ijecs open access www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 3 Issue 1, Jan 2014 Page No. 3660-3667

Survey of Defense-In-Depth Intrusion Detection Framework in Virtual Network System

Rohini, Ratnavel Dr. M. Kamarajan

Subramaniam College Of Engineering, M.E., Ph.D., Ratnavel

Subramaniam College Of Engineering.

Abstract-To prevent vulnerable virtual machines from being compromised in the cloud. A multiphase distributed vulnerability detection, measurement, and countermeasure selection mechanism called NICE. Built on attack graph-based analytical models and reconfigurable virtual network-based countermeasures. Attackers can explore vulnerabilities of a cloud system and compromise virtual machines to deploy further large-scale Distributed Denial-of-Service (DDoS). Open Flow network programming APIs to build a monitor and control plane over distributed programmable virtual switches to significantly improve attack detection and mitigate attack consequences. The system and security evaluations demonstrate the efficiency and effectiveness of the proposed solution.

INTRODUCTION

Network and security management has to assure uninterrupted access to the communication infrastructure. With growing networks and increasing amount of transported data, it gets more and more complicated to supervise the operation of the communication systems. Sometimes computer networks are not well protected against attacks from the outside, so additional surveillance may be necessary. But even well protected networks need surveillance. A lot of these networks are threatened from the inside. Intrusion Detection Systems (IDSs) help securing these networks. This paper focuses on a tool for visualizing and detecting anomalies of the traffic structure. Several distributed Denial of Service attacks have shown the necessity of better protecting computers and networks connected to the Internet. Due to widely available attack tools, attacks of this kind can be carried out by persons without in-depth knowledge of the attacked system. Insufficient protected Open University networks are an example for networks that need additional surveillance. These networks often include vulnerable computers and offer high bandwidth connections to the Internet. These features are the reason why attackers are interested in these networks. The machines in these networks are not the goal of the attacks. Normally, they do not contain interesting information for the attacker, but they are suitable for scanning other networks and starting (for example) Denial of Service attacks.

The Internet is increasingly important as the vehicle for global electronic commerce. Many organize- tins also use Internet TCP/IP protocols to build intra-networks (intranets) to share and disseminate internal information. A large scale attack on these networks can cripple important world-wide Internet operations. The Internet Worm of 1988 caused the Internet to be unavailable for about _vet days Seven years later, there is no system to detect or an- laze such a problem on an Internet-wide scale. The development of a secure infrastructure to defend the Internet and other networks is a major challenge. In this paper, we present the design of the Graph-based Intrusion Detection System (Girds).

Girds' design goal is to analyze network activity on TCP/IP networks with up to several thousand hosts. Its primary function is to detect and anal- lyre large-scale attacks, although it also has the ca- ability of detecting intrusions on individual hosts. Girds aggregates network activity of interest into
Activity graphs, which are evaluated and possibly re- ported to a system security ocher (SSO). The hiker- archival architecture of Grids allows it to scale to large networks. Grads is being designed and built by the authors using formal consensus decision-making and a well- documented software process. We have completed the Grids design and have almost _knishes building a prototype.

This paper is organized as follows. Brier describes related work on intrusion detection systems and motivates the need for Grids'. Section 1.2 discusses classes of attacks that we expect to detect. In the simple Grids' detection algorithm is described, followed by a more detailed discussion in has a treat- meant of the hierarchical approach to scalability and discusses how the hierarchy is managed. outlines the policy language. Covers some limitations of Girds. Finally, presents conclusions and discusses future work.

Network security is a complicated subject, historically only tackled by well-trained and experienced experts. However, as more and more people become ``wired", an increasing number of people need to understand the basics of security in a networked world.

1**.above the Clouds: A Berkeley View of Cloud Computing**

Cloud Computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services. The services themselves have long been referred to as Software as a Service (Seas), so we use that term. The datacenter hardware and software is what we will call a Cloud. When a Cloud is made available in a pay-as-you-go manner to the public, we call it a Public Cloud; the service being sold is Utility Computing. Current examples of public Utility Computing include Amazon Web Services, Google Apennine, and Microsoft Azure. We use the term Private Cloud to refer to internal datacenters of a business or other organization that are not made available to the public.

