Searching Network Devices for Open Ports and Configuration Information Using Ansible

D. Coyle
Department of Computing
Letterkenny Institute of Technology
Letterkenny, Ireland

***Abstract*—Ansible is a powerful configuration management tool, capable of managing vast infrastructures. Port scanning, a necessary part of network security, can have negative effects on network performance and trigger firewalls and other security measures. Combing the two in a secure, isolated environment, the Ansible playbook accompanying this paper attempts to search for open ports on network devices with considerations given for best scripting practices and overall network security.**

Keywords—Ansible, port scanning, network security, scripting, scalability

I. Introduction

Ansible is an agentless software provisioning and configuration management tool, designed to automate the deployment of applications and services on Information Technology (IT) infrastructure. The platform comes with a management console, which can be used to monitor and alter configurations of deployed applications. As a platform it is capable of managing large-scale infrastructures, and as security is always a critical factor in any infrastructure it needs to provide ways to monitor network security. Port scanning could help identify vulnerabilities in networked devices, which could help prevent a network being compromised. It may also become necessary to log the details and configuration of networking devices for audit or compliance reasons. To meet these requirement, this project uses an Ansible playbook to search for open device ports on networked devices, and attempts to download switch and router configurations via a Secure Shell (SSH) connection.

The motivation for this paper cames from the unusual circumstances at the time of writing, to replace a placement period as a requirement for the completion of the BSc Computing in Data Centre Management degree through Letterkenny Institute of Technology. The choice of technologies and aims of the project were to reflect and apply the learning outcomes of this course in a practical context.

II. Project Goal and Literature Review

A. Research Goal

The goal of this project is to create a functioning Ansible script to search for open device ports on networked devices, and to download switch and router configurations via an SSH connection. The components required to meet this goal have been explored through literature review.

B. Ansible Scripting

Ansible scripts are declarative, and take the form of 'YAML Ain't Markup Language' (YAML) 'playbooks' which are run to achieve the desired state configuration specified by these playbooks [1]. A playbook can be broken down into two components containing prescriptive descriptions of how to carry out an operation, 'plays' and 'tasks' [2]. 'Plays' consist of 'tasks', which in turn call Ansible modules to perform a specific task on the target machine.

Ansible playbooks can be run locally, as long as the Ansible package is installed. For large environments, scripts can be 'pushed' to any number of agentless machines [3]. Such a large number of machines are not required to meet research goals, running Ansible locally on a virtual machine (VM) is sufficient.

Best practice for writing playbooks involves starting simple and iterating with more complexity, with a focus on writing consistently in a human-readable manner [4].

C. SSH Protocol

Ansible achieves its agentless nature entirely by utilising SSH [5]. By default, Ansible assumes the use of SSH keys for remote connections, while passwords can be used if preferred or to escalate privilege on the target system [6].

Ansible uses OpenSSH, a secure implementation of the SSH 2 protocol which is integrated into all Linux distributions [7]. The SSH 2 protocol has displaced the older and less secure SSH 1, with a large reduction in cryptographically weak SSH keys and security issues outside of misconfigurations [8]. If used with current software on an updated Linux distribution, Ansible should encounter no security issues caused by SSH when communicating with agentless machines and networked equipment.

D. Port Scanning

Port scanning determines which ports on a networked device are open. The two protocols involved are User Datagram Protocol (UDP) and Transport Control Protocol (TCP), which each have specific approaches associated with scanning their ports.

UDP has two port scanning approaches associated with it. One relies on receiving Internet Control Message Protocol (ICMP) port-unreachable responses, and the other on server-specific probes for responses [9]. The first approach assumes a port with no ICMP response indicates a live service, which can be inaccurate, or even filtered out by a firewall. The second approach specifically scans specific ports for a response from an expected service, which would indicated the live service active on the port. While this second approach is effective, it is time consuming.

