Cost-aware Systemwide Intrusion Defense via Online Forensics and On-demand Detector Deployment

Saman A. Zonouz†, Kaustubh R. Joshi‡, and William H. Sanders†
University of Illinois†and AT&T Labs Research‡
{saliari2, whs}@illinois.edu, kaustubh@research.att.com

ABSTRACT

Balancing the coverage benefits of deploying multiple types of intrusion detection systems against their performance and false alarm costs is an important problem with practical ramifications for runtime security policy. In this position paper, we present an approach to "on-demand" deployment of intrusion detection systems by balancing detection coverage against cost and deploying an IDS only when it is needed. The proposed approach relies on often easy to detect symptoms of attacks, e.g., participation in a botnet or DDoS, and works backwards by iteratively deploying increasingly more localized and powerful detectors closer to the initial attack vector. We accomplish this by characterizing multiple IDS systems in a uniform framework based on their costs and detection capabilities and integrating them, for the first time, into an online system-wide forensics framework. We develop the basic elements of the framework and give an example of its envisioned operation.

Categories and Subject Descriptors

C.2 [**COMPUTER-COMMUNICATION NETWORKS**]: General

General Terms

Security

1. INTRODUCTION

Modern computer systems are often susceptible to many types of vulnerabilities, ranging from buffer overflows, to parser errors, to brute-force credential-guessing attacks. The large number of vulnerability types and the correspondingly numerous types of intrusion detection systems available pose an operational dilemma for system deployment engineers and administrators. How much monitoring is enough? Intrusion detection mechanisms carry a performance price tag not just in terms of the resources they themselves consume, but also in terms of the cross-validation burden they impose by often producing false positives that must be investigated. Thus, deploying all possible types of detectors in the hopes of catching intruders that may exploit a small number of vulnerability types that may actually exist in a system is usually not cost-effective in many non-critical applications.

In this position paper, we propose an alternative to continuous security monitoring for a wide range of security threats by proposing an "on-demand" solution to intrusion detection that explicitly balances the attack coverage of an IDS against its cost and deploys detection systems only when they are needed. Our approach relies on the observation that although it may be difficult to notice an exploit inexpensively and accurately close to its infection point, the ultimate consequences of attacks are often far easier to detect. For example, inexpensive in-network scanning techniques can detect participation in a DDoS attack, botnet, or infection by a worm. Malware scanners can detect the artifacts produced by previously known payloads (even if the attack vector is unknown), and in-host anomaly detectors can detect deviations in a system's performance. Therefore, we envision a system with a hierarchy of IDSes at its disposal, ranging from cheap detectors with limited attack detection capabilities and poor detection latency (e.g., the abovementioned attack consequence detectors) to expensive but more localized and powerful IDSes, such as memory access checkers and taint trackers.

The cheapest detectors are deployed continuously, and when they detect the consequences of an attack, we propose to roll back the system to a clean checkpoint, and employ a forensics algorithm to determine possible paths the attack could have taken to reach its detection point. During the forensics, an optimization problem is solved based on the outputs of the already deployed detectors, the cost and coverages of the unused detectors, and the paths an attack might have taken. Based on the results of this optimization problem, additional intrusion detectors (that are possibly more expensive and/or localized) are deployed to catch and detect the attack at an earlier stage the next time that it is attempted. By iteratively repeating this process, we propose to deploy detectors successively closer to the initial attack vector until such time that the attack can be stopped by automatic prevention techniques such as input signature generation.

We believe our effort is one of the first to propose online selection of security detection mechanisms by balancing their cost against their coverage. By invoking mechanisms "on-demand" only when they are needed to forensically evaluate an attack that is already known to exist for the target system, we can utilize expensive mechanisms such as buffer bounds checking and taint tracking that are known to have good coverage characteristics. Furthermore, since we propose a whole-system solution, it can initiate and configure its forensics analysis based on a wide range of attack consequences that may be many processes and files away from the initial attack vector. Finally, the models we propose are designed so that new security mechanisms can be easily integrated into the decision making, thus providing a flexible and extensible detection solution.

