Using Network Tainting to Bound the Scope of Network Ingress Attacks

Peter Mell
National Institute of Standards and Technology
Gaithersburg, MD
peter.mell@nist.gov

Richard E Harang
U.S. Army Research Laboratory
Adelphi, MD
ICF International, Baltimore, Maryland
richard.e.harang.ctr@mail.mil

***Abstract*— This research describes a novel security metric, network taint, which is related to software taint analysis. We use it here to bound the possible malicious influence of a known compromised node through monitoring and evaluating network flows. The result is a dynamically changing defense-in-depth map that shows threat level indicators gleaned from monotonically decreasing threat chains. We augment this analysis with concepts from the complex networks research area in forming dynamically changing security perimeters and measuring the cardinality of the set of threatened nodes within them. In providing this, we hope to advance network incident response activities by providing a rapid automated initial triage service that can guide and prioritize investigative activities.**

Keywords- network tainting, complex networks, scale-free, security

I. INTRODUCTION

There exists a significant body of literature on software taint analysis [1]. In our work, we abstract this concept away from the host and apply it to an entire network. Our approach, called network tainting, uses internal network flow data to map out the possible influence of a known malicious node on the rest of the network.

This can be useful in situations where traditional security tools detect a network intrusion resulting in the compromise of an internal node. There exists some window of time between the penetration of the target node and actions to remediate or quarantine. During this window, it is important to know whether or not the attacker used this point of leverage in the network to maliciously access other nodes or compromise additional hosts through lateral movement[1]. Traditional security tools may not detect this due to use of normal privileges of the compromised host, use of zero day exploits, use of exploits not visible to existing security tools, or through attacks occurring within unmonitored interior portions of the network[2]. For example, a host that has been compromised at the user level may then be used to place additional malware in a network share that the legitimate user of the compromised host has access to, which is then spread to other users who access that share. This lateral movement may not be detected by a network-based intrusion detection system, since it represents a typical pattern of access, and the content of internal network flows may not be monitored. In such circumstances, once the initial infection is discovered, we wish to be able to quickly and automatically triage the threat situation in order to prioritize investigative activities on those hosts most directly threatened by the hostile activity.

Note that we focus on layer 4 (transport layer) [2] traffic in this analysis, and ignore layer 3 (network layer) [2] attacks, since routers and switches by their nature forward packets, both benign and malicious, regardless of compromise. It also precludes modeling email borne attacks, except when the initial email provides a foothold which is then leveraged into lateral movement as described above.

Network tainting distinguishes between the logical connectivity of a local network (often simply the complete graph) and observed connectivity by identifying chains of communicating nodes that originate from the known hostile node. Given that any network based attack must generate a network flow, an attacker must sequentially penetrate each host on some taint chain in order to attack the next host on the chain. The threat level for the hosts on a particular chain is thus monotonically decreasing in the length of that chain from the compromised host. We use this information to build a map of the network that corresponds to dynamically changing node-to-node communication patterns. Creating this map requires shortest path calculations; however, the standard algorithms do not apply. Thus, we provide a variant on breadth first search that allows duplicate node and edge visits that consume distinct time slices. The result is a taint map that can be used to prioritize the scope and depth of both human and automated analysis. We then augment this analysis through applying concepts from the complex networks literature [3], and derive several key concepts that allow us to prioritize the post-compromise analysis and remediation for hosts on the same network as the compromised system. To test the network tainting concept, we monitored 24 hours of transport layer communication from a production network of 7335 nodes. The monitoring performed was of communication between internal hosts (not the more typical perimeter monitoring of communication with external entities). We then calculated taint measurements (taint, DSP, and HBTN size, described below) from each internal node over increasing periods of time. We describe how to operationally use each of the metrics to prioritize

[1] By lateral movement we are referring to an attacker consecutively penetrating nodes whereby the most recently penetrated node acts as the platform for attacking and penetrating the next node on the overall attack path.

[2] In this case, commercial detection tools might not be deployed on internal network segments due to licensing and maintenance costs while flow archiving may be enabled in that same network infrastructure (thus enabling network tainting analysis).

relevant incident response activities. We then empirically show how, in our situation, attacker activity could be effectively bounded for incident response times of 15 hours or less.

In contrast to much attack graph work, our method ignores specifics of vulnerabilities and dependencies and focuses on observed network traffic. This approach effectively assumes that all nodes are vulnerable to arbitrary exploitation (with 'hub nodes' having special properties making them less vulnerable, discussed below), and focuses on observed traffic to attempt to bound the total potential ingress of an intruder. Related work in [4] explores the notion of "reachability" between pairs of nodes as a component of the construction of attack graphs, however focuses entirely on logical connections obtained from network structure (e.g., logical subnets are assumed completely connected). The work of [5] and [6] examines the notion of separability by defining efficient methods by which certain critical resources can be provably secured within an attack graph. As in [4], this work focuses on logical connections (as opposed to observed network traffic), and emphasizes the provable security of critical resources under a known set of vulnerabilities, rather than our emphasis on bounding total network penetration under our simplified binary classification of 'vulnerable' and 'not vulnerable'.

While our test network was IPv4, our extracted data was the transport layer communication patterns, which do not change substantially between IPv4 and IPv6. This is important because we consider this work in the context of both protocols. The IPv6 application is most effective. For IPv4, we have to take into account that taint sizes may increase due to attacker activity.