TCP has three main port scanning approaches associated with it, all based on the connection-orientated nature of the protocol – the 'threeway handshake' exchange of sycnhronise (SYN) and acknowledgement (ACK) packets. These approaches are referred to as 'connect', 'stealth', and 'zombie' scanning respectively [9]. Connect scanning establishes a full TCP connection for each scanned port, with a successful connection meaning the scanned port is open. Stealth scanning sends an initial SYN packet, and on receiving a SYN+ACK packet, does not respond with the

final ACK packet. The reception of a SYN+ACK packet confirms a service is running on that scanned port. This approach prevents systems that only record completed connections from registering the scan, leading to the 'stealth' name.

The final TCP port scanning approach, zombie scanning, involves very specific requirements from a remote host that results in zero evidence from any scans. It requires a remote system that is idle and not actively communicating with networked devices, and has an incremental Internet Protocol Identification (IPID) sequence. A system with these characteristics is referred to as a 'zombie'. A SYN+ACK packet is sent to the zombie, and its IPID recorded. Then using a spoofed address, a SYN packet is sent to a port on the actual target system. If the target returns a SYN+ACK packet to the zombie, the zombie will increment its IPID by one and return a reset (RST) packet to the target, meaning the target port is open. If the target port was closed, the zombie will not respond to the RST packet, and its IPID will not increment. The RST response from sending a final SYN+ACK packet to the zombie determines if the target port is open or closed – if the IPID has incremented by two the port is open, if it has incremented by one it is closed.

E. Ethical and Security Considerations

Port scanning using these approaches may trigger firewall and other security application responses [10]. Attempting port scanning on a shared network without the consent of those using it can be considered unethical behaviour, and should be limited to a private network if possible. The design and implementation stages of this project have been influenced by these considersations, resulting in a script that attempts to limit impact on the network, and a virtualised environment with best security practices applied.

III. Design and Implementation

The practical element of this project required a virtualised environment with the necessary software to run Ansible scripts. This environment has been designed with the availability and cost of software in mind to make its results reproducible. The Ansible script created to meet project goals is stored on GitHub and publicly available, the link to which can be found in the Appendices of this document along with an installation guide. Also included is a series of shell commands that were used in previous iterations of the playbook to test port scanning functionality, but were eventually replaced with dedicated Ansible modules.

A. Virtual Machine Configuration

Hyper-V was selected as the virtualisation software for this project, being widely available in most versions of Windows 10. VMWare was considered but was incompatible with available hardware. A new Private virtual switch was created, to isolate all attached virtual machines from the host machine [11]. Device passthrough was also disabled to isolate non-essential hardware components and peripherals from the virtual machine. In order to download required dependencies, virtual machines were temporarily given access to the internet using the default Internal Hyper-V switch, then attached to the new Private switch.

B. Operating Systems and Software

Lubuntu 20.04 was chosen as the operating system as it is freely available, and to account for limited hardware capability. It also comes preinstalled with Python 3, eliminating potential compatibility issues with Ansible running under Python 2.7 [12]. Available security updates were installed along with the necessary dependencies to execute the Ansible playbook. No additional software repositories were required to install these dependencies. The Lubuntu virtual machines were configured as standard Generation 2 virtual machines.

In order to download router configurations using Ansible, a compatible command line interface (CLI) routing device was required. VyOS was chosen to act as a virtual router for the project, being setup as a Generation 1 virtual machine for compatibility reasons. Various simulated environments were considered using Cisco images, but due to problems ethically sourcing a compatible image, these attempts were abandoned for a more suitable alternative. No physical router suitable for this project goals was available.

C. Test Environment

Properly implementing and testing the Ansible playbook required the following test environment, based on the previously mentioned virtual machine configuration. A total of three Lubuntu virtual machines were set up. The first was used as the Ansible host, with the other two used as managed devices to run plays against. All three virtual machines required openssh-server, to facilitate communication between them using Ansible commands. The default IP addresses assigned by Hyper-V were not altered. The Ansible host machine required the additional installation of the following packages: ansible, python3-pip, paramiko. The managed devices were then added to Ansible's inventory under their own grouping.

