Technology Independent Honeynet Description Language

Wenjun Fan, David Fernández and Víctor A. Villagrá
Departamento de Ingeniería de Sistemas Telemáticos, Universidad Politécnica de Madrid,
ETSI Telecomunicación, Avda. Complutense 30, 28040, Madrid, Spain
{efan, david, villagra}@dit.upm.es

Keywords: Honeynet Description Language, Honeynet Configuration, Honeynet Management, Network Security.

Abstract: Several languages have been proposed for the task of describing networks of systems, either to help on managing, simulate or deploy testbeds for testing purposes. However, there is no one specifically designed to describe the honeynets, covering the specific characteristics in terms of applications and tools included in the honeypot systems that make the honeynet. In this paper, the requirements of honeynet description are studied and a survey of existing description languages is presented, concluding that a CIM (Common Information Model) match the basic requirements. Thus, a CIM like technology independent honeynet description language (TIHDL) is proposed. The language is defined being independent of the platform where the honeynet will be deployed later, and it can be translated, either using model-driven techniques or other translation mechanisms, into the description languages of honeynet deployment platforms and tools. This approach gives flexibility to allow the use of a combination of heterogeneous deployment platforms. Besides, a flexible virtual honeynet generation tool (HoneyGen) based on the approach and description language proposed and capable of deploying honeynets over VNX (Virtual Networks over LinuX) and Honeyd platforms is presented for validation purposes.

1 INTRODUCTION

Honeypot is an important tool to analyze the adversary's behaviour. The definition of honeypot is: A honeypot is an information system resource whose value lies in unauthorized or illicit use of that resource (Spitzner, L., 2003). A honeynet is a network of honeypots. A honeynet can consist of high-interaction honeypots or low-interaction honeypots or both of them. High-interaction honeypot is designed to capture extensive information on threats. Opposite to low-interaction honeypot which just emulates the operating systems and services, high-interaction provides real systems, application, and services to lure adversaries' snoop and attack. Honeynet creates a highly controlled network, which system administrator can control, and monitor all activity that occurs inside it.

However, one of the biggest challenges we face with most security technologies, including honeynet, is configuration. Especially for the dynamic honeynet (Spitzner, L., 2010), it is very important to reconfigure the honeynet when the network environment is changed. For example, a honeynet is deployed to clone a target network, if the target network changes its configuration, then the honeynet has to be reconfigured too. Another case in point is that the IRS (Intrusion Response System) can divert the intrusion traffic into the honeypots to do investigation. The intrusion traffics also can impact the network environment. Thus, one issue of IRS is to dynamically create and configure the honeypots.

However, it is a complex task to create and configure the dynamic honeynet manually. Thus, it is necessary to use an automated honeynet tool to create, configure, and deploy the dynamic honeynet. Nevertheless, in this kind of automated honeynet tool we need a network description language to make the configuration of the honeynet.

Network description language is used to describe the topology, configuration, and other information of a network. In the network security community, there isn't a network description language against the network of honeypots. As we all know, honeypot can be established based on the physical device. However, with the advantages of virtualization technology, honeynet also can be installed in one host system. What's more, virtual network is a built-in feature of linux kernel, and many more virtual network tools, such as mininet, libvirt by linux bridge, openvswitch, etc. are available. Thus, it is necessary to consider creating a flexible honeynet generation tool that can deploy honeynet over different platforms based on a technology independent honeynet description language.

In this paper, a Common Information Model (http://dmtf.org/standards/cim) like technology independent honeynet description language (TIHDL) is proposed for describing the scenario of honeynet. Based on the TIHDL core, a honeynet request, configuration and management language is made. This language is used in a flexible virtual honeynet tool, which can comprise a combination of heterogeneous platforms for deploying honeynet and also can deploy hybrid honeynet that mixes the low-interaction honeypots and high-interaction honeypots on one virtual network.

The organization of this paper is as follows: in section 2, the technology independent honeynet description language is proposed; in section 3, the architecture of our system is demonstrated; in section 4, a proof of concept and the validation are made; in section 5, a conclusion is made, and some future work is proposed.

