ISSN 0146-4116, Automatic Control and Computer Sciences, 2021, Vol. 55, No. 8, pp. 1194–1200. © Allerton Press, Inc., 2021.
Russian Text © The Author(s), 2020, published in Problemy Informatsionnoi Bezopasnosti, Komp'yuternye Sistemy.

Implementation of Honeypot Systems Based on the Potential Attack Graph

E. V. Zavadskii[a] and D. V. Ivanov[a, *]

[a] *Peter the Great St. Petersburg Polytechnic University, St. Petersburg, 195251 Russia*

**e-mail: 9361023@gmail.com*

Received April 22, 2021; revised April 22, 2021; accepted April 28, 2021

Abstract—In this paper, we propose an implementation of a honeypot system that uses a method of dynamic resource management based on a graph of potential attacks to provide the ability to deploy a virtual network infrastructure of any scale. Its resource consumption is compared with the traditional honeypot system.

Keywords: network infrastructure, hybrid honeypot system, potential attack graph

DOI: 10.3103/S0146411621080460

A method for dynamic resource management of a honeypot system based on a graph of potential attacks to provide the ability to deploy a virtual network infrastructure of any scale under conditions of limited computing resources is proposed in [1]. According to this graph, the configuration of this network changes, adapting to the actions of the attacker.

The hybrid honeypot system, which is a part of this method, consists of nodes of two classes: virtualized and simulated. Thus, one of the main components of the system is the hypervisor.

The KVM (kernel-based virtual machine) hypervisor, the main advantage of which was its free distribution and the availability of a convenient API Libvirt [2] for the Python programming language, was chosen to create a prototype of a honeypot system based on a graph of potential attacks.

HONEYPOT SYSTEM ARCHITECTURE BASED ON THE POTENTIAL ATTACK GRAPH

The architecture of the implemented prototype honeypot system based on the graph of potential attacks is shown in Fig. 1.

The action collection component is responsible for detecting events produced by the attacker in the system under his control and sending them to the control server.

The system management component manages simulators and virtual machines based on events received from the action collection components.

The control component of the simulated nodes implements the start and stop of the simulators at the request of the control component.

The virtualized host management component provides operations with virtual machines on the network being implemented.

Based on the configuration file supplied to the input, which describes the graph of potential attacks for the network being implemented, the developed system deploys a virtual network and changes its structure in terms of changing the type of nodes.

Action Collection Component

The action collection component consists of two pieces of software: a network client and a kernel event collector.

The kernel event collector is the open source project Fibratus [3], with which it is possible to detect events related to processes, streams, dynamic libraries, files, registry keys, network interaction, and operation with system object identifiers.

