occurred on days with other UNCLEAR paths whose edges occurred more infrequently. However, we note that attackers might not always be able to make such stealthy movement: when synthesizing attacks across our sample of 50 random starting users, 37 users could not stealthily access a high-value server; i.e., attackers who compromised these users' machines had no path to our set of sensitive machines, or would need to make at least one rare-edge login to access them.

Although our threat model focuses on interactive attackers who manually perform their movement, attackers could evade detection by installing stealthy malware on a shared server that lies on the path to their final target machine. Such malware could wait until the maximum session duration (time threshold for causally linking two logins together) has elapsed. Once this time has elapsed, the malware could then opportunistically launch the subsequent logins in its attack path whenever a legitimate user (e.g., *Bob*) performs an inbound login into the shared server. This strategy will cause Hopper to causally link the second half of the attack path, that abuses Bob's credentials, to Bob's earlier legitimate logins, creating a BENIGN path that appears to consistently use one set of credentials. Because this approach increases attacker dwell time and their host footprint, complimentary techniques such as binary allow-listing, anti-virus, and additional detection signals (§ 8.3) can help increase the chance of detection.

Missing or inaccurate logging information can also create false negatives, a problem common to any detection strategy. Future work can explore ways to alleviate this challenge by using multiple sources of information to determine the correct attributes of login data. Additionally, organizations can deploy commercial log-hygiene solutions to continuously monitor and collate their logging data.

### 8.2 Generalizability

Although we evaluate Hopper on a large real-world data set, Hopper's performance could change at enterprises with significantly different network architectures and security policies. For example, Dropbox makes a dedicated effort to scope employee access based on the least privileges principle; at organizations where many users have highly privileged access, an attacker may not need to acquire additional credentials to achieve their desired goal. As a result, lateral movement attack paths might not exhibit a switch in credentials, allowing adversaries to evade detection. For such organizations, implementing better permissions hygiene will likely yield greater security benefits than any detection strategy. We view Hopper as a promising direction for securing enterprises against attacks that could succeed in spite of the adoption of such security best practices.

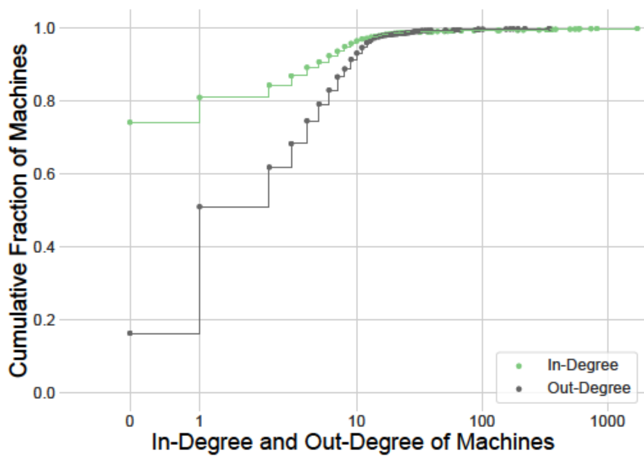


**Figure 7:** The in-degree and out-degree distribution across hosts at Dropbox. The in-degree for a host equals the number of machines that it has received logins from; the out-degree counts how many unique machines each source machine makes at least 1 login into.

With respect to the impact of a network's architecture on Hopper's performance, we observe that two properties contribute to Hopper 's success: a relatively flat network topology and consistent workflows across most users that only access a small subset of machines. Below, we characterize the graph topology at Dropbox, and explain why we believe many organizations will also exhibit these two properties, allowing Hopper to generalize to other networks.

**Network Topology of Dropbox:** If we aggregate all of the logins across our dataset, the unified graph has a diameter of length 7 and an average shortest path length of 2.12 hops. The graph contains 10,434 unique edges, where each edge consists of a (source machine, destination machine) tuple; when edges also include the username involved in a login, the graph contains 27,718 unique edges. Figure 7 shows the in-degree and out-degree distribution for all machines in our data: i.e., the number of distinct machines that a node receives logins from and makes logins to. The servers with in-degrees of over 100 inbound machines correspond to common enterprise services, such as Windows Domain Controllers that handle Kerberos-based authentication, printers, telemetry and logging machines, and servers involved in provisioning new machines. Clients (e.g., laptops) represent 65% of the machines in our data, resulting in many machines with an in-degree of 0. Machines with high out-degrees (logins to over 100 different destinations) correspond to system administrator machines, as well as internal scanning and monitoring servers.

