Framework for Evaluating Collaborative Intrusion Detection Systems

Dennis Grunewald, Joel Chinnow, Rainer Bye, Ahmet Camtepe,
Sahin Albayrak
DAI-Labor — TU Berlin, Ernst-Reuter-Platz 7
firstname.surname@dai-labor.de

Abstract: Securing IT infrastructures of our modern lives is a challenging task because of their increasing complexity, scale and agile nature. Monolithic approaches such as using stand-alone firewalls and IDS devices for protecting the perimeter cannot cope with complex malwares and multistep attacks. Collaborative security emerges as a promising approach. But, research results in collaborative security are not mature, yet, and they require continuous evaluation and testing.

In this work, we present *CIDE*, a *Collaborative Intrusion Detection Extension* for the network security simulation platform (*NeSSi*^{2}). Built-in functionalities include dynamic group formation based on node preferences, group-internal communication, group management and an approach for handling the infection process for malware-based attacks. The CIDE simulation environment provides functionalities for easy implementation of collaborating nodes in large-scale setups. We evaluate the group communication mechanism on the one hand and provide a case study and evaluate our collaborative security evaluation platform in a signature exchange scenario on the other.

1 Introduction

IT infrastructures permeate our daily lives, and our society becomes more dependent on information technologies [Cas05]. Cost reduction, improved business opportunities or quality of services; everyday more systems get connected to the Internet. As the scale of such interconnected IT infrastructures grows, due to short innovation cycles, information and communication technologies become more complex and agile. Therefore, securing IT infrastructures becomes a growing challenge. Monolithic approaches such as using stand-alone virus scanners, firewalls and intrusion detection system (IDS) devices for protecting the perimeter cannot cope with complex malwares and multistep attacks. For example, a single and non-collaborative IDS node suffers from a very limited view and detection capability [ZLK10]. Collaborative security emerges as a promising approach. But, research results in collaborative security are not mature, yet, and they require continuous evaluation and testing.

Collaborative IDS (CIDS) systems generally consist of nodes (also agents or peers) that monitor a portion of a communication network and exchange intrusion-related information amongst each other. First, centralized approaches came up, where a single node

erschienen im Tagungsband der INFORMATIK 2011
Lecture Notes in Informatics, Band P192
ISBN 978-3-88579-286-4

analyses the data that it receives from several monitoring probes which are distributed across the network. Of course the node itself in terms of computing power and the surrounding network in terms of bandwidth consumption quickly become bottlenecks in this approach. Hierarchical systems introduced additional analyser nodes that are organized in a tree structure. The monitoring probes report traffic data to the analyzers in the leaves of this tree which in turn analyze, correlate and compress the data. In suspicious cases, this information is reported to the parent node. Unfortunately, the top-level analyzer is still both, a single point of failure and a potential bottleneck, e.g. when attacks target different areas of the network at the same time, forcing correlation at the top-level node and hence traffic reporting to it. To cope with these drawbacks, several approaches such as using peer-to-peer architectures have been proposed. However, research in this field is far from being mature and is still ongoing [GJMB05], [GTDVMFSC07].

Focus of this paper is the continuous evaluation of new CIDS approaches during their design phase. This is a challenging task due to different aspects. First of all, testing a CIDS solution in a real network is not feasible due to the high risk of interfering with legitimate users (e.g. infecting hosts or link overloading). Secondly, establishing a large-scale testbed is a very costly solution for an attack testing. Finally, CIDS approaches require test and evaluation in their design phase, before the solution is fully implemented. Therefore, we extend our network security simulation platform (NeSSi^2) by adding a large-scale mode with built-in and extensible collaborative security functionalities.

Our solution supports the developer during the design and development of new CIDS approaches. It provides functionalities for easy implementation of collaborating network nodes, such as dynamic group formation based on node preferences, group-internal communication, group management and an approach for handling the infection process for malware-based attacks.

