Fighting Coordinated Attackers with Cross-Organizational Information Sharing

Mark Allman[†], *Ethan Blanton*[‡], *Vern Paxson*[†], *Scott Shenker*[†]
[†]*International Computer Science Institute,* [‡]*Purdue University*

Abstract

In this paper we propose an architecture for using cross-organization information sharing to identify members of a group of hosts enslaved for malicious purposes on the Internet. We root our system in so-called "detectives"—savvy network monitors like sophisticated intrusion detection systems or honeyfarms that have a deep understanding of malicious behavior. We augment information from these detectives with observations from a large array of "witnesses" that are already in-place at many locations in the network. These witnesses are not savvy enough to understand that a particular behavior is malicious, but their simple factual observations can be shared with a detective in order to form a broad picture of a group of bad actors. A key aspect of the system is the design of a lightweight mechanism to reliably share enough information between detectives and witnesses to form an understanding of a group of bad actors without sharing more information than necessary, in order to address privacy and competitive concerns.

1 Introduction

One of the largest current threats to hosts and networks is armies of enslaved hosts ("bots") controlled by a single person or small group. These "botnets" provide an attacker the ability to bring much distributed firepower to bear on a particular target and/or to remain elusive by shifting attacks around the network. The exact procedures for an army of hosts to exchange information and attack other hosts comprise nearly an endless list. Therefore, monitoring the activity of such a group of hosts presents an immense challenge along a number of axes. First, observations from any one point in the network provide only a small view into the overall activity. Second, the vast array of attack vectors and benign communications channels that can be co-opted for control traffic make ferreting out botnet activity very difficult.

To better unmask a group of coordinated attackers we propose a system loosely modeled upon real-world crime fighting. While society employs highly trained crime-fighters ("detectives"), there are not enough such skilled people to monitor all situations where a crime may be committed. As a practical matter, real-world detectives rely on amateurs ("witnesses") who have observations and evidence that aid the detectives in their work. While witnesses are clearly not as skilled and trustworthy as detectives in terms of fighting crime, their value is in their *numbers and prevalence*.

Detectives are charged with detecting patterns of criminal activity, identifying suspects, and then questioning witnesses to fill in the gaps in the detectives' understanding. In particular, we expect detectives to gather information relating to a particular crime—not arbitrary information about arbitrary people or events. Of course, some unrelated information may always "leak" into the process, but anything not germane should be disregarded. Similarly, witnesses should be questioned in such a way that they do not know precisely what they are being asked about, so that they do not learn what criminal activity the detectives are pursuing nor whom the detectives suspect; they only attest to what they have directly observed.

In the realm of fighting groups of coordinated attackers, our detectives are savvy network monitors such as sophisticated intrusion detection systems (IDSs) or honeyfarms [8]. These components of our system can detect "crimes" and discern suspicious patterns of activity. However, as in real life, their viewpoint is too narrow to understand the breadth of activity in disparate corners of the network. Therefore, we also employ general traffic monitors (packet taps, NetFlow logs, proxy cache logs, etc.) as "amateur witnesses" that have evidence to offer, but are themselves not savvy enough to understand that a "crime" has been committed or to put together the complete picture.

In this paper we propose leveraging the *deep understanding* of network detectives and the *broad understanding* of a large number of network witnesses to form a richer understanding of large-scale coordinated attackers. To accomplish this task, we need a way to share information across organizations. Therefore, we offer an information sharing mechanism that (*i*) reveals little-to-no information to anyone who has not witnessed a given event, while still allowing witnesses to provide corroborating evidence and (*ii*) offers the detective reasonable validation that the information from witnesses is sound.

We separate the activity of coordinated attackers into two categories, *attack traffic* and *control traffic*. Attack traffic can range from distributed denial-of-service attacks to scanning for additional vulnerable hosts to recruit into the group. Much research and many products concentrate on finding individual hosts that are actively attacking peers in the network. Control traffic's purpose is twofold: (*i*) for commands to flow from some controller to all members of the group, or (*ii*) for the members of the group to download new malcode or otherwise further prepare for some task (such as an attack). This traffic is more difficult to track than attack traffic precisely because it can appear normal and benign (e.g., simply downloading some data from a URL using HTTP). This normality makes it much harder to identify the traffic as laying the groundwork for an attack. In this paper we focus on using this control traffic to unmask the members in a group of coordinated attackers, even in the absence of an attack.

