Alfonso Valdes
Diego Zamboni (Eds.)

LNCS 3858

Recent Advances in Intrusion Detection

8th International Symposium, RAID 2005
Seattle, WA, USA, September 2005
Revised Papers

Springer

TP309-53
R149
2005

Alfonso Valdes   Diego Zamboni (Eds.)

Recent Advances in Intrusion Detection

8th International Symposium, RAID 2005
Seattle, WA, USA, September 7-9, 2005
Revised Papers

Springer

Volume Editors

Alfonso Valdes
SRI International
333 Ravenswood Ave., Menlo Park, CA 94025, USA
E-mail: alfonso.valdes@sri.com

Diego Zamboni
IBM Research GmbH
Zurich Research Laboratory
Säumerstr. 4, Postfach, 8803 Rüschlikon, Switzerland
E-mail: dza@zurich.ibm.com

Library of Congress Control Number: 2005939042

CR Subject Classification (1998): K.6.5, K.4, E.3, C.2, D.4.6

LNCS Sublibrary: SL 4 – Security and Cryptology

ISSN 0302-9743
ISBN-10 3-540-31778-3 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-31778-4 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11663812 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science 3858

¥444.00元

Preface

On behalf of the Program Committee, it is our pleasure to present the proceedings of the 8th Symposium on Recent Advances in Intrusion Detection (RAID 2005), which took place in Seattle, Washington, USA, September 7-9, 2005.

The symposium brought together leading researchers and practitioners from academia, government and industry to discuss intrusion detection from research as well as commercial prospectives. We also encouraged discussions that addressed issues that arise when studying intrusion detection, including monitoring, performance and validation, from a wider perspective. We had sessions on the detection and containment of Internet worm attacks, anomaly detection, automated response to intrusions, host-based intrusion detection using system calls, network intrusion detection, and intrusion detection, in mobile wireless networks.

The RAID 2005 Program Committee received 83 paper submissions from all over the world. All submissions were carefully reviewed by several members of Program Committee and selection was made on the basis of scientific novelty, importance to the field, and technical quality. Final selection took place at a Program Committee meeting held on May 11 and 12 in Oakland, California. Fifteen papers and two practical experience reports were selected for presentation and publication in the conference proceedings. The keynote address was given by Phil Attfield of the Northwest Security Institute.

A successful symposium is the result of the joint effort of many people. In particular, we would like to thank all authors who submitted papers, whether accepted or not. Out thanks also go to the Program Committee members and additional reviewers for their hard work with the large number of submissions. In addition, we want to thank the General Chair, Ming-Yuh Huang, for handling conference arrangements and finding support from our sponsors. Finally, we extend our thanks to the sponsors: Pacific Northwest National Laboratory, The Boeing Company, the University of Idaho, and Conjungi Security Technologies.

September 2005

Al Valdes
Diego Zamboni

Organization

RAID 2005 was organized by the Boeing Company, Seattle, WA, USA.

Conference Chairs

General Chair	Ming-Yuh Huang (The Boeing Company)
Program Chair	Alfonso Valdes (SRI International)
Program Co-chair	Diego Zamboni (IBM Zurich Research Laboratory)
Publication Chair	Jeff Rowe (UC Davis)
Publicity Chair	Deborah Frincke (Pacific Northwest National Lab)
Sponsorship Chair	Jim Alves-Foss (University of Idaho)

Program Committee

Magnus Almgren	Chalmers, Sweden
Tatsuya Baba	NTT Data, Japan
Sungdeok (Steve) Cha	Korea Advanced Institute of Science and Technology, Korea
Steven Cheung	SRI International, USA
Robert Cunningham	MIT Lincoln Laboratory, USA
Fengmin Gong	McAfee Inc., USA
Farman Jahanian	University of Michigan, USA
Somesh Jha	University of Wisconsin, USA
Klaus Julisch	IBM Research, Switzerland
Chris Kruegel	UCSB, USA
Roy Maxion	Carnegie Mellon University, USA
Ludovic Mé	Supélec, France
George Mohay	Queensland University of Technology, Australia
Peng Ning	North Carolina State University, Raleigh, USA
Vern Paxson	ICSI and LBNL, USA
Jeff Rowe	University of California, Davis, USA
Bill Sanders	University of Illinois, Urbana-Champaign, USA
Dawn Song	Carnegie Mellon University, USA
Sal Stolfo	Columbia University, USA
Kymie Tan	Carnegie Mellon University, USA
Giovanni Vigna	UCSB, USA
Alec Yasinsac	Florida State University, USA
Diego Zamboni	IBM Research, Switzerland

