569 Final Project Writeup

Elaine Cole, Griffin Shaw, Jarek Millburg, Kaiye Yu, Ighor Tavares

General Introduction

For this project, we designed an application that is capable of monitoring and hardening a given network against Mirai-like botnets. Our final product stands out in that it detects vulnerable IoT devices within a network and alerts the user in real-time and also is run on a single device within the given network but capable of securing multiple IoT devices within the same network.

We chose to focus on Mirai and how it is attacks IoT devices due to the publicity of it and its permeability—now that the source code of Mirai has been published, many adaptations of it have come online, and we wanted to develop a practical tool that would address real-world security concerns.

Mirai is a malware botnet that took advantage of insecure IoT devices in a simple but clever way. Mirai would scan networks for open Telnet/SSH ports, and then attempts to log in using default credentials. It's surprising how many users do not change the default password of devices, and our tool serves as a mitigation to this form of attack.

Because of this, we chose to focus on brute force login attempts and thus refer to an IoT device as vulnerable if it uses default login credentials.

Video Demo

A video demo of our IoT Network Canary working can be found here.

IoT Network Canary Overview

Our final project consists of two major components: one serves as a honeypot and notification system, and the other serves as a network security hardener. We explain the details of each component below, and further detail per file how our product works in a later section.

Honeypot & Notification Tool:

Our honeypot sits passively on the network and appears vulnerable to those that try to SSH into it. It listens for brute force connection attempts and tracks them in our own specified auth_log file. Then, if the number of attempts for a given IP address exceeds a user-set heuristic, an alert email is sent to the user's specified email address.

Network Security Hardener:

Our network hardener seeks, on a specified network, devices with open ports that are not defended by non-default credentials and then offers to the user to change the default credentials to something more secure. It first scans for devices with open ports on the given network. Once open ports are detected, we use Ncrack, a network authentication cracking tool, to try and make a connection through the open port. If we are successful, then the user is given the option via command line to change the weak credentials, thereby "hardening" the network security.

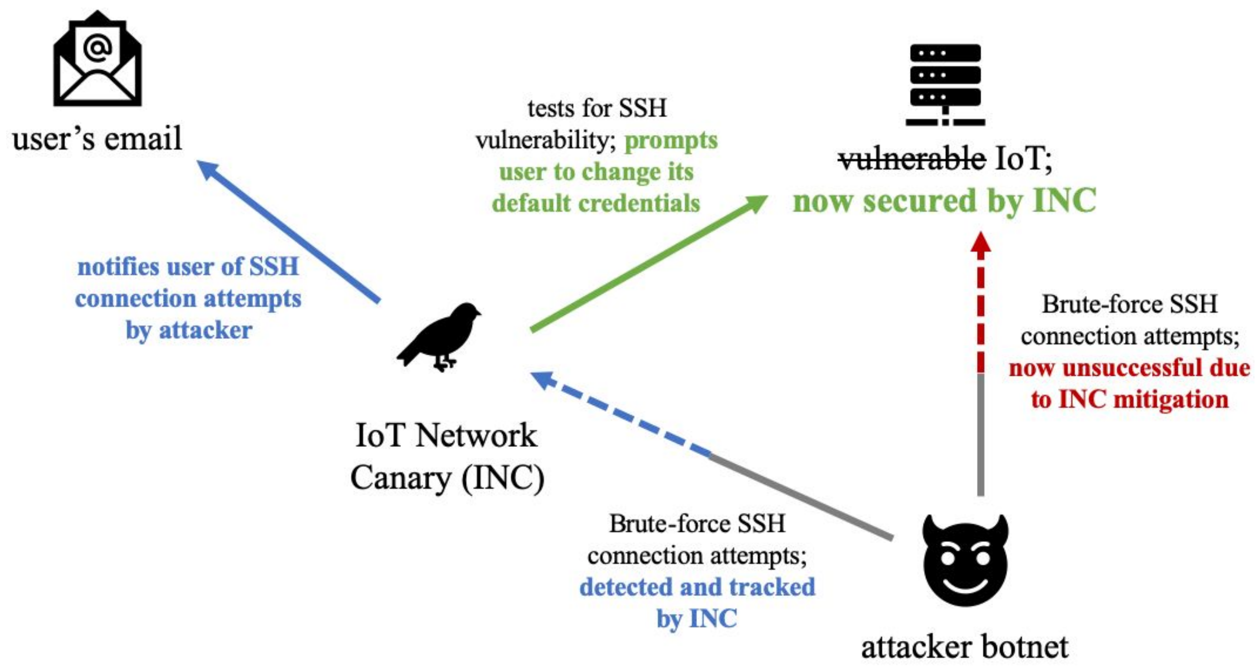
To properly identify vulnerable devices in the network, first it was required to mapped out all the devices currently live in the network. This was accomplished by running a pingless Nmap scan to treat the range of IP as alive and only report the live devices the ones in which responds back. Once the scan is over, the Iot Network Canary (INC) will update its valid ARP table, and collect all the live devices in the network along with its MAC addresses, and create a list to perform a port scan on it. Nmap is again used to identify all devices which contains port 22 open (SSH protocol).

