Automatic Security Analyzing Router OS

Abheesta Vemuru, E. John Bruce, A. Veeramuthu

Abstract*: Most, if not all, Routers in this world are Network-embedded Devices that run vendor firmware that controls all of its' integral functions. This router is usually the first line of defense between a user in that particular network and the devices on the Internet. The case is the same in Offices as well as Homes. While the world has always succeeded in rapidly evolving and advancing technology in all fields, the networking field develops at a relatively slower pace. Routers are vital for the security of any Network considering how they are in a privileged position with respect to the remaining devices on that network. Even so, Routers are more often than not, not updated by the users thus leaving various security vulnerabilities. To fix this, concepts and models have been made to analyze device in a large scale by scanning or by manually testing each device. All these tests and analyses are done externally to the router but not as an internal process by the router itself. It would save a lot of resources if the router was automated to test its own security at regular intervals. Thus, this work will involve the building of a Router from scratch using an Operating System with minimal packages, and shell scripts that build a Dynamic Analysis software.

***Index Terms*: Dynamic Analysis Software, Network-embedded Devices, Operating System, Router, Security, Shell Scripts, Vulnerabilities.**

I. INTRODUCTION

Today, embedded systems are everywhere and we interact with embedded systems extensively. Embedded systems are computer systems with dedicated functionality like routing packets, making phone calls etc. These Embedded Systems are also getting smarter and more complex; while providing administrative and non-administrative interfaces. They may implement complex software systems, complex protocols and process various types of data. The software and data which support the functionality of the system is the firmware which is the focal point of our project. Additionally, Embedded Systems are getting heavily interconnected via private and public networks which is also called Internet of Things, which is simply a large-scale ecosystem of Embedded Systems communicating with each other.

Insecure embedded systems exist everywhere including cars, drones, phones, and various other electrical appliances. Most of these devices stay insecure with no analysis or update to improve.

One of the proposed ideas in the past for a better understanding into this problem was Large Scale Analysis of the security of embedded systems. The problem with Large scale analysis is the Heterogeneity of the kind of Hardware, Architecture, Operating Systems, Users, Requirements and Security Goals we find in various devices. Since these are not the same, the analysis has to be extremely dynamic and large scale becomes near impossible. Another possibility to go around this problem is to manually analyze each device.

Manual Analysis though, on the other hand, do not scale well. Finding and downloading firmware, only to re-discover the same or similar bug from other firmware makes the work exponentially redundant. Previous Approaches to this ideology have included tests on real devices that give accurate results but do not scale well enough. Scanning devices solves this by performing large scale testing but can mostly only test for known vulnerabilities and is not easy to perform deep analysis on individual devices. Most Large-Scale testing are black box approaches and scanning devices over the internet is highly intrusive. The administrator/manufacturer will manually or automatically feed in information to the device which helps the device evaluate itself on the issues of security. The device thus may send security reports, vulnerability/exploit reports and more information to the administrator to help in increasing the security of the very first point of security in most networks.The remaining contents of this paper is organized as follows: in section 2 discussed about the related works of our methods. In section 3 discussed about our proposed model. In section 4 discussed about the model and the experimental setup. In section 5 discussed about the results and discussion. Finally, in section 6 concluded our work.

II. RELATED WORKS

From the development of FIRMADYNE [1], it is hoped that security vulnerability assessment in embedded devices becomes easier. It is also hoped that the automated approach helps in finding newer vulnerabilities in a simpler way.

A Large-scale analysis was done to analyze the security of embedded firmware [2] 1.7 million individual files were used unpacking 32000 firmware images. These were analyzed statically leading to the discovery of 38 new unknown vulnerabilities in over 693 firmware images. Largest ever network survey of TLS and SSH servers [3] performed to find evidence that various vulnerable keys are exposed. A minor percentage of the world's TLS certificates share keys due to insufficient entropy during key generation. With TLS Certificates of servers being exposed while the servers were accessible directly on the internet, this affects end to end encryption. At least 1 lakh IPMI-enabled servers are running publicly accessible IP addresses making them vulnerable to various exploits.