2 HONEYNET DESCRIPTION LANGUAGE

2.1 A Typical Honeynet

In this part, one typical honeynet is presented. Using this typical honeynet, both the adversary's and system administrator's behavior are depicted. Figure 1 shows the typical honeynet. There is an internal subnet that is protected by a firewall and a DMZ subnet that is used to deploy honeynet. In front of the honeynet there is a containment gateway.

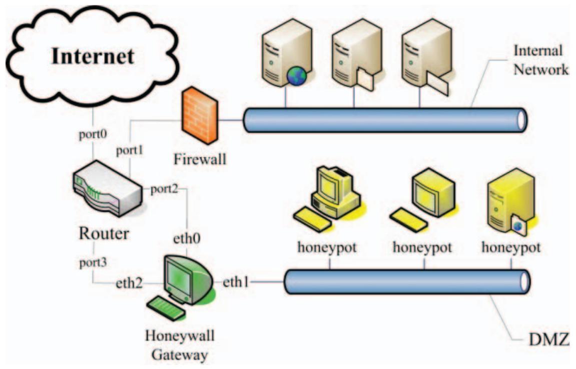

Figure 1: A typical honeynet.

On one hand, from the adversaries' perspective, several steps should be done to compromise a computer system. First, fingerprinting of the system is exposed by some scanning tools, e.g. Nmap (http://nmap.org/) and xprobe2 (http://www.aldeid.com/wiki/Xprobe2). In this step, an intruder can collect the information of the name and version of the operating system, the IP address, the open TCP/UDP ports, and the service provided. Second, the intruder can search the available vulnerabilities of the target system according to the information of the first step. In the second step, Nessus (http://www.tenable.com/products/nessus) is always a good choice. Third, the intruder can use some tools, e.g. Metasploit (http://www.metasploit.com/), to exploit the target system.

On the other hand, from the system administrator's viewpoint, some protective measures must be put into effect. There is a containment gateway called Honeywall which is used to monitor the honeypots, detect the intrusion traffics, observe and record the adversaries' behavior when they compromise the honeypots. At the same time, it also can restrict the outgoing traffics, which means the adversary is confined to compromise other system from the compromised honeypot. Honeywall is a gateway device that separates the honeypots from the rest of the world. Any traffic going to or from the honeypots must go through the Honeywall. This gateway is traditionally a layer 2 bridging/switching device, meaning the device should be invisible to anyone interacting with the honeypots. From the diagram, it can be observed that the honeywall has 3 interfaces. The first 2 interfaces (eth0 and eth1) are what segregate the honeypots from everything else, and they are bridged interfaces that have no IP stack. The 3rd interface (eth2, which is optional) has an IP stack allowing for remote administration.

2.2 The Requirement of Honeynet Description

Table 1: The characteristics for honeynet description.

Network	Application Layer	Protocol Service
	Transport Layer	TCP/UDP port
	Network Layer	IP address
		IP Routing Topology
	Data Link Layer	Network Interface
		Mac Address
	Physical Layer	Device Type
System	Operating System	Name and Version of Operating System
		Name and Version of Software Installed

From both the adversary's and administrator's point of view, the characteristics that must be considered for honeynet description are figured out. The characteristics that have to be used for honeynet description are listed in Table 1. We classify them from the angles of the network TCP/IP stacks and the operating system.

