Evading stepping stone detection by using enough chaff perturbations

No Author Given

No Institute Given

Abstract. ....
....
....
....

1 Introduction

The problem of stepping-stones detection (SSD) has been studied for over 20 years, yet the body of literature fails at providing an informative overview of the detection capabilities of current methods. In this paper, we set out to do just that by evaluating and comparing a number of selected state-of-the-art approaches on a new and independently generated dataset.

In a stepping-stone attack, malicious commands are relayed via a chain of compromised hosts, called stepping-stones, in order to access restricted resources and reduce the chance of being traced back. Real-world attacks using stepping-stone chains include the Operation Aurora [24], the Operation Night Dragon [2], the Black Energy [18] on the Ukrainian powergrid, and the MEDJACK [6] attack where medical devices were used as stepping-stones. The infamous Archimedes tool [3] used by the CIA leverages stepping-stone chains to reach the LAN of target hosts, while the emergence of IoT-devices further increases the attack surface via connected printers, cameras, or even thermostats insert citation from Infosec Resources, 2019.

The detection of interactive stepping-stones is challenging due to various reasons. Attackers are not constrained to specific proxy techniques and can obfuscate relayed traffic with a number of evasion tactics. Packet-based methods are computationally expensive, with scalability and false-positives being a problem for large networks.

Like many intrusion attacks, stepping-stones are rare and there exists no public data representing real stepping-stone behaviour, and researchers have to rely on synthetic data. Some attempts have been made to create publicly available stepping-stone testbeds, yet most researchers evaluate their SSD methods on self-provided data. The underlying traffic generation implementations often vary significantly, which makes a direct comparison of the achieved results impossible.

In this work, we provide the following contributions:

1. We describe a framework to generate data that represents realistic stepping-stone data without bias to particular detection mechanisms. Our framework is scalable and capable of generating sufficient variety in terms of network settings and conducted activity.
2. We release a large and comprehensive dataset suitable for the training of machine-learning-based methods and in-depth performance evaluation. To our knowledge, this is the first public SSD dataset.
3. We re-implemented eight SSD methods that represent the current state-of-the-art and provide a fair evaluation of their capabilities in a number of settings.
4. Our evaluation shows that while most methods can accurately detect command propagation, detection rates plummet when appropriate chaff is inserted. This result disproves the claims made for multiple methods that their detection rates are robust against chaff perturbations.

The rest of the paper is organised as following: Section 1 provides an introduction and background to the problem of stepping-stone detection. Section 2 discusses the particular design of the data generation framework. Section 3 presents the dataset arrangement in terms of background and attack data and discusses evaluation methods. Section 4 discusses the selection process, properties, and implementation of the eight SSD methods that we implemented for evaluation. Section 5 discusses the results achieved by the implemented methods on the given data. Section 6 discussing related work.

1.1 Background

Stepping-stones were first conceptualised by Staniford-Chen and Heberlein in 1995 [23]. In an interactive stepping-stone attack, an attacker located at the origin host, which we call *host O*, sends commands to and awaits their response from a target, *host T*. The commands and responses are proxied via a chain of one or more intermediary stepping-stone hosts, which we call *host* S_1, ..., S_N, such as depicted in Fig. 1. Once a host S_i is brought under control, it can be turned into a stepping-stone with simple tools and steps. Some of the most common set-ups are port forwarding via SSH-tunnels, setting up a backpipe with NetCat, or using metasploit to set up a SOCKS proxy [12].

Typically, host O is located outside the targeted network, and the attack is started by compromising an initial foothold S_i inside the network, from wich the attacker tries to move deeper into the network to reach the most valuable target T [1]. The attacker may use one or more compromised intermediary hosts $S_1, \ldots, S_{i-1}$ outside the target network at different geographical locations in order to make tracing back the attack impossible.

Stepping-stone detection (SSD) is a process of observing all incoming and outgoing connections on a particular host h_i and determining whether it is used to relay commands. This is generally done with no prior information about any other stepping-stone hosts $S_1, \ldots S_N$ or the endpoints O and T. Once h_i is known

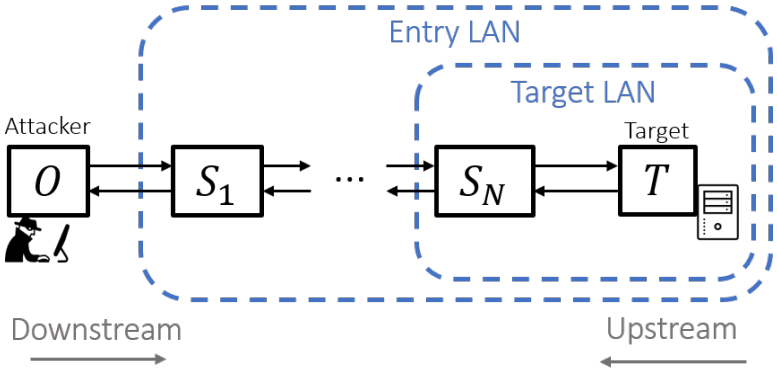

Fig. 1. Depiction of an exemplary stepping-stone chain.

to be a stepping-stone host, further members of $\{S_1, \ldots S_N\}$ can be traced back from h_i.

A popular approach to SSD is to compare connections pairwise to identify whether they carry the same information. A wide-spread way to achieve this is via *watermarking*, an active SSD mechanism. A watermark is a unique binary string, which is usually embedded into a connection by altering the packet inter-arrival times. Decoding the watermark involves capturing candidate flows that might match the water-marked flow and looking for the bits in the flow characteristics. Popular passive SSD methods are based on *packet-correlation* and attempt to identify packets that appear in both connections. Since connections can be encrypted, this is often done by comparing sequences of interarrival times, packet sizes, and the overall number of packets in each connection.

Another prominent approach to detect stepping stones is based on *round-trip-times* (RTTs). The RTT of a connection is the time it takes for a packet to be sent to the receiver plus the time it takes for an acknowledgement of that packet to be received. Since information is relayed over one or more hosts in a stepping stone chain, this has an effect on the overall RTTs which can be observed within individual connections in the chain.

Anomaly-based methods aim to detect the insertion of time delays and chaff perturbations in a connection as deviations from typical TCP-behaviour to indicate of suspicious behaviour.

To avoid detection, several evasive flow transformation techniques exist that aim at decreasing observable correlation between two connections in a chain.