Revised Manuscript Received on May 04, 2019.

Abheesta Vemuru, Department of Computer Science, Sathyabama Institute of Science and Technology, Chennai, Tamil Nadu, India.

E. John Bruce, Department of Computer Science, Sathyabama Institute of Science and Technology, Chennai, Tamil Nadu, India.

A.Veeramuthu, Department of Information Technology, Sathyabama Institute of Science and Technology, Chennai, Tamil Nadu, India.

This is in contrast to recommended best practices. Further suggest strategies for servers currently deployed and propose avenues for future work. IPMI firmware [4] are very often not up to date, carry default passwords unchanged apart from the fact that they run on public IP addresses which they should not be doing for starters. These devices are suggested to be isolated on separate networks, while disabled when there is no necessity.

Emulation was done on firmware images for testing of SoC. FEMU [5] was proposed to SoC Verification, being a hybrid firmware/hardware emulation. While it is time consuming, it provides a deeper perspective into the problem and attempts to allow the analyst to get better insight.

A framework called Avatar [6] was built to perform dynamic analysis of embedded devices. It introduced emulators of the embedded devices to demonstrate each image and then ran the tool to uncover vulnerabilities in each image in various versions.

Previous approaches have also included automated dynamic analysis (FIRMADYNE) [7], where they used firmware image files distributed on vendor support websites to automatically unpack the contents to identify the kernel and extract the filesystem. FIRMADYNE uses web crawlers and QEMU [8] emulation to work on sandboxes on each device to emulate vulnerabilities and exploits and analyze if the device is vulnerable. FIRMADYNE works externally from the device and requires physical access to the device to get information about the device. If not the device, FIRMADYNE also analyzes the firmware directly from the vendor websites. The problem with FIRMADYNE is that it is still an external approach and does not allow the device to automatically do the analysis on itself at regular intervals. FIRMADYNE is an operation that requires a lot of overhead.

Previous approaches have also been too intrusive since they're externally testing the target and more often than not black box approaches.

III. POPOSED MODEL

Our Approach is to make each device run tests on itself consistently by discovering live vulnerabilities from online databases and from Manufacturers. Thus, the approach doesn't have to scale well, there is no intrusive testing since the only device that is involved is itself, it doesn't have to correlate across various firmware since it is much more specific to its' resources alone.

Thus, the embedded system will be dedicated for 2 purposes:

- To route packets and function like any other router
- To understand its own firmware's vulnerabilities and exploits that have been found across the world.

The early stages of Development can be done by running the server on a virtual environment. The Server may run any Open Source Linux operating system though we opt for a Fedora distribution. If the choses Linux Distribution varies, the procedure is subject to vary as well. We configure the device to function as a router and further crawl through popular exploit databases to test and evaluate if our device is vulnerable or exploitable from these threats.

The same can even be functioning as a firewall, thus removing the need of standalone devices for routing and firewalls. Accessing their configuration panels is as good as accessing the server itself. The server may also be virtualized to run two separate operating systems, one for its usual functions while the other operating system is used to run the routing and firewall.

The system first virtualizes the guest in an isolated network environment, and monitors all network interactions to deduce the correct configuration for subsequent analyses. A second test after re-configuring the virtualized environment shall occur using the new required network configuration after the inference. This enables network interaction between the host system and the guest firmware that is being virtualized.

In our approach, we use the analyzing system from within the router machine itself so that it self analyzes. In this way, we can completely govern if we want to follow black box, white box or grey box approaches. Having the security analyzing system within the host itself, allows the analyzer to have complete control in the kind of testing to have. Therefore, we set up the machine using an open source Linux Operating System and a customized network setup. The customized network setup would involve bridging the network connections to enable virtualized server Operating systems to run on the same network rather than running on a shared IP for the whole device. This would be the case if the System Administrator chooses to have two separate operating systems for routing and for server functions. The Linux Operating System operates the routing instead of a dedicated router. The network Configuration will be further explained in upcoming sections.