The VyOS VM required the installation of openssh-server to enable communication with the Ansible host. Once this dependency was installed, VyOS was setup with default credentials [13] and assigned an IP address on the Hyper-V private network, then had its connectivity tested with the other virtual machines. The purpose of this VM was to have its router configuration downloaded by the Ansible playbook. Considering this requirement, it was deemed unnecessary for it to actually route traffic on the network. The VyOs router was added to Ansible's inventory under its own grouping.

D. Ansible Playbook

The Ansible playbook, portscan.yml, is designed to perform two main tasks. Firstly, it gathers and lists port facts on managed devices. To do so, the playbook checks managed devices for the required dependencies, and installs them if not present. Then it uses the appropriate module to find port information, which is then filtered and presented in a human-readable format in the terminal as a debug message. Secondly, the VyOS router is accessed using the appropriate module and has its configuration backed up as a file, saved to the Ansible host machine. Where possible, distribution specific commands were not used when an Ansible equivalent was available. The main modules and plugins used in the playbook, along with the justification for their use, will be discussed.

package_facts – This Ansible module returns information about the packages on a managed device [14]. It gathers the package status from the target device and makes this information available as Ansible facts, which can then be used as variables in plays. It can use 'auto' as a parameter for the package managment system to query, but as this project uses only Debian-based managed devices, this was set to 'apt'. An inventory using Linux distributions with different package managers would require the 'auto' parameter, or have specific parameters used that can be triggered with a

'when:' statement - 'when:' being the clause Ansible uses to check for a condition before running a play [15]. In this case it is being used to check the status of a specific package, and this package is installed on the target system if it is not listed in the gathered facts. This requires passing the 'sudo' password of the target system when initiating the playbook.

listen_port_facts – This Ansible module gathers facts on TCP and UDP ports used by processes on the target system [16]. The module can specifically search for TCP or UDP ports if necessary, but to meet a requirement of this project it was used without additional parameters in order to scan both types of port. The gathered facts are then filtered into a list containing only active 'listening' ports and printed out to console. Process names and ID were not included in this output, being filtered to show only one instance of an active listening port on the target system. This specific play makes use of Ansible's debug function, which is the module used to display the list of ports within the console while to playbook continues to run. It is this module that necessitated the installation of the 'net-tools' package on managed inventory, as the module appears to rely on it to function.

vyos_config – This Ansible module enables management of VyOS router configurations and the active running state [17]. The module is used to back up the running configuration of the VyOS VM directly to the Ansible host. The additional parameters were set to give the configuration file a specific name, and to save it to a location that did not require priveleged access on the Ansible host VM. This module is platform specific, and will not function if the target device is not running VyOS. Ansible does provide a platform-agnostic module compatible with Cisco and other routing devices, 'cli_config' [18]. However due to the issues ethically sourcing Cisco and other manufacturer images, this module was not used as its functionality could not be tested in the test environment.

Initial iterations of the playbook made use of Ansible plays that executed shell commands directly on managed inventory. These commands had to be inserted within plays in the playbook, and while they could be stored as variables, this is not considered best practice as it undermines the value of automation with Ansible [4]. Considering how Ansible is used for configuration management of scalable networks, it would become inefficient if shell commands for multiple client operating systems had to be entered. Additionally, these shell commands could become deprecated at some point, necessitating a rewrite of the playbook. Further attempts to use built-in network commands in Lubuntu such as 'ss' and 'losf' did function when passed directly using shell commands, however as the 'listen_port_facts' module relies on the 'net-tools' package these were not used in the completed playbook. The final version of the playbook is a result of iterative testing, resolving the causes of known failures. The playbook is written to avoid the use of any commands that may throw errors or cause compatibility issues. The amount of dependencies and other packages requiring installation on both the host VM and managed inventory has been kept to a minimum. For those packages that did require installation, they have been installed from official repositories, and can be considered as secure components.