- **Packet transfer delays/drops**: An attacker can choose to apply artificial delays to forwarded packets, or drop certain packets to cause retransmission, in order to creates temporal disparity between connections. Researchers often assume the existance of a maximum tolerable delay [11].
- **Chaff perturbations**: Chaff packets are packets without meaningful content that are added to individual connections in a chain without being forwarded, and cannot be distinguished from real attack packets by a third

party. Adding chaff perturbations alters the overall amount of traffic in a connection and can be used to shape the connection profile towards other traffic types.
- **Repacketisation**: Repacketisation is the practice of combining closely adjacent packets into a larger packet, splitting a packet into multiple smaller packets, or altering the packet content to change observed packet sizes and numbers.
- **Flow splitting/merging**: Since SSD is mostly done on singular connections, an attacker can split the flow of packets to two or more connections and merge them at the target.

In our evaluation, we set out to understand the effect of different evasive methods on detection rates.

2 Data generation setting

Our goal is to simulate data that reflects the different aspects of interactive stepping-stone behaviour in a reproducible manner. For a fair and thorough evaluation, we want to cover different settings and interactions to incorporate enough variation in the data to highlight strengths and weaknesses of different SSD methods.

2.1 Containerisation

In order to consider all these factors, we rely on the virtualisation of networks using containerisation. A container is a standard unit of software that runs standalone in an isolated user space in order to remove platform dependencies. Compared to virtual machines, containers are more lightweight, always start from an identical state, and are highly specialized in their purpose with each container running only a specific piece of software or application.

The advantages of containers over virtual machines enable us to easily control, modify, repeat, and scale a network of hosts while emulating different network settings. The use of containerisation for this project is a continuation of a traffic generation paradigm designed for machine learning, which was originally introduced in [citation currently blinded]. We build our traffic generation framework on the popular containerisation platform *Docker* citation needed?.

2.2 Simulating stepping stones with SSH-tunnels and Docker

We want to capture data not only from one interaction in a fixed stepping-stone chain, but from many interactions and chains with different settings. For that, we run multiple simulations, with each simulation establishing a stepping-stone chain and controlling the interactions between host O and host T. A simulation and the corresponding traffic capture is in a capture-script.

A simulation begins with the start-up of the necessary containers and ends with their takedown. We represent host O, host T, and host $S_1, \ldots, S_n$ with SSH-daemon containers. To establish a connection chain, we connect these containers via SSH-tunnels, with the first tunnel forwarding the required port from host O to host S_1, which is then forwarded to host S_2 by the second tunnel etc. As mentioned by Gordon Fraser [12], this is one of the most common pivoting methods for attackers. Fig. 2 depicts a packet transfer via an exemplary chain.

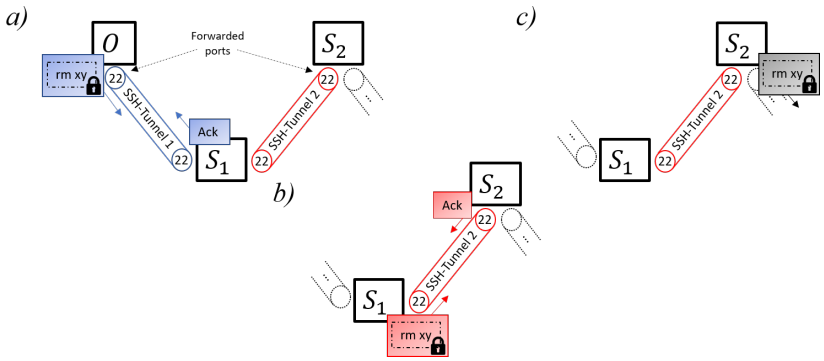

Fig. 2. Depiction of the way a command is packetised, encrypted, and travels through the different stages of the stepping-stone chain via SSH-tunnels.

Traffic is captured both at host T and host S_n, which acts as the final stepping-stone in the chain.

Simulating interactive SSH-traffic In order to generate enough data instances representing interactive stepping stone behaviour, we automatised the communication between host O and host T. For each simulation, we generate a script which passes SSH-commands from host O to host T.

For script-based session creation, several measures have been taken to make them realistic. First, each session tries to mimic a real user's action. We compiled a command database which consists of common commands and their usage frequency, similar to [29].Commands are drawn randomly according to their usage frequency and concatenated to a script. Commands can either be atomic, such as "ls-la" or "pwd", or compound. Compound commands need additional input such as the directory and name of a specific file that is transferred, or input text to fill a file. The content and sometimes length of these inputs as well as transferred files are randomised appropriately when a compound command is drawn. Scripts are of varying length and end once the *End*-command is drawn from the command catalogue.

To simulate human behaviour that is reacting to the response from host T, all commands are separating by *sleep*-commands for time t, which is drawn from

a truncated Pareto-distribution. Paxson et al. [21] have shown that interpacket spacings corresponding to typing and "think time" pauses are well described by Pareto distributions with a shape parameter $\alpha \approx 1.0$. We use a truncated distribution capped at 10s to avoid infinite waiting times since we already included a mechanism to end a script.

Simulating different network settings Hosts in a stepping-stone chains can be separated by varying distances. Some may sit in the same LAN, while others may communicate via the Internet from distant geographical locations. The type of separation between two hosts influences the round-trip-time, bandwidth, and network reliability.

Docker communication takes place over virtual bridge networks, so the throughput is far higher and more reliable than in real-world networks. To retard the quality of the Docker network to realistic levels, we rely on the emulation tool Netem. Netem is a Linux command line tool that allows users to artificially simulate network conditions such as high latency, low bandwidth, or packet corruption/drop in a flexible manner [14].

We apply Netem commands to the network interface of each container, which adds correlated delays to incoming and outgoing packets that are drawn from a normal distribution with mean μ, variance σ^2, and correlation ρ_1. We furthermore apply correlated packet loss and corruption drawn from a binomial distribution with probability p and correlation ρ_2. Lastly, we apply an overall limit B on the bandwidth of container network interfaces.

To allow for different types of host separation, we set the network settings and bandwidth limit for each host container individually before each simulation, and draw each of the given parameters from a suitable distribution. This allows for some hosts to experience very fast and reliable communication while others experience more congested communication. We store the set parameters along with the collected traffic for each simulation to include the effect of network congestion in the evaluation.

2.3 Evasive tactics

Adding transfer delays To increase detection difficulty, we add random transfer delays to individual packets forwarded by stepping stone hosts. This method, often called *jittering*, can destroy time-based watermarks in packet flows and help decrease observable correlation between two connections.