2. APPROACH

We begin by describing the architecture and overall operation of the proposed model. In order to detect attacks' damage to the system, an initial set of intrusion detection systems (IDSes), i.e.,

Table 1: Detection Algorithm Categorization

Detection Policy	Symbol: Mechanism	Cost	Detector
Information flow analysis	Tnt: Taint tracking	Very High	TEMU [13]
Input investigation	FW: Feature-based packet monitoring	Very Low	Firewalls [4]
	Snrt: Content-based packet monitoring (stateless)	Medium	Snort [11]
	App: Application-based IDS (stateful)	Medium	Secerno [1]
Execution monitoring	ClSt: Control Violation: call stack monitoring	High	CS monitoring [6]
	CtFl: Control Violation: control flow integrity monitoring	High	CFI [3]
	DtFl: Data Violation: data flow monitoring	Very High	MemCheck [10]
Consequence detection	AV: Malicious code: executable integrity checking	Low	ClamAV [9]
	Hst: Host-based detection systems	Low	Samhain [15]
	Stat: Statistical anomaly-based	Low	Zabbix [2]

called "attack consequence detectors," would be deployed. Attack consequence detectors are expected to be lightweight detectors that can operate continuously or periodically and detect the eventual symptoms of an intrusion. We assume that consequence detectors cannot be turned off during the attack. Examples include in-network DDoS, BotNet, and worm detection mechanisms, file-integrity checkers run independently of the target machine, or statistical system call/network anomaly detectors. The output of an attack consequence detector is a process, port, or file that exhibits the symptoms of an attack.

Briefly, the core idea of the proposed model is to dynamically (based on ongoing attacks) reconfigure and deploy the required subset (and not all) of the available IDSes, with the lowest posible detection cost in the system. Table 1 shows different types of IDSes categorized by the mechanisms they use to detect misbehaviors. The table also shows the deployment cost for each IDS, i.e., its performance overhead on the system. For each detection mechanism, a real-world software tool is also provided that could be used in our model. Next, the question is "what is the construct that will be needed to balance detection coverage vs. cost in the system, given the available IDSes and their individual costs?" We first need to define the detection coverage for each IDS; in other words, what vulnerability exploitations can each IDS detect? This is answered, in our model, by the detector-capability matrix, which indicates the ability of a given IDS to detect various vulnerability exploitation types. The matrix is defined over the Cartesian product of the vulnerability type set and the set of IDSes, and shows how likely it is that each IDS could detect an exploitation of a specific vulnerability type.

Table 2 illustrates a sample detector capability matrix for the different IDSes and vulnerability types. In particular, each row in the table represents a specific intrusion detector type, and each column in the table addresses a specific vulnerability type. The notations used for vulnerability types are as follows. Buff is Buffer overflows; DngPtr is Dangling pointers; FmtStr is Format string bugs; ShlMC is Shell metachar bugs; SQLIn is SQL injection; CodIn is Code injection; DirTrv is Directory traversal; CSS is Cross-site scripting; HttpHdr is HTTP header injection; HttpRsp is HTTP response splitting; TcTu is Time-of-check-time-of-use; SymRc is Symlink races; CSFor is Cross-site request forgery; ClkJk is Clickjacking; FTPBnc is FTP bounce attack; WrnFtg is Warning fatigue; BlmVic is Blaming the Victim; Race is Race Conditions; PwdDic is Password Dictionary; and Encypt is Encryption Bruteforce.

Each element in Table 2 encodes the detection capability of a particular detector type in identifying individual vulnerability type exploitations. The notations used in the table are as follows: N means that the detection technique cannot detect the exploit; L means that it can only detect a small percentage of these exploits; M means that the detection capability is medium; H means that the exploit is detected with high probability; and C means that the exploit is definitely detected by the detection technique.

Attack Graph Templates. First, using the detection cost of each IDS, the detector-capability matrix, the privilege domains of the system's processes and files, and information about inter-process communication via sockets, shared-memory, and files, an extended attack graph called the *attack graph template* (AGT) should be constructed. Traditionally, an attack graph for a computer system represents a collection of known penetration scenarios according to a set of known vulnerabilities in the system [12]; however, an AGT includes all *possible* attack paths, modeled as a series of privilege escalations accomplished through exploitation of specific types of vulnerabilities, from a state in which an attacker has no access to the system, to a state detectable by the consequence detector. AGTs only show what is possible given the system configuration and communication patterns, not what vulnerabilities actually exist. IDSes must be invoked on the processes to provide evidence of actual vulnerabilities. As an example, the attack graph template for a web server addresses the possibility that the server application, e.g., Apache, might be vulnerable to buffer overflow attacks, even if there is no such known vulnerability in the application.