The remainder of this paper is structured as follows. In section II we define and discuss the network tainting relation and related measurements, provide an algorithm to construct a data structure that allows for rapid computation of them, and discuss practical considerations in IPv4 and IPv6 networks. Section III describes the experiment design and section IV describes the results. Section V discusses possible incorrect hub node identification and remediations. Section VI concludes.

II. NETWORK TAINTING RELATION, ALGORITHM, AND OPERATIONAL USAGE

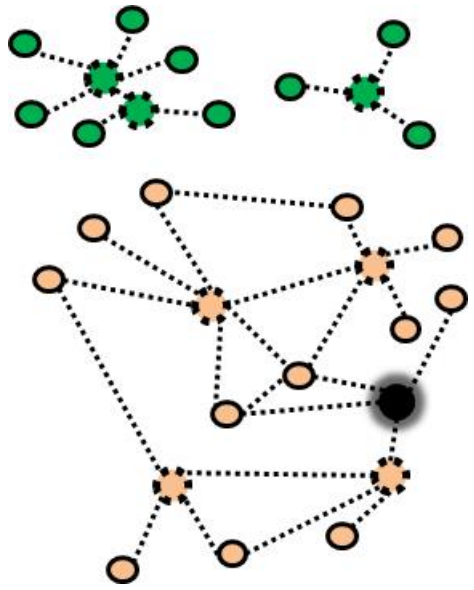
Given a particular network ingress violation, the set of tainted hosts can be identified by evaluating chronologically ordered communication paths originating from the penetrated host using the network tainting relation:

Network Tainting Relation: Host S_1 is said to taint host S_n over a network N at OSI model layer L, through a series of distinct intermediate hosts ($S_2..S_{n-1}$) if and only if: 1) S_1 is known to be hostile, 2) directed communication S_k to S_{k+1} at OSI model layer L exists on N between all sequential pairs of hosts in the chain $S_1..S_n$ for all k such that $1<=k<n$, and 3) for all k such that $1<=k<n-2$, there exists at least one S_k to S_{k+1} packet transmission on the $S_1..S_n$ communication chain prior to at least one S_{k+1} to S_{k+2} packet transmission.

Notice that the tainting relation is restricted to hosts within N and thus we have no need to monitor or model connections to hosts external to N. This is because we are concerned here only with security violations within N from the known point malicious influence. The inability to monitor flows between external nodes makes it impossible to evaluate out taints that propagate through two or more external nodes in succession.

We monitor and evaluate the tainting relation at the transport layer. At this layer, the pairing of nodes within an enterprise network communicating at the transport layer is not random because internal network users often use a consistent set of network services (e.g., email, LDAP, databases, DNS, domain servers, and internal web applications). This consistency limits the growth of the taint sets over time.

Despite this, unrestricted[3] taint sets will often be large due to the scale-free nature of transport layer communication. This is because in scale-free networks, many nodes talk to only a few nodes and a few nodes talk to many nodes, approximating a power law distribution [3]. Since the hub nodes talk to the majority of nodes, the taint can spread quickly within just two hops. Nevertheless, in some cases the tainting relationship will allow for separation of potentially threatened nodes (shaded lighter in orange) and provably untainted nodes that are isolated from the communications graph (shaded darker in green), as illustrated in Figure 1. Note that the figure does not explicitly depict time dependencies that are used to construct the tainting relation.



[3] In some cases we can restrict the propagation of the taint (e.g., when we have knowledge that certain nodes have been evaluated and found clean by the incident response team).

Figure 1. Communication Graph with Tainted (orange) and Safe (green) Nodes

Taint depth: For each node within the same communication graph as the threatened node, we may define the taint depth as the minimum number of nodes through which an attacker would have had to progress to threaten that node. As increased taint depth increases the minimum workload for the attack to gain a foothold in that node, the threat to a node will monotonically decrease along a taint chain as the taint depth increases. Thus, it may be used as a threat level indicator to prioritize investigative efforts. The communication graph then overlaid with the tainted nodes and taint distances can form a dynamically changing and time varying taint depth map of the network with respect to the known compromised node as shown in Figure 2. Note how the nodes are reorganized into rings representing taint depth. This graph representation is convenient and will be readable even for large graphs since no flow edge can skip a taint depth ring. For example, an appropriately time-ordered flow cannot exist between a node in ring 1 and 3, because then that flow would pull the latter node into ring 2. The set of nodes at a taint depth of 1 is of particular interest, as it represents the subset of nodes that are directly accessible in terms of the logical layout of the network that received communication from the compromised node, and hence may also have been directly compromised.

