TopHat: Topology-based Host-Level Attribution for Multi-Stage Attacks in Enterprise Systems using Software Defined Networks

Subramaniyam Kannan, Paul Wood, Somali Chaterji, and Saurabh Bagchi

Purdue University, West Lafayette IN 47907, USA,
[kannan5,pwood,schaterj,sbagchi]@purdue.edu

Abstract. Multi-layer distributed systems, such as those found in corporate systems, are often the target of multi-stage attacks. Such attacks utilize multiple victim machines, in a series, to compromise a target asset deep inside the corporate network. Under such attacks, it is difficult to identify the upstream attacker's identity from a downstream victim machine because of the mixing of multiple network flows. This is known as the attribution problem in security domains. We present TopHat, a system that solves such attribution problems for multi-stage attacks. It does this by using moving target defense, *i.e.*, shuffling the assignment of clients to server replicas, which is achieved through software defined networking. As alerts are generated, TopHat maintains state about the level of risk for each network flow and progressively isolates the malicious flows. Using a simulation, we show that TopHat can identify single and multiple attackers in a variety of systems with different numbers of servers, layers, and clients.

Key words: multi-stage attacks, attack attribution, software defined network, moving target defense

1 Introduction

Multi-stage attacks (MSA) have plagued distributed system administrators for decades. In these attacks, multiple computers are used simultaneously to breach a particular target, and attackers often rely on a series of privilege escalation attacks to circumvent access controls protecting assets. One of the most challenging aspects of MSA comes as an attribution, mixing, or traceability problem [4]. Defenders wish to know what particular network traffic resulted in a privilege escalation, to prevent it in the future, but from a network perspective, the traffic output at each stage is not associated with any particular input. Consequently, defenders cannot distinguish legitimate from malicious network traffic, and identifying, patching, or disrupting vulnerabilities remains a daunting task. In this paper we present TopHat (TOPology-based Host-level ATtribution), a technique for identifying malicious users and their network traffic.

Multi-stage attacks operate on top of distributed systems where each distributed layer has different access privileges to sensitive business assets. An attacker must penetrate multiple layers to access some protected information, a *crown jewel*. As the attacker progresses, she generates some intrusion alerts due to some traffic with a malicious signature passing through intrusion detection

systems (IDS). These alerts, while useful for finding single stage attacks, are less useful in the MSA because the {source,destination} pairs are both machines inside of the distributed system, instead of an outsider and insider (as would be the case for an Internet-facing web server, for example). Consequently, there is no obvious relationship between alerts deep in the distributed system and the outsider, and this problem is referred to as the attribution, traceback or un-mixing problem [20, 3]. In this context, an attributable alert is one which identifies an external source directly, and an unattributable alert is one which identifies no source or identifies an internal or intermediate source, which cannot actually be the attacker.

Existing solutions [16, 13, 1, 5, 15, 2, 19] to the attribution problem have a few common shortfalls that TopHat addresses. First, solutions such as [16, 13, 1] rely on attack graphs to perform alert inferencing, where existing relationships between alerts are known via expert system knowledge. For example, an expert would claim that a port scanning alert deep in the distributed system follows from a wrong password alert in the Internet-facing layers. In practice, such relationships are complex, numerous, and difficult to derive. Furthermore, it is challenging to keep such information updated because systems are dynamic with new vulnerabilities being discovered, new digital assets being brought online, and new users being added. TopHat solves this issue without relying on attack graphs, thus providing a more general, robust, and adaptive solution to solving the attribution problem. Second, solutions such as [5, 15, 2, 19] rely on causal links between stages or layers of the MSA. For example, inside the system, it is known that input I_1 causes output O_1, and these relationships are logged and analyzed so that network traffic can be effectively tagged and tracked in the system. This approach relies on application support, however, to provide the causal links. TopHat does not rely on such information from the underlying application and can identify attackers without this causality link.

TopHat is a network-based solution to the attribution problem. We represent incident flows from external clients to alert sources in a directed acyclic graph, where each node in the graph models the mixing property of intermediate servers and softwares. Some of these flows are malicious, and they generate one or more alerts at various nodes (and at various depths) its path. For each alert, we generate and track partial attribution for all clients that can reach the alerted node as a stateful metric called *risk factor*, or equivalently, *risk value*. TopHat, taking into consideration current risks, adjusts the servers network flows will pass through, using a process called *shuffling* [9]. Through the shuffling process, TopHat isolates the suspect flows and keeps adjusting the risk factor. With a sufficient number of shuffles, the risk factor of the malicious flows exceeds a user-set threshold, *i.e.*, the cumulative partial attributions for an attacker reaches a level of complete attribution, and the attacker is identified[1].

[1] *Terminology clarification*: In this paper, we will use the term "attacker" synonymously with "attacking flow" or "malicious flow". Without loss of generality, we say for ease of exposition, that one client generates one network flow and thus there is a one-to-one correspondence. In parts of the paper, we use the term "client" for

In TopHat, we utilize detection techniques that resemble moving target defenses (MTD)[8], through our shuffling algorithms. Using software defined networks (SDN)[11], TopHat is able to manipulate or re-route the network flows to desired nodes that in turn helps in identifying the attacker in the distributed system. Using SDN-based load balancers[18], entering flows from external clients are mapped to any replica of an entry-level server in the distributed system. Then, whenever an alert is generated, by an IDS placed at a replica of any server in the system, some risk is attributed to all flows that are passing through that server replica. Using two different approaches corresponding to two different variants of TopHat, it tracks this risk and assigns clients so that the malicious flows have progressively increasing risk factor. Finally, those with risk values above a user-settable threshold can be isolated, blocked, or studied in a honey-pot.

Using this approach, TopHat is able to identify a single attacker in a system of 1000 clients and 3 servers at the entry layer in 6 shuffles, requiring 1000 seconds whenever the attacker repeats the attack for approximately every 150 seconds. In the same system with 4 attackers, all of the attackers are identified in 27 shuffles. We also show that the same system with 10 attackers, the shuffling mechanism requires the attacker to repeat their exploits over 1000 times, thus significantly increasing the attacker's efforts under TopHat. Finally, we demonstrate how TopHat impacts the legitimate clients, showing that after 3-4 shuffles a majority of clients can retain connectivity while the attacker is still identified.

The main contributions that we present in this paper are:

1. TopHat can attribute multi-stage attacks on a distributed system to a single external source, without relying on attack graphs or modifying the server softwares.
2. The MTD-style defense significantly increases attacker's effort, and can support identification of multiple simultaneous attackers.
3. TopHat can support high availability for legitimate clients while still identifying attackers in the system.