Steering Committee

Marc Dacier (Chair)	Institut Eurecom, France
Hervé Debar	France Telecom R&D, France
Deborah Frincke	Pacific Northwest National Lab, USA
Ming-Yuh Huang	The Boeing Company, USA
Erland Jonsson	Chalmers, Sweden
Wenke Lee	Georgia Institute of Technology, USA
Ludovic Mé	Supélec, France
S. Felix Wu	UC Davis, USA
Andreas Wespi	IBM Research, Switzerland
Alfonso Valdes	SRI International, USA
Giovanni Vigna	UCSB, USA

Pacific Northwest Local Organizing Committee

Philip Attfield	Northwest Security Institute
Kirk Bailey	City of Seattle
Barbara Endicott-Popovsky	Seattle University
Deborah Frincke	Pacific Northwest National Lab
Ming-Yuh Huang	The Boeing Company
Rita Rutten	Conference Coordinator
Michael A. Simon	Conjungi Networks

Table of Contents

Virtual Playgrounds for Worm Behavior Investigation

Xuxian Jiang[1], Dongyan Xu[1], Helen J. Wang[2], and Eugene H. Spafford[1]

[1] CERIAS and Department of Computer Science,
Purdue University, West Lafayette, IN 47907
{jiangx, dxu, spaf}@cs.purdue.edu
[2] Microsoft Research Redmond, WA 98052
helenw@microsoft.com

Abstract. To detect and defend against Internet worms, researchers have long hoped to have a safe convenient environment to unleash and run real-world worms for close observation of their infection, damage, and propagation. However, major challenges exist in realizing such "worm playgrounds", including the playgrounds' *fidelity, confinement, scalability*, as well as *convenience* in worm experiments. In this paper, we present a *virtualization-based* platform to create virtual worm playgrounds, called *vGrounds*, on top of a physical infrastructure. A vGround is an all-software virtual environment dynamically created for a worm attack. It has realistic end-hosts and network entities, all realized as virtual machines (VMs) and confined in a virtual network (VN). The salient features of vGround include: (1) *high fidelity* supporting real worm codes exploiting real vulnerable services, (2) *strict confinement* making the real Internet totally invisible and unreachable from inside a vGround, (3) *high resource efficiency* achieving sufficiently large scale of worm experiments, and (4) *flexible and efficient worm experiment control* enabling fast (tens of seconds) and automatic generation, re-installation, and final tear-down of vGrounds. Our experiments with real-world worms (including *multi-vector worms and polymorphic worms*) have successfully exhibited their probing and propagation patterns, exploitation steps, and malicious payloads, demonstrating the value of vGrounds for worm detection and defense research.

Keywords: Internet Worms, Intrusion Observation and Analysis, Destructive Experiments.

1 Introduction

In recent worm detection and defense research, we have witnessed increasingly novel features of emerging worms [41] in their infection and propagation strategies. Examples are polymorphic appearance [34], multi-vector infection [15], self-destruction [23], and intelligent payloads such as self-organized attack networks [18] or mass-mailing capability [21]. In order to understand key aspects of worm behavior such as probing, exploitation, propagation, and malicious payloads, researchers have long hoped to have a safe and convenient environment to run and observe real-world worms. Such a "worm playground" environment is useful not only in accessing the impact of worm intrusion and propagation, but also in testing worm detection and defense mechanisms [46, 42, 35, 37].

