Interdependent Security Risk Analysis of Hosts and Flows

Mohsen Rezvani[1] Verica Sekulic[1] Aleksandar Ignjatovic[1]
Elisa Bertino[2] Sanjay Jha[1]

[1] University of New South Wales, Australia
{mrezvani,vericas,ignjat,sanjay}@cse.unsw.edu.au
[2] Department of Computer Science, Purdue University
bertino@cs.purdue.edu

Technical Report
UNSW-CSE-TR-201406
February 2014

THE UNIVERSITY OF
NEW SOUTH WALES

School of Computer Science and Engineering
The University of New South Wales
Sydney 2052, Australia

Abstract

Detection of high risk hosts and flows continues to be a significant problem in security monitoring of high throughput networks. A comprehensive risk assessment method should take into account the risk propagation among risky hosts and flows. In this paper this is achieved by introducing two novel concepts. The first is an interdependency relationship among the risk scores of a network flow and its source and destination hosts. In one hand, the risk score of a host depends on risky flows such a host initiates and is targeted by. On the other hand, the risk score of a flow depends on the risk scores of its source and destination hosts. The second concept, which we call flow provenance, represents risk propagation among network flows which takes into account the likelihood that a particular flow is caused by other flows. Based on these two concepts, we develop an iterative algorithm for computing the risk level of hosts and network flows. We give a rigorous proof that our algorithm rapidly converges to unique risk estimates, and provide its extensive empirical evaluation using two real-world datasets. Our evaluation demonstrates that our method is effective in detecting high risk hosts and flows and is sufficiently efficient to be deployed in high throughput networks.

1 Introduction

A significant challenge for monitoring of large enterprise networks is the complexity of extracting risky network flows from the large quantity of flows. Identifying the most likely malicious activities makes taking effective countermeasures a feasible task. For example, detected high risk network traffic can be forwarded to a deep packet inspection tool, such as an Intrusion Detection System (IDS) [24]. An IDS can then execute specific actions to determine whether there are actual intrusions or other attacks. The security risk level of a network activity such as a network flow can be evaluated based on both the risk level of the content of the flow and the amount of the risk which is propagated by its related flows. Such recursive assessment makes the detection of risky activities rather complicated because it needs to accurately define the relationship among network activities.

Distributed attack scenarios such as distributed denial of service (DDoS) attacks and Botnet initiated attacks are examples of attacks which generate malicious network flows that can be evaluated using the above recursive assessment. In such attack scenarios, an attacker can exploit plenty of methods and vulnerabilities across different systems in the network. A promising solution for risk assessment is one which takes into account inter-flow relationships. One of the best illustrations of the inter-flow relationship is when an attacker creates a web session on a public web server hosted within a demilitarized zone (DMZ), by compromising the web server. Since the web related flow traffic to such server is a permitted flow according the security policy, the attacker can initiate a new connection from the compromised server to a another server in a protected network zone which allows it to download rootkits, send massive spam or scan ports on the second server. This illustrates the fact that, in order to adequately evaluate the level of risk of the initial flow in this scenario, we need to consider the whole interdependency risk relationship among recorded network flows.

This indicates that, in order to address the problem of flow risk assessment, we need a comprehensive solution which considers the whole interdependency risk relationship among the network flows as well as the hosts initiating and targeted by the flows. This can be explained by the principle that the more risky flows a host initiates or is targeted by, the higher the risk level is for the host. Moreover, the risk score of a network flow partially depends on the risk scores of its source and destination hosts. Therefore, there is an interdependency relationship between network flows and hosts with respect to the assessment of their risk scores. The idea of employing link analysis techniques such as PageRank [3] and HITS [15] for detecting relevant IP flows has been proposed in the recent research [30, 29]. However, in such approach the levels of risk of hosts and levels of risk of flows are evaluated separately, without considering their interdependency. Moreover, this type of risk assessment requires an evaluation of two separate dependency graphs for assessing the level of risk for hosts and flows which is very inefficient for high throughput networks.

In our recent work [23] we proposed an interdependency risk model for ranking the risk level of network flows as well as the related hosts. In the proposed flow risk analysis, a network flow is likely to be risky if it is initiated or targeted by risky hosts. We consider a host to be risky if some of its related flows are risky. With such interdependency in mind, we develop an iterative algorithm

for calculating the level of risk of hosts as well as the level of risk of network flows. Moreover, we take into account two different aspects that may influence the level of risk of a network flow, the risk of flow attributes and the risk of the flow provenance. The risk of flow attributes is defined by an aggregation of the risk level of the source and destination hosts of a flow and a predefined risk level of the network flow.

Extending our previous work [23], this paper presents a comprehensive risk assessment method for flows and hosts with an extensive analytical and experimental analysis. With respect to our previous work, this paper introduces some significant new contributions. First, here we augment the flow dependency graph with the notion of weighted flow causality relationship which facilitates a formulation of a weighted risk propagation model. Second, we provide a mathematically rigorous analysis of the behaviour of our iterative algorithm; in particular, its convergence and uniqueness of the solution generated are formally proved. Third, we study the time complexity and memory usage of our risk assessment algorithm, which are important efficiency factors for online monitoring algorithms. Our analysis shows that both the memory usage and the time complexity of our algorithm are linear with respect to the number of network flows in the algorithm time window. The experimental evaluation confirms our analysis regarding the efficiency of the algorithm. Last but not least, sensitivity analysis and experimental evaluations are conducted over two public datasets of packet traces which include two attack scenarios.

The reminder of the paper is organised as follows. The assumptions and preliminary definitions are introduced in Section 2. Section 3 presents the details of our proposed risk computation model. In Section 4, we present the properties of our iterative algorithm. Performance analysis and experimental results are presented in Section 5. The related work is presented in Section 6. Concluding remarks are made in Section 7.

2 Definitions of Basic Notions

In this section, we introduce several concepts used in our risk computation model. These concepts include a basic definition of NetFlow, flow causality relationship, flow dependency graph and flow provenance based on such a graph. Throughout this paper, the terms IP flow and NetFlow are used interchangeably.

2.1 Network Flow

A network flow can be defined as a unidirectional or bidirectional sequence of IP packets that belong to the same communication session between an application at a source host, and an application at a destination host. They also have a few common attributes such as being a member of the same TCP connection or UDP session. A network flow can be identified through the transport header 5-tuple including the source and destination ports, the source and destination IP addresses, and the protocol which usually is TCP or UDP.

Although the aforementioned 5-tuple can identify a network flow, a complete description of a network flow can only be achieved using a full packet trace captured by a tool such as *tcpdump*. However, handling full packet capture for security analysis in a large enterprise network is a significant challenge in

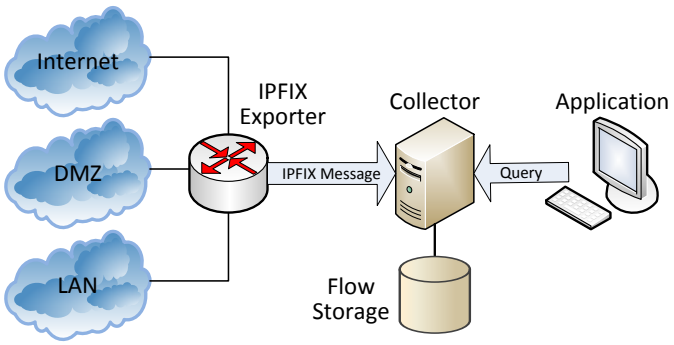


Figure 2.1: IPFIX reference model.

Table 2.1: NetFlow samples based on session 5-tuple.

id	start_t	s_ip	s_port	d_ip	d_port	pro
f_1	108.37	98.114.205.102	1821	192.150.11.111	445	tcp
f_2	108.51	98.114.205.102	1828	192.150.11.111	445	tcp
f_3	110.47	192.150.11.111	1957	98.114.205.102	1924	tcp
f_4	113.46	192.150.11.111	36296	98.114.205.102	8884	tcp
f_5	114.52	98.114.205.102	2152	192.150.11.111	1080	tcp

terms of time and space complexity. Additionally, attackers frequently utilize encryption and other methods to obfuscate their communication and defeat deep packet inspection tools such as an IDS [24]. Furthermore, violation of privacy is also a concern in raw packet analysis techniques.

By aggregating the packets of a flow, a summary information about the flow can be extracted and applied for network management which can reduce the overhead incurred when inspecting the tremendous quantity of network packets in large networks. Thus, in this paper, we employ a summary of flow features in our risk assessment method. These features include the above 5-tuple along with the start time of the network flow.

Figure 2.1 illustrates the overall architecture of the IP Flow Information Export (IPFIX) reference model [25] which is a deployment model for a network flow analysis tool. The three main components in this model consist of an Exporter, a Collector and an Application. Network devices such as switches and routers can support the task of an IPFIX exporter by sending the IPFIX messages to the collectors. The IPFIX collector can receive the flow packets from several exporters and supports a query language on its flow storage. In our work, the Application on Figure 2.1 measures the risk score of flows based on an iterative computational method.

Example 1. As an example of NetFlow, Table 2.1 shows the flows collected from the Honeynet Project, Challenge 1 of the Forensic Challenge 2010 [5]. This table only includes a summary of information for bidirectional and session based flows including the start time and the 5-tuple of the TCP/UDP session.

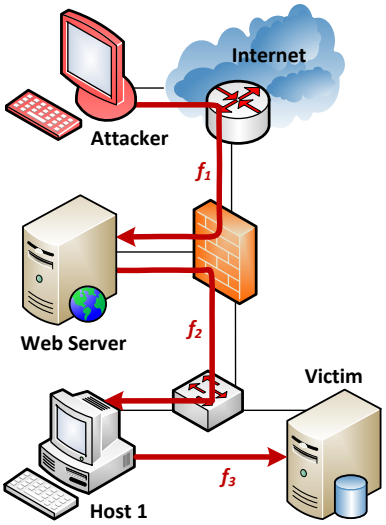


Figure 2.2: An example for anonymity through connection chains.

2.2 Flow Causality

In many real attack scenarios, the attacker typically exploits vulnerabilities in a number of intermediate hosts, called *stepping stones* [1]. In a distributed system, an attack can be carried out by a sequence of network flows in which the first flow in the sequence has originated by the attacker host and the last flow has as destination the target victim of the attack.

There is a large volume of published literature for detecting stepping stones [1]. However, rather than detecting stepping stones, our method relies on the inherent causality relationship which, in such a case, exists between each two consecutive network flows.

Example 2. Figure 2.2 shows a simple attack scenario. In this scenario, the attacker compromises the web server and then gains access to a terminal on the server (flow f_1). From the web server he may connect into vulnerable host 1 through an SSH tunnel using a password cracking attack (flow f_2). The attack on the victim is then launched from host 1 over an internal connection (flow f_3). In this example, flow f_1 triggered flow f_2 and flow f_2 triggered flow f_3.

In order to formally define the notion of flow causality, we use timing information along with source and destination addresses of network flows. In general, we aim to model the flow causality between two flows through the likelihood that one of them triggers the other one. Moreover, we define the notion of weighted causality relationship between two flows because a flow is more likely to be the cause of other flows if these flows started shortly after it. Wang et al. [28] defined a similar *dependency strength* between two flows based on timing information of these flows. Our formal definition of flow causality is an extension of their definition.

Definition 1. ***(Flow Causality)*** *The flow causality between flow f_y with respect to flow f_x, denoted as* $\text{fcs}(f_x, f_y)$*, is a measure of the likelihood that flow*

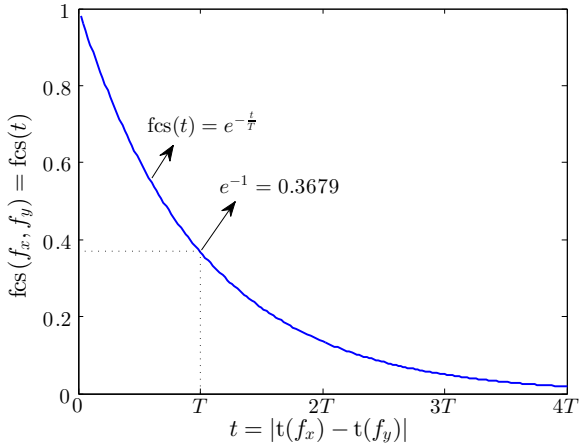


Figure 2.3: Plot of $\text{fcs}(t) = e^{-\frac{t}{\alpha T}}$ with $\alpha = 1$.

f_x triggered flow f_y. It is defined as:

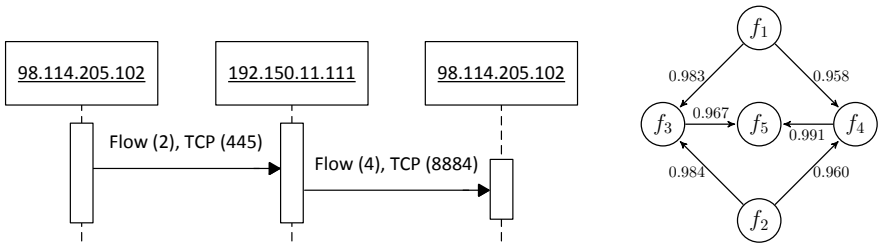
$$\text{fcs}(f_x, f_y) = \begin{cases} e^{-\frac{|\text{t}(f_x)-\text{t}(f_y)|}{\alpha T}} & \textit{if } \text{dst}(f_x) = \text{src}(f_y) \textit{ and} \\ & 0 < t(f_y) - t(f_x) \leq T, \\ 0 & \textit{otherwise}, \end{cases} \tag{2.1}$$

where $\text{t}(f_x)$, $\text{src}(f_x)$ *and* $\text{dst}(f_x)$ *denote the start time, source address and destination address of network flow f_x, respectively. T is the maximum distance between the starting times of two network flows to be considered for causality relationship, and is called causality time interval. Moreover, α is a constant chosen to reflect the impact of the time distance between two flows in the flow causality, and is called timing factor*[1]*.*

As defined above, f_y is likely caused by f_x if during a particular causality time interval T, flow f_y starts after flow f_x and the source address of f_y is the same as the destination address of flow f_x. Clearly, the above definition, if directly applied, may likely lead to false positives in stepping stones detection when some network flows start quickly after other flows without any causality. While this is indeed the case, our iterative procedure is likely to filter out such false positives, unless false positives happen on the entire risky path, which is unlikely. Moreover, we should remember that such unlikely events are acceptable, because we aim at identifying flows which are *likely* to be risky, and also that our proposed risk propagation works quite well without any modification for other definitions of flow causalities.