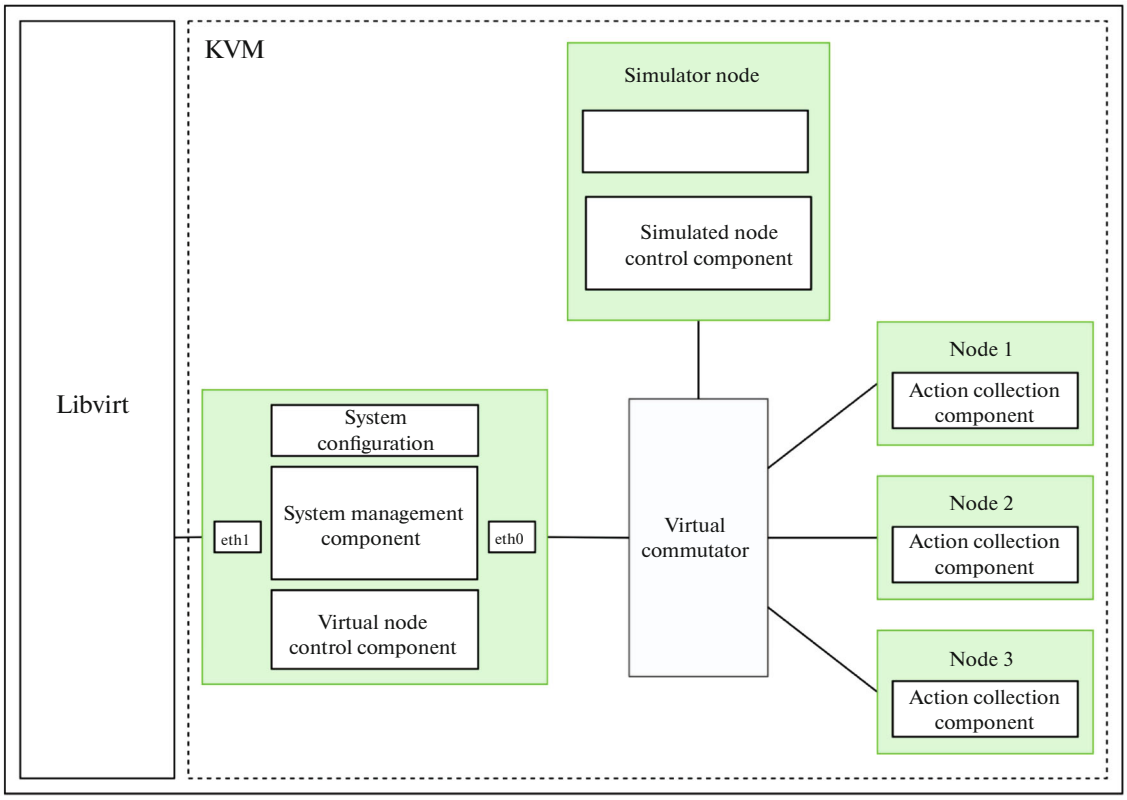

Fig. 1. Honeypot system architecture based on the potential attack graph.

The HTTP client interacts with the system management component to initialize the list of expected events and send detected user actions, as well as launches tools for collecting user actions and processing the monitored events according to the received list. When communicating with the server, the following API requests are used: GET request/agent/init/ is the initialization of the list of events, POST request/agent/event/ is the sending events. The header of each request contains the identifier of the sending node.

The used tools are a part of a prototype honeypot system and do not provide a sufficient level of indistinguishability between a virtual network node and a real device.

Simulated Node Management Component

The simulated node management component is an HTTP server implemented using the modern FastAPI web framework [4] and the Uvicorn asynchronous server [5]. The function of this component is to manage instances of the HoneyD simulator [6].

HoneyD is a daemon that creates simulated nodes on the network that can run on various operating systems and contain various services through the use of the nmap-os-db fingerprint database of the Nmap utility and extension with scripts in Perl and sh [7].

The simulated nodes control component runs the simulator of each node as a separate process, which allows one to disable and enable each of them independently of the others.

To interact with the management server, this component provides API requests shown in Table 1.

Configuration File

The configuration file contains a graph of potential attacks for the implemented virtual network in JSON format, the structure of which is described by a grammar developed using the Pydantic module for the Python programming language.

Figure 2 shows an example of a file describing a two-node virtual network.

Table 1. Requests of the implemented API of the Simulated node control component

Request type	Path	Parameters	Description
POST	/new/	{“Id”: node identifier, “path”: path to HoneyD configuration file}]	Initialization of a structure describing simulators and representing a dictionary, in which the key is the node identifier, and the value is the path to the configuration file
POST	/start/	List of node IDs	Launching simulators, the identifiers of which were passed by the system control component, and saving the object of the Popen class in the previously described structure
POST	/stop/	List of node IDs	Stopping active simulator processes corresponding to the passed identifiers

A node with ID 0 has no a parent (parent_id: null), has one child: the children list contains the child’s ID. Since this node is the entry point to the network, only the virtual machine with the UID 1dd99331-0455-4b7f-bf51-1c3748918e79 is specified for it. The events list contains a description of the events defined by the action collection components and the reactions to them. In this case, if the attacker executes the ssh command 192.168.1.36, then the operation to launch the virtual machine of the node with ID 1 will be performed.