2 Background and Assumptions

2.1 System Model

TopHat is designed to protect a distributed system where servers exist at multiple layers, starting from an external-facing layer (layer 1) to moving progressively within the periphery of the system. A schematic is shown in Figure 1. Each layer comprises multiple server instances, or synonymously, server replicas, which are used for load balancing purposes. TopHat leverages these instances for the purpose of isolating suspect network flows, as we will detail in Section 4. As a running example, we consider a web-based e-Commerce system operated by a publicly traded company. Normal clients access a web front end, layer 1, that connects to a database back end, layer 2, to store orders, interact with inventory, and otherwise manage transactions. In layer 3, a corporate reporting server

"network flow" where such use will not lead to confusion. In places where we talk of attributing an attack and increasing risk factors, we use the term "client".

analyzes the database to create sales reports, track hot products, and manage inventory at a macro level. It interfaces with the database layer and stores reports on layer 4, the corporate file servers. Inside of the corporate file server is an upcoming earnings statement for the next quarter (the *crown jewel*), and its early release would allow for insider trading since the company's performance, relative to projections, can have a significant impact on stock prices. The attacker(s) wish to ex-filtrate the earnings report. More generally, the protected system may comprise an arbitrary number of layers and each layer may have none, one, or more server replicas.

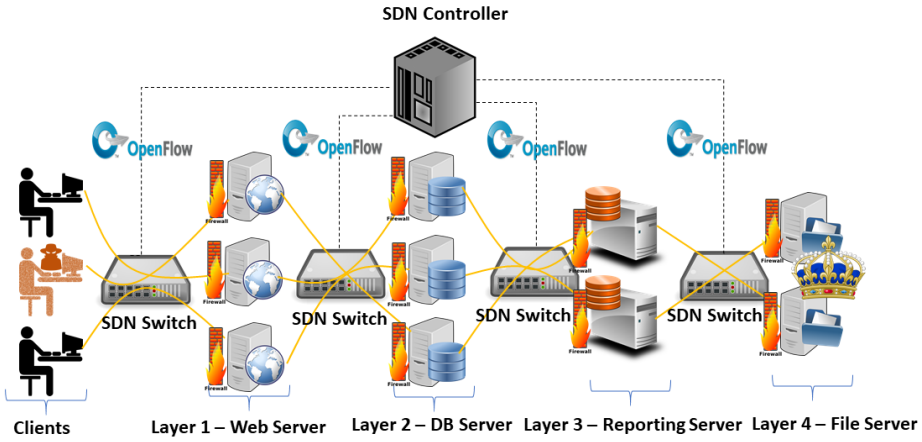


Fig. 1. A sample distributed system that can be protected by TopHat. It shows a multi-layered application and different network flows are intermixed at all layers of the system. Each layer comprises multiple server instances and the connectivity among layers is provided by SDN-enabled network switches.

2.2 Network Structure

The overall network structure for TopHat is shown in Fig. 1. A server type is each distinct kind of server—web server, file server, database server, etc. Each server type has multiple instances in our solution and each layer has servers of one specific kind. Each layer is connected to the next via an SDN capable switch and all these switches are controlled by a centralized SDN controller. The controller, through the switch, controls all layer-to-layer interaction paths. In SDN, as the control plane is separated from the data plane [6] it is possible to dynamically route the traffic to the desired servers by updating the flow table in each of the open flow switch. We use this functionality to route any specific network traffic flow through a specific set of server instances, as determined by the algorithm in TopHat.

Intrusion Detection Systems At its core, TopHat relies on intrusion detection systems to provide the alerts that drive its identification techniques. Each server itself has an IDS running (shown as a firewall in the Fig. 1) so that alerts

can be generated due to ongoing attacks. In today's deployments, IDS are often placed at the periphery, *i.e.*, at layer 1 of the network, and the alerts are often noisy (false positive) due to the wide variety of traffic that reaches the outer layers. TopHat utilizes IDS that are placed deep in the network, and the traffic at these layers is much more regulated due to the more tightly controlled nature of the applications at intermediate levels in the system. For example, a port scan (or its signature) at layer 1 is not necessarily an indication of an attack and is therefore not actionable. At layer 2, however, a port scan is almost certainly a strong alert because there is no legitimate reason for such traffic to exist at that layer in the network. TopHat uses IDS alerts from deep inside the network to regulate the risk value of any network flow. However, due to the stateful nature of its operation, it is capable of tolerating occasional false alerts from some IDS, or even repeated false alerts for a given flow from a small number of layers of the system. An acceptable alternative is to use a lesser number of IDS and/or to use correlation techniques to generate alerts for a specific flow [14]. In this paper, however, we make a simplifying assumption that each server instance has an IDS.

2.3 Legitimate Client Model

We define a legitimate client as a system user that has no malicious intent and is using the target application for its designed purpose. Whenever the client wishes to use the service, it makes a request to the outward facing service IP address. The SDN maintains a whitelist forwarding table in the Internet-facing switch, and the new client (identified by {source IP, source port}) is not in that list. This triggers a control action in the SDN switch—it contacts the controller and asks where to forward the client's network flow. This allows the SDN controller to assign the client to a particular front end server. Once this assignment is complete, the client continues to establish its application level connection, *e.g.*, perform a three-way TCP handshake and make web requests. Each client generates one network flow that touches each of the layers of the distributed application. Further details on this process are described in Section 4.5.

2.4 Attacker Model

The attacker begins as a normal client establishing a connection to the service. Once connected, the attacker looks for vulnerabilities in the outward facing layer 1. If the attacker is detected, then the client is blocked and the attacker must generate a new identity (through a proxy for a new IP for example). Once an exploit is found in layer 1, the attacker stages an attack on layer 2 from inside the periphery of the protected system, *i.e.*from layer 1. In moving from one layer to an inside layer, the attacker leverages elevated privileges that she has gained at the outer layer server. If an ongoing attack is flagged by an IDS at any layer, layer 2 and further inside, then TopHat is activated. If the attack is undetected, then it may proceed to the next layer, until reaching the *crown*

jewel. The attacker will persist until isolated, and then the attacker's exploit paths will be patched via external traffic analysis, such as through a honeypot. The assumption of persistence of the attack is crucial for TopHat to be effective. If the crown jewel is accessed through a flow that does not generate *any* IDS alert, then the attacker is successful and TopHat will never even be invoked. There may be multiple attackers present concurrently in the system. TopHat works by making a few assumptions about the nature of the multi-stage attacker:

- **Persistent Attacks (PA)** if a server is reset, or the attacker connects to a new server, then the attack must be repeated. Predecessor stages in an attack must be repeated if the attacker is re-connected to a new server.
- **Strong Alerts (SA)** the attacker will generate at least one strong alert during a MSA. The strong alert is an alert that with high certainty is known to be part of an attack (*e.g.*, brute force attacks, known exploit signatures, or other high priority[2] alerts). It is important to stress that only in the case of a strong alert is the algorithm of TopHat triggered. If no strong alert is input to TopHat, then the attacker will be successful in reaching the crown jewel.
- **Non-zero Exploit Time (T_X)** each stage of an attack will take non-zero time, with the time for an exploit to be successful (discovery to access transition) being a random variable.