In order to take into account the time distance between two flows for computing the causality weight, we choose an exponential function which decreases sharply as the time distance increases. Figure 2.3 shows a plot of function $\text{fcs}(t) = e^{-\frac{t}{\alpha T}}$ where t is the temporal distance between two flows.

[1]In the experiment and all examples, we set $\alpha = 1$.



(a) Flow sequence diagram. (b) Flow dependency graph.

Figure 2.4: An example for flow dependency graph.

2.3 Flow Dependency Graph

The main idea behind the flow dependency graph is to model causality among network flows in order to measure the propagation of risk across network flows. We will also employ the flow dependency graph in order to extract the probable attack pathways which a network flow has initiated. This dependency graph is based on the notion of graph proposed in [29].

Definition 2. ***(Flow Dependency Graph)*** *The flow dependency graph is a weighted directed graph, generated by a sequence of flows* $F = \{f_1, f_2, \ldots, f_n\}$ *monitored during a particular time window; nodes are the flows in the sequence* F*, while the edges represent possible causality between corresponding flows, with weights of each edge given by the weight function* $\text{fcs}(f_i, f_j)$ *from equation* (2.1)*.*

Note that we assume that all flows are collected by a centralized or a number of distributed monitoring machines and the start time of the flows are assigned by the machines. Moreover, we assume that all clocks on the monitoring machines are synchronized using Network Time Protocol (NTP).

Example 3. Figure 2.4 shows an example of a flow dependency graph for network flows from Table 2.1. Figure 2.4(a) illustrates the temporal sequence of two flows f_2 and f_4 using a sequence diagram in the Unified Modeling Language (UML) notation. Figure 2.4(b) shows the flow dependency graph for all flows including the vertices labeled by the flow identifiers and the edges labeled by the causality among the flows. In this example, for computing the flow causalities, we set the causality time interval to $T = 120$.

A significant benefit of using the flow dependency graph in our risk assessment model is that it allows us to represent the causality between network flows. This causality can be considered as resulting from a possible attack scenario in which the attacker creates the flow A in the first step (from host 1 to host 2) and after compromising the destination of this flow (host 2), the attacker establishes the second flow B (from host 2 to host 3). Moreover, this causality relationship shows that the risk level of flow A not only depends on the risk of its features but it also is influenced by the risk score of flow B because flow A likely leads to flow B.

Lemma 1. *The flow dependency graph is a Directed Acyclic Graphs (DAG).*

Proof. Assuming the opposite, if we have at least one cycle in the graph which is a path $(f_1, f_2, \ldots, f_k)$ such that (f_k, f_1) is also an edge, according to definition 1

for timing information of network flows involved in a causality relationship, we would have $t(f_1) < t(f_2) < \ldots < t(f_k) < t(f_1)$, which is a contradiction.

□

A significant challenge in maintaining a flow dependency graph is the scalability, due to possibly huge number of flows in the graph for high throughput networks. Therefore, in Section 3 we will propose an efficient algorithm for measuring the level of risk from this graph which makes our methodology scalable.

2.4 Flow Provenance

We first introduce the definition of risky path in a flow dependency graph, to be used to represent a potential attack scenario initiated by a flow.

Definition 3. ***Risky Path.*** *A risky path of a network flow f is every path in the flow dependency graph starting at the node corresponding to f and ending at a leaf node in the graph.*

In the risk evaluation algorithm, we will consider the risk level of risky paths for each flow. It is clear that there may be more than one risky path for a flow in the flow dependency graph. We can consider either the highest risk value or sum of risk values of all risky paths for a flow; in our implementations we opted for the sum of risk values of all risky paths. Choosing the sum has an advantage of making all operations linear, which facilitates the proof of convergence of our method, presented in Section 4. More precisely, the flows which participated in more risky paths will be assigned a higher risk score. We now define a new graph, called *the flow provenance*, whose purpose is to simplify an evaluation of the risk level for a flow.

Definition 4. ***Flow Provenance.*** *The flow provenance t_f of a network flow f is a subgraph of the flow dependency graph with the following properties: (1) t_f includes the corresponding node of flow f and all nodes which are included in the risky paths of f; (2) t_f contains all the edges of the flow dependency graph that are between two nodes in t_f.*

Example 4. In the flow dependency graph shown in Figure 2.4(b), there are two different risky paths for network flow f_1: $f_1 \rightarrow f_3 \rightarrow f_5$ and $f_1 \rightarrow f_4 \rightarrow f_5$. The flow provenance of network flow f_1 is the subgraph that includes nodes f_1, f_3, f_4 and f_5 and all of the edges between these nodes.

3 Provenance-Aware Risk Computation

In this section, we first introduce the conceptual organization of our iterative risk computational framework which includes several different risk scores. We then explain the details of computation operations for all of these risk scores. Table 3.1 contains a summary of notations used in this paper.

3.1 Iterative Framework

The main idea behind our risk computation system is to model the network flow monitoring activity as a data management problem. Therefore, the risk

Table 3.1: Notation used in this paper.

Notion	Meaning
F	set of all flows in the current time window
H	set of all hosts in the current time window
w	window length in number of flows
T	causality time interval
$t(f)$	start time of a flow f
$\text{fcs}(f_i, f_j)$	flow causality between two flows f_i and f_j
α	timing factor for the flow causality
$\text{src}(f)$	source host of a flow f
$\text{dst}(f)$	destination host of a flow f
$\text{srv}(f)$	network service of a flow f including protocol and destination port number
$\text{mlw}(s)$	number of malware using a service s
$\widehat{\text{hr}}(h)$	intermediate risk score of a host h
$\text{hr}(h)$	final risk score of a host h for a time window
$\text{ar}(f)$	risk score of a combination of attributes of a flow f
$\text{sr}(s)$	risk score of a network service s
$\text{pr}(f)$	risk score of provenance of a flow f
$\text{npr}(f)$	normalised risk score of provenance of a flow f
$\widehat{\text{fr}}(f)$	intermediate risk score of a flow f
$\text{fr}(f)$	final risk score of a flow f for a time window
$\text{spr}(p)$	risk score of a risky path p
$N_p(f)$	number of risky path shared between a flow f and its parent
$N_t(f)$	number of flow causalities in all risky paths of provenance of a flow f

assessment can be modelled as a negative trust (distrust) computation. Authors in [8, 16] proposed a provenance-based model for data providers which assumes an interdependency relationship between data items and data providers. We model network flows as data items which are provided by network hosts. Hence, we define an interdependency relationship between network hosts and flows in order to formulate the risk measurement for both of them.

The provenance concept in data trustworthiness models represents the path of provisioning a data item, whereas we have defined the concept of provenance of a flow based on flows which are probably caused by that flow. While the data provenance concept is related to the process of generating data items by various data providers, the notion of flow provenance which we have introduced represents network activities which are generated by the flow. Note that in the usual sense of provenance as used in data management, the path of generating data by various data providers is used as a parameter for measuring data trustworthiness [8]. In our definition, the direction is somewhat reversed, in the

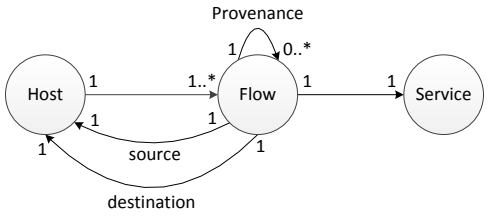


Figure 3.1: Interdependency relationship between host and flow.

sense that the risk of a flow is impacted by the risk of further flows which are caused by such a flow.

Figure 3.1 shows this interdependency between the host and the flow risk levels in our risk computation model. As we can see from this figure, the risk scores are assigned to both hosts and flows, in an interdependent manner. Accordingly, the risk of a host is computed by aggregating of the risk scores of the network flows which are either initiated by or targeted at the host. Therefore, there is a one-to-many relationship between the risk score of a host and its initiated and targeted flows. Furthermore, the level of risk of a flow is measured by the risk scores of source host, destination host, service type, and provenance of the flow. Hence, the dependency of the risk score of a flow to its source, destination and service is a one-to-one relationship, while the flow provenance leads to a one-to-many relationship between the risk score of a flow and risk scores of all flows within the provenance of such flow. Figure 3.1 illustrates these relationships using the UML multiplicity notation.

In order to deploy our risk assessment system on an ISP network, we need a way for handling a large number of flows. The flows monitored are an input stream for our system and they are handled in overlapping time windows. In other words, we apply our risk assessment method on the current window, also using as an initial risk evaluation value for hosts and flows obtained from the previous window. Thus, risk evaluations in each subsequent window are obtained via an update mechanism from the corresponding values from the previous window.

Figure 3.2 illustrates the iterative framework for computing the risk scores of hosts and flows. We first explain the overall architecture of our system, leaving the details to the subsequent sections. As shown in this figure, the two main modules of our framework are the risk evaluation component for the current time window and the update mechanism. Dashed lines are traversed in each iteration within a computation for each window; the solid lines are traversed from one window to the next one.

The risk score of a flow will be defined as an aggregate of the risk score of its attributes and risk score of its provenance. For a set of monitored flows in the current window, we iteratively compute the risk scores for flow attributes and flow provenance respectively. In each iteration these two computed risk scores for each flow are combined together in order to obtain the new value of the risk score of the flow; such a score is used in the subsequent iteration to update the risk score of the provenance as well as to compute the risk scores of the hosts; the risk score of the host is then also used in the next iteration to obtain the risk scores of the attributes.

Such iterative risk computations for the current window will be repeated until the changes in the risk scores become negligible. After finishing such iterative

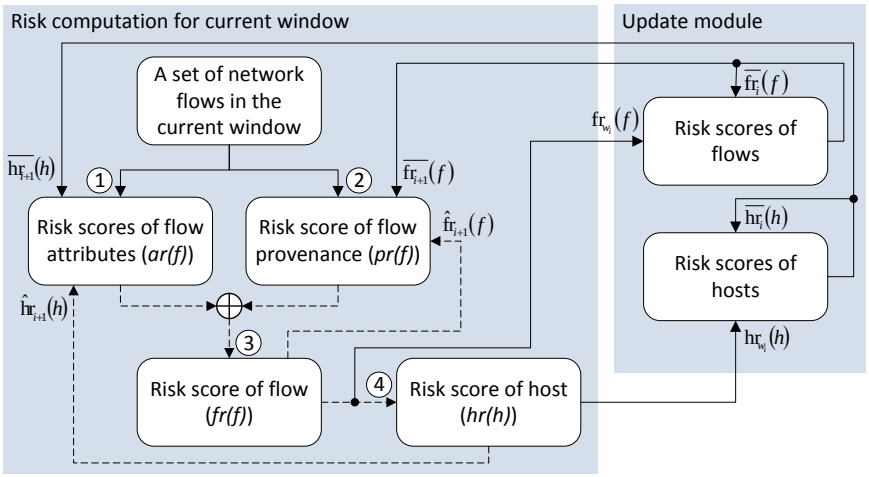


Figure 3.2: An iterative framework of computing risk scores of hosts and flows.

risk computation for the current time window, the risk scores of hosts and the risk scores of the flows are passed to the update module. In the update module, the current results will be combined with the results from the previous time window by producing a weighted sum of such values and then a computation for next time window will be started. We explain the details of such computation process of this framework in the next sections.

3.2 Risk Score Computation for Network Flows

The risk score of each network flow is obtained by aggregating the risk score of its attributes and the risk score of its provenance; the computation of the risk score of the provenance exploits the flow dependency graph.