IV. Results

Initial runs of the playbook resulted in failure – no port information was returned and it was not possible to access the VyOS router. It was at this point that required dependencies that were previously not known about were discovered through error messages. Running the playbook indicated that the 'net-tools' package was required to be present for target systems, in order to listen for port information. Once installed, this portion of the playbook worked as expected. 'Listening' ports for each target in the inventory were displayed in the console as intended in the design. The playbook script was then updated to include a play to install 'net-tools' if it was not present on the target systems. A sample of these shell commands has been included in the appendices as a functioning playbook. Of the four functioning commands, three of them have been commented out to allow the playbook to execute as intended against inventory hosts grouped as 'networkhosts'.

However at this point the playbook could still not access the VyOS router. An error message upon running the playbook revealed the Ansible host itself required the 'paramiko' package. The 'paramiko' package in turn required 'python3-pip' package to install it in the first place. Doing so resulted in the script running without errors, but Ansible was unable to 'ping' the VyOS router. It should be noted that the Ansible 'ping' command is not the same as an ICMP ping [19], rather it is a tool for Ansible to test if its inventory has a usable Python configuration – a successful 'ping' returns a 'pong'. An actual ICMP ping revealed that the Ansible host and VyOS router were configured correctly, and capable of network communication.

Further testing revealed that by default Ansible was attempting a standard SSH connection into the VyOS router, which was causing the playbook to not work as intended. In order to successfully use the 'vyos_config' module, and other platform specific network modules, Ansible needs to be instructed to use the 'network_cli' parameter in its 'ansible_connection' variable [20]. This parameter explicitly tells the Ansible host that the protocol to be used is CLI over SSH, allowing the playbook to run as intended. This variable was set in the Ansible host file, which allows variables for specific hosts and groups to added. In this case, the correct connection type was added as variable. When run again, Ansible successfully contacted the VyOS router, but was not able to access its configuration. While now initiating the correct connection type, Ansible was attempting to login to the VyOS router using an invalid username – in this case the non-root user account of the Ansible host. It is unclear as to why this kept occurring, as during the setup of the test environment it was confirmed that the Ansible host could SSH into the VyOS router with the proper credentials. A workaround to this issue involved inputting the username and password for the VyOS into the Ansible host file as variables. In a production environment, this would not be ideal due to the poor security practice of storing a password in plaintext. To remedy this, it is possible to encrypt individual variables using the Ansible Vault and its variable-level encryption feature [21].

Once these changes were made, the playbook ran as intended until it came to saving the VyOS configuration file to the Ansible host. The thrown error message showed it was a permissions issue, as the playbook did not have the rights to save to the initial target directory on the Ansible host. This was changed to a new folder in the 'tmp' directory, 'tmp/routercfg', resulting in the router configuration being saved and the playbook finally running completely as intended. The saved configuration file, 'vyos_config.cfg', can be viewed in any text editor. At this stage the playbook could be considered fully functional in meeting the requirements of this project.

V. Conclusions

In writing this playbook, the supplied Ansible documentation proved to be sufficient in researching specific modules for use in plays. Ansible comes with modules, designed both by Red Hat and community involvement, that can perform tasks normally carried out by specific programs and packages. However there was an issue with the 'listen_port_facts' module not listing a required dependency in its documentation. This did not prove to be a major issue as the thrown error message indicated that the 'net-tools' package was missing. All other modules were able to be used exactly as described in official documentation. Intitial iterations of the playbook made use of direct shell commands to managed inventory, but after researching Ansible best practice this approach was found to be unsuitable. One of the main issues was the possibility of a command being specific to one distribution, and not present on another. Accounting for this using shell commands would become inefficient in a large environment. The solution was to, where possible, strictly use Ansible modules as components in plays. Doing so ensures playbooks can work at any scale, and greatly simplifies the process of rewriting or updating plays. This plays to Ansible's strength as a configuration management platform, and makes it a powerful tool for system and network administration at any size of infrastructure. Although the test environment was small, it could theoretically be expanded to any size by adding machine IP addresses to the Ansible inventory file. This makes the evaluation of Ansible as a viable tool a relatively simple task, as it does not need to be tested at scale to prove effective.