To simulate a similar brute force attack on the devices found on the network to have open ports like the Mirai Botnet, we utilize a tool developed by Nmap called Ncrack. This tool allows us to attempt brute force connection attempts to many different types of ports on multiple machines with one simple command. Ncrack also comes with its own library of default credentials to use to attempt to access these ports similar to the database of default credentials that Mirai uses when it attempts to connect to your devices. With Ncrack and its library of credentials we can check to see if any of the devices on the network with open ports are still using default credentials. If our attempt with Ncrack is successful then good chance the Mirai botnet would also crack it.

The process of prompting the user to change the vulnerable password of the devices, was accomplished using a shell library called *expect*. It simply "expects" a prompt in which sends the input automatically. IoT Network Canary first asks the user if they would like to change the password of vulnerable devices. Then the *expect* script, automatically connects to the vulnerable device through an ssh session with the cracked credentials, and it changes the password. Once done, the script automatically closes the ssh connection. During the whole process, all the output is redirected to */dev/null* so no log is maintained.

Workflow Diagram

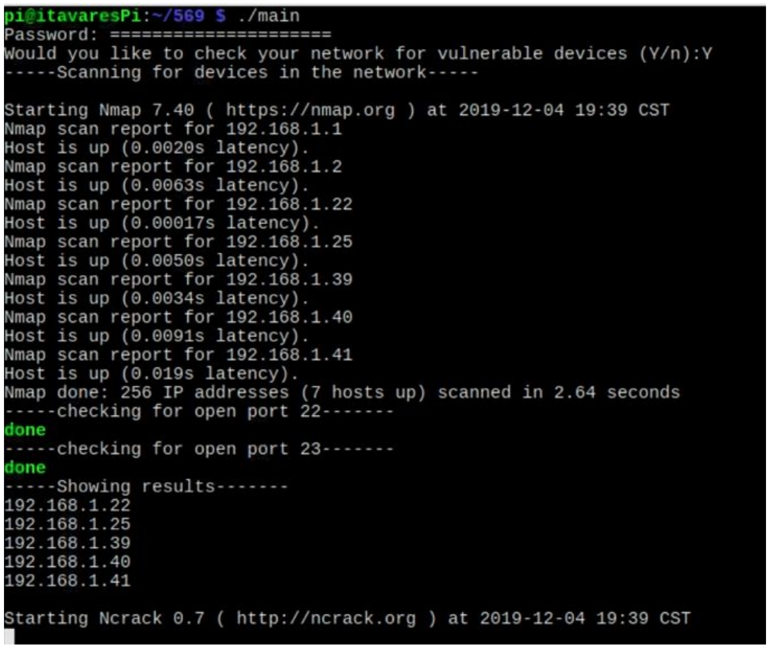
Below our diagram outlines the workflow of our IoT Network Canary (INC):


user's email
tests for SSH vulnerability; prompts user to change its default credentials
~~vulnerable~~ IoT; now secured by INC
notifies user of SSH connection attempts by attacker
Brute-force SSH connection attempts; now unsuccessful due to INC mitigation
IoT Network Canary (INC)
Brute-force SSH connection attempts; detected and tracked by INC
attacker botnet