Risk of Flow Attributes

The risk score of flow attributes is calculated based on the risk scores of source host, destination host, and the service type of the flow. For the risk computation of source and destination hosts, we use the current risk score of related hosts, while for computing the risk of flow service, we need a prior knowledge about the risky services in the network services (see ① in Figure 3.2).

Although we can roughly assign risk scores to network services, we allow the possibility that the risk scores of network services are supplied by the administrator, based on his prior knowledge about the network services. The Emsisoft Portlist [10] lists the ports that are more frequently exploited by malware, such as the TCP and UDP ports, as well as the malware using these ports. Based on this port list, a service risk level for each flow is assigned. In other words, if the destination port of a flow is in this port list, we assign a risk score to its service according to the number of detected malwares using the port. Thus, the risk score of service s obtains as follows:

$$\mathrm{sr}(s) = \frac{1 + \mathrm{mlw}(s)}{\sum_{s' \in Portlist} (1 + \mathrm{mlw}(s'))} \tag{3.1}$$

where mlw(s) is the number of malware using service s reported by Emsisoft Portlist. For example, there are five different malwares exploiting the *Telnet* service, thus mlw(*Telnet*) = 5; however, since no malware has been reported for the *SSH* service, mlw(*SSH*) = 0. Moreover, the denominator term in equation (3.1) is used to normalise the risk score of each flow service in order to bind the risk score to the range of (0,1].

Having obtained the risk scores of all three attributes of a network flow, we can define the risk score of attributes of a flow f as a simple average as follows[1]:

$$\mathrm{ar}(f) = \frac{\mathrm{hr}(\mathrm{src}(f)) + \mathrm{hr}(\mathrm{dst}(f)) + \mathrm{sr}(\mathrm{srv}(f))}{3} \tag{3.2}$$

where src(f), dst(f) and srv(f) denote the source, destination and network service of flow f, respectively.

Risk of Flow Provenance

The risk score of a flow provenance is obtained from the risk scores of the nodes in the provenance (see ② in Figure 3.2). As defined in Section 2, a flow provenance is a subgraph of the flow dependency graph which includes a number of risky paths. The risk score of a risky path is obtained as a weighted sum of the risk scores of the flows within the path, with weight equal to the flow causality between the node and its parent in the risky path. The rationale behind such a risk computation for a risky path is that the risk value indicates the likelihood of an attack proceeding via such risky path. Thus, we compute the risk score of a risky path p, denoted as spr(p), as follows:

$$\mathrm{spr}(p) = \sum_{f \in p} \mathrm{fr}(f) \times \mathrm{fcs}(\mathrm{parent}(f, p), f), \tag{3.3}$$

where fr(f) denotes the risk score of flow f and parent(f, p) is the network flow corresponding to the parent node of f in the risky path p.

Since a flow provenance contains a number of risky paths, we compute the risk score of flow provenance for flow f as the sum of risk scores of the risky paths within flow provenance t_f, i.e.,

$$\mathrm{pr}(f) = \sum_{p \in t_f} \mathrm{spr}(p); \tag{3.4}$$

note that the summation is over all risky paths p within t_f. In order to facilitate the proof of convergence of our iterative risk computation algorithm, we normalised the risk value of flow provenances to ensure that the risk value of flow provenances is in the range of [0,1]; we will prove that this is indeed the case in the next section. To obtain such normalisation, we first compute recursively the total length $N_t(f)$ of all risky paths in t_f; thus, each edge is counted with a multiplicity equal to the number of risky paths which contain that edge.

$$N_t(f) = \begin{cases} 0 & f \text{ is a leaf node,} \\ \sum\limits_{f \to f'} (N_t(f') + N_p(f')) & \text{otherwise.} \end{cases} \tag{3.5}$$

[1] All of the functions in the equations below can be found in Table 3.1.

We now find the largest value of N_t in the entire graph, i.e. we define

$$M = \max\{N_t(f') \ : \ f' \in F\}. \tag{3.6}$$

To compute the value of $\mathrm{pr}(f)$ recursively, we first define a function which for each node f computes recursively the number of risky paths within t_f:

$$N_p(f) = \begin{cases} 1 & f \text{ is a leaf node,} \\ \\ \sum\limits_{f \to f'} N_p(f') & \text{otherwise.} \end{cases} \tag{3.7}$$

It is easy to see that the normalised sum of the right hand side of (3.4), i.e., $\sum\limits_{p \in t_f} \mathrm{spr}(p)/M$ can be written in a recursive manner as

$$\mathrm{npr}(f) = \begin{cases} 0 & f \text{ is a leaf node,} \\ \\ \sum\limits_{f \to f'} \mathrm{npr}(f') + \frac{N_p(f')\mathrm{fr}(f')\mathrm{fcs}(f,f')}{M} & \text{otherwise.} \end{cases} \tag{3.8}$$

According to equations (3.5) and (3.7), the values of $N_t(f)$ and $N_p(f)$ are only dependent on the structure of the flow dependency graph. Thus, for each time window we compute $N_t(f)$ and $N_p(f)$ once, before starting the iterative procedure for risk computation. Therefore, we use a Depth First Search (DFS) algorithm on the flow dependency graph to compute the values of $N_t(f)$ and $N_p(f)$, as shown in Algorithm 1.

Note that equation (3.8) shows that the risk score of the flow provenance of a network flow depends on the risk scores of its children in the flow dependency graph (i.e., nodes f' suct that $f \to f'$). Thus, during the recursive procedure for computing the risk scores, we use in each round of iteration a DFS algorithm to compute the normalised risk scores of provenances for all flows in the graph, as shown in Algorithm 2.

Since the flow dependency graph is mostly a sparse graph, we insert a new virtual node in the graph, with edges to all existing nodes to allow all of the DFS algorithms to traverse all nodes of the graph; this new virtual node is the starting node for DFS algorithms. The algorithms recursively compute and store the values of $N_p(f)$, $N_t(f)$ and $\mathrm{npr}(f)$ for each flow (represented as a node in the graph) in global arrays *np*, *nt* and *npr*, respectively.

Flow Risk Aggregation

As we described, the risk score of a flow is computed by aggregating the risk values of the attributes and provenance of that network flow (see ③ in Figure 3.2) by a weighted sum:

$$\widehat{\mathrm{fr}}(f) = c_f \mathrm{ar}(f) + (1 - c_f)\mathrm{npr}(f) \tag{3.9}$$

where c_f is a constant[2], $0 \leq c_f \leq 1$. Thus, an administrator can select a larger value for c_f when he is confident which the risky services are. On the other hand, a smaller value for this constant is appropriate when a higher weight is to be given to flow provenances.

[2]In the experiment we set $c_f = 0.5$ to equally reflect the importance of $ar(f)$ and $pr(f)$.

Algorithm 1 Recursive algorithm for computing N_t and N_p

```
1: procedure ComputeNtAndNp(Graph G, Vertex v)
2:     if not visited[v] then
3:         visited[v] ← true
4:         Let f_1, ..., f_k be k child nodes of v
5:         if v is a leaf node then
6:             np[v] ← 1
7:             nt[v] ← 0
8:         end if
9:         for all i (i = 1, ..., k) do
10:            if not visited[f_i] then
11:                ComputeNtAndNp(G, f_i)
12:            end if
13:            np[v] ← np[v] + np[i]
14:            nt[v] ← nt[v] + np[i] + nt[i]
15:        end for
16:    end if
17: end procedure
```

Algorithm 2 Recursive algorithm for computing risk of flow provenance

```
1: procedure ProvenanceRisk(Graph G, Vertex v)
2:     if not visited[v] then
3:         visited[v] ← true
4:         Let f_1, ..., f_k be k child nodes of v
5:         if v is a leaf node then
6:             npr[v] ← 0
7:         end if
8:         for all i (i = 1, ..., k) do
9:             if not visited[f_i] then
10:                ProvenanceRisk(G, f_i)
11:            end if
12:            npr[v] ← npr[v] + npr[i] + (np[i] × fr(f_i) × fcs(f_v, f_i))/M
13:        end for
14:    end if
15: end procedure
```

Table 3.2: Results of the risk computation for flows in Figure 2.4(b).

	f_1	f_2	f_3	f_4	f_5
$\text{src}(f)$	1	1	1	1	1
$\text{dst}(f)$	1	1	1	1	1
$\text{srv}(f)$	0.143	0.143	0.143	0.143	0.429
$\text{ar}(f)$	0.714	0.714	0.714	0.714	0.81
$N_p(f)$	2	2	1	1	1
$N_t(f)$	4	4	1	1	1
$\text{pr}(f)$	0.995	0.995	0.248	0.250	0
$\widehat{\text{fr}}(f)$	0.278	0.278	0.156	0.157	0.132

Example 5. Table 3.2 shows an example of the intermediate results of our risk score computation for network flows in the graph shown in Figure 2.4(b). For this example, all risk scores are normalised and the value of c_f is set to 0.5. Moreover, the initial risk score of all network flows, $\text{fr}(f)$, is set to 1.

3.3 Risk Score for Hosts

The risk score of a host is computed based on its engagement in risky network activities (see ④ in Figure 3.2). The network activities of a host are specified by its incoming and outgoing network flows within the current time window. Moreover, incoming and outgoing risky flows have different impact which needs to be taken into account in the computation process. For example, network flows which are initiated by a web server inside a DMZ have more influence on the risk level of the server than incoming flows to such web server. In contrast, incoming flows to a internal desktop host have more impact on the risk level of the host than its outgoing flows. Therefore, we allow the network administrator to manipulate this impact factor based on the network topology. We propose the following equation, used in our experimental evaluations:

$$\widehat{\text{hr}}(h) = \frac{c_{in} \sum\limits_{f \in F_{I,h}} \text{fr}(f)}{|F_{I,h}|} + \frac{(1 - c_{in}) \sum\limits_{f \in F_{O,h}} \text{fr}(f)}{|F_{O,h}|} \tag{3.10}$$

where $F_{I,h}$ and $F_{O,h}$ are the set of incoming and outgoing flows of host h respectively (in the current time window), and $|F|$ is the cardinality of the set F.

In equation (3.10), c_{in} is a constant in the range $0 \leq c_{in} \leq 1$ chosen to reflect the impact of incoming flows in the computation of the risk score. For example, if c_{in} has a large value, especially if $c_{in} > 0.5$, we consider the incoming flows to be more risky than the outgoing flows for the host risk computation. Moreover, in this case a higher proportion of the risk score of a network flow is propagated to its destination host. Such larger value of c_{in} can be used, for example, for detecting victims hosts of an attack. On the other hand, if c_{in} has a smaller value, especially if $c_{in} < 0.5$, we consider the outgoing flows to be more risky than the incoming flows for computing the risk score of a host. In addition, in this case a high proportion of the risk score of a network flow is propagated to

Table 3.3: Results of the risk computation for hosts in Table 2.1.

Host	$c_{in} = 0.2$	$c_{in} = 0.5$	$c_{in} = 0.8$
98.114.205.102	0.612	0.500	0.409
192.150.11.111	0.388	0.500	0.591

its source host. If such a lower value of c_{in} is used, since the originator hosts are highly penalised with such a value, our risk computation method will assign a high value of risk to the attackers hosts in an attack. In summary, if c_{in} is larger, the targeted hosts will be assigned higher values of risk; in contrast, for smaller c_{in}, originators will be assigned higher values of risk. Therefore, we allow such a constant to be adjusted by the network administrator[3].

Example 6. Table 3.3 shows the results of our risk score computation for hosts in the example shown in Table 2.1. For this example, the risk scores of network flows are obtained from the previous section, as shown in Table 3.2 (row labeled $\widehat{\text{fr}}(f)$). All the risk scores for hosts are computed using three different values for constant c_{in}: 0.2, 0.5 and 0.8.

3.4 Iterative Algorithm

As we have explained, an iterative algorithm is employed for computing the risk scores for flows and hosts within each time window. Algorithm 3 shows such iterative process; a host and a flow risk vectors (respectively **hr** and **fr**) are inputs from the previous time window and an input vector F is a set of monitored flows for the current time window. As we will show, our algorithm converges to a unique solution regardless of what the initial assignment of the risk values is; however, passing these values from one window to the next greatly reduces the number of iterations till convergence.

Algorithm 3 Iterative algorithm for risk computation within each time window.

1: **procedure** RiskComputation(**hr**, **fr**, F)
2: Create flow dependency graph G from F
3: ComputeNtAndNp(G, 0) using algorithm 1
4: $M \leftarrow \max\{N_t(f) \ : \ f \in F\}$
5: **repeat**
6: Compute $\text{ar}(f)$ for all $f \in F$ using equation (3.2)
7: Compute $\text{npr}(f)$ for all $f \in F$ using algorithm 2
8: Compute $\widehat{\text{fr}}(f)$ for all $f \in F$ using equation (3.9)
9: Compute $\widehat{\text{hr}}(h)$ for all host h involved in F using equation (3.10)
10: **fr** $\leftarrow \widehat{\textbf{fr}}$
11: **hr** $\leftarrow \widehat{\textbf{hr}}$
12: **until** the change of risk values is smaller than the threshold value
13: **Return fr** and **hr**
14: **end procedure**

[3]In our experiments we set $c_{in} = 0.8$ giving more importance to incoming flows than the outgoing ones in computations of risk scores of hosts.