Thus, Cloud Computing is the sum of Seas and Utility Computing, but does not normally include Private Clouds. We'll generally use Cloud Computing, replacing it with one of the other terms only when clarity demands it. The roles of the people as users or providers of these layers of Cloud Computing, and we'll use those terms to help make our arguments clear. The advantages of Seas to both end users and service providers are well understood. Service providers enjoy greatly simplified software installation and maintenance and centralized control over versioning; end users can access the service "anytime, anywhere", share data and collaborate more easily, and keep their data stored safely in the infrastructure. Cloud Computing does not change these arguments, but it does give more application providers the choice of deploying their product as Seas without provisioning a datacenter: just as the emergence of semiconductor foundries gave chip companies the opportunity to design and sell chips without owning a fib, Cloud Computing allows deploying Seas—and scaling on demand—without building or provisioning a datacenter.

Analogously to how Seas allows the user to offload some problems to the Seas provider, the Seas provider can now offload some of his problems to the Cloud Computing provider. From now on, we will focus on issues related to the potential Seas Provider (Cloud User) and to the Cloud Providers, which have received less attention.

We will argue that all three are important to the technical and economic changes made possible by Cloud Computing. Indeed, past efforts at utility computing failed, and we note that in each case one or two of these three critical characteristics were missing. For example, Intel Computing Services in 2000-2001 required negotiating a contract and longer-term use than per hour. As a successful example, Elastic Compute Cloud (EC2) from Amazon Web Services (AWS) sells 1.0-GHz x86 ISA "slices" for 10 cents per hour, and a new "slice", or instance, can be added in 2 to 5 minutes. Amazon's Scalable Storage Service (S3)

charges $0.12 to $0.15 per gigabyte-month, with additional bandwidth charges of $0.10 to $0.15
Per gigabyte to move data in to and out of AWS over the Internet. Amazon's bet is that by statistically multiplexing multiple instances onto a single physical box, that box can be simultaneously rented to many customers who will not in general interfere with each others' usage While the attraction to Cloud Computing users (Seas providers) is clear, who would become a Cloud Computing provider, and why? To begin with, realizing the economies of scale afforded by statistical multiplexing and bulk purchasing requires the construction of extremely large datacenters. Building, provisioning, and launching such a facility is a hundred-million-dollar undertaking. However, because of the phenomenal growth of Web services through the early 2000's, many large Internet companies, including Amazon, eBay, Google, Microsoft and others, were already doing so. Equally important, these companies also had to develop scalable software infrastructure (such as Map Reduce, the Google File System, Bitable, and Dynamo and the operational expertise to armor their datacenters against potential physical and electronic attacks. Therefore, a necessary but not sufficient condition for a company to become a Cloud Computing provider is that it must have existing investments not only in very large datacenters, but also in large-scale software infrastructure
And operational expertise required running them. Given these conditions, a variety of factors might influence these companies to become Cloud Computing providers:

2.Secure Network ID and Attack measure Count in Virtual Systems

A recent Cloud Security Alliance (CSA) survey shows that among all security issues, abuse and nefarious use of cloud computing is considered as the top security threat in which attackers can exploit vulnerabilities in clouds and utilize cloud system resources to deploy attacks. In traditional data centers, where system administrators have full control over the host machines, Vulnerabilities can be detected and patched by the system administrator in a centralized manner. However, patching known security holes in cloud data centers, where cloud users usually have the privilege to control software installed on their managed VMs, may not work effectively and can violate the Service Level Agreement (SLA). Furthermore, cloud users can install vulnerable software on their VMs, which essentially contributes to loopholes in cloud security. The challenge is to establish an effective vulnerability/attack detection and response system for accurately identifying attacks and minimizing the impact of security breach to cloud users.