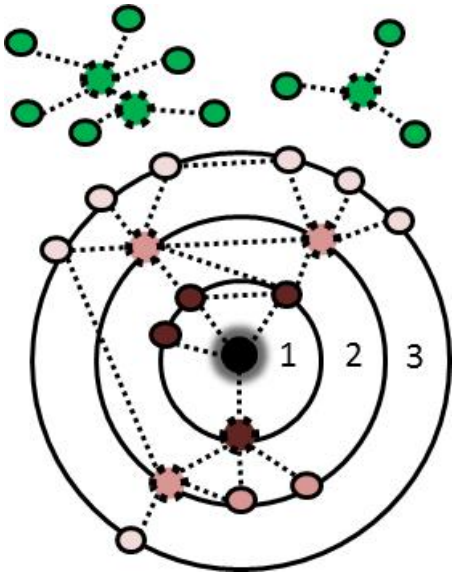


Figure 2. Defense-in-Depth Overlay

In practice, we observe that the separation into isolated sub-graphs as shown in Figure 1 is rarely apparent, and so we extend the simple network tainting relationships with the following concepts:

Hub nodes: The internal transport layer communication structure of many operational networks approximates a scale-free distribution, where many nodes talk to a few nodes and a few nodes talk to many nodes. It is often the case that these high-degree nodes often have four important properties: 1) They are hardened to a similar degree as network perimeter devices. 2) They are specialized devices with more limited attack surfaces than desktop systems with their myriad of attack vectors. 3) They are monitored to a greater degree than other systems. 4) As servers, these tail nodes connect to many other nodes in the network. In the complex networks literature, tail nodes are called 'hub' nodes. We follow this nomenclature, and designate a set of hosts S_i as hub nodes if they satisfy the four properties above. In our diagrams, we indicate hub nodes by nodes with dashed borders (see, e.g. Figure 3).

Dynamic Security Perimeter: The scale-free nature of the network, as well as the existence of hub nodes that require additional effort to compromise, allows us to identify a set of hub nodes that form a dynamically changing security perimeter (DSP) around the known compromised node. This (often small) set of nodes may be subjected to extensive scrutiny to ensure that they have not been penetrated. If one has been found to be penetrated in such a way as to allow attack propagation, then the scale-free nature of the network communication will often result in most of the rest of the network being directly threatened.

Hub bounded threatened nodes: When a DSP can be established, some subset of non-hub nodes will be at relatively small taint depth from the compromised node, and not separated from the known compromised node by the DSP. These hub bounded threatened nodes (HBTN) are at higher risk due to their communication graph proximity to the known compromised node and the fact that they are not protected from the compromised node by the DSP. Thus, they will likely receive priority attention for analysis and, depending upon the local security policy, may undergo quarantines.

The relationship between hub bounded threatened nodes and the dynamic security perimeter is displayed in Figure 3 below.

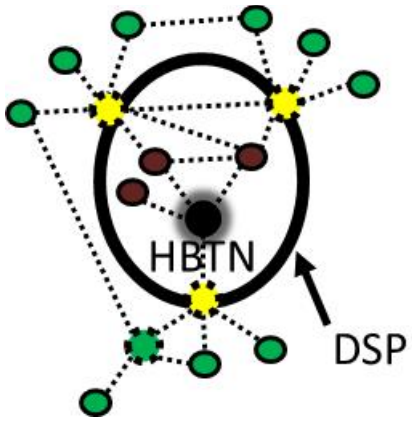


Figure 3. Dynamic Security Perimeter and Hub Bounded Threatened Nodes

The initially compromised node is black, and dashed lines indicate the existence of a taint path from that node to the

other connected nodes. Hub nodes are indicated with dashed edge markers. They are on the DSP (shown by the large black circle) which is the set of hub nodes through which any taint path to the remainder of the graph must pass – which divides the HBTN (dark red nodes inside the circle) from those nodes that are safe conditional on the security of the DSP (green nodes outside the circle).

A. Taint Analysis Algorithm

Assume that a host, *p*, has been penetrated and that an examination of *p* and related security logs reveals that *p* was compromised after time α and was remediated or quarantined at time ω. Also, assume that all transport layer flows internal to the network have been consolidated into a flow log.

A data structure conducive to solving the problem of identifying all tainting relations originating from *p* , as well as identifying tainting depths and isolating the DSP and HBTN sets, can be developed using a graph-based approach. First, pre-process the flow log to remove all flows where either the source or destination is a node not in the network under investigation. Next, remove all flows whose stop time is less than α and remove all flows with source *p* where the start time is greater than ω. This removes all flows that could not have been part of an attack from *p* during the timeframe of investigation. The resulting set of flows may be relevant to the tainting relation. Lastly, we create a directed multi-graph *G* where nodes represent hosts and the flows are the edges connecting the nodes. Label each node in *G* with the host it represents and label each edge with the respective flow start and stop time (chronologically overlapping flows with identical IP addresses may be merged; ports do not need to match).

We evaluate this graph using a variant on breadth first search where edges must be traversed from *p* in chronological order and vertices may be visited more than once. We must allow multiple vertex visits because a short path from *p* to a vertex *t* may consume more time than a long path from *p* to *t*. The longer path with less time may then be able to use additional edges emanating from *t* not available to the short path, because of the chronological ordering requirement. Edges may also be traversed multiple times but each traversal must use a distinct time slice of the flow represented. For example, we must allow multiple traversals of some edge *e* leaving vertex *t* when the taint time for *t* falls between the start and stop time of *e*. In such a case, an attacker can only attack during the latter time slice. The earlier time slice may be used later in the algorithm when a longer path reaches *t* that creates an earlier taint time.

The idea behind the algorithm is to simply traverse all network paths from *p* and to label the visited nodes with the shortest distance and path time from *p*. However, standard shortest path algorithms (e.g., Dijkstra, Bellman-Ford, Floyd Warshall, and Johnson [7]) do not apply because our edge weights do not represent distances but rather ordering requirements for path traversal and because we must allow multiple edge traversals and vertex visits due to the chronological ordering requirement.