The rest of this work is structured as follows. After our prior work in Section 2, our approach for *CIDE* (*Collaborative Intrusion Detection Extension*) is presented in Section 3. We discuss the domain model and its realization before the description of interfaces which allow a user to modify, configure, and finally use the framework. Our case study and evaluation results are presented in Section 4. In Section 5, related work is presented. Finally, we conclude in Section 6.

2 Prior Work

NeSSi^2 [SBC$^+$10][1] is a mature open-source discrete-event network simulator with focus on the evaluation and analysis of security aspects. It has been used in several industry projects, e.g. for evaluating network-centric security mechanisms, or for decision support for performance management in DSL access networks.

NeSSi^2 features a very modular architecture and, since it is implemented in Java, platform-independency. The simulation backend and user interface are decoupled thus allowing

[1] http://www.nessi2.de

ISBN 978-3-88579-286-4

the simulations to be executed on a very powerful machine whilst the setup, monitoring and evaluation can still be carried out on a workstation. Furthermore, NeSSi^2 allows for easy enhancements concerning the applications that are executed on the simulated devices. The handling of the simulation itself as well as the underlying data model can also be modified. The latter defines e.g. the data structures used for network nodes and links. Finally, NeSSi^2 organizes the simulation setups in projects that contain different files for the network topologies, the setups of NeSSi^2 applications and the recording settings. This allows for easy reuse of e.g. topologies for different application setups.

NeSSi^2 *applications* define the behavior of the devices. In order to place them on the nodes, they are bundled to *profiles*. These in turn can be deployed on the network nodes to setup all the bundled applications at once. The topology and the information about the profile deployment form a *scenario*. Finally, the *session* contains scenarios together with simulation-related meta-information like e.g. the number of runs, the length of a run in terms of simulation *ticks* or the *simulation handler* within a *session*.

For the evaluation of huge networks, we extended the simulator with an additional *large-scale mode* (LSM) and the corresponding *simulation handler*. In this mode not every end device is modeled as a node of the network graph [SBC$^+$10].

Consider a typical provider network like shown in Figure 1. The provider maintains an *autonomous system* which is comprised of many *access networks* that the customers' end devices are connected to, usually via a hierarchy of aggregation switches. The autonomous systems are connected with each other, thus building the Internet. The LSM models such an access network as one node (the *netblock router*, NBR) in the network graph and abstracts from all the hundreds of end devices that are connected to it. Furthermore, the process of packet delivery has been simplified in LSM. Hence, unlike in the standard IP mode, the IP layer stack is not part of this model which also decreases the packet sizes, for packets are not wrapped in multiple headers anymore. Moreover, all the end devices that are handled by the same NBR share the same in- and outqueue. The routing of the packets is based on routing tables calculated once at the beginning of the simulation. Each node has its own routing table which contains information about every other host in the same subnet as well as network-address-based information about the other subnets. Thus, the size of the routing tables is reduced like in the border graph protocol (BGP).

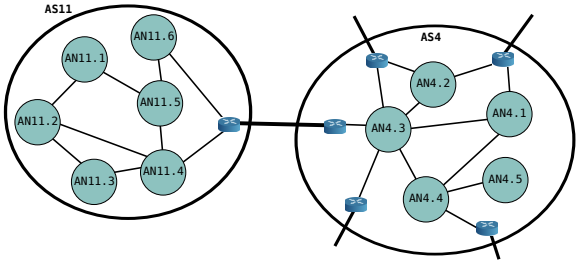

Figure 1: Example topology with two autonomous systems (AS) and their access networks (AN)

In [KKK05], Katti et al. analyzed data from 1700 IDSes/firewalls in the USA and Europe and concluded that groups of 4–6 IDS nodes performed almost as good as when all nodes collaborated with each other – with much less communication overhead. Another indication that a large amount of overhead can be avoided by selecting the collaboration partners

carefully instead of collaborating with every possible node is found in [YBJ04], where nodes are correlated according to their distance in the IP address space.