The second node is implemented in the form of a simulator, the path to the configuration file of which is specified in the “simulator” field, and a virtual machine, which is initially in a shutdown state. When


```
[
    {
        "id": 0,
        "ip": "192.168.1.33",
        "vm": {
            "uuid": "1dd99331-0455-4b7f-bf51-1c3748918e79",
            "initial_state": "power on"
        },
        "parent_id": null,
        "children": [
            1
        ],
        "events": [
            {
                "reasons": [
                    {
                        "kevent": "CreateProcess",
                        "name": "ssh 192.168.1.36"
                    }
                ],
                "action": {
                    "operation": "power on"
                },
                "ids": [
                    1
                ]
            }
        ]
    },
    {
        "id": 1,
        "ip": "192.168.1.36",
        "simulator": "/home/user/Downloads/GDS/sim_assistant/configs/2.config",
        "vm": {
            "uuid": "6ee01d43-c9ce-41eb-b0ea-933f66da3db5",
            "initial_state": "power off"
        },
        "initial_system": "simulator",
        "parent_id": 0
        "timeout": 600
    }
]
```


Fig. 2. Honeypot system configuration based on the potential attack graph.

```
{
        “ip”: “Hypervisor IP”,
        “port”: 16509,
        “type”: 1, # TCP-connection
        “login”: “username”,
        “password”: “password”
}
```

Fig. 3. Structure of the connection setup object.

```
class Action(BaseModel):
    operation: PossibleOperation
    snapshot_name: Optional[str]

class Task(BaseModel):
    sim_id: Optional[int]
    sim_action: Optional[str]
    vm_uuid: str
    vm_action: Action
```

Fig. 4. Structure of the tasks of the virtualized node control component.

deploying a virtual network, this node will be presented as a simulator (initial_system: simulator). When the virtual machine is idle for more than the number of seconds specified in the “timeout” parameter, a snapshot of the current state of the system is taken and then it is turned off.

Virtualized Node Control Component

The virtualized node control component performs operations with the virtual machines of the implemented network and interacts with the simulated node control component to synchronize with the currently active node type.

Virtual machines are managed using the Libvirt-python 6.10 module, which provides an interface for remote interaction with the hypervisor. To improve the experience with this API, the Connect and Instance classes were implemented.

To establish a connection to the hypervisor, the username and password that were specified when configuring the hypervisor must be provided. This component is implemented in the form of the Worker class, which is initialized by the ConnectionSetup class object containing the parameters for connecting to KVM (Fig. 3).

The virtualized node control component performs tasks that are placed in turn by the system management component. Each task is described by the structure shown in Fig. 4.

The “sim_id” and “sim_action” fields are optional since some nodes may not have a simulator configuration, as the node with ID 0 described in the previous subsection.

The “vm_action” field contains possible operations on virtual machines:

— power on—turn on the specified virtual machine;

— power off—turn off the node using the ACPI command;

— suspend—suspends the virtual machine while the current state is saved on the hard disk;

— resume—resuming the operation of the node by restoring the previously saved state;

— revert—return the machine to a specific snapshot of the state, the name of which is specified in the “snapshot_name” field of the nested object of the Action class;

— force off—instant disconnection of the node;

— power cycle—instant shutdown followed by turning on of the virtual machine.

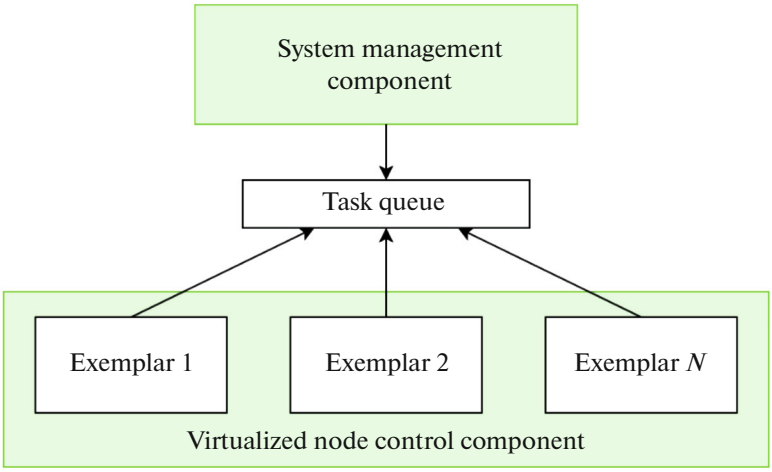

Fig. 5. Scheme of interaction of the system management component and exemplars of the virtualized node control component.