In M. Armrest et al. addressed that protecting" Business continuity and services availability" from service outages is one of the top concerns in cloud computing systems. In a cloud system where the infrastructure is shared by potentially millions of users, abuse and nefarious use of the shared infrastructure benefits attackers to exploit vulnerabilities of the cloud and use its resource to deploy attacks in more efficient ways. Such attacks are more effective in the cloud environment since cloud users usually share computing resources, e.g., being connected through the same switch, sharing with the same data storage and file systems, even with potential attackers. The similar setup for VMs in the cloud, e.g., virtualization techniques.

In this paper, we propose Secure Intrusion Detection and Attack measure exquisite in Virtual Systems to establish a defense-in-depth intrusion detection framework. For better attack detection, NICE incorporates attack graph analytical procedures into the intrusion detection processes. We must note that the design of NICE does not intend to improve any of the existing intrusion detection algorithms; indeed, NICE employs a reconfigurable virtual networking approach to detect and counter the attempts to compromise VMs, thus preventing zombie VMs.

In general, NICE includes two main phases: deploy a lightweight mirroring-based network intrusion detection agent (NICE-A) on each cloud server to capture and analyze cloud traffic. A NICE-A periodically scans the virtual system vulnerabilities within a cloud server to establish Scenario Attack Graph (SAGs), and then based on the severity of identified vulnerability towards the collaborative attack goals, NICE will decide whether or not to put a VM in network inspection state. Once a VM enters inspection state, Deep Packet Inspection (DPI) is applied, and/or virtual network reconfigurations can be

deployed to the inspecting VM to make the potential attack behaviors prominent.

The rest of paper is organized as follows. Section II presents the related work. Section III describes system approach and implementation. System models are described in Section IV describes the approach to hardening the network in NICE. The proposed NICE is presented in Section V and Section VI evaluates NICE in terms of network performance and security. Finally, Section VII describes future work and concludes this paper.

3. BotSniffer: Detecting Bitnet Command and Control Channels In Network Traffic

Bonnets (or, networks of zombies) are recognized as one of the most serious security threats today. Bonnets are different from other forms of malware such as worms in that they use command and control (C&C) channels. It is important to study this bonnet characteristic so as to develop effective countermeasures. First, a bonnet C&C channel is relatively stable and unlikely to change among bots and their variants. Second, it is the essential mechanism that allows a "postmaster" (who controls the Bitnet) to direct the actions of bots in a Bitnet. As such, the C&C channel can be considered the weakest link of a boned. That is, if we can take down an active C&C or simply interrupt the communication to the C&C, the postmaster will not be able to control his bonnet. Moreover, the detection of the C&C channel will reveal both the C&C servers and the bots in a monitored network. Therefore, understanding and detecting the C&Cs has great value in the battle against bonnets.

Many existing Bitnet C&Cs are based on IRC (Internet Relay Chat) protocol, which provides a centralized command and control mechanism. The postmaster can interact with the bots (e.g., issuing commands and receiving responses) in real-time by using IRC PRIVMSG messages.

This simple IRC-based C&C mechanism has proven to be highly successful and has been adopted by many bonnets'. There are also a few bonnets' that use the HTTP protocol for C&C. HTTP-based C&C is still centralized, but the postmaster does not directly interact with the bots using chalice mechanisms. Instead, the bots periodically contact the C&C server(s) to obtain their commands. Because of its proven effectiveness and efficiency, we expect that centralized C&C (e.g., using IRC or HTTP) will still be widely used by burnets in the near future. In this paper, we study the problem of detecting centralized boned C&C channels using network anomaly detection techniques. In particular, we focus on the two commonly used Bitnet C&C mechanisms, namely, IRC and HTTP based C&C channels. Our goal is to develop a detection approach that does not require prior knowledge of a bonnet, e.g., signatures of C&C patterns including the name or IP address of a C&C server. We leave the problem of detection of P2P bonnets' and Peacomm as our future work.