Inspired by the abovementioned works, Bye et al. proposed an approach for managing CIDS in [BCA09]. The IDS nodes collaborate in groups where the group and its members have to fulfill constraints specified per node. E.g. a maximum group size, a certain operating system etc. would be such constraints. In order to manage the group composition, each node has to specify an *objective* for the collaboration (e.g. "signature exchange"), its own *properties* (e.g. the operating system, installed software, network configuration, etc.) as well as *interests*. The latter specify the properties which the other group members should/must have. The authors also discuss to organize the properties in a hierarchy, e.g. "Win XP" and "Windows 7" belonging to the same higher-level property "Windows".

3 Approach

In this Section, we first discuss how the infection process, the countermeasures and the collaboration groups are modeled. After that, we focus on some specific implementation details of CIDE that concern performance and memory usage as well as details of the group management. Then we discuss interfaces which the user needs to know about in order to successfully create his own simulations.

3.1 Domain Model

Before discussing the appliance of countermeasures, we first need to have a look at the infection process. NeSSi^2 applications that are installed on the end devices have a set of vulnerabilities (which might be empty). These vulnerabilities could be exploited by malware applications e.g. to install themselves on the corresponding node. Notice, that it is possible to create malware applications that require multiple vulnerabilities for a successful exploitation. Malware applications spread out by sending their code over the network. The according network packets are routed usually over multiple hops to the target node. Every node on the route can potentially analyze the contents of the packet and refuse forwarding (packet dropping). Hence, *detection units* (DU) are most commonly not residing on end devices but on NBRs because of their greater centrality.

Furthermore, the DU might invoke some collaborative actions like signature exchange or raising of alarms. In order to do so, the DUs are organized in groups according to [BCA09]. For that purpose, every node that likes to collaborate with others needs to provide its objective, interests and properties as described by Bye et al. Furthermore, the collaborating nodes register themselves at a global *registry* that assigns them to a group. This might be an appropriate existing group or if such is not available a newly created one.

ISBN 978-3-88579-286-4

3.2 Realization

All devices (incl. the multitude of end devices abstracted by an NBR) must provide information about present vulnerabilities. Detection units additionally need to hold information about their objective, properties and interests as well as detectable malware types. For handling these features with minimal memory consumption, we present an identifier representation.

First, the interests and properties of a collaborating node are not organized hierarchically, opposed to the original approach described by Bye et al. Instead, they are simplified as a set of features which are represented by an identifier. Identifiers range from 0 to 63, so a long integer value can store a set of identifiers by setting the appropriate bits to 1 whilst leaving the other bits at 0. Furthermore, this kind of representation makes set operations quite easy. E.g. the intersection of two sets can be calculated by a very fast bitwise *AND* operation.

This identifier set approach is applied for the vulnerabilities, the detectable malwares, the properties and the interests. Note though, that a DU can have only one objective. Hence, the described concept of id sets is not used for them, instead they are represented by a one-byte id only. Figure 2 shows an example with three connected NBRs with detection units.

Some of the abstracted end devices in *NetBlock2* are shown. Two of them are uninfected but with vulnerabilities, and one is infected with *Malware1*. The end device *ED2* is safe from infection by *Malware1*, because it does not provide the exploited vulnerability combination. This malware could potentially infect *ED1*, though, because the required vulnerabilites 0 and 1 are present on this host (note the binary notation for id sets in the picture). However, the malicious packet would have to pass the *NBR2* which executes a DU that is able to detect packets of kind 1 and drops them. Hence, it is impossible for the uninfected devices in *NetBlock 2* to get infected with *Malware1*.

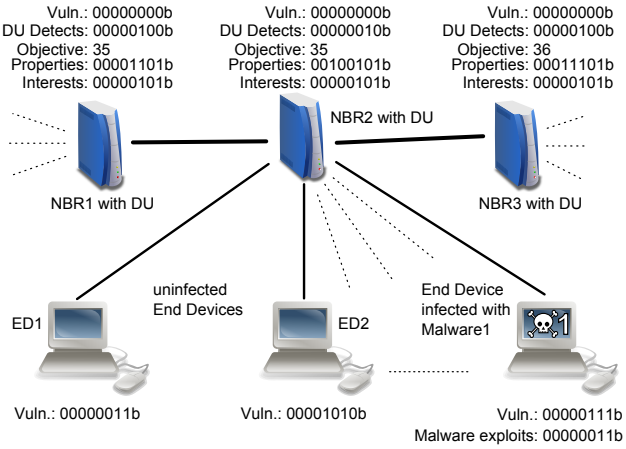

Figure 2: Example for the use of identifier sets with only eight bits of the set representation shown for readability.