A high-level example would be a honeyfarm becoming "infected" by a given attack vector and then observing a remote server from which the bots are instructed to retrieve some piece of malcode. The honeyfarm (the detective, in this case) would query witnesses throughout the network for additional hosts that show similar communication patterns. Our information-sharing technique allows the honeyfarm to uncover other hosts that are likely members of the group of coordinated attackers based on witness "testimony", even though these group members and witnesses are scattered throughout the network (such that the honeyfarm cannot directly observe the behavior). Furthermore, unless a witness has observed the activity in question, the honeyfarm's queries about the pattern are obscured such that the honeyfarm reveals little information to the potential witness.

This paper is organized as follows. In § 2 we briefly describe our proposed architecture. Next, § 3 outlines the underlying information-sharing mechanism that enables the system. § 4 outlines related work. We provide brief conclusions and areas for future attention in § 5.

2 Architecture

The overall architecture of our proposed system consists of three classes of participants: (*i*) a set D of network detectives (e.g., honeyfarms, sophisticated IDSs, etc.), (*ii*) a set W of witnesses, (*iii*) an aggregation entity that can play the part of a trusted organization like Interpol and gather information from a number of detectives' jurisdictions and then distribute the information to information consumers. The general operation is that some D_i finds a pattern and then interrogates witnesses in search of additional hosts that exhibited the given pattern. From the witness testimony, D_i then forms a list of victims V_i. D_i then sends V_i to the Interpol-like aggregator along with the appropriate pattern. The collector can then gather various V_i sets from various detectives together to form a picture of the group of coordinated attackers. We consider each component of the architecture in turn in the following subsections.

2.1 Detectives

The set of detectives is charged with identifying traffic patterns that correspond to malicious behavior and then querying witnesses to uncover additional hosts that have exhibited the same pattern. The detectives aggregate witness responses and reports the results to the collector.

In § 3.4 we discuss "rogue detectives" who attempt to abuse the system by fabricating patterns in order to "fish" for private information not related to malicious activity. To reduce the risk of such fishing attacks in our system, we keep set D *closed*, i.e., membership is known *a priori* and each host in the set can be readily identified (e.g., using a cryptographic key). Since in our architecture the detectives need to be known and trusted, the set is intended to be kept small (e.g., hundreds of monitors). However, we note that the wealth of information in our system comes from witnesses, not detectives, and therefore a small set of the latter should not present a problem.

An immediate question that a detective must tackle after identifying a suspicious pattern involves determining which witnesses to interrogate. Depending on the situation, the appropriate scope of the queries might range from asking one particular witness a quite-localized question to asking the entire set of witnesses a broad one. For instance, if some group of coordinated attackers employs a centralized code-distribution server then ideally a detective could query a single witness close to the server and reap a wealth of information about which hosts have been seen downloading the code. This might be slightly broadened to a small group of witnesses to account for any of multihomed sites, possible artifacts in the witnesses' logging functions due to their use of sampling, and/or witness misbehavior. The downside of targeted querying is that the detective must assess the role of the witness's proximity to the point of interest. On the other end of the spectrum, if the pattern is not host-specific but along the lines of "incoming connection to port X, outgoing connections to ports Y and Z" then querying as many witnesses as possible around the network will give a more complete picture than trying to query any one particular witness. This is easier to accomplish than targeting witnesses because no notion of proximity is required. In between, there are many possibilities for the querying of various fixed or random sets of witnesses. An in-depth exploration of which witnesses to query is beyond the scope of this paper, but a clear candidate for continued investigation.

2.2 Witnesses

We expect W, the set of witnesses, to consist of a large number (thousands) of simple, general traffic monitoring devices—not particularly designed for security monitoring—scattered throughout the Internet. Unlike the closed set of detectives, W is *open*: new monitors can readily join and witnesses do not need to be vetted before they start answering queries. Since many ISPs and organizations do some sort of general traffic monitoring as a matter of course (for provisioning, debugging, etc.) we aim to leverage these resources rather than rely on additional deployment. That said, these monitors will need to be augmented to answer queries from the detectives in our system. Witnesses are expected to simply log "the facts"—that is, direct observations from the network without any analysis. We do not expect witnesses to "judge" traffic. Rather, the function of witnesses is to provide the detectives with observations to allow a picture of large-scale groups of coordinated attackers to be formed by the detectives. We also note that witnesses in our system can only provide information in response to queries from the detectives and therefore cannot contribute arbitrary data to the system. Witnesses are, of course, also free to ignore requests based on local policy. An incentive for witnesses to contribute information is that the aggregated information will then be made available via the collection and distribution system such that the organization providing the witness will ultimately gain an amplified view of coordinated attackers.