We can view this table from bottom to top. The first two characteristics are the name and version of the operating system and the installed software. These two parameters are important for OS and software that always have the vulnerabilities unless they have been patched. Fingerprinting tool can easily detect the OS information that consists of name and version, after that adversary can exploit the vulnerability. Admittedly, system can be updated and patched in time. However, the 0day exploits can compromise the system before it gets patched. The third characteristic is the device type, because in a typical honeynet, there are different devices such as router, containment gateway and honeypot system. They should be identified clearly. Furthermore, the Mac address is used to identify every network interface on the devices. However, some interfaces do not have IP stack but they are linked with other interfaces, thus we need the network interface link to describe this sort of relationship. Moreover, we must know the IP address of each system, and the routing topology. The adversary needs them to find a target system to compromise, and the system administrator needs them to deploy the honeynet. In addition, the systems on internal network always provide services, which always bind ports. As we all know, one particular attack or misconfiguration always aims at one kind of service. The adversary can use the vulnerabilities of service to compromise the system. Thus, another parameter is the sort of service. There are several services can be deployed on server, such as Web service, FTP service, DNS service, Mail service, VoIP service, etc. Besides, a provided service is always combined to an open port. Thus, the last two characteristics are the open port and the service.

All in all, these nine characteristics are only the basic requisite for typical honeynet description. The desired honeynet description has to be able to describe these nine characteristics at least.

2.3 A Survey of Existing Network Description Languages

Even though, there are some description languages, which can be used to describe either networks or systems. For example, ns-3 (http://www.nsnam.org/), a network simulator, provides a complete language for simulating network, and SysML (http://www.omgsysml.org/), the System Modeling Language, offers a flexible and expressive language for system engineering. However, security issues always makes up of network security and computer system security. Thus, for honeynet description, it is necessary to elect a language that describes both network and system. Now that the requirement items for honeynet description are proposed. We can use them to study the existing network description languages. We investigated ten network description languages. They are YANG (Bjorklun, M., 2010), NDL (Grosso, P., et al., 2007), NML (Ham, J.J. van der, et al., 2013), INDL (Ghijsen, M., et al., 2012),

Table 2: A survey of existing network description languages for honeynet description.

Languages / Requirements	YANG of NETCONF	NDL /NML /INDL	NDML+ of CANDY	VXDL	BNF style template language of Honeyd	CIM	Rspec of SFA	GLUE 2.0
Device Type	+	+	+	+	O	+	+	+
Mac address	-	O	-	O	+	+	O	O
Network interface Link	-	+	+	+	-	+	+	-
IP address	+	O	-	O	+	+	+	+
IP routing topology	O	-	-	-	+	+	O	+
Transport layer port	+	-	-	-	+	+	-	-
Application	O	-	O	+	-	+	O	+
Service	O	-	O	O	O	+	+	+
Name and version of Software installed	O	-	-	+	-	+	O	+
Name and version of operating system	O	-	+	+	+	+	+	+
Legend: "+" means "available"; "O" indicates "limited available"; "-" represents "not available"								

NDML+ (Luntovskyy, A., et al., 2008), VXDL (Koslovski, G. P., et al., 2009), Template language of Honeyd (Provos, N., 2004), CIM, Rspec of SFA (http://www.protogeni.net/trac/protogeni/wiki/RSpec) and Glue 2.0 (Andreozzi, S. et al., 2009).

The result of the survey of the existing network description languages are demonstrated in Table 2. From the survey of these existing languages, it is apparent that CIM is the best choice for honeynet description. However, CIM only meets the basic requisite for honeynet description, and it lacks some other useful information, e.g. the level of interaction of honeypot. Furthermore, CIM is not a configuration language. So it can't be adopt directly to configure the scenario of honeynet. Several characteristics of configuration have to be provided in the desired language. Thus, the best way is to create a new CIM like technology independent honeynet description language.

3 HoneyGen

In this section, a flexible virtual honeynet generation tool (HoneyGen) is presented. The architecture of HoneyGen is shown in Fig. 2. Through the whole HoneyGen architecture, the main components are the request processor, the configuration engine, the catalog of honeynet template, the specific translation module and the deployment tools. Besides, the request description, TIHDL and the development tool configuration are employed separately in each step for the honeynet scenario generation.

3.1 The XML Syntax of TIHDL

Figure 3 is the core of TIHDL. This core includes 12 classes and 11 associations. All of the associations are totally employed from the corresponding CIM models. Some classes are immediately adopted from the corresponding CIM models, such as the class ComputerSystem, OperatingSystem, Software, Service, and ProtocolService. Some classes inherit from the corresponding CIM classes, e.g. the class NetworkInterface inherits from the CIM class NetworkPort, the class Net inherits from the CIM class ConnectivityCollection, and the class Honeynet inherits from the CIM class Network. At last, the other three classes ContainmentGateway, Router and Honeypot are proposed by TIHDL and inherit from the CIM class ComputerSystem.