The DUs on *NBR1* and *NBR2* share a common objective and hence can potentially collaborate within the same detection group. Whether they really get assigned to the same group depends on the (exchangeable) algorithm for matching interests and properties with each other as well as specified maximum and actual group sizes. Finally, the DU on *NBR3*

cannot find a collaboration partner amongst the depicted devices, because it is the only one with objective 36.

In order to join a detection group, the nodes register at the global *registry* by invoking the *join* method. The node specifies its objective, interests, properties and the maximum size of the accepted group in this step. Then the registry looks up all existing groups with the specified objective and passes them along with the specfied interests, properties and maximum size to a *group matcher*. This returns the group that the node should join. When the node joins, the group-internal routing (i.e. direct neighbors in the overlay network) is adapted accordingly by the *group organizer* instance and the designated port is registered for the receipt of messages. The group object that is received in return by the joining node allows to leave the group, invoke communication methods, or find other members.

3.3 Interface

The framework allows the user to implement his own malware applications, detection units, group matchings and overlay routings. This section describes what is needed to customize a simulation.

When implementing his own application, the user needs to extend the appropriate abstract class shipped with the framework, e.g. *CollabNBR* for collaborating applications running on NBRs or *AbstractNBEApp* for applications running on end devices. The methods for the initial startup, packet receipt, shutdown, as well as the event-handling method can be overridden to influence the application's behavior. By extending the abstract classes, the new applications become able to access e.g. information about the device they are installed on. In the case that the application is installed on an abstracted end device, the corresponding NBR is accessible, too.

The group matcher and group organizer are created as singleton instances according to the specified settings. They implement the interface *GroupMatcher* or *Organizer*, respectively. Whilst the group matcher interface provides just one method for selecting a group according to specified restrictions, the organizer interface contains three methods. The invocation of *organize* sets up the direct communication partners for all the nodes in the group. An update of the communication partners due to node joining is triggered with *organizeJoin*. And *organizeLeave* updates the partners due to a leaving node. Whilst the *organize* method must return a mapping for each node to its neighbors, the latter two methods are allowed to return a *null* value indicating that reorganization should be delayed until the next packet is sent within the group, e.g. because the update is expensive. The Java classes that are used for the group assignment and the calculation of the routing tables are specified by the values for "lsm-matcher" or "lsm-organizer" in the NeSSi^2 properties of the simulation object that is passed to the simulation backend. By providing his own implementation of these interfaces, the user is free to define the group compositions as well as the behavior of the communication overlay almost arbitrarily. This is especially important in distributed systems, because the communication behavior influences the system performance heavily.

ISBN 978-3-88579-286-4

4 Evaluation

In this section, we evaluate the group communication mechanism on the one hand, and the successful simulation of different collaboration behaviors due to different configuration settings on the other.

4.1 Group Communication

Since the communication behavior is important for distributed systems in general, including CIDS, we evaluate the communication behavior of the presented framework with the following setup where different group organizers are compared and contrasted.

A collaboration group with n members is created. Then the first node sends a message via the group object's *sendToAll* method. When every node has received the message, the simulation is finished and the average number of messages per node as well as the maximum number of hops that a message has taken, are measured. Moreover the number of unreachable hosts is estimated.

The simulation is repeated for several crosslinking strategies, implemented by appropriate *GroupOrganizers*.

The communication partners, i.e. the neighbors in the overlay network, are chosen randomly in all the utilized approaches. However, the number of neighbors, the *blocksize*, is estimated differently depending on the number of nodes in the group.