Few bots in the monitored network, and may contain encrypted communication. However, we observe that the bots of a burnet demonstrate spatial-temporal correlation and similarities due to the nature of their pre-programmed response activities to control commands. This invariant helps us identify C&C within network traffic. For instance, at a similar time, the bots within a bonnet will execute the same command (e.g., obtain system information, scan the network), and report to the C&C server with the progress/result of the task (and these reports are likely to be similar in structure and content). Normal network activities are unlikely to demonstrate such a *synchronized* or *correlated* behavior. Using a sequential hypothesis testing algorithm, when we observe multiple instances of correlated and similar behaviors, we can conclude that a boned is detected. Our research makes several contributions.

First, we study two typical styles of control used in centralized bonnet C&C. The first is the "push" style, where commands are pushed or sent to bots. IRC-based C&C is an example of the push style. The second is the "pull" style, where commands are pulled or downloaded by bots. HTTP-based C&C is an example of the pull style. Observing the spatial temporal correlation and similarity nature of these bonnet C&Cs, we provide a set of heuristics that distinguish C&C traffic from normal traffic.

Second, we propose anomaly-based detection algorithms to identify both IRC and HTTP based C&Cs in a port independent manner. The

advantages of our algorithms include: they do not require prior knowledge of C&C servers or content signatures, they are able to detect encrypted C&C, they do not require a large number of bots to be present in the monitored network, and may even be able to detect a Bitnet with just a single member in the
monitored network in some cases, they have bounded false positive and false negative rates, and do not require a large number of C&C communication packets.

4.NUSMV: a new symbolic model checker

This paper describes the results of a joint project between Carnegie Mellon University (CMU) and Institute per la Ricers Sciatica e Technologic (IRST) whose goal is the development of a new symbolic model checker.1 the new model checker, called NUSMV, is designed to be a well structured, open, _exiles and documented platform for model checking. To be usable in technology transfer projects, NUSMV was designed to be very robust, easy to modify, and close to the standards required by industry. NUSMV is the result of the reengineering and reimplementation of the CMU SMV symbolic model checker. With respect to CMU SMV, NUSMV has been upgraded along three dimensions.

From the point of view of the system functionalities, NUSMV has some features (e.g., multiple interfaces, LTL speci_cations) that enhance the user ability to interact with the system, and provide more heuristics for, e.g., achieving efficiency or partially controlling the state explosion. The system architecture of NUSMV is highly modular (thus allowing for the substitution or elimination of certain modules) and open (thus allowing for the addition of new modules). A further feature is that in NUSMV the user can control, and possibly change, the order of execution of some system modules.

The quality of the implementation is much enhanced. NUSMV is a very robust and well documented system, whose code is (relatively) easy to modify. The paper is organized as follows we briery introduce the logical framework below symbolic model checking; describes the interaction with the system explains the functionalities provided by the system describes the NUSMV system architecture; describes the NUSMV implementation features. Finally, describes the results of some tests and future development directions

The most widely used frication techniques are testing and simulation. In the case of complex, asynchronous systems, however, these techniques can cover only a limited portion of possible behaviors. A complementary frication technique is Temporal Logic Model Checking. In this approach, the varied system is modeled as a niter state transition system, and the speci_cations are expressed in a propositional temporal logic. Then, by exhaustively
Exploring the state space of the state transition system, it is possible to check automatically if the speci_cations are stashed. The termination of model checking is guaranteed by the niceness of the model. One of the most important features of model checking is that, when a speci_cations is found not to hold, a counterexample (i.e., a witness of the offending behavior of the system) is produced.

The current NUSMV input language is essentially the same as the CMU SMV input language The NUSMV input language is designed to allow for the description of _niter state systems. The only data types provided by the language are Booleans, bounded integer sub ranges, and symbolic enumerated types. Moreover, NUSMV allows for the dentitions' of bounded arrays of basic data types. The description of a complex system can be decomposed into modules, and each of them can be instantiated many times.