The test environment, while suitable to meet project requirements, also influenced design choices for the playbook. The limited availability of CLI router images especially influenced the router plays, and is the main reasons why the modules used are not platform-agnostic. It was not possible to ethically source a Cisco router image for use, or from other popular manufacturers. If this was not the case, platform-agnostic modules would have been tested. If writing this playbook for an infrastructure that uses these other routing devices, it would be extremely beneficial to have access to them. The availability an open-source alternative routing operating system was extremely convenient, and perhaps a more fitting choice considering how prevalent virtualisation is in modern infrastructure. The compatibility with most major router manufacturers can only be seen as a positive, even if this specific playbook could not test this thoroughly. The choice of operating system was also limited, but not by issues finding images to use. The limited hardware available for this project necessitated a lightweight operating system that was also readily compatible with Ansible. As such, not including the VyOS router VM, this project only included one type of client. The playbook accounted for this by using modules guaranteed to work with Ubuntu-based systems. While generic modules are available, these could not be verified to work with systems due to this limitation. A production environment is likely to have multiple types of devices, which would not be an issue for a playbook written with this fact in mind. Again this shows the versatility of Ansible in a varied environment.

The research conducted as background reading gave insight into how ports function, and their place in the networking process. Of particular concern was the impact on a network port scanning could have, whether by decreasing performance or triggering security measures. An active port, especially one for a common service, could be more likely to be targeted for attack. This makes tracking these active ports an important part of network security, as a port left exposed unintentionally could compromise a network. However the methods of port scanning discussed have performance disadvantages associated with them. Using them in a manner that has minimal impact on a network while still providing useful information would be the most ideal situation. The Ansible playbook uses the 'listen_port_facts' module to search specifically for services that are currently using ports on that system, and is not a total scan of all port numbers. Doing so limits the impact the play has on performance, while still providing useful information. If specific ports were a priority, or only those running active services, the play could be modified to search for a specific list of ports based on those used by popular services.

If repeating this project, several improvements could be made. The test environment, which while adequate to meet project goals, was very resource intensive. Given that the available hardware was not of high-specification, this sometimes led to slowdown and occasional crashes. Rebuilding the test environment in containers as opposed to virtual machines could be a way to overcome these technical limitations. It would also provide the opportunity to test Ansible in a new environment, which would prove beneficial both as an educational experience and potentially for future use in a production environment. Ideally, router images from different manufacturers would be procured in order to test plays that require little customisation, and if there are any compatibility issues that would not become apparent without testing. A more varied use of operating systems would be advisable, especially ones commonly found in use in production environments. This would allow testing the playbook more extensively, and could lead to changes that make it more compatible with any device added to the Ansible inventory. It would also be interesting to test the capability of Ansible module port scanning abilties using different scanning methods. From what was researched from best practice and individual module documentation, doing so would require the use of additional packages and plugins. Without testing these packages, and looking into their availability on various operating systems, it would not be advisable to include these tools without more research. Using such packages would also go against the practice of using modules with as few requirements and dependencies as possible. Despite these suggestions, the end result was an Ansible playbook that functions as intended and meets project requirements. The problems encountered along the development process demonstrate that writing playbooks without proper version control would be extremely inadvisable. The assigned project goals provided the opportunity to further knowledge of network security practices and technologies, and the benefit configuration management software can bring to them. The ability to manage infrastructure at any scale using simple and versatile scripts will likely prove extremely valuable to any individual involved in IT services.

Appendix I – Ansible Script

Ansible playbook 'portscan.yml': https://github.com/danecode/lyit-ansible-project/blob/master/portscan.yml

Appendix II – Installation and Notes

Installation guide and other notes: https://github.com/danecode/lyit-ansible-project

Appendix III – Shell Commands

Shell commands from previous playbook iterations: https://github.com/danecode/lyit-ansible-project/blob/master/betaportscan.yml

REFERENCES

[1] *F. A. Locati, "Learning Ansible 2 – Second Edition: Vol. 2nd revised edition", Packt Publishing, 2016, pp. 16.*