The four approaches on the estimation of the blocksize, that are compared with each other, are $\lceil\sqrt{n}\rceil$, $\lceil log_e n\rceil$, $\lceil log_{10} n\rceil$ and $\lceil 30\% \cdot n\rceil$, where n is the number of nodes in the group.

In order to prevent statistical outliers to falsify the results, each setup was simulated n times in total and the results of each setup were averaged before they were visualized in the charts, where n is the number of nodes within the group.

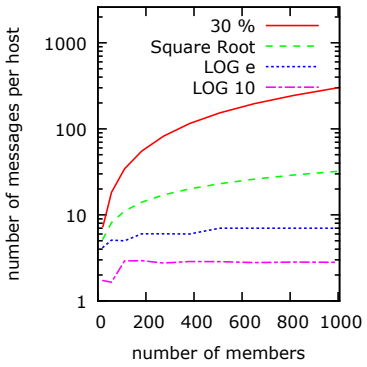

Figure 3: The average number of messages per node

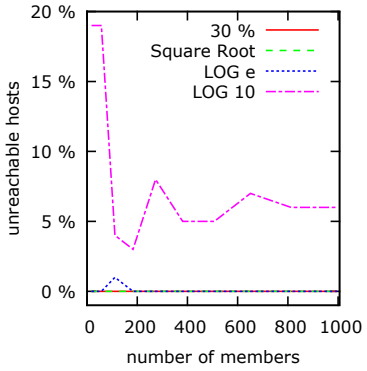

Figure 4: The percentage of unreachable nodes

Figures 3, 4 and 5 compare the number of messages needed per node in order to reach every group member, the number of unreachable nodes in percent as well as the greatest communication distance in terms of maximum hops that a message needed.

In the simulation, each group member was selected to be the original sender of the message once and the values for the number of messages as well as the unreachable nodes were averaged amongst the results for all the members, whereas the maximum hops shown in Figure 5 are the maximum hops of all these simulation runs.

As can be seen in Figure 3, the $\lceil 30\% \cdot n \rceil$ approach created the highest number of messages by far. But on the other hand, every group member did receive the notification after a maximum of only 2 hops, as can be seen in Figure 5. The reason for this is the quite large number of neighbors, which of course reduces the number of maximum hops, but also results in a large number of nodes receiving the message at about the same time, and triggering a forward on each of them leading to a huge message overhead.

The opposite behavior is shown by the $\lceil log_{10} n \rceil$ approach, where the numbers of sent messages per node are quite small, but the maximum number of hops has by far the highest value. The reason for that lies in too few neighbors, which also results in several unreachable nodes (cf. Figure 4).

That logarithmic approaches do not generally perform that bad can also be seen in the figures, as the $\lceil log_e n \rceil$ performs much better, regarding the maximum hops and the number of messages per node. Though there are cases, where some nodes could not be reached, this is by far not as significant as in the $\lceil log_{10} n \rceil$ approach.

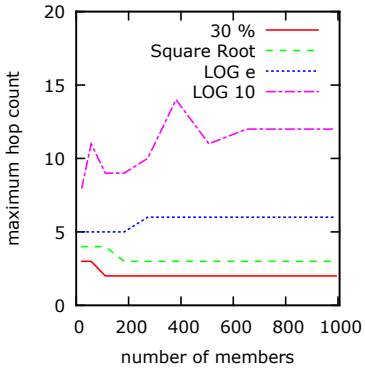

Figure 5: Maximum hops that messages needed in order to reach every group member (maximum of all simulation runs)

Further analysis of these results is not conducted here, as the purpose was to show that the underlying communication behavior can be selected by configuring CIDE accordingly and the results seem plausible as discussed above.

4.2 Case Study – Collaboration for Security

The collaboration mechanism of the framework is evaluated with a drive-by-download scenario where hosts get infected due to visiting a website hosting malicious content.

This is an interesting aspect, because it is one important way to spread malware like trojan horses, adware, or malware granting full remote access to the infected machine, often supported by spam mails linking to malicious web sites. In [PMM+07], Provos et al. analyzed websites indexed by Google. They evidently found malware on 450,000 URLs out of several billion and additional 700,000 URLs seemed malicious, but had lower confidence.