Usage Instructions

```
./main
```

First, run the *main* script which then it will prompt to input the valid password related to the email that will be notified. Once the valid password is given, it will ask the user if they would like to check for any vulnerable devices in the network. If the user says "no", the script will run the honeypot monitoring script.

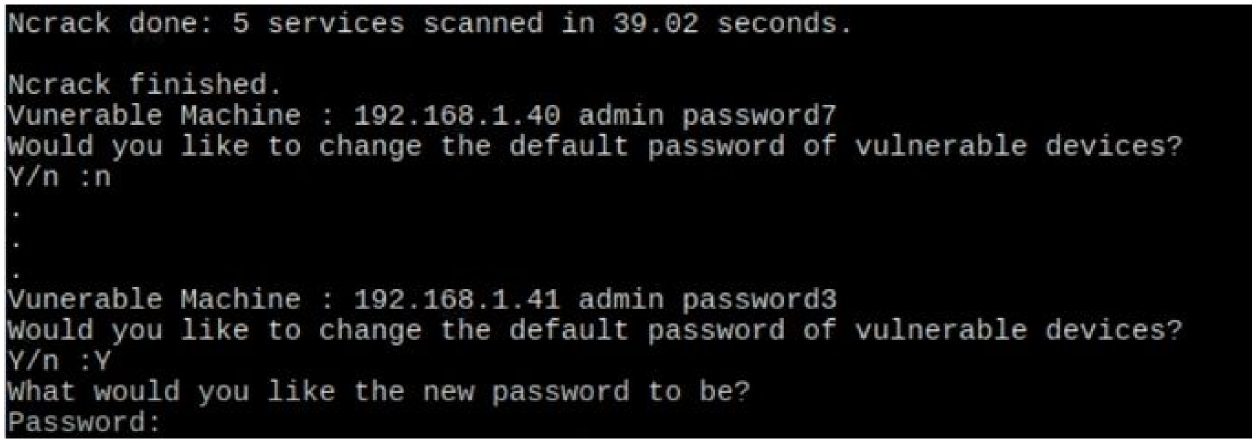


```
pi@itavaresPi:~/569 $ ./main
Password: ====================
Would you like to check your network for vulnerable devices (Y/n):Y
-----Scanning for devices in the network-----

Starting Nmap 7.40 ( https://nmap.org ) at 2019-12-04 19:39 CST
Nmap scan report for 192.168.1.1
Host is up (0.0020s latency).
Nmap scan report for 192.168.1.2
Host is up (0.0063s latency).
Nmap scan report for 192.168.1.22
Host is up (0.00017s latency).
Nmap scan report for 192.168.1.25
Host is up (0.0050s latency).
Nmap scan report for 192.168.1.39
Host is up (0.0034s latency).
Nmap scan report for 192.168.1.40
Host is up (0.0091s latency).
Nmap scan report for 192.168.1.41
Host is up (0.019s latency).
Nmap done: 256 IP addresses (7 hosts up) scanned in 2.64 seconds
-----checking for open port 22-------
done
-----checking for open port 23-------
done
-----Showing results-------
192.168.1.22
192.168.1.25
192.168.1.39
192.168.1.40
192.168.1.41

Starting Ncrack 0.7 ( http://ncrack.org ) at 2019-12-04 19:39 CST
```