Table 3.4: Risk values for flows and hosts in Table 2.1 after each iteration.

Iteration	Flows					Hosts	
	f_1	f_2	f_3	f_4	f_5	h_1	h_2
0	1	1	1	1	1	1	1
1	0.278	0.278	0.156	0.157	0.132	0.388	0.612
2	0.223	0.223	0.176	0.176	0.202	0.411	0.589
3	0.230	0.230	0.174	0.174	0.192	0.409	0.591
4	0.229	0.229	0.174	0.174	0.194	0.409	0.591
5	0.229	0.229	0.174	0.174	0.193	0.409	0.591

Example 7. Table 3.4 shows how the risk values of flows and hosts are updated in the iterative algorithm for sample flows shown in Table 2.1. For this example, all risk scores are normalised and the constants values are set as $c_f = 0.5$ and $c_{in} = 0.8$. Moreover, in this table, h_1 and h_2 denote hosts 98.114.205.102 and 192.150.11.111, respectively.

3.5 Update Process

As shown in Figure 3.2, the update process is one of the two main modules of our risk computation architecture. This module allows the flows to keep streaming into the risk computation system. In the update process and before starting the risk computation for the next time window w_{i+1}, we first obtain initial risk scores $\bar{\text{hr}}_{i+1}(h)$ of each host h and $\bar{\text{fr}}_{i+1}(f)$ of each flow f, to be used as initial values for the computation of the window w_{i+1}. These risk scores are provided by the update mechanism, as a weighted sum of the corresponding values $\text{hr}_{w_i}(h)$ from the previous window w_i and $\bar{\text{hr}}_i(h)$ from the previous values obtained by the update mechanism for each host h, and in the same manner the initial risk scores $\bar{\text{fr}}_{i+1}(f)$ of each flow f, from the corresponding values $\text{fr}_{w_i}(f)$ and $\bar{\text{fr}}_i(f)$:

$$\bar{\text{hr}}_{i+1}(h) = c_{hu}\text{hr}_{w_i}(h) + (1 - c_{hu})\bar{\text{hr}}_i(h) \tag{3.11}$$

$$\bar{\text{fr}}_{i+1}(f) = c_{fu}\text{fr}_{w_i}(f) + (1 - c_{fu})\bar{\text{fr}}_i(f) \tag{3.12}$$

In the above equations, c_{hu} and c_{fu} are constants with $0 \leq c_{hu} \leq 1$ and $0 \leq c_{fu} \leq 1$ which determine relative importance of the values from the current time window versus previous update values. In other words, if c_{hu} and c_{fu} are large, the risk scores can change fast; if c_{hu} and c_{fu} are small, the risk scores will change more slowly from one window to the next [4].

4 Properties of the Algorithm

As we discussed, algorithm 3 iteratively computes the risk scores for flows and hosts within each time window. In this section, we highlight the properties

[4]In our experiment these constants were to 0.5 to equally reflect the importance of risk scores of current time window and the values from the previous updating process. Moreover, the initial values of risk for all objects are set to one at the very beginning of the operation of our system.

of the iterative procedure in our algorithm; we first prove that the difference between risk scores for either a host or a flow at any two consecutive iterations of Algorithm 3 is bounded. Next, we exploit this property to prove the convergence of the algorithm. We then show that the algorithm converges to a unique risk value for each host and each flow. Finally, we formally model the memory usage and time complexity of the algorithm. The idea for exhibiting the risk discrepancy bounds and convergence is inspired by a similar method presented in [21], where it used in the context of bias and prestige of nodes in trust-based networks.

4.1 Risk Discrepancy Bounds

Lemma 2. *The risk value of provenance, flows and hosts is always in the range* $[0, 1]$.

Proof. At the initial stage of our algorithm all risk values of flows, hosts and provenances are set to 1. Assuming that the lemma thesis is true at t^{th} stage of iteration, we prove that it is also true for the values obtained at stage $t+1$. Since

$$\begin{aligned}\mathrm{npr}^{(t+1)}(f) &= \frac{\mathrm{pr}^{(t+1)}(f)}{M} = \frac{\sum_{p\in t_f} \mathrm{spr}^{(t+1)}(p)}{M} \\ &= \frac{\sum_{p\in t_f} \sum_{\substack{f'\in p \\ f'\neq f}} \mathrm{fr}^{(t)}(f')\mathrm{fcs}(parent(f',p),f')}{M} \\ &\leq \frac{\sum_{p\in t_f} \sum_{\substack{f'\in p \\ f'\neq f}} 1}{M} = \frac{N_t(f)}{\max\{N_t(f') \,:\, f'\in F\}} \leq 1.\end{aligned}$$

Using the above fact, we now have for the risk of flows

$$\begin{aligned}\mathrm{fr}^{(t+1)}(f) &= c_f \mathrm{ar}^{(t)}(f) + (1-c_f)\mathrm{npr}^{(t+1)}(f) \\ &= c_f \frac{\mathrm{hr}^{(t)}(\mathrm{src}(f)) + \mathrm{hr}^{(t)}(\mathrm{dst}(f)) + \mathrm{sr}(\mathrm{srv}(f))}{3} + (1-c_f)\mathrm{npr}^{(t+1)}(f) \\ &\leq c_f \frac{2+\mathrm{sr}(\mathrm{srv}(f))}{3} + (1-c_f) \leq 1.\end{aligned}$$

and, using the above for the risk of hosts,

$$\begin{aligned}hr^{(t+1)}(h) &= \frac{c_{in} \sum\limits_{f\in F_{I,h}} \mathrm{fr}^{(t+1)}(f)}{|F_{I,h}|} + \frac{(1-c_{in}) \sum\limits_{f\in F_{O,h}} \mathrm{fr}^{(t+1)}(f)}{|F_{O,h}|} \\ &\leq \frac{c_{in} \sum\limits_{f\in F_{I,h}} 1}{|F_{I,h}|} + \frac{(1-c_{in}) \sum\limits_{f\in F_{O,h}} 1}{|F_{O,h}|} = 1.\end{aligned}$$

□

Corollary 1. *The maximum difference between the risk values of either a flow or a host between any two iterations is 1.*

Proof. Follows immediately from the fact that all risk values are in the range of [0,1]. □

Lemma 3. *The difference of the risk values of any flow f as well as any host h obtained at two consecutive stages of iteration t and $t+1$ is bounded by an exponential function of t:*

$$\left|\mathrm{fr}^{(t+1)}(f) - \mathrm{fr}^{(t)}(f)\right| \leq \left(1 - \frac{1}{3}c_f\right)^t \tag{4.1}$$

$$\left|\mathrm{hr}^{(t+1)}(h) - \mathrm{hr}^{(t)}(h)\right| \leq \left(1 - \frac{1}{3}c_f\right)^t \tag{4.2}$$

Proof. We prove the lemma by mathematical induction. The risk computation of flow f at stage $t+1$ is given by

$$\begin{aligned}\mathrm{fr}^{(t+1)}(f) = c_f \frac{\mathrm{hr}^{(t)}(\mathrm{src}(f)) + \mathrm{hr}^{(t)}(\mathrm{dst}(f)) + \mathrm{sr}^{(t)}(\mathrm{srv}(f))}{3} \\ + (1-c_f)\frac{\sum_{p\in t_f}\sum_{\substack{f'\in p \\ f'\neq f}} \mathrm{fr}^{(t)}(f')\mathrm{fcs}(\mathit{parent}(f',p),f')}{M}\end{aligned} \tag{4.3}$$

Base Case: We first prove the bound for risk values of flows in the case $t=1$.

$$\begin{aligned}&\left|\mathrm{fr}^{(2)}(f) - \mathrm{fr}^{(1)}(f)\right| = \\ &\left|c_f \frac{\mathrm{hr}^{(1)}(\mathrm{src}(f)) - \mathrm{hr}^{(0)}(\mathrm{src}(f)) + \mathrm{hr}^{(1)}(\mathrm{dst}(f)) - \mathrm{hr}^{(0)}(\mathrm{dst}(f)) + \mathrm{sr}^{(1)}(\mathrm{srv}(f)) - \mathrm{sr}^{(0)}(\mathrm{srv}(f))}{3}\right. \\ &\quad \left. + (1-c_f)(\mathrm{npr}^{(1)}(f) - \mathrm{npr}^{(0)}(f))\right| \\ &\leq \frac{c_f}{3}\left|\mathrm{hr}^{(1)}(\mathrm{src}(f)) - \mathrm{hr}^{(0)}(\mathrm{src}(f))\right| + \frac{c_f}{3}\left|\mathrm{hr}^{(1)}(\mathrm{dst}(f)) - \mathrm{hr}^{(0)}(\mathrm{dst}(f))\right| \\ &\quad + \frac{c_f}{3}\left|\mathrm{sr}^{(1)}(\mathrm{srv}(f)) - \mathrm{sr}^{(0)}(\mathrm{srv}(f))\right| + (1-c_f)\left|\mathrm{npr}^{(1)}(f) - \mathrm{npr}^{(0)}(f)\right| \\ &\leq \frac{c_f}{3} + \frac{c_f}{3} + (1-c_f) = 1 - \frac{1}{3}c_f\end{aligned}$$

Note that $\mathrm{sr}^{(1)}(\mathrm{srv}(f)) - \mathrm{sr}^{(0)}(\mathrm{srv}(f)) = 0$ because the risk values of services remain constant during the risk computation process; also according to Corollary 1, the difference between the risk values is bounded by 1. Now, we similarly prove the base case for the risk of hosts.

$$\begin{aligned}&\left|\mathrm{hr}^{(2)}(h) - \mathrm{hr}^{(1)}(h)\right| = \\ &\left|\frac{c_{in}\sum_{f\in F_{I,h}}\mathrm{fr}^{(2)}(f)}{|F_{I,h}|} + \frac{(1-c_{in})\sum_{f\in F_{O,h}}\mathrm{fr}^{(2)}(f)}{|F_{O,h}|} - \frac{c_{in}\sum_{f\in F_{I,h}}\mathrm{fr}^{(1)}(f)}{|F_{I,h}|} - \frac{(1-c_{in})\sum_{f\in F_{O,h}}\mathrm{fr}^{(1)}(f)}{|F_{O,h}|}\right| \\ &\leq \frac{c_{in}\sum_{f\in F_{I,h}}\left|\mathrm{fr}^{(2)}(f) - \mathrm{fr}^{(1)}(f)\right|}{|F_{I,h}|} + \frac{(1-c_{in})\sum_{f\in F_{O,h}}\left|\mathrm{fr}^{(2)}(f) - \mathrm{fr}^{(1)}(f)\right|}{|F_{O,h}|} \\ &\leq \frac{c_{in}\sum_{f\in F_{I,h}}\left(1-\frac{1}{3}c_f\right)}{|F_{I,h}|} + \frac{(1-c_{in})\sum_{f\in F_{O,h}}\left(1-\frac{1}{3}c_f\right)}{|F_{O,h}|} \\ &\leq c_{in}\left(1-\frac{1}{3}c_f\right) + (1-c_{in})\left(1-\frac{1}{3}c_f\right) = 1 - \frac{1}{3}c_f.\end{aligned}$$

Note that we used the fact that $\sum_{f \in F_{I,h}} 1 = |F_{I,h}|$.