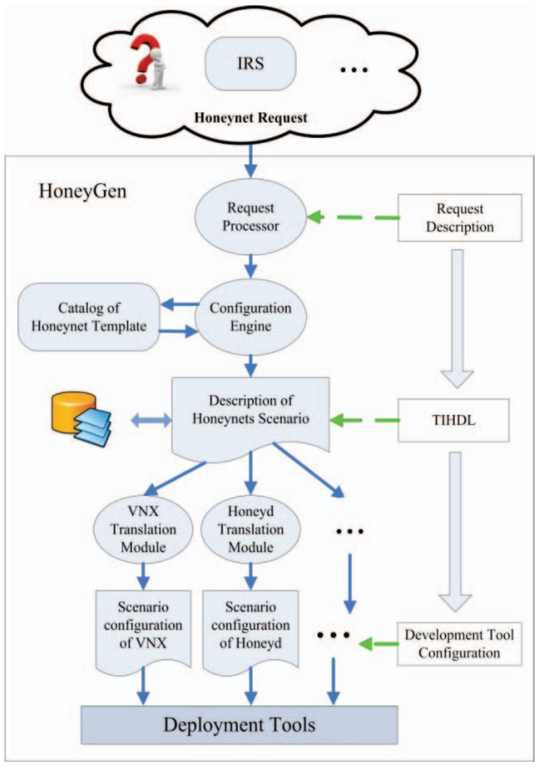
IRS
...
Honeynet Request
HoneyGen
Request Processor
Request Description
Catalog of Honeynet Template
Configuration Engine
Description of Honeynets Scenario
TIHDL
VNX Translation Module
Honeyd Translation Module
...
Scenario configuration of VNX
Scenario configuration of Honeyd
...
Development Tool Configuration
Deployment Tools

Figure 2: The architecture of HoneyGen.

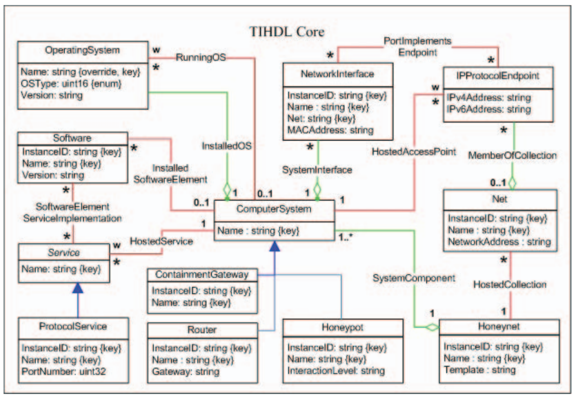

Figure 3: The core of TIHDL.