3 Solution Overview

TopHat utilizes software defined networks (SDN) and intrusion detection systems (IDS) to monitor and attribute alerts to specific attackers. At its core, TopHat sits along side SDN controller software where it can observe the network flows and make decisions about changes to the network. It is installed as an application over a SDN open flow controller, such as an OpenDaylight Controller [12], and interfaces with IDS alerts generated throughout the distributed system. The algorithm then chooses which clients will be connected to which outward-facing servers, and which downstream servers are connected to which upstream servers in the distributed application.

TopHat's algorithm operates by maintaining a risk factor for each connected client and then modifying that risk factor whenever alerts are generated. As more alerts are generated, the attacker's stateful *risk factor* is increased until she can be discriminated from the other connected clients. Whenever an alert is generated, the risk is increased for all the flows that are passing through the alerting service. The clients are then shuffled based on their risk so that over time, the attacker ends up with the maximum risk. The risk factor is initialized to zero for all clients and this monotonically increases with alerts in the system, till the attacker is identified and isolated. Then the risk factors of all the clients that are found to be legitimate in retrospect are reduced (Risk Rebalancing as explained in Section 4.4). We classify our protocol as an instantiation of Moving Target

[2] `http://manual-snort-org.s3-website-us-east-1.amazonaws.com/node31.html#Snort_Default_Classifications` in Snort, rules are tagged with priority where "high" priority correlates with strong in our solution

Defense (MTD), though it is somewhat different from the traditional notion of MTD. Here we are moving the clients and the assignment of flows to servers, while in traditional MTD, the protected system is "moved", *i.e.*reconfigured [7].

3.1 TopHat's Intuition

Several challenges exist in protecting a distributed system that has the structure shown in Fig. 1. First, alerts generated at any layer $(i+1)$ $(i \geq 1)$ look as if they are coming from layer i, not from an external attacker. This argues against the simple solution of blocking flows from a particular source because that would create a Denial of Service—if a server in layer $(i+1)$ blocks a server in layer i, then the application stops working for all the clients connected to that particular server in layer i. TopHat overcomes this limitation by attributing an attack to all clients that are connected to the alerting server in layer i and then stopping the ongoing attack using the MTD approach. When an alert event happens, *all* of the clients are disconnected from the servers in layer 1 (for purposes of randomization), assigned to new servers, and the alerting server is refreshed to a clean state and restarted. TopHat then constantly tracks the attack history of each client with the help of the *risk factor* as we describe in detail in Section 4.1, so that the attacker is identified due to multiple alerts, which in turn is due to the persistence of the attack (as assumed in our attack model). The persistent attack property fundamentally allows TopHat to converge given a sufficient number of alerts.

3.2 Legitimate Client Impacts

The SDN-based shuffling in TopHat can have some negative impacts on legitimate client connections. First, whenever a shuffle involves a client, the client's connection is reset. This overhead cannot be avoided since the attackers and legitimate clients share the same network flow paths—a connection reset that disrupts an attacker's flow also disrupts the legitimate client's flow. Its impact can be mitigated, however, with state management approaches [17]. Second, when a server is being reset and restarted (to clear the infected status), the clients assigned to that server cannot function. This case can be minimized by using fast restart hardware or by keeping hot spares for the server instances. The rate of this exchange is related to the attacker's time to exploit (T_X), so that fast-moving attacks will generate many alerts, which will require many hot spares.

4 Detailed Design of TopHat

TopHat identifies attackers performing multi-stage attacks on a distributed system by algorithmically tracking risky behaviors of the attackers until the suspect flows are identified. In this section, we define two alternate algorithms in TopHat that achieve this goal, each with a different trade-off between the time to identify attacker(s) and the amount of disruption to clients.

4.1 Objective

Ultimately, TopHat is designed to identify attackers in the system. Practically, however, attacker identification in TopHat resembles a probabilistic function rather than an absolute measure because we do not have perfect internal causal relationships between alerts and sources. Therefore, we define identification as an event where a single client has the highest likelihood of being an attacker. In the case of multiple attackers, this process is repeated so that multiple identification events occur until all of the attackers are exhausted.

Alert Group Attribution In TopHat, there always exists a mapping between a server in any particular layer and the clients that, through any possible path, have access to that server. For example, if clients 1-5 are assigned to server S1 in layer L1, and S1/L1 is connected to S2/L2, then an alert sourced from S2/L2 will be attributed to *all* the clients 1-5. The relationship of how any given flow passes through the servers at the different layers is itself controlled by the SDN controller and thus this relationship is always known to our algorithm. Now we define a term *client group*. Consider that an alerting server has flows $F_1, F_2, ..., F_{N_G}$ going through it. By tracing each flow back to layer 1 servers, we can map each flow F_i to the client generating that flow C_i. The clients $C_1, C_2, ..., C_{N_G}$ form the client group here. Each such client has its stateful parameter, *risk factor*, increased by $\frac{1}{N_G}$, where N_G is the number of clients in that particular group. In the earlier example, each client would have its risk increased by 1/5.

We use the notation N_{AS} for the number of servers (or in complete terms, server instances) at the layer at which the alert is generated. If the alert is generated at layer 2, and there are 3 database server instances as in Fig. 1, then $N_{AS} = 3$. In our model for the protected system, there can be a different number of servers at different layers. This parameter is important for determining the convergence time of our algorithms, as we will see next.

Likelihood of a Client Being the Attacker We define the likelihood as follows:

$$P(C_i = A) = \frac{R(C_i)}{\sum R(C_j) \; \forall \; R(C_j) \geq R(C_i)} \tag{1}$$

where C_i is client i, $C_i = A$ is the indicator that C_i is an attacker, $R(C_i)$ is the risk factor of client i, and $\forall \; R(C_j) \geq R(C_i)$ implies that client j has a risk factor at least as large as client i, and $i = j$ is allowed. In this way, if a client has the highest risk factor of any client, then this probability value becomes 1.

Control and Convergence In TopHat, we control, through SDN controller rules, the assignment between clients and the layer 1 servers. Likewise we control the route each flow takes through servers at different layers. The full generality of the design space for TopHat allows for a flow passing through server A in layer i to be mapped to any server B in layer $i+1$. We call this configuration the *non stove-piped configuration* and this allows for the greatest flexibility to mix

and isolate the different flows as they flow through servers at the different layers. However, this increases the amount of state that needs to be maintained at the SDN controller - the mapping of the flow through each layer. For a simpler configuration option, we introduce the *stove-piped configuration* whereby the grouping of flows that are incident on a certain server at layer i is maintained at layer $i+1$, and this holds for all layers in the system. We find that the overhead of state maintenance at the SDN controller is fairly minimal for all but the largest of deployments and therefore the non stove-piped configuration is desired from a security standpoint.