In the evaluation scenario we monitor the number of infected hosts during simulation, hence the activity of the malware after installation is not regarded here, nor is the background traffic.

An infection probability of 0.01 is assumed, though it seems pretty high compared e.g.

to the results of Google's analysis in [PMM+07]. The high probability, however, solely speeds up the simulation and because we are only interested in comparing and contrasting several CIDS setups, we consider this value appropriate for the probability of infection when a new website is visited. Moreover, we assume that the users browse the contents of a visited website more or less thoroughly, i.e. they spend more or less time on one site before loading the next, and that infections only occur at the beginning of this time frame. But, the time frames that are spent on a single site vary a lot, hence we assume the range to be between a few seconds and a few hours.

The network for this scenario consists of 4 autonomous systems, each containing exactly 4 NBRs that are connected with each other and the border routers (cf. Figure 1) with an average link degree of 3.5. Every NBR contains 50 end devices.

There are three different types of applications deployed in the network. Every end device is set up with a *Drive-by-Download application*. It checks randomly every 1–500 ticks with a probability of 0.01, whether the device gets infected due to loading a malicious website. When an infection occurs, one of the *malicious web servers* is chosen and a request is sent. When the reply from the web server is received, the corresponding malware id is added to the list of infections. If the host is infected with every malware present in the network, it stops the infection checking. There are five web servers set up on end devices in different access networks, each hosting another malware type. Moreover, every NBR has installed a detection unit initially equipped with the signature of one of the five malwares.

Hence, the malicious packets that are sent by the web servers as replies are passing DUs on the route back to the browsing client. If one of the DUs detects the packet as malicious, it will be dropped and not reach the requesting node.

In order to enhance the signature base, the DUs collaborate in groups of five. Every time a malicious packet was detected, a signature update is sent to the other group members, as long as these were not updated before with that particular signature from that particular node.

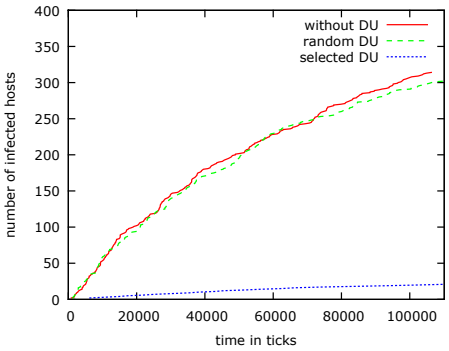

Figure 6: Infections for one malware type.

The scenario is simulated in three different setups. Figure 6 shows how the number of infections develop over time for one of the five malwares. This one malware type is representative, because all types are distributed equally within the network and the same is true for the DUs.

Without any DU, the hosts get infected very quickly. When the DUs form random groups, they are able to drop some of the malicious packets, resulting in marginally less infections on the hosts. In the third scenario, nodes benevolently join the most benefiting group, i.e. a group that does not know the provided signature, yet. This results in far more efficient signature updates as can be seen from the dotted line in Figure 6. The difference between the latter two cases shows that drastically because the maximum group size was chosen to

be exactly the number of malwares. Hence, every initial duplicate of a signature within a group locks out another one.

5 Related Work

Evaluating CIDS in a productive environment is not feasible because the generation of malicious network traffic bears the risk of harming a productive system. However, several approaches for the evaluation can be followed. For instance, the Planet-Lab[2] provides a global research network for the development of new network services. The utilization of a compute cluster like the DETERlab testbed [WS07] that allows for the analysis of detection approaches at a certain abstraction level would be another approach. Nonetheless, network simulations achieve a much higher abstraction level, which generally allows for easier rapid prototyping. Usually, the developing system needs to be analyzed and reworked during the development process which is easier to achieve by using simulators.