Figure 1 shows an AGT designed for a sample LAMP[1] web server. The AGT should be designed only once for a given system. For purposes of clarity, we have limited the tree to address attacks that include sensitive data-based file modification as their final goal. Moreover, due to space limitations, we have bunched together different possible vulnerability types in each application using the application-vulnerability mapping; therefore, a single bunched edge in the figure may represent several edges in the actual AGT. System states are represented by ovals, and the rectangles represent the adversarial actions, i.e., vulnerability exploitations. As illustrated in the figure, the attacker has no control over the system while in the initial state, and can go through several paths to achieve his or her malicious goal.

Intrusion Forensics. During normal operation, the consequence detectors should be turned on, and incremental snapshots of the system should be periodically collected. Deploying the initial consequence detectors guarantees that all the attack paths, from the initial state s_o to one of the goal states $s \in S_s$ in the AGT, is cut (detected) in at least one detection point exploit e^{dp}. In other words, if an attacker tries to compromise the machine while it is operating with the initial IDS set turned on, he or she would be detected and stopped by at least one of the deployed intrusion detectors.

When one of the consequence detectors produces an alert accompanied by the name of a process, file, or socket, the alert would be mapped to a node in the AGT. Then, an iterative forensics analysis process would be started, using the AGT as a starting point. It uses the outputs of the intrusion detectors (starting with the consequence detectors) to determine which paths in the AGT can be implicated or eliminated, and produces a refined AGT that is a subset of the original one. To do so, it relies on the detector-capability matrix. We assume that each IDS either detects or misses an incident. In particular, the intrusion forensics analysis consists of re-creating (or waiting for future similar attacks if the snapshot/replay mechanism is not supported) and analyzing attacks that have previously occurred. More specifically, once an attack is detected at one detection point e^{dp}, the forensics analysis would attempt to figure out the complete exploit sequence E_a through which the attacker has

[1]LAMP is an acronym for Linux, Apache HTTP Server, MySQL database, and PHP, principal components to build a web server.

Table 2: The Detector-Capability Matrix

	Buff	DngPtr	FmtStr	ShlMC	SQLIn	CodIn	DirTrv	CSS	HttpHdr	HttpRsp	TcTu	SymRc	CSFor	ClkJk	FTPBnc	WrnFtg	BlmVic	Race	PwdDic	Encrypt
Tnt	H	M	H	M	C	L	H	C	M	M	L	L	H	H	H	N	N	N	N	N
FW	L	N	L	N	N	L	N	N	L	L	N	N	L	L	L	N	N	N	M	M
Snrt	M	N	M	M	M	M	M	N	M	M	N	N	N	N	M	N	N	N	H	H
App	H	L	H	H	H	H	H	L	C	C	N	N	N	N	H	N	N	N	H	H
ClSt	C	M	H	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
CtFl	C	H	H	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
DtFl	L	L	L	M	C	L	H	C	M	M	L	L	H	H	H	N	N	N	N	N
AV	N	N	N	N	N	M	N	N	N	N	H	H	N	N	N	L	L	L	N	N
Hst	L	L	L	N	N	H	N	N	N	N	H	H	N	N	H	M	M	L	N	N
Stat	M	M	L	N	N	L	N	N	N	N	N	N	N	N	H	N	N	N	H	H

gone from the initial state s_0 in the AGT to the detection point e^{dp}, e.g., $s_0 \xrightarrow{e_i} s_m \xrightarrow{e_j} s_n \ldots s_o^{dp} \xrightarrow{e^{dp}} s^{dp}$. The determined attack path may include exploitation(s) of one or more previously unknown vulnerabilities in the system.

To provide a precise forensics analysis, we in fact need to traverse the time dimension back and forth. In particular, we are interested in time instances when attack steps happened, e.g., $t_{e_i}, t_{e_j}, \ldots, t_{e^{dp}}$ in the example above. Two conceptually similar forensics solutions are provided for different situations. 1) If the infrastructure supports system-wide restore/replay, the restore/replay mechanism would be used to iteratively restore the system snapshot in the past and replay the whole system until the detection point $t_{e^{dp}}$; during each replay, a different IDS would be turned on to gather more evidence about the attack, i.e., to identify which unknown vulnerabilities were exploited during the attack. And 2) if the system-wide restore/replay is not supported by the infrastructure, various IDSes would be deployed to gather more evidence about the attack, but instead of making use of the restore/replay mechanism, the model should wait for the attacker to repeat the attack in the future. Note that during the forensics analysis in the second situation, in which there is no replay mechanism provided, the attacker could possibly cause more damage to the system than he could in the first situation, since in the second situation, the attack would actually be repeated several times by the attacker until the algorithm identifies the attack path and blocks future identical attacks.