The algorithm converges whenever:

$$\exists\ i,\ P(C_i = A) \geq \tau \tag{2}$$

The user-settable parameter τ allows TopHat to control the balance of false alarm and speed of identifying the attacking flows. A higher value of τ will mean fewer legitimate clients will be flagged but the convergence time will also increase. In the extreme, setting $\tau = 1$ will mean that only the flow with the highest risk factor will be designated as malicious. Section 4.6 discusses conditions where convergence may fail with multiple attackers present.

4.2 Uniform Assignment Algorithm ("Uniform")

The uniform assignment algorithm is responsible for assigning arriving client flows at layer 1 to different servers at layer 1. There are N_S assignment pools available, where N_S is the number of servers in layer 1. For each client i, an assignment is made: $A : C_i \rightarrow [1, N_S]$ such that the imbalance in risk between any two servers is minimized. At the beginning of the operation of the system, each client will have the same risk factor and so this will be a uniform random assignment. However, in subsequent mappings (which happen after an alert arrives at TopHat) the risk factors will be different and the mapping A will be a weighted random assignment, using the risk factors as the weights. The goal is to balance the aggregate risk at any of the servers in level 1. The assignment process proceeds as follows:

1. A client seeks an assignment, either when it is connecting to the protected system for the first time, or in response to a disconnection forced by TopHat.
2. The client is given the assignment to $[1, N_S]$ according to the assignment function A.
3. When a new alert is received, the assignments between clients and servers are reset, and all clients return to step 1 and re-assigned to new servers.

This algorithm effectively assigns clients such that there is a uniform aggregate risk assigned to any particular server. In this way, each attribution event reduces the set of ties ($N_T = |\forall\ R(C_j) \geq R(C_i)|$) to $\frac{N_T}{N_{AS}}$, where N_{AS} is the number of servers in the alert layer. For example, initially, if there are 100 clients and 4 alert groups, and every client has a risk of 1, then by Eq. 1, $P(C_i = A) = \frac{1}{100}\ \forall\ i$.

After an attribution event, given uniform assignment (each server having balanced risk of 25, thus 25 clients per server), then the likelihood for those 25 becomes $\frac{1.04}{1.04 \times 25}$ because the 25 clients that were attributed with risk have an additional 0.04 added. The size of the set $|R(C_j) \geq R(C_i)|$ is now $\frac{100}{4} = 25$, following the described reduction.

Convergence Using this algorithm, with N_{AS} servers at the alert generating layer and N_C static clients, a single attacker will be found after $N_R = \lceil \log_{N_{AS}} N_C \rceil$ alerts because the reduction of N_T by $\frac{1}{N_{AS}}$ resembles the height of a balanced tree with N_C leaves and N_{AS} branches at each alert. In a multi-attacker case, each attacker must be responsible for N_R alerts, where each alert independently causes a shuffle, in order to converge—as there are effectively N_A simultaneous risk trees being built. Here in one case, the attacker who generates alerts at a rate faster than the others will be identified first. In some cases, multiple attackers will be present in the same group, or multiple alerts will occur before step 3 is achieved in this algorithm, and these alerts will not count towards N_R. For example, if there are a total of 3 alert groups, and each group generates an alert before a shuffle event, then the risk of all the clients goes up (uniformly in this case) and that makes no progress toward convergence.

4.3 Low-Risk Isolation Algorithm ("LRA")

This variant of the algorithm extends the previous one by sheltering low risk clients into a *safe zone*. A safe zone is defined as a set of servers in layer 1 such that clients which are assigned to this set are not shuffled around by TopHat. Thus, these clients do not suffer from any disconnections and their risk factors do not change. Each alert/attribution event tells TopHat something about who may be the attacker, but it can also indicate who is *not* an attacker. In the uniform case, the legitimate clients are mixed in with the attackers, and this causes them to rise in risk, whenever they share a server with the malicious clients. It also dilutes the attribution power of a single attack since N_G remains near-constant. The Low Risk Assignment (LRA) variant avoids this issue by placing some portion of the clients with the lowest risk into a safe zone:

1. Clients are assigned as in the uniform risk case, except for clients that exist in a safe set S_S, initialized as empty.
2. After an attribution event a portion of the clients, $I_R, I_R \in (0, 1)$, is moved from the active set S_A to the safe set S_S. The $|S_A| \cdot I_R$ clients with the lowest risk are moved to S_S.
3. The assignment of the safe clients S_S is fixed to a particular server, and then the clients in the active set S_A are redistributed among the remaining $N_S - N_{safe}$ servers using the uniform risk approach.
4. In the event an alert is generated from any of the S_S clients, then the entire set of clients is moved back to the S_A set.

Using this approach for a single attacker, much less risk is assigned to the legitimate clients in the system. Additionally, they are provided an uninterrupted

connection path to the protected application, therefore decreasing the negative impacts from TopHat's assignment approach (Section 3.2).

Convergence This approach converges slightly slower than the uniform risk for the same number of servers N_S. Some servers at each layer are saved for handling the clients in the safe set S_S, thus the convergence for a single attacker is $N_R = \lceil \log_{N_{AS}-N_{safe}} N_C \rceil$ where N_{AS} is the number of servers at the alert layer and N_{safe} is the number of servers used for the safe zone. For multiple attackers, there is a chance that an attacker ends up in the S_S set of clients and causes a reset of the set back into the S_A set. This is advantageous to the attacker and slows down the speed of convergence. However, if the multiple attackers have very different times to exploit, then there is less likelihood of the above case because the malicious flows will rarely have low risk factors. The LRA approach is designed to keep trusted clients connected to the application continuously without suffering from any disconnections due to the shuffles of our algorithm.

4.4 Risk Rebalancing Approach ("RRB")

Once one of the attackers is identified by using any one of the above described algorithms, the risk factor of the remaining clients are updated using *Risk Rebalancing* (RRB) technique in order to speed up the convergence to identify the remaining attackers. Each alert attribution is stored in the SDN controller that contains the list of clients and the amount of risk factor attributed to each client due to that particular alert. Whenever an attacker is identified, the list of alerts is searched, and the set of alerts that involved the attacker are collected. The accumulated risk for each client due to each alert in that list is removed because of the insight that the alert is attributable to the now discovered attacker and not the other clients. Thus, legitimate clients have their risk lowered leading to faster identification of the other attackers.

4.5 End-to-end Workflow

We detail the end-to-end workflow of TopHat in the context of an SDN-based system:

1. **Initial:** The SDN switch at ingress node forwards each new client's request to the SDN controller as the flow table will be initially empty. TopHat, which is installed as an application over the SDN controller, stores the associated risk and the server allocated at each layer for all the clients.
2. **Server Assignment:** TopHat assigns each client to a particular server at layer 1 as described in Sections 4.2 and 4.3. Then the corresponding flow rules are installed at the SDN switch in layer 1 and subsequent layers. The initial risk factor of all the clients are set to 0. We denote by T_S the time for server assignment.