A well-known network simulator is ns-2 [RPW09]. It is a discrete-event simulator with support for simulation of TCP as well as multicast protocols. *OTcl* is used as the programming language for the simulation setup and the implementation of the behavior of the simulated nodes[3]. There is no graphical user interface and hence no visual representation of the network for monitoring its status on replay. On the other hand, it supports wireless networks which is very interesting especially for the examination of sensor networks. The Global Mobile Simulator (GloMoSim) [RPW09] focuses on wireless networks. The behavior of the nodes is programmed in *Parsec*, a C-based simulation language with support for parallel event simulation. Unlike ns-2, GloMoSim provides a Java-based visualization tool. GloMoSim is a commercial and closed-source tool, but they provide a research version for academic use. Simulators especially designed for large-scale networks are e.g. TeD, SSF, USSF and SWiMNet [RPW09]. However, none of these simulators provides built-in functionality for the collaboration of network nodes.

6 Conclusion

We presented *CIDE*, a simulation framework that allows for the evaluation of CIDS. For that purpose, we extended a previously developed network simulator (NeSSi[2]) with functionality for managing detection groups. Moreover, the group-internal communication is organized by the framework as well. Then, we have evaluated the functionality of the *CIDE* with a signature-exchange scenario. At the same time, we showed that the organization of detection groups is beneficial for collaborative IDS.

[2] http://www.planet-lab.org
[3] http://www.isi.edu/nsnam/ns/ns-documentation.html

References

[BCA09] Rainer Bye, Seyit Ahmet Camtepe, and Sahin Albayrak. *Collaborative Computer Security and Trust Management*, chapter Teamworking for Security: The Collaborative Approach, page 342. Reference. Information Science Reference, 1 edition, December 2009.

[Cas05] Manuel Castells. *The Network Society: A Cross-Cultural Perspective*. Edward Elgar Pub, July 2005.

[GJMB05] Joaquin García, Michael A. Jaeger, Gero Muehl, and Joan Borrell. Decoupling Components of an Attack Prevention System Using Publish/Subscribe. In *Intelligence in Communication Systems*, volume 190 of *IFIP International Federation for Information Processing*, pages 87–97. Springer Boston, 2005.

[GTDVMFSC07] Pedro García-Teodoro, Jesús Díaz-Verdejo, Gabriel Macía-Fernández, and Leovigildo Sánchez-Casado. Network-based Hybrid Intrusion Detection and Honeysystems as Active Reaction Schemes. In *IJCSNS International Journal of Computer Science and Network Security*, volume 7, pages 62 – 70, 18071 Granada Spain, October 2007. University of Granada.

[KKK05] Sachin Katti, Balachander Krishnamurthy, and Dina Katabi. Collaborating Against Common Enemies. In *IMC '05: Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, pages 34–34, Berkeley, CA, USA, 2005. USENIX Association.

[PMM+07] Niels Provos, Dean McNamee, Panayiotis Mavrommatis, Ke Wang, and Nagendra Modadugu. The Ghost in the Browser Analysis of Web-Based Malware. In *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, pages 4–4, Berkeley, CA, USA, 2007. USENIX Association.

[RPW09] Muhammad Azizur Rahman, Algirdas Pakstas, and Frank Zhigang Wang. Network modelling and simulation tools. *Simulation Modelling Practice and Theory*, pages 1011–1031, 2009.

[SBC+10] Stephan Schmidt, Rainer Bye, Joel Chinnow, Karsten Bsufka, Ahmet Camtepe, and Sahin Albayrak. Application-level Simulation for Network Security. *SIMULATION*, 86(5-6):311–330, 2010.

[WS07] Nicholas Weaver and Robin Sommer. Stress testing cluster Bro. In *Proceedings of the DETER Community Workshop on Cyber Security Experimentation and Test on DETER Community Workshop on Cyber Security Experimentation and Test 2007*, pages 9–9, Berkeley, CA, USA, 2007. USENIX Association.

[YBJ04] Vinod Yegneswaran, Paul Barford, and Somesh Jha. Global Intrusion Detection in the DOMINO Overlay System. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2004*, 2004.

[ZLK10] Chenfeng Vincent Zhou, Christopher Leckie, and Shanika Karunasekera. A survey of coordinated attacks and collaborative intrusion detection. *Computers & Security*, 29(1):124 – 140, 2010.