ASSIGNMENT 6

TRUSTWORTHINESS

Neilos Kotsiopoulos

neko@itu.dk

1) Problem 1: Basis of Trust

You are a software consumer. You purchase a binary B from Amasoft Inc., and run it

Part 1: a) Fill in hB, _, _i. b) Why is this trust assumption (un)reasonable?

<B, ax, AmaSoft>

- A (artifact): B (the binary I purchased)
- B (Basis of Trust): ax (axiomatic – trust is based on an assumption)
- W (either an entity or another artifact): AmaSoft Inc (from where I purchased the binary)

This trust assumption could be considered reasonable if Amasoft Inc. has a reputable track record, if the binary has been well-reviewed by others, or if you've had positive experiences with their products before. It might be considered unreasonable if you have no previous experience with Amasoft Inc. or if there is no publicly available evidence to suggest that their software is reliable or safe

Suppose you also received a test suite T. Before you run B, you read T yourself to check input coverage & tested properties, and run T to confirm that B passes all tests

Part 2: a) hB, _, _i, hT, _, _i. b) Why are these assumption (un)reasonable?

<B, an, T> , <T, ax, AmaSoft>

- A (artifact): B (the binary I purchased), T (the trust Suite)
- B (Basis of Trust): an (analytical – trust is based on my own analysis), ax (axiomatic – trust is based on an assumption)
- W (either an entity or another artifact): T (the test suite, for the binary B), AmaSoft Inc (from where I purchased the binary)

The trust assumption in the binary B becomes more reasonable because it is based on my own analysis of the test suite. Therefore, it is not simply an axiomatic trust in Amasoft Inc., but is supplemented by your analytical investigation

Suppose that instead of T, you receive a proof object P. You read P yourself, and check that P proves correctness of B using a third-party verifier program V

Part 3: a) Propose trust triples. b) Why are these assumption (un)reasonable?

<B, an, P> , <P, ax, AmaSoft>, <V, ax, TP>

- A (artifact): B (the binary), P (the proof object), V (the verifier program)
- B (basis of trust): an (analytical, for the binary B), ax (axiomatic, for both the proof object P and the verifier program V)
- W (either an entity or another artifact): P (the proof object, for the binary B), Amasoft Inc. (the provider of the proof object, for P), TP (Third-party entity - the creator of the verifier program, for V)

The trust assumption in the binary B becomes more reasonable as it's now based on your own analysis of the proof object P and the results from the verifier program V.
The trust assumption in the proof object P might be seen as reasonable if Amasoft Inc. has a good track record or is known for providing valid proof objects.
The trust assumption in the verifier program V might be reasonable if the third-party entity is reputable and known for creating reliable verifier programs. However, it could also be seen as potentially unreasonable if the third-party entity isn't trustworthy, or if the verifier program itself has not been sufficiently vetted

Suppose you read V , to verify that V is correct

Part 2: a) hB, _, _i, hT, _, _i. b) Why are these assumption (un)reasonable?

<B, an, P> , <P, ax, AmaSoft>, <V,an, me>

- A (artifact): B (the binary), P (the proof object), V (the verifier program)
- B (Basis of Trust): an (analytical, for both the binary B and the verifier program V), ax (axiomatic, for the proof object P)
- W (either an entity or another artifact): P (the proof object, for the binary B), Amasoft Inc. (the provider of the proof object, for P), me (as verified the verifier program V)

The trust assumption in the binary B remains reasonable as it's based on your own analysis of the proof object P and the verified program V.
The trust assumption in the proof object P continues to be based on the reputation of Amasoft Inc.
The trust assumption in the verifier program V is now more reasonable since I have personally analyzed and validated it. This reduces the dependency on the third-party entity's trustworthiness

Suppose, instead of relying on a proof, you run B0—the result of rewriting B using a third-party program rewriter R (e.g. a monitor).

Part 5: a) Propose trust triples. b) Why are these assumption (un)reasonable?

<B’, sy, R> , <R, ax, TP>

- A (artifact): B’ (the new binary), R (the rewriter program)
- B (Basis of Trust): sy (synthesized, for the new binary B’), ax (axiomatic, for the rewriter program R)
- W (either an entity or another artifact): R (the rewriter program, for the new binary B’), TP / Third-party entity (the creator of the rewriter program, for R)

The trust assumption in the new binary B’ is based on the rewriter program R that was used to create it. This could be seen as reasonable if the third-party program rewriter R is reliable and known to produce secure and functional binaries.
The trust assumption in the rewriter program R is axiomatic and depends on the trustworthiness of the third-party entity that created it.