Based on the core of TIHDL, a honeynet request, configuration and management language schema using XML syntax is made. The XML schema of CIM isn't adopted for several reasons. First, the XML schema of CIM is complex and not easy to know well for the users who are not familiar with CIM. Second, many elements of XML schema of CIM is redundant for honeynet description, but it lacks some requisite elements, e.g. the level of interaction. Third, it is not convenient to use the XML schema of CIM to describe the scenario when the users make a request. For example, the users even have to describe every association. Forth, it is also need to add some more attributes for request and reconfiguration in the XML schema. An example of honeynet scenario based on the XML schema of TIHDL can be depicted as follows:

```
<honeynet>
  <name>test_dit</name>
  <net id=“1”>
     <name>net0</name>
  </net>
   <net id=“2”>
     <name>net1</name>
  </net>
  <net id=“3”>
     <name>net2</name>
  </net>
  <net id=“4”>
     <name>net3</name>
  </net>
  <router id=“1”>
      <name>r1</name>
      <if id=“1” net=“net0” >
         <name>eth0</name>
         <mac_addr>00:00:00:ee:db:11</mac_addr>
         <ipv4>192.168.1.10/24</ipv4>
    </if>
    <if id=“2” net=“net1” >
         <name>eth1</name>
         <mac_addr>00:00:00:ee:db:12</mac_addr>
         <ipv4>10.1.0.1/24</ipv4>
    </if>
    <if id=“3” net=“net2” >
         <name>eth2</name>
         <mac_addr>00:00:00:ee:db:13</mac_addr>
         <ipv4>10.2.0.1/24</ipv4>
    </if>
    <route id=“1”>
         <dst>default</dst>
         <gw>192.168.1.1</gw>
    </route>
    <operating_system>
          <name>ubuntu</name>
           <version>12.04</version>
    </operating_system>
 </router>
 <containmentgateway>
    <name> honeywall</name>
    <if id=“1” net=“net1” >
        <name>eth0</name>
        <mac_addr>00:00:00:ee:db:14</mac_addr>
    </if>
    <if id=“2” net=“net3” >
        <name>eth1</name>
        <mac_addr>00:00:00:ee:db:15</mac_addr>
    </if>
    <if id=“3” net=“net2” >
          <name>eth2</name>
          <mac_addr>00:00:00:ee:db:16</mac_addr>
          <ipv4>10.2.0.2/24</ipv4>
      </if>
      <operating_system>
             <name>Roo</name>
             <version>1.4</version>
      </operating_system>
   </containmentgateway>
   <honeypot id=“1”>
        <name> pc1</name>
        <interaction_level>high</interaction_level>
        <if id=“1” net=“net3”>
           <name>ethernet adapter</name>
           <mac_addr>00:00:00:ee:db:17</mac_addr>
           <ipv4>10.1.0.2/24</ipv4>
       </if>
        <operating_system>
           <name>Windows XP professional</name>
           <version>sp2</version>
        </operating_system>
        <software id=“1”>
             <name>FoxMail</name>
             <version>5.0</version>
        </software>
</honeynet>
```

This technology independent honeynet description has 4 nets, 1 router, 1 containmentgateway and 1 honeypot. HoneyGen has its own configuration file where the user can specify some platform related parameters, i.e. set Honeyd as the low interaction honeypot, indicate KVM based virtual machine as the high interaction and specify the Honeywall as the containmentgateway. In this description, the containmentgateway is a Honeywall, so it has one layer 3 interface and two layer 2 interfaces. The honeypot namely pc1 is installed Windows XP professional Sp2. The pc1 is also installed a software called FoxMail5.0, which has the buffer overflow vulnerability.

3.2 Catalog of Honeynet Template

The catalog of honeynet template is preset as a repository in this system. The TIHDL is used to describe the honeynet template. The name of the template file represents the template name in the repository.

3.3 Request Processor

The request processor is the trigger of HoneyGen. It provides an API that allows the user to call for honeynet scenario. When it receives a request, it will check the request syntax and the values of the elements. If everything is correct, it will activate the tool to generate the honeynet on demand. The request processor provides a flexible rule to receive the honeynet request. First, it can accept the request that describes every honeypot system and all of the values of the elements. Second, it also can accept the request that only provides a template name.

3.4 Configuration Engine

Configuration Engine is used to process the request from request processor. If the request only provides a template name, it will search the template in the repository according to the template name and then produce the configuration file. What's more important, our configuration engine can reconfigure the configuration of the honeynet on the fly. All of the elements (e.g. net, router, honeypot, if) with complex type in the XML schema have an attribute called "request", which can be set to four values: create, add, del and set. If the user wants to modify the configuration, the attribute "request" must be specified.

3.5 Transformation Module

The general configuration file cannot be used to deploy honeypots, but it can be translated to be a specific configuration file that can be recognized by the specific deployment tools. Thus, transformation module is developed to take this responsibility. In our tool, Honeyd and VNX (Fernandez, D., et al., 2011) are elected as the deployment tools to take the responsibilities of deploying low-interaction honeypots and high-interaction honeypots. The configuration of VNX scenario is based on XML syntax, thus it does not need to do any syntax change. However, the Honeyd template language is BNF (Backus–Naur Form) based, and the Honeyd configuration template is text based. So, the transformation module of Honeyd must map the general XML based configuration file to the text based Honeyd configuration template.