[2] *Red Hat, Inc.,"Ansible In Depth", White Paper, 2017, pp. 2. [Online]. Available: https://www.ansible.com/hubfs/pdfs/Ansible-InDepth-WhitePaper.pdf. (Accessed May 30 2020).*

[3] *F. A. Locati, "Learning Ansible 2 – Second Edition: Vol. 2nd revised edition", Packt Publishing, 2016, pp. 12.*

[4] *T. Appnel, "Ansible Best Practices: The Essentials", 2018. [Online]. Available: https://www.ansible.com/hubfs/2018_Content/AA%20BOS%202018%20Slides/Ansible%20Best%20Practices.pdf. (Accessed: May 30 2020).*

[5] *R. Das, "Extending Ansible", 2016, Packt Publishing, pp. 2.*

[6] *Red Hat, Inc., "Connection methods and details", 2020. [Online]. Available: https://docs.ansible.com/ansible/latest/user_guide/connection_details.html.*

[7] *OpenSSH, "OpenSHH Users", openssh.com, https://www.openssh.com/users.html. (Accessed May 30 2020).*

[8] *O. Gasser, R. Holz and G. Carle, "A deeper understanding of SSH: Results from Internet-wide scans," 2014 IEEE Network Operations and Management Symposium (NOMS), Krakow, 2014, pp. 1-9, doi: 10.1109/NOMS.2014.6838249.*

[9] *J. Hutchens, "Kali Linux Network Scanning Cookbook", Packt Publishing, 2014, pp. 126.*

[10] *M. Ring, D. Landes, and A. Hotho, "Detection of slow port scans in flow-based network traffic", PLoS ONE, 13(9), 2018, pp. 1–18. doi: 10.1371/journal.pone.0204507.*

[11] *Microsoft, "Create a virtual switch for Hyper-V virtual machines", https://docs.microsoft.com/en-us/windows-server/virtualization/hyper-v/get-started/create-a-virtual-switch-for-hyper-v-virtual-machines. (Accessed: June 5 2020).*

[12] *Red Hat, Inc., "Python 3 Support", https://docs.ansible.com/ansible/latest/reference_appendices/python_3_support.html. (Accessed: June 5 2020).*

[13] *VyOS Wiki, "User Guide – Installation", https://wiki.vyos.net/wiki/User_Guide#Installation. (Accessed: June 7 2020).*

[14] *Red Hat, Inc., "package_facts – package information as facts", https://docs.ansible.com/ansible/latest/modules/package_facts_module.html. (Accessed: June 7 2020).*

[15] *Red Hat, Inc., "Conditionals – The When Statement", https://docs.ansible.com/ansible/latest/user_guide/playbooks_conditionals.html#the-when-statement. (Accessed: June 7 2020).*

[16] *Red Hat, Inc., "listen_ports_facts – Gather facts on processes listening on TCP and UDP ports", https://docs.ansible.com/ansible/latest/modules/listen_ports_facts_module.html. (Accessed: June 8 2020).*

[17] *Red Hat, Inc., "vyos_config – Manage VyOS configuration on remote device", https://docs.ansible.com/ansible/latest/modules/vyos_config_module.html. (Accessed: June 8 2020).*

[18] *Red Hat, Inc., "Ansible Network Examples", https://docs.ansible.com/ansible/latest/network/user_guide/network_best_practices_2.5.html#example-2-simplifying-playbooks-with-network-agnostic-modules. (Accessed: June 8 2020).*

[19] *Red Hat, Inc., "ping – Try to connect to host, verify a usable python and return 'pong' on success", https://docs.ansible.com/ansible/latest/modules/ping_module.html. (Accessed: June 9 2020).*

[20] *Red Hat, Inc., "How Network Automaton is Different – Multiple Communication Protocols", https://docs.ansible.com/ansible/latest/network/getting_started/network_differences.html#execution-on-the-control-node. (Accessed: June 9 2020).*

[21] *Red Hat, Inc., "Ansible Vault – Variable-level encryption", https://docs.ansible.com/ansible/latest/user_guide/vault.html#variable-level-encryption. (Accessed: June 9 2020).*