2.3 Collection and Distribution

The Interpol-like collector is a known and trusted entity that aggregates the information collected by the hosts in D and makes the information publicly available. Since the members of D are well-known, it is tractable to only accept input from trustworthy parties.

The network Interpol serves several key functions. First, it can aggregate information from many detectives to form a more comprehensive picture of groups of coordinated attackers. Second, the collector is responsible for making the results public, but must do so in a way such that the source of each individual piece of information is masked.[1] In addition to collecting and aggregating the information, the collector makes the data publicly available (perhaps via some intermediary distribution points). Doing so allows services to be built that offer the information in myriad ways that operators may find useful. Example services include: simple mirrors of the data via FTP or HTTP, a database server that accepts rich queries, behavioral database entries (ala [1]), or insertion of the data into a robust distributed data structure such as a DHT for reliable dissemination.

Finally, we stress that the collector's role is to aggregate and serve the information, not design the policy. The collector can provide information that will inform policy decisions, but those decisions are left in local hands.

[1] Additional ways to thwart tracking may also be useful to employ, such as Mobile Honeypots [3].

3 Information Sharing Primitive

While the last section sketches our overall architecture, this section focuses on a "loose private matching" scheme to facilitate information exchange between detectives and witness that conforms to the principles outlined in § 1.

3.1 Loose Private Matching

The key idea behind the "loose private matching" mechanism is to enable detectives to encode a query (traffic pattern to look for) in such a way that *(i)* anyone who has actually observed the traffic described by the pattern will be able to recognize it, but *(ii)* the encoding is also ambiguous enough that it could describe a variety of traffic patterns, and therefore it reveals little information to entities that have *not* observed the given traffic pattern. We enforce this distinction by requiring that witnesses who wish to attest to having seen traffic fitting a given pattern must encrypt their responses using the decoded pattern itself as a shared secret. The detective therefore gains a reasonable (not perfect—see below) confidence that the witness indeed observed the traffic in question.

To develop this approach, we consider that patterns being queried are defined by some set of observed actions that a detective can piece together. To illustrate, suppose a honeyfarm H is attacked by some host A that is scanning for vulnerable hosts to recruit into a group of bad actors via an SQL exploit. Furthermore, after H is "infected" it is then asked to TFTP some malcode from code server C. A natural pattern a detective might develop from this interaction is "incoming SQL hit from A arrives at some host X, which in turn initiates an outgoing TFTP request to host C". Any X (such as H) that satisfies this pattern could be assumed to be infected in the same manner as H. The pattern could be loosened up such that any communication from host A (a known bad actor) could be used instead of just SQL connections to handle attackers that use multiple attack vectors could be found. Or, any TFTP to host C could be taken as an indication that the host initiating the connection has been infected. Clearly, these are not iron-clad signatures for an attack, and care must be taken to narrow the scope of queries. For instance, if the malcode happens to have been left on a popular blogging site B and infected machines fetch it via HTTP then using the pattern of "HTTP transactions to B" is not going to be a useful pattern in finding infected machines.

After forming a pattern, the components of the pattern, $C_1 \ldots C_n$, are then hashed together to form a key, $K = H(C_1, \ldots, C_n)$, which is then used to query witnesses for hosts with similar traffic patterns. The witnesses that receive the query then consult their logs for hosts having communications that match the requested key. Only if the witness has seen the given pattern will it be able to untangle the given key and provide a useful response. The response is encrypted using the decoded components of K as the shared secret.

Consider an example where a key is constructed by a honeyfarm with a destination IP address d_h, a transport protocol t_h and a destination port number p_h as $K_h = H(d_h, t_h, p_h)$. Now, consider K_h being sent to some number of witnesses with the intent of obtaining a list of source IP addresses that have communicated with hosts in the fashion described in the pattern K_h.