3.6 Deployment Tools

As we all know that there are many kinds of virtualization software can be used to deploy virtual honeypots, such as VMware, Xen, Honeyd, etc. In our development tool, Honeyd and VNX are elected as the frameworks to deploy low-interaction honeypots and high-interaction honeypots.

Honeyd is a low-interaction honeypot production, which can emulate distributed honeypots and services. However, there are many stand-alone honeypots need a distributed deployment tool to hold and deploy them. On the other hand, high-interaction virtual honeypot also needs guest operating system and virtual machine as the deployment carrier to contain it. In our implementation, VNX is employed to provide guest virtual machine and virtual network for the high-interaction honeypots and the stand-alone low-interaction honeypots. VNX uses qcow2 format file system to startup the virtual machine. The services can be installed on the root file system previously.

Besides, honeypot deployment tool should have the capability for reconfiguring the scenario on the fly. Honeyd has a doorway called Honeydctl to communicate the inner workings of Honeyd. Honeydctl can be used to reconfigure the templates on the fly. All commands used in the regular configuration file of Honeyd template are supported after "honeydctl>", however, this is also the shortcoming of Honeydctl. The user has to input the commands interactively, but instead of interactively interacting with Honeydctl, an automatic capability of reconfiguration is desired. In fact, the author of Honeyd leaves us a UNIX socket located in /var/run/honeyd.sock. Using this socket, we can create a client socket to communicate with the inner workings of Honeyd, in other words, we can reconfigure the Honeyd templates by the client socket with a script that consists of Honeyd commands.

On the other hand, the author of VNX developed the dynamic configuration capability. The user only needs to write a reconfiguration file based on XML syntax, later he can use VNX to automatically reconfigure the scenario by processing the reconfiguration file.

4 IMPLEMENTATION

In this section, our HoneyGen is validated by employing Honeyd and VNX to deploy honeynet.

4.1 Validation

The validation experiment is outlined in Figure 4. Both Honeyd and VNX were validated to be able to deploy the different scenarios based on the transformation modules. The average transformation time is less than 1s, which is very quick. It is not necessary to manually make the honeynet scenario, only needs to send a primitive request, and then the honeypot deployment will be done. After deployment, the user can reconfigure the scenario in term of requirements, and it is very convenient. Besides, the users can also customize their own honeynet template and store them in the repository.

Practically, due to the benefits of the technologies of VLAN and Open vSwitch (OVS), we can add the virtual honeypots into any target network. Figure 4 also shows the way to tag different virtual honeypots into different production networks. The VM1 is tagged on the internal network while the VM2 and VM3 are integrated into the DMZ network.

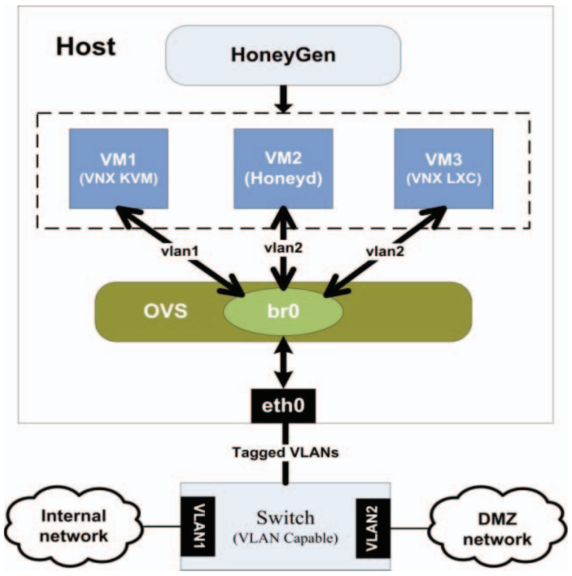

Figure 4: Validation experiment setup.