3. **Connection Establishment:** Each client establishes a connection with the servers at layer 1 using TCP 3-way handshake. At this point, all the clients except the attackers can access the servers in subsequent layers using their respective access privilege. (T_C: Time to establish connection).
4. **Attacker Exploration:** In order to get access to the subsequent layers, the attackers have to explore the layer 1 server for vulnerabilities and then exploit a vulnerability. Let T_x denote the time to exploit a server at a particular layer. T_x varies across different layers and across different attackers.
5. **Alert Generation:** The attacker continues to compromise the servers at subsequent layers until an IDS detects a malicious action (e.g., port scan, known CVE, etc.) or alert correlation from multiple IDS alerts generates a strong alert. Let T_A be the time to generate a strong alert.
6. **Connection Termination:** The strong alert is sent to the SDN controller, which initiates the shuffling by disconnecting the clients from the servers in layer 1 (except those in the safe set for LRA) and reassigning them.
7. **Risk Updation:** The risk factor of the clients are updated according to either the Uniform or the LRA scheme. Let T_{RA} be the time to update risk values.
8. **Attacker Identification:** After the risk updation, the probability of each client is calculated using Equation (1). The clients with a probability $P(C_i = A) \geq \tau$ are identified as attackers and isolated.
9. **Risk Rebalance:** After the attacker is identified, TopHat rebalances the risk factor of all the remaining clients (Section 4.4).
10. **Server Reset:** TopHat instructs the SDN controller to reset all the active servers in the network by broadcasting a control message which ensures that the attackers need to exploit it again, in order to re-initiate the MSA. Let T_R be time to reset a server.
11. **Connection Re-establishment:** All the clients including the attackers will re-initiate connections to the servers in layer 1 and the steps repeat.

4.6 Multiple Attackers

Multiple simultaneous attackers can be handled by TopHat, without any modification. We model multiple attackers as each having independent, random times to exploit (T_X), where a successful exploit results in an alert being generated. If one attacker is more aggressive (smaller T_X), then alerts will be generated due to this attacker and this attacker will be identified by TopHat before moving on to the next attacker. This essentially makes the process of identifying multiple attackers sequential. If on the other hand, there are multiple attackers with similar T_X values, then it will be a matter of chance which attacker gets identified first. But the risk factor of the other attackers will be retained in TopHat, thereby helping in the convergence time for the subsequent attackers.

False Positives and Mitigation It is possible for TopHat to generate false positives with multiple attackers present that have similar T_X. For example, if there are four clients C1-C4, of which C2 and C4 are malicious and two servers

S1 and S2. In the first round, C1 and C2 are assigned to S1 and C3 and C4 to S2. C2 alerts resulting in reshuffling. In the next round, C1 and C4 happen to be assigned to S1 and C2 and C3 to S2. Now C4 alerts and as a result, the legitimate client C1 is falsely flagged. This is a relatively rare occurrence and we show the false positive rate in Experiment 4 (it is below 5% even in the most pathological case).

5 Experimentation

5.1 Model System

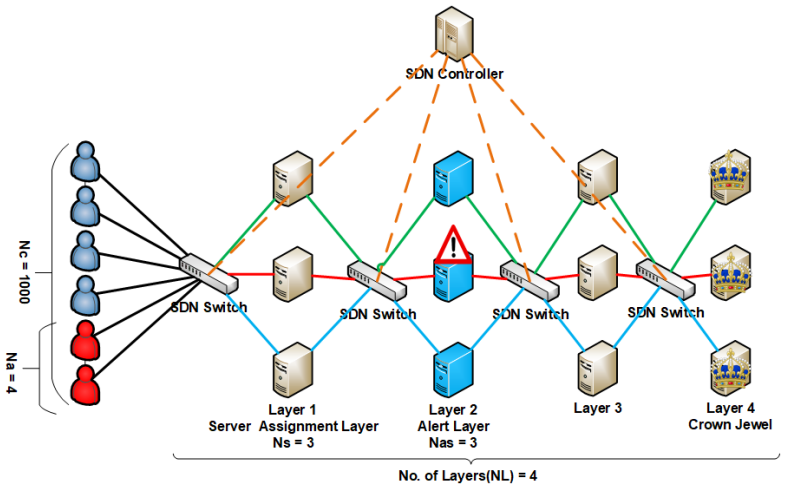


Fig. 2. SDN model system considered for the evaluation section.

Table 1. Default Network and Time Parameter Values.

Notation	Meaning	Default Value
N_C	No. of clients	1000
N_A	No. of attackers	4
N_S	No. of servers at layer 1	3
N_L	No. of layers	4
L_{alert}	Strong alert layer number	2
I_R	Ratio of clients moved from active set to safe set	0.25
T_S	Server allocation time	1 ms
T_C	Connection establishment time	30 ms
T_A	Alert generation time	1 ms
T_{RA}	Risk assignment + attacker identification time	1 ms
T_R	Server reset time	45 s [10]
T_X	Attacker Exploit Time	Normal Distribution

The figure 2 describes the default SDN Network and tables 1 and 2 shows the default values for the network parameters and the exploit time for 4 attackers respectively that are considered for the evaluation of the experiments described

Table 2. Default Attacker Exploit Time T_X for 4 Attackers.

Attacker No.	T_X at Layer 1		T_X at Layer 2	
	Mean (s)	Variance (s)	Mean (s)	Variance (s)
1	20	5	30	5
2	40	5	60	5
3	10	2	15	2
4	80	5	80	5

in the below sections . All the experiments are evaluated using the default values unless otherwise specified. As shown in figure 2, for the sake of simplicity we consider that each server in layer i has a stove piped connection or one-one connection (represented by different colors) to any server in layer i+1 in order to avoid mixing of network flows at later stages. The experiment 5 shows the convergence for non-stove piped case. In LRA approach the server 3 is considered to be safe server and the clients in active set S_A are shuffled between the server 1 and server 2.

Simulation Environment Along with TopHat, the SDN environment is re-modeled using the network and time parameters in C++[3]. Each event in the SDN environment is represented by a corresponding time component as described in section 4.5 and the network elements are given by the parameters in table 1. The clients are assigned to the available servers using uniform random distribution and the attacker's exploit time is modeled based on normal distribution as in table 2. For each attacker, the exploit time varies by mean across each layer and varies by variance across different iterations or shuffles. For some experiments, where multiple simulations can be aggregated, we take the median of 20 runs to provide data smoothness with respect to the random attack times.

Evaluation Parameters For simplification, we assume all the clients send requests to the servers at layer 1 at the same time. All the experiments described below are evaluated using the following parameters:

Experiment Time: This parameter indicates the time at which particular event like server assignment or alert generation happens.