When a task is received, an operation with the virtual machine is initially performed, and then POST /stop/ or /start/ requests are sent to the control component of the simulated nodes to stop or start, respectively.

Taking a snapshot and then restoring it was chosen as the method of suspending and resuming the operation of virtualized nodes, with which the attacker interacted. This method allows one to preserve all of the attacker's work in a given operating system while providing a faster recovery rate than a normal launch.

System Management Component

The system management component receives as input a configuration file with a graph of potential attacks and a file with authentication data for connecting to the hypervisor.

When parsing the graph of potential attacks, the correctness of the structure is checked using the previously described grammar and the consistency of the node configurations specified in it according to the following rules:

(1) There must be no nodes without specifying at least one type of node implementation.

(2) When specifying the initial type of the node (initial_state),

(a) in the absence of a specified simulator, the initial node type must not be a simulator;

(b) in the absence of a specified virtual machine, the initial node type must not be a virtual machine;

(c) if the base state of the virtual machine is "power on", then the simulator cannot be specified as the initial node type.

(3) If both the virtual machine and the simulator are specified in the configuration, then the initial node type must be specified.

(4) In the absence of a specified simulator, the initial state of the virtual machine cannot be "power off".

After checking the configuration file, the control component of the simulated nodes is initialized using the POST request /new/ and then the simulators required at the initial stage are launched.

To manage virtual machines, multiple instances of the virtualized node control component are created. The speed of operations with virtual machines depends on their number, since the operation of returning to a snapshot of the state is blocking. Therefore, this parameter must be selected according to the amount of available resources and the graph of potential attacks of the implemented virtual network.

The interaction of the system management component with the instances of the virtualized node control component is carried out using the task queue (Fig. 5).

This component is an HTTP server that provides an API for interacting with the action collection component:

— initialization of the list of expected events when starting the virtualized node;

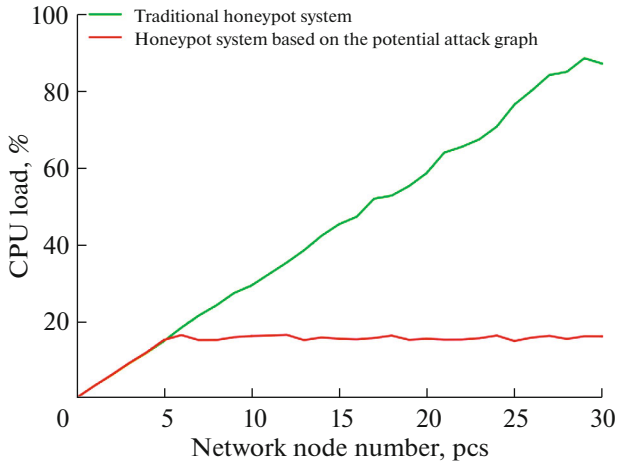

Fig. 6. Graph of the dependence of the processor load on the number of virtual network nodes.

— obtaining data on events caused by the attacker actions.

When events according to the graph of potential attacks are received, a task, which is sent to the queue for execution by one of the instances of the virtualized node control component, is formed.

COMPARISON OF RESOURCE COSTS WHEN USING THE TRADITIONAL HONEYPOT SYSTEM AND THE HONEYPOT SYSTEM BASED ON THE POTENTIAL ATTACK GRAPH

To evaluate the developed prototype of the honeypot system based on the graph of potential attacks, the resources consumed by it were compared with that for the system without using the proposed approach when each node of the virtual network is represented by a virtual machine.