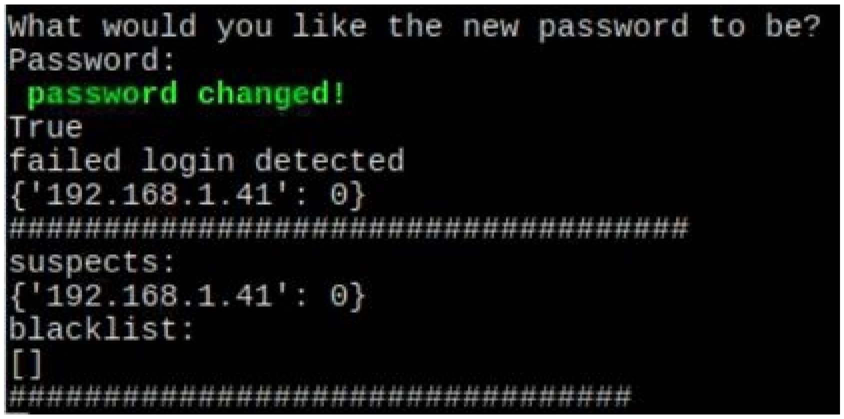
If user says “yes” , the script will proceed to identify which devices contains open ssh port, and then run Ncrack to finally identify if any devices are vulnerable to default credentials.



```
Ncrack done: 5 services scanned in 39.02 seconds.

Ncrack finished.
Vunerable Machine : 192.168.1.40 admin password7
Would you like to change the default password of vulnerable devices?
Y/n :n
.
.
.
Vunerable Machine : 192.168.1.41 admin password3
Would you like to change the default password of vulnerable devices?
Y/n :Y
What would you like the new password to be?
Password:
```

Once Ncrack is finished, the script will once again prompt the user if they want to change the default password of the vulnerable devices. If the user chooses to change the password, they will be asked to input a new secure password.



```
What would you like the new password to be?
Password:
 password changed!
True
failed login detected
{'192.168.1.41': 0}
####################################
suspects:
{'192.168.1.41': 0}
blacklist:
[]
#################################
```

The script will then automatically change the password of the vulnerable device and notice the user that it was successfully changed. Once finished, the monitoring script will start.

Code Layout

Our repository is located here. Below we outline, per file, the specific code functionality.

main

Our main script starts out by checking to see if the Ncrack program is currently installed on the system you are running on as it is required in order to brute force the open ports on potentially vulnerable devices on the network.

Next we take in a password from the user that is the password for the email account that is used later on the connect to the email server to send out the alert to the user that they are under attack. Following this, we prompt the user for permission to check the network for vulnerable devices.

If the user agrees to check for vulnerable devices our program then proceeds to call the first script called *whiteWorm.sh* which when finished will create a text file listing all devices with open SSH ports. This file will then be used as the input file of targets for our Ncrack command which will attempt to brute force the SSH port on the devices output all results to another text file. For the purpose of our demonstration we also utilize a *username.txt* & *password.txt* file when running Ncrack. This takes in a list of usernames and passwords we will use in our efforts to find out the credentials of the devices. We do this to shorten the time it will take to run Ncrack and show that it is possible to find the credentials and show how we can read the output to find the IPs, username, and password for every device we access. If we do not use these files then the program will use its default credential library instead. We follow this up by reading the output file and grabbing every line that has "ssh:" which will only appear on lines where we have found the credentials for a target. We cut and trim these lines in order to grab only the target IP, username, and password and append these results to another text file to be used in the next step. This is done with the following line: `cat output.txt | grep "ssh:" | cut -d " " -f1,4,5 | tr =d "'" > crackedCreds.txt.`

If we are able to find vulnerable devices, then we will proceed to call our *secureCreds.sh* script which will read in our final output file containing the IP, username, and password for cracked devices and prompt the user to change the credentials. If we are unable to crack any of the devices, then we will proceed to let them know that we found no vulnerable devices, and continue to clean up our directory by removing the files we created in order to store any potential information about the devices. Afterwards, we run our honey pot to start listening for connection attempts and brute force attacks on our network.