During each attack replay, the best new set of detectors, which provide the best balance of cost vs. detection coverage based on the AGT, would be picked and deployed (in addition to the consequence detectors). The goal is to pick detectors that will best help refine the AGT even further, and ideally isolate a single attack path starting from the initial exploit to the detected attack consequence. After selecting a new set of detectors, the system should be rolled back to a previous snapshot, deploy the detectors, and wait for a repeat attempt of the attack. On the next alert, from either the consequence detectors or the newly deployed detectors, the forensics engine is reinvoked. Once the same attack is repeated, it would be detected sooner than the previous one, and the actual detector outputs provide more evidence to help the forensics engine further refine the AGT. If the alert is due to a new attack that cannot be explained by the refined AGT, the forensics engine would refine the original AGT using the detector outputs. In this manner, a repeated attack would be detected successively closer to its exploit source until eventually, the detection point would be close enough to the actual attack vector that a mitigation mechanism such as Sweeper [14] or RRE [16] can then be used to block the attack.

Monitor Selection. Once the forensics analysis figures out the attack path from the initial state s_0 to the detection point in the AGT, the model would select the cost-optimal set of IDSes to be deployed in the system permanently from then on.

In particular, the IDS selection problem here consists of two subproblems regarding known and unknown system vulnerabilities. First, the algorithm should select a set of IDSes to deploy that can cooperatively detect known vulnerability exploitations, i.e., those identified at some point by the forensics analysis. The decision-making upon the IDS set would be accomplished by optimizing the trade-off between the overall detection cost of the selected IDS set, and the damage cost by the attackers due to unmonitored vulnerability exploitations if their corresponding IDSes are not turned on. More specifically, an IDS set D^* is called optimal if it satisfies the equation $D^* \leftarrow \arg\min_{D_i \subseteq D} \frac{\texttt{PrfCost}(D_i) \times \texttt{DmgCost}(AGT, D_i)}{\texttt{Cap}(D_i)}$, where `PrfCost` represents performance overhead due to deployment of a subset of IDSes D_i (D denotes the set of available IDSes); `DmgCost` formulates potential damages due to missed exploitations in AGT after turning on the IDSes D_i; and `Cap` denotes the detection capability of the IDS set (in other words, what portion of the attack paths from the initial to the goal state in AGT would be monitored if the IDSes D_i are deployed).

The second subproblem addresses the rest of the vulnerability exploitations, i.e., state transitions in the AGT that are not yet known but might potentially exist in the system and therefore should be taken care of. Regarding the unknown vulnerability exploitations in the AGT, the set of IDSes should be picked such that they cover (detect) the maximum possible number, i.e., not necessarily all, of the unknown exploitations in the AGT while keeping the overall detection cost below a predefined overhead threshold that indicates how much system resources the administrators are willing to devote to IDSes. The main reason to have a cost threshold is that the number of unknown vulnerabilities in the system could be so large that if all of them were addressed by IDS deployments, the system's main operation and availability could be significantly affected.

If the size of an AGT for a system is manageable in practice, then the attack path coverage measure `Cap` could be calculated directly, i.e., by analyzing all attack paths. However, it is sometimes infeasible to calculate the `Cap` by considering all attack paths in AGT. The proposed model could employ Hoeffding's inequality to estimate `Cap` by considering a sufficient number of randomly sampled attack paths in AGT in order to guarantee the confidence level of the `Cap` function.