**Impact of Different Network Configurations:** One of the biggest challenges that Hopper faces is the risk of path explosion and an overwhelming number of suspicious paths with unclear causality. This situation can occur if many servers have large numbers of users that access them, who then launch outbound logins from the common servers to other machines. If this behavior occurs multiple times along a path, it risks an exponential increase in the number of paths that Hopper will infer. This path explosion might lead not only to unsuitable run-time performance (e.g., consuming too much memory), but could also lead to a large number of false positives. If many of these incorrectly inferred movement paths have a suspicious set of features, then Hopper may generate a substantial number of false alerts related to these paths. Two factors mitigated the problem of path explosion in our data set: a relatively flat network topology and the natural clustering of user access patterns to a few work-related machines.

Flat networks arise because most (non-sysadmin) user activity consists of direct logins from their client machines to the server that hosts their desired functionality or data. Moreover, because many servers provide a limited UI and set of functionality, they often do not provide an easy way to launch outbound logins. This property means that even when a server has many inbound logins from users, it often does not risk path explosion because subsequent outbound logins do not occur. We expect that even as the number of users and servers increases, these natural habits will keep access patterns relatively flat; this behavior will increase the number of short login paths, but continue to limit the number of long paths. At Dropbox, we did observe processes that generated long paths, such as when users need to access a server by tunneling through a gateway (bastion) machine, automated activity (e.g., domain controllers iteratively synchronizing data amongst each other), and system administrator activity. However, most of the paths from these activities either do not contain both attack properties (e.g., no switch in credentials or no new access for the path's potential causal users), or they get removed by Hopper's filtering procedure since they do not pose a large risk for lateral movement (§ 3.1).

Second, users tend to access machines for a specific job function, creating a sparse graph where different subsets of logins naturally cluster around a small group of machines (e.g., at Dropbox over 90% of machines have an in-degree $\leq 10$ and an out-degree $\leq 10$). Implementing least privileges, where users have access to only a small set of machines relevant to their work, also reinforces this common behavior. As a result, most machines only get accessed by a limited set of users, which reduces path explosion and the number of paths with unclear causality. Furthermore, because users accessing a shared server typically work on the same team or have similar job roles, their credentials often have similar privileges and they tend to access the same broader set of machines. Thus, even when Hopper produces paths with unclear causality, these paths often do not provide access to an unauthorized machine for their causal user (the second attack property), and get marked as benign. Since this property arises from common user behavior and security policies, and has been observed at different organizations [44], we expect many other networks exhibit similar partitioning.

**Hopper's Causality Time Threshold:** Hopper uses a time-based threshold, equal to the maximum remote session duration at an organization, to help infer when logins form a movement path (§ 5). We discussed this session duration with the security teams of multiple companies, and all of them implement a similar length policy for remote login sessions (e.g., ssh and RDP), based on commonly-adopted, best-practice recommendations [14], and in some cases compliance and cyber-insurance guidelines [7,21,26]. Additionally, even if we doubled the 24-hour threshold that Hopper used in our evaluation, Hopper achieves an 89.9% detection (true positive) rate while generating an average of 9 false alarms / day.

### 8.3 Extending Hopper

To further improve Hopper's performance, future work could explore prioritizing paths that involve particularly sensitive credentials or machines. For example, Hopper could assign a higher anomaly score to any path that accesses a sensitive machine (specified by an organization). Similarly, Hopper could prioritize paths where the causal user elevates themselves to an administrator account over the course of the path's logins.

Complementary work uses system logs to detect suspicious host activity that aligns with attacker behavior enumerated in the MITRE ATT&CK framework [18,24,25,38]. Organizations could combine these approaches with Hopper to gain insight into both malicious host activity as well as suspicious (lateral) movement between hosts.

Finally, Hopper would generate fewer false positives if it more precisely inferred causally-linked logins. Future work could explore how drawing upon additional data sets, such as network traffic or host logs, could enable more accurate causal inference. For example, to determine which inbound login caused an outbound login, Hopper could analyze the inbound versus outbound network flows across the candidate logins to pinpoint pairs with overlapping timing and flow sizes.