Inductive Step: We assume the bound to be true for iteration t for every flow f and every host h, thus $\left|\mathrm{fr}^{(t+1)}(f) - \mathrm{fr}^{(t)}(f)\right| \leq \left(1 - \frac{1}{3}c_f\right)^t$ and $\left|\mathrm{hr}^{(t+1)}(h) - \mathrm{hr}^{(t)}(h)\right| \leq \left(1 - \frac{1}{3}c_f\right)^t$. We first show that the bound is true for iteration $t+1$ for every flow f as follows:

$$
\begin{aligned}
&\left|\mathrm{fr}^{(t+2)}(f) - \mathrm{fr}^{(t+1)}(f)\right| = \\
&\left| c_f \frac{\mathrm{hr}^{(t+1)}(\mathrm{src}(f)) - \mathrm{hr}^{(t)}(\mathrm{src}(f)) + \mathrm{hr}^{(t+1)}(\mathrm{dst}(f)) - \mathrm{hr}^{(t)}(\mathrm{dst}(f)) + \mathrm{sr}^{(t+1)}(\mathrm{srv}(f)) - \mathrm{sr}^{(t)}(\mathrm{srv}(f))}{3} \right. \\
&\qquad \left. + (1 - c_f) \frac{\sum_{p \in t_f} \sum_{\substack{f' \in p \\ f' \neq f}} \mathrm{fcs}(parent(f', p), f') \left(\mathrm{fr}^{(t+1)}(f') - \mathrm{fr}^{(t)}(f')\right)}{M} \right| \\
&\leq \frac{c_f}{3} \left|\mathrm{hr}^{(t+1)}(\mathrm{src}(f)) - \mathrm{hr}^{(t)}(\mathrm{src}(f))\right| + \frac{c_f}{3} \left|\mathrm{hr}^{(t+1)}(\mathrm{dst}(f)) - \mathrm{hr}^{(t)}(\mathrm{dst}(f))\right| \\
&\qquad + (1 - c_f) \frac{\sum_{p \in t_f} \sum_{\substack{f' \in p \\ f' \neq f}} \left|\mathrm{fr}^{(t+1)}(f') - \mathrm{fr}^{(t)}(f')\right|}{M} \\
&\leq \frac{c_f}{3} \left(1 - \frac{1}{3}c_f\right)^t + \frac{c_f}{3} \left(1 - \frac{1}{3}c_f\right)^t + (1 - c_f) \frac{N_t(f) \left(1 - \frac{1}{3}c_f\right)^t}{M} \\
&\leq \frac{c_f}{3} \left(1 - \frac{1}{3}c_f\right)^t + \frac{c_f}{3} \left(1 - \frac{1}{3}c_f\right)^t + (1 - c_f) \left(1 - \frac{1}{3}c_f\right)^t \\
&= \left(1 - \frac{1}{3}c_f\right)^{t+1}
\end{aligned}
$$

Again, we have used the fact that $\sum_{p \in t_f} \sum_{\substack{f' \in p \\ f' \neq f}} 1 = N_t(f) \leq \max\{N_t(f') : f' \in F\}$. Now we similarly show that the bound is true for iteration $t+1$ for every host h as follows:

$$
\begin{aligned}
&|\mathrm{hr}^{(t+2)}(h) - \mathrm{hr}^{(t+1)}(h)| = \\
&\left| \frac{c_{in} \sum_{f \in F_{I,h}} \mathrm{fr}^{(t+2)}(f)}{|F_{I,h}|} + \frac{(1 - c_{in}) \sum_{f \in F_{O,h}} \mathrm{fr}^{(t+2)}(f)}{|F_{O,h}|} - \frac{c_{in} \sum_{f \in F_{I,h}} \mathrm{fr}^{(t+1)}(f)}{|F_{I,h}|} - \frac{(1 - c_{in}) \sum_{f \in F_{O,h}} \mathrm{fr}^{(t+1)}(f)}{|F_{O,h}|} \right| \\
&\leq \frac{c_{in} \sum_{f \in F_{I,h}} \left|\mathrm{fr}^{(t+2)}(f) - \mathrm{fr}^{(t+1)}(f)\right|}{|F_{I,h}|} + \frac{(1 - c_{in}) \sum_{f \in F_{O,h}} \left|\mathrm{fr}^{(t+2)}(f) - \mathrm{fr}^{(t+1)}(f)\right|}{|F_{O,h}|} \\
&\leq c_{in} \left(1 - \frac{1}{3}c_f\right)^{t+1} + (1 - c_{in}) \left(1 - \frac{1}{3}c_f\right)^{t+1} = \left(1 - \frac{1}{3}c_f\right)^{t+1}
\end{aligned}
$$

□

4.2 Proof of Convergence

Using the above risk discrepancy bound, we now show that the risk scores for both flows and hosts converge.

Theorem 1. *For every flow $f \in F$, sequence $\{\mathrm{fr}^{(t)}(f)\}$, $t \in \mathbb{N}$, and for every host $h \in H$, sequence $\{\mathrm{hr}^{(t)}(h)\}$, $t \in \mathbb{N}$, converge.*

Proof. To show convergence, we will show that for every flow f, sequence $\{\mathrm{fr}^{(t)}(f), n \in \mathbb{N}\}$ and for every host h, sequence $\{\mathrm{hr}^{(t)}(h), n \in \mathbb{N}\}$ are Cauchy sequences. This follows immediately from the bound obtained in Lemma 3; for every flow $f \in F$ and $\forall n, m \in \mathbb{N}$, we have

$$\begin{aligned}
&\left|\mathrm{fr}^{(n+m)}(f) - \mathrm{fr}^{(n)}(f)\right| \\
&\quad\leq \left|\mathrm{fr}^{(n+m)}(f) - \mathrm{fr}^{(n+m-1)}(f)\right| + \cdots + \left|\mathrm{fr}^{(n+1)}(f) - \mathrm{fr}^{(n)}(f)\right| \\
&\quad\leq \left(1 - \frac{1}{3}c_f\right)^{n+m-1} + \cdots + \left(1 - \frac{1}{3}c_f\right)^{n} \\
&\quad= \left(1 - \frac{1}{3}c_f\right)^{n} \times \left[\left(1 - \frac{1}{3}c_f\right)^{m-1} + \cdots + 1\right] \\
&\quad= \left(1 - \frac{1}{3}c_f\right)^{n} \frac{1 - \left(1 - \frac{1}{3}c_f\right)^{m}}{1 - \left(1 - \frac{1}{3}c_f\right)} \\
&\quad\leq \left(1 - \frac{1}{3}c_f\right)^{n} \frac{1}{1 - \left(1 - \frac{1}{3}c_f\right)} \\
&\quad= \frac{3}{c_f}\left(1 - \frac{1}{3}c_f\right)^{n}.
\end{aligned}$$

Similarly, for every host h and $\forall n, m \in \mathbb{N}$, we have

$$\left|\mathrm{hr}^{(n+m)}(h) - \mathrm{hr}^{(n)}(h)\right| \leq \frac{3}{c_f}\left(1 - \frac{1}{3}c_f\right)^{n}.$$

Thus, when n is sufficiently large, for all m the values of $\left|\mathrm{fr}^{(n+m)}(f) - \mathrm{fr}^{(n)}(f)\right|$ and $\left|\mathrm{hr}^{(n+m)}(h) - \mathrm{hr}^{(n)}(h)\right|$ can be made arbitrarily small. The claim now follows from the fact that all Cauchy sequences on reals are convergent. □

We now show that our algorithm converges rapidly.

Lemma 4. *For every value $\varepsilon > 0$ of the threshold, algorithm 3 converges after $N = \delta + \log_{(1-\frac{1}{3}c_f)} \varepsilon$ many steps.*

Proof. Since by the previous Theorem

$$\begin{aligned}
\left|\mathrm{fr}^{(n+m)}(f) - \mathrm{fr}^{(n)}(f)\right| &\leq \frac{3}{c_f}\left(1 - \frac{1}{3}c_f\right)^{n}; \\
\left|\mathrm{hr}^{(n+m)}(h) - \mathrm{hr}^{(n)}(h)\right| &\leq \frac{3}{c_f}\left(1 - \frac{1}{3}c_f\right)^{n},
\end{aligned}$$

the same inequalities hold for the limit as $m \to \infty$; thus, denoting the limits by $\mathrm{fr}^{(\infty)}(f)$ and $\mathrm{hr}^{(\infty)}(f)$ respectively, we obtain

$$\left|\mathrm{fr}^{(\infty)}(f) - \mathrm{fr}^{(n)}(f)\right| \leq \frac{3}{c_f}\left(1 - \frac{1}{3}c_f\right)^n;$$
$$\left|\mathrm{hr}^{(\infty)}(h) - \mathrm{hr}^{(n)}(h)\right| \leq \frac{3}{c_f}\left(1 - \frac{1}{3}c_f\right)^n.$$

Thus, both $\left|\mathrm{fr}^{(\infty)}(f) - \mathrm{fr}^{(n)}(f)\right| \leq \varepsilon$ and $\left|\mathrm{hr}^{(\infty)}(h) - \mathrm{hr}^{(n)}(h)\right| \leq \varepsilon$ whenever $\frac{3}{c_f}\left(1 - \frac{1}{3}c_f\right)^n < \varepsilon$; taking the logarithm of both sides we get that this happens just in case $\log_{(1-\frac{1}{3}c_f)} \frac{3}{c_f} + n \leq \log_{(1-\frac{1}{3}c_f)} \varepsilon$, which proves our claim with $\delta = \log_{(1-\frac{1}{3}c_f)} \frac{c_f}{3}$. □

4.3 Proof of Uniqueness

In this section, we prove that algorithm 3 provides unique solutions for risk scores of both hosts and flows.

Theorem 2. *For any given assignments of the risk values of services, for every flow $f \in F$, sequence $\{\mathrm{fr}^{(t)}(f)\}$, $t \in \mathbb{N}$, and for every host $h \in H$, sequence $\{\mathrm{hr}^{(t)}(h)\}$, $n \in \mathbb{N}$, converge to unique values, independent on the initial values of $\mathrm{fr}^{(0)}(f)$ and $\mathrm{hr}^{(0)}(h)$.*

Proof. We prove the uniqueness of the fixed point in our iterative algorithm using "proof by contradiction". If the algorithm does not provide a unique fixed point, then we have at least two different fixed points which provide different risk scores for both hosts and flows. Assume the provided risk value for flow f (host h) by the first and second fixed points are $\mathrm{fr}_1^*(f)$ and $\mathrm{fr}_2^*(f)$ ($\mathrm{hr}_1^*(h)$ and $\mathrm{hr}_2^*(h)$), respectively. Denote the corresponding normalized flow provenance risk value by $\mathrm{npr}_k^*(f)$ for the k-th fixed point. We first prove the following inequalities by mathematical induction.

$$\forall n \in \mathbb{N},\ \ |\mathrm{fr}_1^*(f) - \mathrm{fr}_2^*(f)| \leq \left(1 - \frac{1}{3}c_f\right)^n; \tag{4.4}$$

$$\forall n \in \mathbb{N},\ \ |\mathrm{hr}_1^*(h) - \mathrm{hr}_2^*(h)| \leq \left(1 - \frac{1}{3}c_f\right)^n. \tag{4.5}$$

Base Case: From equation (4.3) in the fixed point and for the case $n = 1$, we have

$$\begin{aligned}
&|\mathrm{fr}_1^*(f) - \mathrm{fr}_2^*(f)| = \\
&\left|c_f \frac{\mathrm{hr}_1^*(\mathrm{src}(f)) - \mathrm{hr}_2^*(\mathrm{src}(f)) + \mathrm{hr}_1^*(\mathrm{dst}(f)) - \mathrm{hr}_2^*(\mathrm{dst}(f))}{3} + (1 - c_f)(\mathrm{npr}_1^*(f) - \mathrm{npr}_2^*(f))\right| \\
&\leq \frac{c_f}{3}\left|\mathrm{hr}_1^*(\mathrm{src}(f)) - \mathrm{hr}_2^*(\mathrm{src}(f))\right| + \frac{c_f}{3}\left|\mathrm{hr}_1^*(\mathrm{dst}(f)) - \mathrm{hr}_2^*(\mathrm{dst}(f))\right| \\
&\quad + (1 - c_f)\left|\mathrm{npr}_1^*(f) - \mathrm{npr}_2^*(f)\right| \\
&\leq \frac{c_f}{3} + \frac{c_f}{3} + (1 - c_f) = 1 - \frac{1}{3}c_f
\end{aligned}$$

Now, we similarly prove the base case ($n = 1$) for the risk of hosts.

$$
\begin{aligned}
&|\mathrm{hr}_1^*(h) - \mathrm{hr}_2^*(h)| = \\
&\left| \frac{c_{in} \sum\limits_{f \in F_{I,h}} \mathrm{fr}_1^*(f)}{|F_{I,h}|} + \frac{(1-c_{in}) \sum\limits_{f \in F_{O,h}} \mathrm{fr}_1^*(f)}{|F_{O,h}|} - \frac{c_{in} \sum\limits_{f \in F_{I,h}} \mathrm{fr}_2^*(f)}{|F_{I,h}|} - \frac{(1-c_{in}) \sum\limits_{f \in F_{O,h}} \mathrm{fr}_2^*(f)}{|F_{O,h}|} \right| \\
&\leq \frac{c_{in} \sum\limits_{f \in F_{I,h}} |\mathrm{fr}_1^*(f) - \mathrm{fr}_2^*(f)|}{|F_{I,h}|} + \frac{(1-c_{in}) \sum\limits_{f \in F_{O,h}} |\mathrm{fr}_1^*(f) - \mathrm{fr}_2^*(f)|}{|F_{O,h}|} \\
&\leq \frac{c_{in} \sum\limits_{f \in F_{I,h}} \left(1 - \frac{1}{3}c_f\right)}{|F_{I,h}|} + \frac{(1-c_{in}) \sum\limits_{f \in F_{O,h}} \left(1 - \frac{1}{3}c_f\right)}{|F_{O,h}|} \\
&\leq c_{in} \left(1 - \frac{1}{3}c_f\right) + (1-c_{in}) \left(1 - \frac{1}{3}c_f\right) = 1 - \frac{1}{3}c_f.
\end{aligned}
$$

Note that we used the fact that $\sum\limits_{f \in F_{I,h}} 1 = |F_{I,h}|$.