Hoeffding's inequality [7]: Let $X_1, \ldots, X_n$ be independent and identically distributed random variables, where all X_i's are bounded so that $X_i \in [b_l, b_u]$, and let $\hat{X} = \frac{1}{n} \cdot \sum_{i=1}^{n} X_i$. Then, the following inequality holds: $Pr(|\hat{X} - E[X]| \geq \varepsilon) \leq 2.\exp\left(-\frac{2n^2\varepsilon^2}{\sum_{i=1}^{n}(b_u - b_l)^2}\right)$. Generally, Hoeffding gives an upper bound on the probability that the sum of random variables will deviate from its expected value. Using the Hoeffding inequality, we calculate the least sufficient number of random paths n to estimate attack path coverage with the specified confidence level ε guaranteed. In our case, random variables X_i take on either 0 or 1 depending on whether d_i cut the attack path under consideration; therefore, $b_l = 0$ and $b_u = 1$. Our empirical estimate, i.e., `Cap`, is represented as $\hat{X}$ in the inequality. $E[X]$, in the inequality, represents the actual portion of attack paths that get detected by d_i; as mentioned before, accurate computation of $E[X]$ requires individual analysis of all attack paths in the AGT.

Case Study. Having discussed the proposed model, here we describe an attack scenario, and explain how the proposed approach would work against the attack using the AGT for a LAMP system (discussed earlier).

The attack will first exploit a buffer overflow vulnerability in the web server from a remote host. The exploit would give the attacker a shell on the system with user-level privilege, as the web server is assumed to be running in the user domain. In the second step of the attack, a local brute-force password attack is launched to obtain the root domain access. Finally, one of the sensitive root domain files (selected to be monitored by one of the consequence detectors) is modified, causing the consequence detector to send out alerts.

The alerts make the model start its forensics analysis. Initially, the system's AGT (whose summarized version is shown in Figure 1) consists of 110 attack paths from the initial to final state. As mentioned, only one of the paths was traversed during the attack;

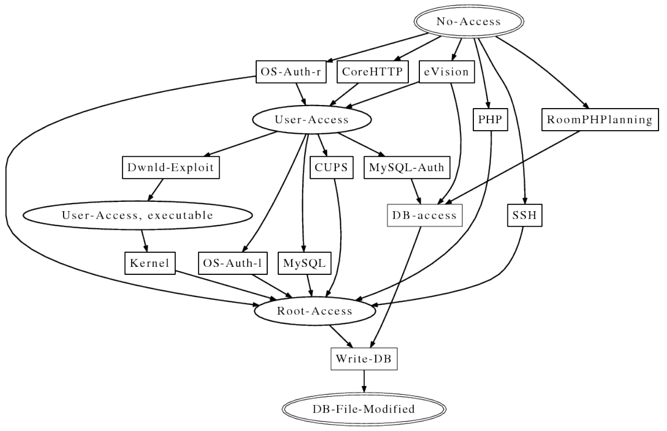

Figure 1: The Scenario Graph of a LAMP Setup

however, the engine does not initially know that. Therefore, during the intrusion forensics analysis, it attempts to isolate the one actually traversed.

As mentioned earlier, there are two main factors affecting the order in which IDSes are deployed: cost and capability (coverage) of each IDS. We manually did all the calculations, and concluded that, Snort would be picked as the first IDS according to its cost and the detector-capability matrix. Therefore, a clean snapshot of the system would be restored; Snort would be turned on; and the system (and hence the attack) would be replayed by the model. As we know the actual attack path, Snort could not detect any misbehavior using its signature rulesets. Then, the AGT of the system is pruned according to the used IDS (Snort). The refined AGT turns out to include 78 paths from its initial to goal state. Similarly, the clean snapshot of the system is restored and replayed, with the ClamAV turned on. After second iteration of the forensics analysis, the pruned AGT consists of 37 paths; however, no exploitation would be detected by the ClamAV antivirus, because indeed no virus is downloaded during the attack.

Next, the Zabbix anomaly detector would be deployed. As the brute-force attack overuses the system resources, Zabbix flags a suspicious anomaly. The refined AGT at this stage consisted of 14 attack paths. In the next forensics iteration, Libsafe, which causes very low performance overhead on average, would be deployed. Once the attack was replayed, Libsafe would be able to detect the entry point of the attack, i.e., the buffer overflow exploitation. Consequently, Samhain, Snort, ClamAV, Zabbix, and Libsafe cooperatively could identify the exact traversed (previously unknown) attack path in the AGT. Having identified the attack path in AGT, a clean snapshot of the system would be restored, and the system would continue its normal operation. Additionally, the set of IDSes that contributed towards the forensics analysis, i.e., Libsafe and Zabbix, are selected to be turned on permanently to block similar future attacks that try to exploit the vulnerabilities just identified by the forensics analysis.