A. Algorithm and Pseudo Code

The following involves the building of the self-analyzing security script which is given in Algorithm 1.

Algorithm 1: Linux Exploit Suggester

```
# Check Operating System version, Kernel
# Version and other configuration versions.
UNAME_A=""
# parsed data for current OS
KERNEL=""
OS=""
DISTRO=""
ARCH=""
PKG_LIST=""
# kernel config
KCONFIG=""
# Now pull each CVE file & initialize variable
CVELIST_FILE=""
opt_fetch_bins=false
opt_fetch_srcs=false
opt_kernel_version=false
#Initialize all other variables
#If test if true, print the Result with CVE Banner
Iterate EXPLOITS[index]=
Name:
```

```
$name[CVE-2004-1235]
Reqs: pkg=linux-kernel,ver=2.4.29
Tags:
Rank: 1
analysis-url:
http://isec.pl/vulnerabilities/isec-0021-uselib.txt
bin-url:
https://web.archive.org/web/20111103042904/http://tarantu
la.by.ru/localroot/2.6.x/elflbl
exploit-db: 744
# sort exploits based on calculated Rank
IFS=$'\n'
SORTED_EXPLOITS=sort and assign exploits
unset IFS
#Display the sorted exploits
# Display result (standard)
   Print $NAME
   Print $details
      Print tags from  $tags
      Print rank from $RANK
      Print url from $src_url
      Print url from  $ext_url
      Print comments after assigning them to $comments
      if opt_full is true then
        Add option  -n "$reqs"  & echo it
        Add option -n "$EXPLOIT_DB"  & echo
         author=$ search for “author” from $EXP
         Print author
         If end
```

B. Operating System

The operating system used in this Router is Linux Kernel based. We use a Fedora [9] Distribution. Red Hat [10] Enterprise Linux 7 is used in our setup, but it is possible to be replaced by any other Fedora System and possibly by other Linux Distros as well. CentOS [11] is a non-commercial version of Red Hat Enterprise Linux which is similar to our setup. The same operating system is used in experimental devices as well. Using a minimal version of this Operating system is optimal and efficient in this setup. The Linux Operating System used versus a regular Router OS gives us the advantage of functionality as well as the merge of 2 separate devices, that is Server and Router into one single device. But another option that the System Administrator has is to run the Router OS as a separate OS on the device while running the server functions separately on another OS.

Note that it is recommended to have the System Administrator uses a user of no write permissions. The user should also not be in the sudoers file. This way, whenever root is not logged in, no malicious scripts can be executed and attackers cannot compromise the security system itself.

IV. EXPERIMENTAL SETUP

A. Network Topology

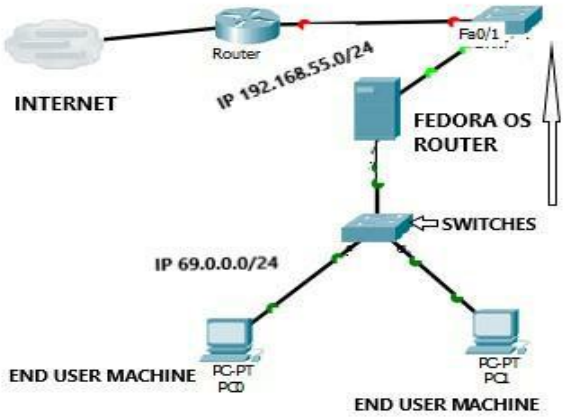

Figure 1 Network topology setup

The Fedora Based OS Router acts with two or more interfaces and routes between networks. We use any machine that can run an Operating system ranging from any microprocessor to a full-fledged server. Thus, removing the need of any dedicated router in this setup. Note that the two networks connected to the Router have different Network IPs in the example topology setup is shown in Figure 1.