Assume that some witness finds three records that match K_h—with two of these records matching the query sent by the honeyfarm, and the third being a coincidental hash collision. The witness cannot determine which of these matching records (if any), are correct, so all matches are returned. Assume that the source IP address of each matching record s_i is associated with a three-tuple $T_i = \{d_i, t_i, p_i\}$, which represents the material hashed to produce the key matching s_i. In our example, $T_1 = T_2 = \{d_1, t_1, p_1\} = \{d_2, t_2, p_2\}$, and $T_3 = \{d_3, t_3, p_3\}$, a different tuple than T_1 and T_2. The witness forms two responses to be returned to the honeyfarm. The responses consist of a list of addresses d_i from the query followed by each appropriate s_i. These records are encrypted *using T_i as the shared secret*. Specifically, the witness forms the two responses $R_1 = E_{T_1}(\{d_1, s_1, s_2\})$ and $R_2 = E_{T_3}(\{d_3, s_3\})$.

The honeyfarm can now decrypt both responses using $T_h = \{d_h, t_h, p_h\}$ as the shared secret. When decrypting R_1, the honeyfarm will find $d_h = d_1$ as the first address in the list, and so will know that the rest of the addresses in this response are valid for the given query. When similarly decrypting R_2, the honeyfarm will not find d_h as the first item in the returned list, and therefore will know that this response is meaningless and was caused by a collision at the witness. Note that the honeyfarm still does not know either the values d_3 or s_3, as the decryption of R_2 using the inappropriate key $T_h \neq T_3$ yields random data.

We also note that patterns can consist of more than one key that can then be logically connected to form a more specific query. E.g., K_1 may be "source IP A, destination port S" and K_2 may be "destination IP C, destination port T". The query could then be for any host X that the witness observes that satisfies both K_1 and K_2.

3.2 Example

As a concrete example of the above notions, consider a query which requests the source IP address for all hosts having communications that match a pattern that encompasses the destination IP address (4 bytes), transport protocol number (1 byte) and destination port number (2 bytes). From these 7 bytes a key K is formed by taking the product of the bytes (with any zeros rounded up to one). This hash has two crucial properties: (*i*) the hash space is large ($1 \ldots 255^7$, excluding numbers with a prime factor larger than 255) and (*ii*) collisions are guaranteed to be possible in theory. Assuming the protocol number and port number remain the same, IP addresses $a.b.c.d$, $a.c.b.d$ and $a.b/2.c \cdot 2.d$ will all yield the same value (assuming that b and c are even). This ambiguity is critical because it largely prevents anyone who has not seen the corresponding traffic from understanding the question and forming a valid response.

To assess this simple hash function we analyze one day's worth of connection logs from ICSI's border. We used the log from July 27 2006, which consists of roughly 6.2 million connections. We compute a hash using the product of the bytes in the three fields described above for each connection. We find that 11% of the connections hash to a unique K that is not shared by another three-tuple in the dataset. Therefore, 89% of the connections hash to a K with a collision. This indicates that collisions are not just theoretically possible, but ambiguities do in fact naturally occur when using byte-wise multiplication as a simple hash.

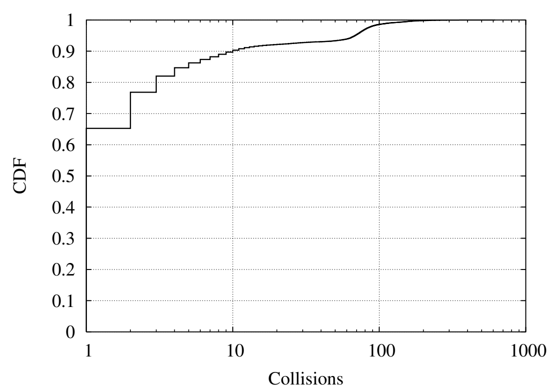

Figure 1: Collisions per key.

Figure 1 shows the distribution of collisions per key in our dataset. The figure shows that roughly two-thirds of the keys are used to cover the 11% of connection that hash to a non-shared K. Further, 90% of the keys correspond to 10 or fewer three-tuples and nearly all keys correspond to 100 or fewer three-tuples. This shows that while there is ambiguity in this particular hash function, the ambiguity likely does not present a logistical problem in transmitting massive amount of data that then needs decrypted by a detective using this hash function. The amount of ambiguity can also be increased with the application of the modulus operator to K such that the size of the hash space is decreased. Alternatively, the ambiguity can potentially decreased by using a smaller window of time such that less traffic is observed.