The algorithm allows one to restrict taint propagation by providing as input a list *T* of nodes that are to bound the taint propagation. This can be used to input a set of nodes that have already been evaluated and found clean (i.e., evidence indicates that the host has not been penetrated in such a way that it could propagate attacks to other hosts in the network). We also use *T* in calculating the DSP (described later in detail).

Figure 4 provides the algorithm. To review the inputs, *G* is a graph of the transport layer flows, *p* is the initial penetrated node, α is the time of penetration, and *T* is the set of nodes that should bound the taint.

Taint_Analysis (*G*, *p*, α, *T*):

1. Remove all edges from G whose origin is a node in T.
2. Label all nodes in G with taint_time = ∞
3. p.taint_time = α
4. Make queues Q1, Q2
5. taint_set = []
6. Q1.enqueue (p)
7. Depth = 1
8. While Q1 is not empty:
    a. v = Q1.dequeue()
    b. For all edges e in G. outedges(v) where e.end_time ≥ v.taint_time do:
        i. newv = G.adjacentvertex(v,e)
        ii. taint_time= max(v.taint_time,e.start_time)
        iii. if e.start_time ≥ v.taint_time: remove e from G else e.end_time=decrement(v.taint_time)
        iv. if taint_time < newv.taint_time, newv.taint_time=taint_time
        v. if newv is not in Q2: Q2.enqueue(newv)
        vi. if newv is not in taint_set: taintset.append([newv,depth])
    c. if Q1 is empty:
        i. Q1 = Q2
        ii. Q2 = []
        iii. Depth = Depth+1
9. touched_T=T ∩ taint_set
10. taint_set=taint_set-T
11. Return taint_set, touched_T

Figure 4. Taint Search Algorithm Pseudocode

Steps 1-7 initialize the data structures. Flows from nodes in *T* are removed from *G*. Node *p* is assigned the initial compromise time, α. Queue *Q1* is created to keep a set of all vertices to be visited at a particular depth from *p*. Queue *Q2* is created to keep track of the vertices to be visited at

the next depth. Step 8 iteratively processes sets of nodes at increasing depths from *p*. Once *Q1* is empty and all nodes at a particular depth have been processed, step 8c injects into *Q1* the set of nodes at the next depth thereby implementing a variant of breadth first search. Step 8a chooses the next vertex to process. Step 8b processes each edge leaving the chosen vertex and acts on those that satisfy the chronological ordering constraint of the network tainting relation. Step 8.b.i identifies a new node to taint based on the chosen edge. Step 8.b.ii identifies a candidate taint time for the new vertex. Step 8.b.iii removes the processed edge from the graph if the entire edge's time slice will be consumed. Otherwise, the edge's end time is adjusted downward to delete the time slice that we processed and to allow for future processing of the unprocessed time slice. Step 8.b.iv updates the newly identified vertex's taint time if the candidate time is less than the current time. Step 8.b.v puts the new node on the queue for the next level of searching (if it isn't already on the queue). Step 8.b.vi adds the new node to the taint set. Step 9 determines which nodes from *T* were in the taint set. Step 10 removes from the taint_set any nodes from *T*. Step 11 returns the taint set and the set of nodes from *T* that bounded the initial penetrated node. Note that the taint_set returns the node names, earliest taint time, and shortest taint depth. The earliest taint time and shortest taint depth may come from two completely independent taint chains and thus the values are not necessarily related. In other words, the taint chain producing the shortest taint depth may not also have produced the earliest taint time.

We now investigate the computational complexity of the algorithm. The taint times labeled on vertices as the algorithm processes taint chains are monotonically increasing per the algorithm. Also an individual vertex's taint time monotonically decreases as the algorithm progresses (from an initial value of infinity). These two facts make it impossible for cycles to be repeatedly traversed. This means that an assignment of a taint time to a vertex can at most trigger a reassignment of the taint time of all other vertices. In processing such reassignments all edges may be traversed creating an upper bound of $O(n \times e)$ for the algorithm (with n representing the number of vertices and e the number of edges). We have constructed hypothetical network models that exhibit this worst case time complexity using very artificial graphs and edge label assignments. In practice though, the algorithm runs quickly (as shown empirically in section IV.D), and may be used to construct all of the metrics discussed above.

After detecting a network ingress attack to some host *p* and after *p* has been remediated or quarantined, the Taint_Analysis algorithm can be employed with *T*=[] to discover all threatened nodes. Nodes that are not tainted are safe from the set of attacks modeled by this method originating from *p* (even those that traverse multiple intermediaries within the network prior to reaching their final target). This is because from *p*, there will not exist any chronologically ordered internal network communication paths terminating at a safe node (see Figure 1). By retaining the minimum observed value for Depth as nodes are added to the taint set, we also provide the tainting depth metric as discussed above and displayed in Figure 2.

Finally, given some set of hub nodes, the identification of the DSP surrounding the compromised hub node and the HBTN set is likewise straightforward. For this, we set T=[the set of highly connected servers] and re-run the Taint_Analysis algorithm. In this case we use the touched_T output to identify a set of hub nodes that form a DSP around the known compromised host. Any nodes that remain within the taint set are therefore contained between the originally compromised node *p* and the DSP, and thus form the set of hub bounded threatened nodes (HBTN); see Figure 3.