We add transfer delays to forward packets again by using NetEm. We draw delays for departing packets on a hosts from a uniform distribution, covering the interval $[0, \delta_D]$. This particular choice has been suggested by Padhye et al. [20] in order to mimic the interarrival distributions of streaming services. The value of δ_D is fixed before each simulation and can be varied to allow for different degrees of packet jittering.

As pointed out by Donoho et al. [11], excessively long delays often lead to difficulties in the TCP-protocol due to the significant increase of packet reordering

and response time-outs. We set the maximum value for δ_D at 1500 ms. As we will see in Section 5, this is enough to render watermarking methods obsolete while flow decorrelation can be achieved more effectively by using chaff perturbations.

Adding chaff perturbation In addition to transfer delays, we insert chaff packets to individual connections in the chain to increase detection difficulty. These chaff packets do not contain actual information and act as noise to decorrelate individual connections in the chain. To add and filter packets in a connection, we open additional ports in each SSH-tunnel that are however not forwarded through the entire chain. This means that each chaff packet only appears in the connection it was inserted to, making the chaff perturbation in two different connections independent of each other. We then use a NetCat client containers to send and receive packets on the additional ports in both directions.

Again, Padhye et al. [20] suggest to generate chaff in a way that mimics the flow characteristics of streaming services in order to both spread the added perturbations evenly across the connection and increase the difficulty of detecting the perturbation itself. Therefore, the size of the transferred data is drawn from a truncated Lognormal-distribution with mean μ_C to mimic video-streaming traffic. Similarly, packets are sent in intervals of random length δ_c drawn from a uniform distribution that covers the interval $[\delta_c/2, \delta_c]$ to mimic a relatively constant packet flow, typical for video-streaming. By adjusting δ_C, we can control the amount of chaff sent.

Repacketisation One benefit of using SSH-tunnels from an attackers perspective is that packets are not simply forwarded, but a tunnel acts as an independent encrypted TCP-connection along with independent packet confirmations and repacketisation.

Flow splitting/merging None of the methods we encountered are currently capable of dealing with flow splitting or merging. We therefore did not implement this tactic as the results would be uninformative.

Fig. 3 depicts the interplay of the different containers in our simulation.

3 Evaluation data

The quality of both the stepping-stone and background data is crucial for a fair evaluation.

First of all, we want to look at a variety of attack scenarios to highlight the strengths and weaknesses of different approaches. We created three main attack datasets that contain different forms and amounts of evasive behaviour, and a smaller dataset to highlight the influence of different network settings and chain lengths. To present a valuable false positive test, we provide three datasets with benign background traffic. The first contains general real-world traffic, while the second and third contain benign data that bears similar traffic characteristics as

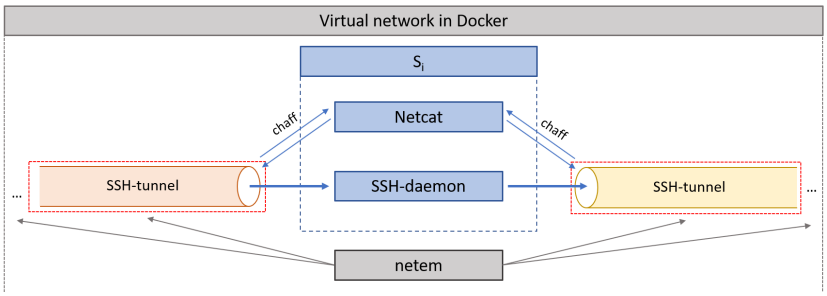

Fig. 3. Depiction the simulation setup for each host in the chain.

the attack data. Furthermore, we need to generate sufficient data to effectively train some of the included methods.

We now describe our data assembly in more detail.

3.1 Stepping-stone data

We generate stepping-stone data following the steps described in Section 2. We create our main datasets using a chain of four stepping-stones S_1, S_2, S_3, and S_4. We subdivide into three datasets according to the two discussed evasion tactics, transfer delays and chaff perturbations. We first capture data without either of these methods in **dataset A**. We then capture data once with added transfer delays with varying δ_D to control delays in **dataset B**, and once with added chaff perturbations of varying δ_C in **dataset C**. Each dataset contains 30.000 connection pairs.

We furthermore create a smaller **dataset D** with differing numbers of stepping-stones (1,3,5, and 8 jumps) without evasive tactics to evaluate this effect on RTT-based methods. For every jump, we generated 1.000 connection pairs.

For all datasets, we randomise and record network settings. In Section 5.6, we examine the effect of extreme network settings on detection rates.

3.2 Benign data

We need to provide a realistic background of benign data that reflects the heterogeneous nature of regular network traffic, both for evaluation of false positives and for the training of ML-based methods.

We include real-world traffic traces, taken from the **CAIDA** *2018 Anonymized Internet Traces* dataset [4]. This data contains traces collected from high-speed monitors on a commercial backbone link, and is often used for research on the characteristics of Internet traffic, including application breakdown, security events, geographic and topological distribution, flow volume and duration.

The CAIDA dataset provides suitable general background traffic. However, to sufficiently test for false-positive, we need to include benign traffic that has

similar characteristics to the attack traffic and was generated in a similar network environment, which the CAIDA dataset not necessarily provides. For that, we created a set of interactive SSH-connections that are conducted directly between the client and the server instead of via a stepping-stone. We follow the same procedure as described in Section 2.2 with the same randomised network congestion settings as described in 2.2. We then proceed to pair connections by random.

Additionally, we include a third type of benign traffic to test false-positives of methods aiming to spot chaff perturbations. Since we followed the findings of Padhye et al. [20] to generate perturbations that resemble multimedia streams, we want to include actual multimedia streams. For that, we captured traffic from a server streaming video to a client. Video content is randomised and multiple values for the streaming quality are used. The server and client are hosted in the same virtual Docker network with randomised network congestion settings. Again, we pair generated connections by random. However, for each pair the direction of the transfer has to be either to or from the common host to either resemble a client requesting two videos or a server sending to videos.

We merge the three datasets to create our benign background dataset, with the CAIDA part containing 60.000 connection pairs, while the other two each contain 20.000 connection pairs.