In addition, several root file systems of honeypots were customized by us for different use cases. Table 3 shows the customized root file systems of honeypots with the functions.

By using these customized file systems of honeypots. Our tool implemented several use cases. The first one was the typical honeynet as figure 1 illustrated, where the containment gateway is the Honeywall. The second use case, we deployed a KVM based cuckoo sandbox (http://www.cuckoosandbox.org/), which is used to analyze the malware and untrusted program as a client honeypot. In the third case, we deployed a scenario which contained a number of Ubuntu LXC-based file systems, which are used to investigate different network intrusions. The TIHDL can describe all of these use cases.

Table 3: The file systems of honeypots.

OS	Honeypot	Function
Roo 1.4	Honeywall	ContainmentGateway
WinXP pro sp2	Honeybot	Capture and interact unsolicited traffic
Win7 pro sp1	Cuckoo Sandbox	Automated malware analysis
Ubuntu 11.10	REMnux 5	Reverse-Engineering Malware analysis
Ubuntu 12.10 LXC	Dionaea, Glastopf, TinyHoneypot	Trap malware and web attacking, keystroke record

4.2 Performance Evaluation

VNX can deploy virtual machine based on KVM or LXC. In the case of virtual machine based on KVM, VNX can emulate various operating systems. Thus, we can use KVM based virtual machine to deploy high-interaction honeypots on demand. The problem of this method is the performance. One example is demonstrated in Fig.5. It costs 40.02s to start up one KVM based virtual honeypot in our host. We also found that after the honeypot switching on, the cost of virtual memory descended to 726MB. However, the physical memory used situation is quite stable after16.87s, it maintained on more or less 160MB, and at the end it stayed on 206MB.

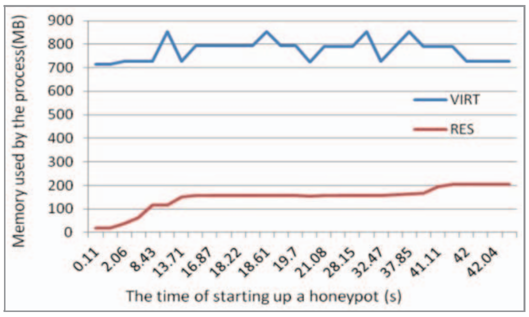

Figure 5: Performance test for one KVM-based VM.

On the other hand, the virtual machine based on LXC only can emulate the Linux operation system that uses the same Linux kernel with the host. Thus, this method lacks fidelity, but it has a high performance even under a large-scale deployment. As far as we know, several low-interaction honeypot productions, such as Dionaea (http://dionaea.carnivore.it/) and Glastopf (http://glastopf.org/) are stand-alone honeypot productions. So we install them in the file system of the LXC-based virtual machine. After that, we can quickly and largely deploy the distributed LXC-based virtual honeypots.

For evaluation of the proposed development tool VNX, we recorded the performance data when we implemented the deployment. The system parameters of the host node were: CPU, 4 Intel(R) Core(TM) i5-3470 CPU @ 3.20GHz; RAM, 16GB; OS, Ubuntu 13.10; Kernel, Linux 3.11.0-26-generic. We took five parameters into account to evaluate the performance: RES (Physical memory used from the process), VIRT (Virtual memory used by the process), %CPU (The percentage of CPU used by the process), %MEM (The percentage of RAM used by the process), TIME+ (The total time of active of this process).

We deployed 10 honeypots on a DMZ subnet alongside a target network. The host system launched 10 processes, and each process corresponding to each honeypots. We recorded the largest value among these 10 processes with those five parameters, and the results were: TIME+, 1:08; RES, 206m; VIRT, 864m; %CPU, 118; %MEM 1.3. The total time for starting up 10 honeypots is less than 5 minutes. But it is a long delay for intrusion traffic redirection into the honeypots. Thus, it is better to keep the high-interaction honeypots running when the redirected intrusion traffics come.