3. CHALLENGES

Having discussed the idea of balancing detection cost vs. its coverage, here we explain several challenges that still remain before we can make the proposed model practical and widely deployable. We hope to address these challenges in future work and validate our approach using a comprehensive set of attacks in a real-world environment.

First, the construction of AGTs must be automatic for the approach to be useful for systems without dedicated security teams. We believe that outputs from forensics tools can help. For example, BackTracker [8] can produce graphs of all communications between system components that lead to a particular target process, file, or port. We could use such graphs as templates to construct the AGT starting from a consequence detection trigger. Additionally, our detector selection ideas must account for the fact that security detectors often have false positives and false negatives. Therefore, the simple binary output model used in this paper will have to be generalized to a model that uses information about detector quality to probabilistically score paths in the AGT instead of completely pruning them. To truly close the loop from online forensics to attack vector detection to mitigation, our approach will require integration with attack prevention and mitigation mechanisms such as attack signature generation and blocking (using tools such as [5] and [14]) or automatic quarantining.

4. CONCLUSION

In this paper, we presented a model for a cost-aware intrusion detection system that would use online forensics and on-demand IDS deployment. The proposed model would enable systems to defend against attacks that exploit various classes of previously unknown vulnerabilities. Upon detection of the high-level consequences of an attack, an automated attack-graph-template-based forensics analysis is used to iteratively deduce an attack path, comprising a sequence of vulnerability exploitations, that led to the detected consequence. An optimal set of monitors is then deployed to allow future attacks using the same exploit to be detected close enough to the intrusion point for automated mitigation mechanisms to be utilized. Our experiments show that our framework can deploy off-the-shelf IDSes only when they are needed and help protect systems against previously unknown vulnerabilities with minimal snapshotting overheads during normal operation.

Acknowledgment

This material is based upon work supported by the Department of Energy under Award Number DE-OE0000097. The authors would also like to thank Jenny Applequist and Aashish Sharma for their revision and valuble comments on IDSes, respectively.

5. REFERENCES

[1] Secerno available at http://www.secerno.com/, 2010.
[2] Zabbix available at http://www.zabbix.org/, 2010.
[3] M. Abadi, M. Budiu, Ú. Erlingsson, and J. Ligatti. Control-flow integrity. In *CCS*, pages 340–53, 2005.
[4] D. Chapman. *Cisco Secure PIX Firewalls*. 2001.
[5] M. Costa, M. Castro, L. Zhou, L. Zhang, and M. Peinado. Bouncer: securing software by blocking bad input. In *SOSP*, pages 117–30, 2007.
[6] H. Feng, O. Kolesnikov, P. Fogla, W. Lee, and W. Gong. Anomaly detection using call stack information. In *IEEE-S&P*, page 62, 2003.
[7] W. Hoeffding. Probability inequalities for sums of bounded random variables. *JASA*, 58(301):13–30, 1963.
[8] Samuel T. King and Peter M. Chen. Backtracking intrusions. *SIGOPS Oper. Syst. Rev.*, 37(5):223–36, 2003.
[9] T. Kojm. Clamav: http://www.clamav.net/, 2009.
[10] N. Nethercote and J. Seward. Valgrind: A program supervision framework. *Elec. Notes in Theoretical Com. Sci.*, 89(2):44 – 66, 2003. RV Workshop.
[11] M. Roesch. Snort - lightweight intrusion detection for networks. In *USENIX-LISA*, pages 229–38, 1999.
[12] B. Schneier. Attack trees. *Dr. Dobb's Journal*, 1999.
[13] D. Song, D. Brumley, H. Yin, J. Caballero, I. Jager, M. Kang, Z. Liang, J. Newsome, P. Poosankam, and P. Saxena. BitBlaze: A new approach to computer security via binary analysis. In *ICISS*, 2008.
[14] J. Tucek, J. Newsome, S. Lu, C. Huang, S. Xanthos, D. Brumley, Y. Zhou, and D. Song. Sweeper: A lightweight end-to-end system for defending against fast worms. *SIGOPS Oper. Syst. Rev.*, 41(3):115–28, 2007.
[15] B. Wotring, B. Potter, M. Ranum, and R. Wichmann. *Host Integrity Monitoring Using Osiris and Samhain*. Syngress Publishing, 2005.
[16] S.A. Zonouz, H. Khurana, W.H. Sanders, and T.M. Yardley. RRE: A game-theoretic intrusion Response and Recovery Engine. In *DSN*, pages 439–48, 2009.