	Label	Nr. of conn.	Purpose
SS data	set A	30,000	Baseline attack data without evasion tactics
	set B	30,000	Inclusion of delays with varying δ_D
	set C	30,000	Inclusion of chaff with varying δ_C
	set D	40,000	Data from chains of different lengths, no evasion tactics
Benign data	CAIDA	60,000	General background data
	SSH	20,000	Background data similar to attack commands
	Multimedia	20,000	Background data similar to chaff perturbations

Table 1. Summary of different components in our evaluation data.

We are aware that the amount of interactive SSH traffic and multimedia streams in this setting is inflated from a realistic setting, but we believe that this setting is more suitable to highlight the strengths and drawbacks of SSD methods. In the evaluation, we will also state individual false positives for each of the three datasets to indicate which type of traffic present the biggest challenge for the selected SSD methods. Table 1 summarises the different parts in our evaluation data.

3.3 Data preparation

We group packets to connections according to the usual 5-tuple consisting of {Src. IP, Dst. IP, Src. port, Dst. port, IP protocol}. We select and pair TCP-connections at random to make connection pairs.

Different approaches require different amounts of packets to make predictions. To create a fair playing field, we only look at connections that exchange more than 1500 packets and exclude shorter connections from both the background data. This number seems like a suitable minimal limit for a successful interactive stepping-stone chain as all of the selected methods are supposed to make successful detection with less packets, and there were no connections with less packets in the stepping-stone dataset. The first 1500 packets are then passed to each SSD method to make a prediction.

The initialisation of the SSH-tunnels usually follows a distinct pattern that can be learned and consequently boost detection rates for ML-based methods. Since this pattern is not necessarily representative of actual stepping-stone behaviour, we remove the first thirty packets of all captured connections. If necessary, we also remove the last thirty packets to avoid the same issue for connection closures.

All of the above presented methods classify individual connections or pairs of connections as malicious or benign. True stepping stone connections are rare compared to benign ones, making their detection an imbalanced classification problem. An appropriate measure to evaluate SSD methods are false positive and false negative rates as well as the *Area-under-ROC-curve* (AUC) for threshold-based methods.

4 Selected SSD methods and Implementation

The main contribution of this work is to offer a modern evaluation of the current state of SSD methods. A range of underlying techniques exist for SSD, and we try to include approaches from every area to create an informative overview and highlight strengths and weaknesses.

We surveyed publications to create a collection of SSD methods. We started with the publications from surveys [22, 25], and then added impactful recent publications found via Google Scholar[1].

From here, we selected approaches based on the following criteria:

1. The achieved detection and false positive rates claimed by the authors,
2. and whether the model design shows robustness against any evasion tactics as claimed by the authors.
3. We always selected the latest versions if a method has been improved or updated by the authors.

[1] keywords “connection”, “correlation” “stepping-stone”, “detection”, “attack”, “chaff perturbation”

Below, we describe the selected methods. Table 2 contains a summary of the included methods. We labelled each method to make referring to it in the evaluation easier.

Category	Approach	TP	FP	Robustness	Label
Packet-corr.	Yang, 2011 [31]	100%	0%	jitter/$<$ 80% chaff	PContext
Neural networks	Nasr, 2018 [19]	90%	0.0002%	small jitter	DeepCorr
	Wu, 2010 [28]	100%	0%	-	WuNeur
RTT-based	Yang, 2015 [32]	not provided		50% chaff	RWalk
	Huang, 2016 [16]	85%	5%	-	Crossover
Anomaly-based	Crescenzo, 2011 [8]	99%	1%	jitter/chaff	Ano1
	Huang, 2011 [15, 10]	95%	0%	$>$ 25% chaff/ $>$ 0.2s jitter	Ano2
Watermarking	Wang, 2011 [26]	100%	0.5%	$<$ 1.4s jitter	WM

Table 2. Summary of included SSD-methods along with the claimed true positive and false positive rates and evasion robustness by the corresponding authors. We added labels to each method for later reference.

PContext, 2011 Yang et al. [31] compare sequences of interarrival times in connection pairs to detect potential stepping-stone behaviour. For that, the context of a packet is defined as the packet interarrival times around that packet. The context of packets is extracted from each connection, and their respective contextual distance is estimated using the Pearson correlation. The authors propose to only collect interarrival times from *Echo*-packets instead of *Send*-packets to resist evasion tactics such as chaff and jitter.The authors evaluate their results on connection pairs with up to 100% chaff ratio, with the model being able to successfully detect connection relays in all cases.

WuNeur, 2010 A notable initial example of neural network applications to SSD came from Wu et al. [27], and which was later improved by the authors [27]. The designed neural network model is based on sequences of RTTs. For this, a packet matching algorithm is used to compute RTTs, which are then fed as a fixed-length sequence into a feed-forward network to predict the downstream length of the chain. The network itself only contains one hidden layer and is relatively small. RTT-based methods achieves good results only if RTTs are small, i.e. the stepping-stone chain is completely contained within one LAN-network.

DeepCorr, 2018 A more recent example of neural network application comes from Nasr et al. [19], who train a deep convolutional neural network to identify correlation between two connections from the upstream and downstream interarrival times and packet sizes in each connection. The trained network is large, with over 200 input filters and consists of three convolutional and three feed-forward layers. The trained model was initially applied to a dataset of Tor-connections

as well, where the authors achieved strong results. The authors achieve a 90% detection rate with 0.02% false positives.

RWalk, 2015 This model by Yang et al. [32] combines packet-counting methods and RTT mining methods to improve detection results from [30]. The authors improve widely-used packet-counting methods to resist chaff perturbation by counting the number of round-trips in a connection to determine if the connection is being relayed. Packet pairs representing a round-trip for each connection are estimated using a combination of packet matching and clustering, and counted as N_{in} and N_{out}. The authors then claim that the value of $N_{in} - N_{out}$ is only bounded if the two connections are relayed.

C.Over, 2016 This method by Huang et al. [16] improves the detection methods proposed by Ding et al. [9]. The authors target relayed interactive SSH communication at the end of a connection chain. Their detection model is built on the fact that in a long connection chain, the round-trip-time of a packet may be longer than the intervals between two consecutive keystrokes. In a long connection chain, this will result in cross-overs between request and response, which causes the curve of sorted upstream RTTs to rise more steeply than in a regular connection. The authors do not state a universal threshold value and instead suggest a method to estimate the appropriate value for a given setting.