## 9 Conclusion

This paper presented Hopper, a system that develops a graphical model of enterprise logins to detect lateral movement. On a 15-month enterprise data set, Hopper detected 94.5% of realistic attack scenarios at a false positive rate of 0.0012. These results illustrate the power of a causal understanding of the movement paths that users make between *internal* enterprise machines. By identifying which logins belong to the same logical movement path and the user responsible for initiating each path, Hopper can identify a diverse range of attacks while generating 8× fewer false positives than prior state-of-the-art. Although common authentication logs make inferring precise causality difficult, Hopper's use of specification-based anomaly detection — selectively applying anomaly detection only in cases of high uncertainty — enables our approach to achieve good detection performance.


## Acknowledgements

We thank Dropbox's security team for supporting this research, and John Cramb in particular for his help conducting the red team exercise. This work was supported in part by the Hewlett Foundation through the Center for Long-Term Cybersecurity, NSF grants CNS-1237265 and CNS-1705050, an NSF GRFP Fellowship, the UCSD CSE Postdoctoral Fellows program, the Irwin Mark and Joan Klein Jacobs Chair in Information and Computer Science (UCSD), by generous gifts from Google and Facebook, and operational support from the UCSD Center for Networked Systems.


## References


[1] Russ Allbery. remctl: Remote authenticated command execution. https://github.com/rra/remctl, 2018.

[2] Atul Bohara, Mohammad A Noureddine, Ahmed Fawaz, and William H Sanders. An unsupervised multi-detector approach for identifying malicious lateral movement. In IEEE Symposium on Reliable Distributed Systems (SRDS), 2017.

[3] Xander Bouwman, Harm Griffioen, Jelle Egbers, Christian Doerr, Bram Klievink, and Michel van Eeten. A different cup of TI? the added value of commercial threat intelligence. In USENIX Security Symposium, 2020.

[4] Benjamin Bowman, Craig Laprade, Yuede Ji, and H. Howie Huang. Detecting lateral movement in enterprise computer networks with unsupervised graph AI. In International Symposium on Research in Attacks, Intrusions and Defenses (RAID), 2020.

[5] CERT. Advanced persistent threat activity targeting energy and other critical infrastructure sectors. https://www.us-cert.gov/ncas/alerts/TA17-293A, 2017.

[6] Cisco. What is network segmentation? https://www.cisco.com/c/en/us/products/security/what-is-network-segmentation.html, 2019.

[7] PCI Security Standards Council. PCI DSS Prioritized Approach for PCI DSS 3.2. https://www.pcisecuritystandards.org/documents/Prioritized-Approach-for-PCI_DSS-v3_2.pdf?agreement=true&time=1469037392985, 2016. Section 8.1.8.

[8] CrowdStrike. Lateral movement. https://www.crowdstrike.com/epp-101/lateral-movement/, Sep 2019.

[9] Assaf Dahan. Operation cobalt kitty. https://www.cybereason.com/blog/operation-cobalt-kitty-apt, 2017.

[10] John Dunagan, Alice X Zheng, and Daniel R Simon. Heat-ray: combating identity snowball attacks using machinelearning, combinatorial optimization and attack graphs. In ACM Symposium on Operating Systems Principles (SOSP), 2009.

[11] Jim Finkle and Susan Heavey. Target says it declined to act on early alert of cyber breach. http://www.reuters.com/article/us-target-breach-idUSBREA2C14F20140313, Mar 2014.

[12] Scott Freitas, Andrew Wicker, Duen Horng Chau, and Joshua Neil. D2M: Dynamic Defense and Modeling of Adversarial Movement in Networks. In SIAM International Conference on Data Mining, 2020.

[13] Sergiu Gatlan. Microsoft shares tactics used in human-operated ransomware attacks. https://www.bleepingcomputer.com/news/security/microsoft-shares-tactics-used-in-human-operated-ransomware-attacks/, Mar 2020.

[14] Paul A. Grassi, Elaine M. Newton, Ray A. Perlner, Andrew R. Regenscheid, James L. Fenton, William E. Burr, Justin P. Richer, Naomi B. Lefkovitz, Yee-Yin Choong, Kristen K. Greene, Jamie M. Danker, and Mary F. Theofanos. NIST Special Publication 800-63B: Digital Identity Guidelines. https://doi.org/10.6028/NIST.SP.800-63b, 2017. Section 4.3 – 4.5.