Inductive Step: We assume the equations (4.4) and (4.5) to be true in the case n for every flow f and every host h, thus $|\mathrm{fr}_1^*(f) - \mathrm{fr}_2^*(f)| \leq \left(1 - \frac{1}{3}c_f\right)^n$ and $|\mathrm{hr}_1^*(h) - \mathrm{hr}_2^*(h)| \leq \left(1 - \frac{1}{3}c_f\right)^n$. We first show that inequality (4.4) is true in the case $n+1$ for every flow f as follows:

$$
\begin{aligned}
&|\mathrm{fr}_1^*(f) - \mathrm{fr}_2^*(f)| = \\
&\left| c_f \frac{\mathrm{hr}_1^*(\mathrm{src}(f)) - \mathrm{hr}_2^*(\mathrm{src}(f)) + \mathrm{hr}_1^*(\mathrm{dst}(f)) - \mathrm{hr}_2^*(\mathrm{dst}(f))}{3} + (1-c_f)(\mathrm{npr}_1^*(f) - \mathrm{npr}_2^*(f)) \right| \\
&\leq \frac{c_f}{3} |\mathrm{hr}_1^*(\mathrm{src}(f)) - \mathrm{hr}_2^*(\mathrm{src}(f))| + \frac{c_f}{3} |\mathrm{hr}_1^*(\mathrm{dst}(f)) - \mathrm{hr}_2^*(\mathrm{dst}(f))| \\
&\quad + (1-c_f) |\mathrm{npr}_1^*(f) - \mathrm{npr}_2^*(f)| \\
&\leq \frac{c_f}{3} \left(1 - \frac{1}{3}c_f\right)^n + \frac{c_f}{3} \left(1 - \frac{1}{3}c_f\right)^n + (1-c_f) \frac{N_t(f) \left(1 - \frac{1}{3}c_f\right)^n}{M} \\
&\leq \frac{c_f}{3} \left(1 - \frac{1}{3}c_f\right)^n + \frac{c_f}{3} \left(1 - \frac{1}{3}c_f\right)^n + (1-c_f) \left(1 - \frac{1}{3}c_f\right)^n \\
&= \left(1 - \frac{1}{3}c_f\right)^{n+1}
\end{aligned}
$$

Again, we have used the fact that $\sum_{p \in t_f} \sum\limits_{\substack{f' \in p \\ f' \neq f}} 1 = N_t(f) \leq \max\{N_t(f') : f' \in F\}$. Now we similarly show that the bound is true for the iteration $n+1$ for every host h as follows:

$$
\begin{aligned}
&|\mathrm{hr}_1^*(h) - \mathrm{hr}_2^*(h)| = \\
&\left| \frac{c_{in} \sum\limits_{f \in F_{I,h}} \mathrm{fr}_1^*(f)}{|F_{I,h}|} + \frac{(1-c_{in}) \sum\limits_{f \in F_{O,h}} \mathrm{fr}_1^*(f)}{|F_{O,h}|} - \frac{c_{in} \sum\limits_{f \in F_{I,h}} \mathrm{fr}_2^*(f)}{|F_{I,h}|} - \frac{(1-c_{in}) \sum\limits_{f \in F_{O,h}} \mathrm{fr}_2^*(f)}{|F_{O,h}|} \right| \\
&\leq \frac{c_{in} \sum\limits_{f \in F_{I,h}} |\mathrm{fr}_1^*(f) - \mathrm{fr}_2^*(f)|}{|F_{I,h}|} + \frac{(1-c_{in}) \sum\limits_{f \in F_{O,h}} |\mathrm{fr}_1^*(f) - \mathrm{fr}_2^*(f)|}{|F_{O,h}|} \\
&\leq c_{in} \left(1 - \frac{1}{3} c_f\right)^{n+1} + (1-c_{in}) \left(1 - \frac{1}{3} c_f\right)^{n+1} = \left(1 - \frac{1}{3} c_f\right)^{n+1}
\end{aligned}
$$

Now, we proved the inequalities hold for any $n \in \mathbb{N}$; thus, we obtain

$$
\begin{aligned}
|\mathrm{fr}_1^*(f) - \mathrm{fr}_2^*(f)| &\leq \lim_{n \to \infty} \left(1 - \frac{1}{3} c_f\right)^n = 0; \\
|\mathrm{hr}_1^*(h) - \mathrm{hr}_2^*(h)| &\leq \lim_{n \to \infty} \left(1 - \frac{1}{3} c_f\right)^n = 0.
\end{aligned}
$$

The above inequalities are possible if and only if $\mathrm{fr}_1^*(f) = \mathrm{fr}_2^*(f)$ and $\mathrm{hr}_1^*(h) = \mathrm{hr}_2^*(h)$, which proves our claim.

□

4.4 Memory Usage and Complexity Analysis

In order to formulate the time and space complexity of our iterative risk computation in algorithm 3, we assume that the size of the window is $w = |F|$.

The memory required the algorithm mainly consists of the memory needed for storing of all risk values for hosts and flows in the window as well as the temporary values used by the recursive implementation of the DFS-like algorithm for computing risk of flow provenances. Clearly, the memory required for keeping the risk scores for hosts and flows including the intermediate and temporary risk scores in a window with size w is in $O(w)$. Note that the number of hosts is less than the number of flows in a window. Also, the space complexity of algorithm 2 is in $O(w)$ which is equal to the space complexity of a DFS algorithm in general [7]. Moreover, the space complexity for maintaining the flow dependency graph is in $O(|V| + |E|)$ by using an adjacency matrix where $|V|$ and $|E|$ are the number of nodes and edges in the graph, respectively. Although the maximum number of edges in a DAG is $\frac{|V|(|V|-1)}{2}$, the results of our experiments over two real datasets show that the number of edges in the flow dependency graph is around $|E| = O(2 * |V|)$ (see Section 5.2). Thus, the storage complexity of algorithm 3 is in $O(w)$.

The time complexity of the risk computation for all flow attributes, hosts, and the aggregation of the risks of flow attributes and flow provenance in a single iteration is in $O(w)$. Moreover, the computation complexity of risk normalisation is in $O(w)$ as the normalisation is done through dividing each risk value by the sum of all risk scores.

The risk of the flow provenance in Section 3.2 is computed by algorithm 2 which is a DFS-like algorithm over the flow dependency graph. Generally, the

complexity of a DFS algorithm is in $O(|V|+|E|)$ where $|V|$ and $|E|$ are the number of nodes and edges in the graph, respectively [7]. Since the flow dependency graph is sparse, the complexity of the risk computation for the flow provenance is actually linear in the number of flows. Thus, in total, each iteration in our algorithm requires $O(w)$ steps, and for k iterations, the total running time for the iterative algorithm is in $O(k \times w)$. The fact that the total number of iterations is logarithmic in the value of the threshold guarantees the efficiency of our method.

5 Experimental Evaluation

In this section we present the results of a performance evaluation of our system in order to validate its effectiveness and efficiency.

5.1 Experimental Environment

Our experiments were conducted on two public datasets which include two attack scenarios. To evaluate the effectiveness, we performed our risk computation on both of these datasets and show that the model assigns high risk scores to the victims and attackers which are involved in the attack scenarios. To evaluate the efficiency, we measured the elapsed time for our risk assessment process and two other recent models based on PageRank and HITS algorithms [30, 29], and we show that our model is superior to those two methods in terms of computational complexity. Moreover, we show that the performance of our approach is adequate for handling high throughput networks.

All the experiments have been conducted on an iMac PC with 2.00GHz Intel Core 2 Duo processor and 4GB RAM running Ubuntu 12.04 LTS. The program code has been written in Java with JDK 1.7. Table 5.1 summarizes the experimental parameters which are used for all experiments.

Table 5.1: Experimental parameters.

Parameter	Value
Causality time interval	600 seconds
# of flows within a time window	2000
Length of sliding time window (# of flows)	400
The source of risky ports	EMSISOFT
The maximum input cache size (# of flows)	10000
Accuracy threshold in iterative algorithm	0.001
Constants c_f, c_{hu} and c_{fu}	0.5
Constant c_{in}	0.8
Constant α	1

Honeynet Dataset

In order to evaluate the effectiveness of our model, we applied the model to the public traces which were captured by the Honeynet project, Scan 18 [5]. This dataset includes an attack scenario which consists of a sequence of attacker activities including scanning, compromising, downloading and installing a Rootkit, and sending spam emails. In this attack scenario, the attacker compromised a local honeypot machine with IP address 172.16.1.108 using at least two different stepping stones to scan and attack the network. Moreover, attacker exploited a number of different IP addresses including 211.185.125.124, 211.180.229.190, 193.231.236.41, 216.136.129.14, and 209.61.188.33 [29].

MIT Lincoln Dataset

We also applied our risk computation model to the MIT Lincoln dataset [9] which is used extensively for evaluating intrusion detection systems. Although there are several reservations regarding the accuracy of this dataset [19, 18], to our knowledge, no further datasets have been released.

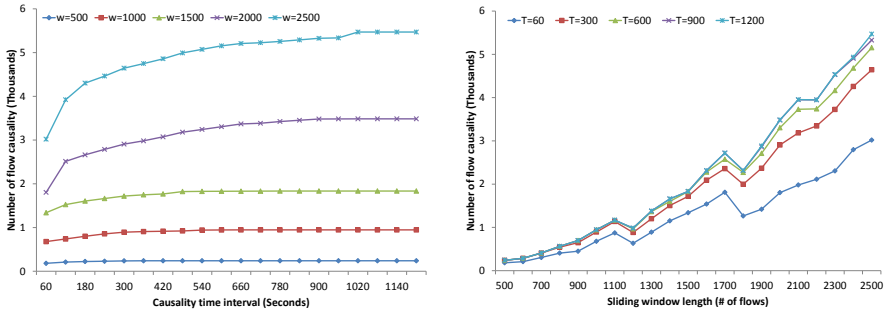
The MIT Lincoln dataset includes a Distributed Denial of Service (DDoS) attack scenario and was originally proposed by a DARPA project for evaluating Intrusion Detection Systems. The data file was collected over a span of approximately 3 hours on Tuesday, 7 March 2000, from 9:25 AM to 12:35 PM and the five phases of the attack scenario were [9]:

- IPsweep of the network from a remote site
- Probe of live IP's to look for the sadmind daemon running on Solaris hosts
- Breaking via the sadmind vulnerability, both successful and unsuccessful on those hosts
- Installation of the trojan mstream DDoS software on three hosts at the network
- Launching the DDoS

Based on the above steps, the attacker installs hacking tools on three machines inside the network with IP addresses 172.16.115.20, 172.16.112.50, and 172.16.112.10. After compromising these three victims, the attacker performs a DDoS attack from all these nodes to the victim machine 131.84.1.31 by flooding packets. The statistical overview of the dataset is presented in Table 5.2.

Table 5.2: The MIT Lincoln flow statistics.

The size of the pcap file	117M
Number of packets	649787
Capture duration	11652 seconds
Data bit rate	76680.78 bits/sec
Number of nodes (hosts)	34521
Number of flows	103006



(a) (b)

Figure 5.1: Average number of flow causalities.

5.2 Sensitivity Analysis

Beyond investigating the effectiveness and efficiency of our risk assessment approach, we also measured the sensitivity of the results with respect to the risk computation parameters: the length of the sliding time window and the causality time interval (Section 2.2). Both these parameters impact the efficiency of our approach because the number of flow causalities depends on the parameter. Clearly, the number of causalities identifies the number of edges in the flow dependency graph which is a significant parameter for the computational complexity of our risk computation algorithm, as shown in Section 4.4. Therefore, in this section we investigate the number of flow causalities according to the different values for these two parameters by running experiments over a subset of the MIT Lincoln dataset including its first 10000 network flows.

Figure 5.1 compares the average number of flow causalities (edges in the flow dependency graph) on the MIT Lincoln dataset with respect to both the sliding window length and causality time interval. One can see in Figure 5.1(a) that by increasing the causality interval time, we achieve a stable number of flow causalities for each different value of sliding window length. This can be explained by the fact that a very large value of the causality interval time exceeds the maximum time interval between two network flows within the window. Since we defined the sliding window based on the number of network flows, an administrator can estimate a maximum threshold for the causality time interval according to the window length and the throughput of the network.

Figure 5.1(b) confirms that large values of the causality time interval similarly increase the number of flow causalities. Clearly, an administrator may choose the time interval value $T = 600$ as it generates a large enough number of flow causalities. Moreover, the sliding window length can be selected based on the available memory and processing power.

5.3 Effectiveness

We evaluated the effectiveness of our solution by running the system on both Honeynet Scan 18 and Lincoln datasets. The Honeynet dataset has only 47 flows and our tool performed the risk analysis in a single window; experimental parameters are presented in Table 5.1. In order to detect the victim hosts and flows, we set c_{in} to 0.8 to give more importance to incoming flows than outgoing ones in computations of risk scores of hosts. The top 10 high risk hosts and

flows ranked by our tool are reported in Table 5.3. In this table, the flows and hosts indicated in bold are the actual victims in the attack scenario according to the Honeynet dataset. These results show that our system assigned high risk values to the main victim with IP address 172.16.1.108. Moreover, our detected third high risk host with IP address 172.16.1.103 is the next victim which responded to SYN scan of the attacker in the dataset [5]. As one can see in this table, the risk of the first host is very high compared with the risk of the second victim. This is due to the fact that in the attack the host 211.185.125.124 launched a sequence of a "RPC GETPORT Call" followed by the actual buffer-overflow attack, first to 172.16.1.103 and then 172.16.1.108. Moreover, the host 172.16.1.103 was immune from this attack while the host 172.16.1.108 has been affected by it [5].