In the scenario that the System Administrator runs the server and the router Operating System separately on the same system, both these operating systems should be bridging the connection to make sure they are on the same network. Or, the router may be given the bridged connection while the server gets an internal NAT connection which is not in the scope of this project.

B. Virtual Networks

We run our project in a virtualized environment on VMWare Workstation [12]. The network needs to be customized via the Virtual Network Editor details as shown in Figure 2.

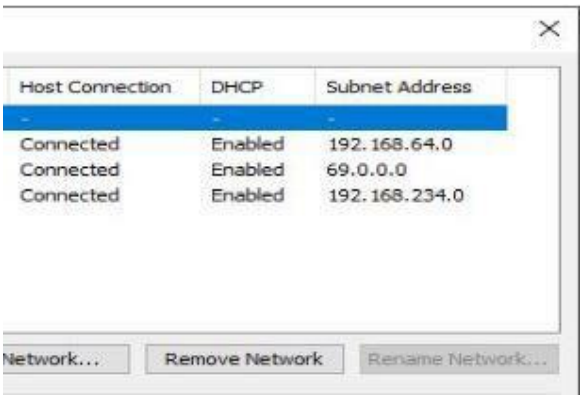

Figure 2 Virtual networks details

Among the four networks shown in Figure 2, two are used. The first one highlighted is a Bridged connection that acts like a real physical connection and shares the same address as the real network of the host machine’s network address.

This allows the Router OS we are virtualizing to have the same IP as our host machine has. The second network being used in this setup is 69.0.0.0/24 which is a virtual classless network. In the following figures, an example configuration of the device and the router adapters are given.

Figure 3 End user device network configuration

In Figure 3 shows the adapter of the device that acts as the end-user. The device has only one adapter that is connected to the 69.0.0.0 network with a subnet of 255.255.255.0. It has been given the address 69.0.0.129 by our router.

In Figure 4 represents the configuration of 2 adapters in the router machine. First adapter represents the connection a router has to the WAN network while the second interface represents the LAN network that the router connects to. This LAN network is what our end user device connects to.

- LAN - 69.0.0.0/24
- WAN - 192.168.55.0/24

The device thus routes packets to and from these two networks, therefore acting like a fully functioning router. All packets received from one network that want to access the other network, pass through this router as it acts as the gateway for the network 69.0.0.0/24.

Figure 4 Router adapter network configuration

Ping tests from the device to the internet as well as the Router Gateway prove that our router is indeed functioning as required.

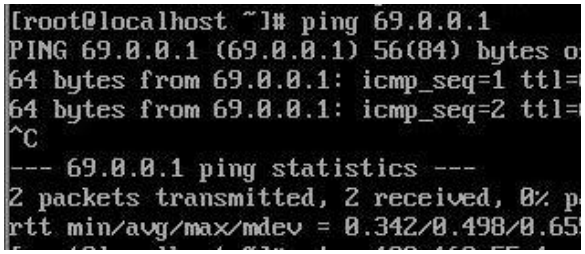

Figure 5 Ping test from device to gateway

Ping test from Device to gateway are successful as shown in Figure 5.

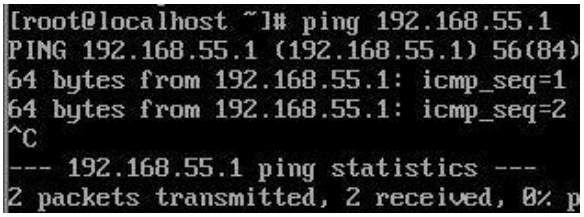

Figure 6 Ping test from device to external node

The Router (gateway) is also able to ping its own gateway in the WAN network is shown in Figure 6.