[15] Robert Hackett. Anthem, a major health insurer, suffered a massive hack. http://fortune.com/2015/02/05/anthem-suffers-hack/, Feb 2015.

[16] Aric Hagberg, Nathan Lemons, Alex Kent, and Joshua Neil. Connected Components and Credential Hopping in Authentication Graphs. In International Conference on Signal-Image Technology and Internet-Based Systems, 2014.

[17] Alexander Hanel. Big game hunting with ryuk: Another lucrative targeted ransomware. https://www.crowdstrike.com/blog/big-game-hunting-with-ryuk-another-lucrative-targeted-ransomware/, Jan 2019.

[18] Wajih Ul Hassan, Adam Bates, and Daniel Marino. Tactical provenance analysis for endpoint detection and response systems. In IEEE Symposium on Security & Privacy, 2020.

[19] Wajih Ul Hassan, Mohammad A Noureddine, Pubali Datta, and Adam Bates. Omegalog: High-fidelity attack investigation via transparent multi-layer log analysis. In Network and Distributed System Security Symposium, 2020.

[20] Sarah Hawley, Ben Read, Cristiana Brafman-Kittner, Nalani Fraser, Andrew Thompson, Yuri Rozhansky, and Sanaz Yashar. APT39: An iranian cyber espionage group focused on personal information. https://www.fireeye.com/blog/threat-research/2019/01/apt39-iranian-cyber-espionage-group-focused-on-personal-information.html, Jan 2019.

[21] HIPAA. HIPAA: 45 CFR § 164.312: Technical safeguards. https://www.law.cornell.edu/cfr/text/45/164.312, 2013. Section 164.312(a)(2)(iii).

[22] Grant Ho, Mayank Dhiman, Devdatta Akhawe, Vern Paxson, Stefan Savage, Geoffrey M. Voelker, and David Wagner. Hopper: Modeling and Detecting Lateral Movement (Extended Report). In arXiv, 2021.

[23] Grant Ho, Aashish Sharma, Mobin Javed, Vern Paxson, and David Wagner. Detecting credential spearphishing in enterprise settings. In USENIX Security Symposium, 2017.

[24] Md Nahid Hossain, Sanaz Sheikhi, and R Sekar. Combating dependence explosion in forensic analysis using alternative tag propagation semantics. In IEEE Symposium on Security & Privacy, 2020.

[25] Md Nahid Hossain, Junao Wang, Ofir Weisse, R Sekar, Daniel Genkin, Boyuan He, Scott D Stoller, Gan Fang, Frank Piessens, and Evan Downing. Dependence-preserving data compaction for scalable forensic analysis. In USENIX Security Symposium, 2018.

[26] IRS. Safeguard Security Report. https://www.irs.gov/pub/irs-utl/irs_safeguards_annotated_ssr_template.pdf, 2014. Section 9.3.1.

[27] Alexander D Kent, Lorie M Liebrock, and Joshua C Neil. Authentication graphs: Analyzing user behavior within an enterprise network. Computers & Security, 2015.

[28] Robert M. Lee, Michael J. Assante, and Tim Conway. Analysis of the cyber attack on the ukrainian power grid. https://ics.sans.org/media/E-ISAC_SANS_Ukraine_DUC_5.pdf, Mar 2016.

[29] Fucheng Liu, Yu Wen, Dongxue Zhang, Xihe Jiang, Xinyu Xing, and Dan Meng. Log2vec: A Heterogeneous Graph Embedding Based Approach for Detecting Cyber Threats within Enterprise. In ACM Conference on Computer and Communications Security (CCS), 2019.

[30] Qingyun Liu, Jack W Stokes, Rob Mead, Tim Burrell, Ian Hellen, John Lambert, Andrey Marochko, and Weidong Cui. Latte: Large-scale lateral movement detection. In IEEE Military Communications Conference (MILCOM), 2018.

[31] Mandiant. Apt1: Exposing one of china's cyber espionage units. https://www.fireeye.com/content/dam/fireeye-www/services/pdfs/mandiant-apt1-report.pdf, 2013.