Nevertheless, when we deployed ten LXC-based virtual honeypots, the values of these five parameters were: TIME+, 0:00.68; RES, 37m; VIRT, 168m; %CPU, 22.5; %MEM, 0.2. Form this result, we found that the startup delay of LCX-based virtual honeypots was very short, less than 1 second for 10 virtual honeypots to boot up, and the resource occupation was also quite little. So, if the fidelity of the virtual honeynet is not the most important consideration, for the large-scale virtual honeypots deployment and immediate intrusion response by interesting traffics redirection, the LXC-based virtual honeypots is the better choice.

5 CONCLUSIONS

In this paper, a new approach for the creation and management of honeynets based on the use of a technology independent honeynet description language has been presented. The language is a CIM like flexible language designed to describe honeynets, with a simple syntax easy to understand. It takes into account the characteristics and the special requirements of Honeynets. Besides, a flexible virtual honeynet tool named HoneyGen that uses the specification language to create and modify honeynets has been developed as a tool to validate all the ideas presented. The results of the experiments made show that the HoneyGen can be used to quickly and flexibly deploy virtual Honeynets based on two different deployment platforms: VNX and Honeyd.

For the future work, there are plans to extend the HoneyGen tool to other deployment platforms like cloud infrastructures management tools, to study the automatic model-driven based translation process and to employ this approach in some real security project and deploy the honeynet in some production network to investigate network intrusion.

ACKNOWLEDGEMENTS

This work is funded by the Spanish MICINN (project RECLAMO, Virtual and Collaborative Honeynets based on Trust Management and Autonomous Systems applied to Intrusion Management, with code TIN2011-28287-C02-01.

REFERENCES

Spitzner, L., 2003. Honeypots Definitions and Value of Honeypots. From *http://www.tracking-hackers.com*.

Spitzner, L., 2010. Dynamic Honeypot. From *http://www.symantec.com/connect/articles/dynamic-honeypots*.

Bjorklun, M., 2010. YANG-A Data Modeling Language for the Netowork Configuration Protocol (NETCONF). *RFC 6020*.

Grosso, P., Dijkstra, F., Ham, J. van der, and Laat, C.T.A.M., 2007. Network Description Language -- Semantic Web For Hybrid Networks. In *The TERENA Networking Conference*.

Ham, J. van der, Dijkstra, F., Łapacz, R., and Brown, A., 2013. The Network Markup Language (NML) A Standardized Network Topology Abstraction for Inter-domain and Cross-layer Network Applications. In *The TERENA Networking Conference*.

Ghijsen, M., Ham, J. van der, Grosso, P., and Laat, C., 2012. Towards an Infrastructure Description Language for Modeling Computing Infrastructures. In *IEEE 10th International Symposium on Parallel and Distributed Processing with Applications (ISPA)*.

Luntovskyy, A., Trofimova, T., Trofimova, N., Gütter, D., and Schill, A., 2008. To a Proposal towards Standardization of Network Design Markup Language. In *International Network Optimization Conference (INOC'07), Spa, Belgium*.

Koslovski, G. P., Primet, P. V.-B., and Charão, A. S., 2009. VXDL: Virtual Resources and Interconnection Networks Description Language. In *Networks for Grid Applications, Vol. 2 of Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Springer Berlin Heidelberg*.

Provos, N., 2004. A Virtual Honeypot Framework. In *SSYM'04 Proceedings of the 13th conference on USENIX Security Symposium.*

Andreozzi, S., Burke, S., Ehm, F., Field, L., Galang, G., Konya, B., Litmaath, M., Millar, P., and Navarro, JP, 2009. GLUE Specification v. 2.0. From *http://www.ogf.org/documents/GFD.147.pdf.*

Fernandez, D., Cordero, A., Somavilla, J., Rodriguez, J., Corchero, A., Tarrafeta, L., and Galan, F., 2011. Distributed virtual scenarios over multi-host Linux environments. In *5th International DMTF Academic Alliance Workshop on Systems and Virtualization Management (SVM).*