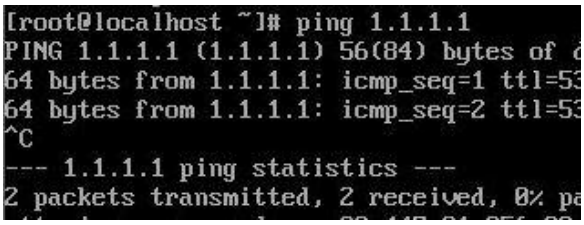

Figure 7 Ping test from device to Internet

An attempt to ping the internet from the device is also successful as we try to ping 1.1.1.1 DNS server, as shown in Figure 7.

Fig. 8: Ping test from device to Internet hostname

We also try to test if domain resolving is working and thus ping a hostname. It is successful which is shown in Figure 8.

V. RESULTS AND DISCUSSION

This Module runs on a Shell Script (Refer Algorithm 1) that reads banner versions of the device and crawls through the internet to find possible exploits. More often than not, fixing these vulnerabilities is updating the system and patching it. Which can be added in a cronjob. There can be more custom scripts added to run and piped into files for logging the assessment. Since the router machine is as good as any Open Source Linux Operating System, we can customize the router as much as we want and thus create the security system that is suitable for any scenario. Various ways of identifying exploits include identifying the banner of the operating systems, kernel, or versions of the softwares/services that are run. Scripts can be run in various languages to automate the analyzing and crawling of the world wide web for new vulnerabilities and exploits found for each version. Here, we use a Linux Exploit Suggester Shell Script that enumerates the version of our machine and then identifies potentially harmful exploits that apply for this version of the machines' configuration.

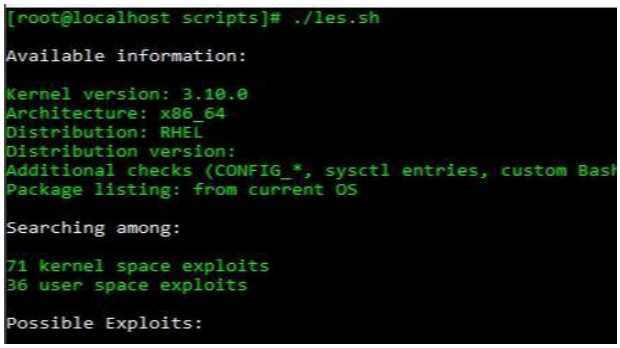

Figure 9 Result of shell script execution

The result of the shell script is shown in Figure 9, first enumerates the version of the Linux Machine and then searches through available exploits across the worldwide web. The results are then printed under the section "Possible Exploits". This result can be then saved into a log file. An even better way to run this would be to execute this shell script as a cronjob and save the output into a text file which is shown in Figure 10. Thus, this text file works as a log file which regularly gets updated by the automated system of crontab.

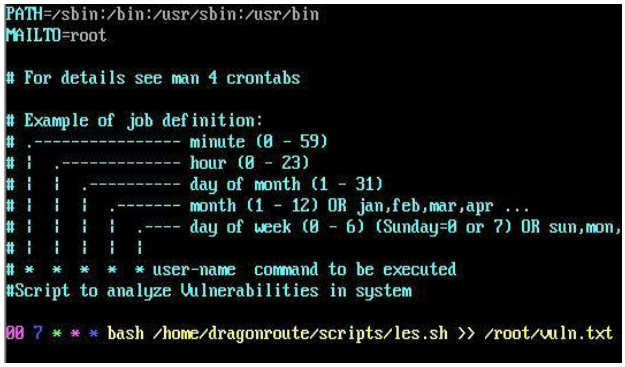

Figure 10 Adding the shell script as cronjob

Now, the admin can simply access the vuln.txt file regularly to check the security health of this particular router. The admin can also do this remotely, and save the file in lesser privileged accounts as needed. In-out current setup, exploits found from this method include the famous Dirty Cow which is given in Figure 11.