Ano1, 2011 Crescenzo et al. [8] have proposed an assembly of three anomaly-based methods to detect time delays and chaff perturbations in a selected connection. A response-time based method is targeted at detecting packet time-delays by flagging acknowledgement packets if their response-time exceeds the average RTT by a fixed threshold. A distance-based method compares the similarity of downstream with upstream packet sequences in order to detect inserted chaff packets. Finally, a causality-based method verifies that the number of ON-intervals in upstream and downstream direction match each other to detect chaff. The authors claim successful detection for chaff ratios 25% or more, and for delays introduced to up to 70% of all packets.

Ano2, 2011/2013 Huang et al. [15] proposed an anomaly-based method to detect chaff perturbations based on their assessment that interarrival times in regular connections tend to follow a Pareto or Lognormal distribution, which chaffed connections supposedly do not. The authors extract packet interarrival times and calculate statistical score describing the fit to the distributions that is used to classify connections into normal and chaffed connections. A similar approach from the same authors [10] is directed towards the detection of packet jittering. The authors achieve 95% detection rate on connections with 50% chaff ratio and more while retaining zero false positives using a small set of chaffed interactive SSH stepping-stone connections.

WM, 2010 Flow watermarking is the most effective way at correlating connections with low false positives. However, as found by [17], usually not very robust against timing and chaff perturbation attacks. We have selected the approach developed by Wang et al. [26] to be included in our evaluation because the authors state at least some resistance against timing perturbations. The authors assume some limits to an adversary's timing perturbations, such as a bound on the delays. The authors divide a sequence of packets into pairs, whose inter-arrival time is perturbed using a new watermarking function. The introduced watermark is invisible for third-parties, and the authors achieve 100% TP with 0.5% FP while claiming resistance against timing perturbations of up to $1.4s$.

4.1 Implementation of selected approaches

In order to evaluate the above selected method on our generated data, we had to reimplement the algorithms according to the steps and design described in the corresponding publications. We implemented all methods in Python, and PyTorch if necessary. To overcome unclear points or adjust to the new setting, we had to make a small number of judgement calls, the most important of which we mention here for completeness:

For all methods, either varied the particular separation threshold AUC calculation, or estimated it to yield a particular false positive rate across all methods. For both anomaly-based methods, we had to estimate multiple thresholds. We found appropriate values that yield the required FP rate and maximise the detection rate over all datasets via grid-search optimisation.

For the WM approach, we had to set the quantization step size $s = 0.4$s, and the redundancy number $m = 11$ to consistently embed the watermark in the first 1500 packets. These values were also used by the authors.

Both DeepCorr and WuNeur are machine-learning based algorithms, which need to be trained on appropriate training data before the evaluation. Since DeepCorr is a large model and requires a lot of training data, we train it using 80% of the evaluation data with the proportions described in Table 1. WuNeur requires less amounts of training data and is more sensitive to differing chain lengths, which is why we use 1000 connection pairs from each dataset A,B,C, and D, as well as 6000 connection pairs from the training data. We train each method for 500 epochs.

5 Results

5.1 Data without evasion tactics

First, we look at the detection rates for traffic from stepping-stones that did not use any evasive tactics, i.e. $S_1, \ldots, S_4$ are only forwarding the commands and responses. The successful detection of this tytpe of behaviour with low false-positives should be the minimum requirement for any of the selected methods except for the anomaly-based approaches. Since these aim to detect evasive behaviour, we exclude them from this analysis.

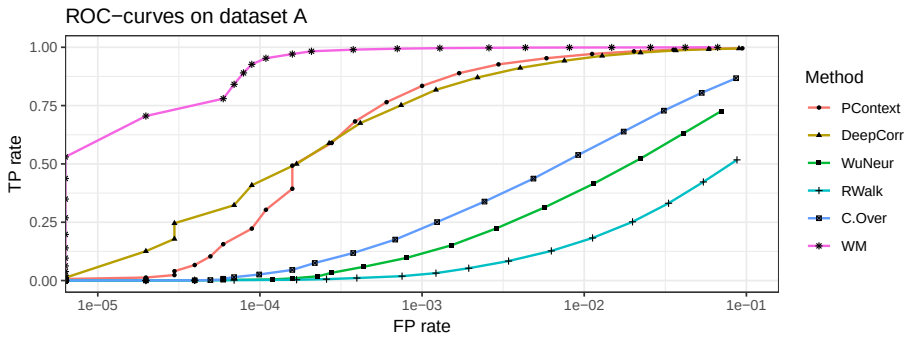

Fig. 4. ROC-curves for different SSD methods on dataset A (no evasive tactics). Anomaly-based methods are excluded.

	PContext	DeepCorr	WuNeur	RWalk	C.Over	WM
AUC	0.998	0.997	0.939	0.853	0.965	0.9998

Table 3. AUC-scores for different methods on stepping-stone data without evasive tactics.

To get clear picture of the capabilities of the selected methods, we look at the ROC-curve in Fig. 4, which plots the true positive rate against the false positive rate for various detection threshold settings. Table 3 depicts the overall AUC-scores.

Unsurprisingly, the watermarking method achieves already high detection results without any false-positives. Both the PContext and DeepCorr models start to yield good detection results of around 80% at a FP rate lower than 0.1%, with the PContext method slightly outpacing the DeepCorr method.

RTT-based methods seem to not perform as well compared to the other included methods, however the observed ROC curve seems to be in general in agreement of the stated detection rates.

5.2 Delays

We now consider the effect of transfer delays added by the attacker to packets on the detection rates. For that, we pick detection thresholds for each SSD methods corresponding to a FP rate of 0.4% as most methods are able to achieve some detection results at this rate. We look at delays added to only to outgoing packets on S_4, the last stepping stone in the chain. Fig. 5 depicts evolution of detection rates in dependence of the maximum delay δ_D.

As visible, both anomaly-based methods are capable of detecting added delays relatively reliably. Furthermore, both the detection rates of DeepCorr and the rtt-based C.Over only decrease slightly under the influence of delays. Detection rates for all other methods decrease significantly to the point where no

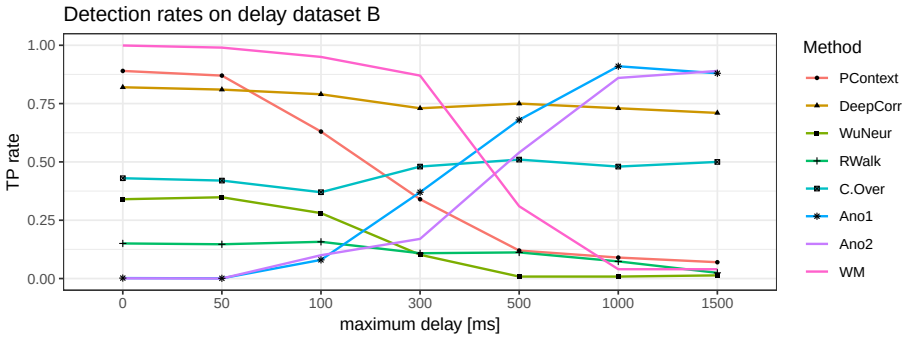

Fig. 5. Detection rates in dependence of δ_D for different methods on dataset B with a fixed FP rate of 0.4%.

meaningful predictions can be made. This is also reflected by the AUC-scores for traffic with $\delta_D = 1000ms$, given in Table 4.

