nodes to miniaturized computers without considering the loads on each miniaturized computer. So one miniaturized computer may finish work much earlier than some other ones. On the other hand, each individual computers to be scanned for vulnerabilities may have different number of services running there. Thus, it may take significant long time to scan a computer than others. We designed two dynamic load balancing schemes: load balancing based on CPU loads of penetration testing boxes and load balancing based on services running on individual computers to be scanned. We illustrate these two schemes as follows.

The dynamic load balancing scheme based on CPU loads of penetration testing boxes monitor changes on the miniaturized computers – such as current CPU usage – and factor these changes into the algorithm for distributing vulnerability assessment tasks among miniaturized computers in the resource pool. The proposed algorithm works as follows. At the beginning, each miniaturized computer is assigned one node for scanning. Then the vulnerability scanning process is started. Some miniaturized computers may finish vulnerability scanning earlier than others. Generally speaking, when the scanning is just started, the CPU load is high. Then it is lowered after certain amount of time. This can be seen from Figure 9 which depicts the CPU loads of 1-minute, 5-minute, and 15-minute duration of the scanning. Once the CPU load on one miniaturized computer drops below certain threshold, a new node in the network will be assigned to it for scanning. The process continues until all nodes in the network are scanned.

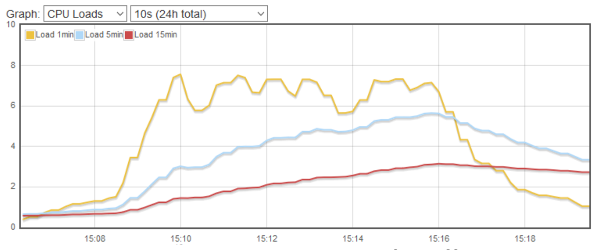

Figure 9. Penetration testing CPU load for different time durations

An example is as follows: miniaturized computer M and N are employed for scanning subnet G on a network. Subnet G contains 6 hosts, Host A, B, C, D, E, and F. When an OpenVAS scan of the subnet is initiated from the AWS server, each host out of these 6 will be assigned either miniaturized computer M or N for vulnerability scanning purpose. At the beginning, node A is assigned to M and node B is assigned to N. Then M and N start their scanning processes. Usually there is a surge on CPU usage when the vulnerability scanning process just starts. After a certain amount of time, the CPU load starts to drop. Let us say, after 5 minutes, CPU load on the miniaturized computer N drops below the preset threshold. Then node C is assigned to N for vulnerability scanning also. Now N is responsible for finishing up scanning for node B and starting scanning node C. This process continues until all 6 nodes are scanned for vulnerability assessment.

The dynamic load balancing scheme based on services running on individual computers to be scanned works as follows. First a port scanning is performed on the nodes in the network. This can be done by using the round robin scheme by assigning nodes to miniaturized computers. This step is similar to the steps illustrated in Section V.A. The only difference is that only port scanning is performed here. The port scanning process is used to identify the number of services running on individual nodes. This is because the more the number of services running on a node, the longer it takes to scan vulnerabilities on the node. After the port scanning step is finished, different number of miniaturized computers are assigned to each node in the network for vulnerability scanning. The assignment process is weighted based on the number of running services on a node. Multiple miniaturized computers assigned to the same node can perform the vulnerability scanning in parallel to speed up the process.

The benefits of dynamic load balancing scheme based either on CPU loads of penetration testing boxes or services running on nodes are increased efficiency and performance of vulnerability scanning process. The results from dynamic load balancing proved the advantage over the simple round robin scheme.

5.3 Clustering Scheme

The next option is clustering. There are resources out there on miniaturized computer clustering resulting in improved processing power [23, 24]. Cluster computing consists of multiple computing nodes interconnected so that they act as a single, more powerful, computing node [23, 24]. This type of computing scheme is actually used for supercomputers across the world. The idea behind this option is that if multiple miniaturized computers are scanning the same subnet, they can be clustered together to act as one, more powerful, node. This would increase the processing power of the new powerful “single-node” penetration testing box and again increase the overall performance of the system as a whole.

An example is as follows: There are 5 miniaturized computers used for vulnerability assessment for services running on a network. Miniaturized computer M, N, and O are employed for vulnerability scanning for subnet A, while P and Q are utilized for vulnerability scanning for subnet B. When the AWS server determines that M, N, and O are in the same subnet on a network, a clustering command would be sent to the nodes, effectively combining them into one new “single-node”, say, node MNO, which would now combine the resources of all individual nodes into one for undertaking vulnerability assessment tasks. The same would be done for P and Q on subnet B, resulting in node PQ. The advantage of employing clustering scheme is that clustering command will be able to simplify task scheduling for multiple miniaturized computers.

5.4 Utilizing unused resources for vulnerability scanning

This method could also be translated to adhere to the more popular distributed architecture described throughout this paper. As distributed computing projects, such as SETI@home [25], become more mainstream, the idea of distributed computing for combined processing power is becoming more viable for other computing tasks. In this instance we would create software that utilizes the unused resources of other nodes throughout the network for the scanning process of a target node. This option could be seen effectively as a scaled botnet used to maximize resource utilization towards vulnerability scanning.