This provides the user with a modular and hierarchical description, and supports the dentition of reusable components. Each module dense a _niter state machine. Modules can be composed either synchronously or asynchronously using interleaving. In synchronous composition a single step in the composition corresponds to a single step in each of the components. In asynchronous composition with interleaving a single step of the composition corresponds to a single step performed by exactly one component. The NUSMV input language allows describing Deterministic and non deterministic systems. A NUSMV program can describe both the model and the speci_cations.

Gives a small example of a NUSMV program. The example in Figure 4 is a model of a 3 bit

binary counter circuit. It illustrates the damnation of Reusable modules and expressions. The module counter cell is instantiated three times, with names bit0, bit1 and bit2. The module counter cell has a formal parameter carry in. In the instantiation of the module, actual signals (1 for the instance bit0, bit0.carry out for the instance bit1 and bit1.carry out for the instance bit2) are plugged in for the formal parameters, thus linking the module instance to the program (a module can be seen as a subroutine). The property that we want to check is .invariantly eventually the counter count till 8. Which is expressed in CTL using the design state variables as .

It is also possible to specify the transition relation and the set of initial states of a module by means of propositional formulas, using the keywords TRANS, and INIT respectively. This provides the user with a lot of freedom in designing systems
there is an equivalent damnation of module counter cell using propositional formulas.

there is an equivalent damnation of module counter cell using propositional formulas

5.Scalable, Graph-Based Network Vulnerability Analysis

Researchers and penetration testers often organize these chains of exploits into graphs or trees. In either case, a designated node (or set of nodes) represents the initial state(s), where a state is defined by assigning a set of values to relevant system attributes, including specific vulnerabilities on various hosts in the network, connectivity between hosts, and attacker access privileges on various hosts. Each transition in the tree (or graph) represents a specific exploit that an attacker can carry out. For example, the 'sshd buffer overflow' exploit, carried out from a specific host controlled by the attacker towards a victim host, lets the attacker obtain root access privileges on the victim, thereby changing the state of the system. Although the precise definitions of attack graph and attack tree vary by author, it is useful to think of an attack tree as a structure in which each possible exploit chain ends in a leaf state that satisfies the attacker's goal, and an attack graph as a consolidation of the attack tree in which some or all common states are merged.

The basic observation behind this paper is that attack graphs can easily be far too large to be practical. Paper provides some support for our position on this: in a scaling exercise with 5 hosts, 8 exploits, and the vulnerabilities associated with those exploits, Mums reportedly Took 2 hours to execute, with most of that time spent on graph manipulation. The resulting attack graph had 5948 nodes and 68364 edges. The state space in that example was represented with 229 bits. By contrast, to encode such a problem with the methods Presented in this paper, we need, at most, 229 nodes, one for each bit in the state representation. Each of these nodes must be able to store a constant amount of information about however many exploits can change the value of that particular node from 'false' to 'true'. It is clear that our structure is dramatically smaller, even for this relatively limited, from a real world perspective, example.

However, we don't lose any of the information in our encoding. That is, we present an algorithm that explodes our structure into an attack tree. At the same time, we aren't required to generate an attack tree (or graph) to carry out our subsequent analysis. We give worst case bounds on our algorithms as we give our presentation.

The cost we pay in this paper for our dramatically smaller data structure is monotonicity. Simply stated, monotonicity means that no action an attacker takes interferes with the attacker's ability to take any other action. We return to the issue of monotonicity throughout the paper. Our basic position is that monotonicity is a reasonable modeling assumption in many network analysis situations.

Commercial vulnerability scanners are quite effective at what they do - namely identifying vulnerabilities in a specific host. However, a variety of authors have noted that identifying vulnerabilities in isolation is only a small part of securing a network, and that a significant issue is identifying which vulnerabilities an attacker can take advantage of through a chain of exploits. For example, an attacker might exploit a defect in a particular version of ftp to overwrite the .roosts file on a victim machine. In the next step, the attacker could remotely log in to the victim. In a subsequent step, the attacker could use the victim machine as a base to launch another exploit on a

new victim, and so on. There are numerous examples of such chains in the literature, and extensive databases of exploits tailored to specific software and services are available on the web.

