Automating Operations for WVU Information Security

Nicholas Hill

Abstract

A python program that automates the process of updating definitions of university owned networks in Tenable SecurityCenter.

Introduction

West Virginia University manages over 4000 individually defined subnetworks spread over 3 distinct campuses across the state. Vulnerability management is critical to ensure that university-owned assets are known so they can be mitigated by the groups that manage them. Information Security Services has integrated Tenable SecurityCenter to scan, manage and report security vulnerabilities across the network for various IT groups. As new buildings are appearing and old buildings are disappearing, networks are being redefined constantly. These changes need to be reflected in SecurityCenter so that the correct IT group can be notified and mitigate the vulnerabilities associated with their assets. This software automates the process of updating SecurityCenter so that assets can be quickly and accurately redefined.

Past Solutions

This task was originally handled by pulling data from SecurityCenter and Network Operations, and importing them into an Access database, where predefined queries could be used to manually observe changes in the system. This process would often take several days to manually check each asset, and often contained human error.

Initial Design

The initial approach was to create a script that would simply parse a CSV input file of the currently defined networks, place each network definition into a dictionary, and store each dictionary definition into a list. The script would then make API calls to SecurityCenter to pull the records that it has for each defined network. At this point, the structures could be iterated and compared to view the differences between the records, operating similarly to a relational database.

Design Improvement

The number of exceptions and differences in data format standards made the comparison difficult. Additionally, more data had to be collected to accurately define new assets and safely remove old assets.

Overall Architecture of Current System

The software was improved by rewriting the code into a full object-oriented program. This program contains the following components:

Config Parser

This class parses a configuration file for the software. The configuration file is intended to define variables, rules and values for items that may change over time. This also includes exceptions to various naming conventions, so that they're contained and managed in one central location. This was implemented using a lex-like technique, and allows for easy customization that was meant to resemble natural human language. The syntax is comparable to creating rules in UFW.

Internal Network Information Puller

This component contains functions used to pull the internal network information from network operations. This was created into its own component for future proofing.

SecurityCenter Data Puller

This component contains functions to pull data from SecurityCenter. This was created as its own component for reasons similar to the class stated above. In the event that SecurityCenter changes its API, the only code that would need to be rewritten would theoretically be contained within this class.

Data Manager

This class is meant to be initialized as an object and is responsible for storing the data pulled from both SecurityCenter and Network Operations. This class also contains methods to pull all of the relevant information needed to complete the comparison through the functions defined in the SecurityCenter data puller and network information puller components.

Comparator

This component contains the functions used for comparing the data and creating templates of changes that need to be made to SecurityCenter. This will find issues with capitalization, bad names, bad IP addresses, duplicates, and missing records.

SecurityCenter API Wrapper
As a SecurityCenter administrator, I have found several small bugs and inconveniences with the system. This API wrapper remediates those issues to ensure that the updates posted to the SecurityCenter don't break or disrupt any running scans or corrupt any data.

IP Range Comparator
This class provides a way to compare IP ranges of several different notations and provides methods to determine if ranges overlap, are contained, or are equal. It accepts strings of single IP addresses, ranges defined with a dash (i.e. 192.168.2.0-192.168.2.8) and ranges defined by the CIDR notation used in SecurityCenter. The development of this class was a major component to the success of this software, as well as several other scripts used for operations within Information Security and other IT infrastructure groups.

The Algorithm in Detail

This seemingly trivial task has taken over a year of development. This section is intended to describe the detail required to automate this task:

1. **Parse the Configuration File.** This uses a lex-like algorithm to parse various rules, exceptions, and parameters for use in the script such as hostname, records to ignore, and custom IT manager mappings.

2. **Download the data from SecurityCenter.** This was done easily with an existing Python SecurityCenter API wrapper available on Github.

3. **Download the data from Network Operations.** This step actually required leveraging a vulnerability to obtain the data. (Their system is still protected externally)

4. **Convert the SecurityCenter data into comparable tables.** This is one of the most computationally intensive parts of this software. This can be broken down into several substeps:
    a. The format of the data from network operations is to provide the network IP address as one field, the network subnet mask in CIDR notation as one field, the IT manager responsible for the network as one field, and it's description as one field. The assets in SecurityCenter are named by the description in the network operations data. There are often several networks under the same description. SecurityCenter doesn't allow assets

under the same name, so all of the records under the same description need to be combined into one record.

b. Some of the networks defined by in network operations data under the same description contain both public and private addresses. These are also handled differently in our configuration of SecurityCenter. Both the public and private ranges need to be separated into two distinct records, with the description of the private record ending with the suffix “(private)”.
c. SecurityCenter accepts IP ranges given in CIDR notation, but it parses these definitions as ranges rather than networks. For example, 192.168.0.23/24 does not define a machine 192.168.0.23 in the subnet 255.255.255.0, but as the IP range of 192.168.0.23-192.168.1.23. Additionally, SecurityCenter automatically merges adjacent IP ranges, so a list of IP ranges [192.168.0.0/24, 192.168.1.0/25] would be automatically converted to 192.168.0.0-192.168.1.127. Because of this, the IP ranges cannot be compared by string values, but they must actually be parsed and understood as a range by the software. Due to the unusual convention, A class was implemented to define an IP range. This is described in depth in the components section. Each record for both the network operations data and the SecurityCenter data uses this class to define their IP ranges.

5. Compare the tables. There are 5 types of checks that are completed: check for bad names, check for bad IP ranges, check for bad groups, check if the range exists in SecurityCenter, and check if the range is defined by Network Operations.

6. Stage the changes for SecurityCenter. As stated earlier, SecurityCenter has a couple of small bugs that occur during various operations. These operations must be completed using specific synchronous procedures.

7. Change bad asset names. This is an easy process. This just uses a simple PATCH request with the changed values in json.

8. Change bad asset IP ranges. This is also a simple PATCH request with the changed values in json.

7. Add new assets. This process is more complex because it not only requires creating the asset in SecurityCenter, but also adding it to the appropriate group and assigning it to a scheduled scan for that group. When the data manager gets initialized, it pulls information for scans, groups and users currently in the system. The user can be guessed by looking for matching names of registered SecurityCenter users to the manager of the IP range defined in the data from Network Operations. Once this information is found, a POST request with the relevant data can be sent. The asset will then be added to this group. The scan can be found by filtering out the list of scans held by the data manager to only the ones under the group of the newly assigned asset. From there, the unscheduled scans are filtered out, and then a regular expression is applied to find the appropriate scan based on the characteristics of the IP range. The IP range will either be public or private, and the scan names can be searched based on whether the scan is for public or private IPs, based on the convention implemented to our specific system.

8. Delete the old assets. This is another process that isn't very straightforward. If an asset gets deleted before being removed from a scan, the scan will still contain the deleted asset and error out if the scan starts. This is an issue, as these scans happen only so often. To ensure that assets are deleted without breaking scans, each scan must be iterated through to determine what scans include the asset that is about to be deleted. This data is stored within the data manager as well. Once this list is constructed, the asset must be removed from each scan. This is done with a PATCH request as well, but this time it contains data that needs to be retained. Each ID of every asset except for the one that needs to be removed must be sent in the request. Once the asset has been removed from every scan, it can be safely deleted.

Conclusion

This document provides an in-depth description of a solution that was implemented to automate the operations associated with managing SecurityCenter at an enterprise level. This was intended to show the complexity and development skills required to implement a seemingly trivial task with several edge cases and exceptions.