Despite its usefulness, there are difficulties in realizing a worm playground. Major challenges include the playground's *fidelity, confinement, scalability, resource efficiency*, as well as the *convenience in worm experiment setup and control*. Currently, a common practice is to deploy a dedicated testbed with a large number of physical machines, and to use these machines as nodes in the worm playground. However, this approach may not effectively address the above challenges, for the following reasons: (1) Due to the coarse granularity (one physical host) of playground entities, the scale of a worm playground is constrained by the number of physical hosts, affecting the full exhibition of worm propagation behavior; (2) By nature, worm experiments are *destructive*. With physical hosts as playground nodes, it is a time-consuming and error-prone manual task for worm researchers to re-install, re-configure, and reboot worm-infected hosts between experiment runs; and (3) Using physical hosts for worm tests may lead to security risk and impact leakage, because the hosts may connect to machines *outside* the playground. However, if we make the testbed a physically-disconnected "island", the testbed will no longer be share-able to remote researchers.

The contribution of our work is the design, implementation, and evaluation of a *virtualization-based* platform to quickly create safe virtual worm playgrounds called *vGrounds*, on top of general-purpose infrastructures. Our vGround platform can be readily used to analyze Linux worms, which represent a non-negligible source of insecurity especially with the rise of popularity of Linux in servers' market. Though the current prototype does not support Windows-based worms, our design principles and concepts can also be applied to build Windows-based vGrounds.

The vGround platform can conveniently turn a physical infrastructure into a base to host vGrounds. An infrastructure can be a single physical machine, a local cluster, or a multi-domain overlay infrastructure such as PlanetLab [7]. A vGround is an all-software virtual environment with realistic end-hosts and network entities, all realized as virtual machines (VMs). Furthermore, a virtual network (VN) connects these VMs and *confines* worm traffic within the vGround. The salient features of vGround include:

- *High fidelity.* By running real-world OS, application, and networking software, a vGround allows *real* worm code to propagate as in the real Internet. Our full-system virtualization approach achieves the fidelity that leads to more opportunities to capture nuances, tricks, and variations of worms, compared with simulation-based approaches [39]. For example, one of our vGround-based experiments *identified a misstatement in a well-known worm bulletin*[1].
- *Strict confinement.* Under our VM and VN (virtual network) technologies, the real Internet is totally invisible (unaddressable) from inside a vGround, preventing the leakage of negative impact caused by worm infection, propagation, and malicious payloads [16, 23] into the underlying infrastructure and cascadingly, the rest of the Internet. Furthermore, the damages caused by a worm only affect the virtual entities and components in one vGround and therefore do *not* affect other vGrounds running on the same infrastructure.
- *Flexible and efficient worm experiment control.* Due to the all-software nature of vGrounds, the instantiation, re-installation, and final tear-down of a vGround are

[1] The misstatement is now fixed and the authors have agreed not to disclose the details.

both fast and automatic, saving worm researchers both time and labor. For example, in our Lion worm experiment, it only takes 60, 90, and 10 seconds, respectively, to generate, bootstrap, and tear-down the vGround with 2000 virtual nodes. Such efficiency is essential when performing *multiple runs of a destructive experiment*. These operations can take hours or even days if the same experiment is performed directly on physical hosts. More importantly, the operations can be started by the researchers *without* the administrator privilege of the underlying infrastructure.

– *High resource efficiency.* Because of the scalability of our virtualization techniques, the scale of a vGround can be magnitudes larger than the number of physical machines in the infrastructure. In our current implementation, one physical host can support several *hundred* VMs. For example, we have tested the propagation of Lion worms [16] in a vGround with 2000 virtual end hosts, based on 10 physical nodes in a Linux cluster.

  However, we would like to point out that although such scalability is effective in exposing worm propagation strategies based on our limited physical resources (Section 4), it is *not* comparable to the scale achieved by worm simulations. Having different focuses and experiment purposes, vGround is more suitable for analyzing detailed worm actions and damages, while the simulation-based approach is better for modeling worm propagation under Internet scale and topology. Also, lacking realistic background computation and traffic load, current vGrounds are *not appropriate for accurate quantitative modeling of worms*.

We are not aware of similar worm playground platforms with all the above features that are widely deployable on general-purpose infrastructures. We have successfully run real worms, including multi-vector worms and polymorphic worms, in vGrounds on our *desktops, local clusters*, and *PlanetLab*. Our experiments are able to fully exhibit the worms' probing and propagation patterns, exploitation attempts, and malicious payloads, demonstrating the value of vGrounds in worm detection and defense research.