We treat vulnerabilities, attacker access privileges, and network connectivity in a way similar to other authors but with some simplification. A vulnerability is a fact about the system that, on the one hand, potentially enables some exploit to be carried out, and on the other, is the

Result of some exploit. Vulnerability might be running a particular version of some operating system on a given host. Attacker privileges and network connectivity are both straightforward to model; the fact that an attacker has a certain privilege level on a given host is an atomic fact, as is whether two hosts have a type of connectivity required for a given exploit.

The simplification is that we group together attacker access privileges, network connectivity, and vulnerabilities into generic *attributes* in our model. Thus, if an attacker has ftp access to a given host, we model this as an atomic attribute. Similarly, if a '.roosts' file includes a given host, we record that as an atomic attribute as well.

The state space in that example was represented with 229 bits. By contrast, to encode such a problem with the methods Presented in this paper, we need, at most, 229 nodes, one for each bit in the state representation.

CONCLUSION

The event generating system has to be improved. The concepts are interesting, but additional work is needed to optimize the process. The system is still early work. It is possible to automatically detect anomalies in the communication structure of a surveyed network, but the goal of detecting a large number of different attacks is not yet reached. The fact that some of the attacks could be discovered by the system without any knowledge on the used attack techniques encourages us to further research. It is easy to see that this approach to intrusion detection only is appropriate for intruders causing an significant traffic in the supervised network. The proposed framework leverages Open Flow network programming APIs to build a monitor and control plane over distributed programmable virtual switches to significantly improve attack detection and mitigate attack consequences. The system and security evaluations demonstrate the efficiency and effectiveness of the proposed solution.

REFERENCES

[1] "A View of Cloud Computing," M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A.Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M.Zaharia, ACM Comm., vol. 53,no. 4, pp. 50-58, Apr. 2010.

[2] "Measuring Network Security Using Bayesian Network-Based Attack Graphs," M.Frigault and L. Wang, Proc. IEEE 32nd Ann. Int'l Conf. Computer Software and Applications (COMPSAC '08),pp. 698-703, Aug. 2008.

[3] "Security and Privacy Challenges in Cloud Computing Environments," H. Takabi, J.B. Joshi, and G. Ahn, IEEE Security and Privacy, vol. 8, no. 6, pp. 24-31, Dec. 2010.

[4] "BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic,"G. Gu, J. Zhang, and W. Lee, Proc. 15th Ann. Network and Distributed Sytem Security Symp. (NDSS '08), Feb.2008.

[5] "NuSMV: A New Symbolic Model Checker," http://afrodite .itc.it:1024/nusmv. Aug. 2012.

[6] "Scalable, graphbasednetwork vulnerability analysis," P. Ammann, D. Wijesekera, and S. Kaushik, Proc. 9th ACM Conf.Computer and Comm. Security (CCS '02), pp. 217-224, 2002

[7] "Dynamic Security Risk Management Using Bayesian Attack Graphs,"N. Poolsappasit, R. Dewri, and I. Ray,IEEE Trans.Dependable and Secure Computing, vol. 9, no. 1, pp. 61-74, Feb. 2012.

[8] "A New Alert Correlation Algorithm Based on Attack Graph," S. Roschke, F. Cheng, and C. Meinel, Proc. Fourth Int'l Conf.Computational Intelligence in Security for Information Systems,pp. 58-67, 2011

[9] "Network Security Management Using ARP Spoofing,"K. Kwon, S. Ahn, and J. Chung, Proc. Int'l Conf. Computational Science and Its Applications (ICCSA '04), pp. 142-149, 2004.

[10] "Automated Generation and Analysis of Attack Graphs,"O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J.M. Wing,Proc. IEEE Symp. Security and Privacy, pp. 273-284, 2002,