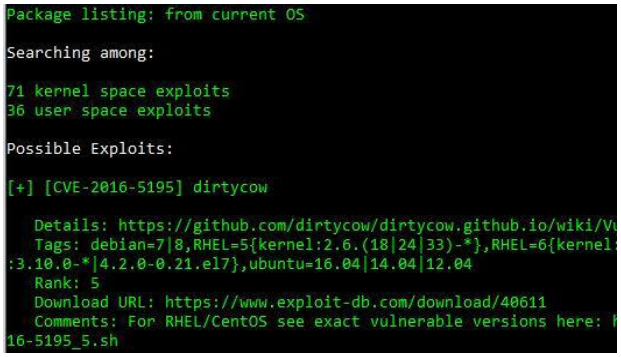

Figure 11 Dirty cow exploit found

Further cronjobs can be added to run more dynamic scripts and even to update or upgrade the system at regular intervals. The output of update and upgrade should also be logged into a file to keep a track of any errors and to also understand all distributions that have been updates and upgraded.

VI. CONCLUSION

This method helps us decrease the amount of hardware being used in a normal topology. Thousands of rupees or sometimes lakhs, are used on networking devices even when the overhead taken by the networking device can be pretty much accommodated within the free space of a typical server in the topology. Servers can double up to work as a router/firewall and thus save a lot of money. Furthermore, Servers have been a mandatory unit in most topologies. Considering the fact that servers are inexpensive and irreplaceable, routers can be replaced instead by powerful devices that have the power to route instead. This also gives us more control. Updating a router is usually dependent on when the OEM releases the patch and the capability of the router always depends on the manufacturer. When we let servers run as routers as well, we use open source operating system that anyone within the team run scripts/commands to modify the router as and when required. This allows for faster fixes of patches and also allows for complete integrity over the device. Money can also be spent directly on buying the hardware required to upgrade the server as required rather than on buying a completely different device for the same purpose. From this Concept we build a fully functioning router than analyzes its own security because of the fact that it is running an Open Source Operating System. Using and encouraging the use of Open Source Operating Systems in routers also encourages the community to test various configurations and scenarios and also write various scripts which help in making the system far stronger. Therefore, it is quite decisive, that the use of Open Source Operating System to route in a network and self-analyze security is a far more efficient and progressive idea.

REFERENCES

1. Daming D. Chen, Manuel Egeley, Maverick Woo, and David Brumley, "Towards Automated Dynamic Analysis for Linux-based Embedded Firmware" Internet Society, pp.1-16. 2016.
2. Andrei Costin, Jonas Zaddach, Aurélien Francillon, and Davide Balzarotti, Eurecom, "A Large-Scale Analysis of the Security of Embedded Firmware" Proceedings of the 23rd USENIX Security Symposium, USENIX Association, pp. 95-110, 2014.
3. Nadia Heninger, Zakir Durumeric, Eric Wustrow, J. Alex Halderman, "Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices" Proceedings of the 21st USENIX Security Symposium, USENIX Association, pp. 1-21, 2012.
4. Anthony J. Bonkoski, Russ Bielawski, J. Alex Halderman," Illuminating the Security Issues Surrounding Lights-Out Server Management" Proceedings of the 7th USENIX Workshop on Offensive Technologies, p. 1-9, 2013.
5. Hao Li, Dong Tong, Kan Huang, Xu Cheng, "FEMU: A firmware-based emulation framework for SoC verification" IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, pp.257-266, 2010.
6. Jonas Zaddach, Luca Bruno, Aurelien Francillon, Davide Balzarotti, "Avatar: A Framework to Support Dynamic Security Analysis of Embedded Systems' Firmwares" pp. 1-16, 2014.
7. "FIRMADYNE" [Online], Available: https://github.com/firmadyne/firmadyne
8. "QEMU" [Online] Available: https://www.qemu.org/
9. "Fedora" [Online]. Available: https://getfedora.org/
10. "Red Hat" [Online]. Available: https://www.redhat.com/en
11. "CentOS Project" [Online] Available: https://www.centos.org/
12. "VMWare Workstation" [Online] Available: https://www.vmware.com/
13. "Dirty Cow Exploit" [Online] Available: https://dirtycow.ninja/