We stress that we are not proposing this hash as *ideal*. We offer this hash as a simple proof-of-concept that the general idea has promise. For instance, a scheme such as

Private Stream Searching [2] provides many of the desirable qualities we sketch above (and more) at additional computational cost. Crucial future work will clearly involve a survey of alternate hash algorithms and a deeper analysis of the properties of such algorithms.

3.3 Query Language

As described above, the detectives and witnesses have a shared understanding of the components of the queries (IP addresses, port numbers, etc.). While we do not have space to dig into the details of the query language in this paper, we note several possible approaches. First, a standard set of common queries could be defined and a query identifier could be used to synchronize the detectives and witnesses. These fixed hashes could be calculated as the records are initially captured and stored with the records such that a simple lookup on a given key would be straightforward. The downside of such an approach is that the system is locked into a stock set of queries. Another approach is to make the queries self-describing. For instance, the queries may come with a bit mask indicating which components from the traffic are included in the hash. This is more flexible than a system with a standard set of queries, at the price of computing a hash for every stored record every time a query arrives. A third approach is a hybrid—with a set of common questions and a self-describing mechanism for richer queries.

3.4 Cheating

3.4.1 Detectives

The fundamental way that detectives can cheat is to fabricate a query to fish for private information. For instance, a query could easily be constructed that asks for hosts that accessed some unsavory web server. This is essentially inherent in the mechanism. Even if the queries include additional hard-to-fabricate evidence that network traffic has been observed (e.g., a TCP initial sequence number as "proof of standing") and can be verified by a witness,[2] the bar for cheating is only slightly raised. The detective then only needs to observe or execute some access to the resource in question to then gain the appropriate credentials to fish for a broader set of private information. One way to possibly mitigate the impact of fishing attacks is for witnesses to not answer queries about some pattern until a number of independent detectives have requested information about the same pattern. This offers some assurance that a rogue detective is not simply trying to coax witnesses to send private information that has no relevance to attacks. The nature of our architecture aids the mitigation of this fishing attack, as well, because we intend the system to consist of a fairly small number of detectives and to gain the bulk of the information about the members of groups of coordinated attackers from witnesses. Therefore, as sketched in § 2 the set of detectives is known and assumed trustworthy in our architecture (with the caveat that witnesses can clearly further constrain detectives using the thresholding approach sketched above).

[2]This would limit the witnesses that can be queried to those along the path of the specific observed traffic that the detective describes.

3.4.2 Witnesses

Witnesses can either withhold information or fabricate information in response to queries. Our envisioned system includes a large number of witnesses with many vantage points that are likely overlapping. Therefore, the fact that some witness W_i withholds some record does not mean that another witness W_j will not furnish that record to the querying detective. A witness can also try to inject bogus records into a response in two ways. First, bogus records could be piggybacked on a legitimate response. In other words, the witness was able to untangle the query by looking through the local logs, but then instead of simply reporting legitimate log contents, either a completely bogus list or a partially bogus list is returned in an attempt to implicate innocent actors. This can be effectively mitigated within our proposed system by collecting multiple independent witness statements about some actor before making a decision. Another variant of the injection attack is to attempt to crack the hash and respond to a query despite having not seen corresponding traffic. In this case, the chances of the "witness" guessing the correct components the detective used to form a hash are dependent on the hash function, and with an appropriate hash function this should be quite difficult. Further, enough ambiguity in the hash should be in place such that a brute force responding with all possible combinations of the initial components should be readily apparent.

A final form of attack, difficult to defend against, is when an adversary is able to correlate across multiple queries (either made to multiple witnesses, or a succession of queries made to the same witness) to infer what information a detective seeks. Even without multiple queries, an adversary can make some inferences in this regard by inspecting the witness records that match a given query and assessing which matches likely reflect more interesting behavior than others.

4 Related Work

Sharing information across networks and organizations to aid security is not a novel concept. [7] outlines a system that allows sharing of information across local or remote instances of the Bro IDS. The system relies on prearranged certificates to authenticate peers and can scope the information being shared with each peer. The scheme presented in this paper does not require pre-shared certificates, nor do all components of the system need to be well-known, but the information shared is also not as rich as two IDSs could agree to share.

A general "private matching" approach is given in [4] whereby two encryption functions exist such that $E_1(E_2(x)) = E_2(E_1(x))$. Two peers can exchange their encrypted version of some x and determine if x is the same without revealing x. We use a loose form of private matching that does not rely on a shared understanding of encryption functions or keys, which hinders scalability.