While the WM method is robust against transfer delays up to $\delta_D = 500$ms, this value is smaller than the one claimed by the authors. This might however be a result of the slightly smaller quantisation step size that we used. It is surprising that the PContext method shows only little robustness against transfer delays, which contradicts the authors claims. A possible explanation is the false assumption that relying on *Echo*-packets increases robustness, since we also successfully apply delays to them while additional non-delayed *Echo*-packets are generated by the tunnels.

	PContext	DeepCorr	WuNeur	RWalk	C.Over	Ano1	Ano2	WM
AUC	0.639	0.995	0.619	0.639	0.953	0.997	0.996	0.559

Table 4. AUC-scores for SSD methods with added transfer delays at $\delta_D = 1000ms$.

5.3 Chaff

We now consider the effect of chaff perturbations added by the attacker to individual connections on the detection rates. Again we pick detection thresholds for each SSD methods corresponding to a FP rate of 0.4%.

Chaff packets are added to both the connection between S_3 and S_4 as well as between S_4 and host T as described in Section 2.3. Fig. 6 depicts evolution of detection rates in dependence of the ratio of number of chaff packets to packets from the actual interaction.

As visible, all methods struggle to detect stepping stones once the chaff packets become the majority of the transferred traffic. This is also evident from the AUC-scores given in Table 4. Several approaches claimed to be resistent to chaff

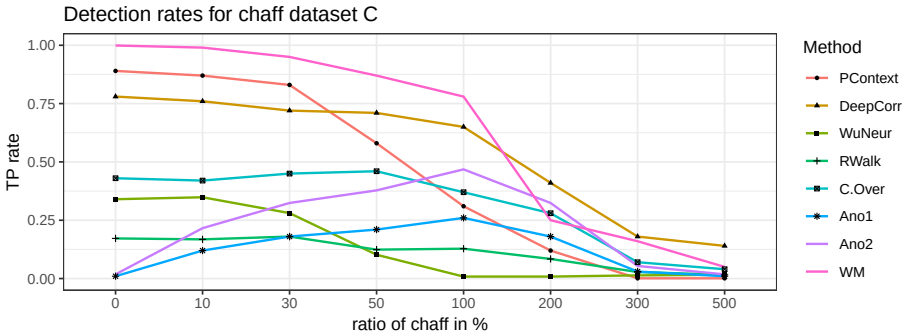

Fig. 6. Detection rates in dependence of δ_C for different methods on dataset C with a fixed FP rate of 0.4%

perturbations, however prior evaluations never looked at chaff ratios that exceed 100%.

It is surprising that the anomaly detection methods do not perform better at detecting chaff perturbations. Chaff in both approaches was however evaluated with different traffic generation distribution and not compared against a background of traffic following a similar generation distribution, which could explain the disagreement between the results we are finding here.

Overall, this result is in disagreement with the "robustness against chaff" claims made for four of the selected approaches, namely PContext, RWalk, Ano1, and Ano2.

	PContext	DeepCorr	WuNeur	RWalk	C.Over	Ano1	Ano2	WM
AUC	0.637	0.885	0.613	0.642	0.590	0.781	0.738	0.838

Table 5. AUC-scores for SSD methods with added chaff at 300% ratio.

5.4 False positives

Table 6 depicts the relative contribution[2] at $FP = 0.4\%$ of each of the three benign data types to the overall false positive rate. Most methods have more problems with the heterogeneous nature the CAIDA traces, with only PContext and DeepCorr seeing most false positives in the SSH traffic.

The multimedia traffic is causing most problems for the anomaly-based methods, persumably because it follows a similar distribution as the generated chaff perturbations.

[2] after adjusting for their weight

	PContext	DeepCorr	WuNeur	RWalk	C.Over	Ano1	Ano2	WM
CAIDA	0.37	0.37	0.51	0.67	0.57	0.49	0.36	0.82
SSH	0.52	0.46	0.22	0.23	0.32	0.08	0.00	0.12
multimedia	0.11	0.17	0.27	0.10	0.11	0.43	0.64	0.06

Table 6. Relative contribution in % of different benign data to the FP rate.

5.5 Influence of chain length

In this section, we look at the effect of differing chain lengths on the detection rates. We only focus on RTT-based methods here since the other methods do not seem to be significantly influenced. Since RTT-based methods aim to measure the effect of packets travelling via multiple hosts, it is unsurprising that they perform better at detecting longer chains.

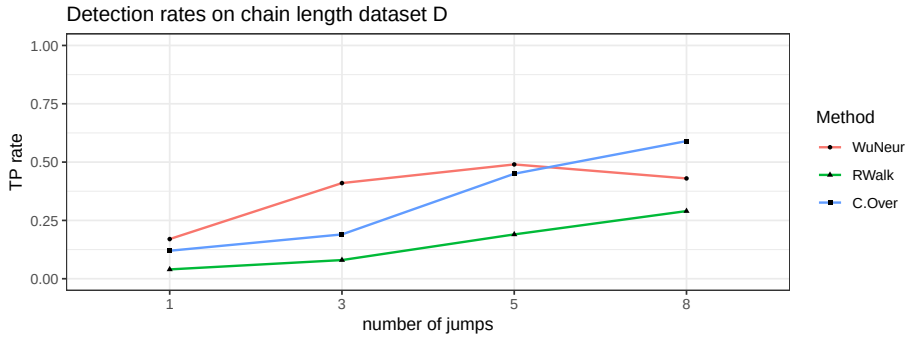

Fig. 7. Detection rates in dependence of chain length for different methods on dataset D with a fixed FP rate of 0.4%

Of the RTT-based methods, only C.Over was able to yield consistent detection rates under transfer delays. Interestingly, if the C.Over method is applied to connections between S_3 and S_4 instead of between S_4 and the target, detection rates decrease in the same manner as for other RTT-based methods. This is not surprising as the underlying assumption for robustness for this approach relies on Echo-packets not being delayed.