You are now instead managing a team at AmaSoft, Inc., that you task with writing a module with security goals, for a larger software product. Amy & Bob are on your team. Amy writes a crypto module M (using standard library only). You compile the module to binary B, and intend to use B.

Part 6: a) Propose trust triples. b) Why are these assumption (un)reasonable?

<B, sy, M> , <M, ax, Amy>,

- A (artifact): B (the binary), M (the module)
- B (Basis of Trust): sy (synthesized, for the binary B), ax (axiomatic, for the module M)
- W (either an entity or another artifact): M (the module, for the binary B), Amy (the author of the module, for M)

The trust assumption in the binary B is reasonable because it's synthesized from the module M which Amy wrote.
The trust assumption in the module M is axiomatic, based on my faith in Amy's skills and integrity. The use of the standard library only can also be seen as reasonable as it limits the attack surface by using well-vetted and commonly used libraries.

You ask Bob to review M; the code passes Bob’s review

Part 7: a) Propose trust triples. b) Why are these assumption (un)reasonable?

<B, sy, M> , <M, an, Bob>

- A (artifact): B (the binary), M (the module)
- B (Basis of Trust): sy (synthesized, for the binary B), an (analytical, for the module M)
- W (either an entity or another artifact): M (the module, for the binary B), Bob (the reviewer, for M)

The trust assumption in the binary B remains the same and is still reasonable as it's synthesized from the module M, which now has been reviewed by Bob.
The trust assumption in the module M changes from axiomatic (based on Amy) to analytical, now based on Bob's review. This makes the trust assumption more reasonable because Bob provides an additional layer of scrutiny, validation, and assurance for the code's reliability and security – given that Bob can provide those, otherwise it becomes unreasonable.

You ask Amy & Bob to write a test suite T. Before you use B, you read T yourself to check input coverage & tested properties, and run T to confirm that B passes all tests.

Part 8: a) Propose trust triples. b) Why are these assumption (un)reasonable?

<B, sy, T> , <T, an, me>, <M, an, Bob>

- A (artifact): B (the binary), T (the test suite), M (the module)
- B (Basis of Trust): sy (synthesized, for the binary B), an (analytical, for the test suite T), an (analytical, for the module M)
- W (either an entity or another artifact): T (the test suite, for the binary B), me (the reviewer, for T), Bob (the reviewer, for M)

The trust assumption in the binary B changes to being based on the test suite T, which makes it more reasonable. The binary B is synthesized from module M and confirmed by tests T. This gives a higher degree of assurance because the binary has been vetted through a detailed set of tests.
The trust assumption in the test suite T is based on my own analytical trust, but also reducing reliance on external entities.
The trust assumption in the module M remains the same and is based on Bob's review.

You ask Amy to instead use commons-crypto, an open-source crypto-library. You read M to confirm that M indeed is just a thin wrapper around the library.

Part 9: a) Propose trust triples. b) Why are these assumption (un)reasonable?

<M, an, me> , <CC, ax, OSC>, <B, sy, M>

- A (artifact): M (the module), CC (commons-crypto / the crypto library), B (the binary)
- B (Basis of Trust): an (analytical, for the module M), ax (axiomatic, for the crypto library commons-crypto), sy (synthesized, for the binary B)
- W (either an entity or another artifact): me (the reviewer, for M), Open Source Community - OSC (the creator of the crypto library, for commons-crypto), M (the module, for the binary B)

The trust assumption in the module M is based on my own analytical trust. This is reasonable because you reviewed the module myself and confirmed it is a thin wrapper around the library.
The trust assumption in the crypto library commons-crypto is axiomatic and depends on the Open Source Community. This assumption could be seen as reasonable if the library is well-maintained, widely used, and has been subject to rigorous peer review.
The trust assumption in the binary B remains the same and is still reasonable as it's synthesized from the module M. However, now the module M includes the trusted open-source crypto-library commons-crypto, potentially increasing the level of trust.

Amy re-implements M in Paragon. You check the security policy embedded in M, and confirm that the Paragon compiler successfully compiles M.

