Cowrie Honeypot Configuration Raspberry Pi 5

System: Raspberry Pi 5 (Debian-based)
Client Device: Macbook Pro 2021
Environment: Python 3.11.2, Cowrie 2.6.1, Twisted 25.5.0

Setup

- Directory: /opt/cowrie
- Virtual Environment: /opt.cowrie/cowrie-env
- Main Configuration Files:

  /opt/cowrie/etc/cowrie.cfg.dist (default)

  /opt/cowrie/etc/cowrie.cfg.local (overrides)

  /opt/cowrie/etc/cowrie.cfg (optional, reads if present)

Commands

- cd /opt/cowrie
- bin/cowrie start -n (start cowrie no daemon)
- bin/cowrie stop (stop cowrie)
- ssh fakeuser@<cowrie device ip> -p 2222 (Cowrie by default listens on port 2222) This is the command to ssh as an intruder into Cowrie machine(pi) on said port through another machine (in my case I used a Macbook Pro).

SSH Authentication Configuration

Cowrie allows us to simulate a login using a fake authentication, within cowrie.cfg.local I used one authentication class, then attempted another two as well:

- *HoneyPotAuthRandom*

Authentication Method: Will randomly allow login after a random number of failed attempts into a fake shell. This allows us to simulate a more realistic expereince. Login attempts are produced on the host machine and any commands attempted within the fake shell are produced as well.

Usage: We mimicked a random ssh configuration where attackers are able to grain access sometimes after using multiple attempts.

- *UserDB* (In progress)

  Authentication Method: Allows login with username password combination which are available in userdb.txt. I did like this method as it does still allow for some form of concealment and uncertainty.

  Outcome: I tried to configure this but was unable to up to this point during testing. Cowries ssh prompt only asked for a password and never ended up requesting the usernmae password combination although I configured it this way in the file. Therefore, unable to simulate a successful login using UserDB. Ive come to the conclusion thus far that it is most likely an issue in the formatting or a structural issue within the configuration. I will be looking to set this up in the future though.

- *HoneyPotAuthAlways* (In progress)

  Authentication Method: This allows for any username password combo to be accepted on the first try. The login prompt only gave us a single password field so our testing didnt behave the way that worked for HoneyPotAuthAlways. I also do believe that this method wasnt beneficial or wasnt too clever in terms of uncertainty and concealment, but can allow us to simulate brute force success after a fixed number of attempts.

  Outcome: Despite this I do think setting HoneyPotAuthAlways and UserDB up in the future will be beneficial for testing purposes and a more rounded understanding of more authentication methods.

Troubleshooting

- *UserDB Login Fail*

  Problem: Cowrie would not prompt the client machine(macbook pro) for bothnusername and password when trying to ssh into the system(pi). It would only ask for a password.

  Cause: UserDB requires full credential handling, it appears that this may not be supported by our specific type of SSH session's behavior. The userdb.txt file possibly being misconfigured could have also contributed to the issue.
  Solution: We switched to use HoneyPotAuthRandom rather than UserDB which allows access after a few failed attempts regardless of any sort of credentials.(Other than the original ssh into the correct ip and port).
  Takeaway: Cowrie's simulated SSH login may not support username password validation unless configured in a specific way and having the correct permission to do so. IT could also have something to do with the system machine (pi) and the way it allows for client devices to ssh into it. The client machine may also have contributed to the problem. Looking further into this to try and get to the root cause and get it to work.

- *Password Only Prompt*

  Problem: SSH would only ask for a password even when a username was passed, and or userdb was attempted to be configured.(ssh fakeuser@ip -p #)

  Cause: This could be from Cowrie handling username internally and not exposiong multi step login prompts by default.

  Takeaway: I wasnot able to change this or move forward from the issue so I worked around it by using HoneyPotAuthRandom rather than UserDB or HoneyPotAuthAny for my authentication.

- *Configuration Not Applying*

  Problem: The changes I made to cowrie.cfg.local wouldnt always work or reflect when testing.

  Cause: This was because Cowrie may have deafaulyed to cowrie.cfg.dist or the file itself had syntax errors.

Solution: I was able to resolve this by verifying the file location along with the syntax and restarting and testing Cowrie after each change I made.
Restarting Cowrie: bin/cowrie stop && bin/cowrie start -n
Confirming the configuration is being loaded using: grep "Reading configuration from" cowrie.log

- *File Permission Errors*

Problem: Cowrie couldnt access logs or environment files.

Cause: Some of the files were owned by root.

Solution: I used: sudo chown -R $USER:$USER /opt/cowrie to make the lkog directories and virtual environments user writable.

Key Takeaways & Insight

- I have an understanding of how Cowrie simulates a real SSH server and produces logs of attacker activity using fake shell sessions.
- Learned how different authentication classe affect attacker behavior based on realism, this includes:

    HoneyPotAuthRandom being used to simulate an unmforeseen brute force entrance.

    UserDB to potentially use in a credential based simulation(in progress)

    HoneyPotAuthAlways for consistent bruite force success simulation(in progress)

- I ran into the issue of the login prompt not necessarily being set up for a username and password so troubleshooting through that I have found a few potential causes including the ones stated earlier in troubleshooting that I can keep note of as I move forward using Cowrie.
- Cowire can behave differently depending on the resources the device you are using has available(Pi vs VM) and that some features including ful command support or fake shell persistence depend on a deeper configuration.

- This project underlined the importance of validating confiug file loading and file permissions in the Cowrie environment.

Improvement and In Progress

. Future Improvements & In-Progress Tasks
Re-attempt configuration and testing of UserDB authentication to allow specific login credentials, once proper formatting and login handling are confirmed.

Revisit HoneyPotAuthAlways to simulate fixed-pattern brute-force success and evaluate how Cowrie logs these scenarios.

Investigate and expand Cowrie's shell emulation and fake file system (honeyfs) so attackers can interact longer with a realistic environment.

Improve persistence of fake shell sessions after login to simulate more realistic post-authentication behavior.

Continue exploring how Cowrie's internal handling of usernames impacts which auth classes can be used.

Forward logs to a centralized system (e.g., ELK, Splunk) for deeper analysis.

Add simulated commands (ls, cat, wget, etc.) to enrich attacker interaction.

Test blocking or alerting based on IPs after repeated logins to simulate detection or defense mechanisms.

Explore Cowrie deployment in a cloud environment or segmented network for more advanced honeypot experimentation.