[32] Sadegh M Milajerdi, Rigel Gjomemo, Birhanu Eshete, R Sekar, and VN Venkatakrishnan. Holmes: real-time apt detection through correlation of suspicious information flows. In IEEE Symposium on Security & Privacy, 2019.

[33] Ellen Nakashima. Chinese breach data of 4 million federal workers. https://www.washingtonpost.com/world/national-security/chinese-hackers-breach-federal-governments-personnel-office/2015/06/04/889c0e52-0af7-11e5-95fd-d580f1c5d44e_story.html, Jun 2015.

[34] NCSC. Joint report on publicly available hacking tools. https://www.ncsc.gov.uk/report/joint-report-on-publicly-available-hacking-tools, 2018.

[35] Amirreza Niakanlahiji, Jinpeng Wei, Md Rabbi Alam, Qingyang Wang, and Bei-Tseng Chu. Shadowmove: A stealthy lateral movement strategy. In USENIX Security Symposium, 2020.

[36] Novetta. Operation SMN: Axiom Threat Actor Group Report. http://www.novetta.com/wp-content/uploads/2014/11/Executive_Summary-Final_1.pdf, Nov 2014.

[37] The Council of Economic Advisors. The cost of malicious cyber activity to the u.s. economy. https://www.whitehouse.gov/wp-content/uploads/2018/03/The-Cost-of-Malicious-Cyber-Activity-to-the-U.S.-Economy.pdf, Mar 2018.

[38] Thomas Pasquier, Xueyuan Han, Mark Goldstein, Thomas Moyer, David Eyers, Margo Seltzer, and Jean Bacon. Practical whole-system provenance capture. In Symposium on Cloud Computing, 2017.

[39] Fred Plan, Nalani Fraser, Jacqueline O'Leary, Vincent Cannon, and Ben Read. APT40: Examining a china-nexus espionage actor. https://www.fireeye.com/blog/threat-research/2019/03/apt40-examining-a-china-nexus-espionage-actor.html, Mar 2019.

[40] Emilie Purvine, John R Johnson, and Chaomei Lo. A graph-based impact metric for mitigating lateral movement cyber attacks. In ACM Workshop on Automated Decision Making for Active Cyber Defense, 2016.

[41] Steve Reilly. Records: Energy department struck by cyber attacks. http://www.usatoday.com/story/news/2015/09/09/cyber-attacks-doe-energy/71929786/, Sep 2015.

[42] Andy Robbin, Rohan Vazarkar, and Will Schroeder. Bloodhound: Six degrees of domain admin. https://bloodhound.readthedocs.io/en/latest/index.html/, 2020.

[43] Hamza Saleem and Muhammad Naveed. SoK: Anatomy of Data Breaches. Proceedings on Privacy Enhancing Technologies, 2020.

[44] Hossein Siadati and Nasir Memon. Detecting structurally anomalous logins within enterprise networks. In ACM Conference on Computer and Communications Security (CCS), 2017.

[45] Counter Threat Unit Research Team. Bronze union: Cyberespionage persists despite disclosures. https://www.secureworks.com/research/bronze-union, Jun 2017.

[46] The MITRE Corporation. MITRE ATT&CK Matrix. https://attack.mitre.org/, 2020.

[47] TrendMicro. Lateral movement: How do threat actors move deeper into your network? http://about-threats.trendmicro.com/cloud-content/us/ent-primers/pdf/tlp_lateral_movement.pdf, 2013.

[48] Liam Tung. Ransomware: These sophisticated attacks are delivering 'devastating' payloads, warns microsoft. https://www.zdnet.com/article/ransomware-these-sophisticated-attacks-are-delivering-devastating-payloads-warns-microsoft/, Mar 2020.

[49] Wikipedia. Network segmentation. https://en.wikipedia.org/wiki/Network_segmentation, Sep 2019.

[50] Florian Wilkens, Steffen Haas, Dominik Kaaser, Peter Kling, and Mathias Fischer. Towards Efficient Reconstruction of Attacker Lateral Movement. In Conference on Availability, Reliability and Security (ARES), 2019.

[51] Kim Zetter. Inside the cunning, unprecedented hack of ukraine's power grid. https://www.wired.com/2016/03/inside-cunning-unprecedented-hack-ukraines-power-grid/, Mar 2016.