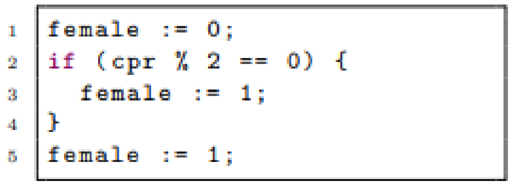
Part 10: a) Propose trust triples. b) Why are these assumption (un)reasonable?

<M, an, me> , <P, ax, Comp>, <B, sy, M>

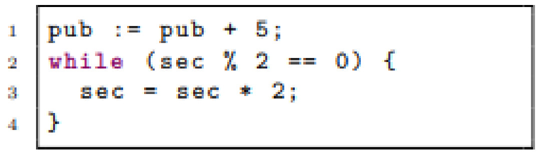
- A (artifact): M (the module), P - Paragon compiler (the compiler), B (the binary)
- B (Basis of Trust): an (analytical, for the module M), ax (axiomatic, for the Paragon compiler), sy (synthesized, for the binary B)
- W (either an entity or another artifact): me (the reviewer, for M), Comp - Compiler Developer (the creator of the compiler, for Paragon compiler), M (the module, for the binary B)

The trust assumption in the Paragon compiler is axiomatic and depends on the Compiler Developer. This assumption can be reasonable if the compiler is widely used, well-maintained, and has been subjected to rigorous review.
The trust assumption in the binary B remains the same and is still reasonable as it's synthesized from the module M. However, this time, the module M includes the security policies defined in Paragon, which could potentially increase the level of trust, assuming that the policies are sound and appropriately cover the module's functionality.

2) Problem 2: IFC



```
female := 0;
if (cpr % 2 == 0) {
  female := 1;
}
female := 1;
```



```
pub := pub + 5;
while (sec % 2 == 0) {
  sec = sec * 2;
}
```

First, consider the program on the left. Suppose female is labeled L and cpr labeled H.

Part 1: This program is information-flow secure. Explain why.

The program is information-flow secure because regardless of the 'cpr' value, 'female' is always set to 1, making 'female' (L) independent of 'cpr' (H). This ensures no sensitive information is leaked from the high-level 'cpr' variable to the low-level 'female' variable.

Part 2: Does check accept this program? Justify your answer.

No, check will not accept this program. This is because it directly assigns a high confidentiality level variable ('cpr') to a lower one ('female') – through cpr % 2 female , which violates Denning's rules of Simple Security Condition.

Part 3: What happens if we run this program monitored with cpr = 2106905060?

The monitor will terminate the program. This is because 'cpr' (H) is an even number (2106905060) influences 'female'(L), which is a violation of the policy that high confidentiality level variables cannot influence low confidentiality level variables.

Part 4: What happens if we run this program monitored with cpr = 2106905065?

The program does not violate the policy of noninterference between high and low confidentiality levels. This is because 'cpr' (H) is an odd number (2106905065) influences 'female'(L), the 'if' statement returns false. So, the monitor will not terminate the program.

Part 5: What would an L-observer have to observe (besides input and output memories), for this program to not be information-flow secure?

The observer would have to have additional access or insight into the program's control flow of information. Specifically, the observer would need to know whether or not the 'if' condition evaluating whether the 'cpr' is even and executed, as this would indirectly reveal information about the high-level 'cpr' variable, violating the noninterference principle.

We now consider the program on the right. Suppose pub is labeled L and sec labeled H.

Part 6: This program is information-flow secure. Explain why.

The program is information-flow secure because the low-level variable 'pub'(L) is updated independently of the high-level variable 'sec'(H). Therefore, changes in 'sec'(H) cannot be inferred from observing 'pub', ensuring the noninterference principle is upheld.

Part 7: Does check accept this program? Justify your answer.

Yes, the check accepts this program. This is because the 'sec'(H) variable does not influence the variable 'pub'(L), so no information leakage or security violations observed. The operations on each variable are independent.

Part 8: What happens if we run this program monitored?

It will not terminate the program because it doesn't violate any information flow rules. The variable 'sec'(H) and 'pub'(L) are manipulated independently of each other.

Part 9: What would an L-observer have to observe (besides input and output memories), for this program to not be information-flow secure?

An L-observer could observe the execution time of the program. Higher execution time might mean that the 'sec'(H) variable is even and the loop is executed, which is a side channel that leaks high-level information to the L-observer, compromising the information-flow security.