In order to detect the hosts that are originators of attacks in the Honeynet dataset, we set c_{in} to 0.2. Table 5.4 shows the top 10 high risk hosts and flows, ranked by our tool in this experiment. In this table, the hosts indicated in bold are the actual originators of the attack scenario in the Honeynet dataset. Our tool ranked four out of five attackers (except 216.136.129.14) in the top 7 hosts (see column 3 and 4). This can be explained by the fact that by setting a low value for parameter c_{in}, our tool assigns a higher level of risk to the hosts which initiate the highest risk flows.

Table 5.3: Risk evaluation results for Honeynet Scan 18 dataset with $c_{in} = 0.8$.

High risk flows		High risk hosts	
Risk	Flow	Risk	Host
0.0599	172.16.1.108:1026:193.231.236.41:21:TCP	0.3250	**172.16.1.108**
0.0517	172.16.1.108:1029:209.61.188.33:25:TCP	0.1518	211.185.125.124
0.0498	172.16.1.108:1028:216.136.129.14:25:TCP	0.1164	**172.16.1.103**
0.0480	211.185.125.124:790:172.16.1.108:111:UDP	0.1060	193.231.236.41
0.0480	211.185.125.124:3500:172.16.1.108:111:TCP	0.0486	209.61.188.33
0.0453	172.16.1.108:931:211.185.125.124:791:UDP	0.0399	216.136.129.14
0.0448	172.16.1.108:39168:211.185.125.124:4450:TCP	0.0394	172.16.1.107
0.0427	193.231.236.41:1522:172.16.1.108:113:TCP	0.0266	216.168.224.69
0.0427	193.231.236.41:1519:172.16.1.108:113:TCP	0.0237	172.16.1.106
0.0427	193.231.236.41:1516:172.16.1.108:113:TCP	0.0232	172.16.1.105

In order to evaluate the effectiveness of our risk assessment methodology for very large datasets, we applied our tool to the MIT Lincoln dataset. In order to perform a risk assessment for a dataset of size 103006 flows (see Table 5.2) and with a window size of 2000 flows with 20% sliding factor (see Table 5.1), the tool utilized 65 sliding time windows. At the end of each time window w_i, $1 \leq i \leq 65$, we have partial risk scores $\mathrm{hr}_{w_i}(h)$ and $\mathrm{fr}_{w_i}(f)$ for hosts and flows, respectively. In practice, such results, obtained in real time, would be used by the network administrator to monitor for possible malicious activities. We now evaluate these results for each time window as well as the final results after finishing the risk computation for all of the 65 windows.

Table 5.4: Risk evaluation results for the Honeynet Scan 18 dataset with $c_{in} = 0.2$.

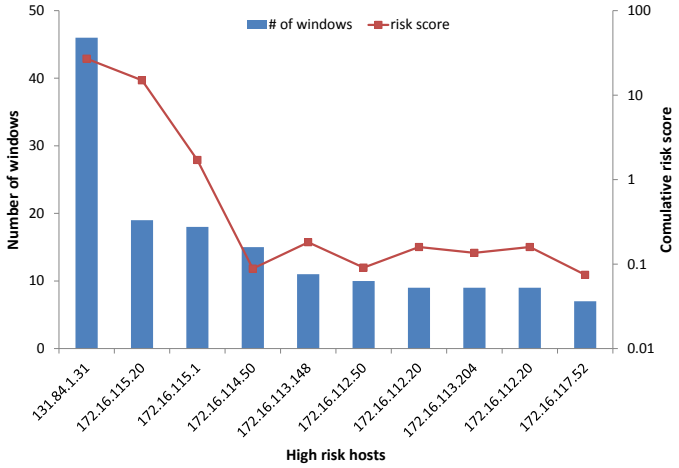
High risk flows		High risk hosts	
Risk	Flow	Risk	Host
0.0533	211.185.125.124:790:172.16.1.108:111:UDP	0.3062	**211.185.125.124**
0.0533	211.185.125.124:3500:172.16.1.108:111:TCP	0.2897	172.16.1.108
0.0506	172.16.1.108:931:211.185.125.124:791:UDP	0.0990	**193.231.236.41**
0.0498	172.16.1.108:39168:211.185.125.124:4450:TCP	0.0783	**211.180.229.190**
0.0487	172.16.1.108:1026:193.231.236.41:21:TCP	0.0564	172.16.1.103
0.0416	172.16.1.108:1029:209.61.188.33:25:TCP	0.0557	203.111.78.182
0.0408	211.185.125.124:789:172.16.1.103:111:UDP	0.0307	**209.61.188.33**
0.0408	211.185.125.124:3495:172.16.1.103:111:TCP	0.0108	65.195.31.2
0.0390	172.16.1.108:1028:216.136.129.14:25:TCP	0.0097	172.16.1.107
0.0377	172.16.1.103:32773:211.185.125.124:790:UDP	0.0086	172.16.1.106

Figure 5.2 shows the highest risk hosts detected by our system, based on the results in each time window and with respect to two different values for parameter c_{in}. The bar charts in this figure show for each host in how many windows (out of all 65 windows) the host was detected as a high risk host. The line charts in this figure shows the cumulative risk computed for the highest risk hosts. One can see in Figure 5.2(a) that by setting c_{in} to 0.8 and in both metrics, our risk assessment model detected the main victim as the highest risk host (the host with IP address 131.84.1.31). The highest values of risk for address 131.84.1.31 in both experiments can be the result of the huge number of flows created during the DDoS attack scenario to the host.

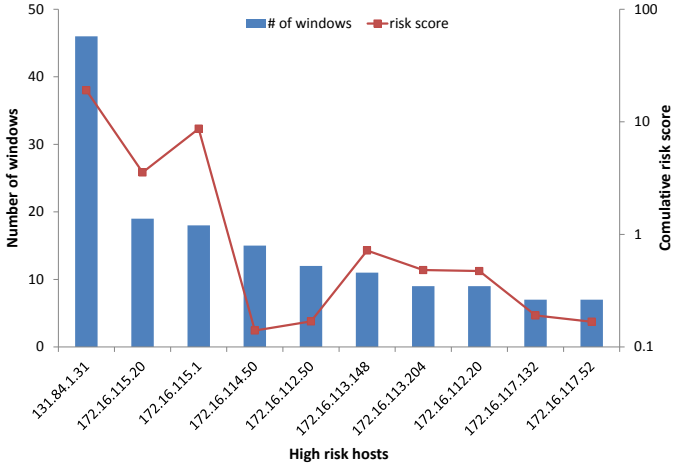
Similar to the previous experiment on the Honeynet dataset, in order to detect the originators of the DDoS attack in the MIT Lincoln dataset, we set c_{in} to 0.2. As can be seen in Figure 5.2(b), our tool ranked two out of three attackers including 172.16.115.20 and 172.16.112.50 (except 172.16.112.10) in the top 20 hosts out of about 3500 hosts (see Table 5.2). Interestingly, both HITS and PageRank algorithms failed to rank the third victim (172.16.112.10) in the top ranked hosts [30].

Although, the results validate the effectiveness of our risk assessment method for small and large networks, our tool could not rank the attacker address 216.136.129.14 in the Honeynet dataset and 172.16.112.10 in the MIT Lincoln dataset among the top 10 high risk hosts. We believe that a better initial values for our tool's parameters including risk level for risky services as well as constant parameters (c_f, c_{hu}, c_{fu} and c_{in}) based on prior knowledge about the network topology would improve the accuracy of our attacker and victim detection.

It is important to notice that the main objective of our risk assessment model is to rank the risk of network activities and therefore, the proposed solution does not replace the functions of an intrusion detection system (IDS). However, results from our risk assessment tool can be provided in real time to security tools with deep inspection capabilities such as an IDS [24]. The IDS can then execute specific actions to determine whether there are actual intrusions or other



(a) $c_{in} = 0.8$.



(b) $c_{in} = 0.2$.

Figure 5.2: High risk hosts in the MIT Lincoln dataset.

attacks.

5.4 Efficiency

We compared our approach with the approach in [30] with respect to effectiveness. The comparison was made using the Honeynet dataset. The comparison results show that both approaches gave the highest risk values to attacker and victim nodes and flows. However, our approach shows a clear advantage in terms of efficiency.

To better explain the efficiency of our methodology, remember that we use only one dependency graph for computing risk scores of both hosts and flows. On the other hand, the approach in [30] runs some link analysis algorithms on two graphs, one for hosts and another for flows. However, the efficient computation of PageRank and HITS algorithms over even one very large dependency graph for a large scale network is a great challenge [2], let alone on two huge graphs.

The sliding time window implementation for input flows in our risk compu-

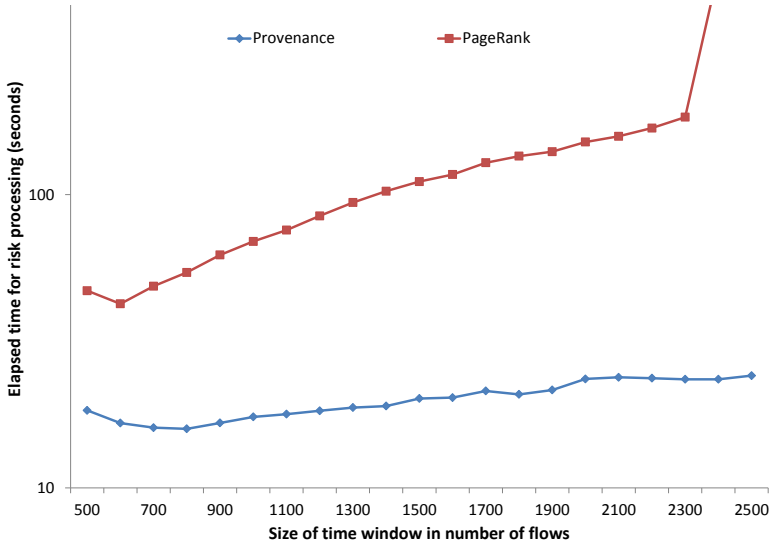
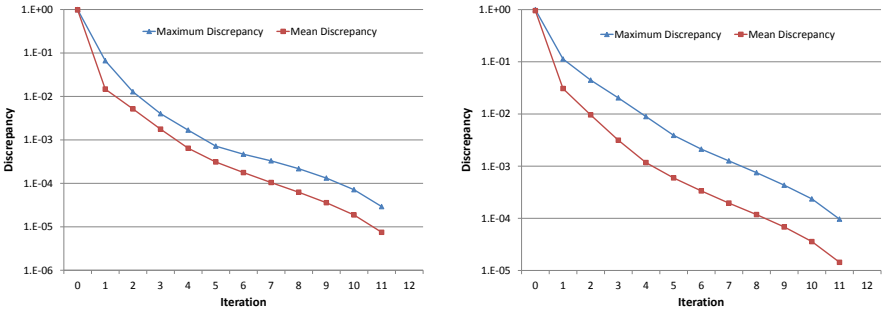


Figure 5.3: Elapsed time for risk computation techniques based on the size of time window.

tation allows our approach to be deployed as an online monitoring tool for large scale networks. Moreover, for achieving high efficiency, a network administrator can alter the performance parameters of the tool based on the hardware and the network throughput. The other advantage of our methodology stems from the use of sliding, overlapping windows; while this reduces the computation requirements for each window, our methodology does not suffer in effectiveness due to the fact that information from each new window is combined with the information obtained from the past windows.

We quantify the efficiency of our tool by analysing its memory usage and processing time. The main parameter which influences the memory usage of the system is the maximum window size for the flow stream. Thus, we evaluated the processing time (the elapsed CPU time) of our provenance-aware algorithm along with the risk assessment approaches proposed in [29, 30] based on different size of the time window. Authors in [29] proposed a method based on PageRank algorithm to evaluate the risk scores of network flows in a flow dependency graph, while they suggested a host dependency graph for computing the risk scores of hosts using the similar algorithm. Therefore, in order to evaluate the efficiency of their approach, we applied the PageRank algorithm on both flow and host dependency graphs and then considered the sum of the elapsed times for the efficiency of the approach (PageRank algorithm). Figure 5.3 shows the results of this experiment.

From Figure 5.3, we can see that in the algorithms proposed in [29, 30] the elapsed time for computing the risk scores of hosts and flows by applying the PageRank algorithm increases significantly as the size of time window increases (the red line). This can be explained by the fact that the size of flow dependency graph is directly dependent on the size of the time window and the high processing time comes from the application of PageRank to a very large graph. However, our provenance-aware mechanism computes both the scores of hosts and flows for such time window in a reasonable processing time (the blue line). Moreover, the results in Figure 5.3 show that the processing time of our



(a) Flow risk discrepancies.

(b) Host risk discrepancies.

Figure 5.4: Exponentially decreasing discrepancies.

method increases much slower for large time windows and thus can handle high throughput networks.