whiteWorm.sh

The idea of creating the whiteWorm script was to reproduce a similar attack of the Mirai botnet. The main goal of whiteWord is to identify all the devices in a network, and check which ones are vulnerable to a potential ssh/telnet brute force. Our script at the moment, checks for open SSH ports (22). First, the script will run an pingless Nmap scan, so we can treat as all the ip addresses alive in the network and identify only the ones which responds back. Once it has a list of online devices, it gets its from a simple arp scan, in which then, it performs another Nmap scan, however this time, checking for open SSH ports. This is possible with the following line: `nmap -PN -p 22 --open -oG - ${getip}.0-$max_ip | awk '$NF~/ssh/{print $2}'`. This line will only print out devices with open SSH ports. Once the script finishes, we will have a list of devices with open SSH ports.

emailTools.py

This python script uses the SMTP protocol client library `smtplib` to send the user an email alerting them that there have been too many brute force connection attempts.

First, we initialize a server to send an email through with `initServer()`. This takes in a password for our email account which we send the email from, sudormrf569@gmail.com, and connects to the smtp.gmail.com server.

Next, we call our function `sendEmail()`. This takes in the offending IP address which was attempting to connect via brute force, and we send an email to our defined receiving email addresses to notify them.

To use this as a proof of concept, we have hardcoded the email addresses that we send the alerts to. Further functionality would involve an easier way to set the receiving email address, and adjusting the email message content that we send.

secureCreds.sh and ssh_change

These scripts are responsible for changing default credentials on vulnerable devices in the network.

The first script, *secureCreds.sh*, goes through a list created the *main* script, after performing the brute-force attack, which contains the IP addresses of vulnerable devices that we have identified. Then, it asks the user if they would like to change the password of these vulnerable devices. It is key to make the new password anonymous. When prompting the user for a new password, we used special flags on the command read to maintain the new input secret: `read -sp 'Password: ' passvar;`.

The second script, *ssh_change*, is written using the `expect` shell libraries with the purpose of automating the process. The script first takes four parameters to do this:

```
set username    [lindex $argv 0];
set oldpass     [lindex $argv 1];
set newpass     [lindex $argv 2];
set host        [lindex $argv 3];
```

Then, the script connects to the device through SSH connection with their vulnerable credentials that we know from our implementation in *main* via: `spawn ssh -l $username $host`, and once it is connected, it changes the password with the new password given by the user: `expect "Enter new expert password (again):" ;  send "$newpass\r".` An important step in this script is, at the end of it, to ensure that it exits the current SSH connection to avoid any unwanted live sessions: `expect eof" ; expect "$ "; send "exit\n".`To test this script functionality, we simply tried to connect to the vulnerable device with the bad credentials before and then after the script ran.

ssh_check.py

In order to check if there is any brute force SSH attempt, the easiest way is to go to `/etc/auth.log` (our honeypot is implemented on a Raspberry Pi 3) and check the history of SSH recorded in it. However, such a method would require root privilege of the pi, which violates the least privilege policy. Thus, we decided to record all the logs into another log file that the user creates. In our case, it is called "`myauth.log`", so we do not need to track back to the /etc folder with root privilege.

ssh_check.py is the file that check the brutal force attempt by reading from `myauth.log`. It reads the last line in the `myauth.log` every 0.1 second. If the program finds word "Failed" in the last line of myauth.log, it means there is a failed ssh login attempt. In this case, we use regular expressions to find the IP address of the suspicious behavior and add it to the suspicious list. Along with each IP address, we also record the number of failed SSH login attempts. When the number reaches some threshold (e.g. 10 times of failed login), our program will add this certain IP address to our blacklist. Our original plan was to block such an IP address from our honeypot. However, to do that, we would need to write the IP address we want to block to the `/etc/hosts.deny` file, which also requires root privilege to write to. Because of the least privilege policy, we decided to remove this functionality. Instead, we decide to write an automatic email program (*emailtool.py*) that will send an email to target email address to warn the user about the IP address.