Part 10: How much does this program leak in that case?

In the worst-case scenario, this program leaks a single bit of information per execution, which indicates whether 'sec'(H) is even or odd, determined by the duration of the execution as explained above.

3) Problem 3: IFC for App-Specific Security Goals

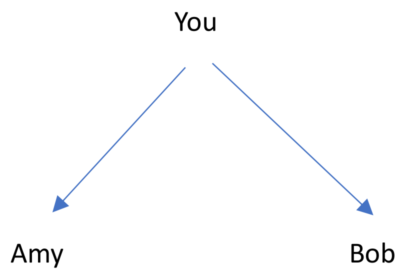
We see how information-flow control can enforce application-specific security goals.

Static Policies

Accompanying this assignment, you will find a code skeleton that implements PrettyBook: a pretty useless social network. PrettyBook operates on information that concerns three principals: Amy, Bob, and You. Amy and Bob are Your friends. You would like to share messages that you type with your friends.

Part 1: How must the principals Amy, Bob and You be ordered for messages from You to be permitted to flow to Amy and Bob, respectively? Draw the partial order?

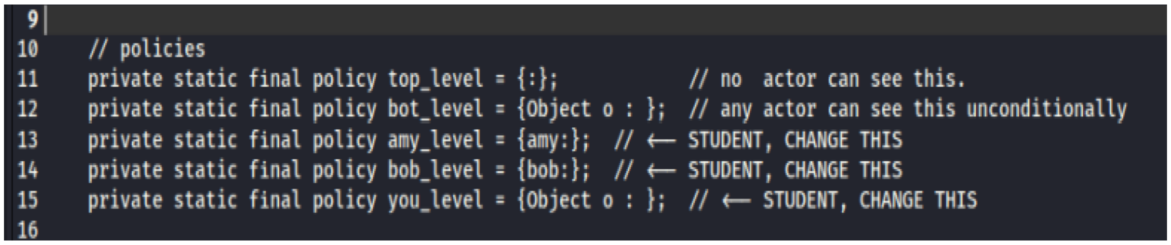
Given that there are no other constraints on the security information flow, the below diagram shows the information flow from “You” ,”Amy” and “Bob”. This structure enables "You" to send messages to "Amy" and "Bob", but not the other way around.



Part 2: How would you implement the above-stated ordering in the provided implementation of PrettyBook? Only the lines specified (with comments) need to be modified. Provide the updated lines in your solution.

I have to set the policy in the social Class, so the “**you_level”** policy will allow any actor to see its data, which implies that information can flow from "You" to anyone else. On the other hand, “**amy_level”** and “**bob_level”** policies should only allow "Amy" and "Bob" respectively to see their data, thus limiting who can receive their information.

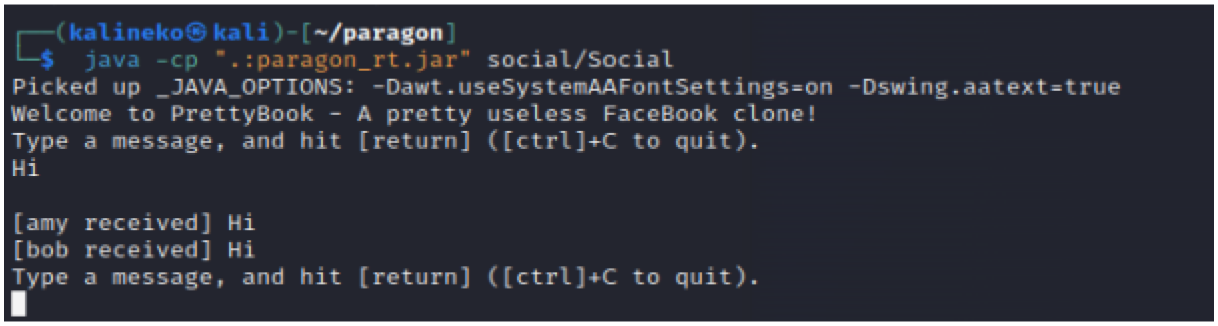
Below is the snippet of implementing such a change:



```

  // policies
  private static final policy top_level = {:};           // no  actor can see this.
  private static final policy bot_level = {Object o : };  // any actor can see this unconditionally
  private static final policy amy_level = {amy:};  // ⟵ STUDENT, CHANGE THIS
  private static final policy bob_level = {bob:};  // ⟵ STUDENT, CHANGE THIS
  private static final policy you_level = {Object o : };  // ⟵ STUDENT, CHANGE THIS