The rest of this paper is organized as follows: Section 2 provides an overview of the vGround approach. The detailed design is presented in Section 3. Section 4 demonstrates the effectiveness of vGround using our experiments with several real-world worms. A discussion on its limitations and extensions is presented in Section 5. Related works are discussed in Section 6. Finally, Section 7 concludes this paper.

2 The vGround Approach

A vGround is a virtualization-based self-confined worm playground where not only each entity, including an end host, a firewall, a router, and even a network cable, is fully virtualized, but also every communication traffic is strictly confined within. Due to its virtualization-based nature and associated self-confinement property, a vGround can be safely created on a wide range of general-purpose infrastructures, including regular desktops, local clusters, and even wide-area shared infrastructures such as PlanetLab. For example, Figure 1 shows a simple vGround (the vGrounds in our worm experiments are much larger in scale) which is created on top of three PlanetLab hosts A, B, and C.

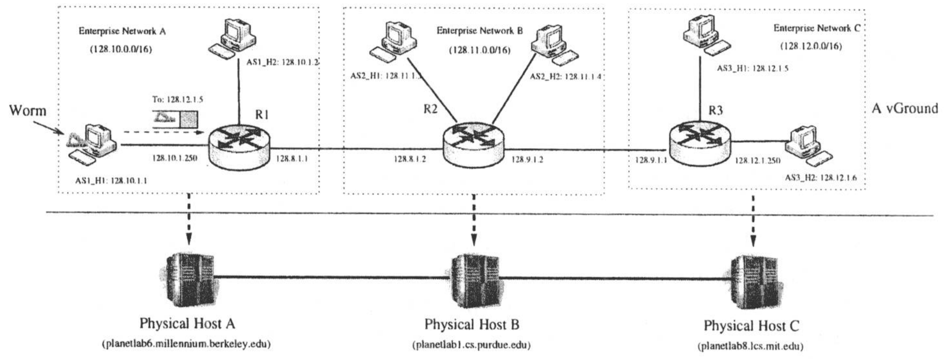


Fig. 1. A PlanetLab-based vGround for worm experiment

The vGround includes three virtual enterprise networks connected by three virtual routers ($R1$, $R2$, and $R3$). Within the vGround, the "seed" worm node ($AS1\_H1$ in network A 128.10.0.0/16) is starting to infect other nodes running vulnerable services. Note that a vGround essentially appears as a virtual Internet whose network address assignment can be totally orthogonal to that of the real Internet. Furthermore, multiple simultaneously running vGrounds can safely overlap their address space without affecting each other as one vGround is completely invisible to another vGround.

Using a vGround specification language, a worm researcher will be able to specify the worm experiment setup in a vGround, including software systems and services, IP addresses, and routing information of virtual nodes (i.e. virtual end hosts and routers). Given the specification, the vGround platform will perform *automatic* vGround instantiation, bootstrapping, and clean-up. In a typical worm experiment, multiple runs are often needed as each different run is configured with a different parameter setting (e.g., different worm signatures [8, 1] and different traffic throttling thresholds[46]). However, because of the worm's destructive behavior, the vGround will be completely unusable after each run and need to be re-installed. *The vGround platform is especially efficient in supporting such an iterative worm experiment workflow.*

2.1 Key vGround Techniques

Existing full-system virtualization is adopted to achieve high *fidelity* of vGrounds. Worms infect machines by remotely exploiting certain vulnerabilities in OS or application services (e.g., BIND, Sendmail, DNS). Therefore, the vulnerabilities provided by vGrounds should be the same as those in real software systems. As such, vGround can not only be leveraged for experimenting worms propagating via known vulnerabilities, but also be useful for discovering worms exploiting *unknown vulnerabilities*, of which worm simulations are *not* capable.