Convergence Time: The time at which single attacker or all the attackers are found.

Probability of Attacker found (P_A) This is the probability of attacker being identified correctly as an attacker given by equation 1

Percentage of Failed Transactions (PFT) This parameter indicates the number of client disruptions during the time of attacker identification. It is a function of time given by

$$PFT(t) = \frac{No.of FailedTransactions}{TotalNo.of Transactions} \tag{3}$$

where we model client transactions as continuous time event for simplicity. We aggregate PFT across clients and all time to compute a **cumulative PFT** for

[3] `https://github.rcac.purdue.edu/DependableComputingSystemsLab/TopHat`

the purpose of comparing per-simulation metrics. Note that the PFT is per-client, and not all clients are disrupted simultaneously during a shuffle event in TopHat.

5.2 Experiment 1: Convergence over Time

The experiment 1 demonstrate TopHat's operation in the time domain for both single and multiple attackers. During each attack, the two primary metrics (PFT and P_A) are collected based on the experiment time at which an alert is generated. Default values are used for all parameters except N_A. Fig. 3 shows the results from our simulation, with the single attacker in Fig. 3a and two attackers in Fig. 3b. The results are explained in the next sections.

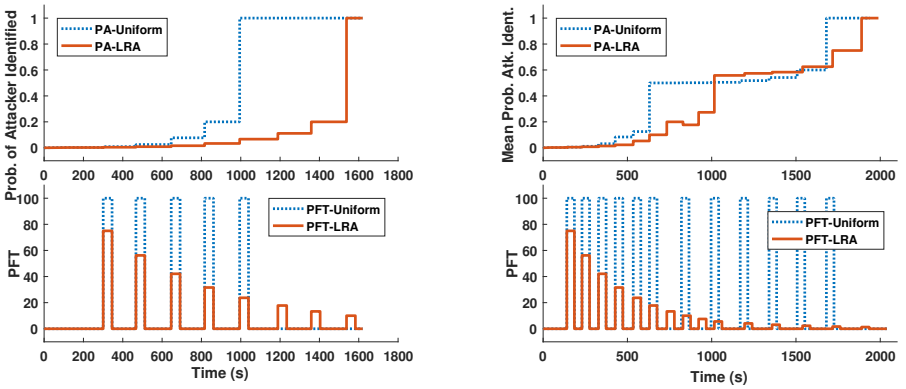
Convergence In case of single attacker, the convergence is given directly by P_A and for two attacker's case, it is given by average probability. At each alert generation, the probability is updated, and the value for the attacker increases as shown in the figures. The experiment time at which the P_A becomes is the time at which the attacker is found. The uniform algorithm converges more quickly in both cases primarily because it has 3 servers to use for risk attribution while LRA reserves a server for the safe pool and uses only 2 servers for risk attribution. The step function increases as the number of ties are broken, and the attacker is repeatedly involved in high-risk attribution events. We find that the algorithm does converge and single out the attacker in both cases given sufficient time, demonstrating our primary claim.

In the case of multiple attackers, one attacker has a faster exploit time than the other. Since shuffles occur on the fast attacker's alert, the slow attacker is statistically unlikely to ever generate an alert until the fast attacker has been disrupted. This causes a time-domain crowding of alerts early in the simulation until the first attacker is identified, and then the alerts become more spaced out as opposed to an independent case where the alerts would be interleaved. Upon close inspection, one of the LRA's potential weaknesses can be seen in that it is using more shuffles to identify the attacker in the two attacker case. Furthermore, because the slow attacker has low risk, she can be placed in the safe zone, and it is more likely that a slow attacker can generate an alert inside of the safe area–something that does not happen in this experiment, but will in a later experiment. Of note, the dip in probability around 750 seconds for LRA is due to the metric being a mean: the slow attacker's probability goes down offsetting the rise in the fast attacker's probability.

PFT The PFT shows how clients are impacted through time. In all cases, the width of the PFT bar represents the reset time for cleaning impacted servers in the system T_R. For the uniform algorithm, all clients are re-assigned and all servers on the attack path are cleaned, resulting in outages for all of the clients, hence the peak is always at 100. In the LRA case, only those clients remaining in the active set are impacted for each attack. This results in a decaying PFT over time as the low risk clients are assigned to the safe server at the rate I_R.

Consequently, system operators have a choice between faster convergence and attacker identification (the Uniform variant) or slower convergence with better client access (the LRA variant).

For multiple attackers, in the Uniform case the PFT follows the single attacker profile, but it is repeated for the second attacker with a higher width due to T_X. For the LRA case, since less shuffling servers are available, it takes more alerts to converge and thus more shuffles, and more period of high PFT. Of note, however, is that the PFT is never reset–that is the pool of safe clients never generates an alert.



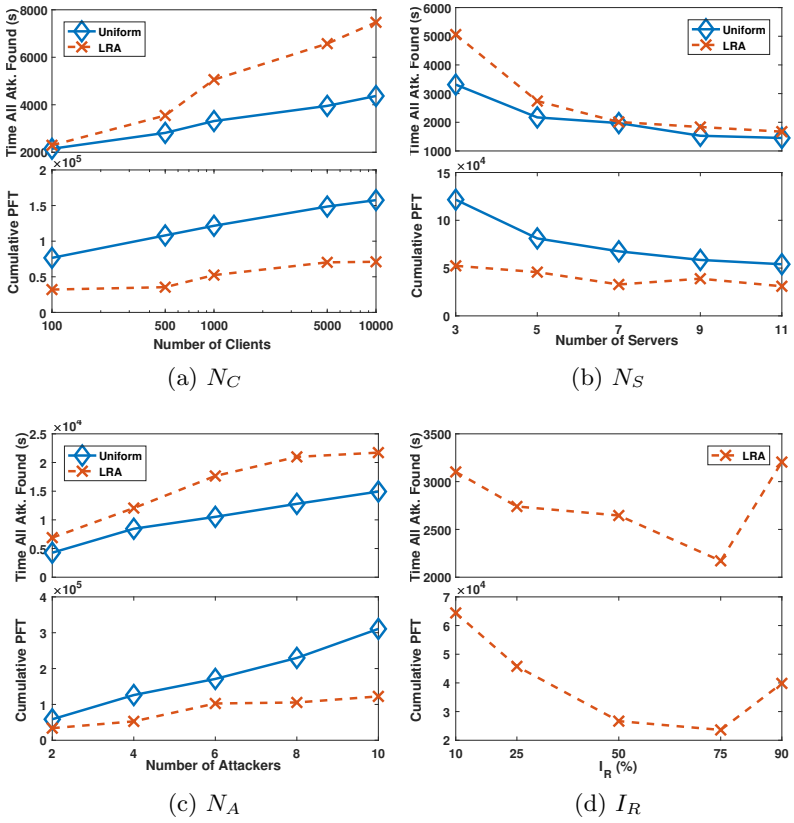
(a) Single Attacker (b) Two Attackers

Fig. 3. The convergence of TopHat is shown for the single attacker and two attacker cases with both the uniform and low-risk assignment (LRA) algorithms. The LRA converges more slowly (top) but has better access for the legitimate clients (bottom).