The SPIE system [6] allows victims of network attacks to trace the attack back to its origin without relying on the (possibly spoofed) IP address by having routers keep a history of all the packets forwarded (in a Bloom filter). This history can then be queried by producing one of the attack packets—with routers indicating whether or not they have forwarded the given packet. This is an instance of the system we propose in this paper. However, we expand the notion to include less specific questions and non-binary answers.

[1] proposes a system that allows for the reporting of malicious hosts into an open database for anyone to use in forming policy decisions (e.g., "host X is a scanner"). The validity of the information in the database is left to the consumer's assessment of the information provider's reputation. The system assumes a large number of intelligent monitors to collect wide-scale information, whereas we focus on leveraging information from generic network monitors that cannot make behavioral judgments on their own.

Finally, [5] suggests that, rather than looking at the low-level details of communications (e.g., payloads of IRC packets), large groups of coordinated attackers can be identified by deriving communication patterns in the form of a who-talks-to-who "contact graph" from an aggregate view of the traffic (or, even a subset of the aggregate view). Our proposed scheme is similar, but aims to form notions of botnets by using observations of malicious activity to chase down similar activity elsewhere. Further, we offer a scheme that allows pertinent information exchange and does not rely on arbitrary traffic data.

5 Conclusions and Future Work

In this paper, we make several contributions. First, we offer a system that leverages existing wide-scale generic traffic monitoring ("witnesses") to aid a much smaller group of intelligent systems ("detectives") in forming both a *deep* and *broad* understanding of groups of coordinated attackers that can then be used network-wide. Second, we offer a framework for identifying coordinated attackers that does not rely on the specifics of the way any particular botnet operates and therefore may provide a longer shelf-life than schemes that rely on intimate knowledge of botnet behavior. Finally, we sketch a loose private matching mechanism to allow for information sharing about mutually observed network events. The mechanism has promise within the system we have outlined, in addition to possible other tasks where scoped sharing of data across organizations is useful.

While we believe this paper offers a number of novel contributions, much additional work on nearly all aspects of the proposed system is required. For instance, further investigation into hash functions for use within the loose private matching scheme is needed. In addition, an investigation into deriving activity patterns is required such that honeyfarms can compute these patterns on the fly. Such an investigation will also provide information about the critical components of the patterns, which will aid in the design of a query language that detectives and witnesses can share. Finally, a large number of logistical questions remain, such as, will operators find sharing in the fashion presented in this paper is reasonably safe given the potential benefits?

Acknowledgments

Discussions with a large group of people have helped our thinking along, including Marina Blanton, Jason Franklin, David Lapsley, Carl Livadas, Doug Maughn, Robin Sommer, Tim Strayer and Robert Walsh. This work was supported in part by the National Science Foundation under grants ITR/ANI-0205519, NSF-0433702 and STI-0334088, for which we are grateful.

References

[1] M. Allman, E. Blanton, and V. Paxson. An Architecture for Developing Behavioral History. In *Proceedings of USENIX Workshop on Steps to Reducing Unwanted Traffic on the Internet*, July 2005.

[2] J. Bethencourt, D. Song, and B. Waters. New Constructions and Practical Applications for Private Stream Searching (Extended Abstract). In *IEEE Symposium on Security and Privacy*, May 2006.

[3] B. Krishnamurthy. Mohonk: Mobile honeypots to Trace Unwanted Traffic Early. In *ACM SIGCOMM Network Troubleshooting Workshop*, Sept. 2004.

[4] Y. Li, J. Tygar, and J. Hellerstein. Private Matching. *Computer Security in the 21st Century*, pages 25–50, 2005.

[5] V. Sekar, Y. Xie, D. Maltz, M. Reiter, and H. Zhang. Toward a Framework for Internet Forensic Analysis. In *Proceedings of ACM SIGCOMM HotNets*, 2004.

[6] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer. Single-IP Packet Traceback. *IEEE/ACM Transactions on Networking*, 10(6):721–734, Dec. 2002.

[7] R. Sommer and V. Paxson. Exploiting Independent State For Network Intrusion Detection. In *Proceedings of ACSAC*, 2005.

[8] M. Vrable, J. Ma, J. Chen, D. Moore, E. VandeKieft, A. Snoeren, G. Voelker, and S. Savage. Scalability, Fidelity and Containment in the Potemkin Virtual Honeyfarm. In *ACM Symposium on Operating System Principles*, Oct. 2005.