5.6 Influence of network settings

Finally, we look at the effect of different nework settings. We only show methods that show significant effects and omitted bandwidth from the evaluation as different values do not seem to have any effect on detection rates.

As visible in Table 7, the three RTT-based methods show different responses to small/large average round-trip-times. While WuNeur, as expected from prior

	Value	TP deviation from average				
		DeepCorr	WuNeur	RWalk	C.Over	WM
RTT	5ms	−0.2%	+41.3%	−42.3%	−36%	+0.03%
	70ms	−5.6%	−5.8%	+35.1%	+51%	−2.2%
Packet loss	0%	+1.2%	+1.3%	+2.1%	+4.3%	+0.02%
	7%	−9.1%	−1.1%	−3.1%	−7.3%	−9.7%

Table 7. Influence of network congestion on detection rates at a fixed FP rate of 0.4%. The given percentages are describing the change of the detection rate under the given congestion setting when compared to the overall average.

results, performs better in LAN settings, detection rates of the RWalk and C.Over methods are boosted by larger RTTs. All methods profit from lower packet losses.

5.7 Summary

Overall, detection rates on dataset A are mostly in line with the claimed capabilities except for RWalk, although detection rates are slightly lower than stated by most authors. Delay perturbation increases detection difficulty for most methods, except for Ano1, Ano2, and DeepCorr, which contradicts robustness claims for PContext and to some extend WM. Our inserted chaff perturbations however render detection impossible for all methods examined, which contradicts robustness claims for PContext, Ano1, Ano2, and RWalk, although the claims were made based on lower chaff levels than we tested.

Unsurprisingly, longer chains yield higher detection rates for RTT-based methods. Different network transmission settings seem to have little influence on detection rates.

6 Related work

6.1 Testbeds and data

In 2006, Xin et al. [29] developed a standard test bed for stepping-stone detection, called *SST*. The main objectives in the development of SST were to enable a reproducible evaluation of stepping stone chain detection algorithms with easy configuration, management and operation. The tool allows for an arbitrary number of intermediate hosts and generates scripts to mimic interactive SSH and TelNet connections. To insert time delays and chaff perturbations, the authors modified the OpenSSH protocol on the intermediary hosts. Delays can be drawn from a uniform distribution while chaff can be drawn from a Poisson or Pareto distribution. To our knowledge, SST has only been used for evaluation by Zhang et al. [33], and is not available anymore. The authors give little details about the implemented evasive tactics.

Another approach to use publicly available data comes from Houmansadr et al. [19]. The authors use the well-known CAIDA anonymised data traces

[4] to evaluate different watermarking methods for SSD. To simulate stepping stone behaviour, packet delays and drops are imposed retroactively on selected connections using Laplace and Bernoulli distributions with different rates. While this procedure seems sufficient for the evaluation of watermarking methods, it falls short on simulating the effects of an actual connection chain and leaves out chaff perturbations.

We find that when authors evaluate methods on self-generated data, tested evasive behaviours are often lacking analytical discussion and their implementation is too simplistic, leading to increased detection rates. An example of this can be seen in the evaluation of Ano1 [8], where a standard option in netcat is used to generate chaff perturbations for evaluation, or for PContext [32] where simulated chaff is added randomly after the traffic collection.Furthermore, often a relatively low limit on the amount of inserted chaff perturbations is assumed without obvious reason, thus avoiding evaluation at higher ratios. None of the methods we reviewed evaluate at chaff ratios above 100%.

6.2 Surveys

Wang et al. [25] recently conducted an extensive survey of stepping stone intrusion detection. The authors group methods according to the respective methodology but do not cover graph-based methods such as [13], or anomaly-based methods such as [8].The authors then proceed to explain the different methods and highlight their benefits and shortcomings as well as open problems. Shullich et al. [22] in 2011 also conducted a survey on stepping stone intrusion detection. The authors perform a similar grouping of methods, but also discuss related work in evasion tactics and test frameworks. The authors furthermore give an outlook on areas for future research, such as hacker motivation, the cardinality problem when correlating connection pairs, the difficulty of tracing back chains through firewalls, the lack of real-world data examples, or detection in covert channels or protocols such as UDP. Both surveys neither provide an evaluation of detection rates nor a direct comparison of the rates achieved by the corresponding authors.

6.3 Areas not covered in this work

A different direction in SSD that we did not discuss in this work focuses more on the general communication behaviour of selected hosts rather than individual connections. Features include the timely correlation of connecting IP-address on a selected host or unusual paths of simultaneously existing connections within a computer network. Notable examples include Apruzzese et al. [5] and Gamarra et al. [13], who both use graph-based models of network connections to identify suspicious relay hosts. Emulating these features realistically in network traffic datasets is difficult, and there exist little research on how strongly actual attack behaviour influences these features. Since our dataset focuses only on individual connections and corresponding evasion, we are not able to include behaviour-based approaches in our evaluation.

Coskun et al. [7] identify another form of stepping-stones called *store-and-forward*, which transfer data within files in a non-interactive manner. Though harder to detect than interactive connections, this procedure limits the attackers ability to explore the target, which is why SSD research has been primarily concerned with interactive stepping-stones.

7 Conclusion

In this work, we set out to evaluate the state-of-the-art of SSD methods using a comprehensive data generation framework. Our framework simulates realistic stepping-stone behaviour with SSH-tunnels in different settings. The implemented evasive perturbation tactics follow the findings of Padhye et al. [20] to resemble streaming services, and are adjustable in volume. We released a large dataset that highlights multiple aspects in SSD, and is suitable to train ML-based methods.

Overall, our results show that attackers can reliably evade detection by using the right type and amount of chaff perturbation, which disproves several claims made about the robustness against this evasive tactic. Although less significant, our implemented delay perturbations still affect detection rates for most methods, but are more visible to anomaly-based methods.

Currently, it seems that watermarking methods are currently most suited at detecting simple stepping-stones in real-life deployment due to very low false-positives. The performance of DeepCorr indicates that deep neural networks show the most potential at detecting methods that are subject to chaff or delay perturbations if they are trained on suitable data, and area we believe is worth more exploration. We find that detection and false-positive rates for RTT-based methods are significantly lower than for other methods, which makes them less suitable as stand-alone solutions.