```



```
┌──(kalineko㉿kali)-[~/paragon]
└─$ java -cp ".:paragon_rt.jar" social/Social
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Welcome to PrettyBook - A pretty useless FaceBook clone!
Type a message, and hit [return] ([ctrl]+C to quit).
Hi

[amy received] Hi
[bob received] Hi
Type a message, and hit [return] ([ctrl]+C to quit).
█
```

Dynamic Policies

Accompanying this assignment, you find a modified version of PrettyBook, that now has extended menu options. You'd like to share messages that you type with your friends, only while they are permitted to receive your message, as dictated by a flow lock.

Part 3: How would you modify the level for Amy, Bob and You, to achieve this effect?

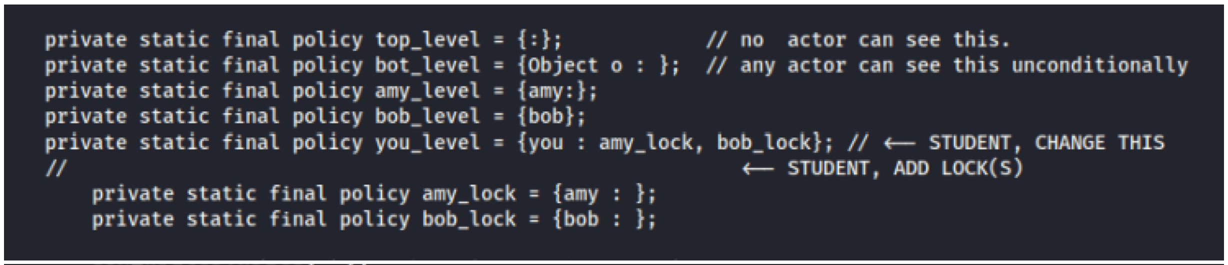
In dynamic policies, in order to control the information flow, I need to use locks. The locks can be toggled on and off to either allow or disallow information flows.

Part 4: How would you modify the implementation of PrettyBook, to achieve this effect?

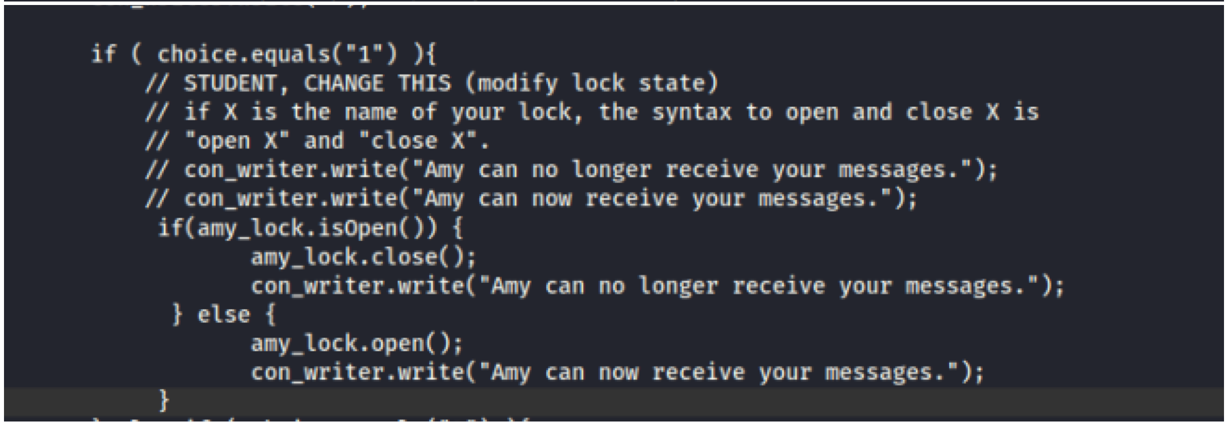
First, I would have to add the locks ("amy_lock" and "bob_lock") and change the security policies so "amy" can receive information only if "amy_lock" is closed and not the opposite. The same applies to "bob". The "you_level" remains unchanged, as "you" is allowed to share message with any other principal without restrictions.

The locks are initially closed, meaning information can flow. When the lock is opened, information cannot flow.