All of these metrics can be generated quickly and automatically, thus enabling prioritization of incident response activities. The most obvious use is to prioritize evaluation of nodes (both human directed and automated) to determine whether or not they have been attacked. Usually nodes with a small taint depth should be evaluated before those with large taint depths. Another use is to apply differing security policy to nodes of varying taint based threat indicator levels (e.g., quarantining nodes with small taint depths). It should be emphasized though that taint depth is an indicator of threat and not a full measurement. If shown to be not compromised, the DSP can be used to tightly bound the attacker to the HBTN nodes. The HBTN nodes, those of the highest threat, can be quarantined and/or evaluated in order of increasing depth from the known compromised node.

If or when incident analysis determines that nodes are clean (or at least were not penetrated in such a way that they could propagate attacks), the Taint_Analysis algorithm can be iteratively re-run with the set *T* augmented with the set of known "clean" nodes. These results then can further prioritize subsequent analysis activities.

B. Applicability to IPv4 and IPv6 networks

While our experimentation, as described below, focuses on IPv4 networks, the structure of IPv6 networks prohibits several forms of attack that make the tainting metrics more robust in that setting.

Due to the large size of standard subnets in IPv6 networks (2^{64} addresses, the size of the entire IPv4 address space), scanning is typically ineffective [8] (we assume that the attacker has no out-of-band knowledge of the network topology), and so attackers are forced to attack between hosts in a manner that follows existing benign communication paths. In IPv4 networks, however, an attacker can actively scan the comparatively small number of addresses with little difficulty. A full subnet scan would quickly add many new flows to the network and thereby affect network taint calculations (see related discussion in section V). However a more cautious attacker within an IPv4 network may choose not to scan in order to maintain an element of stealth, since scan detection technology is

granular enough that scans can often be detected after just 4 to 5 failed connections (see, e.g., the Threshold Random Walk technique [9]).

Consider the HTBN metric. An attacker can quickly increase the metric value by launching random attacks in an IPv4 network as seen in Figure 5. Assume that new targets are selected at random; some attacks may target nodes within the existing HTBN set and so not increase the taint size while others will add a whole new communication sub-graph to the HTBN set. As can be seen, a larger number of hubs mitigates this growth but cannot eliminate it. At an extreme, each attack will add at least one new tainted host regardless of the number of hubs. In the case of many such attacks the taint set will be large, but justifiably so as many hosts have been threatened by the large number of attacks. Anomalously large taint sizes could even be studied in future research as an additional indicator of active attackers launching undetected attacks (how this would compare to existing anomaly based approaches would need to be investigated).

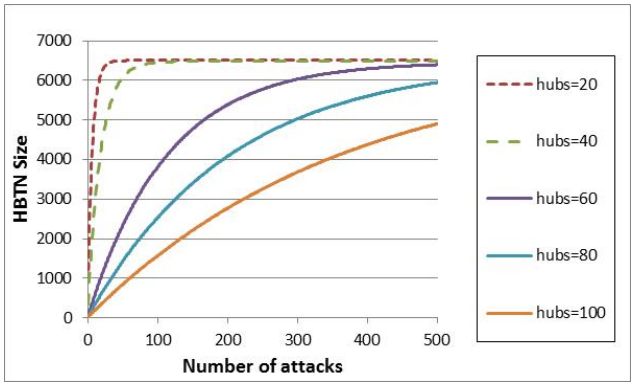


Figure 5. IPv4 Attacks Increasing HBTN Size

In summary, the HBTN size will not be effected by attacker activity in IPv6 networks. In IPv4, however, active attacking (especially random scans) may dramatically increase HBTN size. This is not necessarily bad though as unusually large HBTN sizes will be indicators of an active attacker. Also, since scanning is typically easily detected (and not usually present on internal networks), attackers on IPv4 network may choose to operate with more stealth. Under such situations, the HBTN sizes will be small on IPv4 networks. The derivation of the analytic results presented above is available in the technical appendix.

III. EXPERIMENTAL DESIGN

We monitored all internal network flows within a large production network of 7335 nodes for 24h in the summer of 2013. We began monitoring at noon on a Tuesday of a normal work week. We then subsetted the 24h flow log such that each subset started at the beginning and covered an increasing timespan (using increments of 1h). We used these logs to model network ingresses where incident response teams take an increasing amount of time to quarantine or otherwise remediate a known compromised host (from 1h to 24h). For each subsetted log file, we run the taint analysis algorithm on every node in the network. We thus presume that each node on the network could have been compromised with equal probability (including the hub nodes). This is not strictly realistic as some nodes do not communicate with external hosts, but does allow for the initial penetration to occur through alternate attack vectors (e.g., email, file sharing, or portable media).

For hub node identification to be used in dynamic security perimeter analysis, we took the full 24h log and iteratively extracted nodes with the highest number of communicating partners. Once a node was extracted, its flows to the other nodes were deleted so that the next identified hub would not be promoted based on communication with existing hubs. In this way, we generated an ordered set of hub nodes from which we could select subsets of varying sizes for the purposes of analysis. We fixed this set regardless of whether or not a hub node actually appeared in a particular flow subset (we could have dynamically chosen hubs per each flow subset). This decision negatively influenced our results for the smaller incident response times, but gave us a consistent set of hubs across all experiments. While informal comparisons showed that the hub nodes identified as described above did in fact correspond well to the set of hub nodes as known in the network, more detailed strategies for extracting and confirming hub nodes are beyond the scope of this work.