5.3 Experiment 2: Convergence vs. Parameters

This experiment explores the convergence properties of both the Uniform and Low Risk Assessment (LRA) approaches to attacker identification. We explore four parameters: the number of clients N_C, servers at the alert layer N_S, attackers N_A, and the LRA's active-to-safe movement ratio I_R. All of these results are in Fig. 4 and described in the following sections.

Number of Clients This experiment, show in Fig. 4a, increases the number of clients connected to a system with the default number of servers and attackers. The x-axis is show in log scale, and the time to find all attackers is linear in the convergence time for the uniform case, matching the expectation from Section 4.2. For the LRA, it is roughly linear, but it suffers from placing attackers into the safe pool of clients. In the PFT metric, the LRA performs much better due to this safe pool, as expected. This experiment confirms TopHat's scalability with the number of clients.



(a) N_C (b) N_S

(c) N_A (d) I_R

Fig. 4. Convergence time and the client's cumulative PFT is shown for four parameters in our simulation.

Number of Servers Adding additional servers to TopHat, thus increasing N_{AS}, improves convergence speed incrementally by $\frac{log(N_S+1)}{log(N_S)}$. This is because additional servers provides finer granularity in the alert attribution phase, thus singling out attackers more quickly. Fig. 4b shows this convergence trend for the two algorithms. The same general trends from earlier experiments hold, but LRA's advantage in PFT begins to disappear with a large number of shuffling servers and only a single safe server.

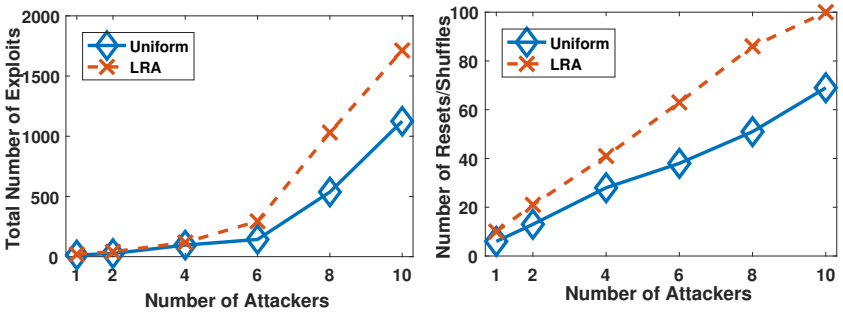
Number of Attackers As the number of attackers increases, the number of alerts required to identify the attacker increases linearly. Fig. 4c shows the trend with N_A in the range of 2 to 10. At higher number of attackers, it takes longer to converge as expected in the uniform case, but it also impacts the LRA super-linearly because it increases the probability that a slow, low-risk attacker will enter the safe pool. Even with this risk, the LRA is still able to outperform the Uniform algorithm in the PFT metric.

LRA Movement Ratio This experiment, as shown in Fig. 4d, only applies to the LRA algorithm. The I_R parameter controls how many clients are relocated

from the active set to the safe set after each shuffle. If this ratio is too low, then the convergence speed and PFT will be the same as uniform but with 1 less server in the shuffle set. If this ratio is too high, then up to half of the shuffles will be wasted on safe server alerts—the alert will come from the server that has all of the clients connected to it, and no useful attribution can take place. At an ideal ratio, the attacker has limited chance of being moved to the safe server, which in this case is 75%. In future work, the ratio can be modulated based on an estimate for the number of attackers in the system.

5.4 Experiment 3: Attacker Effort

In this experiment, we demonstrate how TopHat, by utilizing MTD, is able to increase the total attack effort that must be expended to compromise the protected system. We measure attacker effort as the number of times a server must be compromised, at any layer, by any attacker. This includes the effort spent exploiting servers that have been reset. We also measure the number of shuffles or alerts generated in the system, and this metric covers the number of trials an attacker has at penetrating a system for which the exploit is not known.



(a) Total effort for all attackers (b) Number of attempts at alert stage

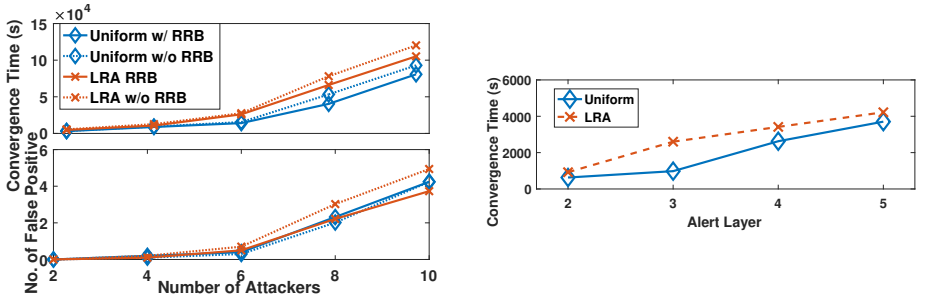
Fig. 5. Each attacker in the system repeatedly compromises various servers in the system. These figures demonstrate how many total compromises occurred before the attacker was found and show the number of alerts generated during the attacker's exploration phase.

Fig. 5 shows the effort in these two metrics. In Fig. 5a, the total exploits goes up with the number of attackers. This process is not linear, however, because many attackers will be reset even when they do not generate an alert themselves due to the moving target nature of TopHat. Each attacker may penetrate layer 1 and be shuffled before making an attempt on layer 2, for example. Consequently, TopHat is able to make it much more difficult to attack the system when multiple attackers are present, even if the attacker identification takes some time. In Fig. 5b, the total number of resets are shown. This scales roughly linearly with the number of attackers because there is a lower limit to this number until the attackers can be found, as described in Section 4.2. Of note here, however, is that there are a limited number of exploit attempts allowed at layer 2 before the

attackers are identified and a layer 1 patch can be created. These 5-100 alerts will attribute the attacker, and upstream compromises (at layer 1) can be patched as a result, a key benefit of TopHat.

5.5 Experiment 4: Effect of Risk Re-balancing

For this experiment, we evaluate the impact of the risk re-balancing (RRB) technique (Section 4.4) on the convergence time and the false positives. We stress the system by having multiple attackers with the same distribution for T_X. Without RRB, when an attacker is identified, the risk for *all* other clients is reset to zero. With RRB, when an attacker is identified, only the legitimate clients that had been mixed in with the identified attacker have their risk reduced, not reset to zero.



(a) Impact of maintaining state (w/ RRB) on convergence and false positive rates.

(b) As the alert becomes deeper in the system, it requires more shuffles and thus increases the time to converge.