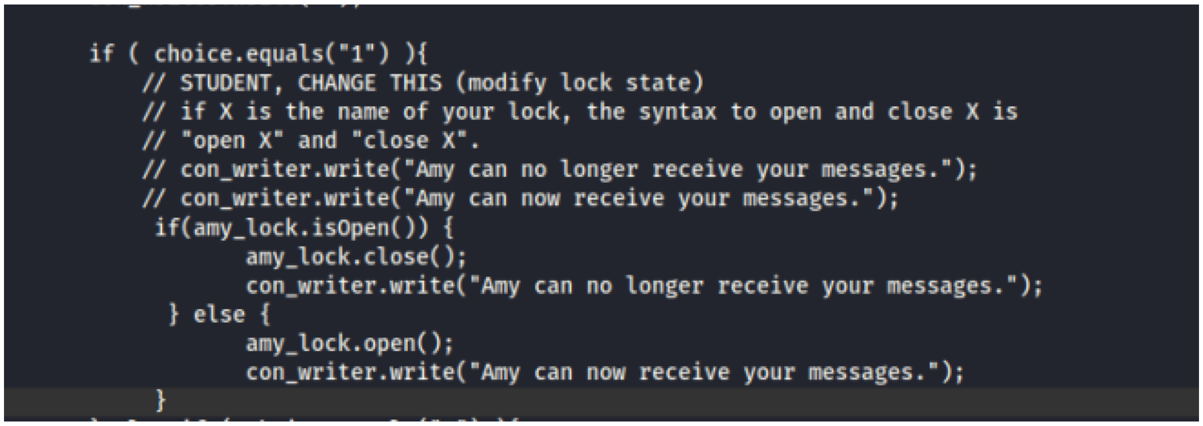
Please see the below snippet:



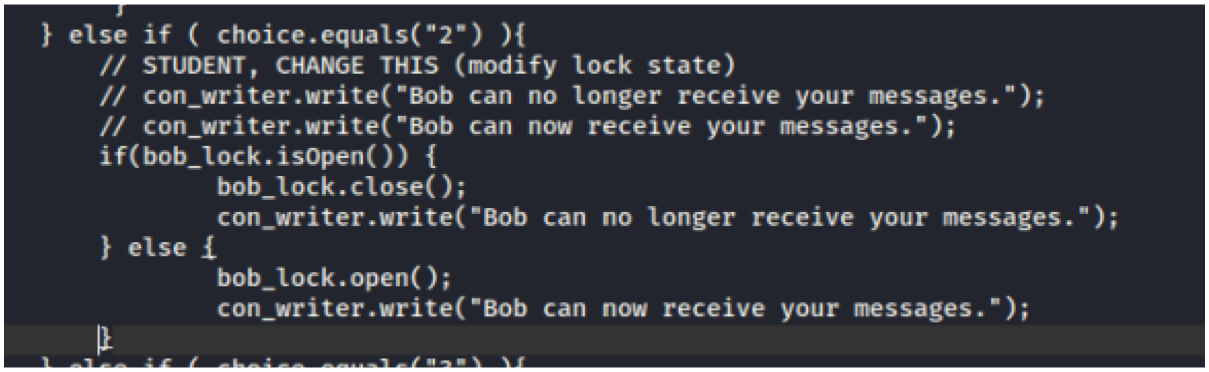
```
private static final policy top_level = {:};           // no  actor can see this.
private static final policy bot_level = {Object o : };  // any actor can see this unconditionally
private static final policy amy_level = {amy:};
private static final policy bob_level = {bob};
private static final policy you_level = {you : amy_lock, bob_lock}; // ⟵ STUDENT, CHANGE THIS
//                                                         ⟵ STUDENT, ADD LOCK(S)
    private static final policy amy_lock = {amy : };
    private static final policy bob_lock = {bob : };
```