There exist various VM technologies that enable full-system virtualization. Examples include Virtual PC [12], VMware [13], Denali [49], Xen [26], and User-Mode Linux (UML) [30]. The differences in their implementations lead to different levels of cost, deployability and configurability: VMware and similarly Virtual PC require several loadable kernel modules for virtualizing underlying physical resources; Xen and Denali "paravirtualize" physical resources by running *in place of* host OS; and UML is mainly a *user-level* implementation through system call virtualization. We choose UML in the current vGround implementation so that the deployment of vGround does not require the root privilege of the shared infrastructure. As a result, current vGround prototype can be widely deployed in most Linux-based systems (including PlanetLab). However, we would like to point out that the original UML itself is *not* able to satisfy the vGround needs. As described next, *we have developed new extensions to UML.*

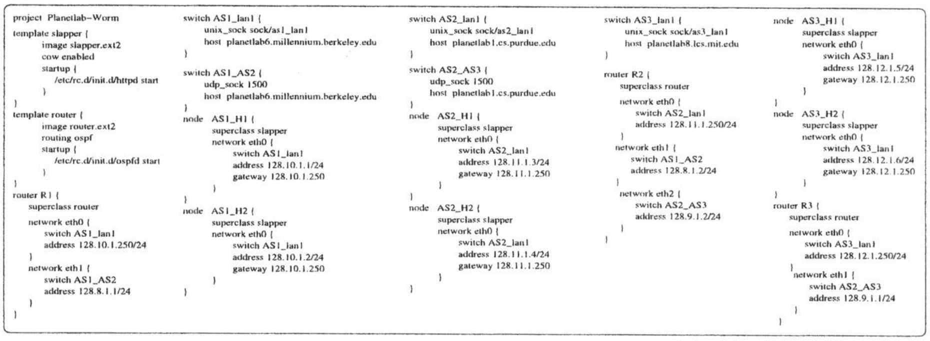
New network virtualization techniques are developed to achieve vGround *confinement.* Simply running a worm experiment in a number of VMs *will not* confine the worm traffic just within these VMs and thus prevent potential worm "leakage". Although existing UML implementation does have some support for virtual networking, it is still not capable of organizing different VMs into an *isolated* virtual topology. In particular, when the underlying shared infrastructure spans multiple physical domains, additional VPN softwares are needed to create the illusion of the virtual Internet. However, there are two notable weaknesses: (1) a VPN does not hide the existence of the underlying physical hosts and their network connections, which fails to meet the strict confinement requirement; (2) a VPN usually needs to be statically/manually configured as it requires the root privilege to manipulate the routing table, which fails to meet the flexible experiment control requirement. As our solution, we have developed a link-layer network virtualization technique to create a VN for VMs in a vGround. The VN reliably intercepts the traffic at the link-layer and is thus able to constrain both the topology and volume of traffic generated by the VMs. Such a VN essentially enables the illusion as a "virtual Internet" (though with a smaller scale) with its own IP address space and router infrastructure. More importantly, the VN and the real Internet are, by nature of our VN implementation, *mutually un-addressable.*

New optimization techniques are developed to improve vGround *scalability, efficiency, and flexibility.* To increase the number of VMs that can be supported in one physical host, the resource consumption of each individual VM should be conserved. For example, a full-system image of Red-Hat $9.0/7.2$ requires approximately $1G/700M$ disk space. For a vGround of 100 VMs, a naive approach would require at least $100G/70G$ disk space. Our optimization techniques exploit the fact that a large portion of the VM images is the *same* and can be shared among the VMs. Furthermore, some services, libraries, and software packages in the VM image are *not* relevant to the worm being tested, and could therefore be safely removed. We also develop a *new method* to safely and efficiently generate VM images in each physical host (Section 3.4). Finally, a *new technique is being developed to enable worm-driven vGround growth*: new virtual nodes/subnets can be added to the vGround at runtime in reaction to a worm's infection intent.

2.2 Advanced vGround User Configurability

The vGround platform provides a vGround specification language to worm researchers. There are two major types of entities - *network* and *virtual node*, in the vGround specification language. A *network* is the medium of communication among *virtual nodes*. A virtual node can be an end-host, a router, a firewall, or an IDS system and it has one or more network interface cards (NICs) - each with an IP addresses. In addition, the virtual nodes are properly connected using proper routing mechanisms. Currently, the vGround platform supports RIP, OSPF, and BGP protocols.