The experiment was carried out using a server with the following characteristics:

— processor Intel (R) Xeon (R) CPU E5-2630 v4 @ 2.20GHz;

— number of cores is 10;

— RAM 768 GB;

— type of hard disk is SSD;

— hard disk capacity is 2.4 TB.

In the comparison, the same virtual machines were used, for which the following resources were allocated: 2 cores, 4 GB of RAM, and a 60 GB hard disk.

Figure 6 shows a plot of the dependence of the load on the central processor on the number of nodes of the deployed virtual networks of honeypot systems.

Similar to a traditional honeypot system, the dependence for a system based on a graph of potential attacks increases linearly until it reaches a fixed number of initially active virtualized nodes. In this case, the network contains five virtual machines while the rest of the nodes are simulated.

The maximal number of concurrent nodes in a virtual network during testing is 30. It is difficult to further increase the size of the network using a traditional honeypot system. At the same time, the honeypot system based on the graph of potential attacks allows the network to continue to grow since the minimal required number of active virtual machines during its operation is maintained by replacing the virtualized nodes with simulated ones in the event of an attacker's inactivity after the time interval specified in the configuration file.

Fluctuations in the processor load pattern are caused by the execution of necessary tasks by the operating systems of the virtual machines.

In the experiment, virtual networks were deployed in a small office consisting of user computers.

Industrial networks, in addition to user computers, include various controllers, HMI panels, and other devices with various software and hardware. Since hybrid honeypots include virtualized hosts, it is necessary to use emulation tools to create virtual machines as the above devices.

WSN networks use low-power devices based on a variety of architectures. However, due to the specifics of attacks on this class of networks, it is necessary to analyze the interaction of a compromised node with

other nodes in order to determine the actions of an attacker. A method for adapting the behavior of nodes, which can be used to create a honeypot system based on a graph of potential attacks for WSN networks, is proposed in [8].

CONCLUSIONS

The developed prototype of a honeypot system based on a graph of potential attacks on the base of the KVM hypervisor, which includes a data collection component, control components for virtualized and simulated nodes, and a system management component, allows deploying both standard office and industrial networks.

Comparison of this prototype with the traditional honeypot system showed that its use requires fewer computing resources and, as a result, less financial cost for creating a similar virtual network infrastructure.

FUNDING

The work was supported by the Russian Foundation for Basic Research, project no. 18-29-03102.

CONFLICT OF INTEREST

The authors declare that they have no conflicts of interest.

REFERENCES

1. Zavadskii, E.V. and Ivanov, D.V., Countering information threats using Honeypot systems based on a graph of potential attacks, *Probl. Inf. Bezop. Komp'yut. Sist.*, 2021, no. 1, pp. 79–85.
2. Libvirt application development guide using python. https://libvirt.org/docs/libvirt-appdev-guide-python/en-US/pdf/Version-1.1-Libvirt_Application_Development_Guide_Using_Python-en-US.pdf. Cited September 21, 2020.
3. Tool documentation Fibratus. https://www.fibratus.io/#/overview/what-is-fibratus. Cited December 21, 2020.
4. Framework FastAPI. https://fastapi.tiangolo.com. Cited December 10, 2020.
5. Asynchronous server Uvicorn parameters. https://www.uvicorn.org/settings/. Cited December 10, 2020.
6. Repository HoneyD. https://github.com/DataSoft/Honeyd. Cited December 11, 2020.
7. Fingeprint database Nmap. https://github.com/nmap/nmap/blob/master/nmap-os-db. Cited December 11, 2020.
8. Ovasapyan, T. and Moskvin, D., Security provision in WSN on the basis of the adaptive behavior of nodes, *Fourth World Conf. on Smart Trends in Systems, Security and Sustainability (WorldS4)*, London, 2020, IEEE, 2020, pp. 81–85.
https://doi.org/10.1109/WorldS450073.2020.9210421

Translated by A. Ivanov