IV. RESULTS

We now provide empirical results, using taint analysis on our production network for the following security metrics: tainted nodes, dynamic security perimeters (DSPs), and hub bounded threatened nodes (HBTNs).

A. Tainted Nodes

The identification of tainted nodes indicates which nodes could have been attacked from the known compromised node. Conversely, this reveals which nodes are safe from the network based attacks covered by our model. For our 7335 node network, a mean of 2961 nodes (40 %) were safe and 4374 nodes (60 %) were tainted after 1 hour. The taint size rises rapidly as the response time lengthens as shown in Figure 6. At approximately 10 hours, the slope of the increase of the taint size decreases; examination of the data suggests that a combination of network usage patterns (decreasing human activity due to time of day) and saturation of normal connections are the primary causes.

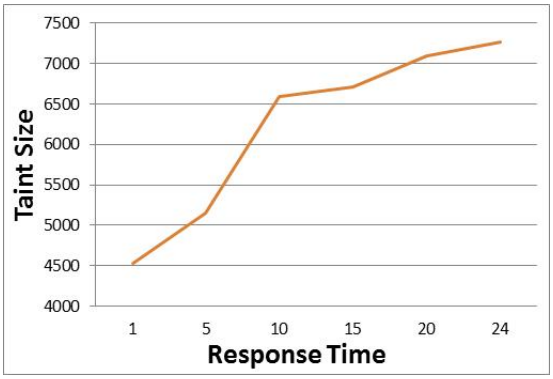


Figure 6. Median Taint Sizes Across Variable Response Times

While these taint numbers are large, they can be used in conjunction with taint depth to prioritize automated analysis of the network. Our experiments show that tainted nodes are distributed up to a hop distance of 11 from a known compromised node. Figure 7 shows the proportion of nodes at differing taint depths. The peak at a depth of 2 to 4 is primarily due to the effect of hub nodes, which are themselves tightly connected and thus offer short paths between most pairs of nodes.

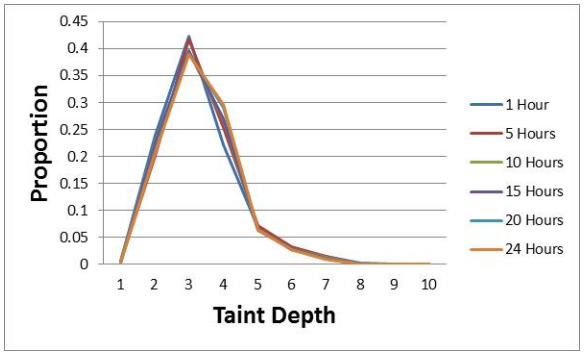


Figure 7. Proportion of Tainted Nodes at Varying Taint Depths

To provide an example, at a response time of 5 hours, the mean number of tainted nodes at depths from 1 to 8 is as follows: 45, 1480, 3072, 1856, 533, 239, 104, and 7.

An important metric is the mean cardinality of the set of nodes at a taint distance of 1. Varying by response time, this hit a maximum of 44.5 at 5h and a minimum of 27.1 at 24h. As described in the operational procedures section, these nodes should receive great scrutiny and will likely be the focus of initial evaluation efforts.

However, if the goal is to bound the attackers influence to quickly ensure that critical servers were not compromised, then the DSP nodes may compete for inspection priority. Somewhat surprising, the set of DSP nodes is on average smaller than the set of taint distance 1 nodes.

B. Dynamic Security Perimeters

DSPs create a barrier that may restrict an attacker's influence to a smaller set of HBTN nodes (to be discussed in the next section). The DSP nodes are a set of hub nodes that encircle the known compromised host in the communication graph. DSPs represent a dynamically changing defense-in-depth layer, not present in most security architectures. Figure 8 shows the DSP sizes given a varying sized set of hub nodes and varying response times.

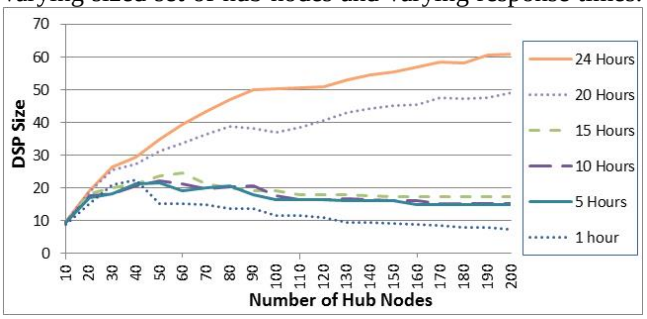


Figure 8. Mean DSP Size vs. Number Hub Nodes

For 70 hub nodes (1 % of the nodes on the network) at 15h, the mean DSP size is just 21.3 nodes. The 15h maximum was a mean of 24.6 nodes with 60 hub nodes. At one hour response time and 70 hubs, the mean DSP size is just 15 nodes. Notice how all of these statistics are less than the mean number of *p*'s immediate neighbors (which varied between 27 and 45 depending upon response time).

In Figure 8, the initial increase in DSP size from 10 hub nodes to 20 correlates closely to the number of hub nodes as each added hub node gets included within the DSP. However, this correlation is lost after around 20 hub nodes and the results become more dependent on the response time. For response times above 15h, the DSP continues to rise; possibly due to time-of-day effect (15-20h corresponds to clock times of 0300-0800, covering the beginning of the working day). This reflects the added flows making it more difficult to bound the known compromised node with hub nodes. For response times of 15h or less the DSP size actually falls after the 40 to 60 hub nodes are added. Overall, the DSP size is small and stays fairly constant for response times of 15h or less. This is important because the DSP nodes will need to undergo analysis to make sure that they have not been penetrated.