In order to conveniently specify and efficiently generate various system images, the language defines the following notions: (1) A *system template* contains the basic VM system image which is *common* among multiple virtual nodes. If a virtual node is derived from a system template, the node will inherit all the capabilities specified in the system template. The definition of system template is motivated by the observation that most end-hosts to be victimized by a certain worm look quite similar from the worm's perspective. (2) A *cluster* of nodes is the group of nodes located in the same subnet. The user may specify that they inherit from the same system template, with their IP addresses sharing the same subnet prefix.



```
project Planetlab-Worm
template slapper {
        image slapper.ext2
        cow enabled
        startup {
           /etc/rc.d/init.d/httpd start
        }
}
template router {
        image router.ext2
        routing ospf
        startup {
           /etc/rc.d/init.d/ospfd start
        }
}
router R1 {
    superclass router
    network eth0 {
        switch AS1_lan1
        address 128.10.1.250/24
    }
    network eth1 {
        switch AS1_AS2
        address 128.8.1.1/24
    }
}

switch AS1_lan1 {
      unix_sock sock/as1_lan1
      host planetlab6.millennium.berkeley.edu
}

switch AS1_AS2 {
      udp_sock 1500
      host planetlab6.millennium.berkeley.edu
}
node  AS1_H1 {
        superclass slapper
        network eth0 {
              switch AS1_lan1
              address 128.10.1.1/24
              gateway 128.10.1.250
        }
}
node  AS1_H2 {
        superclass slapper
        network eth0 {
              switch AS1_lan1
              address 128.10.1.2/24
              gateway 128.10.1.250
        }
}

switch AS2_lan1 {
      unix_sock sock/as2_lan1
      host planetlab1.cs.purdue.edu
}

switch AS2_AS3 {
      udp_sock 1500
      host planetlab1.cs.purdue.edu
}
node  AS2_H1 {
        superclass slapper
        network eth0 {
              switch AS2_lan1
              address 128.11.1.3/24
              gateway 128.11.1.250
        }
}
node  AS2_H2 {
        superclass slapper
        network eth0 {
              switch AS2_lan1
              address 128.11.1.4/24
              gateway 128.11.1.250
        }
}

switch AS3_lan1 {
      unix_sock sock/as3_lan1
      host planetlab8.lcs.mit.edu
}

router R2 {
    superclass router
    network eth0 {
        switch AS2_lan1
        address 128.11.1.250/24
    }
   network eth1 {
      switch AS1_AS2
      address 128.8.1.2/24
   }
    network eth2 {
        switch AS2_AS3
        address 128.9.1.2/24
    }
}

node  AS3_H1 {
        superclass slapper
        network eth0 {
              switch AS3_lan1
              address 128.12.1.5/24
              gateway 128.12.1.250
        }
}
node  AS3_H2 {
        superclass slapper
        network eth0 {
              switch AS3_lan1
              address 128.12.1.6/24
              gateway 128.12.1.250
        }
}
router R3 {
    superclass router
    network eth0 {
        switch AS3_lan1
        address 128.12.1.250/24
    }
     network eth1 {
        switch AS2_AS3
        address 128.9.1.1/24
     }
}
```

Fig. 2. A sample vGround specification

As an example, Figure 2 shows the specification for the vGround in Figure 1. The keyword *template* indicates the system template used to generate other images files. For example, the image *slapper.ext2* is used to generate the images of the following end-hosts: $AS1\_H1$, $AS1\_H2$, $AS2\_H1$, $AS2\_H2$, $AS3\_H1$, and $AS3\_H2$; while the image *router.ext2* is used to generate the images of routers $R1$, $R2$, and $R3$. The keyword *switch* indicates the creation of a *network* connecting various virtual nodes. The internal keywords $unix\_sock$ and udp_sock indicate different network virtualization techniques based on UNIX and INET-4 sockets, respectively. Note that the keyword *cluster* is not used in this example. However, for a large-scale vGround, it is more convenient to use *cluster* to specify a subnet, which has a large number of end-hosts of similar configuration.

After a vGround is created, the vGround platform also provides a collection of toolkits to unleash the worm, collect worm infection traces, monitor worm propagation status, and re-install or tear-down the vGround. More details will be described in Sections 3 and 4.