One example for utilizing unused resources for vulnerability scanning is fairly simple. If we have node M, N, and O on a network, when a vulnerability scan is sent to node M and node N and O are idle, the system would utilize some of node N and O’s processing power for scanning node M for vulnerabilities. This would speed up scanning efficiency without adding extra computing equipment.

6. Conclusions and future work

The need for a safe computing environment will only grow with time. Projects like this provide alternatives to the expensive and resource intensive devices currently being used in the industry for vulnerability assessment and penetration testing. The design, implementation, and experiment of the project prove the viability of automated vulnerability assessment using OpenVAS and miniaturized computers. This paper outlines the methods for the project as well as the theory behind an automated distributed architecture for vulnerability assessment. After overcoming software and hardware incompatibilities, a boilerplate OS image for the Raspberry Pi devices was developed for use on new nodes and to help further enhance interest in the research. Multiple miniaturized computers can be plug into different networks and register themselves with our dashboard application which serves as the command center for performing vulnerability assessment on different networks.

Although penetration testing employing miniaturized computers is still in its early stage, systems utilizing these small low-power and low footprint devices may one day be the standard for IT security analysts worldwide.

This research was done with the intention of being continued, either by researchers or the open source community. Each process has a clear ending point where someone else can pick up and continue to work. Some of the top priority future endeavors are listed here:

- Optimize load balancing scheme for Raspberry Pi’s on the same network for increased performance during network scans.
- Output more details about both the risk level of returned vulnerabilities and the priority in which they need to be taken care of on the dashboard.
- Allow users to choose the depth of their vulnerability scan.

7. Acknowledgments

This project is supported in part by a NKU Faculty Summer Fellowship, a NKU UR STEM project fund, and Greaves Undergraduate Summer Research Fellowship.

8. References

[1] OpenVAS, http://www.openvas.org

[2]Application Penetration Testing,www.trustwave.com/ apppentest.php

[3] E. U. Küçüksille, M. A. Yalçınkaya, S. Ganal, Developing a Penetration Test Methodology in Ensuring Router Security and Testing It in a Virtual Laboratory, in the Proceedings of the 8th International Conference on Security of Information and Networks, 2015.

[4] R. Li, D. Abendroth, X. Lin, Y. Guo, H. Baek, E. Eide, R. Ricci, J. Merwe, Potassium: Penetration Testing as a Service, in the Proceedings of the Sixth ACM Symposium on Cloud Computing, 2015.

[5] P. Tilak, White Paper on Penetration Testing, www.docstoc.com/docs/70280500/White-Paper-on-Penetration-Testing.

[6] X. Qiu, Q. Jia, S. Wang, C. Xia, L. Lv, Automatic Generation Algorithm of Penetration Graph in Penetration Testing, in the Proceedings of the 2014 Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 2014.

[7] B. Stepien, L. Peyton, Innovation and Evolution in Integrated Web Application Testing with TTCN-3, International Journal on Software Tools for Technology Transfer (STTT), Volume 16, Issue 3, 2014.

[8] M. Bishop, About Penetration Testing, IEEE Security & Privacy, Volume 5, Issue 6, 2007.

[9] J. Muniz, Penetration testing with Raspberry Pi, Packt Publishing, 2015

[10] Pwn Plug, www.pwnieexpress.com. (Access date: 2 December 2015).

[11] MiniPwner, www.minipwner.com. (Access date: 7 December 2015)

[12] WiFi pineapple, www.wifipineapple.com. (Access date: 10 December 2015)

[13]Distributed Pentest Using Pi,github.com/ spinnyhatkid/PentestInABox

[14]Distributed Pentest Using Pi – Configuration and README, github.com/spinnyhatkid/PentestSSHConfigs

[15]Nessus, www.tenable.com/products/nessus-vulnerability-scanner

[16] Nexpose, www.rapid7.com/products/nexpose/

[17] Core Impact, www.coresecurity.com/core-impact

[18] Canvas, www.immunityinc.com/products/canvas

[19] A. S. Milani and N. J. Navimipour, Load balancing mechanisms and techniques in the cloud environments: Systematic literature review and future trends, Journal of Network and Computer Applications, vol. 71, pp. 86–98, Jun. 2016.

[20] A. M. Alakeel, A Guide to Dynamic Load Balancing in Distributed Computer Systems, International Journal of Computer Science and Network Security, vol. 10, no. 6, pp. 153–160, Jun. 2010.

[21] A. Sharma and S. Verma, A Survey Report on Load Balancing Algorithm in Grid Computing Environment, International Journal of Advanced Engineering Research and Studies, pp. 128–132, Jan. 2015.

[22] A. T. Chronopoulos and S. Penmatsa, Dynamic Multi-User Load Balancing in Distributed Systems, In the proceedings of 2007 IEEE International Parallel and Distributed Processing Symposium, 2007.

[23] Z. Huang, H. Situ, and P. Werstein, Load Balancing in a Cluster Computer, in the proceedings of 2006 Seventh International Conference on Parallel and Distributed Computing, Applications and Technologies, 2006.

[24] J. Kiepert, RPi Cluster: Creating a Raspberry Pi-based Beowulf Cluster.

[25] Seti@home, https://setiathome.berkeley.edu/