C. Hub Bounded Threatened Nodes

Lastly, we look at the HBTN size. These nodes are at the greatest threat from the known compromised node (for attacks models by our method). The threat level of these nodes can be compared through using taint distance as an indicator, but for this analysis we just evaluate the HBTN size. Figure 9 shows the number of HBTN nodes given a varying sized set of hub nodes with curves for select time intervals.

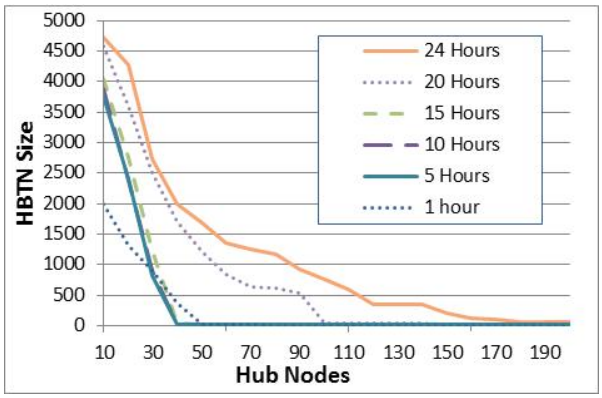


Figure 9. Median HBTN Size

The curves representing 5h, 10h, and 15h are similar indicating that HBTN sizes will be roughly equal regardless of the response time (up to 15h). This matches the incident response time results obtained with DSP size.

Up to and including 15h, as long as the set of hub nodes is at least 70 (1 % of the nodes on the network), then the median HBTN size is at most 22. After 15h though, the HBTN sizes grow significantly. At 20h, the median HBTN size with 70 hub nodes is a large 638. With a 1h response time, it is just 14 nodes.

D. Execution Time Statistics

All experiments were performed on a commodity desktop computer using 3GHz quad-core Intel processors and 8GB of RAM under Python version 2.7.2[4].

The most computationally expensive part of running the Taint_Analysis algorithm is in building the communication graph. It takes a mean of 220s when using the full 24h data feed. Using less data to model faster incident response times produces an approximately linear decrease since the primary operation iteratively adds flows to the graph. Note that graph building can be done in an online always-on fashion so that this step is not be necessary in order to run the Taint_Analysis algorithm.

Actually running the Taint_Analysis algorithm takes much less time. For the worst case of T=[] with 24h of data, it takes a mean of 12.8s. As the size of T increases, the execution time decreases as shown in Figure 10.

[4] Any mention of commercial products or reference to commercial organizations is for information only; it does not imply recommendation or endorsement by the U.S. government nor does it imply that the products mentioned are necessarily the best available for the purpose.

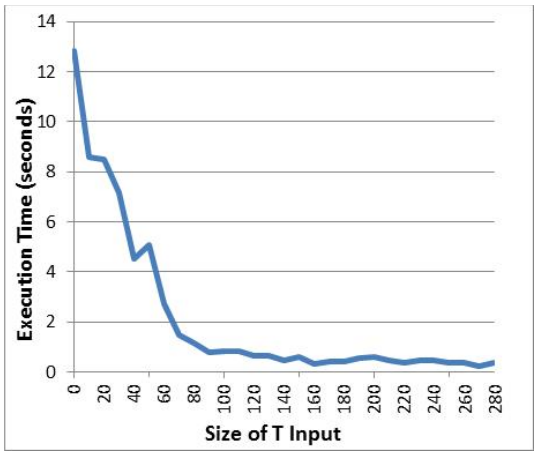


Figure 10. Taint_Analysis Execution Time for 24 Hours of Data

V. INCORRECT HUB NODE IDENTIFICATION

Our experimental approach for hub node identification was to assign as hub nodes high degree nodes in the evaluated flow set. While exhaustive characterization of the hub selection methodology is beyond the scope of this paper, we did find it to produce qualitatively good results compared against known servers on the network. However, when using this method, it is possible that an attacker who compromises a host and immediately begins to move laterally within the network to a wide range of hosts may cause some of the nodes it uses to be labeled as hub nodes. This would create error in the DSP and HBTN calculations. Several possible remediating measures present themselves:

First, directionality of traffic may be incorporated into the calculations, noting which hosts initiated the flows under evaluation. Attackers attempting to move laterally will generally be the initiators of flows, while known hubs offering services to the network will typically be the recipients of connection attempts. A pre-processing step to identify hubs based on flow orientation could be performed, whereupon the compromised nodes initiating large numbers of flows would be correctly identified as spreading a taint to a large portion of the network.

Second, for smaller networks, maintenance of a list of hubs should be tractable. While this would rely upon some degree of record keeping, a listing of hub Internet Protocol (IP) addresses would cause new hub nodes to immediately stand out as above, and either be classified as subverted nodes or as new hub nodes in need of inclusion in the list of hubs.

Finally, tracking a side count of the degree of each node over windows of time might allow rapid identification of nodes that suddenly change behavior. As this information could be constructed in the course of the taint propagation algorithm, and would require storage of an unsigned integer for each node in the network, the additional overhead created by this approach would be negligible. Nodes that are observed to have a significant increase in degree from

one iteration of the tainting algorithm to the next, or to have exhibited steady growth in degree distribution over time, would be flagged for investigation as potentially compromised before being added to the list of hubs in the taint propagation algorithm.