Fig. 6. Experiments 4 (left) and 5 (right)

Fig. 6a shows the impact of RRB on both the Uniform and the LRA algorithms. In both cases, the use of the RRB speeds up convergence as expected. The number of false positives is higher for LRA. This is because the placement of many clients in the safe zone and subsequent alerts from that zone can degrade the process of identification of the attackers. TopHat is still able to provide low false positive rates (less than 0.5%) for small numbers of attackers relative to the total number of clients (10), even in this challenging scenario of similarly aggressive attackers.

5.6 Experiment 5: Effect of Number of Server Replicas

For this experiment, we use a system with 5 layers having [5, 4, 3, 2, 2] replicas in the layers, starting from layer 1. The inter-layer connections are uniformly balanced as much as possible. We evaluate the impact of alert depth on the risk attribution algorithm. Fig. 6b shows the impact of the alert layer on the convergence speed of both algorithms. The number of replicas decreases as one goes further inside the system. This is not uncommon because the number of

requests that touch servers deep inside the enterprise typically decrease. We expect that alerts deep in the system will provide less discriminating information about the attackers because the shuffling can occur with coarser granularity, thus lumping more number of clients (legitimate with a few attacking) together on the same server. In the case of LRA, the safe zone is on a single stove-piped layer while the other shuffling servers are all connected into the multi-layer system. As the layer deepens, it is similar to reducing N_S because the size of the alert group increases and the number of groups N_G decreases. Therefore there is a logarithmic increase in the convergence time as the depth of the alert layer increases.

6 Discussion and Future Work

In this paper, we presented a solution to the attribution problem using moving target style defense through shuffling. This approach provides robust mitigation for a variety of attacks for which strong IDS alerts exist, which is a cornerstone assumption for TopHat. This assumption can be relaxed with the use of a more probabilistic risk assessment model that adapts risk to the quality of the alert, so that alerts based on weak signatures can still be useful in identifying attackers. TopHat also relies on restricted intra-layer communications which limits its applicability to some applications. In some attack models, an attacker could move laterally between one server and another if such a channel exists. We can extend TopHat by requiring network IDS placement between servers in the same layer, which will enable detection of such lateral movement. Finally, in TopHat, we have not provided any elastic replica management. In cloud environments, it may be useful to scale the number of servers based on the risk and alerts generated in the system to more quickly find the adversary and restore client connectivity. In future work, we will analyze when to expand or contract the replica set.

7 Conclusion

In this paper, we presented TopHat, a solution to the problem of attributing an alert to an attacker in a multi-layered system. The problem is challenging due to the mixing of multiple flows at servers inside the periphery of the system. TopHat utilizes moving target defense techniques, namely shuffling, implemented on top of a software defined network infrastructure. We provided two algorithms for shuffling, one that focuses on convergence speed and another that focuses on improving client connectivity during attacks. Further, we show that TopHat increases the attackers effort by requiring multiple re-exploiting of the target systems. We evaluate TopHat using the metrics of time to detect and isolate the attackers and the impact on the legitimate clients in the system. Using our system, network administrators can begin to attribute alerts and attacks, to external flows so that they may be blocked or studied for further defense improvement.

References

1. F. Alserhani, M. Akhlaq, I. U. Awan, A. J. Cullen, and P. Mirchandani. Mars: multi-stage attack recognition system. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 753–759. IEEE, 2010.
2. T. Baba and S. Matsuda. Tracing network attacks to their sources. *IEEE Internet Computing*, 6(2):20–26, 2002.
3. D. D. Clark and S. Landau. The problem isn't attribution: it's multi-stage attacks. In *Proceedings of the Re-architecting the Internet Workshop*, page 11. ACM, 2010.
4. D. D. Clark and S. Landau. Untangling attribution. *Harv. Nat'l Sec. J.*, 2:323, 2011.
5. J. Dawkins and J. Hale. A systematic approach to multi-stage network attack analysis. In *Information Assurance Workshop, 2004. Proceedings. Second IEEE International*, pages 48–56. IEEE, 2004.
6. N. Feamster, J. Rexford, and E. Zegura. The road to sdn: an intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review*, 44(2):87–98, 2014.
7. J. H. Jafarian, E. Al-Shaer, and Q. Duan. Openflow random host mutation: transparent moving target defense using software defined networking. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 127–132. ACM, 2012.
8. P. Kampanakis, H. Perros, and T. Beyene. Sdn-based solutions for moving target defense network protection. In *A World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on*, pages 1–6. IEEE, 2014.
9. D. C. MacFarland and C. A. Shue. The sdn shuffle: creating a moving-target defense using host-based software-defined networking. In *Proceedings of the Second ACM Workshop on Moving Target Defense*, pages 37–41. ACM, 2015.
10. M. Mao and M. Humphrey. A performance study on the vm startup time in the cloud. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 423–430. IEEE, 2012.
11. N. McKeown. Software-defined networking. *INFOCOM keynote talk*, 17(2):30–32, 2009.
12. J. Medved, R. Varga, A. Tkacik, and K. Gray. Opendaylight: Towards a model-driven sdn controller architecture. In *A World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on*, pages 1–6. IEEE, 2014.
13. G. Modelo-Howard, J. Sweval, and S. Bagchi. Secure configuration of intrusion detection sensors for changing enterprise systems. In *International Conference on Security and Privacy in Communication Systems (SecureComm)*, pages 39–58. Springer, 2011.
14. S. Noel, E. Robertson, and S. Jajodia. Correlating intrusion events and building attack scenarios through attack graph distances. In *Computer Security Applications Conference, 2004. 20th Annual*, pages 350–359. IEEE, 2004.
15. S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical network support for ip traceback. In *ACM SIGCOMM Computer Communication Review*, volume 30-4, pages 295–306. ACM, 2000.
16. W. T. Strayer, C. E. Jones, B. I. Schwartz, J. Mikkelson, and C. Livadas. Architecture for multi-stage network attack traceback. In *Local Computer Networks, 2005. 30th Anniversary. The IEEE Conference on*, pages 8–pp. IEEE, 2005.

17. F. Sultan, K. Srinivasan, D. Iyer, and L. Iftode. Migratory tcp: Connection migration for service continuity in the internet. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 469–470. IEEE, 2002.
18. R. Wang, D. Butnariu, J. Rexford, et al. Openflow-based server load balancing gone wild. *Hot-ICE*, 11:12–12, 2011.
19. Z. Xu, Z. Wu, Z. Li, K. Jee, J. Rhee, X. Xiao, F. Xu, H. Wang, and G. Jiang. High fidelity data reduction for big data security dependency analyses. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 504–516. ACM, 2016.
20. Y. Zhu and R. Bettati. Unmixing mix traffic. In *International Workshop on Privacy Enhancing Technologies*, pages 110–127. Springer, 2005.