References

1. M-trends 2015: A view from the front lines. Technical report, 2015.
2. Mcafee technical report on night dragon operation. Technical report, 2015.
3. Archimedes documentation, 2017.
4. The CAIDA UCSD Anonymized Internet Traces 2018, 2018. Accessed: 2020-02-10.
5. Giovanni Apruzzese, Fabio Pierazzi, Michele Colajanni, and Mirco Marchetti. Detection and threat prioritization of pivoting attacks in large networks. *IEEE Transactions on Emerging Topics in Computing*, 2017.
6. Luis Ayala. Active medical device cyber-attacks. In *Cybersecurity for Hospitals and Healthcare Facilities*, pages 19–37. Springer, 2016.
7. Baris Coskun and Nasir Memon. Efficient detection of delay-constrained relay nodes. In *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, pages 353–362. IEEE, 2007.
8. Giovanni Di Crescenzo, Abhrajit Ghosh, Abhinay Kampasi, Rajesh Talpade, and Yin Zhang. Detecting anomalies in active insider stepping stone attacks. *JoWUA*, 2(1):103–120, 2011.

9. Wei Ding, Matthew J Hausknecht, Shou-Hsuan Stephen Huang, and Zach Riggle. Detecting stepping-stone intruders with long connection chains. In *2009 Fifth International Conference on Information Assurance and Security*, volume 2, pages 665–669. IEEE, 2009.
10. Wei Ding, Khoa Le, and Shou-Hsuan Stephen Huang. Detecting stepping-stones under the influence of packet jittering. In *2013 9th International Conference on Information Assurance and Security (IAS)*, pages 31–36. IEEE, 2013.
11. David L Donoho, Ana Georgina Flesia, Umesh Shankar, Vern Paxson, Jason Coit, and Stuart Staniford. Multiscale stepping-stone detection: Detecting pairs of jittered interactive streams by exploiting maximum tolerable delay. In *International Workshop on Recent Advances in Intrusion Detection*, pages 17–35. Springer, 2002.
12. Gordon Fraser. Tunneling, pivoting, and webapplication penetrationtesting. Technical report, SANS, 2015.
13. Marco Gamarra, Sachin Shetty, David M Nicol, Oscar Gonazlez, Charles A Kamhoua, and Laurent Njilla. Analysis of stepping stone attacks in dynamic vulnerability graphs. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2018.
14. Stephen Hemminger et al. Network emulation with netem. In *Linux conf au*, pages 18–23, 2005.
15. Shou-Hsuan Stephen Huang and Ying-Wei Kuo. Detecting chaff perturbation on stepping-stone connection. In *2011 IEEE 17th International Conference on Parallel and Distributed Systems*, pages 660–667. IEEE, 2011.
16. Shou-Hsuan Stephen Huang, Hongyang Zhang, and Michael Phay. Detecting stepping-stone intruders by identifying crossover packets in ssh connections. In *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, pages 1043–1050. IEEE, 2016.
17. Alfonso Iacovazzi and Yuval Elovici. Network flow watermarking: A survey. *IEEE Communications Surveys & Tutorials*, 19(1):512–530, 2016.
18. Robert M. Lee, Michael J. Assante, and Tim Conway. Analysis of the cyber attack onthe ukrainian power grid. Technical report, E-ISAC, 2016.
19. Milad Nasr, Alireza Bahramali, and Amir Houmansadr. Deepcorr: Strong flow correlation attacks on tor using deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1962–1976, 2018.
20. Jaideep D Padhye, Kush Kothari, Madhu Venkateshaiah, and Matthew Wright. Evading stepping-stone detection under the cloak of streaming media with sneak. *Computer Networks*, 54(13):2310–2325, 2010.
21. Vern Paxson and Sally Floyd. Wide area traffic: the failure of poisson modeling. *IEEE/ACM Transactions on networking*, 3(3):226–244, 1995.
22. Robert Shullich, Jie Chu, Ping Ji, and Weifeng Chen. A survey of research in stepping-stone detection. *" International Journal of Electronic Commerce Studies"*, 2(2):103–126, 2011.
23. Stuart Staniford-Chen and L Todd Heberlein. Holding intruders accountable on the internet. In *Proceedings 1995 IEEE Symposium on Security and Privacy*, pages 39–49. IEEE, 1995.
24. Colin Tankard. Advanced persistent threats and how to monitor and deter them. *Network security*, 2011(8):16–19, 2011.
25. Lixin Wang and Jianhua Yang. A research survey in stepping-stone intrusion detection. *EURASIP Journal on Wireless Communications and Networking*, 2018(1):276, 2018.

26. Xinyuan Wang and Douglas Reeves. Robust correlation of encrypted attack traffic through stepping stones by flow watermarking. *IEEE Transactions on Dependable and Secure Computing*, 8(3):434–449, 2010.
27. Han-Ching Wu and Shou-Hsuan Stephen Huang. Performance of neural networks in stepping-stone intrusion detection. In *2008 IEEE International Conference on Networking, Sensing and Control*, pages 608–613. IEEE, 2008.
28. Han-Ching Wu and Shou-Hsuan Stephen Huang. Neural networks-based detection of stepping-stone intrusion. *expert systems with applications*, 37(2):1431–1437, 2010.
29. Jianqiang Xin, Lingeng Zhang, Brad Aswegan, John Dickerson, T Daniels, and Yong Guan. A testbed for evaluation and analysis of stepping stone attack attribution techniques. In *2nd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, 2006. TRIDENTCOM 2006.*, pages 9–pp. IEEE, 2006.
30. Jianhua Yang and Shou-Hsuan Stephen Huang. Mining tcp/ip packets to detect stepping-stone intrusion. *computers & security*, 26(7-8):479–484, 2007.
31. Jianhua Yang and David Woolbright. Correlating tcp/ip packet contexts to detect stepping-stone intrusion. *Computers & Security*, 30(6-7):538–546, 2011.
32. Jianhua Yang and Yongzhong Zhang. Rtt-based random walk approach to detect stepping-stone intrusion. In *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, pages 558–563. IEEE, 2015.
33. Linfeng Zhang, Anthony Persaud, Alan Johnson, and Yong Guan. Stepping stone attack attribution in non-cooperative ip networks. In *Proc. of the 25th IEEE International Performance Computing and Communications Conference (IPCCC 2006)*, 2005.