VI. CONCLUSION AND FUTURE WORK

We present and analyze the concept of network tainting and apply it to bounding the influence of an attacker within a network. For this, we provide a network tainting relation and the Taint_Analysis algorithm, presenting empirical results of applying the algorithm to a mid-scale production network. The algorithm exploits the time-ordered nature of network connections to bound the set of internal nodes that an attacker may have been able to access via lateral movement from an initial point of compromise in time.

We show how taint analysis effectively cuts the communication graph into nodes that are tainted to varying degrees and nodes that are safe (against the set of attacks modeled by this method). We show how taint distances vary allowing for prioritized evaluation. We also leverage the scale-free behavior of the communication graph to identify a dynamic security perimeter (DSP) that represents a barrier to attacker propagation within the network. Finally, we measure the set of Hub Bounded Threatened Nodes (HBTNs), those nodes that are closest to the known compromised host and which are not protected by a DSP.

In future work, we will consider additional qualitative inputs from network operations such as node value. The full set of tainted hosts can be intersected with the organization's list of servers with high value data, as defined by security risk analysis and policy, to create a list of affected high-value servers and their taint distances from the compromised node. This essentially evaluates the taint-based threat level indicator for each valuable resource that may be a specific target of a network incursion. Such information could further prioritize analysis tasks by incorporating this value into a planning model for determining the optimal order of investigating and remediating nodes. If certain nodes are known to contain critical information that requires a higher degree of protection while remediation efforts are incomplete, the taint graph may also be used to identify a time-ordered analogue of a max-flow min-cut solution: a set of links that can be cut to isolate the tainted nodes from the critical resource with otherwise minimal disruption to the network.

In doing this work (both present and future), we hope to advance network incident response activities by providing an automated and rapid initial triage service that can guide and prioritize investigative activities.

VII. APPENDIX – DERIVATION OF RESULTS OF SECTION II.B

The results of Section II.B were calculated using inductive reasoning as follows. Let *n* represent the size of the network, *x* represent the number of attacks, *m* represent the mean communication sub-graph size, and *h* represent the number of hub nodes. Let T(x) be the mean number of tainted hosts. T(0)=m-1 since the average sub-graph size is m and the initial compromised host is not counted as tainted. T(x) can then be calculated as shown in equation 1 where S(x) represents the probability that the x^{th} attack connects to a new communication sub-graph.

$$T(x) = T(x-1) + S(x)m \qquad (1)$$

S(x) can be calculated by finding the probability of using the set of nodes in the network that have not yet been attacked. S(x) then is equal to the remaining nodes minus the number of tainted nodes minus the number of non-attacked hubs, all divided by the number of remaining nodes. S(0) is defined as equal to 1 since the initial compromised host will be part of some communication sub-graph. We then derive S(x) as shown in equation 2. In the equation, H(x) is the probability that the x^{th} attack is against a hub node.

$$S(x) = ((n-x) - \sum_{i=0}^{x-1} S(i)(m-1) + \sum_{i=0}^{x-1}(1-S(i)) - (h - \sum_{i=0}^{x-1} H(i))) \ /(n-x) \qquad (2)$$

H(x) can be derived as shown in equation 3. H(0) is equal to 0 as the initial compromised node will not be labeled a hub node.

$$H(x) = h - \sum_{i=0}^{x-1} H(i) \qquad (3)$$

ACKNOWLEDGMENTS

This research was sponsored by the U.S. Army Research Labs and the National Institute of Standards and Technology (NIST). It was partially accomplished under Army Contract Number W911QX-07-F-0023. The views and conclusions contained in this document are those of the authors, and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory, NIST, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes, notwithstanding any copyright notation hereon.

REFERENCES

[1] E. J. Schwartz, T. Avgerinos and D. Brumley, "All You Ever Wanted to Know About Dynamic Taint Analysis and Forward Symbolic Execution (but might have been afraid to ask)," in *2010 IEEE Symposium on Security and Privacy*, Oakland, 2010.

[2] *Open Systems Interconnection Model*, ISO/IEC Standard 7498-1, 1994.

[3] A.-L. Barabasi, "The Architecture of Complexity," *IEEE Control Systems Magazine*, pp. 33-42, 2007.

[4] K. Ingols, R. Lippmann and K. Piwowarski, "Practical attack graph generation for network defense," in *Computer Security Applications Conference (ACSAC)*, 2006.

[5] S. Noel, S. Jajodia, B. O'Berry and M. Jacobs, "Efficient minimum-cost network hardening via exploit dependency graphs," in *Annual Computer Security Applications Conference*, 2003.

[6] M. Albanese, S. Jajodia and a. S. Noel, "Time-efficient and cost-effective network hardening using attack graphs," in *International Conference on Dependable Systems and Networks* , 2012.

[7] T. Cormen, C. Leiserson and R. Rivest, Introduction to Algorithms, MIT Press, McGraw-Hill, 1994.

[8] Chown, *RFC 5157: IPv6 Implications for Network Scanning*, IETF, 2008.

[9] J. Jung and V. Paxson et al., "Fast portscan detection using sequential hypothesis testing," in *IEEE Symposium on Security and Privacy*, Oakland, CA, 2004.