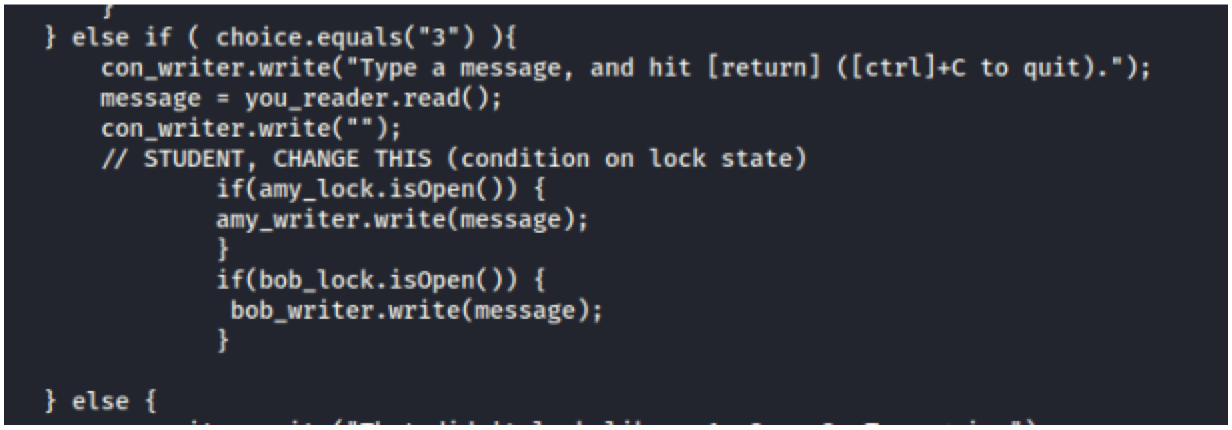
```
    if ( choice.equals("1") ){
        // STUDENT, CHANGE THIS (modify lock state)
        // if X is the name of your lock, the syntax to open and close X is
        // "open X" and "close X".
        // con_writer.write("Amy can no longer receive your messages.");
        // con_writer.write("Amy can now receive your messages.");
         if(amy_lock.isOpen()) {
                amy_lock.close();
                con_writer.write("Amy can no longer receive your messages.");
          } else {
                amy_lock.open();
                con_writer.write("Amy can now receive your messages.");
         }
```



```
    if ( choice.equals("1") ){
        // STUDENT, CHANGE THIS (modify lock state)
        // if X is the name of your lock, the syntax to open and close X is
        // "open X" and "close X".
        // con_writer.write("Amy can no longer receive your messages.");
        // con_writer.write("Amy can now receive your messages.");
         if(amy_lock.isOpen()) {
                amy_lock.close();
                con_writer.write("Amy can no longer receive your messages.");
          } else {
                amy_lock.open();
                con_writer.write("Amy can now receive your messages.");
         }
```



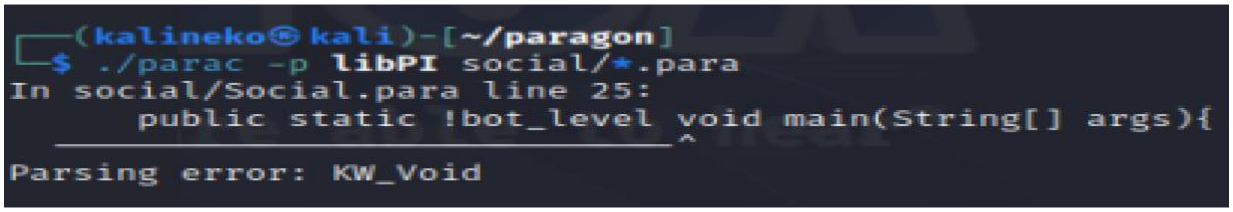
```
    } else if ( choice.equals("2") ){
        // STUDENT, CHANGE THIS (modify lock state)
        // con_writer.write("Bob can no longer receive your messages.");
        // con_writer.write("Bob can now receive your messages.");
        if(bob_lock.isOpen()) {
                bob_lock.close();
                con_writer.write("Bob can no longer receive your messages.");
        } else {
                bob_lock.open();
                con_writer.write("Bob can now receive your messages.");
        }
```



```
    } else if ( choice.equals("3") ){
        con_writer.write("Type a message, and hit [return] ([ctrl]+C to quit).");
        message = you_reader.read();
        con_writer.write("");
        // STUDENT, CHANGE THIS (condition on lock state)
                if(amy_lock.isOpen()) {
                amy_writer.write(message);
                }
                if(bob_lock.isOpen()) {
                 bob_writer.write(message);
                }

    } else {
```

Unfortunately, I was not able to parse the dynamic policies, as I was receiving the following error:



```
┌──(kalineko㉿kali)-[~/paragon]
└─$ ./parac -p libPI social/*.para
In social/Social.para line 25:
    public static !bot_level void main(String[] args){
    _______________________________________^
Parsing error: KW_Void
```