5.5 Analysis of Discrepancy and Convergence

In this section, we perform a set of experiments to analyze the properties of our iterative algorithm in terms of discrepancy and convergence. Thus, we investigate two types of discrepancies for both the risk values of flows and hosts computed in each iteration of algorithm 3 over the Honeynet Scan 18 dataset. For each of flows and hosts risk values, we define the maximum discrepancy by choosing the worst-case discrepancy for all flows and hosts, respectively. Therefore, the maximum discrepancy at iteration l is computed as follows:

$$discrepancy_{\mathrm{fr}}^{(l)} = \max\left\{\left|\widehat{\mathrm{fr}}^{(\infty)}(f) - \widehat{\mathrm{fr}}^{(l)}(f)\right| \ : \ f \in F\right\}$$

$$discrepancy_{\mathrm{hr}}^{(l)} = \max\left\{\left|\widehat{\mathrm{hr}}^{(\infty)}(h) - \widehat{\mathrm{hr}}^{(l)}(h)\right| \ : \ h \in H\right\}$$

We also define the mean discrepancy of risk values over all flows and hosts as follows:

$$discrepancy_{\mathrm{fr}}^{(l)} = \frac{1}{|F|}\sum_{f\in F}\left|\widehat{\mathrm{fr}}^{(\infty)}(f) - \widehat{\mathrm{fr}}^{(l)}(f)\right|$$

$$discrepancy_{\mathrm{hr}}^{(l)} = \frac{1}{|H|}\sum_{h\in H}\left|\widehat{\mathrm{hr}}^{(\infty)}(h) - \widehat{\mathrm{hr}}^{(l)}(h)\right|$$

where H denotes the set of all hosts in the current time window. Figure 5.4 illustrates how the aforementioned discrepancy decline for both flow and host risk values. Clearly, the algorithm converges after 12 iterations.

6 Related Work

Work related to our research falls into three categories: dependency discovery among network traffic, static risk assessment and data provenance management.

There are a number of papers investigating how to discover potential dependencies among network flows [6, 13, 14, 26]. Chen et al. in [6] introduced the Orion system that discovers dependencies for enterprise applications by using

packet headers and timing information. Iliofotou et al. in [13] proposed the use of Traffic Dispersion Graphs (TDGs) as a way to monitor, analyze, and visualise network traffic by modelling the hosts as a social network. The key contribution in [14] is a novel statistical rule mining solution, called eXpose, to extract significant communication patterns in a packet trace. Savilla and Ou in [26] presented AssetRank, a generalisation of PageRank algorithm, which automatically digests the dependency relations in an attack graph as well as the baseline information of the vulnerability attributes to compute the relative importance of attacker assets. Halappanavar et al. in [12] proposed a networks-of-networks (NoN) model of interactions between heterogeneous entities for an enterprise cyber system. Moreover, they showed how the graph-theoretic method can be applied on the top their model for detection of critical nodes and their dependencies, reachability analysis, models for cascading failures and subgraph pattern mining. While these approaches exploited techniques for dependency analysis on network activities, our method employs provenance relations among network flows as well as interdependency relation between hosts and flows in order to detect high risk hosts and flows.

The work most closely related to our research is by Wang et al. [28, 29, 30], which aims at risk assessment for hosts and flows by employing link analysis algorithms such as PageRank and HITS. They introduced flow dependency graph and applied PageRank and HITS algorithms to the graph in order to obtain risk scores of network flows [28, 29]. Due to the huge number of flows in a high throughput network and high computational cost of these algorithms on a very large graph, the efficiency of such an approach is a significant challenge. Therefore, they proposed the notion of host dependency graph [30] and showed that the new graph has a lower number of nodes and edges than the flow dependency graph. However, by applying the link analysis algorithms on the host dependency graph, such approach only obtains risk scores for hosts, without any information about risk scores of flows. We have proposed the idea of flow provenance along with interdependency relationship between hosts and flows in order to evaluate efficiently the risk of both of them.

There are lots of studies on network and information security risk assessments. Ben Mahmoud et al. in [17] proposed a security risk assessment framework to measure quantitatively the risk propagation among nodes connected to the network through taking into account the interconnection between nodes. Rahman et al. in [22] presented a qualitative network security risk analysis using declarative logic. They formalized transitive reachability to compute exposure of vulnerabilities on hosts and then employed their risk assessment model to synthesize necessary firewall rules and host replacements. Both of the above risk assessment methodologies are based on static network information including hosts vulnerabilities, host impact and connectivity among them. Thus, they do not consider dynamic risky behavior of hosts; however we modeled dynamic risk propagation among network activities through modeling the risk provenance of the network flows. Clearly, the result of these static risk assessment methods can be used as a priori knowledge about the risk scores of hosts and services in our risk assessment framework.

A large number of approaches have been proposed for trust frameworks and data provenance management [4, 8, 16, 27], but none of them dealt with risk assessment of network flows based on flow provenance. Dai et al. in [8], proposed a provenance-aware trust model for data management which takes into

account various parameters that may affect the data trustworthiness including data similarity, data conflict, path similarity and data deduction. Moreover, they considered the inter-dependency of trustworthiness between data items and the appropriate data provider. Also, they enhanced their trust model for sensor networks where the information keeps streaming into the system [16]. Our idea for proposing a provenance-aware model for risk assessment on network flows is inspired by the provenance-based trust model proposed by these two approaches. Etuk et al. in [11] proposed *TAF*, a trust assessment framework for streaming information to estimate the quality of inferences under information uncertainty. They also reported new challenges for investigating trust over streaming data such as trade-offs between quality and latency. An interesting research issue that we mention as a part of future work is to improve the efficiency of our risk assessment framework through the ideas employed in the trust frameworks for streaming data.

To the best of our knowledge, no existing work considers the provenance and interdependency between hosts and flows in order to assess risk on network activities. Different from the existing works, in this paper, we employ a novel, effective and efficient risk assessment solution for evaluating both risk scores of hosts and flows simultaneously.

7 Conclusions

This paper has presented a novel risk assessment method for hosts and flows which takes into account the interdependency between the risk of hosts and flows as well as the flow provenance. The update mechanism as an integral part of the proposed risk assessment solution allows a deployment of our approach in real time. Besides proving convergence of our iterative algorithm and providing analytic estimates for its performance, we have also evaluated the effectiveness and efficiency of our method by performing experiments on two publicly available datasets. Results show that our method is effective for assigning high risk scores to hosts and flows involved in attack scenarios as well as efficient in terms of processing time for performing risk assessment in a high throughput network.

As future work, we plan to extend our risk assessment framework to data streaming. In addition, we plan to integrate our framework into OpenFlow and Software Defined Network (SDN) architectures [20] which can improve the deployment of our system as an online monitoring system.

Bibliography

[1] Ahmad Almulhem and Issa Traore. A survey of connection-chains detection techniques. In *Communications, Computers and Signal Processing, 2007. PacRim 2007. IEEE Pacific Rim Conference on*, pages 219–222, 2007.

[2] Pavel Berkhin. A survey on PageRank computing. *Internet Mathematics*, 2(1):73–120, 2005.

[3] Allan Borodin, Gareth O. Roberts, Jeffrey S. Rosenthal, and Panayiotis Tsaparas. Link analysis ranking: algorithms, theory, and experiments. *ACM Trans. Internet Technol.*, 5(1):231–297, February 2005.

[4] Peter Buneman, Sanjeev Khanna, and Tan Wang-Chiew. Why and where: A characterization of data provenance. In Jan Van den Bussche and Victor Vianu, editors, *Database Theory ICDT 2001*, volume 1973 of *Lecture Notes in Computer Science*, pages 316–330. Springer Berlin / Heidelberg.

[5] The Honeynet Project Challenges. Scan 18 and challenge 1 of the forensic challenge 2010. `http://www.honeynet.org/challenges`, 2012. [Online; accessed 1-August-2013].

[6] Xu Chen, Ming Zhang, Z. Morley Mao, and Paramvir Bahl. Automating network application dependency discovery: experiences, limitations, and new solutions. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, OSDI'08, pages 117–130, 2008.

[7] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition.* The MIT Press, 3rd edition, 2009.

[8] Chenyun Dai, Dan Lin, Elisa Bertino, and Murat Kantarcioglu. An approach to evaluate data trustworthiness based on data provenance. In *Proceedings of the 5th VLDB workshop on Secure Data Management*, SDM '08, pages 82–98, 2008.

[9] MIT Lincoln Laboratory: Cyber Systems & Technology: DARPA Intrusion Detection. Lincoln laboratory scenario (DDoS) 1.0, 2012.

[10] Emsisoft. Emsisoft portlist - all known TCP and UDP ports of malware, trojans, spyware, viruses. `http://www.emsisoft.com/en/kb/portlist/`, 2012. [Online; accessed 1-October-2012].

[11] Anthony Etuk, Timothy J. Norman, Chatschik Bisdikian, and Mudhakar Srivatsa. TAF: A trust assessment framework for inferencing with uncertain streaming information. In *5th International Workshop on Information Quality and Quality of Service for Pervasive Computing*, pages 475–480, 2013.

[12] Mahantesh Halappanavar, Sutanay Choudhury, Emilie Hogan, Peter Hui, John R. Johnson, Indrajit Ray, and Lawrence B. Holder. Towards a network-of-networks framework for cyber security. In *Intelligence and Security Informatics (ISI), 2013 IEEE International Conference on*, pages 106–108, 2013.

[13] Marios Iliofotou, Prashanth Pappu, Michalis Faloutsos, Michael Mitzenmacher, Sumeet Singh, and George Varghese. Network monitoring using traffic dispersion graphs (TDGs). In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, IMC '07, pages 315–320, 2007.

[14] Srikanth Kandula, Ranveer Chandra, and Dina Katabi. What's going on?: learning communication rules in edge networks. *SIGCOMM Comput. Commun. Rev.*, 38(4):87–98, August 2008.

[15] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, September 1999.

[16] Hyo-Sang Lim, Yang-Sae Moon, and Elisa Bertino. Provenance-based trustworthiness assessment in sensor networks. In *Proceedings of the Seventh International Workshop on Data Management for Sensor Networks*, DMSN '10, pages 2–7, 2010.

[17] Mohamed Slim Ben Mahmoud, Nicolas Larrieu, and Alain Pirovano. Quantitative risk assessment to enhance aeromacs security in sesar. In *Integrated Communications, Navigation and Surveillance Conference (ICNS), 2012*, pages C7–1–C7–15, 2012.

[18] Matthew V. Mahoney and Philip K. Chan. An analysis of the 1999 DARPA/Lincoln laboratory evaluation data for network anomaly detection. In *In Proceedings of the Sixth International Symposium on Recent Advances in Intrusion Detection*, pages 220–237. Springer-Verlag, 2003.

[19] John McHugh. The 1998 Lincoln laboratory ids evaluation. In *Recent Advances in Intrusion Detection*, volume 1907 of *Lecture Notes in Computer Science*, pages 145–161. Springer Berlin Heidelberg, 2000.

[20] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008.

[21] Abhinav Mishra and Arnab Bhattacharya. Finding the bias and prestige of nodes in networks based on trust scores. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 567–576, New York, NY, USA, 2011. ACM.

[22] Mohammad Ashiqur Rahman and Ehab Al-Shaer. A formal approach for network security management based on qualitative risk analysis. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pages 244–251, 2013.

[23] Mohsen Rezvani, Aleksandar Ignjatovic, Elisa Bertino, and Sanjay Jha. Provenance-aware security risk analysis for hosts and network flows. In *Network Operations and Management Symposium, 2014. NOMS 2014. IEEE/IFIP*, May 2014.

[24] Martin Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX conference on System administration*, LISA '99, pages 229–238, Berkeley, CA, USA, 1999. USENIX Association.

[25] G. Sadasivan, N. Brownlee, B. Claise, and J. Quittek. Architecture for IP Flow Information Export. RFC 5470 (Informational), March 2009. Updated by RFC 6183.

[26] Reginald E. Sawilla and Xinming Ou. Identifying critical attack assets in dependency attack graphs. In *Proceedings of the 13th European Symposium on Research in Computer Security: Computer Security*, ESORICS '08, pages 18–34, 2008.

[27] Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. A survey of data provenance in e-science. *SIGMOD Rec.*, 34(3):31–36, September 2005.

[28] Shaonan Wang, Radu State, Mohamed Ourdane, and Thomas Engel. FlowRank: ranking NetFlow records. In *Proceedings of the 6th International Wireless Communications and Mobile Computing Conference*, IWCMC '10, pages 484–488, 2010.

[29] Shaonan Wang, Radu State, Mohamed Ourdane, and Thomas Engel. Mining netflow records for critical network activities. In *Proceedings of the Mechanisms for autonomous management of networks and services, and 4th international conference on Autonomous infrastructure, management and security*, AIMS'10, pages 135–146, 2010.

[30] Shaonan Wang, Radu State, Mohamed Ourdane, and Thomas Engel. RiskRank: Security risk ranking for IP flow records. In *Network and Service Management (CNSM), 2010 International Conference on*, pages 56 –63, oct. 2010.